fn make_noisy_top_k

Michael Shoemate

September 25, 2025

Proves soundness of make_noisy_top_k in mod.rs at commit f5bb719 (outdated[1]). make_noisy_top_k returns a Measurement that noisily selects the indices of the greatest scores from a vector of input scores.

1 Background

This mechanism fulfills the same purpose as the exponential mechanism, where the release is the best candidate's index from a finite set. The naive implementation of the exponential mechanism samples an index k from $[m] = 1, \ldots, m$, where m is the number of candidates, with probability p_i assigned to each candidate's index i as a function of their score s_i. The output is drawn via inverse transform sampling by outputting the smallest index k for which the cumulative probability is greater than some $u \sim \text{Uniform}(0, 1)$.

$$\mathcal{M}_{\text{naive}}([s_1, \ldots, s_m]) = \min\{k : \sum_{i=1}^{k} p_i \geq u\} \tag{1}$$

The probability of index k being selected is the normalization of its likelihood $\exp(s_k/\tau)$. As a candidate's score s_k increases, the candidate becomes exponentially more likely to be selected:

$$p_k = \frac{\exp(s_k/\tau)}{\sum_{i=1}^{m} \exp(s_i/\tau)} \tag{2}$$

This equation introduces a new temperature parameter, τ, which calibrates how distinguishable scores are from each other. As temperature increases, the categorical output distribution tends towards higher entropy/uniformity and becomes more privacy-preserving. As temperature decreases, the categorical distribution tends towards a one-hot vector (where each candidate has zero probability, except for the candidate with the maximum score, which has probability one), becoming less private. Temperature is related to the privacy loss parameter (d_out) and sensitivity of the scoring function (Δ) as follows:

$$\tau = \Delta/\texttt{d\_out} \tag{3}$$

A precise definition of Δ will come later and is captured by the metrics we use on the score vector $s = [s_1, \ldots, s_m]$. When d_out increases, temperature decreases, and candidates become more distinguishable from each other. We also divide scores by their global sensitivity to normalize the sensitivity to one. In the differential privacy literature for the exponential mechanism, the sensitivity is often multiplied by two. In OpenDP's make_noisy_top_k this factor is bundled into the Δ term, which is expressed in terms of a metric that captures monotonicity.

[1] See new changes with git diff f5bb719..15da7ff rust/src/measurements/noisy_top_k/mod.rs

1.1 Sampling Vulnerabilities

In practice, computing $\exp(s_i/\tau)$ is prone to zero underflow (where a non-zero quantity rounds down to zero) and overflow (where a large finite quantity is replaced with infinity) due to finite/limited data representation. Specifically, a scaled score s_i/τ of just -709 underflows to zero and $+710$ overflows to infinity when stored in a 64-bit float.

A simple improvement is to shift the scores by subtracting the greatest score from all scores. In idealized arithmetic, the resulting probabilities are not affected by shifts in the underlying scores. On finite data types, this shift prevents a catastrophic overflow, but makes underflow more likely, causing tail values of the distribution to round to zero. The inverse transform sampling step is also subject to accumulated rounding errors from the arithmetic and sum, which influence the likelihood of being chosen.

These potential vulnerabilities can be addressed via the Gumbel-max trick. The naive mechanism $\mathcal{M}_{\text{naive}}$ implemented with infinite-precision arithmetic is equivalent in distribution to the following mechanism:

$$\mathcal{M}([s_1, \ldots, s_m]) = \text{argmax}_i g_i, \tag{4}$$

where each $g_i \sim \text{Gumbel}(\mu = s_i, \beta = \tau)$.

1.2 Noise Distribution

`make_noisy_top_k` can also be configured to satisfy either `MaxDivergence` or `ZeroConcentratedDivergence`. Gumbel noise is used when `output_measure` is `ZeroConcentratedDivergence`, and exponential noise is used when `output_measure` is `MaxDivergence`. These choices of noise distributions minimize the necessary noise variance for their respective privacy measures.

