BM25 Pseudo Relevance Feedback Using Anserini at Waseda University

Zhaohao Zeng
Waseda University
Tokyo, Japan
zhaohao@fuji.waseda.jp

Tetsuya Sakai
Waseda University
Tokyo, Japan
tetsuyasakai@acm.org

ABSTRACT

We built a Docker image for BM25PRF (BM25 with Pseudo Relevance Feedback) retrieval model with Anserini. Also, grid search is provided in the Docker image for parameter tuning. Experimental results suggest that BM25PRF with default parameters outperforms vanilla BM25 on `robust04`, but tuning parameters on 49 topics of `robust04` did not further improve its effectiveness.

Image Source: github.com/osirrc/anserini-bm25prf-docker
Docker Hub: hub.docker.com/r/osirrc2019/anserini-bm25prf

1 OVERVIEW

BM25 has been widely used as a baseline model for text retrieval tasks. However, some researches only implement the vanilla form of BM25 without query expansion and parameter tuning. As a result, the performance of BM25 may be underestimated [2]. In our Docker image, we implemented BM25PRF [3], which utilises Pesudo Relevance Feedback (PRF) to expand queries for BM25. We also implemented parameter tuning in the Docker image because we believe how to obtain the optimised parameters is also an important part of reproducible research. We built BM25PRF and parameter tuning with Anserini [5], a toolkit built on top of Lucene for replicable IR research.

2 RETRIEVAL MODELS

Given a query q, BM25PRF [3] ranks the collection with the classic BM25 first, and then extracts m terms that have high Offer Weights (OW) from the top R ranked documents to expand the query. To calculate the Offer Weight for a term t_i, its Relevance Weight (RW) needs to be calculated first as follows:

$$RW(t_i) = \log \frac{(r+0.5)(N-n-R+r+0.5)}{(n-r+0.5)(R-r+0.5)} \tag{1}$$

where r is the Document Frequency (DF) of term t_i in the top R documents, n is the DF of t_i in the whole collection, and N is the number of documents in the collection. Then, the Offer Weight is

$$OW(t_i) = RW(t_i) \cdot r \tag{2}$$

However, in practice a term that has high r will tend to have high OW, so some common words (e.g., be, been) may have high OW and be selected as expansion terms. Since the common words are not informative, they may be not helpful for ranking. Thus, in our Docker image, logarithm is applied to r in the OW calculation to alleviate this problem according to Sakai and Robertson [4]:

$$OW(t_i) = RW(t_i) \cdot \log(r) \tag{3}$$

After query expansion, the expanded terms will be used for the second search using a BM25 variant: for one term t_i and one document d_j, the score $s(t_i, d_j)$ is calculated as follows:

$$s(t_i, d_j) = \begin{cases} s'(t_i, d_j) & \text{if } t_i \in q \\ w \cdot s'(t_i, d_j) & \text{else} \end{cases} \tag{4}$$

$$s'(t_i, d_j) = \frac{RW(t_i) \cdot TF(t_i, d_j) \cdot (K1+1)}{K1 \cdot ((1-b) + (b \cdot (NDL(d_j)))) + TF(t_i, d_j)} \tag{5}$$

where $TF(t_i, d_j)$ is the term frequency of term t_i in d_j, $NDL(d_j)$ is the normalised document length of d_j: $NDL(d_j) = \frac{N \cdot |d_j|}{\sum_k^N |d_k|}$, w is the weight of new terms, and $K1$ and b are the hyper-parameters of BM25. All the tunable hyper-parameters are shown in Table 1.

Table 1: Tunable parameters of BM25PRF and their search spaces in the parameter tuning script.

	Search space	Default	Note
$K1$	0.1 - 0.9 step=0.1	0.9	$K1$ of the first search
b	0.1 - 0.9 step=0.1	0.4	b of the first search
$K1_{prf}$	0.1 - 0.9 step=0.1	0.9	$K1$ of the second search
b_{prf}	0.1 - 0.9 step=0.1	0.4	b of the second search
R	$\{5, 10, 20\}$	10	num of relevant docs
w	$\{0.1, 0.2, 0.5, 1\}$	0.2	weight of new terms
m	$\{0, 5, 10, 20, 40\}$	20	num of new terms

3 TECHNICAL DESIGN

Supported Collections:
`robust04`

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

Since the BM25PRF retrieval model is not included in the original Anserini library, we forked its repository and added two JAVA classes: *BM25PRFSimilarity* and *BM25PRFReRanker* by extending the *Similarity* Class and the *ReRanker* Class, respectively. Thus, the implemented BM25PRF can be utilised on any collections supported by Anserini, though we only tested it on the `robust04` collection in this paper. Python scripts are used as hooks to run the necessary commands (e.g., `index` and `search`) via jig.[1] Jig is a tool provided

[1]https://github.com/osirrc/jig

by the OSIRRC organisers to operate the Docker images which follow the OSIRRC specification.

