Privacy Proofs for OpenDP: Geometric Measurement

Vicki Xu, Hanwen Zhang, Zachary Ratliff

Summer 2022

Contents

1 Algorithm Implementation

1.1 Code in Rust

The current OpenDP library contains the `make_base_geometric` function implementing geometric measurement. This is defined in the adjacent `mod.rs` file.

In `make_base_geometric`, which accepts a parameter `scale` of type `QO` and a parameter `bounds` of type `Option<(D::Atom, D::Atom)>`, the function takes in a data point `arg` of type `D::Atom`, `scale`, and `bounds`. This function will call `sample_two_sided_geometric`, which provides a sampling from the two-sided geometric distribution centered at `arg` with scale parameter `scale` and saturated at the bounds denoted by `bounds`. If `bounds` is not provided, then the two-sided geometric sample saturates at the bounds of the data type, `MAX` and `MIN`. The function returns the sampled element given the parameters.

1.2 Pseudo Code in Python

We present a simplified Python-like pseudocode of the Rust implementation below. The necessary definitions for the non-sampler pseudocode can be found at "List of definitions used in the pseudocode". (vicki) fix link For samplers (such as `sample_bernoulli`), the definitions will be found at (vicki) insert link here

Preconditions

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

- **User-specified types:**
  - Variable `scale` must be of type `QO`
  - Variable `bounds` must be of type `Option(D::Atom, D::Atom)`
  - Type `D` must have trait `GeometricDomain`
  - Type `D::Atom` must have trait `Integer`
  - Type `QO` must have traits `Float` and `InfCast<D::Atom>`

Postconditions

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

```
# implement noise function for scalar input
def noise_function<AllDomain<T>>(scale : QO, bounds? : {T, T}):
    def function(arg : D::Atom, scale : QO, bounds?: {D::Atom, D::Atom}) ->
     <D, QO>:
        return sample_two_sided_geometric(arg, scale, bounds)

    return function

# implement noise function for vector input
def noise_function<VectorDomain<AllDomain<T>>(scale : QO, bounds? : {T, T})
    :
    def function(arg : Vector<D::Atom>, scale : QO, bounds?: {D::Atom, D::
    Atom}) -> <D, QO>:
        noised = new Vector<AllDomain<T>>
        for (item in arg):
           noised.add(sample_two_sided_geometric(item, scale, bounds))

        return noised

    return function

# parameterize by domain D
def make_base_geometric<D>(scale: QO, bounds={"lower": D::Atom::MIN, "upper
    ": D::Atom::MAX}):
    input_domain = D
    output_domain = D
    # --------------------------------
    # To reduce redundancy, we don't copy and paste the whole function.
    # The idea is to have different input metrics for two input settings.

    # check for scalar or vector input
    if isinstance(D, AllDomain<T>):
        input_metric = AbsoluteDistance<D::Atom>
    elif isinstance(D, VectorDomain<AllDomain<T>>):
        input_metric = L1Distance<D::Atom>
    # --------------------------------

    similarity_metric = MaxDivergence<QO>
    function = input_domain.noise_function(scale, bounds)

    # check scale sign
    if (scale < 0):
        raise Exception("scale must not be negative")

    # check bounds - can discuss if python dict is best representation of
    this
    if (bounds["lower"] > bounds["upper"]):
        raise Exception("lower may not be greater than upper")

    def privacy_map(d_in: D::Atom) -> QO:
        if (d_in < 0):
            raise Exception("sensitivity must be non-negative")
        if (scale == 0):
```

```
            return QO::MAX
        return inf_div(d_in, scale)

    return Measurement(input_domain, output_domain, function, input_metric,
     similarity_metric, privacy_map)
```

(vicki) marked bounds with [SELF::MIN, SELF::MAX] because it's an optional type in the original Rust code, and if not specified, then in SampleTwoSidedGeometric SELF::MIN and SELF::MAX are substituted

1.3 Proof

Theorem 1.1. *For every setting of the input parameters* `constant` *to* `make_base_geometric` *such that the given preconditions hold, the transformation returned by* `make_base_geometric` *satisfies the following statements:*

1. (Domain-metric compatibility.) *The domain* `input_domain` *matches one of the possible domains listed in the definition of* `input_metric`*.*

2. (Privacy guarantee.) *Let* `d_in` *have the associated type for* `input_metric`*, and let* D *have associated type for* `similarity_metric`*. For every pair of elements* v, w *in* `input_domain` *and every* `d_in`*, if* v, w *are* `d_in`*-close under* `input_metric`*, then* `function`(v)*,* `function`(w) *are* `map(d_in)`*-close with respect to* D*.*

Proof.

