Summer of Math Exposition 4: Sampling

Ian Chen

August 7, 2025

Sampling is a fundamental part in statistical studies. Instead of collecting data from a large population, which is often infeasible at worse and expensive at best, statisticians collect a ***representative sample*** and infer conclusions about the population from the sample. Here, we will explore algorithms to efficiently sample from data streams.

We expect a prerequisite knowledge of Big-O and elementary probability. See §4 for a refresher.

In §1, we develop the problem and a simple solution. Then, releasing the assumption that we know the population size, we explore a simple type of resevoir sampling algorithm in §2 and an application in §2.1. Finally, we refine it to optimality in §3. Proofs for some theorems are in §5, but I encourage the reader to attempt them as exercises.

1 Problem Definition

Let $D = (x_1, x_2, \ldots)$ be a data stream with weights $(w_1, w_2, \ldots)$. Let k be the number of samples we wish the collect, and N the size of the stream (if known). We wish to compute a sample $S(D) \subset D$, where $\mathbf{Pr}(x_i \in S(D)) \propto w_i$ and $\big|S(D)\big| = \min(n, \big|D\big|)$. That is, we want to sample without replacement. We are allowed to use the *random*(), generating a uniform real in $[0, 1)$, and the *randint*(a, b), generating a uniform integer in $[a, b]$, functions.

Here's an example:

There are a few properties of this approach that can be undesireable, which all stem from the fact that we need to store the entire data stream. This is because we make multiple passes of the data, first in finding the sum of all the weights, and a new pass when selecting an element (and removing). From here on, we explore resevoir sampling (Vitter 1985), which are single-pass sampling techniques.

2 Bottom-k Sampling

First, let us assume unit weights, namely $w_i = 1$. Then, we can select a random element by assigning a random key y_i for each x_i, and picking the minimum. Moreover, we can select k items by picking the k smallest keys. The correctness is established by the following theorem.

Theorem 1. *Let $Y_1, \ldots, Y_n \overset{iid}{\sim} Y$, and suppose WLOG that they are distinct. Then, $\mathbf{Pr}(r_i \leq k) = \frac{k}{n}$, where $r_i = \big|\{Y_j \mid Y_j \leq Y_i\}\big|$ is the rank of Y_i.*

Now, suppose that the weights are not unit weight, we need to generate the keys y_i according to the weight w_i. One approach given by Cohen and Kaplan 2007 is to do this using the exponential distribution. Note that we can generate an exponential distribution from $U \sim U(0,1)$ by the transform $-\theta \log(U) \sim Exp(\theta)$.

2.1 Application

This section requires terminology of graph theory and can be skipped. It is adapted from §3 of Cohen 1997.

Suppose we have a simple directed graph $G = (V, E)$ with *vertex weights*. For each vertex $v \in V$, we want to sample k vertices (w/o replacement) from its 2-hop neighborhood, with probability proportional to the vertex weights. Cohen 1997 use this sample to estimate the size of the neighborhood, but we focus more on the sampling process here.

Following the ideas developed in §2, we first assign a random key for each vertex. Then, for each vertex, we need to compute the k least keys in its neighborhood.

```
SampleFromNeighborhood(V, E, k):
  for i from 1 → k:
      Generate y_j ~ Exp(w_j) for all j
      for j in increasing y_j order:
          if v_j is done, then skip
          N ← two-hop neighborhood of v_j
          for u ∈ N:
              Add v_j to S(u) if u has not been added this iteration yet
              Mark u as done
```

Theorem 2. *For each vertex, we can collect samples of k vertices (w/o) replacement from its 2-hop neighborhood in $O(k(n \log n + m))$ time.*

Proof. First, we find that Algorithm x correctly solves the problem because for each vertex, it finds the minimum k keys in its reachability set. Thus, by our analysis in §2, we correctly generate a weighted sample.

Now, we prove the runtime. Consider an iteration of the algorithm. First, we generate random weights, which is done in $O(n)$ time, and sort them in $O(n \log n)$ time. Finally, we proceed by doing a reachability search, which visits every edge at most once and so takes $O(m)$ time. Repeating this algorithm k times, we obtain a final runtime of $O(k(n \log n + m))$. □

3 Optimal Resevoir Sampling

For optimality, see Vitter 1985. See Li 1994 for unweighted resevoir sampling, and Efraimidis and Spirakis 2006 for weighted sampling.

4 Prerequisites

5 Proofs

Theorem 1. Suppose this. Then, we win. □

References

Cohen, Edith (Dec. 1997). "Size-Estimation Framework with Applications to Transitive Closure and Reachability". In: *Journal of Computer and System Sciences* 55.3, pp. 441–453. ISSN: 0022-0000. DOI: 10.1006/jcss.1997.1534. URL: http://dx.doi.org/10.1006/jcss.1997.1534.

Cohen, Edith and Haim Kaplan (Aug. 2007). "Summarizing data using bottom-k sketches". In: *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*. PODC07. ACM, pp. 225–234. DOI: 10.1145/1281100.1281133. URL: http://dx.doi.org/10.1145/1281100.1281133.

Efraimidis, Pavlos S. and Paul G. Spirakis (Mar. 2006). "Weighted random sampling with a reservoir". In: *Information Processing Letters* 97.5, pp. 181–185. ISSN: 0020-0190. DOI: 10.1016/j.ipl.2005.11.003. URL: http://dx.doi.org/10.1016/j.ipl.2005.11.003.

Li, Kim-Hung (Dec. 1994). "Reservoir-sampling algorithms of time complexity O(n(1 + log(N/n)))". In: *ACM Trans. Math. Softw.* 20.4, pp. 481–493. ISSN: 0098-3500. DOI: 10.1145/198429.198435. URL: https://doi.org/10.1145/198429.198435.

Vitter, Jeffrey S. (Mar. 1985). "Random sampling with a reservoir". In: *ACM Trans. Math. Softw.* 11.1, pp. 37–57. ISSN: 0098-3500. DOI: 10.1145/3147.3165. URL: https://doi.org/10.1145/3147.3165.