12 Streaming and Sampling

The data stream model is one of the most surprisingly persistent settings to study data analysis. It simultaneously captures several important concepts while modeling an important realistic setting.

- It models **time series data** capturing important sequential nature of observations; and how this can affect analysis or how that affect can be avoided.
- It reveals the core challenges of summarizing data, and what is possible to compute using a very **small amount of space**.
- It is one of the simplest setting that captures important aspects of **big data**; e.g., when data is too big to fit on one machine.
- It models important settings on the internet such as on a **router** or **very busy website** where one is trying to stay on top of trends.

12.1 Streaming

Streaming is a model of computation that emphasizes *space* over all else. The goal is to compute something using as little storage space as possible. So much so that we cannot even store the input. Typically, you get to read the data once, you can then store something about the data, and then let it go forever! Or sometimes, less dramatically, you can make 2 or more passes on the data.

Formally, there is a stream $A = \langle a_1, a_2, \ldots, a_m \rangle$ of m items where (for this lecture) each $a_i \in [n]$. This means, the size of each a_i is about $\log n$ (to represent which element), and just to count how many items you have seen requires space $\log m$ (although if you allow approximations you can reduce this). Unless otherwise specified, $\log$ is used to represent $\log_2$ that is the base-2 logarithm. The goal is to compute a summary S_A using space that is only $\mathsf{poly}(\log n, \log m)$.

12.1.1 Warm Up

Can we maintain the *sum* of items in a stream $S_m = \sum_{i=1}^{m} a_i$?
Yes, just a single counter $S_i = \sum_{j=1}^{m}$; each observation we just increment. This takes $O(\log(mn))$ space.

How about the *average* of items in the stream $V_m = \frac{1}{m}\sum_{i=1}^{m} a_i$?
Yes, but now we need to maintain two objects: the sum S_m, and also a counter $C_m = m$. Then we can reconstruct $V_m = S_m/C_m$.

12.2 Sampling

The most universal and natural summary of a large data set X is a random sample S. There are two main forms of random samples:

- **with replacement random sampling** means that each $s \in S$ is drawn iid from X. It allows for two $s, s' \in S$ to be the same item $x \in X$; so we can draw the same item twice. But the samples are independent which makes it easier to analyze.
- **without replacement random sampling** which means that S is a subset of X, so if all $x, x' \in X$ are distinct, then all $s, s' \in S$ are also distinct. We do not get the independence of the iid property; however, these samples are better representatives of the full data sets X.

12.2.1 Reservoir Sampling

The more famous way to draw a single random sample from a set X is called *reservoir sampling*. It keeps a single item $s \in X$ as the *reservoir* as it scans over X. Assume X is stored as a sequence $\langle x_1, x_2, \ldots, x_i, \ldots, x_n \rangle$. When we consider x_i, we keep x_i with probability $1/i$; if so it replaces $s \leftarrow x_i$; otherwise we keep s as it was before. It then handles x_{i+1} and so on.

Algorithm 12.2.1 Reservoir Sampling(X)

```
s ← x_1
for i = 2 to n do
    Generate u ~ Unif(0, 1]
    if (u ≤ 1/i) then
        s ← x_i
return s
```

At any point in the stream $X_i = \langle x_1, x_2, \ldots, x_i \rangle$, the maintained s is a uniform random sample from X. It does not depend on stream order.

The argument is inductive. The first point x_1 is initially kept with probability $1/1 = 1$, so always. This is the base case. Then x_2 is kept with probability $1/2$, and at that point x_1 is also kept with probability $1/2$. Now inductively, at the start of processing the ith point, each item has been kept with probability $1/(i-1)$. The new item x_i is kept with the correct probability $1/i$. And each other item is kept with probability $\frac{1}{i-1}\frac{i-1}{i} = \frac{1}{i}$ – also the correct probability.

To get a with replacement random sample of size k, we can just maintain k independent reservoir samplers in parallel.

12.2.2 Reservoir Sampling Without Replacement

To maintain k samples *without* replacement we can also extend reservoir sampling. We keep the first k items (put in the reservoir). And then for item x_i (with $i > k$) we put it in the reservoir with probability k/i; in this case it kicks out a random item.

