Towards an Information Retrieval Evaluation Library

Discussion Paper

Elias Bassani[1,2]

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

Abstract
This manuscript discusses our ongoing work on `ranx`, a Python evaluation library for Information Retrieval. First, we introduce our work, summarize the already available functionalities, show the user-friendly nature of our tool through code snippets, and briefly discuss the technologies we relied on for the implementation and their advantages. Then, we present the upcoming features, such as several Metasearch algorithms, and introduce the long-term goals of our project.

Keywords
Information Retrieval, Evaluation, Comparison, Metasearch, Fusion

1. Introduction

Nowadays, the development of novel Information Retrieval models usually undergoes an offline evaluation step where the results of different models are compared on the same set of queries to determine whether improvements over the state-of-the-art have been reached [1, 2]. To evaluate the retrieval effectiveness of the compared models, researchers rely on multiple metrics, such as *Reciprocal Rank*, *Average Precision*, and *Normalized Discounted Cumulative Gain* [3].

Over the years, multiple software libraries have been proposed to perform this assessment [4, 5, 6, 7, 8, 9, 10, 11]. However, in our opinion, those libraries still lack a stress-free user-friendly interface. Therefore, we recently proposed `ranx`[1][12], a Python library built following a user-centered design [13] to provide an easy-to-use tool for Information Retrieval researchers. `ranx` offers several ranking evaluation metrics and allows users to compare the results of different systems in just a few lines of code, while providing top-notch efficiency thanks to `Numba` [14], a *just-in-time* compiler [15] for Python and NumPy [16, 17, 18] code.

In the following sections, we first summarize the functionalities currently offered by `ranx`. Then, we present the upcoming features. Finally, we introduce the long-term goal of our project.

IIR2022: 12th Italian Information Retrieval Workshop, June 29 - June 30th, 2022, Milan, Italy
e.bassani3@campus.unimib.it (E. Bassani)
0000-0001-7922-2578 (E. Bassani)

[1]https://github.com/AmenRa/ranx

2. Overview

In this section, we present the main functionalities `ranx` provides, show its user-friendly nature through some code snippets, and discuss its implementation and the advantages brought by the employed technologies. More details and examples are available in the official repository.

2.1. Qrels and Run

First, `ranx` provides a convenient way of managing the data needed for evaluating and comparing different retrieval models: the *query relevance judgments* (*qrels*) and ranked lists of documents retrieved for those queries by the systems (*runs*). `ranx` implements two custom classes for these kinds of data: `Qrels` and `Run`. In particular, data can be loaded from Python dictionaries and Pandas DataFrames [19] or read from TREC-style files and JSON files. Moreover, `ranx` integrates seamlessly with *ir-datasets* [20], allowing the users to load qrels for *several* Information Retrieval datasets, such as those from TREC's challenges[2], BEIR [21], and MS MARCO [22]. Figure 1 shows the standard way of creating `Qrels` and `Run` instances. `ranx` takes care of sorting the result lists so that the user does not have to think about it. To learn more about `Qrels` and `Run`, we invite the reader to follow our online Jupyter Notebook[3].

```python
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)

qrels = Qrels.from_ir_datasets("msmarco-document/dev")
```

Figure 1: Qrels and Run

2.2. Metrics, Evaluation, and Comparison

`ranx` provides the most commonly used ranking evaluation metrics[4] such as *Reciprocal Rank*, *Average Precision*, and *Normalized Discounted Cumulative Gain* [3]. These metrics can be used to evaluate a *run* in a single line of code, as depicted in Figure 2. As the figure shows, `ranx` allows the user to provide one or multiple metrics and define cut-offs using a convenient syntax. Additional information can be found online[5].

`ranx` also offers functionalities to compare *runs* and perform statistical tests. As shown in Figure 3, by providing the *query relevance judgments* and a list of *runs* and defining the desired

[2]https://trec.nist.gov
[3]https://colab.research.google.com/github/AmenRa/ranx/blob/master/notebooks/2_qrels_and_run.ipynb
[4]A complete list of the implemented metrics can be found here: https://github.com/AmenRa/ranx#metrics
[5]https://colab.research.google.com/github/AmenRa/ranx/blob/master/notebooks/3_evaluation.ipynb

metrics, the `compare` function performs a comparison of the *runs*. It returns a `Report` instance, which stores the information produced by the `compare` function and can be printed as in Figure 3 or exported as a LaTeX table, ready for a scientific publication. The code underlying Table 1 was generated by `ranx`. To learn more about comparing different runs, we invite the reader to follow our online Jupyter Notebook[6].

```python
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}
```

