fn make_base_discrete_exponential

Michael Shoemate

October 11, 2023

This proof resides in **"contrib"** because it has not completed the vetting process.

Proves soundness of `make_base_discrete_exponential` in `mod.rs` at commit f5bb719 (outdated[1]).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

This released index can be later be used to index into a public candidate set (postprocessing).

Vetting History

- Pull Request #456

The naive implementation samples some index k from a categorical distribution, with probabilities assigned to each candidate relative to their score. We may use inverse transform sampling to select the smallest index k for which the cumulative probability is greater than some $U \sim Uniform(0, 1)$.

$$M(s) = argmin_k \sum_i^k p_i >= U \tag{1}$$

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

$$p_k = \frac{e^{s_k/\tau}}{\sum_i e^{s_i/\tau}} \tag{2}$$

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 entropy-/uniformity and becomes more privacy preserving. As temperature decreases, the categorical distribution tends towards a one-hot vector, becoming less private. Temperature is related to ϵ and the sensitivity (Δ) of the scoring function as follows:

$$\tau = \Delta/\epsilon \tag{3}$$

When ϵ 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 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..86d3c6d rust/src/measurements/discrete_exponential/mod.rs`

1 Gumbel Reparameterization

In practice, computing $e^{s_i/\tau}$ is prone to zero underflow and overflow. Specifically, a scaled score 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 is also subject to accumulated rounding errors from the arithmetic and sum, which influence the likelihood of being chosen.

The Gumbel-max trick may instead be used to privately select an index. Let $K = argmax_k G_k$, a random variable representing the selected index. Denote the k^{th} noisy score as $G_k \sim Gumbel(\mu = s_k/\tau)$. K can be sampled via an inverse transform, where u_k is sampled iid uniformly from $(0, 1)$:

$$M(s) = argmax_k(s_k/\tau - log(-log(u_k))) \tag{4}$$

Theorem 1.1. Sampling from K is equivalent to sampling from the softmax, because $P(K = k) = p_k$. [1]

$$\begin{aligned}
P(K = k) &= P(G_k = max_i G_i) && \text{by definition of K} \\
&= P(-log(Z_k/N) = max_i - log(Z_k/N)) && \text{by 1.2} \\
&= P(log(Z_k/N) = min_i log(Z_k/N)) && \text{since } max - a_i = -min_i a_i \\
&= P(Z_k = min_i Z_i) && \text{simplify monotonic terms} \\
&= P(Z_k \leq min_{i \neq k} Z_i) \\
&= P(Z_k \leq Q) && \text{by 1.3 where } Q \sim Exp(\sum_{i \neq k} p_i) \\
&= \frac{p_k}{p_k + \sum_{i \neq k} p_i} && \text{by 1.4} \\
&= p_k && \text{since } p_k + \sum_{i \neq k} p_i = 1
\end{aligned}$$

Lemma 1.2. $G_k = -log(Z_k/N)$ where $Z_k \sim Exp(p_k)$ and normalization term $N = \sum_i e^{s_i/\tau}$.

$$\begin{aligned}
G_k &= s_k/\tau - log(-log(U_k)) && \text{Gumbel PDF centered at } s_k/\tau \\
&= log(e^{s_k/\tau}) - log(-log(U_k)) \\
&= log(p_k N) - log(-log(U_k)) && \text{since } p_k = e^{s_k/\tau}/N \\
&= log(p_k N/(-log(U_k))) \\
&= -log(-log(U_k)/(p_k N)) \\
&= -log(Z_k/N) && \text{substitute } Z_k = -log(U_k)/p_k
\end{aligned}$$

Lemma 1.3. If $X_1 \sim Exp(\lambda_1)$, $X_2 \sim Exp(\lambda_2)$ and $Z \sim Exp(\lambda_1 + \lambda_2)$, then $min(X_1, X_2) \sim Z$.