Algorithm 12.2.2 Reservoir Sampling(X, k)

```
S ← {x_1, x_2, ..., x_k}
for i = k + 1 to n do
    Generate u ~ Unif(0, 1]
    if (u ≤ k/i) then
        remove random s' from S
        Put x_i into S
return S
```

By roughly the same inductive argument we can show that the set S is a without replacement uniform random sample from X of size k. The first k points are kept deterministically as the base case. For the inductive step, item x_i is kept correctly with probability k/i. And all other items were in the reservoir with probability $\frac{k}{i-1}$ and are selected to be replaced with probability $\frac{k}{i}\frac{1}{k} = \frac{1}{i}$. Hence they are still in the reservoir after the step with probability $\frac{k}{i-1}(1 - \frac{1}{i}) = \frac{k}{i}$ as desired.

12.2.3 Weighted Random Sampling

In another important setting, each item x_i comes with a weight w_i of how important it is. We want to keep an item x_i with probability w_i/W_i where $W_i = \sum_{i=1}^{i} w_i$.

Reservoir sampling extends naturally for 1 item; we maintain the sum W_i and item x_i is kept with probability w_i/W_i. It extends to the k with replacement version by keeping k such samplers in parallel.

12.2.4 Bottom-k Sampling

Another approach to sampling without replacement is bottom-k sampling. We assign each x_i a uniform $u_i \sim \mathsf{Unif}(0, 1]$. Then we maintain the item with smallest u_i value for one sample. To get k uniform random samples, we keep the items with smallest k u_i values.

This approach extends to weighted without replacement sampling. We now maintain a priority $\rho_i = -\frac{1}{w_i}\ln(u_i)$. This has the nice property that probability item i is the smallest probability is w_i/W_i. And the k smallest item are a perfect weighted without replacement random samples.

12.3 Quantiles

Another important but simple problem for streaming data is the *quantiles* problem. For this we consider the ordering of the elements in $[n]$ as important. In fact, its typically easier to think of each element being a real value $a_i \in \mathbb{R}$ so that they are continuously valued and we have a comparison operator $<$. Think of a_i as the number of milliseconds someone spent on a visit to a website before clicking a link. Or a_i could be the amount of money spent on a transaction. Or a_i could be the amount of rainfall in a day.

Now instead of searching for frequently occurring items (since we may never see the same item twice) it is better to treat these as draws from a continuous distribution over $\mathbb{R}$. In this case, two very similar (but perhaps not identical) values are essentially equivalent. The simplest well-defined interaction with such a distribution is through the associated cumulative density function. That is, given any value v, we can ask what fraction of items have value less than or equal to v. We can define the *rank* of v over a stream A as

$$\mathsf{rank}_A(v) = |\{a_i \in A \mid a_i \leq v\}|.$$

Now an *ε-approximate quantiles data structure* Q_A returns a value $Q_A(v)$ for all v such that

$$|Q_A(v) - \mathsf{rank}_A(v)/m| \leq \varepsilon.$$

By combining two such queries, we can also ask what fraction of data falls between two values v_1 and v_2 as $Q_A(v_2) - Q_A(v_1)$.

Size bounds. If we are not concerned about streaming, we can easily construct a data structure of size $1/\varepsilon$. We simply sort all values in A, and then select a subset B of size $1/\varepsilon$ elements, evenly spaced in that sorted order. Then $Q_A(v) = \mathsf{rank}_B(v)/|B|$. This is the smallest possible in general.

If we maintain a random sample Q of size $k = O((1/\varepsilon^2)\log(1/\delta))$, then it provides this ε-approximate quantile structure with probability at least $1 - \delta$.

Streaming algorithms are known of size $O((1/\varepsilon)\log\log(1/\varepsilon))$ (which is the smallest possible size).

Median. Additionally, such a summary also encodes properties like the approximate median. This is the value for which $\mathsf{rank}_A(v)/m = 0.5$ (naively one may have to find this by binary search, if the structure is a set B and $Q_A(v) = \mathsf{rank}_B(v)/|B|$, then we can also maintain this directly. In addition to a basic quantiles sketch, we will describe a simpler "frugal" variant which can maintain values like the approximate median (or any other quantile) approximately without maintaining all quantiles.

12.3.1 Merging Quantiles

The key idea in efficient quantiles sketches are being able to merge two sketches without increasing the error or the size. For quantiles, this works be a simple procedure describe below, where each of the Q_A data structures is simply a set B which returns $\mathsf{rank}_B(v)/|B|$.

