ranx.fuse: A Python Library for Metasearch

Elias Bassani
Consorzio per il Trasferimento Tecnologico - C2T
University of Milano-Bicocca
Milan, Italy
e.bassani3@campus.unimib.it

Luca Romelli
University of Milano-Bicocca
Milan, Italy
l.romelli@campus.unimib.it

ABSTRACT

This paper presents `ranx.fuse`, a Python library for Metasearch. Built following a user-centered design, it provides easy-to-use tools for combining the results of multiple search engines. `ranx.fuse` comprises 25 Metasearch algorithms implemented with `Numba`, a *just-in-time* compiler for Python code, for efficient vector operations and automatic parallelization. Moreover, in conjunction with the Metasearch algorithms, our library implements six normalization strategies that transform the search engines' result lists to make them comparable, a mandatory step for Metasearch. Finally, as many Metasearch algorithms require a training or optimization step, `ranx.fuse` offers a convenient functionality for their optimization that evaluates pre-defined hyper-parameters configurations via grid search. By relying on the provided functions, the user can optimally combine the results of multiple search engines in very few lines of code. `ranx.fuse` can also serve as a user-friendly tool for fusing the rankings computed by a first-stage retriever and a re-ranker, as a library providing several baselines for Metasearch, and as a playground to test novel normalization strategies.

CCS CONCEPTS

• **Information systems → Combination, fusion and federated search**.

KEYWORDS

Information Retrieval, Metasearch, Fusion, Tool

ACM Reference Format:
Elias Bassani and Luca Romelli. 2022. `ranx.fuse`: A Python Library for Metasearch. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM '22), October 17–21, 2022, Atlanta, GA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3511808.3557207

1 INTRODUCTION

Metasearch [1], sometimes called data-fusion [9, 14–16, 26, 28], is the problem of combining the results returned by multiple search engines in response to a given query in a way that optimizes the performance of their combination. Previous works [1, 3, 5, 7, 13–16, 21, 22, 26, 28] have shown this combination to consistently improve the retrieval effectiveness of the combined systems. As discussed by Aslam et al. [1], Metasearch algorithms can be applied *externally*, when the combined search engines are completely independent from each other, or *internally*, when a single search engine comprises multiple retrieval models. In the latter case, Metasearch techniques allow decomposing a large and monolithic search engine into smaller and more specialized modules, whose results are fused after the retrieval phase. Metasearch techniques are usually classified as score-based [7, 13, 28] or rank-based [1, 3, 5, 14–16, 21, 22, 26] methods depending on whether they need relevance scores or just the ranking positions of the retrieved documents. They can also be divided into supervised and unsupervised methods, whether they require training data or not.

Despite Metasearch has long been studied in Information Retrieval and simple combination schemes, such as the weighted sum of multiple relevance scores, are often used for fusing the rankings produced by modern multi-stage retrieval systems [8, 17, 27], most of the proposed algorithms do not have a publicly available implementation. Moreover, there is a lack of a dedicated library providing several of these algorithms for research and optimization in a single package. As far as we know, `TrecTools`[1] [23] and `Polyfuse`[2] [11] are the only available tools exposing some Metasearch algorithms. Unfortunately, they both provide few fusion methods and lack advanced functionalities for fusion optimization. `TrecTools` is meant for meta-analysis purposes of TREC-like campaigns' submissions rather than Metasearch research or optimization, which explains the lack of more algorithms and advanced features. `Polyfuse` was a companion tool for the SIGIR 2018's tutorial on fusion [11]. It comes as a simple command-line tool with limited functionalities rather than a full-fledged library for Metasearch, limiting its adoption as a day-to-day tool for Information Retrieval researchers and practitioners. The unavailability of a dedicated Metasearch library providing working implementations of several algorithms and advanced functionalities for fusion optimization motivates the work we have undergone to build the Python library for Metasearch we present in this paper: `ranx.fuse`.

