Dockerizing Automatic Routing Runs for The Open-Source IR Replicability Challenge (OSIRRC 2019)

Timo Breuer
firstname.lastname@th-koeln.de
Technische Hochschule Köln
Cologne, Germany

Philipp Schaer
firstname.lastname@th-koeln.de
Technische Hochschule Köln
Cologne, Germany

ABSTRACT

In the following, we describe our contribution to the Docker infrastructure for ad hoc retrieval experiments initiated by the Open-Source IR Replicability Challenge (OSIRRC) at SIGIR 2019. We contribute automatic routing runs as Grossman and Cormack specified them during their participation in the TREC Common Core track 2017. Reimplementations of these runs are motivated by the CENTRE lab held at the CLEF conference in 2019. More specifically, we investigated the replicability and reproducibility of WCRobust04 and WCRobust0405. In the following, we give insights into the adaption of our replicated CENTRE submissions and report on our experiences made.

Image Source:
`https://github.com/osirrc/irc-centre2019-docker`
Docker Hub:
`https://hub.docker.com/r/osirrc2019/irc-centre2019`

1 INTRODUCTION

In 2018 the ACM introduced Artifact and Review Badging[1] concerned with procedures assuring repeatability, replicability, and reproducibility. According to the ACM definitions, reproducibility assumes stated precision to be obtainable with a different team and a different setup. Preliminarily, the stated precision should be replicable by a different team with the original setup of the artifacts. Thus replicability is an essential requirement towards reproducible results. Both objectives are closely coupled, and requirements of reproducibility are related to those of replicability.

Empirical studies manifest most findings in the field of information retrieval (IR). Generally, these findings have to be replicable and reproducible in order to bring further use for future research and applications. Results may become invalid under slightly different conditions, thus reasons for non-reproducibility are manifold. Choosing weak baselines, selective reporting, or hidden and missing information are some reasons for non-reproducible findings. The well-known meta-studies by Armstrong [1, 2] reveal the problem of illusory evaluation gains when comparing to weak or inappropriate baselines. More recent studies by Wang et al. [12] or Lin et al. [9] show that this circumstance is still a huge problem, especially, with regards to neural IR.

During the last years the IR community launched several *Evaluation as a Service* initiatives [8]. Attempts were made towards keeping test collections consistent across different systems, providing infrastructures for replicable environments, or increasing transparency by the use of open-source software.

The OSIRRC workshop[2] located at SIGIR 2019 is devoted to the replicability of ad hoc retrieval systems. A major subject of interest is the integration of IR systems into a Docker infrastructure. Participants contribute existing IR systems by adapting them to pre-defined interfaces. The organizers put the focus on standard test collections in order to keep underlying data across different systems consistent.

Docker facilitates the deployment of complex software systems. Dependencies and configurations can be specified in a standardized way. The resulting images will be run with the help of os-level virtualization. A growing community and efficient resource management make Docker preferable to other virtualization alternatives. Using Docker addresses some barriers to replicability. With the help of clearly specified environments, configuration errors and obscurities can be reduced to a minimum. An early attempt at making retrieval results replicable with Docker was made by Yang et al. [11]. The authors describe an online service, which evaluates submitted retrieval systems run in Docker containers. Likewise, Crane [5] proposes Docker as packaging tool for machine learning systems with multiple parameters.

The CENTRE lab at CLEF is another initiative concerned with the replicability and reproducibility of IR systems. Our participation in CENTRE@CLEF19 [6] was devoted to the replicability and reproducibility of automatic routing runs. In order to contribute our code submissions to the IR community, we integrate them into the proposed Docker infrastructure of OSIRRC.

The remainder of the paper is structured as follows. In section 2, we will summarize our submission to CENTRE@CLEF19. In this context, the general concept of automatic routing runs will be introduced. Section 3 gives insights into the adaption of our reimplementations to the Docker infrastructure. The last section will conclude with the resulting benefits and experiences made.

2 REIMPLEMENTATION OF AUTOMATIC ROUTING RUNS

In the context of CENTRE@CLEF19, participants were obliged to replicate, reproduce, and generalize IR systems submitted at previous conferences. Our participation in the CENTRE lab was motivated by the replication and reproduction of automatic routing runs submitted by Grossman and Cormack to the TREC Common Core Track in 2017 [4]. According to the guidelines of the CENTRE lab, participants have to reimplement original procedures.