Figure 2: Evaluation

```python
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
)

print(report)

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

Figure 3: Comparison and Report

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	$\mathbf{0.4053}^{abcd}$	$\mathbf{0.4061}^{abcd}$	$\mathbf{0.4512}^{abcd}$

[6]https://colab.research.google.com/github/AmenRa/ranx/blob/master/notebooks/4_comparison_and_report.ipynb

2.3. Backend

In addition to its user-friendly interface, `ranx` is also very efficient due to its `Numba`-based implementation. `Numba`[14] is a *just-in-time*[15] compiler for Python and NumPy[16, 17, 18] that translates and compiles for-loop-based code to high-speed vector operations and allows for automatic parallelization, which is very handy on modern multi-core CPUs. Almost every operation performed by `ranx` relies on `Numba`-compiled code. The internal data structures used by `Qrels` and `Run` and all the evaluation metrics provided by `ranx` are built on top of `Numba`. Our implementation allows for conducting evaluations and comparisons much faster than other popular Python evaluation libraries for Information Retrieval. Table 2 reports the execution time of different metrics in `ranx` and `pytrec_eval`, a Python wrapper for `trec_eval`, the standard Information Retrieval evaluation library.

Table 2
Efficiency comparison between `ranx` (using different number of threads) and `pytrec_eval` (pytrec), a Python interface to `trec_eval`. 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×

3. Upcoming Features

We are currently implementing several Metasearch [23] algorithms, such as `comb_min` [24], `comb_max` [24], `comb_med` [24], `comb_anz` [24], `comb_mnz` [24], `comb_sum` [24], `comb_gmnz` [25], `RRF` [26], `MAPFuse` [27], `ISR` [28], `Log_ISR` [28], `LogN_ISR` [28], and many more. Our goal is to offer a Python implementation for all those methods with a standardized interface. Moreover, we want to provide a working and easy-to-use implementation of those models that could serve as baselines for researchers working on Metasearch algorithms. Moreover, we argue young researchers in the Deep Learning-based Information Retrieval era have little knowledge regarding Metasearch methods as they *often* rely on the weighted sum to fuse lexical matching scores, such as those computed by BM25 [29], and semantic matching scores computed by Transformer-based [30] rankers [31]. We hope that our work can stimulate researchers to explore different fusion approaches. As many Metasearch algorithms require to be tuned, we are also working on an `auto-tune` functionality that takes care of trying different hyper-parameters configurations and finding the best performing one with no user effort.

4. Conclusion and Long-term Goals

To conclude our discussion, we introduce the long-term goals of our library. Besides adding more metrics and other Metasearch methods, we plan to build a companion repository for storing runs of state-of-the-art models accompanied by rich metadata for searching and indexing. By integrating this online repository with `ranx`, we aim to allow researchers to download pre-computed runs and compare the results of their models with those of state-of-the-art approaches in just a few seconds. We think such functionality could help accelerate research in Information Retrieval, allowing researchers to rapidly find appropriate baselines and avoiding time-consuming and error-prone tasks entirely, such as re-implementing or re-training complex retrieval models from scratch. Moreover, sharing runs of state-of-the-art models could promote virtuous behaviors and transparency and reduce electricity consumption and pollution.

References

[1] D. Harman, Information Retrieval Evaluation, Synthesis Lectures on Information Concepts, Retrieval, and Services, Morgan & Claypool Publishers, 2011.

[2] M. Sanderson, Test collection based evaluation of information retrieval systems, Found. Trends Inf. Retr. 4 (2010) 247–375.

[3] K. Järvelin, J. Kekäläinen, Cumulated gain-based evaluation of IR techniques, ACM Trans. Inf. Syst. 20 (2002) 422–446.

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

[5] C. Macdonald, N. Tonellotto, Declarative experimentation in information retrieval using pyterrier, in: ICTIR, ACM, 2020, pp. 161–168.

[6] C. Macdonald, N. Tonellotto, S. MacAvaney, I. Ounis, Pyterrier: Declarative experimentation in python from BM25 to dense retrieval, in: CIKM, ACM, 2021, pp. 4526–4533.

[7] C. V. Gysel, M. de Rijke, Pytrec_eval: An extremely fast python interface to trec_eval, in: SIGIR, ACM, 2018, pp. 873–876.

[8] J. R. M. Palotti, H. Scells, G. Zuccon, Trectools: an open-source python library for information retrieval practitioners involved in trec-like campaigns, in: SIGIR, ACM, 2019, pp. 1325–1328.

[9] T. Breuer, N. Ferro, M. Maistro, P. Schaer, repro_eval: A python interface to reproducibility measures of system-oriented IR experiments, in: ECIR (2), volume 12657 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 481–486.