Grid search is also provided in the Docker image for parameter tuning, and can be executed using the `train` hook of jig. It performs search and evaluation for every combination of parameters specified. To reduce the search space of grid search, our tuning process consists of two steps. First, it performs search on a validation set using the original BM25 to find the optimal parameters of it (i.e., $K1$ and b) based on Mean P@20. The $K1$ and b are the parameters for the initial iteration of BM25PRF, so precision may be important for extracting effective expansion terms. Then, the tuned $K1$ and b are frozen, and the other parameters of BM25PRF (i.e., $K1_{prf}$, b_{prf}, R, w, and m) are tuned on the validation set based on MAP.

4 RESULTS

The parameter tuning was performed on 49 topics[2] of `robust04`, and the tuned parameters are shown in Table 2. As shown in Table 3, the BM25PRF outperforms the vanilla BM25, but the tuned hyper-parameters do not improve BM25PRF's performance on `robust04`. This may be because the validation set used for tuning is too small, and the parameters have been overfitted. Since the goal of this study is about using Docker for reproducible IR research instead of demonstrating the effectiveness of BM25PRF and grid search, we do not further discuss the performance in this paper.

Table 2: Tuned hyper-parameters.

	$K1$	b	$K1_{prf}$	b_{prf}	m	R	w
Tuned Value	0.9	0.2	0.9	0.6	40	10	0.1

Table 3: BM25PRF performance on robust04.

Model	MAP	P@30
BM25 [1]	0.2531	0.3102
BM25PRF (default parameters)	0.2928	0.3438
BM25PRF (tuned parameters)	0.2916	0.3396

5 OSIRRC EXPERIENCE

Docker has been widely used in industry for delivering software, but we found that using Docker to manage an experimental environment has advantages for research usage as well. First, it is easier to configure environments with Docker than with a bare-metal server, especially for deep learning scenarios where a lot of packages (e.g., GPU driver, CUDA and cuDNN) need to be installed. Moreover, Docker makes experiments more trackable. Research code is usually messy, lacks documentation, and may need a lot of changes during the experiment, so even the author may have difficulty to remember the whole change log. Since each Docker tag is an executable archive, it provides a kind of version control on executables. Moreover, if the docker images follow some common specification like the ones we build for OSIRRC, running the research codes we wrote several months ago is not a nightmare anymore.

However, the biggest obstacle we faced during the development for OSIRRC is that it is more difficult to debug with Docker and jig. For example, there is no simple approach to setting a debugger into the Docker container when it was launched by jig. Furthermore, current jig assumes that the index files are built inside the Docker container and commit the Docker and built index as a new image, which means that the index needs to be built again after modifying the source code. While it is not a serious problem for small collections like `robust04`, it may take too much time for large collections. To solve this problem, we think jig should allow users to mount external index when launching the `search` hook. Although mounting external data into a Docker container is a standard action when using Docker's command line tools directly, but OSIRRC expects Docker images to be operated through jig, which currently does not provide such a feature.

[2]The topics ids of the validation set are provided by the OSIRRC organisers in jig: https://github.com/osirrc/jig/tree/master/sample_training_validation_query_ids

REFERENCES

[1] Ryan Clancy and Jimmy Lin. 2019. osirrc/anserini-docker: OSIRRC @ SIGIR 2019 Docker Image for Anserini. https://doi.org/10.5281/zenodo.3246820

[2] Jimmy Lin. 2019. The Neural Hype and Comparisons Against Weak Baselines. In *ACM SIGIR Forum*, Vol. 52. ACM, 40–51.

[3] Stephen E. Robertson and Karen Spärck Jones. 1994. *Simple, proven approaches to text retrieval.* Technical Report 356. Computer Laboratory University of Cambridge.

[4] Tetsuya Sakai and Stephen E Robertson. 2002. Relative and absolute term selection criteria: a comparative study for English and Japanese IR. In *SIGIR 2002.* 411–412.

[5] P. Yang, H. Fang, and J. Lin. 2017. Anserini: Enabling the Use of Lucene for Information Retrieval Research. In *SIGIR 2017.* 1253–1256.