[1] https://www.acm.org/publications/policies/artifact-review-badging

[2] https://osirrc.github.io/osirrc2019/

Run	Test	Training
WCRobust04	New York Times	Robust04
WCRobust0405	New York Times	Robust04+05

Table 1: Run constellations: Replicated runs are made of rankings from New York Times (NYT) documents. The underlying data of Robust04 and Robust05 are the TREC Disks 4&5 (minus congressional records) and the AQUAINT corpus, respectively.

Replicability is evaluated by applying reimplementations to data collections originally used. Reproducibility is evaluated by applying reimplementations to new data collections. In the following, we will describe the general concept of automatic routing runs, we will continue with some insights into our reimplementations and conclude this section with the replicated results.

2.1 Automatic Routing Runs

Grossman's and Cormack's contributions to the TREC Common Core 2017 track follow either a continuous active learning or routing approach [7]. We focus on the latter in accordance with the CENTRE guidelines. As the authors point out, automatic routing runs are based on deriving ranking profiles for specific topics. With the help of relevance judgments, these profiles are constructed. Opposed to other retrieval approaches, no explicit query is needed in order to derive a document ranking for a specific topic. Typically, queries are stemmed from topics and corresponding narratives. The proposed routing mechanisms, however, do not require such an input. Grossman and Cormack chose to implement the profile derivation with the help of a logistic regression classifier. They convert text documents into numerical representations by the determination of tfidf weights. Subsequently, they train the classifier with these tfidf features in combination with binary relevance judgments. The classifier is used to rank documents of another corpus which is different from the one used for training. The likelihood of documents being relevant will serve as a score. Since there is no human intervention in the described procedure, it is fully automatic. It has to be considered, that this approach is limited to corpora which share relevance judgments for the same topics. The entire corpus is ranked, whereas for each topic, the 10,000 highest ranking documents are used for evaluation. Grossman and Cormack rank the New York Times corpus with training data based on the Robust04 collection. The resulting run is titled WCRobust04. By enriching training data with documents from the Robust05 collection, they acquire improvements in terms of MAP and P@10. The resulting run is titled WCRobust0405. Table 1 shows an overview of run constellations as they were used in the context of the CENTRE lab.

2.2 Reimplementation

Based on the description by Grossman and Cormack, we chose to reimplement the system from scratch. CENTRE organizers premise the use of open-source software. The Python community offers a rich toolset of open and free software. We pursued a Python-only implementation, since the required components were largely available in existing packages. A detailed description of our implementation is available in our workshop report [4]. The general workflow can be split into two processing stages.

2.2.1 Data preparation. The first stage will prepare corpora data. Besides different compression formats, we also consider diverging text formatting. We adapted the preparation steps specifically to the characteristics of the corpora. After extraction, single documents will be written to files which contain parsed text data. Grossman and Cormack envisage a union corpus in order to derive tfidf-features. The corpus with training samples as well as the corpus to be ranked are supposed to be unified. This proceeding results in training features that are augmented by the vocabulary of the corpus to be ranked. In their contribution to the ECIR reproducibility workshop in 2019 Yu et al. consider this augmentation to be insignificant [13]. In our experimental setups, we compare resulting runs of augmented and non-augmented training features and confirm the assumptions made by Yu et al. It is reasonable to neglect tfidf-derivation from a unified corpus. Features can be taken solely from the training corpus without negatively affecting evaluation measures. Due to these findings, we deviate from the original procedure with regards to the tfidf-derivation. Our training features are exclusively derived from Robust corpora.

2.2.2 Training & Prediction. Single document files with parsed text result from the data preparation step. The scikit-learn package [10] offers the possibility to derive tfidf-features by implementing the TfidfVectorizer. A term-document matrix is built by providing document files from the Robust corpora to the TfidfVectorizer. Text documents from all corpora are converted to numerical representations with regards to this matrix. Converting documents from the New York Times corpus, for instance, can result in vectors that do not cover the complete vocabulary of specific documents. This is a requirement for retrieving document vectors of equal length. As pointed out above, this is not affecting evaluation outcomes. Relevance judgments are converted to a binary scale and serve for selecting training documents from the Robust corpora. Based on a given topic, judged documents are converted to features vectors and are prepared as training input for the logistic regression classifier. We dump the training data in SVMlight format to keep our workflow compatible with other machine learning frameworks. In our case, we make use of the LogisticRegression model from the scikit-learn package. The model is trained topic-wise with features being either relevant or not. Afterwards, each document of the new corpus will be scored. We order documents by descending score for each topic. The 10,000 highest ranking documents form the final ranking of a single topic.

