Pseudo-Relevance Feedback with Dense Retrievers in Pyserini

Hang Li*
University of Queensland
Brisbane, Australia
hang.li@uq.edu.au

Shengyao Zhuang*
University of Queensland
Brisbane, Australia
s.zhuang@uq.edu.au

Xueguang Ma
University of Waterloo
Waterloo, Canada
x93ma@uwaterloo.ca

Jimmy Lin
University of Waterloo
Waterloo, Canada
jimmylin@uwaterloo.ca

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

ABSTRACT

Transformer-based Dense Retrievers (DRs) are attracting extensive attention because of their effectiveness paired with high efficiency. In this context, few Pseudo-Relevance Feedback (PRF) methods applied to DRs have emerged. However, the absence of a general framework for performing PRF with DRs has made the empirical evaluation, comparison and reproduction of these methods challenging and time-consuming, especially across different DR models developed by different teams of researchers.

To tackle this and speed up research into PRF methods for DRs, we showcase a new PRF framework that we implemented as a feature in *Pyserini* – an easy-to-use Python Information Retrieval toolkit. In particular, we leverage Pyserini's DR framework and expand it with a PRF framework that abstracts the PRF process away from the specific DR model used. This new functionality in Pyserini allows to easily experiment with PRF methods across different DR models and datasets. Our framework comes with a number of recently proposed PRF methods built into it. Experiments within our framework show that this new PRF feature improves the effectiveness of the DR models currently available in Pyserini.

KEYWORDS

Pyserini, Dense Retriever, Pseudo-Relevance Feedback

ACM Reference Format:
Hang Li, Shengyao Zhuang, Xueguang Ma, Jimmy Lin, and Guido Zuccon. 2022. Pseudo-Relevance Feedback with Dense Retrievers in Pyserini. In *Australasian Document Computing Symposium (ADCS '22), December 15–16, 2022, Adelaide, SA, Australia.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3572960.3572982

1 INTRODUCTION

Recent advances in Information Retrieval have seen the introduction of Dense Retrievers (DRs), where a transformer-based deep language model (e.g., BERT [3]) is used to separately encode queries and documents in low dimensional embeddings (dense vectors). These embeddings are then used as representation for retrieval via vector similarity search. Compared to cross-encoder-based rankers, such as monoBERT [17], DRs provide at times lower effectiveness, but also a consistently much lower query latency, making their use into actual, real-time retrieval applications more feasible than for cross-encoders. To support research on DRs, the Pyserini toolkit has recently added DRs among the methods it makes available for retrieval [11], providing easy-to-use dense vector search APIs. In addition the toolkit allows easy integration and reproduction of newly proposed DR methods by simply exploiting the model checkpoint of the new DR for generating a dense index, which is then added to the Pyserini DR model base.

Pseudo-Relevance Feedback (PRF) methods such as Rocchio [19] and RM3 [15] are known to often improve the effectiveness of traditional bag-of-words (BoW) retrievers such as BM25. On the other hand, recent pre-trained transformer-based dense retrievers (DRs) have shown higher effectiveness than BoW retrievers, while at the same time achieving similar query latency [12]. It is then natural to consider performing PRF in the context of DRs [8–10, 20, 21, 23]. With the field moving forward, challenges and difficulties have become clear: For researchers devising new PRF methods, investigating the effectiveness of their method in the context of DR is hard. This is because each DR method is developed, released, and maintained by different groups, with different practices and code standards. Hence, in order to test the effectiveness of a new PRF method, extra efforts will be paid on integrating DR models and code written by different groups.

In this context, we present a new PRF framework that implemented as a new feature in Pyserini [11] – a popular DR Python toolkit. Pyserini has integrated different DR models and provided easy-to-use dense vector search APIs. For newly invented DR models, an access to the model checkpoint is sufficient to generate a new dense index and add it into the Pyserini DR model base. Given these advantages provided by Pyserini in support of DR research, we believe it is necessary and beneficial to develop a PRF feature for Pyserini which allows a PRF method designed for DRs to be easily tested across different DRs implemented in Pyserini, or vice versa, which allows a new DR model to be easily tested across different PRF methods.

Figure 1 illustrates the proposed PRF framework for DRs in Pyserini. The framework comprises two rounds of DR search. In

*Both authors contributed equally to the paper.

ADCS '22, December 15–16, 2022, Adelaide, SA, Australia
© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0021-7/22/12…$15.00
https://doi.org/10.1145/3572960.3572982

the first round, the `QueryEncoder` class first encodes the query text into a vector. Then, the `SimpleDenseSearcher` class performs a standard dense vector similarity search and returns the top-k nearest neighbors (documents/passages). In the second round, the `DenseVectorPrf` class takes the returned documents as input and creates a new query vector based on the selected PRF method. Finally, the `SimpleDenseSearcher` performs another search but with the new query vector, producing as output the final ranking.

