TrecTools: an Open-source Python Library for Information Retrieval Practitioners Involved in TREC-like Campaigns

Joao Palotti
Qatar Computing Research Institute
Doha, Qatar
jpalotti@hbku.edu.qa

Harrisen Scells
The University of Queensland
Brisbane, Australia
h.scell@uq.net.au

Guido Zuccon
The University of Queensland
Brisbane, Australia
g.zuccon@uq.edu.au

ABSTRACT

This paper introduces TrecTools, a Python library for assisting Information Retrieval (IR) practitioners with TREC-like campaigns. IR practitioners tasked with activities like building test collections, evaluating systems, or analysing results from empirical experiments commonly have to resort to use a number of different software tools and scripts that each perform an individual functionality – and at times they even have to implement ad-hoc scripts of their own. TrecTools aims to provide a unified environment for performing these common activities.

Written in the most popular programming language for Data Science, Python, TrecTools offers an object-oriented, easily extensible library. Existing systems, e.g., `trec_eval`, have considerable barrier to entry when it comes to modify or extend them. Furthermore, many existing IR measures and tools are implemented independently of each other, in different programming languages. TrecTools seeks to lower the barrier to entry and to unify existing tools, frameworks and activities into one common umbrella. Widespread adoption of a centralised solution for developing, evaluating, and analysing TREC-like campaigns will ease the burden on organisers and provide participants and users with a standard environment for common IR experimental activities.

TrecTools is distributed as an open source library under the MIT license at https://github.com/joaopalotti/trectools

CCS CONCEPTS

• **Information systems → Evaluation of retrieval results**; **Test collections**;

ACM Reference Format:
Joao 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 '19), July 21–25, 2019, Paris, France.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3331184.3331399

SIGIR '19, July 21–25, 2019, Paris, France

ACM ISBN 978-1-4503-6172-9/19/07…$15.00
https://doi.org/10.1145/3331184.3331399

1 INTRODUCTION

Rigorous empirical evaluation is a cornerstone of Information Retrieval (IR) research, as demonstrated by the many research efforts centred around the creation of test collections and resources in evaluation campaigns and shared-tasks like the Text Retrieval Conference (TREC), the Conference and Labs of the Evaluation Forum (CLEF), the NII Testbeds and Community for information access Research (NTCIR), etc..

Creating test collections for IR evaluation and participating to such shared tasks is time consuming – with a large amount of effort spent in implementing the evaluation settings, rather than the methods to address the IR task, e.g., creating baselines, selecting documents for assessment, performing relevance assessment, measuring system effectiveness, etc. In this paper we present `TrecTools`[1], an open source Python software package to support the creation and use of IR evaluation resources. TrecTools aims to support both IR campaign organizers and participants to deal with a number of recurrent, tedious and time-consuming procedures. For evaluation campaigns organisers, TrecTools allows one to (1) easily create common IR baselines using popular toolkits such as Indri [20] and Terrier [12], (2) create document pools for relevance assessment from retrieval runs, making many popular pooling strategies available, (3) evaluate each considered run using a wide range of evaluation measures, (4) perform statistical significance analysis between runs and baselines, and (5) create standard visualisations and reporting of IR systems effectiveness. For evaluation campaigns participants and subsequent users (i.e., those that did not originally participate in the campaign), TrecTools allows one to perform the relevant aforementioned tasks at the campaign level (i.e., across all participants) or at an individual level (i.e., for a single participant in a campaign).

Before TrecTools, IR practitioners were required to use a series of *separate*, *independent* and *unlinked* tools such as `trec_eval` for computing evaluation measures, workspaces e.g., in R/Matlab for statistical analysis and result plotting, etc., and were required to implement their own routines for other tasks, such as pooling. TrecTools aims to provide practitioners with a unified environment to perform these common tasks, as well as standard and verified (i.e., by automatically comparing the output of existing tools, e.g., `trec_eval`, using unit tests) implementations of common approaches (evaluation measures, pooling strategies, etc.) for each task. In particular, TrecTools goes beyond the popular software package `trec_eval` [4] by additionally implementing a number of common IR workflows, such as document pool creation and result visualisation, as well as extend the range of evaluation measures

[1] See http://www.ielab.io/trectools

supported. TrecTools is built upon standard Data Science libraries in Python, such as Numpy, Scipy, Pandas and Matplotlib, intending to allow for a rapid and smooth learning curve for new users.

