fn make_randomized_response_bitvec

Abigail Gentle

April 21, 2025

This proof resides in **"contrib"** because it has not completed the vetting process.

Proves soundness of `make_randomized_response_bitvec` in `mod.rs` at commit cfd1bec5 (outdated[1]).

1 Introduction

This is a randomizer in the local model of differential privacy, which is a simplification of the RAPPOR protocol [1]. Under local differential privacy each user is guaranteed ε-DP for their response. RAPPOR achieves local differentially private frequency estimation by flipping each bit of an input vector with fixed probability. Because the noise is added mechanically we can efficiently post-process a set of outputs from this randomizer with the function `debias_randomized_response_bitvec`, which sums and normalises a vector of private outputs to account for the bias introduced. The input to this function are elements from a `BitVectorDomain` with fixed `max_weight`, setting this value is necessary in order to bound the sensitivity of the function. we detail two use cases below to explain two possible settings of `max_weight`, which we will denote m.

Frequency estimation is the problem of generating a histogram of the empirical frequencies of categorical elements. Any categorical domain of size k can be mapped to the set $[k] = \{0, 1, \ldots, k-1\}$, and we can represent any element $j \in [k]$ as the j'th standard basis vector so that $m = 1$. In cases where the domain is extremely large or even unbounded (such as a set of strings) we can hash the domain elements using a bloom filter with m hash functions. In order to decode a set of outputs generated in this fashion we must perform a regression against the bloom filter representations of a set of candidate elements, neither bloom filters nor the regression tools required for this are implemented here, but details can be found in the original RAPPOR paper. We will require that $2m + 1 < k$, but m should be much less than k in order to preserve utility. As we will prove in 3.2 privacy degrades linearly with m.

2 Hoare Triple

2.1 Preconditions

- Variable `input_domain` must be of type `BitVectorDomain`, with `max_weight`.
- Variable `input_metric` must be of type `DiscreteDistance`.
- Variable `f` must be of type `f64`.
- Variable `constant_time` must be of type `bool`.

[1] See new changes with `git diff cfd1bec5..aafdf46 rust/src/measurements/randomized_response_bitvec/mod.rs`

2.2 Pseudocode

```
def make_randomized_response_bitvec(
        input_domain: BitVectorDomain,
        input_metric: DiscreteDistance,
        f: f64,
        constant_time: bool):
    output_measure = MaxDivergence(f64)

    if f <= 0.0 or f > 1: #
        raise Exception("Probability must be in (0.0, 1]")

    m = input_domain.max_weight
    if m is None:
        raise Exception("make_randomized_response_bitvec requires a max_weight on the
    input_domain")

    epsilon = 2 * m * log((2 - f) / f)
    def privacy_map(d_in: IntDistance): #
        if d_in == 0:
            return 0.
        if d_in > 1:
            raise ValueError("d_in must be 0 or 1.")
        return epsilon
    def function(arg: BitVector) -> BitVector: #
        k = len(arg)
        noise_vector = [bool.sample_bernoulli(f/2, constant_time) for _ in range(k)]
        return xor(arg, noise_vector)

    return Measurement(input_domain, function, input_metric, output_measure, privacy_map)
```

2.3 Postcondition

Theorem 2.1. For every setting of the input parameters `(f, m, constant_time)` to `make_randomized_response_bitvec` such that the given preconditions hold, `make_randomized_response_bitvec` raises an exception (at compile time or run time) or returns a valid measurement. A valid measurement has the following property:

1. (Privacy guarantee). For every pair of elements x, x' in `input_domain` and for every pair (`d_in`, `d_out`), where `d_in` has the associated type for `input_metric` and `d_out` has the associated type for `output_measure`, if x, x' are `d_in`-close under `input_metric`, `privacy_map(d_in)` does not raise an exception, and `privacy_map(d_in)` $\leq$ `d_out`, then `function`(x), `function`(x') are `d_out`-close under `output_measure`.

3 Proof

3.1 Privacy

Note 1 (Proof relies on correctness of Bernoulli sampler)**.** The following proof makes use of the following lemma that asserts the correctness of the Bernoulli sampler function.

Lemma 3.1. If system entropy is not sufficient, `sample_bernoulli` raises an error. Otherwise, `sample_bernoulli(f/2, constant_time)`, the Bernoulli sampler function used in `make_randomized_response_bool`, returns `true` with probability (`prob`) and returns `false` with probability (1 - `f/2`).