Since the permute-and-flip mechanism is equivalent to report noisy max exponential, and it can be implemented with discrete distributions, the permute-and-flip mechanism is used instead.

1.3 Top K

In the case of Gumbel noise, the distribution of the top k indices is equivalent to adding gumbel noise once, and then returning the top k indices. This one-shot mechanism avoids needing to peel the selected candidate from the candidate set and re-run the mechanism.

However, this peeling routine is still used for the permute-and-flip mechanism, to release the top k elements. The privacy argument proceeds via composition.

2 Hoare Triple

Precondition

Compiler-verified

- `MO` is a type with trait `TopKMeasure`
- `TIA` (atomic input type) is a type with trait `Number`

Caller-verified

None

Pseudocode

```
def make_noisy_top_k(
    input_domain: VectorDomain[AtomDomain[TIA]],
    input_metric: LInfDistance[TIA],
    privacy_measure: MO,
    k: usize,
    scale: f64,
    negate: bool,
) -> Measurement:
    if input_domain.element_domain.nan():  #
        raise "input domain elements must be non-nan"

    if input_domain.size is not None:
        if k > input_domain.size:
            raise "k must not exceed the number of candidates"

    if not scale.is_finite() or scale.is_sign_negative():  #
        raise "scale must be finite and non-negative"

    monotonic = input_metric.monotonic

    def privacy_map(d_in: TIA):  #
        # convert to range distance
        d_in = d_in if monotonic else d_in.inf_add(d_in)
        d_in = f64.inf_cast(d_in)  #

        if d_in.is_sign_negative():  #
            raise "sensitivity must be non-negative"

        if d_in.is_zero():  #
            return 0.0

        if scale.is_zero():  #
            return f64.INFINITY

        #
        return MO.privacy_map(d_in, scale).inf_mul(f64.inf_cast(k))

    return Measurement.new(
        input_domain=input_domain,
        input_metric=input_metric,
        output_measure=privacy_measure,
        function=lambda x: MO.noisy_top_k(x, scale, k, negate),
        privacy_map=privacy_map,
    )
```

Postcondition

Theorem 2.1. For every setting of the input parameters `input_domain, input_metric, output_measure, k, scale, negate, MO, TIA` to `make_noisy_top_k` such that the given preconditions hold, `make_noisy_top_k` raises an error (at compile time or run time) or returns a valid measurement. A valid measurement has the following properties:

1. (Data-independent runtime errors). For every pair of members x and x' in `input_domain`, `invoke`(x) and `invoke`(x') either both return the same error or neither return an error.

2. (Privacy guarantee). For every pair of members x and x' in `input_domain` and for every pair (`d_in`, `d_out`), where `d_in` has the associated type for `input_metric` and `d_out` has the associated type for `output_measure`, if x, x' are `d_in`-close under `input_metric`, `privacy_map`(`d_in`) does not raise

an error, and privacy_map(d_in) = d_out, then function(x), function(x') are d_out-close under output_measure.

Proof of data-independent errors. By the postcondition of TopKMeasurenoisy_top_k, the only source of error is due to entropy exhaustion, which could be data-dependent, due to differing number of expected random draws depending on the input dataset.

Therefore, the mechanism only satisfies the requirement for data-independent errors conditioned on entropy not being exhausted. □

Proof of privacy guarantee. When d_in is zero, by line 29, the privacy loss is zero, satisfying the postcondition. Otherwise when scale is zero, by line 32, the privacy loss is infinite, also satisfying the postcondition.

By the checks on lines 9 and 16, the preconditions for TopKMeasurenoisy_top_k are satisfied. Additionally by the checks on lines 26, 29 and 32, the preconditions for TopKMeasureprivacy_map are satisfied.

By the postcondition of TopKMeasure and adaptive composition, the d_outon line 35 satisfies the postcondition. □