2 RELATED WORK

`trec_eval` is perhaps the most popular software used by IR researchers. This tool takes as input a run and a relevance assessments (qrels) files, and outputs many of the core IR evaluation measures, including mean average precision (MAP), binary preference (bpref) and precision at different cut-offs (e.g., P@10). The input formats imposed by `trec_eval` (i.e., the TREC result and qrel formats) have become a de-facto standard for other, subsequent tools. However, there are a number of evaluation measures not implemented in `trec_eval`, including ad-hoc measures like Rank Biased Precision (RBP) [14], diversity measures like α-nDCG [5], risk-sensitive measures like U-risk [24], and multi-dimensional relevance measures [27]. `TrecTools`, on the other hand, implements all the measures listed above, and eases the implementation of new measures through extensible classes; we furthermore plan to implement new measures arising from the C/W/L framework [3], e.g., IFT goal and rate, and the bejewelled player model. In addition, it facilitates the analysis of the evaluation results by providing automated statistical analysis, plots generation and LaTeXtables generation.

Similar to TrecTools, the Java-based `EvALL` tool provides native and verified implementations of most IR evaluation measures, and includes features such as statistical significance analysis, results visualisation, and LaTeXtables generation [1]. TrecTools goes however beyond EvALL's functionalities by supporting other tasks in the collection creation pipeline, e.g, providing utilities for creating retrieval baselines and assessment pools.

`PyIndri` [23] and `Pytrec_eval` [22] are Python interfaces to the Indri IR toolkit and `trec_eval`, respectively. While comparable to TrecTools with respect to baselines creation and IR measures provision (although still limited to those in `trec_eval`), these tools lack many of TrecTools' functionalities (e.g., results analysis, tables and assessment pools generation, etc.).

Other tools have been released to aid evaluation campaign organisers, including `VisualPool` [10] and `ircor` [21]. VisualPool allows to visualise the results of using different document pooling strategies, thus informing collection creators about the effect of using a strategy over another for selecting documents for relevance assessment. Many of the popular pooling strategies implemented in VisualPool are also available in TrecTools, e.g., Depth@K [19], CombMAXTake@N or CombMNZTake@N [13], RRFTake@N [6] and RBPTake@N [14]. `ircor` is an R package that provides implementations of correlation measures for comparing results rankings or system rankings. This tool implements for example τ-AP [26], a extension of the Kendall-τ correlation that puts higher weight to matches found at higher rank positions. TrecTool also provides an implementation of τ-AP (as well as, Kendall-τ, Pearson and Spearman). Nevertheless, ircor implements variations of τ-AP that cope with ties [21]: a functionality not yet present in TrecTool – but planned for future releases.

3 TRECTOOLS FEATURES

TrecTools is implemented in Python using standard Data Science libraries and using the object-oriented paradigm. Each of the key components of an evaluation campaign is mapped to a class: classes for runs (TrecRun), topics/queries (TrecTopic), assessment pools (TrecPools), relevance assessments (TrecQrel) and the evaluation results (TrecRes). Evaluation results can be produced by TrecTools itself using the evaluation metrics implemented in the tool, or be imported from the output file of `trec_eval` and derivatives. General features of TrecTools are shown in Figure 1. The features that are currently implemented and available to use in TrecTools are as follows.

Querying IR Systems. Benchmark runs can be obtained directly from one of the IR toolkits that are integrated in TrecTools. There is support for issuing full-text queries to Indri, Terrier[2] and PISA[3] toolkits. Future releases will include other toolkits (e.g., Elasticsearch, Anserini [25], etc.) and support for specific query languages (e.g., Indri's query language, Boolean queries). Examples of baselines creation are given in Figure 2.

Pooling Techniques. The following techniques for pool creation from a set of runs are implemented: Depth@K [19], Take@N [11], Comb-Min/Max/Med/Sum/ANZ/MNZ [13], RRFTake@N [6], RBPTake@N [14]. Examples are shown in Figure 3.

Evaluation Measures. Currently implemented and verified measures include widely used metrics such as Precision at depth K, Recall at depth K, MAP, NDCG, Bpref and RBP [14], as well as recently developed ones, such as uBpref [15], uRBP [27] and the MM framework [17]. Implemented in TrecTools is the option to break ties using document score (i.e., similar to trec_eval), or document ranking (i.e., similar to the original implementation of RBP[4]). Additionally, TrecTools also allows to compute the residual of the evaluation measure and analyse the relative presence of un-assessed documents. Examples are given in Figure 4.

