Privacy Proofs for OpenDP: Impute Uniform Float Transformation

Grace Tian

Summer 2021

Contents

1 Algorithm Implementation

1.1 Code in Rust

The current OpenDP library contains the `make_impute_uniform_float` function implementing the impute uniform float function. This is defined in lines 14-30 of the file `impute.rs` in the Git repository https://github.com/opendp/opendp/blob/21-impute/rust/opendp/src/trans/impute.rs#L14-L30

```rust
12  /// A [`Transformation`] that imputes elementwise with a sample from Uniform(lower, upper).
13  /// Maps a Vec<T> -> Vec<T>, where the input is a type with built-in nullity.
14  pub fn make_impute_uniform_float<M, T>(
15      lower: T, upper: T,
16  ) -> Fallible<Transformation<VectorDomain<InherentNullDomain<AllDomain<T>>>, VectorDomain<AllDomain<T>>, M, M>>
17      where M: DatasetMetric,
18            for<'a> T: 'static + Float + SampleUniform + Clone + Sub<Output=T> + Mul<&'a T, Output=T> + Add<&'a T, Output=T> + InherentNull {
19      if lower.is_nan() { return fallible!(MakeTransformation, "lower may not be nan"); }
20      if upper.is_nan() { return fallible!(MakeTransformation, "upper may not be nan"); }
21      if lower > upper { return fallible!(MakeTransformation, "lower may not be greater than upper") }
22      let scale = upper.clone() - lower.clone();
23
24      make_row_by_row_fallible(
25          InherentNullDomain::new(AllDomain::new()),
26          AllDomain::new(),
27          move |v| if v.is_null() {
28              T::sample_standard_uniform(false).map(|v| v * &scale + &lower)
29          } else { Ok(v.clone()) })
30  }
```

1.2 Pseudo Code in Python

Preconditions

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

- **User-specified types:**

  - Variables `lower` and `upper` must be of type `T`
  - Type `T` must have traits `float`, `SampleUniform`, `Clone`, `Sub(Output=T)`, `Mul`, `Add`, and `InherentNull`.

Postconditions

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

```python
def make_impute_uniform_float(lower : T, upper : T):
    input_domain = VectorDomain(InherentNullDomain(AllDomain(T)));
    output_domain = VectorDomain(AllDomain(T))
    input_metric = SymmetricDistance()
    output_metric = SymmetricDistance()

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

    # should input to function include inherent null?
    def function(data : Vec(T)) -> Vec(T):
        return list(map(Uniform(lower, upper), data))

    let stability_relation = (d_in <= d_out);

    return Transformation(input_domain, output_domain, function,
    input_metric, output_metric, stability_relation)
    # TODO replace with return row_by_row_fallible
```

2 Proof

Theorem 2.1. *For every setting of the input parameters* `(lower, upper)` *to* `make_impute_uniform_float` *such that the given preconditions hold, the transformation returned by* `make_impute_uniform_float` *has the following properties:*

1. (Appropriate output domain). *If vector v is in the* `input_domain`*, then* `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 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_impute_uniform_float`, this corresponds to showing that for every vector v of elements of type `InherentNullDomain(T)` (grace) T? or InherentNullDomain???, `function`(v) is a vector of elements of type `T`.

(grace) TODO We show the type signature + nullity works

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_impute_constant`, the input domain is `VectorDomain(InherentNullDomain(AllDomain(T)))` and the output domain is `VectorDomain(AllDomain(T))`.

3. **(Stability guarantee).**

   We know that vectors v, w are d_{in}-close, and that $d_{in} \leq d_{out}$ because `Relation`(d_{in}, d_{out}) `= True`. By the histogram notation, this means that

   $$d_{Sym}(v, w) = \|h_v - h_w\|_1 = \sum_z |h_v(z) - h_w(z)| \leq d_{in}.$$

   Recall that the `make_impute_uniform_float` transformation only changes the null values in the vectors v and w. Therefore it suffices to consider only the subset of null elements in $Multiset(v)$ and $Multiset(w)$, which we denote respectively as v^* and w^*.

   From the histogram notation, we have $h_v(\texttt{null})$ and $h_w(\texttt{null})$ nulls respectively in vectors v and w. By the stability for randomness corollary, we can fix the random seed r, and say it produces the sequence $(r_1, r_2, r_3, \ldots)$ of randomly generated uniforms from `Unif(lower, upper)` in this specific order. In other words, the ith `null` in v or w corresponds to r_i in function(v) or function(w). Therefore the symmetric distance of $function(v^*)$ and `function`(w^*) is bounded:

   $$\sum_{r_i \in r} \left|h_{\texttt{function}(v^*)}(r_i) - h_{\texttt{function}(w^*)}(r_i)\right| \leq |h_v(\texttt{null}) - h_w(\texttt{null})|$$

   The remaining non-null values in v and z stay the same after the transformation, so the transformations are d_{out}-close:

   $$d_{sym}(\texttt{function}(v), \texttt{function}(w)) = \sum_z \left|h_{\texttt{function}(v)}(z) - h_{\texttt{function}(w)}(z)\right|$$

   $$\leq \sum_z |h_v(z) - h_w(z)| \leq d_{in} \leq d_{out}$$

   $\square$