2.2.3 Outcomes. While replicated P@10 values come close to those given by Grossman and Cormack, MAP values stay below the baseline (more details in section 3.4). Especially reproduced values (derived from another corpus) drop significantly and offer a starting point for future investigations [4].

3 DOCKER ADAPTION

In the following, we describe the portation of our replicated CENTRE submission into Docker images. More specifically we give insights, how the workflow is adapted to the given hooks. In this

context, we refer to the procedure illustrated in the previous section. Contributors of retrieval systems have to adapt their systems with respect to pre-defined hooks. These hooks are implemented with the help of scripts for initialization, indexing, searching, training, and interaction. They should be located in the root directory of the Docker containers. A Python-based framework ("jig") will call these scripts and invoke the corresponding processing steps. Automatic routing runs slightly differ from the conventional ad hoc approach. Instead of deriving rankings based on a query, a classification model is devised based on judged documents for a given topic. The general workflow of our submission is illustrated in figure 1.

Supported Collections:
`robust04`, `robust05`, `core17`

Supported Hooks:
`init`, `index`, `search`

3.1 Dockerfile

Since our implementation is completely done with Python, the image relies on an existing Python 3 image. Upon image building, directories will be made and the three scripts for initialization, indexing, and searching will be copied. The required corpora will be mounted as volumes when starting the container.

3.2 Hooks

3.2.1 Initialization. On initialization, the source code will be downloaded from a public GitHub repository. Required Python packages will be installed. Depending on the specified run, either WCRobust04 or WCRobust0405 will be replicated, and the corresponding scripts are prepared.

3.2.2 Indexing. After successful initialization, indexing is done by determining tfidf-features. Data extraction and text processing result in single documents for each corpus. A term-document matrix is constructed by using the TfidfVectorizer of the scikit-learn package. In combination with qrel files from Robust04 and Robust05, documents will be picked and transformed into tfidf-features with respect to the term-document matrix. Likewise, the entire NYT corpus is transformed into a numerical representation according to this matrix. At the end of the indexing process, a Python shelf containing tfidf-features of all documents from the NYT corpus and SVMlight formatted tfidf-features remain as artifacts. They will be committed to the resulting image, all other artifacts like the vectorizer of extracted document files have to be deleted in order to keep the image size low.

3.2.3 Searching. The "jig" will start a new container running the committed image of the previous step. In our case, the "searching" process consists of training a topic model and scoring the entire NYT corpus for each topic. We make use of the logistic regression classifier implemented in the scikit-learn package, although other machine learning models should be easy to integrate. The 10,000 highest scorings for each topic will be merged into one final run file. The "jig" will handle evaluation by using `trec_eval`.

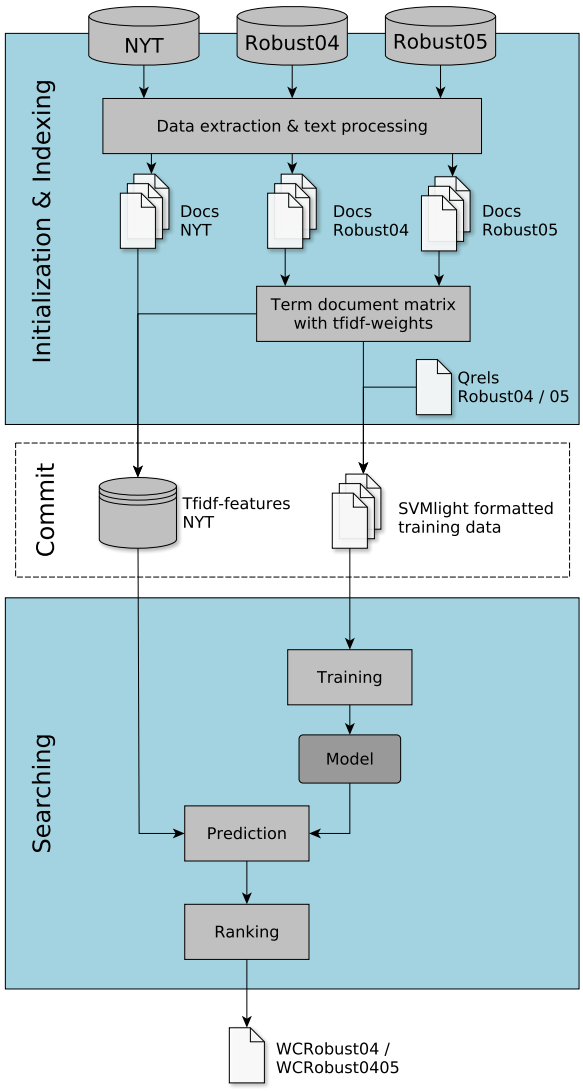

Figure 1: Depiction of how our reimplementation of automatic routing runs is adapted to the workflow given by the "jig". The two cyan boxes include the processing steps which are conducted in the running container instances. The objects within the dashed rectangle are committed to the Docker image after indexing is done.

