Privacy Proofs for OpenDP: Row Transform

Grace Tian

Summer 2021

Contents

1 Algorithm Implementation

1.1 Code in Rust

The current OpenDP library contains the `make_row_by_row` function implementing the row transform function. This is defined in lines 10-26 of the file `manipulation.rs` in the Git repository (https://github.com/opendp/opendp/blob/main/rust/opendp/src/trans/manipulation.rs#L10-L26).

```
9  /// Constructs a [`Transformation`] representing an arbitrary row-by-row transformation.
10 pub(crate) fn make_row_by_row<'a, DIA, DOA, M, F: 'static + Fn(&DIA::Carrier) -> DOA::Carrier>(
11     atom_input_domain: DIA,
12     atom_output_domain: DOA,
13     atom_function: F
14 ) -> Fallible<Transformation<VectorDomain<DIA>, VectorDomain<DOA>, M, M>>
15     where DIA: Domain, DOA: Domain,
16           DIA::Carrier: 'static, DOA::Carrier: 'static,
17           M: DatasetMetric {
18     Ok(Transformation::new(
19         VectorDomain::new(atom_input_domain),
20         VectorDomain::new(atom_output_domain),
21         Function::new(move |arg: &Vec<DIA::Carrier>|
22             arg.iter().map(|v| atom_function(v)).collect()),
23         M::default(),
24         M::default(),
25         StabilityRelation::new_from_constant(1_u32)))
26 }
```

1.2 Pseudo Code in Python

Preconditions

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

- **User-specified types:**
  - Variable `atom_input_domain` has type `DIA`
  - Variable `atom_output_domain` has type `DOA`
  - Variable `atom_function` has type `F`
  - Types `DIA` and `DOA` have trait `Domain`
  - Type `F` has trait `Fn(&DIA::Carrier) -> DOA::Carrier)`

Postconditions

- Either a valid `Transformation` is returned or an error is returned.

```
def make_row_by_row(atom_input_domain : DIA, atom_output_domain : DOA, atom_function : F):
    input_domain = VectorDomain(DIA);
    output_domain = VectorDomain(DOA)
    input_metric = SymmetricDistance()
    output_metric = SymmetricDistance()

    def Relation(d_in : u32, d_out : u32) -> bool:
        return d_out <= d_in*1

    def function(data : Vec[DIA]) -> Vec[DOA]:
        return list(map(atom_function, data))

    return Transformation(input_domain, output_domain, function, input_metric, output_metric, stability_relation=Relation)
```

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 parameters (`atom_input_domain`, `atom_output_domain`, `atom_function`) to `make_row_by_row` such that the given preconditions hold, the transformation returned by `make_row_by_row` has the following properties:*

1. (Appropriate output domain). *For every element v in `input_domain`, `function`(v) is in `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 every pair (`d_in`, `d_out`), where `d_in` is of the associated type for `input_metric` and `d_out` is the associated type for `output_metric`, if v, w are `d_in`-close under `input_metric` and `Relation`(`d_in`, `d_out`) $=$ `True`, then `function`(v), `function`(w) are `d_out`-close under `output_metric`.*

Proof. 1. **(Appropriate output domain).** In the case of `make_row_by_row`, this corresponds to showing that for every vector v of elements of type `DIA`, `function`(v) is a vector of elements of type `DOA`.

The `function`(v) has type `Vec[DOA]` follows from the assumption that element v is in `input_domain` and from the type signature of `function` in line 10 of the pseudocode (Section 1.2), which takes in an element of type `Vec(DIA)` and returns an element of type `Vec[DOA]`. If the Rust code compiles correctly, then the type correctness follows from the definition of the type signature enforced by Rust. Otherwise, the code raises a compile time error for incorrect input type. (grace) Silvia wrote that it raises compile time error for incorrect input type, does it do the same for incorrect output type?

(grace) I think checking type signature is sufficient for this pf.

2. **(Domain-metric compatibility).** The Symmetric distance is both the `input_metric` and `output_metric`. Symmetric distance is compatible with `VectorDomain(T)` for any generic type `T`, as stated in "List of definitions used in the pseudocode". The theorem holds because for `make_row_by_row`, the input domain is `VectorDomain(DIA)` and the output domain is `VectorDomain(DOA)`.

3. **(Stability guarantee).** From Lemma 3.1 in "List of definitions used in the proofs" on the symmetric distance of row transform, we know that

$$d_{Sym}(\texttt{function}(v), \texttt{function}(w)) \leq d_{Sym}(v, w).$$

Because $\texttt{Relation}(\texttt{d\_in}, \texttt{d\_out}) = \texttt{True}$, it follows that $\texttt{d\_in} \leq \texttt{d\_out}$ by the `is_equal` stability relation defined in the pseduocode. Since vector inputs v, w are `d_in`-close, then the symmetric distance is bounded by `d_in` by definition the symmetric distance is bounded by d_{in}: $d_{Sym}(v, w) \leq \texttt{d\_in}$. It finally follows that the transformations are `d_out`-close: $d_{Sym}(\texttt{function}(v), \texttt{function}(w)) \leq \texttt{d\_out}$.

□