Correlation and Agreement Analysis. The Pearson, Spearman, Kendall and τ_{ap} correlations between system rankings can be computed directly using TrecTools. Agreement measures between relevance assessment sets can be obtained with Kappa or Jaccard. Examples are provided in Figures 5 and 6.

Fusion Techniques. Runs can be fused using the following techniques: Comb-Max/Min/Sum/Mnz/Anz/Med (both using scores and document rankings) [9], RBPFusion [14], RRFFusion [6], or BordaCountFusion [2]. Fusion techniques are provided for meta-analysis. Examples are shown in Figure 7.

4 CONCLUSION

In this paper we introduced TrecTools, an open source Python library for assisting IR practitioners with TREC-like evaluation campaigns. Some of the use cases for campaign organisers using TrecTools include automatically creating baselines by querying the Indri and Terrier IR toolkits (Figure 2), creating meta-rankings using multiple runs (Figure 3), performing analysis of agreement between assessments made by different assessors (e.g., Kappa coefficient, Figure 6), and producing visualisations and LaTeX results tables (Figure 1). Some of the use cases for participants of campaigns and regular users include performing statistical significance analysis between runs and LaTeXresults tables (Figures 4). While TrecTools

[2]Thanks to Craig Macdonald for implementing support for Terrier v5.0.
[3]Thanks to Antonio Mallia for implementing support for PISA (https://github.com/pisa-engine/pisa)
[4]Available at https://people.eng.unimelb.edu.au/ammoffat/abstracts/mz08acmtois.html

```python
from trectools import TrecQrel, procedures

qrels_file = "./qrel/robust03_qrels.txt"
qrels = TrecQrel(qrels_file)

# Generates a P@10 graph with all the runs in a directory
path_to_runs = "./robust03/runs/"
runs = procedures.list_of_runs_from_path(path_to_runs, "*.gz")

results = procedures.evaluate_runs(runs, qrels, per_query=True)
p10 = procedures.extract_metric_from_results(results, "P_10")
procedures.plot_system_rank(p10, display_metric="P@10")
# Sample output with one run for each participating team in robust03:
```

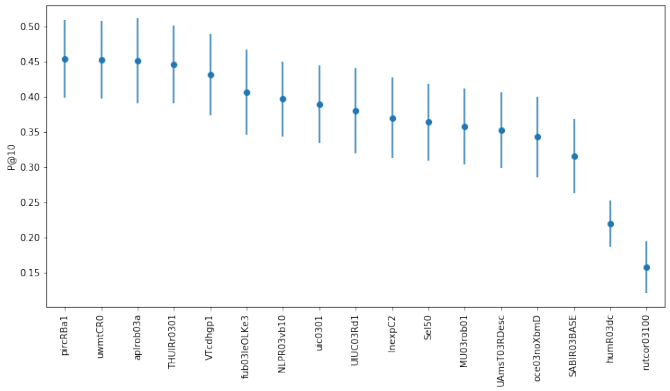

Figure 1: Code Snippets and toy examples with TrecTools. Note the plot is generated by simply calling the method `procedures.plot_system_rank()`.

does not help with obtaining relevance assessments, it can be integrated into existing tools such as `Revelation!` [8]. TrecTools is by no means a 'completed' package: it is open to new evaluation measures and activities as suggested and contributed by the community – and it is fully extensible. Despite the care taken for correctness of results, no software is bullet proof — although, for TrecTools, unit tests are written for each component of the library and methods are also validated against previously released tools for an activity, if any. TrecTools has already been successfully used throughout the creation and results analysis of the CLEF eHealth evaluation campaigns [7, 16, 18, 28].

Acknowledgements. Guido Zuccon is the recipient of an Australian Research Council DECRA Research Fellowship (DE180101579).

