Harvard Privacy Tools Project

Data Processing Notes

Christian Covington

February 26, 2020

1 Overview

This document contains notes on our choices regarding “data processing”, which I am using to stand in for all things related to data bookkeeping (data bounds, imputation, sample size calculations, etc.).

2 Clamping

Many operations in the library require the definition of a set of possible data values we would like to use for said operation. For numeric variables, this typically takes the form of a closed interval $[min, max] \in \mathbb{R}$ or $\mathbb{Z}$. For categorical variables, this is a discrete set of elements. In our system, we have slightly different notions of clamping for numeric and categorical variables. We consider $f64$ and $i64$ to be allowable numeric types, and $f64, i64, bool$, and $String$ to be allowable categorical types.

Our method for clamping numeric types should be familiar; it accepts data, a min, and a max, and maps data elements outside of the interval $[min, max]$ to the nearest point in the interval. Null values (represented as NAN for the $f64$ type and which do not exist for the $i64$ type) are preserved by numeric clamping.

Our categorical clamping accepts data, a set of feasible values, and a null value. Data elements that are outside the set of feasible values are mapped to the null value. Null values are preserved by the categorical clamping.

3 Imputation

Much like clamping, we have slightly different notions of imputation for numeric and categorical variables. In this case, we consider only $f64$ to be an allowable numeric type, while

$f64, i64, bool$, and $String$ are allowable categorical types.

Numeric imputation is parameterized by min, max, and a supported probability distribution (e.g. Gaussian or Uniform) with appropriate arguments. NAN values are replaced with a draw from the provided distribution.

Categorical imputation is parameterized by categories, probabilities, and a null value. Each data element equal to the null value is replaced with a draw from the set of categories, according to the probabilities provided.

4 Public vs. Private n

We want our library to support cases both in which the analyst does and does not have access to the number of records in the data. As such, the analyst must, at the beginning of any analysis, choose an $\hat{n}$. Throughout the section, I will use n to be the actual number of elements in the data and $\hat{n}$ as the estimate used for the analysis.

4.1 Choosing an n

We offer two ways to choose an $\hat{n}$; the analyst make their own guess about n, or they can consume some privacy budget to estimate n. In the case of a user-provided $\hat{n}$, this could be because n is actually public knowledge (in which case the user's estimate should be correct), or it could be a guess by the user. Whether $\hat{n}$ is provided by the user or estimated, this $\hat{n}$ will be used throughout the entire analysis.

We chose this strategy for a few reasons. First, we want the library to be general and this strategy seemed to meet that criterion. Second, we did not feel comfortable having different processes for public vs. private n, e.g. one in which the library automatically finds the sample size from public metadata and treats it as ground truth if the metadata states that $n \neq private$. Because public metadata are not typically assumed to be stored securely, it is possible that a metadata file could be changed nefariously, for example from $n = private$ to $n = 1,000$ (and thus $n \neq private$). In such a case, we would not want the library to accept sample sizes from public metadata without suspicion; if it did so, then it would likely throw errors if the actual size of the data were not n, thus revealing to the adversary that the actual size of the data $\neq 1,000$. By not distinguishing between public and private n and treating any user-input as a guess, we are making our system robust to metadata attacks. This does come with the downside that we have no notion of verifying n within the library, this verification must come from users' trust in the veracity of the public metadata.

4.2 Functions dependent on $\hat{n}$

It is very common for our functions to be dependent on $\hat{n}$, which adds a layer of complication. Say, for example, that we want the mean of a vector x. Typically, we would write

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i,$$

but in our setting we do not know n. We could just replace $1/n$ with $1/\hat{n}$ and keep the summation over the actual n elements, but this would bias our estimate. We will instead change our data x to ensure that $\hat{n}$ is the correct number of elements in x.

4.3 Making data consistent with $\hat{n}$

If $\hat{n} < n$, then we can subsample x to get $\tilde{x}$ where $|\tilde{x}| = \hat{n}$.
If $\hat{n} > n$, then we will add elements to x to get $\tilde{x}$ where $|\tilde{x}| = \hat{n}$.
We generate these new elements by imputing values for the $\hat{n} - n$ elements we need to create, using the user-provided data bounds specified for the statistic in question. This could be done via any imputation method supported by the library.
We considered sampling with replacement to generate $\tilde{x}$ for the case where $\hat{n} > n$, but this complicates sensitivity calculations as any element in x could appear in $\tilde{x}$ multiple times. We could sample with replacement in a way that guarantees for $\hat{n} < c \cdot n$ that any element in x occurs in $\tilde{x}$ at most c times, but we cannot establish such a c (with probability 1 that our c is correct) without leaking extra information about n.

4.4 Accuracy/Error

Our accuracy and error calculations always assume that $\tilde{x}$ is the true data in question. That is, the additional error induced by subsampling or imputing extra values is not considered.

4.5 Maintaining Subset Consistency

There are cases where a user may want to enforce consistent subsampling across multiple calls to the data – that is, ensuring that subsampling produces the same subset, and arranged in the same order. To this end, we let users define a set of indices that can be used arbitrarily often across an analysis.

4.5.1 Defining the index set

In order to define a set of indices, we use the function $create\_sampling\_indices(k, n)$, where k is the size of the desired subset and n is the size of the original data. The function returns k integers from the set $\{0, 1, \ldots, n-1\}$, sampled uniformly at random, that will serve as the indices for subsampling.