

fn make_report_noisy_max_gumbel

Michael Shoemate

December 19, 2024

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

Proves soundness of `make_report_noisy_max_gumbel` in `mod.rs` at commit `f5bb719` (outdated¹). `make_report_noisy_max_gumbel` 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 $U \sim Uniform(0, 1)$.

$$M(s) = \operatorname{argmin}_k \sum_i^k p_i \geq U \quad (1)$$

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.

$$p_k = \frac{e^{s_k/\tau}}{\sum_i e^{s_i/\tau}} \quad (2)$$

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:

$$\tau = \Delta/\epsilon \quad (3)$$

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.

¹See new changes with `git diff f5bb719..58dbf54 rust/src/measurements/gumbel_max/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 $K = \operatorname{argmax}_k G_k$, a random variable representing the selected index. Denote the k^{th} noisy score as $G_k \sim \text{Gumbel}(\mu = s_k/\tau)$. K can be sampled via an inverse transform, where u_k is sampled iid uniformly from $(0, 1)$:

$$M(s) = \operatorname{argmax}_k (s_k/\tau - \log(-\log(u_k))) \quad (4)$$

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

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

Lemma 1.2. $G_k = -\log(Z_k/N)$ where $Z_k \sim \text{Exp}(p_k)$ 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 $X_1 \sim \text{Exp}(\lambda_1)$, $X_2 \sim \text{Exp}(\lambda_2)$ and $Z \sim \text{Exp}(\lambda_1 + \lambda_2)$, 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 $X_1 \sim \text{Exp}(\lambda_1)$, $X_2 \sim \text{Exp}(\lambda_2)$, 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 x and x' 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:

$$\Delta_\infty = \max_{x \sim x'} d_\infty(f(x), f(x')) = \max_{x \sim x'} \max_i |f(x)_i - f(x')_i| \quad (5)$$

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, **RangeDistance**, characterizing the greatest difference in scores:

$$\Delta_{\text{Range}} = \max_{x \sim x'} d_{\text{Range}}(f(x), f(x')) = \max_{x \sim x'} \max_{i,j} |(f(x)_i - f(x')_i) - (f(x)_j - f(x')_j)| \quad (6)$$

Consider when the scoring function is not monotonic. The sensitivity is maximized when $x_i - x'_i$ and $x_j - x'_j$ 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 $x_i - x'_i$ term matches the sign of the $x_j - x'_j$ 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.

Given that both the infinity-distance and range-distance are useful, the mechanism still uses the infinity-distance, but an additional boolean is stored in the metric to indicate when the score is monotonic.

2 Hoare Triple

Precondition

- TIA (input atom type) is a type with traits **Number** and **CastInternalRational**
- Q0 (output distance type) is a type with traits **Float**, **CastInternalRational** and **DistanceConstant** from type TIA

Function

```
1 def make_report_noisy_max_gumbel(
2   input_domain: VectorDomain[AtomDomain[TIA]],
3   input_metric: RangeDistance[TIA],
```

```

4     scale: Q0,
5     optimize: Union[Literal["max"], Literal["min"]]
6 ) -> Measurement:
7     if input_domain.element_domain.nullable:
8         raise ValueError("input domain must be non-nullable")
9
10    if scale < 0:
11        raise ValueError("scale must be non-negative")
12
13    if optimize == "max":
14        sign = +1
15    elif optimize == "min":
16        sign = -1
17    else:
18        raise ValueError("must specify optimization")
19
20    scale_frac = Fraction(scale)
21
22    def function(scores: list[TIA]):
23        def map_gumbel(score):
24            return GumbelPSRN(shift=sign * Fraction(score), scale=scale_frac)
25        gumbel_scores = map(map_gumbel, scores)
26
27        def reduce_best(a, b):
28            return a if a[1].greater_than(b[1]) else b
29        return reduce(reduce_best, enumerate(gumbel_scores))[0]
30
31    def privacy_map(d_in: TIA):
32        # convert to range distance
33        # will multiply by 2 if not monotonic
34        d_in = input_metric.range_distance(d_in)
35
36        d_in = Q0.inf_cast(d_in)
37        if d_in < 0:
38            raise ValueError("input distance must be non-negative")
39
40        if d_in == 0:
41            return 0
42
43        return d_in.inf_div(scale)
44
45    return Measurement(
46        input_domain=input_domain,
47        function=function,
48        input_metric=input_metric,
49        output_metric=MaxDivergence(Q0),
50        privacy_map=privacy_map,
51    )

```

Postcondition

Theorem 2.1. For every setting of the input parameters `input_domain`, `input_metric`, `scale`, `optimize`, `TIA`, `Q0` to `make_report_noisy_max_gumbel` such that the given preconditions hold, `make_report_noisy_max_gumbel` 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 x, 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 exception, and `privacy_map(d_in) \leq d_out`, then `function(x), function(x')` 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 scale is also required to be positive.