REFERENCES

[1] Enrique Amigó, Jorge Carrillo-de Albornoz, Mario Almagro-Cádiz, Julio Gonzalo, Javier Rodríguez-Vidal, and Felisa Verdejo. 2017. Evall: Open access evaluation for information access systems. In *SIGIR*. ACM, 1301–1304.
[2] Javed A. Aslam and Mark Montague. 2001. Models for Metasearch. In *SIGIR*. ACM, 276–284. https://doi.org/10.1145/383952.384007
[3] Leif Azzopardi, Paul Thomas, and Alistair Moffat. 2019. cwl_eval: An Evaluation Tool for Information Retrieval. In *SIGIR*. ACM.
[4] Chris Buckley et al. 2004. The trec_eval evaluation package.
[5] Charles LA Clarke, Maheedhar Kolla, Gordon V Cormack, Olga Vechtomova, Azin Ashkan, Stefan Büttcher, and Ian MacKinnon. 2008. Novelty and diversity in information retrieval evaluation. In *SIGIR*. ACM, 659–666.
[6] Gordon V Cormack, Charles LA Clarke, and Stefan Buettcher. 2009. Reciprocal rank fusion outperforms condorcet and individual rank learning methods.. In *SIGIR*, Vol. 9. 758–759.
[7] Jimmy, Guido Zuccon, João Palotti, Lorraine Goeuriot, and Liadh Kelly. 2018. Overview of the CLEF 2018 Consumer Health Search Task. In *CLEF*. http://ceur-ws.org/Vol-2125/invited_paper_17.pdf
[8] Bevan Koopman and Guido Zuccon. 2014. Relevation!: An open source system for information retrieval relevance assessment. In *SIGIR*. 1243–1244.

```python
from trectools import TrecTopics, TrecTerrier, TrecIndri

# Loads some topics from a file (e.g., topics.txt)
"""
<topics>
<topic number="201" type="single">
<query>amazon raspberry pi</query>
<description> You have heard quite a lot about cheap computing as being
   the way of the future,
including one recent model called a Raspberry Pi. You start thinking about
   buying one, and wonder how much they cost.
</description>
</topic>
</topics>
"""
topics = TrecTopics().read_topics_from_file("topics.txt")
# Or...load topics from a Python dictionary
topics = TrecTopics(topics={'201': u'amazon_raspberry_pi'})
topics.printfile(fileformat="terrier")
#<topics>
# <top>
# <num>201</num>
# <title>amazon raspberry pi</title>
# </top>
#</topics>

topics.printfile(fileformat="indri")
#<parameters>
# <trecFormat>true</trecFormat>
# <query>
# <id>201</id>
# <text>#combine( amazon raspberry pi )</text>
# </query>
#</parameters>

topics.printfile(fileformat="indribaseline")
#<parameters>
# <trecFormat>true</trecFormat>
# <query>
# <id>201</id>
# <text>amazon raspberry pi</text>
# </query>
#</parameters>

tt = TrecTerrier(bin_path="<PATH>/terrier/bin/") # where trec_terrier.sh
   is located
# Runs PL2 model from Terrier with Query Expansion
tr = tt.run(index="<PATH>/terrier/var/index", topics="topics.xml.gz",
   qexp=True,
model="PL2", result_file="terrier.baseline", expTerms=5, expDocs=3,
   expModel="Bo1")

ti = TrecIndri(bin_path="~/<PATH>/indri/bin/") # where IndriRunQuery is
   located
ti.run(index="<PATH>/indriindex", topics, model="dirichlet",
   parameters={"mu":2500},
result_file="trec_indri.run", ndocs=1000, qexp=True, expTerms=5, expDocs=3)
```

Figure 2: Code Snippets for manipulating topic formats and querying IR toolkits (shown here: Terrier and Indri).

