Privacy Proofs for OpenDP: Lipschitz Sized Mean for Proportion CI (for Partitioned Data)

Summer 2022

Contents

1 Algorithm Implementation

1.1 Code in Rust

The current OpenDP library contains the `make_lipschitz_sized_proportion_ci_mean` function estimating the overall sample proportion for partitioned data. This is defined in lines 34-182 of the file `mod.rs` in the Git repository https://github.com/opendp/opendp/blob/e9a8ce533a900a6561c0ea3be6bea8c985311374/rust/src/trans/proportion_ci/mod.rs#L134-L182.

1.2 Pseudo Code in Python

Preconditions

To ensure the correctness of the output, we require the following preconditions:

- `User-specified types:`
  - Variable `sample_sizes` must be of type `Vec<usize>`.
  - Variable `strat_sizes` must be of type `Vec<usize>`.
  - `TA`: must be of type `float`.

Postconditions

- A `transformation` is returned (i.e., if a `transformation` cannot be returned successfully, then an error should be returned).

Pseudo Code

```
def (sample_sizes: Vec<TA>, strat_sizes: Vec<TA>) -> TA:
    '''
    :param strat_sizes: the population size of each stratum
    :param sample_sizes: sample sizes in each stratum
    '''
    input_domain = ProductDomain<AllDomain<TA>>
    output_domain = AllDomain<TA>
    strat_weights = strat_sizes / sum(strat_sizes)
    def function(sample_sums: Vec<TA>) -> TA:
        strat_means = sample_sums / sample_sizes
        return sum(strat_weights * strat_means)
    input_metric = ProductMetric<AbsoluteDistance<TA>>
    output_metric = AbsoluteDistance<TA>
    def stability_map(d_in: AbsoluteDistance<TA>) -> TA:
        sens = max(strat_weights / sample_sizes)
        return d_in * sens

    return Transformation(input_domain, output_domain, function,
    input_metric, output_metric, stability_map)
```

2 Proof

Theorem 2.1. *For every setting of the input parameters* `sample_sizes` *and* `strat_sizes` *to* `make_lipschitz_sized_proportion_ci_mean` *such that the given preconditions hold,* `make_lipschitz_sized_proportion_ci_mean` *raises an exception (at compile time or run-time) or returns a valid* `transformation` *with the following properties:*

1. **(Appropriate output domain).** *For every vector v in the input domain,* `function`(v) *is in the output domain.*

2. **(Domain-metric compatibility).** *The domain* `input_domain` *matches one of the possible domains listed in the definition of* `input_metric`*, and likewise* `output_domain` *matches one of the possible domains listed in the definition of* `output_metric`*.*

3. **(Stability guarantee).** *For every pair of elements v, w in* `input_domain` *and for any* `d_in`*, where* `d_in` *has the associated type for* `input_metric`*, if v, w are* `d_in`*-close under* `input_metric`*, then* `function(v)`*,* `function(w)` *are* `stability_map(d_in)`*-close under* `output_metric`*.*

Proof.

1. **(Appropriate output domain).** From line 6, we know `strat_weights` is of type `Vec<TA>`. On line 12, `strat_means` will also be of type `Vec<TA>` since both `strat_sizes` and `sample_sizes` are of type `Vec<TA>`. The function returns a sum of a vector of type `Vec<TA>`, then the sum will be of type `TA`. That is, the output is in the output domain `AllDomain<TA>`.

2. **(Domain-metric compatibility).** The input domain of `make_lipschitz_sized_proportion_ci_mean` is `ProductDomain` of `AllDomain<TA>` and the input metric is `ProductMetric` of `AbsoluteDistance<TA>`. Each component of the input is in `AllDomain<TA>`. Since `AllDomain<TA>` matches one of the

possible domains listed in the definition of `AbsoluteDistance`, the input domain is compatible with the input metric.

Also, it follows directly that the output domain (`AllDomain<TA>`) is compatible with the output metric (`AbsoluteDistance<TA>`).

3. **(Stability guarantee.)** Let `Abs` stand for `AbsoluteDistance`. If v, w are `d_in`-close, then by the definition 1,

$$d_{\texttt{PM},\texttt{Abs}}(v, w) = \sum_i d_{\texttt{Abs}}(v_i, w_i) \leq \texttt{d\_in},$$

Let f denote the function in `make_lipschitz_sized_proportion_ci_mean`. For ease of notation, let c_i and n_i denote the ith element of `stra_weights` and `sample_sizes`, respectively. Then

$$\begin{aligned} d_{\texttt{Abs}}(f(v), f(w)) &= \left| \sum_i c_i \cdot \frac{v_i}{n_i} - \sum_i c_i \cdot \frac{w_i}{n_i} \right| \\ &= \sum_i \frac{c_i}{n_i} |v_i - w_i| \\ &= \sum_i \frac{c_i}{n_i} \cdot d_{\texttt{Abs}}(v_i, w_i) \\ &\leq \max_i \frac{c_i}{n_i} \cdot \sum_i d_{\texttt{Abs}}(v_i, w_i) \\ &\leq \max_i \frac{c_i}{n_i} \cdot \texttt{d\_in}. \end{aligned}$$

That is, $f(v)$ and $f(w)$ are `stability_map(d_in)`-close.

□

Definition 1 (Distance under `ProductMetric`)**.** *Let $d_{PM,M}$ denote the distance under `ProductMetric(M)` where `M` is a valid metric. Then $d_{PM,M}$ is defined as the sum of distance under each `M`. Specifically, for any v, w in the input domain and v_i, w_i denote their ith entry, respectively,*

(i) for input metric `MI`,

$$d_{PM,MI}(v, w) = \sum_i d_{MI}(v_i, w_i).$$

(ii) for output metric `MO`,

$$d_{PM,MO}(g(v), g(w)) = \sum_i d_{MO}(f_i(v_i), f_i(w_i)),$$

where g and f_i denote the `function` in their corresponding `Transformation`.