

# fn make\_noisy\_top\_k

Michael Shoemate

February 5, 2026

Proves soundness of `make_noisy_top_k` in `mod.rs` at commit `f5bb719` (outdated<sup>1</sup>).  
`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

```
1 def make_noisy_top_k(  
2     input_domain: VectorDomain[AtomDomain[TIA]],  
3     input_metric: LInfDistance[TIA],  
4     privacy_measure: MO,  
5     k: usize,  
6     scale: f64,  
7     negate: bool,  
8 ) -> Measurement:  
9     if input_domain.element_domain.nan(): #  
10         raise "input domain elements must be non-nan"  
11  
12     if input_domain.size is not None:  
13         if k > input_domain.size:  
14             raise "k must not exceed the number of candidates"  
15  
16     if not scale.is_finite() or scale.is_sign_negative(): #  
17         raise "scale must be finite and non-negative"  
18  
19     monotonic = input_metric.monotonic  
20  
21     def privacy_map(d_in: TIA): #  
22         # convert to range distance  
23         d_in = d_in if monotonic else d_in.inf_add(d_in)  
24         d_in = f64.inf_cast(d_in) #
```

---

<sup>1</sup>See new changes with `git diff f5bb719..f4b102f rust/src/measurements/noisy_top_k/mod.rs`

```

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

*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 `noisy_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_out` on line 35 satisfies the postcondition.  $\square$