`ranx.fuse` provides a user-friendly interface to 25 Metasearch algorithms, implemented leveraging `Numba` [12], a *just-in-time* [2] compiler for Python code, for efficient vector operations and automatic parallelization. Our library offers both score-based and rank-based fusion approaches, all of which can be accessed through a convenient interface, the method `fuse`, as we will discuss later in this paper. Additionally, `ranx.fuse` allows the user to experiment with six normalization strategies to transform the results of different search engines to make them comparable, which is mandatory for the correct application of many Metasearch algorithms [6, 20].

CIKM '22, October 17–21, 2022, Atlanta, GA, USA

ACM ISBN 978-1-4503-9236-5/22/10…$15.00
https://doi.org/10.1145/3511808.3557207

[1] https://github.com/joaopalotti/trectools
[2] https://github.com/rmit-ir/polyfuse

Finally, since several Metasearch algorithms require a training or optimization phase, `ranx.fuse` provides a convenient feature for their optimization that automatically evaluates pre-defined hyper-parameters configurations via grid search. As `ranx.fuse` leverages the `Numba`-based data structures and metrics provided by `ranx` [4], a recently introduced Python library for ranking evaluation and comparison, we decided to integrate it into `ranx`'s code base and share both of them as a single package. By doing so, both libraries can benefit when the other receives low-level optimizations.

In the next section, we introduce the implemented functionalities and provide usage examples that show the user-centered design of `ranx.fuse` and its usability. Then, we compare `ranx.fuse` with already available tools for Metasearch, *i.e.*, `TrecTools` and `Polyfuse`, highlighting their main differences from both a functional and a usability perspective. Finally, we discuss some use cases for our library to show the multi-faceted utility of a publicly available tool providing Metasearch functionalities.

2 SYSTEM OVERVIEW

In this section, we present the main functionalities provided by `ranx.fuse`: the normalization strategies, the `fuse` method, and the `optimize_fusion` method. More details and examples are available in the official repository[3], documentation[4], and Jupyter Notebook[5].

As already mentioned, `ranx.fuse` leverages the data structures provided by `ranx` [4], a recently introduced Python library for ranking evaluation and comparison. For completeness, before introducing the main `ranx.fuse`'s Metasearch functionalities, we show how to load the data they require using `ranx`.

2.1 Loading Qrels and Runs

The data needed for optimizing and using the Metasearch functionalities provided by `ranx.fuse` are of two kinds: the query relevance judgments (qrels), which store the ground truth for conducting the optimization phase, and the results (runs) returned by the search engines that the user intend to combine. For these two kinds of data, `ranx` provides two dedicated data structures implemented with `Numba`: `Qrels` and `Run`. The preferred way for creating a `Qrels` and `Run` instances is converting Python dictionaries as reported in Listing 1. `Qrels` and `Runs` can also be read from TREC-style and JSON files, converted from Pandas DataFrames [19], or (for Qrels only) loaded from `ir-datasets` [18]. For more information and examples we invite the reader to refer to this Jupyter Notebook[6].

2.2 Normalization

The first step when fusing the results of multiple retrieval systems with score-based fusion methods is to normalize them to make them comparable. This operation is required as different retrieval models estimate relevance scores on different scales and ranges and distribute them differently [20]. For example, the classic probabilistic retrieval model BM25 [25] outputs unbounded positive

```python
from ranx import Qrels

qrels_dict = {
    "q_1": {"d_1": 1, "d_2": 2},
    "q_2": {"d_3": 2, "d_2": 1, "d_5": 3},
}

qrels = Qrels(qrels_dict)
```

Listing 1: Instantiating `Qrels`. *Same procedure for* `Run`.

relevance scores, while modern Deep Learning-based retrieval systems, relying on the dot-product or the cosine similarity to compute relevance scores, often output unbounded scores or scores in the interval $[-1, 1]$. To this extent, `ranx.fuse` offers six normalization strategies: Min-Max Norm, Max Norm, Sum Norm [20], ZMUV Norm [20], Rank Norm [24], and Borda Norm [24]. Note that Rank Norm and Borda Norm transform the original relevance-based scores assigned to the documents into scores based on their positioning in the list of ranked results. Therefore, when using these normalization strategies, score-based fusion methods act as rank-based methods. The reader can refer to the package documentation for further details on the provided normalization strategies [7].