Now given two such sets B_1 and B_2, both of size s, representing sets A_1 and A_2 both of size t. To merge the summaries, we let $B = B_1 \cup B_2$. Its not hard to see that if

$$|\mathsf{rank}_{A_1}(v)/t - \mathsf{rank}_{B_1}(v)/s| \leq \varepsilon \quad \text{and} \quad |\mathsf{rank}_{A_2}(v)/t - \mathsf{rank}_{B_2}(v)/s| \leq \varepsilon,$$

then by examining $\mathsf{rank}_A(v)$ and $(t/s)\mathsf{rank}_B(v)$,

$$|\mathsf{rank}_A(v)/t - \mathsf{rank}_B(v)/s| \leq \varepsilon.$$

However, $|B| = 2s$, so the size has doubled. To reduce the size we do the following simple step. We sort B, and let B_e be all points in B with even indices, and let B_o be all points in B with odd indices. Then we let the new sketch B' be either B_e or B_o, chosen at random.

With this sketch, we don't expect (in the expected value sense) to over- or under-count any rank query. But the process still seems like it should add some error. It turns out not too much is added, since the previous levels induce far less additive error than the current ones. Moreover, if we increase the sketch size s from $1/\varepsilon$ to $k_\varepsilon = O((1/\varepsilon)\sqrt{\log(1/\varepsilon)\log(\delta)})$, then with probability at least $1-\delta$, the error is never more than ε after any number of merges.

However, this requires that we only merge summaries B_1 and B_2 that represent exactly the same size sets A_1 and A_2. To deal with this issue, each summary will actually store up to $g_\varepsilon = O(\log(m\varepsilon))$ sets, where the jth set $B(j)$ (if it exists) represents a set of size $m/2^j$ for some $j \in [0, g_\varepsilon]$. Then on a merge, starting at the large-index layers, we merge pairwise (if there is more than one of some type), and push the merged sketches on up the representation, potentially increasing the height of the structure g_ε by 1. Or in the streaming setting, we can just add a single point to a buffer of size k_ε, then merge with the bottom ($j = g_\varepsilon$) layer.

This takes overall space $O((1/\varepsilon)\log(\varepsilon m)\sqrt{\log 1/\varepsilon}\log(1/\delta))$ to guarantee on a data set of size m, that the normalized rank has at most ε error, with probability at least $1-\delta$. With some care to how the hierarchy is managed, the size can be reduced to $O((1/\varepsilon)\log\log(1/\varepsilon)\log(1/\delta))$.

12.3.2 Frugal Median

The Frugal estimate of the median can be maintained easily as followed over an ordered set of integers. The simplest version just maintains a single label $\ell \in [n]$. Initially set $\ell = 0$ (or any value). Then if $a_i > \ell$, then increment ℓ. If $a_i < \ell$, then decrement ℓ. Psuedocode is in Algorithm 12.3.1.

Algorithm 12.3.1 Frugal Median(A)

```
Set ℓ = 0.
for i = 1 to m do
  if (a_i > ℓ) then
    ℓ ← ℓ + 1.
  if (a_i < ℓ) then
    ℓ ← ℓ − 1.
return ℓ.
```

This can be generalized to any quantile, say trying to find just the value v such that $\mathsf{rank}_A(v)/m = 0.75$. Then we use a bit of randomization.

Algorithm 12.3.2 Frugal Quantile(A, ϕ) e.g. $\phi = 0.75$

Set $\ell = 0$.
for $i = 1$ **to** m **do**
  $r = \mathsf{Unif}(0, 1)$ (at random)
  if $(a_i > \ell$ **and** $r > 1 - \phi)$ **then**
    $\ell \leftarrow \ell + 1$.
  if $(a_i < \ell$ **and** $r > \phi)$ **then**
    $\ell \leftarrow \ell - 1$.
return ℓ.

The bounds for this algorithm are not as absolutely strong as for the Misra-Gries algorithm, but it uses far less space. For instance, for the median version let M be the integer value of the true median, and say we are happy with any value v such that $\mathsf{rank}_A(v)/m \in [1/2 - \varepsilon, 1/2 + \varepsilon]$ for some small value $\varepsilon \in (0, 1/2)$. The with probability at least $1 - \delta$ after $\frac{M \log(1/\delta)}{\varepsilon}$ steps, our estimate will be within the desired range.

The bounds are better if we start our estimate at a value closer to v^* than 0. Also, if we are using an extra small counter, then we can adaptively change the amount we increment or decrement the label, and decrease the number of steps we need.