# fn make_noisy_top_k

Michael Shoemate

December 30, 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Proves soundness of `make_noisy_top_k` in `mod.rs` at commit f5bb719 (outdated[1]).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 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)  #
```

---

[1]See new changes with `git diff f5bb719..2cc14de rust/src/measurements/noisy_top_k/mod.rs`

```
25
26          if d_in.is_sign_negative():   #
27              raise "sensitivity must be non-negative"
28
29          if d_in.is_zero():   #
30              return 0.0
31
32          if scale.is_zero():   #
33              return f64.INFINITY
34
35          #
36          return MO.privacy_map(d_in, scale).inf_mul(f64.inf_cast(k))
37
38      return Measurement.new(
39          input_domain=input_domain,
40          input_metric=input_metric,
41          output_measure=privacy_measure,
42          function=lambda x: noisy_top_k(x, scale, k, negate, MO.REPLACEMENT),
43          privacy_map=privacy_map,
44      )
```

## Postcondition

**Theorem 1.1.** 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
    $(\mathtt{d\_in}, \mathtt{d\_out})$, 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 `noisy_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 when 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 postcondi-
tion. 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 `noisy_top_k` are satisfied. Additionally by the
checks on lines 26, 29 and 32, the preconditions for `TopKMeasure`privacy_map` are satisfied.

   By the postcondition of `TopKMeasure` and adaptive composition, the `d_out`on line 35 satisfies the post-
condition. □