fn make_quantile_score_candidates

Michael Shoemate Christian Covington Ira Globus-Harrus

July 24, 2025

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

Proves soundness of `make_quantile_score_candidates` in `mod.rs` at commit f5bb719 (outdated[1]). `make_quantile_score_candidates` returns a Transformation that takes a numeric vector database and a vector of numeric quantile candidates, and returns a vector of scores, where higher scores correspond to more accurate candidates.

1 Intuition

The quantile score function scores each c in a set of candidates C.

$$s_i \quad = -|(1-\alpha) \cdot \#(x < C_i) - \alpha \cdot \#(x > C_i)| \tag{1}$$

Where $\#(x < C_i) = |\{x \in x | x < C_i\}|$ is the number of values in x less than C_i, and similarly for other variations of inequalities. The scalar score function can be equivalently stated:

$$\begin{aligned} s_i &= -|(1-\alpha) \cdot \#(x < c) - \alpha \cdot \#(x > c)| &(2)\\ &= -|(1-\alpha) \cdot \#(x < c) - \alpha \cdot (|x| - \#(x < c) - \#(x = c))| &(3)\\ &= -|\#(x < c) - \alpha \cdot (|x| - \#(x = c))| &(4) \end{aligned}$$

It has an intuitive interpretation as $-|candidate\_rank - ideal\_rank|$, where the absolute distance between the candidate and ideal rank penalizes the score. The ideal rank does not include values in the dataset equal to the candidate. This scoring function considers higher scores better, and the score is maximized at zero when the candidate rank is equivalent to the rank at the ideal α-quantile.

The scalar scorer is almost equivalent to Smith's[1], but adjusts for a source of bias when there are values in the dataset equal to the candidate. For comparison, we can equivalently write the OpenDP scorer as if there were some α-discount on dataset entries equal to the candidate.

$$\begin{array}{cl} OpenDP & -|\#(x < c) + \alpha \cdot \#(x = c) - \alpha \cdot |x|| \\ Smith & -|\#(x < c) + 1 \cdot \#(x = c) - \alpha \cdot |x|| \end{array}$$

Observing that $\#(x \leq c) = \#(x < c) + 1 \cdot \#(x = c)$.

1.1 Examples

Let $x = \{0, 1, 2, 3, 4\}$ and $\alpha = 0.5$ (median):

[1] See new changes with `git diff f5bb719..1f8625a rust/src/transformations/quantile_score_candidates/mod.rs`

$$score(x, 0, \alpha) = -|0 - .5 \cdot (5 - 1)| = -2$$
$$score(x, 1, \alpha) = -|1 - .5 \cdot (5 - 1)| = -1$$
$$score(x, 2, \alpha) = -|2 - .5 \cdot (5 - 1)| = -0$$
$$score(x, 3, \alpha) = -|3 - .5 \cdot (5 - 1)| = -1$$
$$score(x, 4, \alpha) = -|4 - .5 \cdot (5 - 1)| = -2$$

The score is maximized by the candidate at the true median.
Let $x = \{0, 1, 2, 3, 4, 5\}$ and $\alpha = 0.5$ (median):

$$score(x, 0, \alpha) = -|0 - .5 \cdot (6 - 1)| = -2.5$$
$$score(x, 1, \alpha) = -|1 - .5 \cdot (6 - 1)| = -1.5$$
$$score(x, 2, \alpha) = -|2 - .5 \cdot (6 - 1)| = -0.5$$
$$score(x, 3, \alpha) = -|3 - .5 \cdot (6 - 1)| = -0.5$$
$$score(x, 4, \alpha) = -|4 - .5 \cdot (6 - 1)| = -1.5$$
$$score(x, 5, \alpha) = -|5 - .5 \cdot (6 - 1)| = -2.5$$

The two candidates nearest the median are scored equally and highest.
Let $x = \{0, 1, 2, 3, 4\}$ and $\alpha = 0.25$ (first quartile):

$$score(x, 0, \alpha) = -|0 - .25 \cdot (5 - 1)| = -1$$
$$score(x, 1, \alpha) = -|1 - .25 \cdot (5 - 1)| = -0$$
$$score(x, 2, \alpha) = -|2 - .25 \cdot (5 - 1)| = -1$$
$$score(x, 3, \alpha) = -|3 - .25 \cdot (5 - 1)| = -2$$
$$score(x, 4, \alpha) = -|4 - .25 \cdot (5 - 1)| = -3$$

As expected, the score is maximized when $c = 1$.
Let $x = \{0, 1, 2, 3, 4, 5\}$ and $\alpha = 0.25$ (first quartile):

$$score(x, 0, \alpha) = -|0 - .25 \cdot (6 - 1)| = -1.25$$
$$score(x, 1, \alpha) = -|1 - .25 \cdot (6 - 1)| = -0.25$$
$$score(x, 2, \alpha) = -|2 - .25 \cdot (6 - 1)| = -0.75$$
$$score(x, 3, \alpha) = -|3 - .25 \cdot (6 - 1)| = -1.75$$
$$score(x, 4, \alpha) = -|4 - .25 \cdot (6 - 1)| = -2.75$$
$$score(x, 5, \alpha) = -|5 - .25 \cdot (6 - 1)| = -3.75$$

The ideal rank is 1.25. The nearest candidate, 1, has the greatest score, followed by 2, and then 0.