[10] C. Lucchese, C. I. Muntean, F. M. Nardini, R. Perego, S. Trani, Rankeval: Evaluation and investigation of ranking models, SoftwareX 12 (2020) 100614.

[11] C. Lucchese, C. I. Muntean, F. M. Nardini, R. Perego, S. Trani, Rankeval: An evaluation and analysis framework for learning-to-rank solutions, in: SIGIR, ACM, 2017, pp. 1281–1284.

[12] E. Bassani, ranx: A blazing-fast python library for ranking evaluation and comparison, in: M. Hagen, S. Verberne, C. Macdonald, C. Seifert, K. Balog, K. Nørvåg, V. Setty (Eds.), Advances in Information Retrieval - 44th European Conference on IR Research, ECIR 2022, Stavanger, Norway, April 10-14, 2022, Proceedings, Part II, volume 13186 of *Lec-*

ture Notes in Computer Science, Springer, 2022, pp. 259–264. URL: https://doi.org/10.1007/978-3-030-99739-7_30. doi:10.1007/978-3-030-99739-7_30.
[13] C. Abras, D. Maloney-Krichmar, J. Preece, et al., User-centered design, Bainbridge, W. Encyclopedia of Human-Computer Interaction. Thousand Oaks: Sage Publications 37 (2004) 445–456.
[14] S. K. Lam, A. Pitrou, S. Seibert, Numba: a llvm-based python JIT compiler, in: LLVM@SC, ACM, 2015, pp. 7:1–7:6.
[15] J. Aycock, A brief history of just-in-time, ACM Comput. Surv. 35 (2003) 97–113.
[16] T. E. Oliphant, A guide to NumPy, volume 1, Trelgol Publishing USA, 2006.
[17] S. van der Walt, S. C. Colbert, G. Varoquaux, The numpy array: A structure for efficient numerical computation, Comput. Sci. Eng. 13 (2011) 22–30.
[18] C. R. Harris, K. J. Millman, S. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, T. E. Oliphant, Array programming with numpy, Nat. 585 (2020) 357–362.
[19] W. McKinney, et al., pandas: a foundational python library for data analysis and statistics, Python for high performance and scientific computing 14 (2011) 1–9.
[20] S. MacAvaney, A. Yates, S. Feldman, D. Downey, A. Cohan, N. Goharian, Simplified data wrangling with ir_datasets, in: SIGIR, ACM, 2021, pp. 2429–2436.
[21] N. Thakur, N. Reimers, A. Rücklé, A. Srivastava, I. Gurevych, BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models, in: J. Vanschoren, S. Yeung (Eds.), Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual, 2021. URL: https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/65b9eea6e1cc6bb9f0cd2a47751a186f-Abstract-round2.html.
[22] T. Nguyen, M. Rosenberg, X. Song, J. Gao, S. Tiwary, R. Majumder, L. Deng, MS MARCO: A human generated machine reading comprehension dataset, in: T. R. Besold, A. Bordes, A. S. d'Avila Garcez, G. Wayne (Eds.), Proceedings of the Workshop on Cognitive Computation: Integrating neural and symbolic approaches 2016 co-located with the 30th Annual Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain, December 9, 2016, volume 1773 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2016. URL: http://ceur-ws.org/Vol-1773/CoCoNIPS_2016_paper9.pdf.
[23] J. A. Aslam, M. H. Montague, Models for metasearch, in: W. B. Croft, D. J. Harper, D. H. Kraft, J. Zobel (Eds.), SIGIR 2001: Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, September 9-13, 2001, New Orleans, Louisiana, USA, ACM, 2001, pp. 275–284. URL: https://doi.org/10.1145/383952.384007. doi:10.1145/383952.384007.
[24] E. A. Fox, J. A. Shaw, Combination of multiple searches, in: TREC, volume 500-215 of *NIST Special Publication*, National Institute of Standards and Technology (NIST), 1993, pp. 243–252.
[25] J. H. Lee, Analyses of multiple evidence combination, in: SIGIR, ACM, 1997, pp. 267–276.
[26] G. V. Cormack, C. L. A. Clarke, S. Büttcher, Reciprocal rank fusion outperforms condorcet and individual rank learning methods, in: SIGIR, ACM, 2009, pp. 758–759.

[27] D. Lillis, L. Zhang, F. Toolan, R. W. Collier, D. Leonard, J. Dunnion, Estimating probabilities for effective data fusion, in: F. Crestani, S. Marchand-Maillet, H. Chen, E. N. Efthimiadis, J. Savoy (Eds.), Proceeding of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2010, Geneva, Switzerland, July 19-23, 2010, ACM, 2010, pp. 347–354. URL: https://doi.org/10.1145/1835449.1835508. doi:10.1145/1835449.1835508.

[28] A. Mourão, F. Martins, J. Magalhães, Multimodal medical information retrieval with unsupervised rank fusion, Comput. Medical Imaging Graph. 39 (2015) 35–45. URL: https://doi.org/10.1016/j.compmedimag.2014.05.006. doi:10.1016/j.compmedimag.2014.05.006.

[29] S. E. Robertson, S. Walker, Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval, in: Proceedings of the 17th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval. Dublin, Ireland, 3-6 July 1994 (Special Issue of the SIGIR Forum), ACM/Springer, 1994. doi:10.1007/978-1-4471-2099-5_24.

[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, in: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, 2017.

[31] J. Lin, R. Nogueira, A. Yates, Pretrained Transformers for Text Ranking: BERT and Beyond, Synthesis Lectures on Human Language Technologies, Morgan & Claypool Publishers, 2021. URL: https://doi.org/10.2200/S01123ED1V01Y202108HLT053. doi:10.2200/S01123ED1V01Y202108HLT053.