ranx: A Blazing-Fast Python Library for Ranking Evaluation and Comparison

Elias Bassani[1,2]

[1]Consorzio per il Trasferimento Tecnologico - C2T [2]University of Milano-Bicocca, Milan, Italy

What is ranx?

- `ranx` is a Python evaluation library for Information Retrieval.
- `ranx` embraces a *Plug & Play* philosophy, providing a user-friendly interface to the most common ranking evaluation metrics.
- `ranx` is built on top of `Numba` [3], a *just-in-time* [1] compiler for Python code, that allows high-speed vector operations and automatic parallelization.

Main Features

- Convenient way of managing qrels, runs, and evaluation results.
- Qrels and runs can be imported (exported) from (to) Python dictionaries, JSON files, TREC-Style files, and Pandas DataFrames.
- Automatic sorting and data checking, so you can focus on what matter.
- Compute multiple metrics with a single line of code.
- Compare different runs and perform statistical testing with one function call.
- Visualize well-formatted comparison tables directly in your terminal or in a Jupyter Notebook.
- Export LaTeX tables ready for your scientific publications.
- The efficiency brought by `Numba` [3] makes the adoption of `ranx` convenient even for industrial applications.

Available Metrics

- Hits
- Hit Rate
- Precision
- Recall
- F1
- r-Precision
- Mean Reciprocal Rank (MRR)
- Mean Average Precision (MAP)
- Normalized Discounted Cumulative Gain (NDCG)

Each metric supports user-defined cutoffs (*metric "at k"*).

All the available metrics were tested against `trec_eval` [4] for correctness.

Export LaTeX Tables

The LaTeX code of the following results table and its caption were generated with `ranx` by simply calling `report.to_latex()`.

Table 1. Overall effectiveness of the models. Best results are highlighted in boldface. Superscripts denote statistically significant differences in Fisher's Randomization Test with $p \leq 0.01$.

#	Model	MAP@100	MRR@100	NDCG@10
a	model_1	0.3202^{b}	0.3207^{b}	0.3684^{bc}
b	model_2	0.2332	0.2339	0.239
c	model_3	0.3082^{b}	0.3089^{b}	0.3295^{b}
d	model_4	0.3664^{abc}	0.3668^{abc}	0.4078^{abc}
e	model_5	$\mathbf{0.4053}^{abcd}$	$\mathbf{0.4061}^{abcd}$	$\mathbf{0.4512}^{abcd}$

Quick Overview

```python
# QRELS AND RUN ------------------------------------------------------
from ranx import Qrels, Run

qrels_dict = { "q_1": { "d_12": 5, "d_25": 3 },
               "q_2": { "d_11": 6, "d_22": 1 } }

run_dict  =  { "q_1": { "d_12": 0.9, "d_23": 0.8, "d_25": 0.7,
                        "d_36": 0.6, "d_32": 0.5, "d_35": 0.4  },
               "q_2": { "d_12": 0.9, "d_11": 0.8, "d_25": 0.7,
                        "d_36": 0.6, "d_22": 0.5, "d_35": 0.4  } }

qrels = Qrels(qrels_dict)
run = Run(run_dict)
```

```python
# EVALUATE -----------------------------------------------------------
from ranx import evaluate

# Compute score for a single metric
evaluate(qrels, run, "ndcg@5")
>>> 0.7861

# Compute scores for multiple metrics at once
evaluate(qrels, run, ["map@5", "mrr"])
>>> {"map@5": 0.6416, "mrr": 0.75}
```

```python
# COMPARE ------------------------------------------------------------
from ranx import compare

# Compare different runs and perform statistical tests
report = compare(
    qrels=qrels,
    runs=[run_1, run_2, run_3, run_4, run_5],
    metrics=["map@100", "mrr@100", "ndcg@10"],
    max_p=0.01  # P-value threshold
)
```

```python
# REPORT -------------------------------------------------------------
print(report)

#    Model    MAP@100     MRR@100     NDCG@10
---  -------  --------    --------    ---------
a    model_1  0.320[b]     0.320[b]    0.368[b c]
b    model_2  0.233        0.234       0.239
c    model_3  0.308[b]     0.309[b]    0.330[b]
d    model_4  0.366[a b c] 0.367[a b c] 0.408[a b c]
e    model_5  0.405[a b c d] 0.406[a b c d] 0.451[a b c d]
```

Efficiency

Here is reported an efficiency comparison between `ranx` (using different number of threads) and `pytrec_eval` (pytrec) [2], a Python interface to `trec_eval` [4]. The comparison was conducted with synthetic data. Queries have 1-to-10 relevant documents. Retrieved lists contain 100 documents. NDCG, MAP, and MRR were computed on the entire lists. Results are reported in milliseconds. Speed-ups were computed w.r.t. `pytrec_eval`.

metric	queries	pytrec	ranx t=1		ranx t=2		ranx t=4		ranx t=8	
NDCG	1 000	28	4	7.0×	3	9.3×	2	14.0×	2	14.0×
	10 000	291	35	8.3×	24	12.1×	18	16.2×	15	19.4×
	100 000	2 991	347	8.6×	230	13.0×	178	16.8×	152	19.7×
MAP	1 000	27	2	13.5×	2	13.5×	1	27.0×	1	27.0×
	10 000	286	21	13.6×	13	22.0×	9	31.8×	7	40.9×
	100 000	2 950	210	14.0×	126	23.4×	84	35.1×	69	42.8×
MRR	1 000	28	1	28.0×	1	28.0×	1	28.0×	1	28.0×
	10 000	283	7	40.4×	6	47.2×	4	70.8×	4	70.8×
	100 000	2 935	74	39.7×	57	51.5×	44	66.7×	38	77.2×

Online Resources

- Lean more about `ranx` at https://amenra.github.io/ranx/ (or scan the QR Code below).
- Would you like to see other features implemented? Feel free to open a feature request on our repository: https://github.com/AmenRa/ranx.

References

[1] John Aycock. A brief history of just-in-time. *ACM Comput. Surv.*, 35(2):97–113, 2003.

[2] Christophe Van Gysel and Maarten de Rijke. Pytrec_eval: An extremely fast python interface to trec_eval. In *SIGIR*, pages 873–876. ACM, 2018.

[3] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: a llvm-based python JIT compiler. In *LLVM@SC*, pages 7:1–7:6. ACM, 2015.

[4] E Voorhees and D Harman. Experiment and evaluation in information retrieval, 2005.