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

Proof. For each score s_k , `function` samples a Gumbel random variable centered at $\text{sign} \cdot s_k/\tau$. 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 $U_k \sim \text{Uniform}(0, 1)$. 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. \square

Lemma 3.2. Assume x, x' in `input_domain`. Then $\ln \left(\frac{\sum_i \exp(\frac{\epsilon x'_i}{\Delta})}{\sum_i \exp(\frac{\epsilon x_i}{\Delta})} \right) \leq \frac{\epsilon \max_j (x'_j - x_j)}{\Delta}$.

Proof.

$$\begin{aligned} \ln \left(\frac{\sum_i \exp(\frac{\epsilon x'_i}{\Delta})}{\sum_i \exp(\frac{\epsilon x_i}{\Delta})} \right) &= \ln \left(\frac{\sum_i \exp(\frac{\epsilon(x'_i - x_i + x_i)}{\Delta})}{\sum_i \exp(\frac{\epsilon x_i}{\Delta})} \right) \\ &= \ln \left(\frac{\sum_i \exp(\frac{\epsilon(x'_i - x_i)}{\Delta}) \exp(\frac{\epsilon x_i}{\Delta})}{\sum_i \exp(\frac{\epsilon x_i}{\Delta})} \right) \\ &\leq \ln \left(\frac{\exp(\frac{\epsilon \max_j (x'_j - x_j)}{\Delta}) \sum_i \exp(\frac{\epsilon x_i}{\Delta})}{\sum_i \exp(\frac{\epsilon x_i}{\Delta})} \right) \\ &= \frac{\epsilon \max_j (x'_j - x_j)}{\Delta} \end{aligned}$$

\square

Assume x, x' in `input_domain` are d_{in} -close under `LInfDistance` and $\text{privacy_map}(d_{\text{in}}) \leq d_{\text{out}}$. Let the output random variables be denoted $Y \sim \text{function}(x)$ and $Y' \sim \text{function}(x')$.

$$\begin{aligned}
& \max_{x \sim x'} D_\infty(Y || Y') \\
& \leq \max_{x \sim x'} \max_i \ln \left(\frac{\Pr[\text{function}(x) = i]}{\Pr[\text{function}(x') = i]} \right) && \text{by MaxDivergence} \\
& = \max_{x \sim x'} \max_i \ln \left(\frac{\Pr[\arg\max_k (x_k/\tau - \ln(-\ln(U_k))) = i]}{\Pr[\arg\max_k (x'_k/\tau - \ln(-\ln(U_k))) = i]} \right) && \text{by 3.1, substitute function} \\
& = \max_{x \sim x'} \max_i \ln \left(\frac{\exp \frac{x_i}{\tau}}{\sum_k \exp \frac{x_k}{\tau}} \bigg/ \frac{\exp \frac{x'_i}{\tau}}{\sum_k \exp \frac{x'_k}{\tau}} \right) && \text{by 1} \\
& = \max_{x \sim x'} \max_i \ln \left(\frac{\exp \frac{x_i}{\tau} \sum_k \exp \frac{x'_k}{\tau}}{\exp \frac{x'_i}{\tau} \sum_k \exp \frac{x_k}{\tau}} \right) \\
& = \max_{x \sim x'} \max_i \ln \left(\frac{\exp \frac{x_i}{\tau}}{\exp \frac{x'_i}{\tau}} \right) + \ln \left(\frac{\sum_k \exp \frac{x'_k}{\tau}}{\sum_k \exp \frac{x_k}{\tau}} \right) \\
& = \max_{x \sim x'} \frac{\max_i (x_i - x'_i)}{\tau} + \ln \left(\frac{\sum_k \exp \frac{x'_k}{\tau}}{\sum_k \exp \frac{x_k}{\tau}} \right) \\
& \leq \max_{x \sim x'} \frac{\max_i (x_i - x'_i)}{\tau} + \frac{\max_j (x'_j - x_j)}{\tau} && \text{by 3.2} \\
& \leq \frac{\max_{x \sim x'} \max_{i,j} |(x_i - x'_i) - (x_j - x'_j)|}{\tau} \\
& = d_{in}/\tau && \text{by RangeDistance}
\end{aligned}$$

Since this expression aligns with the pseudocode given for the privacy map, it has been shown that `function(x)` and `function(x')` 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.