Written by Grace Tian

1 Impute Uniform Float(TODO)

1.1 Code in Rust

Rust code here https://github.com/opendp/opendp/blob/21-impute/rust/opendp/src/trans/impute.rs

```rust
/// A [`Transformation`] that imputes elementwise with a sample from Uniform(lower, upper).
/// Maps a Vec<T> -> Vec<T>, where the input is a type with built-in nullity.
pub fn make_impute_uniform_float<M, T>(
    lower: T, upper: T,
) -> Fallible<Transformation<VectorDomain<InherentNullDomain<AllDomain<T>>>, VectorDomain<AllDomain<T>>, M, M>>
    where M: DatasetMetric,
          for<'a> T: 'static + Float + SampleUniform + Clone + Sub<Output=T> + Mul<&'a T, Output=T> + Add<&'a T, Output=T> + InherentNull {
    if lower.is_nan() { return fallible!(MakeTransformation, "lower may not be nan"); }
    if upper.is_nan() { return fallible!(MakeTransformation, "upper may not be nan"); }
    if lower > upper { return fallible!(MakeTransformation, "lower may not be greater than upper") }
    let scale = upper.clone() - lower.clone();

    make_row_by_row_fallible(
        InherentNullDomain::new(AllDomain::new()),
        AllDomain::new(),
        move |v| if v.is_null() {
            T::sample_standard_uniform(false).map(|v| v * &scale + &lower)
        } else { Ok(v.clone()) })
}
```

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 `static 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()

    assert .... (TODO);


    def function(data):
        return Uniform(lower, upper) if data is null else return data;

    let stability_relation = (d_in <= d_out);

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

1.2.1 Conditions as specified in Pseudocode (Delete?)

- Input Domain: all vector domain of type `T` and null values
- Output Domain: all vector domain of type `bool`
- Function: return vector where null values are replaced with Uniform(lower, upper) sampling
- Input Metric: Symmetric Distance
- Output Metric: same as Input Metric
- Stability Relation: function that takes in 32 bit integers d_{in} and d_{out} and returns whether $d_{in} \leq d_{out}$.

1.3 Proof

Theorem 1.1. *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` in input domain, then `function(v)` is in output domain.* *(grace) Do I need to say otherwise the program will raise an `Exception`?*

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). *If two vector inputs v, w are "d_{in} - close" and if `Relation(`d_{in}`,` d_{out}`) = True` then the corresponding outputs $function(v), function(w)$ are "d_{out} - close".*

Proof. 1. **(Appropriate output domain).** The output correctness follows from the type signature of the function in the rust code.

2. **(Domain-metric compatibility).**

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(`null`) and h_w(`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:

$$\begin{aligned} d_{sym}(\texttt{function}(v), \texttt{function}(w)) &= \sum_z \left| h_{\texttt{function}(v)}(z) - h_{\texttt{function}(w)}(z) \right| \\ &\leq \sum_z |h_v(z) - h_w(z)| \leq d_{in} \leq d_{out} \end{aligned}$$

□

2 Impute Constant (TODO)

```rust
/// A [`Transformation`] that imputes elementwise with a constant value.
/// Maps a Vec<Option<T>> -> Vec<T> if input domain is AllDomain<Option<T>>,
///     or Vec<T> -> Vec<T> if input domain is NullableDomain<AllDomain<T>>
/// Type argument DA is "Domain of the Atom"; the domain type inside VectorDomain.
pub fn make_impute_constant<DA, M>(
    constant: DA::NonNull
) -> Fallible<Transformation<VectorDomain<DA>, VectorDomain<AllDomain<DA::NonNull>>, M, M>>
    where DA: ImputableDomain,
          DA::NonNull: 'static + Clone,
          DA::Carrier: 'static,
          M: DatasetMetric {
    if DA::is_null(&constant) { return fallible!(MakeTransformation, "Constant may not be null.") }

    make_row_by_row(
        DA::new(),
        AllDomain::new(),
        move |v| DA::impute_constant(v, &constant).clone())
}
```

2.1 Pseudo code

Impute from page 4 in Smart Noise framework: https://github.com/opendp/smartnoise-core/blob/develop/whitepapers/data_processing/data_processing.pdf

```
def make_impute_constant(constant, input_metric, output_metric):
    let input_domain = VectorDomain<AllDomain<Option<T>>;
    let output_domain = VectorDomain<AllDomain<T>;
    let function(data): return constant;

    let stability_relation = (d_out >= d_in);

    return Transformation(input_domain, output_domain, function(data?),
   input_metric, output_metric, stability_relation)
```

3 Proof

Theorem 3.1. *For every setting of the input parameters to* `make_impute_constant`*, the transformation returned by* `make_impute_constant` *has the following properties*

1. (Appropriate output domain). *If vector* `v` *of type* `T` *is in the input domain, then* `function(v)` *is in the output domain, otherwise the program will raise an* `Exception`*.*

2. (Stability Guarantee). *If two vector inputs v, w are "d_{in} - close" and if* `Relation(`d_{in}`,` d_{out}`) = True` *then the corresponding outputs $function(v), function(w)$ are "d_{out} - close".*

Proof. 1.

2. We know that $d_{in} \leq d_{out}$ because `Relation(`d_{in}, d_{out}`) = True`. Since the vectors v, w are d_{in}-close, then $d_{Sym}(v, w) \leq d_{in}$.

   The function transformation just replaces the `null` element in vectors v and w with `constant`. Since the null element is also counted toward the symmetric distance of the transformation, the symmetric distance of `function`(v) and `function`(w) stays the same. Therefore the transformation is d_{out} close: $d_{sym}(\texttt{function}(v), \texttt{function}(w)) = d_{sym}(v, w) \leq d_{in} \leq d_{out}$

$\square$