[9] Joon Ho Lee. 1997. Analyses of Multiple Evidence Combination. In *SIGIR*. ACM, 267–276. https://doi.org/10.1145/258525.258587
[10] Aldo Lipani, Mihai Lupu, and Allan Hanbury. 2017. Visual Pool: A Tool to Visualize and Interact with the Pooling Method. In *SIGIR*. ACM, 1321–1324. https://doi.org/10.1145/3077136.3084146
[11] Aldo Lipani, Joao Palotti, Mihai Lupu, Florina Piroi, Guido Zuccon, and Allan Hanbury. 2017. Fixed-cost pooling strategies based on IR evaluation measures. In *ECIR*. Springer, 357–368.
[12] Craig Macdonald, Richard McCreadie, Rodrygo LT Santos, and Iadh Ounis. 2012. From puppy to maturity: Experiences in developing Terrier. *Proc. of OSIR at SIGIR* (2012), 60–63.
[13] Craig Macdonald and Iadh Ounis. 2006. Voting for candidates: adapting data fusion techniques for an expert search task. In *CIKM*. ACM, 387–396.
[14] Alistair Moffat and Justin Zobel. 2008. Rank-biased Precision for Measurement of Retrieval Effectiveness. *ACM Trans. Inf. Syst.* 27, 1, Article 2 (Dec. 2008), 27 pages. https://doi.org/10.1145/1416950.1416952
[15] Joao Palotti, Lorraine Goeuriot, Guido Zuccon, and Allan Hanbury. 2016. Ranking health web pages with relevance and understandability. In *SIGIR*. ACM, 965–968.
[16] João Palotti, Guido Zuccon, Lorraine Goeuriot, Liadh Kelly, Allan Hanbury, Gareth J. F. Jones, Mihai Lupu, and Pavel Pecina. 2015. ShARe/CLEF eHealth Evaluation Lab 2015, Task 2: User-centred Health Information Retrieval. In *CLEF*.
[17] Joao Palotti, Guido Zuccon, and Allan Hanbury. 2018. MM: A new Framework for Multidimensional Evaluation of Search Engines. In *CIKM*. ACM, 1699–1702.

```
from trectools import TrecPool, TrecRun

r1 = TrecRun("./robust03/runs/input.aplrob03a.gz")
r2 = TrecRun("./robust03/runs/input.UIUC03Rd1.gz")

len(r1.topics()) # 100 topics

# Creates document pools with r1 and r2 using different strategies:

# Strategy1: Creates a pool with top 10 documents of each run:
pool1 = TrecPool.make_pool([r1, r2], strategy="topX", topX=10) # Pool with
   1636 unique documents.

# Strategy2: Creates a pool with 2000 documents (20 per topic) using the
   reciprocal ranking strategy by Gordon, Clake and Buettcher:
pool2 = TrecPool.make_pool([r1,r2], strategy="rrf", topX=20, rrf_den=60) #
   Pool with 2000 unique documents.

# Check to see which pool covers better my run r1
pool1.check_coverage(r1, topX=10) # 10.0
pool2.check_coverage(r1, topX=10) # 8.35

# Export documents to be judged using Relevation! visual assessing system
pool1.export_document_list(filename="mypool.txt", with_format="relevation")
```

Figure 3: Code Snippets for generating and exporting document pools using different pooling strategies.

```
from trectools import TrecQrel, TrecRun, TrecEval

# A typical evaluation workflow
r1 = TrecRun("./robust03/runs/input.aplrob03a.gz")
r1.topics()[:5] # Shows the first 5 topics: 601, 602, 603, 604, 605

qrels = TrecQrel("./robust03/qrel/robust03_qrels.txt")

te = TrecEval(r1, qrels)
rbp, residuals = te.getRBP()           # RBP: 0.474, Residuals: 0.001
p100 = te.getPrecisionAtDepth(100)     # P@100: 0.186

# Check if documents retrieved by the system were judged:
r1.get_mean_coverage(qrels, topX=10)   # 9.99
r1.get_mean_coverage(qrels, topX=1000) # 481.390
# On average for system 'input.aplrob03a' participating in robust03, 480
   documents out of 1000 were judged.

# Loads another run
r2 = TrecRun("./robust03/runs/input.UIUC03Rd1.gz")

# Check how many documents, on average, in the top 10 of r1 were retrieved
   in the top 10 of r2
r1.check_run_coverage(r2, topX=10) # 3.64

# Evaluates r1 and r2 using all implemented evaluation metrics
result_r1 = r1.evaluate_run(qrels, per_query=True)
result_r2 = r2.evaluate_run(qrels, per_query=True)

# Inspect for statistically significant differences between the two runs
   for  P_10 using two-tailed Student t-test
pvalue = result_r1.compare_with(result_r2, metric="P_10") # pvalue: 0.0167
```

Figure 4: Code snippets showing evaluation options available in TrecTools.

[18] João Palotti, Guido Zuccon, Jimmy, Pavel Pecina, Mihai Lupu, Lorraine Goeuriot, Liadh Kelly, and Allan Hanbury. 2017. CLEF 2017 Task Overview: The IR Task at the eHealth Evaluation Lab - Evaluating Retrieval Methods for Consumer Health Search. In *CLEF*. http://ceur-ws.org/Vol-1866/invited_paper_16.pdf

[19] K Spark-Jones. 1975. Report on the need for and provision of an'ideal'information retrieval test collection. *Computer Laboratory* (1975).

[20] Trevor Strohman, Donald Metzler, Howard Turtle, and W Bruce Croft. 2005. Indri: A language model-based search engine for complex queries. In *Proceedings of the International Conference on Intelligent Analysis*, Vol. 2. 2–6.

[21] Julián Urbano and Mónica Marrero. 2017. The Treatment of Ties in AP Correlation. In *SIGIR*. 321–324.