To test and validate our PRF framework within Pyserini, in this paper we describe our implementation of two DR-based PRF methods, *Vector-Based PRF with Average* and *Vector-Based PRF with Rocchio*, based on the original techniques by Li et al. [9]. We first confirm that it is possible to reproduce these methods and the related results from the original paper within our Pyserini PRF framework. In particular, we test the effectiveness of the PRF methods with the DR models and datasets that are made available off-the-shelf by Pyserini, including some that were not considered by Li et al.. Our results show that the effectiveness of all DR models currently in Pyserini is improved when using the newly implemented PRF feature. We then note that this finding complements the results originally reported by Li et al., and this was easily obtainable largely because Pyserini already integrates a wide array of DR models and common datasets. Finally, our PRF framework can be further expanded by implementing other PRF methods designed for DRs.

2 DENSE RETRIEVER PRF

Next, we briefly describe the two DR-based PRF methods we implemented [9]; we refer the reader to Li et al. [9] for details.

Vector-Based PRF with Average. The vector of the original query $V(Q_{original})$ and the vectors of the top-k pseudo feedback passages $V(p_1)$, ..., $V(p_k)$ are averaged to form the new query vector $V_{Q_{new}}$:

$$V_{Q_{new}} = Avg(V(Q_{original}), V(p_1), ..., V(p_k)) \quad (1)$$

By doing so, the signal provided by the original query is treated as being equivalent to that from any of the top-k feedback passages.

Vector-Based PRF with Rocchio. This method is an adaptation of the original Rocchio method for relevance feedback [19] to the context of deep dense language models. Here, the original query vector is modified by moving it towards the average of the top-k feedback passage vectors by assigning different weights to query and (the combination of) feedback passages, thus controlling the contribution of each component towards the final score. Note that negative feedback is omitted, but it is possible to model this too following the original formulation from Rocchio [19]. Formally, the new query vector is obtained as:

$$V_{Q_{new}} = \alpha * V(Q_{original}) + \beta * Avg(V(p_1), ..., V(p_k)) \quad (2)$$

where α controls the weight assigned to the original query vector and β the weight assigned to the PRF signal.

3 DR-BASED PRF FRAMEWORK IN PYSERINI

The PRF framework for DRs we integrated into Pyserini consists of three core modules: the `SimpleDenseSearcher`, the `QueryEncoder`, and the `DenseVectorPrf`. They are respectively responsible for encoding the query text into vectors based on different DR models (`QueryEncoder`), retrieving the PRF candidates from the index (`SimpleDenseSearcher`), and performing either Average PRF or Rocchio PRF (`DenseVectorPrf`). These are implemented as abstract classes, making it easy to extend the framework to other PRF methods and DR models. Next, we describe the implementation of these classes and provide examples of how to run DR-based PRF within Pyserini.

Code Sample 1: Average PRF implementation in Pyserini.

```
        # get all the PRF candidate vectors
        all_candidate_embs = [item.vectors for item in prf_candidates]

        # stack query vector and PRF candidate vectors
        # then get mean as new query vector
        new_emb_qs = np.mean(
        np.vstack((emb_qs[0], all_candidate_embs)), axis=0)

        # return new query vector as numpy array
        new_emb_qs = np.array([new_emb_qs]).astype('float32')
```

Code Sample 2: Rocchio PRF implementation in Pyserini.

```
        # get all the PRF candidate vectors
        all_candidate_embs = [item.vectors for item in prf_candidates]

        # get weighted mean of candidate vectors
        weighted_mean_doc_embs = rocchio_beta * np.mean(
        all_candidate_embs, axis=0)

        # get weighted query vector
        weighted_query_embs = rocchio_alpha * emb_qs[0]

        # get sum of the weighted vectors
        new_emb_q = np.sum(
        np.vstack((weighted_query_embs, weighted_mean_doc_embs)), axis=0)

        # return new query vector as numpy array
        new_emb_q = np.array([new_emb_q]).astype('float32')
```

3.1 Modules and Configuration

The PRF module in Pyserini consists of three main components, which covers the full process of encoding query text, retrieving PRF candidates, performing PRF, generates the new query, and final retrieval.

Figure 1 illustrates the interaction between the three main modules in our PRF framework. The `QueryEncoder` module is responsible for encoding the query text into vectors. The `SimpleDenseSearcher` module is responsible for the retrieval of the candidate documents to be sent for PRF, and for the final retrieval using the newly generated PRF query. The `DenseVectorPrf` module takes the top-k results produced by the first round of `SimpleDenseSearcher` and it produces the vector for the new PRF query: how this is done depends on the specific PRF method used.

`QueryEncoder`. The `QueryEncoder` module is an abstract class in Pyserini, which is responsible for encoding the raw query text into vectors for later retrieval of PRF candidates and generation of the new query vector. Currently the `QueryEncoder` module supports encoding with ANCE [22], DPR [7], TCT-ColBERT V1 [13], TCT-ColBERT V2 [14], DistillBERT KD [4], DistillBERT Balanced [5], and SBERT [18]. Different DR models can be selected either programmatically or via command line arguments.