2.3 Fusion

Table 1 lists the fusion algorithms provided by `ranx.fuse` and reports whether they require a training phase or have hyper-parameters that need to be optimized. We classify those algorithms into four categories: score-based methods, rank-based methods, probabilistic methods, and voting-based methods. Score-based methods [7, 13, 28] combine the relevance scores given to the documents retrieved by multiple search engines to derive the scores used to decide the final arrangement for the retrieved documents. Rank-based methods [3, 5, 22] rely only on the positioning of the documents retrieved by the considered search engines to derive the final ranking. Rank-based methods are particularly useful when the relevance scores given to the documents retrieved by the different search engines are unavailable (this is the case of Web search aggregators such as Kayak[8] and Skyscanner[9]). A special case of the rank-based methods are the probabilistic methods [1, 14–16, 26], which derive a probability distribution of the relevance over the ranking positions, *i.e.*, for each search engine, they assign to each ranking position the probability of finding a relevant document in that specific position. Probabilistic methods require a training phase to estimate this probability distribution. The voting-based methods [1, 21] adapt voting procedures, such as Borda Count and the Condorcet election method, to Metasearch, combining the preferences of multiple "experts", *i.e.*, the search engines. The voting-based methods can be considered as a special case of rank-based methods as they only rely on the ranking positions of the documents.

To simplify the use of the provided algorithms, `ranx.fuse` exposes a single interface to access them: the method `fuse`, shown

[3] https://github.com/AmenRa/ranx
[4] https://amenra.github.io/ranx
[5] https://colab.research.google.com/github/AmenRa/ranx/blob/master/notebooks/5_fusion.ipynb
[6] https://colab.research.google.com/github/AmenRa/ranx/blob/master/notebooks/2_qrels_and_run.ipynb
[7] https://amenra.github.io/ranx/normalization
[8] https://www.kayak.it
[9] https://www.skyscanner.it

Table 1: Provided fusion algorithms. Supervised means the algorithm requires a training phase. Params column indicates whether the algorithm has hyper-parameters that need to be optimized. TT column indicates whether the algorithm is provided by `TrecTools`. PF column indicates whether the algorithm is provided by `Polyfuse`.

Score-based Methods				
Name	**Supervised**	**Params**	**TT**	**PF**
CombANZ [7]	✗	✗	✓	✓
CombMAX [7]	✗	✗	✓	✓
CombMED [7]	✗	✗	✓	✓
CombMIN [7]	✗	✗	✓	✓
CombMNZ [7]	✗	✗	✓	✓
CombSUM [7]	✗	✗	✓	✓
CombGMNZ [13]	✗	✓	✗	✗
Mixed [28]	✗	✓	✗	✗
WMNZ [28]	✗	✓	✗	✗
Weighted Sum	✗	✓	✗	✗
Rank-based Methods				
Name	**Supervised**	**Params**	**TT**	**PF**
ISR [22]	✗	✗	✗	✓
Log_ISR [22]	✗	✗	✗	✓
LogN_ISR [22]	✗	✓	✗	✗
RBC [3]	✗	✓	✓	✓
RRF [5]	✗	✓	✓	✓
Probabilistic Methods				
Name	**Supervised**	**Params**	**TT**	**PF**
BayesFuse [1]	✓	✗	✗	✗
MAPFuse [16]	✓	✗	✗	✗
PosFuse [16]	✓	✗	✗	✗
ProbFuse [14]	✓	✓	✗	✗
SegFuse [26]	✓	✗	✗	✗
SlideFuse [15]	✓	✓	✗	✗
Voting-based Methods				
Name	**Supervised**	**Params**	**TT**	**PF**
BordaFuse [1]	✗	✗	✓	✓
Weighted BordaFuse [1]	✗	✓	✗	✗
Condorcet [21]	✗	✗	✗	✗
Weighted Condorcet [21]	✗	✓	✗	✗

in Listing 2. This function takes as input the runs to be combined (the ranked lists of documents retrieved by the search systems for multiple queries), the name of the normalization strategy to apply before fusion, and the name of the fusion method. Optionally, as discussed in the next section, the `fuse` function can take in input some parameters required by the chosen fusion method.