$$\begin{aligned}
P(min(X_1, X_2) \geq x) &= P(X_1 \geq x)P(X_2 \geq x) && \text{by independence} \\
&= e^{-\lambda_1 x} e^{-\lambda_2 x} && \text{substitute exponential density} \\
&= e^{-(\lambda_1 + \lambda_2)x} \\
&= P(Z \geq x) && \text{substitute exponential density}
\end{aligned}$$

Lemma 1.4. If $X_1 \sim Exp(\lambda_1)$, $X_2 \sim Exp(\lambda_2)$, then $P(X_1 \leq X_2) = \frac{\lambda_1}{\lambda_1+\lambda_2}$.

$$\begin{aligned} P(X_1 \leq X_2) &= \int_0^\infty \int_{x_1}^\infty \lambda_1 \lambda_2 e^{-\lambda_1 x_1} e^{-\lambda_2 x_2} \, dx_1 dx_2 \\ &= \int_0^\infty -\lambda e^{-(\lambda_1+\lambda_2)x_1} \, dx_1 \\ &= \frac{\lambda_1}{\lambda_1 + \lambda_2} \end{aligned}$$

1.1 Metric

We need a metric that captures the distance between score vectors u and v respectively on neighboring datasets. The i^{th} element of each score vector is the score for the i^{th} candidate. The sensitivity of the scoring function can be measured in terms of the L_∞ norm, which we name the `LInfDistance`. It characterizes the greatest that any one score may change:

$$\Delta_\infty = \max_{u \sim v} d_\infty(f(u), f(v)) = \max_{u \sim v} \max_i |f(u)_i - f(v)_i| \tag{5}$$

Unfortunately, this choice of metric always results in a loosening by a factor of 2 when evaluating the privacy guarantee of the exponential mechanism. This is because both the i^{th} likelihood and normalization term may vary in opposite directions, resulting in a more distinguishing event. However, this loosening is not necessary if we can prove that the scoring function is monotonic, because the i^{th} likelihood and normalization term will always vary in the same direction.

We instead use a slight adjustment to this metric, `LInfDiffDistance`, characterizing the greatest difference in scores:

$$\Delta_{\infty'} = \max_{u \sim v} d_{\infty'}(f(u), f(v)) = \max_{u \sim v} \max_{ij} |(f(u)_i - f(v)_i) - (f(u)_j - f(v)_j)| \tag{6}$$

Consider when the scoring function is not monotonic. The sensitivity is maximized when $u_i - v_i$ and $u_j - v_j$ vary maximally in opposite directions, resulting in the same loosening factor of 2. On the other hand, when the scoring function is monotonic, the sign of the $u_i - v_i$ term matches the sign of the $u_j - v_j$ term, and their magnitudes cancel. Therefore, when the scorer is monotonic, the sensitivity is maximized when one term is zero. It is shown in 3.1 that a tighter analysis of the exponential mechanism is compatible with a score vector whose sensitivity is expressed in terms of this metric.

2 Hoare Triple

Precondition

- `TIA` (input atom type) is a type with traits `Number` and `CastInternalRational`
- `QO` (output distance type) is a type with traits `Float`, `CastInternalRational` and `DistanceConstant` from type `TIA`

Function

```
def make_base_discrete_exponential(
    input_domain: VectorDomain[AtomDomain[TIA]],
    input_metric: LInfDiffDistance[TIA]
    temperature: QO,
    optimize: Union[Literal["max"], Literal["min"]]
```

```
) -> Measurement:
    if input_domain.element_domain.nullable:
        raise ValueError("input domain must be non-nullable")

    if temperature <= 0:
        raise ValueError("temperature must be positive")

    if optimize == "max":
        sign = +1
    elif optimize == "min":
        sign = -1
    else:
        raise ValueError("must specify optimization")

    temp_frac = Fraction(temperature)

    def function(scores: List[TIA]):
        def map_gumbel(score):
            return GumbelPSRN(shift=sign * Fraction(score) / temp_frac)
        gumbel_scores = map(map_gumbel, scores)

        def reduce_best(a, b):
            return a if a[1].greater_than(b[1]) else b
        return reduce(reduce_best, enumerate(gumbel_scores))[0]

    def privacy_map(d_in: TIA):
        d_in = QO.inf_cast(d_in)
        if d_in < 0:
            raise ValueError("input distance must be non-negative")

        if d_in == 0:
            return 0

        return d_in.inf_div(temperature)

    return Measurement(
        input_domain=input_domain,
        function=function,
        input_metric=input_metric,
        output_metric=MaxDivergence(QO),
        privacy_map=privacy_map,
    )
```

