Privately Releasing Quantiles

Nathan Manohar

Harvard College ‘16 and Summer 2014 Intern

Privacy Tools for Sharing Research Data

A National Science Foundation Secure and Trustworthy Cyberspace Project

PROBLEM

How can we accurately release quantiles of a data set while simultaneously preserving differential privacy?

More formally, suppose we have a dataset

$$D = (x_1, x_2, ..., x_n)$$

where the entries come from a finite set of real numbers. We define the kth q-quantile of D as a number z that is larger than a k/q fraction of the entries of D.

In other words, z is a real number such that

$$\left|\{i : x_i \le z\}\right| = \left\lfloor \frac{kn}{q} \right\rfloor$$

In order to preserve differential privacy, we will have to release an

α-approximate kth q-quantile which is a number z such that

$$\left|\{i : x_i \le z\}\right| \in \left(\frac{k}{q} \pm \alpha\right) n$$

Our problem is to release z in a way that minimizes α while preserving (ε,δ)-differential privacy.

APPROACH

To privately release quantiles, we will use an algorithm that releases an approximate CDF of the data while preserving differential privacy.

The approximate CDF can then be used to approximate any quantile for the data set. To do this, one simply calculates kn/q and then finds the point on the CDF with y coordinate kn/q. The corresponding x coordinate for this point is then z.

The main reason for outputting an approximate CDF as opposed to a set of quantiles is that the approximate CDF can be computed using an ε amount of our privacy budget, but once it is computed, it can be used to approximate any quantile. In contrast, computing a set of quantiles and releasing them will use up an ε amount of the privacy budget every time we have a different set of quantiles we wish to compute.

BINARY TREE ALGORITHM

The binary tree algorithm works as follows:

- First, create a binary tree with the elements of the data universe as the leaves of the binary tree.
- Then, iterate through the data set and store the counts of each element in the data set in the leaf of the binary tree corresponding to that element. So, the leaves of the binary tree contain the number of times each element occurred in the data set.
- Next, add the counts in adjacent nodes together and repeat this process to form the entire binary tree.
- Then, add Lap((log D)/ε) noise to every node in the binary tree
- Next, create the approximate CDF by using the binary tree to compute the count in the intervals [min, t] for various t. Since our data universe might be very large, we cannot necessarily iterate t over the entire data universe and instead will need to choose a step size to increase t by every time.
- One problem with just using this algorithm is that the approximate CDF that is computed will not necessarily be monotone since we added Laplace noise. We will want to fix this by smoothing the CDF to make it monotone.

SMOOTHING ALGORITHM

We can think of the approximate CDF as a vector where the ith element of the vector is the count of the number of elements in the data set between [min, i * step_size].

Clearly, if this vector is to represent a CDF, the entries should be monotonically increasing since the counts can only increase if we increase the size of the interval we are considering.

So, we want to take our approximate CDF vector and make it monotone in a way that will minimize the change in the vector

SMOOTHING ALGORITHM

Formally, the smoothing algorithm works as follows:

- Given a vector v, the algorithm outputs a vector v' that minimizes the l_2 norm $||v' - v||$ subject to the constraint that v' is monotone.
- The kth element of v' is given by

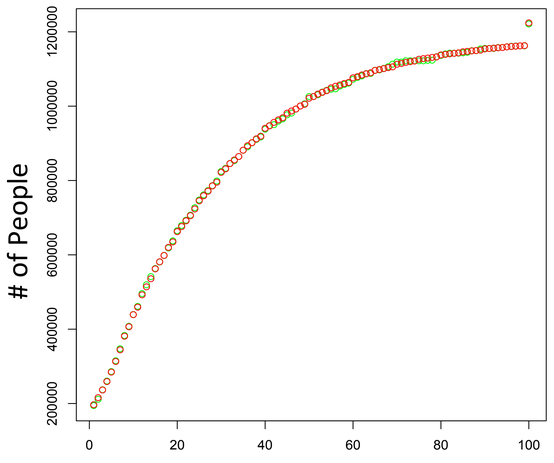
$$\min_{j \in [k,n]} \max_{i \in [1,j]} M[i,j]$$

where $M[i,j]$ is the mean of the elements of v between the ith and jth indices.

- So, simply applying the smoothing algorithm to the output of the binary tree algorithm gives a monotone approximate CDF for the data set that can be used to approximate quantiles.

RESULTS

Below is a plot comparing the output of the above algorithm to the original CDF on a large data set. The error is interpreted as follows: with probability 1 – β, the count given by an interval query to the approximate CDF is with ± α of the value given by the same query to the actual CDF.

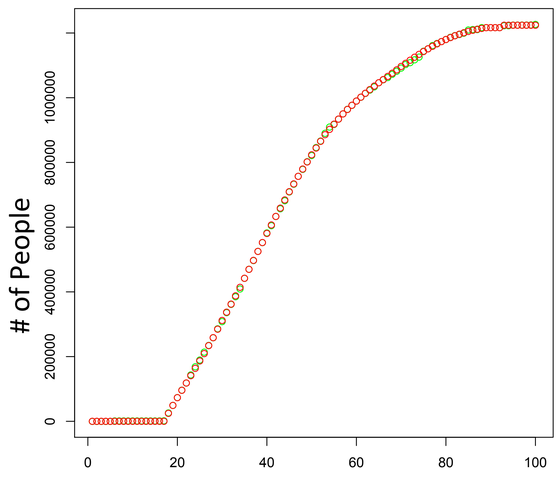


Red: Original CDF
Green: Approx. CDF
Eps = 0.01
Beta = 0.05
Alpha = 9333.16

# of Points Exceeding Error in 10 Trials (Expectation of 5):
0, 0, 1, 1, 2, 0, 0, 0, 0, 1

RESULTS

Below is another plot comparing the approximate CDF to the actual CDF on the same large data set, but this time the quantiles are computed for the age characteristic. Here, we see that the algorithm has very good accuracy even when the counts in each bin are large.



Red: Original CDF
Green: Approx. CDF
Eps = 0.01
Beta = 0.05
Alpha = 9333.16

# of Points Exceeding Error in 10 Trials (Expectation of 5):
0, 12, 2, 4, 3, 4, 0, 1, 1, 0

FUTURE WORK

- Find a good way to graphically show error
- More testing on different data sets
- Lower bound the expected error
- Potentially modify the binary algorithm to add less noise to nodes closer to the root and more to nodes closer to the leaves (a biased node close to the root affects accuracy much more than one close to the leaves)

ACKNOWLEDGEMENTS

I would like to thank my advisors, Salil Vadhan, Kobbi Nissim and Mark Bun, for their help and support throughout the summer. I would also like to thank the Harvard College Research Program for providing the financial aid for this project. Finally, I would like to thank the other members of the Privacy Tools for Sharing Research Data Project for their contributions to the project.

CRCS Center for Research on Computation and Society at Harvard School of Engineering and Applied Sciences

BERKMAN CENTER FOR INTERNET & SOCIETY AT HARVARD UNIVERSITY

The Institute for Quantitative Social Science at Harvard University

DATA PRIVACY LAB