2.4 Fusion optimization

As reported in Table 1, many fusion algorithms require a training or optimization step. To this extent, `ranx.fuse` implements

```python
from ranx import fuse

combined_run = fuse(
    runs=[run_1, run_2],  # List of Runs to fuse
    norm="min-max",       # Normalization strategy
    method="sum",         # Alias for CombSUM
)
```

Listing 2: Usage example of the `fuse` method.

the functionalities needed to optimize those algorithms. Instead of exposing several training and optimization functions, `ranx.fuse` conveniently provides the users with a single easy-to-use interface, the `optimize_fusion` method, to optimize those algorithms with ease. Under the hood, it routes the input parameters to the correct functions and performs other operations if needed. In the case of the algorithms requiring hyper-parameters optimization, `ranx.fuse` comes with pre-defined hyper-parameters search spaces, which can be altered by the user. During optimization, `ranx.fuse` automatically generates and evaluates several hyper-parameters configurations via grid search, relieving the users of the burden of implementing such procedure for each of the provided algorithms. Once the optimization is complete, `optimize_fusion` outputs the trained/optimized parameters for correctly fusing the ranked lists produced by the search engines the user want to combine.

Listing 3 shows a usage example of the `optimize_fusion` method. In this case, the goal is to find the Weighted Sum algorithm's weights configuration that maximize the Normalized Discounted Cumulative Gain [10] at depth 100 (NDCG@100). To do so, the user provides some training data: the query relevance judgments for some training queries (qrels) and the results lists (runs) produced for those queries by the retrieval systems she aims to combine. After normalization, `optimize_fusion` will automatically evaluate several weights configuration and output the best one. At this point, the user provides the runs for the test queries and the best weights configuration previously found to the `fuse` method to compute the final combined run.

In addition to finding the best hyper-parameters configuration for a given Metasearch algorithm, `optimize_fusion` can optionally return (by setting the parameter `return_optimization_report` to `True`) a report of the evaluated configurations for inspection. Table 2 shows the report for a Weighted Sum optimization over two runs using NDCG@100 as the metric to maximize. The first column of the table shows the configuration, while the second shows the score of NDCG@100 obtained with that configuration.

Advanced usage examples are provided in the Jupyter Notebook previously referenced.

3 COMPARISON WITH AVAILABLE TOOLS

In this section, we compare `ranx.fuse` with existing tools for Metasearch, *i.e.*, `TrecTools` [23] and `Polyfuse` [11], highlighting their differences and pointing out the innovative aspects of the software library we present in this paper.

First, we introduce the tools considered for the comparison. `TrecTools` was proposed by Palotti et al. [23] as an analysis tool to

```python
from ranx import fuse, optimize_fusion

best_params = optimize_fusion(
    qrels=train_qrels,
    runs=[train_run_1, train_run_2],
    norm="min-max",
    method="wsum",  # Alias for Weighted Sum
    # The metric to maximize during optimization
    metric="ndcg@100",
)

combined_test_run = fuse(
    runs=[test_run_1, test_run_2],
    norm="min-max",
    method="wsum",
    params=best_params,
)
```

Listing 3: Usage example of the `optimize_fusion` method.

Table 2: Optimization report.

Weights	NDCG@100
(0.0, 1.0)	0.502
(0.1, 0.9)	0.517
(0.2, 0.8)	0.531
(0.3, 0.7)	0.553
(0.4, 0.6)	**0.556**
(0.5, 0.5)	0.543
(0.6, 0.4)	0.528
(0.7, 0.3)	0.511
(0.8, 0.2)	0.493
(0.9, 0.1)	0.480
(1.0, 0.0)	0.452

support TREC-like campaigns. It provides several functionalities, such as evaluation metrics for Information Retrieval and fusion algorithms. `Polyfuse` was a companion software shared with the participants of the SIGIR 2018's tutorial on fusion [11]. It comes as a command-line tool providing Metasearch functionalities.

Table 1 reports the fusion approaches provided by `ranx.fuse` and whether they are available in `TrecTools` (TT column) or `Polyfuse` (PL column). Both `TrecTools` and `Polyfuse` provide a small subset of the fusion methods available in `ranx.fuse`. Specifically, `TrecTools` and `Polyfuse` implement nine and eleven Metasearch algorithms, respectively. Conversely, `ranx.fuse` provides all the algorithms supported by the other considered tools for Metasearch, and many others, accounting for 25 fusion methods. We also notice neither `TrecTools` nor `Polyfuse` provides methods requiring a supervised training phase, while `ranx.fuse` supports six.