Theorem 3.2 ([1])**.** The program `make_randomized_response_bitvec` satisfies ε-DP, where

$$\varepsilon = 2m \log\left(\frac{2-f}{f}\right).$$

Proof. Let $y = (y_1, \ldots, y_j, \ldots, y_k)$ be a randomised report generated by applying `make_randomized_response_bitvec` to an input vector $x = (x_1, \ldots, x_j, \ldots, x_k)$. The output of the algorithm satisfies the following conditional probabilities,

$$\mathbb{P}[y_j = 1 \mid x_j = 1] = 1 - \frac{1}{2}f \tag{1}$$

$$\mathbb{P}[y_j = 1 \mid x_j = 0] = \frac{1}{2}f \tag{2}$$

The probability of observing any given report Y is $\mathbb{P}[Y = y|X = x]$. Suppose $x = (x, \ldots, x_k)$ is a single Boolean vector with at most m ones. Without loss of generality, as the probabilities are invariant under permutation, assume that $x^* = (x = 1, \ldots, x_m = 1, x_{m+1} = 0, \ldots, x_k = 0)$, then we have

$$\mathbb{P}[Y = y \mid X = x^*] = \prod_{i=1}^{m} \left(\frac{1}{2}f\right)^{1-y_i} \left(1 - \frac{1}{2}f\right)^{y_i} \times \prod_{i=m+1}^{k} \left(\frac{1}{2}f\right)^{y_i} \left(1 - \frac{1}{2}f\right)^{1-y_i}.$$

Then let D be the ratio of two such conditional probabilities with neighbouring inputs x, x', and let S be the range of `make_randomized_response_bitvec`.

$$\begin{aligned}
D &= \frac{\mathbb{P}[Y \in S \mid X = x]}{\mathbb{P}[Y \in S \mid X = x']} && (3)\\
&= \frac{\sum_{y \in S} \mathbb{P}[Y = y \mid X = x]}{\sum_{y \in S} \mathbb{P}[Y = y \mid X = x']} \\
&\leq \max_{y \in S} \frac{\mathbb{P}[Y = y \mid X = x]}{\mathbb{P}[Y = y \mid X = x']} \\
&= \max_{y \in S} \frac{\prod_{j=1}^{m} \left(\frac{1}{2}f\right)^{1-y_j} \left(1 - \frac{1}{2}f\right)^{y_j} \times \prod_{j=m+1}^{k} \left(\frac{1}{2}f\right)^{y_j} \left(1 - \frac{1}{2}f\right)^{1-y_j}}{\prod_{j=1}^{m} \left(\frac{1}{2}f\right)^{y_j} \left(1 - \frac{1}{2}f\right)^{1-y_j} \times \prod_{j=m+1}^{2m} \left(\frac{1}{2}f\right)^{1-y_j} \left(1 - \frac{1}{2}f\right)^{y_j} \times \prod_{j=2m+1}^{k} \left(\frac{1}{2}f\right)^{y_j} \left(1 - \frac{1}{2}f\right)^{1-y_j}} \\
&= \max_{y \in S} \frac{\prod_{j=1}^{m} \left(\frac{1}{2}f\right)^{1-y_j} \left(1 - \frac{1}{2}f\right)^{y_j} \times \prod_{j=m+1}^{2m} \left(\frac{1}{2}f\right)^{y_j} \left(1 - \frac{1}{2}f\right)^{1-y_j}}{\prod_{j=1}^{m} \left(\frac{1}{2}f\right)^{y_j} \left(1 - \frac{1}{2}f\right)^{1-y_j} \times \prod_{j=m+1}^{2m} \left(\frac{1}{2}f\right)^{1-y_j} \left(1 - \frac{1}{2}f\right)^{y_j}} && (4)\\
&= \max_{y \in S} \left[\prod_{j=1}^{m} \left(\frac{1}{2}f\right)^{2(1-y_j)} \left(1 - \frac{1}{2}f\right)^{2y_j} \times \prod_{j=m+1}^{2m} \left(\frac{1}{2}f\right)^{2y_j} \left(1 - \frac{1}{2}f\right)^{2(1-y_j)}\right] && (5)
\end{aligned}$$

Notice that, by Equation 4, the privacy is not dependent on k and Equation 5 is maximised when $y = 1, \ldots, y_m = 1$, and $y_{m+1}, \ldots, y_{2m} = 0$, giving

$$\begin{aligned}
D &\leq \left(1 - \frac{1}{2}f\right)^{2m} \times \left(\frac{1}{2}f\right)^{-2m} \\
&= \left(\frac{2-f}{f}\right)^{2m}
\end{aligned}$$

Therefore,

$$\varepsilon \leq 2m \log\left(\frac{2-f}{f}\right), \tag{6}$$

which completes the proof. □

3.2 Utility

Theorem 3.3. The program `debias_randomized_response_bitvec` is an unbiased frequency estimator.