1. **(Domain-metric compatibility.)** For `make_base_geometric`, this corresponds to showing `AllDomain(T)` is compatible with `AbsoluteDivergence`. This follows directly from the definition of `AbsoluteDivergence`, as stated in the "List of definitions used in the pseudocode".

2. **(Privacy guarantee.)** The following proof is built up on Theorem 2.1 in the proof of `sample_two_sided_geometric(shift, scale, bounds)`, which is the following: (hanwen) link to be put here

   > **Theorem 1.2.** *For every setting of input parameters* `shift, scale, bounds` *such that the preconditions hold,* `sample_two_sided_geometric` *returns a draw from the censored two-sided geometric distribution with scale parameter* `scale`*, centered at* `shift`*, and saturated at the bounds denoted by* `bounds`*.*

   Before we begin our proof, we note that the measurement is parameterized by a domain `D` of type `AllDomain<T>` or `VectorDomain<AllDomain<T>>`.

   Let v, w be two datasets that are `d_in`-close with respect to `input_metric`, which is defined to be `L1Distance` with the vector case (`D = VectorDomain<AllDomain<T>>`) or `AbsoluteDistance` with the scalar case (`D = AllDomain<T>`). We only prove the case for when `D = VectorDomain<AllDomain<T>>` since the scalar case trivially follows.

   According to its definition in the proof definition document, `d_in`$= d_{L1}(v, w) = \sum_{i=1}^{n} |v_i - w_i|$, where n is the dimension of v, w. Here it is implicitly assumed that $|v| = |w|$.

Let $s =$ `scale`, $[a, b] =$ `bounds`, $Y =$ `function`(v), and $Z =$ `function`(w). Our goal is to show that `MaxDivergence` $D_\infty(Y||Z)$, also defined in the proof definition document, adheres to the privacy guarantee.

Observe that by the definition of `noise_function` implemented for `VectorDomain<AllDomain<T>>`, and by Theorem 1.2, it follows that `function`() returns a random variable $Z \sim$ `shift` + Geo(`scale`), with probability mass function $f_Z(z) \propto \exp\{-|z - \texttt{shift}|/\texttt{scale}\}$ along each axis of the input vector. So it follows that

$$D_\infty(Y||Z) = \max_{S \subseteq Supp(Y)} \left[\ln\left(\frac{\Pr[Y \in S]}{\Pr[Z \in S]}\right)\right]$$

$$= \max_{y \in Supp(Y)} \left[\ln\left(\frac{\Pr[\texttt{function}(v) = y]}{\Pr[\texttt{function}(w) = y]}\right)\right]$$

$$= \max_{y \in Supp(Y)} \sum_{i=1}^{n} \left[\ln\left(\frac{\exp\{-|y_i - v_i|/s\}}{\exp\{-|y_i - w_i|/s\}}\right)\right]$$

$$= \max_{y \in Supp(Y)} \sum_{i=1}^{n} [\ln(\exp\{(|y_i - w_i| - |y_i - v_i|)/s\})]$$

$$= \max_{y \in Supp(Y)} \frac{1}{s} \sum_{i=1}^{n} |y_i - w_i| - |y_i - v_i|$$

$$\leq \frac{1}{s} \sum_{i=1}^{n} |w_i - v_i| \quad \text{by the Triangle inequality}$$

$$= \frac{1}{s} \sum_{i=1}^{n} |w_i - v_i| = \texttt{d\_in}/\texttt{scale}$$

Failure cases. We still need to account for failure cases within the `privacy_map` code. There is one place the code raises an exception:

(a) `inf_div` fails. This step is only reached if `d_in` is nonnegative. As defined in the pseudocode definitions doc, `inf_div` throws an exception if division overflows from a 32-bit integer.

Otherwise, `privacy_map` returns `inf_div(d_in, scale)`.

□