[22] Christophe Van Gysel and Maarten de Rijke. 2018. Pytrec_Eval: An Extremely Fast Python Interface to Trec_Eval. In *SIGIR*. ACM, 873–876.

[23] Christophe Van Gysel, Evangelos Kanoulas, and Maarten de Rijke. 2017. Pyndri: a Python Interface to the Indri Search Engine. In *ECIR*, Vol. 2017. Springer.

[24] Lidan Wang, Paul N Bennett, and Kevyn Collins-Thompson. 2012. Robust ranking models via risk-sensitive optimization. In *SIGIR*. ACM, 761–770.


```
from trectools import misc, TrecRun, TrecQrel, procedures

qrels_file = "./robust03/qrel/robust03_qrels.txt"
path_to_runs = "./robust03/runs/"

qrels = TrecQrel(qrels_file)

runs = procedures.list_of_runs_from_path(path_to_runs, "*.gz")

results = procedures.evaluate_runs(runs, qrels, per_query=True)

# check the system correlation between P@10 and MAP using Kendall's tau
   for all systems participating in a campaign
misc.get_correlation( misc.sort_systems_by(results, "P_10"),
                      misc.sort_systems_by(results, "map"), correlation =
                         "kendall") # Correlation: 0.7647

# check the system correlation between P@10 and MAP using Tau's ap for all
   systems participating in a campaign
misc.get_correlation( misc.sort_systems_by(results, "P_10"),
                      misc.sort_systems_by(results, "map"), correlation =
                         "tauap") # Correlation: 0.77413
```

Figure 5: Code Snippets for obtaining correlation measures from a set of runs.

```
# Code snippet to check correlation between two sets of relevance
   assessment (e.g., made by different cohorts - assessments made by
   medical doctors Vs. crowdsourced assessments)
from trectools import  TrecQrel

original_qrels_file =  "./robust03/qrel/robust03_qrels.txt"
# Changed the first 10 assessments from 0 to 1
modified_qrels_file = "./robust03/qrel/mod_robust03_qrels.txt"

original_qrels = TrecQrel(original_qrels_file)
modified_qrels = TrecQrel(modified_qrels_file)

# Overall agreement
original_qrels.check_agreement(modified_qrels) # 0.99
# Fleiss' kappa agreement
original_qrels.check_kappa(modified_qrels) # P0: 1.00, Pe = 0.90
# Jaccard similarity coefficient
original_qrels.check_jaccard(modified_qrels) # 0.99
# 3x3 confusion matrix (labels 0, 1 or 2)
original_qrels.check_confusion_matrix(modified_qrels)
# [[122712     10      0]
# [     0   5667      0]
# [     0      0    407]]
```

Figure 6: Code Snippets for obtaining agreement measures from a pair of relevance assessments.

```
from trectools import TrecRun, TrecEval, fusion

r1 = TrecRun("./robust03/runs/input.aplrob03a.gz")
r2 = TrecRun("./robust03/runs/input.UIUC03Rd1.gz")

# Easy way to create new baselines by fusing existing runs:
fused_run = fusion.reciprocal_rank_fusion([r1,r2])
TrecEval(r1, qrels).getPrecisionAtDepth(25)          # P@25: 0.3392
TrecEval(r2, qrels).getPrecisionAtDepth(25)          # P@25: 0.2872
TrecEval(fused_run, qrels).getPrecisionAtDepth(25)   # P@25: 0.3436

# Save run to disk with all its topics
fused_run.print_subset("my_fused_run.txt", topics=fused_run.topics())
```

Figure 7: Code Snippets for generating fusing two runs (Reciprocal Rank fusion shown here).

[25] Peilin Yang, Hui Fang, and Jimmy Lin. 2018. Anserini: Reproducible ranking baselines using Lucene. *JDIQ* 10, 4 (2018), 16.

[26] Emine Yilmaz, Javed A Aslam, and Stephen Robertson. 2008. A new rank correlation coefficient for information retrieval. In *SIGIR*. ACM, 587–594.

[27] Guido Zuccon. 2016. Understandability biased evaluation for information retrieval. In *ECIR*. Springer, 280–292.

[28] Guido Zuccon, João Palotti, Lorraine Goeuriot, Liadh Kelly, Mihai Lupu, Pavel Pecina, Henning Mueller, Julie Budaher, and Anthony Deacon. 2016. The IR Task at the CLEF eHealth Evaluation Lab 2016: User-centred Health Information Retrieval. In *CLEF*, Vol. 1609. 15–27.