Unfortunately, both `TrecTools` and `Polyfuse` lack advanced functionalities for fusion optimization. In contrast, `ranx.fuse` implements the fusion optimization functionality described in Section 2.4, which we think will be very convenient for both researchers and practitioners willing to maximize the performance of their Information Retrieval systems. While `ranx.fuse` and `Polyfuse` implement normalization strategies, we noticed `TrecTools` does not provide any. Note that we found those implemented by `Polyfuse` by digging into its source code, as they are not documented.

From a usability perspective, we argue `ranx.fuse` shines among the other tools, providing easy-to-use functionalities developed following a user-centered design and accounting for young researchers with different backgrounds. As pointed out in Section 2, `ranx.fuse` offers documentation (which we will enrich over time) pointing to the original papers of the implemented Metasearch algorithms and providing their BibTex references. Moreover, it provides a Jupyter Notebook showing its main features through a hands-on approach. `TrecTools` offers standard access to the implemented fusion algorithms through distinct functions, while `Polyfuse` comes as a command-line tool only. Both `TrecTools` and `Polyfuse` provide a simple usage example in their repository's readme files as the only documentation for the fusion algorithms they implement.

4 USE CASES

In this section, we discuss some use cases for our proposed library. First, `ranx.fuse` provides the users with working implementations for several Metasearch algorithms that were previously unavailable, allowing the users to try different approaches to fuse the results of multiple search engines. Second, our library offers new opportunities to combine the rankings produced by modern two-stage retrieval pipelines, usually composed of a term matching first-stage retriever, e.g., BM25, and a neural re-ranker. In those pipelines, multiple rankings computed in response to the same queries are often combined through a weighted sum of the scores of the documents assigned by the different retrieval modules [8, 17, 27]. Moreover, finding the best way to combine those rankings could also be convenient for mining hard negatives to use for training Deep Learning-based retrieval models [29]. Finally, researchers can leverage the functions our library exposes to test novel normalization strategies with several Metasearch algorithms in a simple and efficient way. To do so, they can set the `norm` parameter to `null` when calling `fuse` or `optimize_fusion`, causing the normalization step to be bypassed. By doing so, they can test novel normalization strategies by providing data normalized with the approach at test.

5 CONCLUSION

In this paper, we presented `ranx.fuse`, the first Python library dedicated to Metasearch. It provides a user-friendly interface to 25 Metasearch algorithms, both score-based and rank-based methods, six normalization strategies, and an automatic procedure for optimizing the algorithms requiring a training or optimization phase. We compared `ranx.fuse` with the available tools for Metasearch, highlighting the differences among them and pointing out the innovative aspects of our software library from both a functional and a usability perspective. In the future, we intend to enrich `ranx.fuse` with other Metasearch algorithms to expand the already large pool of available methods, aiming at making it the standard library for Metasearch. As all tool provided to a community needs refinement to accommodate specific users' needs and uses, we are open to feature requests and suggestions.

REFERENCES

[1] Javed A. Aslam and Mark H. Montague. 2001. Models for Metasearch. In *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*, W. Bruce Croft, David J. Harper, Donald H. Kraft, and Justin Zobel (Eds.). ACM, 275–284. https://doi.org/10.1145/383952.384007

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

[3] Peter Bailey, Alistair Moffat, Falk Scholer, and Paul Thomas. 2017. Retrieval Consistency in the Presence of Query Variations. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, August 7-11, 2017*, Noriko Kando, Tetsuya Sakai, Hideo Joho, Hang Li, Arjen P. de Vries, and Ryen W. White (Eds.). ACM, 395–404. https://doi.org/10.1145/3077136.3080839

[4] Elias Bassani. 2022. ranx: A Blazing-Fast Python Library for Ranking Evaluation and Comparison. In *Advances in Information Retrieval - 44th European Conference on IR Research, ECIR 2022, Stavanger, Norway, April 10-14, 2022, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 13186)*, Matthias Hagen, Suzan Verberne, Craig Macdonald, Christin Seifert, Krisztian Balog, Kjetil Nørvåg, and Vinay Setty (Eds.). Springer, 259–264. https://doi.org/10.1007/978-3-030-99739-7_30

[5] Gordon V. Cormack, Charles L. A. Clarke, and Stefan Büttcher. 2009. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *SIGIR*. ACM, 758–759.