3.3 Modifications

Our CENTRE submission[3] was adaptable with little effort. The following modifications were necessary in order to prepare the code for the Docker infrastructure.

After initialization and indexing are done, the "jig" will commit the container's changes, including the indexed collection. The new image will be run in a second container which conducts the ranking. Due to this given workflow, we were obliged to split our CENTRE

[3] https://bitbucket.org/centre_eval/c2019_irc/

	Run	MAP	P@10
Baseline	WCRobust04	0.3711	0.6460
	WCRobust0405	0.4278	0.7500
Replicability	WCRobust04	0.2971	0.6820
	WCRobust0405	0.3539	0.7360

Table 2: Results of replicated runs in comparison to the baseline which is given by Grossman and Cormack. All runs are based on 50 topics. [7].

submission into two main processing steps. We decided to keep the tfidf artifacts only in order to keep the committed image as small as possible. The text artifacts resulting from the preprocessing will be deleted after the determination of the term-document matrix/TfidfVectorizer and tfidf-features. At first, we omitted the removal of unneeded documents, resulting in large Docker image sizes that could not be handled on moderate hardware.

The data extraction and text processing steps are parallelized, speeding up the preprocessing. In this context special attention had to be paid to the compressed files by the same name with different endings (.0z, .1z, .2z), since extracting these files in parallel will result in name conflicts.

3.4 Evaluation Outcomes

The evaluation outcomes of our replicated runs are given in table 2. We were not able to fully replicate the baseline given by Grossman and Cormack. Replicated evaluation outcomes are slightly worse compared to the originals but are similar to results achieved by Yu et al. with their *classification only* approach [13]. Evaluation outcomes can be improved by enriching training data with an additional corpus (in this case combining Robust04 and Robust05). By using the OSSIRC Docker infrastructure, we can rebuild the exact environment used for our submissions to CENTRE@CLEF2019. The resulting evaluation outcomes match those which were achieved during our participation at the CENTRE lab.

3.5 Limitations

At the current state, rankings for replicated runs are possible. This means, only the NYT and Robust corpora will be processed correctly by our Docker image. In the future, support of the Washington Post corpus can be integrated for the further investigation of reproducibility. Going a step further, the complete data preparation step could be extended to more general compatibility with other test collections or generic text data. At the moment, the routines are highly adjusted to the given workflow by Grossman and Cormack and the underlying data.

Even though the data preparation is parallelized, it takes a while to index the corpora. In order to reduce indexing time, the text preprocessing can be omitted, leading to compromises between execution time and evaluation measures.

Predictions of the logistic regression classifier are used for scoring documents. Currently, the corresponding tfidf-features are stored in a Python shelf and will be read out sequentially for classification. This step should be parallelized for the reduction of ranking time.

4 CONCLUSION

We contributed our CENTRE@CLEF19 submissions to the Docker infrastructure initiated by the OSIRRC workshop. Our original code submission reimplemented automatic routing runs as they were described by Grossman and Cormack [7]. In the course of our CENTRE participation, we investigated the replicability and reproducibility of the given procedure. We focus on contributing replicated runs to the Docker infrastructure. CENTRE defines replicability by using the original test collection in combination with a different setup. Thus our reimplementations rank documents of the NYT corpus by using Robust corpora for the training of topic models.

Adaptions to the Docker infrastructure were realizable with little effort. We adjusted the workflow with regard to the given hooks. The resulting runs exactly match those which were replicated in the context of our CENTRE participation. Due to the encapsulation into Docker images, less configuration effort is required and our experimental environment can be exactly reconstructed. Required components are taken from existing Docker images and Python packages.

Starting points for future improvements were elaborated in the previous section. Investigations on reproducibility can be made possible by integrating the Washington Post corpus into our workflow. In this context, the support of other test collections might also be interesting. Parallelizing classifications can reduce execution time of the ranking. An archived version of our submitted Docker image is available at Zenodo [3].