Postcondition

For every setting of the input parameters `input_domain, input_metric, temperature, optimize, TIA, QO` to `make_base_discrete_exponential` such that the given preconditions hold, `make_base_discrete_exponential` raises an exception (at compile time or run time) or returns a valid measurement. A valid measurement has the following property:

1. (Privacy guarantee). For every pair of elements u, v 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 u, v are `d_in`-close under `input_metric`, `privacy_map(d_in)` does not raise an exception, and `privacy_map(d_in)` $\leq$ `d_out`, then `function`(u), `function`(v) are `d_out`-close under `output_measure`.

3 Proof

3.1 Privacy Guarantee

To ensure that the Gumbel sample is valid, the `input_domain` is required to be non-null. The temperature is also required to be positive, because epsilon is non-negative, and to avoid division by zero.

Technically, the input domain is required to have known size, because neighboring sets of scores with different sizes are incomparable. The drawback to enforcing this requirement is that it requires the user to specify the size of the input domain in advance, which is inconvenient. Instead, we allow the input domain to have unknown size, and define the distance between two score vectors with different lengths to be infinite. Therefore, in the map, if the input distance is finite, then we only need to consider neighboring score vectors with the same size. For this reason, the domain descriptor about the dataset size is ignored.

Lemma 3.1. By the definition of `function`in the pseudocode, for any x in `input_domain`, $\Pr[\texttt{function}(x) = i] = \Pr[\text{argmax}_k(u_k/\tau - \ln(-\ln(U_k))) = i]$.

Proof. For each score s_k, `function` samples a Gumbel random variable centered at $sign \cdot s_k/\tau$. The choice of sign does not affect the privacy guarantee, so we omit it from further analysis. Sampling from a Gumbel distribution is equivalent to adding a draw from $-\ln(-\ln(U_k))$, where $U_k \sim Uniform(0, 1)$. The algorithm only returns the index of the maximum Gumbel random variable, therefore the probability of returning i is the probability that the i^{th} Gumbel random variable is the maximum. □

Lemma 3.2. Assume u, v in `input_domain`. Then $\ln\left(\frac{\sum_i \exp\left(\frac{\epsilon v_i}{\Delta}\right)}{\sum_i \exp\left(\frac{\epsilon u_i}{\Delta}\right)}\right) \leq \frac{\epsilon \max_j(v_j - u_j)}{\Delta}$.

Proof.

$$
\begin{aligned}
\ln\left(\frac{\sum_i \exp\left(\frac{\epsilon v_i}{\Delta}\right)}{\sum_i \exp\left(\frac{\epsilon u_i}{\Delta}\right)}\right) &= \ln\left(\frac{\sum_i \exp\left(\frac{\epsilon(v_i - u_i + u_i)}{\Delta}\right)}{\sum_i \exp\left(\frac{\epsilon u_i}{\Delta}\right)}\right) \\
&= \ln\left(\frac{\sum_i \exp\left(\frac{\epsilon(v_i - u_i)}{\Delta}\right)\exp\left(\frac{\epsilon(u_i)}{\Delta}\right)}{\sum_i \exp\left(\frac{\epsilon u_i}{\Delta}\right)}\right) \\
&\leq \ln\left(\frac{\exp\left(\frac{\epsilon \max_j(v_j - u_j)}{\Delta}\right)\sum_i \exp\left(\frac{\epsilon(u_i)}{\Delta}\right)}{\sum_i \exp\left(\frac{\epsilon u_i}{\Delta}\right)}\right) \\
&= \frac{\epsilon \max_j(v_j - u_j)}{\Delta}
\end{aligned}
$$