[6] W Bruce Croft. 2002. Combining approaches to information retrieval. In *Advances in information retrieval*. Springer, 1–36.

[7] Edward A. Fox and Joseph A. Shaw. 1993. Combination of Multiple Searches. In *TREC (NIST Special Publication, Vol. 500-215)*. National Institute of Standards and Technology (NIST), 243–252.

[8] Luyu Gao, Zhuyun Dai, Zhen Fan, and Jamie Callan. 2020. Complementing Lexical Retrieval with Semantic Residual Embedding. *CoRR* abs/2004.13969 (2020). arXiv:2004.13969 https://arxiv.org/abs/2004.13969

[9] D. Frank Hsu and Isak Taksa. 2005. Comparing Rank and Score Combination Methods for Data Fusion in Information Retrieval. *Inf. Retr.* 8, 3 (2005), 449–480. https://doi.org/10.1007/s10791-005-6994-4

[10] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Trans. Inf. Syst.* 20, 4 (2002), 422–446.

[11] Oren Kurland and J. Shane Culpepper. 2018. Fusion in Information Retrieval: SIGIR 2018 Half-Day Tutorial. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, SIGIR 2018, Ann Arbor, MI, USA, July 08-12, 2018*, Kevyn Collins-Thompson, Qiaozhu Mei, Brian D. Davison, Yiqun Liu, and Emine Yilmaz (Eds.). ACM, 1383–1386. https://doi.org/10.1145/3209978.3210186

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

[13] Joon Ho Lee. 1997. Analyses of Multiple Evidence Combination. In *SIGIR*. ACM, 267–276.

[14] David Lillis, Fergus Toolan, Rem W. Collier, and John Dunnion. 2006. ProbFuse: a probabilistic approach to data fusion. In *SIGIR 2006: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Seattle, Washington, USA, August 6-11, 2006*, Efthimis N. Efthimiadis, Susan T. Dumais, David Hawking, and Kalervo Järvelin (Eds.). ACM, 139–146. https://doi.org/10.1145/1148170.1148197

[15] David Lillis, Fergus Toolan, Rem W. Collier, and John Dunnion. 2008. Extending Probabilistic Data Fusion Using Sliding Windows. In *Advances in Information Retrieval , 30th European Conference on IR Research, ECIR 2008, Glasgow, UK, March 30-April 3, 2008. Proceedings (Lecture Notes in Computer Science, Vol. 4956)*, Craig Macdonald, Iadh Ounis, Vassilis Plachouras, Ian Ruthven, and Ryen W. White (Eds.). Springer, 358–369. https://doi.org/10.1007/978-3-540-78646-7_33

[16] David Lillis, Lusheng Zhang, Fergus Toolan, Rem W. Collier, David Leonard, and John Dunnion. 2010. Estimating probabilities for effective data fusion. In *Proceeding of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2010, Geneva, Switzerland, July 19-23, 2010*, Fabio Crestani, Stéphane Marchand-Maillet, Hsin-Hsi Chen, Efthimis N. Efthimiadis, and Jacques Savoy (Eds.). ACM, 347–354. https://doi.org/10.1145/1835449.1835508

[17] Jimmy Lin, Rodrigo Nogueira, and Andrew Yates. 2021. *Pretrained Transformers for Text Ranking: BERT and Beyond*. Morgan & Claypool Publishers. https://doi.org/10.2200/S01123ED1V01Y202108HLT053

[18] Sean MacAvaney, Andrew Yates, Sergey Feldman, Doug Downey, Arman Cohan, and Nazli Goharian. 2021. Simplified Data Wrangling with ir_datasets. In *SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, July 11-15, 2021*, Fernando Diaz, Chirag Shah, Torsten Suel, Pablo Castells, Rosie Jones, and Tetsuya Sakai (Eds.). ACM, 2429–2436. https://doi.org/10.1145/3404835.3463254

[19] Wes McKinney et al. 2011. pandas: a foundational Python library for data analysis and statistics. *Python for high performance and scientific computing* 14, 9 (2011), 1–9.