REFERENCES

[1] Timothy G. Armstrong, Alistair Moffat, William Webber, and Justin Zobel. 2009. Has Adhoc Retrieval Improved Since 1994?. In *Proceedings of the 32Nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '09)*. ACM, New York, NY, USA, 692–693. https://doi.org/10.1145/1571941.1572081

[2] Timothy G. Armstrong, Alistair Moffat, William Webber, and Justin Zobel. 2009. Improvements That Don'T Add Up: Ad-hoc Retrieval Results Since 1998. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM '09)*. ACM, New York, NY, USA, 601–610. https://doi.org/10.1145/1645953.1646031

[3] Timo Breuer and Philipp Schaer. 2019. osirrc/irc-centre2019-docker: OSIRRC @ SIGIR 2019 Docker Image for IRC-CENTRE2019. https://doi.org/10.5281/zenodo.3245439

[4] Timo Breuer and Philipp Schaer. 2019. Replicability and Reproducibility of Automatic Routing Runs. In *Working Notes of CLEF 2019 - Conference and Labs of the Evaluation Forum (CEUR Workshop Proceedings)*. CEUR-WS.org. (accepted).

[5] Matt Crane. 2018. Questionable Answers in Question Answering Research: Reproducibility and Variability of Published Results. *TACL* 6 (2018), 241–252. https://transacl.org/ojs/index.php/tacl/article/view/1299

[6] Nicola Ferro, Norbert Fuhr, Maria Maistro, Tetsuya Sakai, and Ian Soboroff. 2019. CENTRE@CLEF 2019. In *Advances in Information Retrieval - 41st European Conference on IR Research, ECIR 2019, Cologne, Germany, April 14-18, 2019, Proceedings, Part II (Lecture Notes in Computer Science)*, Leif Azzopardi, Benno Stein, Norbert Fuhr, Philipp Mayr, Claudia Hauff, and Djoerd Hiemstra (Eds.), Vol. 11438. Springer, 283–290. https://doi.org/10.1007/978-3-030-15719-7_38

[7] Maura R. Grossman and Gordon V. Cormack. 2017. MRG_UWaterloo and WaterlooCormack Participation in the TREC 2017 Common Core Track. In *Proceedings of The Twenty-Sixth Text REtrieval Conference, TREC 2017, Gaithersburg, Maryland, USA, November 15-17, 2017*, Ellen M. Voorhees and Angela Ellis (Eds.), Vol. Special Publication 500-324. National Institute of Standards and Technology (NIST). https://trec.nist.gov/pubs/trec26/papers/MRG_UWaterloo-CC.pdf

[8] Frank Hopfgartner, Allan Hanbury, Henning Müller, Ivan Eggel, Krisztian Balog, Torben Brodt, Gordon V. Cormack, Jimmy Lin, Jayashree Kalpathy-Cramer, Noriko Kando, Makoto P. Kato, Anastasia Krithara, Tim Gollub, Martin Potthast, Evelyne Viegas, and Simon Mercer. 2018. Evaluation-as-a-Service for the Computational Sciences: Overview and Outlook. *J. Data and Information Quality* 10, 4, Article 15 (Oct. 2018), 32 pages. https://doi.org/10.1145/3239570

[9] Jimmy Lin. 2019. The Neural Hype and Comparisons Against Weak Baselines. *SIGIR Forum* 52, 2 (Jan. 2019), 40–51. https://doi.org/10.1145/3308774.3308781

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[11] Peilin Yang and Hui Fang. 2016. A Reproducibility Study of Information Retrieval Models. In *Proceedings of the 2016 ACM International Conference on the Theory of Information Retrieval (ICTIR '16)*. ACM, New York, NY, USA, 77–86. https://doi.org/10.1145/2970398.2970415

[12] Wei Yang, Kuang Lu, Peilin Yang, and Jimmy Lin. 2019. Critically Examining the "Neural Hype": Weak Baselines and the Additivity of Effectiveness Gains from Neural Ranking Models. *CoRR* abs/1904.09171 (2019). arXiv:1904.09171 http://arxiv.org/abs/1904.09171

[13] Ruifan Yu, Yuhao Xie, and Jimmy Lin. 2019. Simple Techniques for Cross-Collection Relevance Feedback. In *Advances in Information Retrieval - 41st European Conference on IR Research, ECIR 2019, Cologne, Germany, April 14-18, 2019, Proceedings, Part I (Lecture Notes in Computer Science)*, Leif Azzopardi, Benno Stein, Norbert Fuhr, Philipp Mayr, Claudia Hauff, and Djoerd Hiemstra (Eds.), Vol. 11437. Springer, 397–409. https://doi.org/10.1007/978-3-030-15712-8_26