2 Finite Data Types

The previous equation assumes the existence of real numbers to represent α. We instead assume α is rational, such that $\alpha = \frac{\alpha_{num}}{\alpha_{den}}$. Multiply the equation through by α_{den} to get the following, which only uses integers:

$$\text{score}(x, c, \alpha_{num}, \alpha_{den}) = -|\alpha_{den} \cdot \#(x < c) - \alpha_{num} \cdot (|x| - \#(x = c))| \tag{5}$$

This adjustment also increases the sensitivity by a factor α_{den}, but does not affect the utility. We now make the scoring strictly non-negative.

- Drop the negation and instead configure the exponential mechanism to minimize the score.
- Compute the absolute difference in a function that swaps the order of arguments to keep the sign positive.

$$\text{score}(x, c, \alpha_{num}, \alpha_{den}) = \text{abs\_diff}(\alpha_{den} \cdot \#(x < c), \alpha_{num} \cdot (|x| - \#(x = c))) \tag{6}$$

To prevent a numerical overflow when computing the arguments to `abs_diff`, first choose a data type that the scores are to be represented in. If the number of records is greater than can be represented in this data type, then sample the dataset down to at most this number of records. Notice that when any given record is added or removed, the counts differ by no more than they would have without this sampling down. In the OpenDP implementation, the dataset size may be no greater than the max value of a Rust usize, because each index into the dataset maps to a distinct computer memory address.

Now allocate some of the bits of the data type for the alpha denominator, and use the remaining bits for counts of up to l, where l is the effective dataset size. From this set-up, we choose an α_{den} such that $\alpha_{den} \cdot l$ is representable. Since $\alpha_{num} \leq \alpha_{den}$, $\alpha_{num} \cdot l$ is representable. Since the dataset size fits in the choice of data type, then $|x|$ is representable. Therefore, no quantity in the following equation is not representable.

$$\text{score\_candidates}_i(x, C, \alpha_{num}, \alpha_{den}, l) = \text{abs\_diff}(\alpha_{den} \cdot \min(\#(x < C_i), l), \alpha_{num} \cdot \min(|\#(x > C_i), l)) \tag{7}$$

Should we compute counts with a 64-bit integer, we might choose α_{den} to be 10,000. This would allow for a fine fractional approximation of alpha, while still leaving enough room for datasets on the order of 10^{15} elements.

3 Hoare Triple

Precondition

- `MI` is a type with trait `UnboundedMetric`.
- `TIA` (input atom type) is a type with trait `Number`.

Function

```python
def make_quantile_score_candidates(
    input_domain: VectorDomain[AtomDomain[TIA]],
    input_metric: MI,
    candidates: list[TIA],
    alpha: f64,
) -> Transformation:
    if input_domain.element_domain.nan():
        raise ValueError("input_domain members must have non-nan elements")

    check_candidates(candidates)

    size = input_domain.size
    if size is not None:
        size = u64.exact_int_cast(input_domain.size)

    alpha_num, alpha_den, size_limit = score_candidates_constants(size, alpha)

    def function(arg: list[TIA]) -> list[u64]:
        scores = compute_score(arg, candidates, alpha_num, alpha_den, size_limit)
        return Vec.from_iter(scores)  # like calling list(s) on an iter s

```

```python
    return Transformation(
        input_domain=input_domain,
        output_domain=VectorDomain(
            element_domain=AtomDomain(T=u64), size=len(candidates)
        ),
        function=function,
        input_metric=input_metric,
        output_metric=LInfDistance.default(T=u64),
        stability_map=StabilityMap.new_fallible( # ‘\label{map}‘
            score_candidates_map(
                alpha_num,
                alpha_den,
                input_domain.size.is_some(),
            )
        ),
    )
```

Postcondition

Theorem 3.1. For every setting of the input parameters (`input_domain, input_metric, candidates, alpha, MI, TIA`) to `make_quantile_score_candidates` such that the given preconditions hold, `make_quantile_score_candidates` raises an exception (at compile time or run time) or returns a valid transformation. A valid transformation has the following properties:

1. (Appropriate output domain). For every element x in `input_domain`, `function`(x) is in `output_domain` or raises a data-independent runtime exception.

2. (Stability 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_metric`, if x, x' are `d_in`-close under `input_metric`, `stability_map`(`d_in`) does not raise an exception, and `stability_map`(`d_in`) $\leq$ `d_out`, then `function`(x), `function`(x') are `d_out`-close under `output_metric`.

Proof of Appropriate Output Domain. The raw type and domain are equivalent, save for potential nullity in the atomic type. The scalar scorer structurally cannot emit null. Therefore the output of the function is a member of the output domain. □

Proof of Stability. The constructor first performs checks to ensure that the preconditions on `score_candidates` and `score_candidates_map` are met. It checks that vectors in the input domain do not contain null values, that the candidates are strictly increasing and totally ordered, that alpha is in the range $[0, 1]$, and computes a `size_limit` for which `size_limit`$\cdot$`alpha_den` does not overflow a `u64`, and that `alpha_num` $\leq$ `alpha_den`.

Therefore the preconditions of both `score_candidates` and `score_candidates_map` are met.

Further, by the postcondition of `score_candidates`, the conditions of the postcondition of `score_candidates_map` function are met, meaning that the transformation is stable. □

References

[1] Adam Smith. Privacy-preserving statistical estimation with optimal convergence rates. In *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing*, STOC '11, page 813–822, New York, NY, USA, 2011. Association for Computing Machinery.