[20] Mark H. Montague and Javed A. Aslam. 2001. Relevance Score Normalization for Metasearch. In *Proceedings of the 2001 ACM CIKM International Conference on Information and Knowledge Management, Atlanta, Georgia, USA, November 5-10, 2001*. ACM, 427–433. https://doi.org/10.1145/502585.502657

[21] Mark H. Montague and Javed A. Aslam. 2002. Condorcet fusion for improved retrieval. In *Proceedings of the 2002 ACM CIKM International Conference on Information and Knowledge Management, McLean, VA, USA, November 4-9, 2002*. ACM, 538–548. https://doi.org/10.1145/584792.584881

[22] André Mourão, Flávio Martins, and João Magalhães. 2015. Multimodal medical information retrieval with unsupervised rank fusion. *Comput. Medical Imaging Graph.* 39 (2015), 35–45. https://doi.org/10.1016/j.compmedimag.2014.05.006

[23] João R. M. Palotti, Harrisen Scells, and Guido Zuccon. 2019. TrecTools: an Open-source Python Library for Information Retrieval Practitioners Involved in TREC-like Campaigns. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2019, Paris, France, July 21-25, 2019*, Benjamin Piwowarski, Max Chevalier, Éric Gaussier, Yoelle Maarek, Jian-Yun Nie, and Falk Scholer (Eds.). ACM, 1325–1328. https://doi.org/10.1145/3331184.3331399

[24] M. Elena Renda and Umberto Straccia. 2003. Web Metasearch: Rank vs. Score Based Rank Aggregation Methods. In *Proceedings of the 2003 ACM Symposium on Applied Computing (SAC), March 9-12, 2003, Melbourne, FL, USA*, Gary B. Lamont, Hisham Haddad, George A. Papadopoulos, and Brajendra Panda (Eds.). ACM, 841–846. https://doi.org/10.1145/952532.952698

[25] Stephen E. Robertson and Steve Walker. 1994. 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)*, W. Bruce Croft and C. J. van Rijsbergen (Eds.). ACM/Springer, 232–241. https://doi.org/10.1007/978-1-4471-2099-5_24

[26] Milad Shokouhi. 2007. Segmentation of Search Engine Results for Effective Data-Fusion. In *Advances in Information Retrieval, 29th European Conference on IR Research, ECIR 2007, Rome, Italy, April 2-5, 2007, Proceedings (Lecture Notes in Computer Science, Vol. 4425)*, Giambattista Amati, Claudio Carpineto, and Giovanni Romano (Eds.). Springer, 185–197. https://doi.org/10.1007/978-3-540-71496-5_19

[27] Shuai Wang, Shengyao Zhuang, and Guido Zuccon. 2021. BERT-based Dense Retrievers Require Interpolation with BM25 for Effective Passage Retrieval. In *ICTIR '21: The 2021 ACM SIGIR International Conference on the Theory of Information Retrieval, Virtual Event, Canada, July 11, 2021*, Faegheh Hasibi, Yi Fang, and Akiko Aizawa (Eds.). ACM, 317–324. https://doi.org/10.1145/3471158.3472233

[28] Shengli Wu and Fabio Crestani. 2002. Data fusion with estimated weights. In *Proceedings of the 2002 ACM CIKM International Conference on Information and Knowledge Management, McLean, VA, USA, November 4-9, 2002*. ACM, 648–651. https://doi.org/10.1145/584792.584908

[29] Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Jiafeng Guo, Min Zhang, and Shaoping Ma. 2021. Optimizing Dense Retrieval Model Training with Hard Negatives. In *SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, July 11-15, 2021*, Fernando Diaz, Chirag Shah, Torsten Suel, Pablo Castells, Rosie Jones, and Tetsuya Sakai (Eds.). ACM, 1503–1512. https://doi.org/10.1145/3404835.3462880