Proof. By independence we can focus on the contribution of one message. Given an input $\bar{x} = (x_1, \ldots, x_j, \ldots, x_k)$, denote the output of `make_randomized_response_bitvec` on this input as $\bar{y}$. Assume each x_j was drawn from a distribution with probability $p(x_j)$, we can replace this assumption with the true frequency of x_j later. Each coordinate of $\bar{y}$ is therefore given as follows,

$$y_j = \text{Bern}\left(1 - \frac{1}{2}f\right) x_j + \text{Bern}\left(\frac{1}{2}f\right)(1 - x_j).$$

And the expectation of $\bar{y}_j$ is

$$\begin{aligned}\mathbb{E}\left[y_j\right] &= (1 - \frac{1}{2}f)p(x_j) + \frac{1}{2}f(1 - p(x_j)) \\ &= (1-f)p(x_j) + \frac{1}{2}f \end{aligned} \tag{7}$$

Therefore rearranging 7 we can estimate $p(x_j)$ as,

$$\hat{p}(x) = \frac{y_j - \frac{1}{2}f}{1-f} \tag{8}$$

Given n outputs, let $Y = \sum_{i=1}^{n} \bar{y}_i$ be the element-wise sum of the vector outputs. We can use the above to create an estimator the true counts $\hat{q}$ as,

$$\hat{q}_j = \frac{Y_j - n\frac{1}{2}f}{1-f} \tag{9}$$

Which is the estimator in `debias_randomized_response_bitvec`. □

Theorem 3.4. The program `debias_randomized_response_bitvec` is a frequency estimator with mean squared error

$$\mathbb{E}[\|q - \hat{q}\|_2^2] = \frac{nk\left(f - \frac{f^2}{2}\right)}{2(1-f)^2}.$$

Proof. Let $q = q(X^n) = \sum_{j=1}^{n} \bar{x}_i$ be the true (non-private) vector sum of the inputs. Our goal is to find the mean squared error of our estimate from this vector. Notice that each coordinate Y_j of $Y = \sum_{i=1}^{n} \bar{y}_i$, is a sum of Bernoulli random variables with probability $\frac{1}{2}f$ or $(1 - \frac{1}{2}f)$ meaning their sum is that of two binomials with equal variance (by the symmetry of their probabilities). Using this we get that the variance is,

$$\text{Var}(Y_j) = n\frac{f}{2}\left(1 - \frac{f}{2}\right), \tag{10}$$

and the mean squared error is,

$$\mathbb{E}[\|\hat{q} - q\|_2^2] = \mathbb{E}\left[\sum_{j=1}^{k}(\hat{q}_j - q_j)^2\right] = \sum_{j=1}^{k}\mathbb{E}[(\hat{q}_j - q_j)^2]$$

$$
\begin{aligned}
&= \sum_{j=1}^{k} \mathbb{E}[(\hat{q_j} - \mathbb{E}[\hat{q_j}])^2] && \text{(By unbiasedness, Theorem 3.3)} \\
&= \sum_{j=1}^{k} \mathrm{Var}(\hat{q_j}) = \sum_{j=1}^{k} \mathrm{Var}\left(\frac{\hat{Y_j} - n\frac{f}{2}}{1-f}\right) && \text{(By Equation 9)} \\
&= \sum_{j=1}^{k} \mathrm{Var}\left(\frac{Y_j}{1-f}\right) \\
&= \sum_{j=1}^{k} \frac{\mathrm{Var}(Y_i)}{(1-f)^2} \\
&= k\left(\frac{n\frac{f}{2}\left(1-\frac{f}{2}\right)}{(1-f)^2}\right) && \text{(By Equation 10)} \\
&= \frac{nk\left(f - \frac{f^2}{2}\right)}{2(1-f)^2}
\end{aligned}
$$

□

References

[1] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 21st ACM Conference on Computer and Communications Security*, Scottsdale, Arizona, 2014.