□

Assume u, v in `input_domain` are `d_in`-close under `LInfDiffDistance` and `privacy_map(d_in)` $\leq$ `d_out`.

$$
\begin{aligned}
&\max_{u \sim v} D_\infty(\texttt{function}(u), \texttt{function}(v)) \\
&= \max_{u \sim v} \max_i \ln\left(\frac{\Pr[\texttt{function}(u) = i]}{\Pr[\texttt{function}(v) = i]}\right) && \text{by MaxDivergence} \\
&= \max_{u \sim v} \max_i \ln\left(\frac{\Pr[\text{argmax}_k(u_k/\tau - \ln(-\ln(U_k))) = i]}{\Pr[\text{argmax}_k(v_k/\tau - \ln(-\ln(U_k))) = i]}\right) && \text{by 3.1, substitute function}
\end{aligned}
$$

Assuming `privacy_map(d_in)` $\leq$ `d_out` $= \epsilon$, then $\tau \geq \Delta/\epsilon$.

$$
\begin{aligned}
&\leq \max_{u \sim v} \max_i \ln \left(\frac{\Pr[\operatorname{argmax}_k(u_k \epsilon/\Delta - \ln(-\ln(U_k))) = i]}{\Pr[\operatorname{argmax}_k(v_k \epsilon/\Delta - \ln(-\ln(U_k))) = i]} \right) \\
&= \max_{u \sim v} \max_i \ln \left(\frac{\exp\left(\frac{\epsilon u_i}{\Delta}\right)}{\sum_k \exp\left(\frac{\epsilon u_k}{\Delta}\right)} \Big/ \frac{\exp\left(\frac{\epsilon v_i}{\Delta}\right)}{\sum_k \exp\left(\frac{\epsilon v_k}{\Delta}\right)} \right) && \text{by 1} \\
&= \max_{u \sim v} \max_i \ln \left(\frac{\exp\left(\frac{\epsilon u_i}{\Delta}\right)}{\exp\left(\frac{\epsilon v_i}{\Delta}\right)} \frac{\sum_k \exp\left(\frac{\epsilon v_k}{\Delta}\right)}{\sum_k \exp\left(\frac{\epsilon u_k}{\Delta}\right)} \right) \\
&= \max_{u \sim v} \max_i \ln \left(\frac{\exp\left(\frac{\epsilon u_i}{\Delta}\right)}{\exp\left(\frac{\epsilon v_i}{\Delta}\right)} \right) + \ln \left(\frac{\sum_k \exp\left(\frac{\epsilon v_k}{\Delta}\right)}{\sum_k \exp\left(\frac{\epsilon u_k}{\Delta}\right)} \right) \\
&= \max_{u \sim v} \max_i \frac{\epsilon(u_i - v_i)}{\Delta} + \ln \left(\frac{\sum_k \exp\left(\frac{\epsilon v_k}{\Delta}\right)}{\sum_k \exp\left(\frac{\epsilon u_k}{\Delta}\right)} \right) \\
&\leq \max_{u \sim v} \max_i \frac{\epsilon(u_i - v_i)}{\Delta} + \frac{\epsilon \max_j (v_j - u_j)}{\Delta} && \text{by 3.2} \\
&\leq \epsilon \max_{u \sim v} \frac{\max_{ij} |(u_i - v_i) - (u_j - v_j)|}{\Delta} \\
&\leq \epsilon && \text{by LInfDiffDistance} \\
&= \texttt{d\_out}
\end{aligned}
$$

It has been shown that `function`(u) and `function`(v) are `d_out`-close under `output_measure` under the definitions of `function` and `privacy_map`, and the conditions on the input distance and privacy map.

References

[1] Andrés Muñoz Medina and Jennifer Gillenwater. Duff: A dataset-distance-based utility function family for the exponential mechanism. *ArXiv*, abs/2010.04235, 2020.