Privacy Proofs for OpenDP: Partition Map (Transformation)

Summer 2022

Contents

1 Algorithm Implementation **1**

1.1 Code in Rust 1

1.2 Pseudo Code in Python 1

2 Proof **2**

1 Algorithm Implementation

1.1 Code in Rust

The current OpenDP library contains the `make_map_partition_trans` function constructing a `Transformation` for partitioned data based on a list of `Transformations`. This is defined in lines 12-73 of the file `mod.rs` in the Git repository https://github.com/opendp/opendp/blob/449222066a006a19d15bb68e62a87105ce49eb15/rust/src/comb/partition_map/mod.rs#L12-L73.

1.2 Pseudo Code in Python

Preconditions

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

- `User-specified types:`
  - Variable `transformations` must be a vector of elements of class `transformation`.
  - `output_metric::Distance` must have trait `TotalOrd`.
  - `input_domain`: must have trait `Domain`.
  - `output_domain`: must have trait `Domain`.
  - `input_metric`: must have trait `Metric`.
  - `output_metric`: must have trait `metric`.

Postconditions

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

Pseudo Code

```python
def make_map_partition_trans(trans: List[Transformation]) -> Transformation:
    input_domain = ProductDomain(t.input_domain for t in trans)
    output_domain = ProductDomain(t.output_domain for t in trans)
    def function(data: Vec<DI::Carrier>) -> Vec<DO::Carrier>:
        output = []
        for part, t in zip(data, trans):
            output.append(t.function(part))
        return output
    input_metric = ProductMetric(trans.input_metric)
    output_metric = ProductMetric(trans.output_metric)
    stability_map(d_in: input_metric::Distance) -> output_metric::Distance:
        return max(t.map(d_in) for t in trans)

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

2 Proof

The necessary definitions for the proof can be found at "List of definitions used in the proofs".

Theorem 2.1. *For every setting of the input parameter `transformations` to `make_map_partition_trans` such that the given preconditions hold, `make_map_partition_trans` raises an exception (at compile time or runtime) 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).** Let $\texttt{t}_i$ denote the ith element of the variable `transformations`. Since $\texttt{t}_i$ is a `transformation`, for any v_i in the input domain of $\texttt{t}_i$, $\texttt{function}_i(v_i)$ is in the output domain of $\texttt{t}_i$. Therefore, for any $v = \{v_i\}_i$ in the `ProductDomain` of the input domains of $\texttt{t}_i$, $\texttt{function}(v) = \{\texttt{function}_i(v_i)\}_i$ is in the `ProductDomain` of the input domains of $\texttt{t}_i$. That is, for every v in the input domain of `make_map_partition_trans`, `function`(v) is in the output domain of `make_map_partition_trans`.

2. **(Domain-metric compatibility).** The `input_domain` of `make_map_partition_trans` is `ProductDomain` and the `input_metric` is `ProductMetric`. Since each component

t of the variable `transformations` is a `transformation`, `t.input_domain` matches one of the possible domains listed in the definition of `t.input_metric`. Therefore, the product of `t.input_domain` is compatible with the product of `t.input_metric`. The same argument holds for `output_metric`.

3. **(Stability guarantee.)** Let $\texttt{t}_i$ be the ith element of the variable `transformations`, and `MI` and `MO` denote the `input_metric` and the `output_metric` of $\texttt{t}_i$, respectively. Since for any i, $\texttt{t}_i$ is a `Transformation`, for any v_i, w_i in the input domain of $\texttt{t}_i$,

$$d_{\texttt{MI}}(v_i, w_i) \leq \texttt{d\_in}$$

implies

$$d_{\texttt{MO}}(f_i(v_i), f_i(w_i)) \leq \texttt{t}_i.\texttt{stability\_map(d\_in)} \cdot d_{\texttt{MI}}(v_i, w_i),$$

where f_i denotes the `function` in $\texttt{t}_i$, v_i and w_i are the ith component of v and w, respectively. Let $d_{\texttt{PM,MI}}$ denote the distance under `ProductMetric(MI)`. By definition 1(i),

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

Note that given $d_{\texttt{PM,MI}}(v, w) \leq \texttt{d\_in}$, by the nature of partitioned data, all $d_{\texttt{MI}}(v_i, w_i)$ are zeros except for at most one. Let g denote the function in `make_map_partition_trans`. By Definition 1(ii), we then have

$$\begin{aligned} d_{\texttt{PM,MO}}(g(v), g(w)) &= \sum_i d_{\texttt{MO}}(f_i(v_i), f_i(w_i)) \\ &\leq \sum_i \texttt{t}_i.\texttt{stability\_map(d\_in)} \cdot d_{\texttt{MI}}(v_i, w_i) \\ &\leq max_i(\texttt{t}_i.\texttt{stability\_map(d\_in)} \cdot d_{\texttt{MI}}(v_i, w_i)) \\ &\leq max_i(\texttt{t}_i.\texttt{stability\_map(d\_in)}) \cdot \texttt{d\_in}. \end{aligned}$$

Therefore, we have shown that $g(v)$ and $g(w)$ are `stability_map(d_in)`-close if v and w are `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`.