`SimpleDenseSearcher`. The `SimpleDenseSearcher` module is also an abstract class in Pyserini, which, in the first round of retrieval,

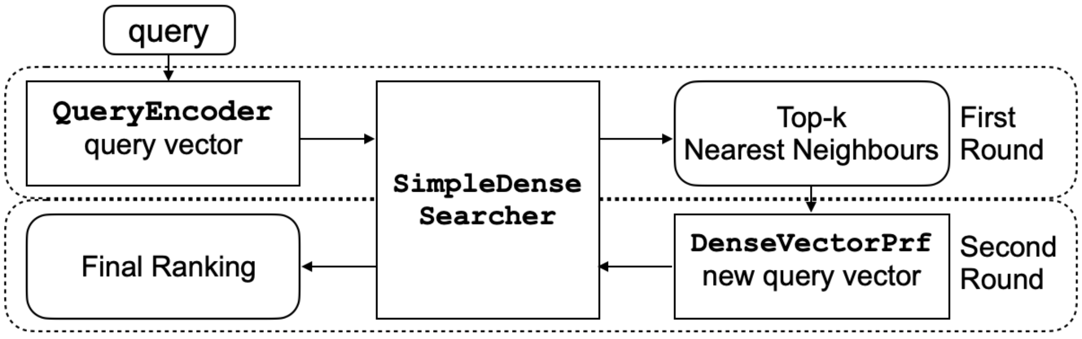

Figure 1: The proposed Pyserini Dense Retriever-based Pseudo-Relevance Feedback framework.

is responsible for retrieving the top-k (PRF depth) candidates to be used as PRF signal. To do this, it uses the query encoded by `QueryEncoder`. In the second round of retrieval, this module is responsible for retrieving the final results using the new query generated by the `DenseVectorPrf` module. `SimpleDenseSearcher` requires a Faiss [6] index as one of the inputs and it performs the pairwise dot product between the query and document representations, for each document in the index. This module can also work with any of the DRs already supported by Pyserini.

`DenseVectorPrf`. The `DenseVectorPrf` module, also an abstract class, is responsible for performing the actual PRF process and generate the new query vector representation. Currently, the Average PRF and Rocchio PRF approaches [9] are supported by the `DenseVectorPrf` module. For the Average approach (Eq. 1), we first stack the query vector and the PRF candidate vectors. Then we compute the average of the stacked vectors and obtain the final vector representing the new PRF query, as shown in Code Sample 1. For Rocchio, we first stack the PRF candidate vectors and compute their mean. Then we linearly interpolate the query vector and the average PRF candidate vector, using the interpolation parameters α and β (Eq. 2), as shown in Code Sample 2.

Code Sample 3: Usage of Average PRF in Pyserini with batch processing.

```
$ python -m pyserini.dsearch --topics dl19-passage \
    --index msmarco-passage-ance-bf \
    --encoder castorini/ance-msmarco-passage \
    --output test_ance_avg_prf3_batch.res \
    --batch-size 64 \
    --threads 12 \
    --prf-depth 3 \
    --prf-method avg
```

Code Sample 4: Usage of Rocchio PRF in Pyserini with batch processing.

```
$ python -m pyserini.dsearch --topics dl19-passage \
    --index msmarco-passage-ance-bf \
    --encoder castorini/ance-msmarco-passage \
    --output test_ance_rocchio_prf5_batch.res \
    --batch-size 64 \
    --threads 12 \
    --prf-depth 5 \
    --prf-method rocchio \
    --rocchio-alpha 0.4 \
    --rocchio-beta 0.6
```

3.2 Usage

Our PRF module is easy to use, as illustrated by Code Sample 3, which executes Average PRF on top of ANCE using batch retrieval mode, and Code Sample 4, which executes Rocchio PRF on top of ANCE. The `dsearch` command accepts the following inputs:

- `--topics`: the path to the raw query text file;
- `--index`: the path to the pre-built Faiss index;
- `--encoder`: the DR model's name, or the path to a local model checkpoint;
- `--output`: the path to where the result file should be saved;
- `--batch-size`: the number of queries to be batched for group processing;
- `--threads`: the maximum number of threads to use;
- `--prf-depth`: the number of candidate vectors for PRF;
- `--prf-method`: specifies the PRF method, which currently can either be `avg` or `rocchio`;
- `--rocchio-alpha`: the value of the parameter α in Rocchio PRF (default: $\alpha = 0.9$);
- `--rocchio-beta`: the value of the parameter β in Rocchio PRF (default: $\beta = 0.1$);

Note, if `--batch-size` and `--threads` are both 1, then the PRF module executes queries in sequential order; if `--batch-size` is > 1, then the PRF module executes queries in batch mode.

Because Pyserini ships already with numerous pre-built dense indexes, along with raw and encoded queries for common datasets used in research, the application of these PRF methods to such dataset is fairly trivial: one just needs to modify the input parameters accordingly to the chosen dataset and the required queries, dense indexes, and encoded queries are automatically downloaded. This removes the need for the individual researcher to build the dense index, or even identify its location among other's repositories – thus ultimately accelerating the reproduction of experiments.

Table 1: Comparison between DR with and without PRF on two popular TREC DL datasets. All PRF parameters are not tuned: PRF depth =3; for Rocchio $\alpha = 0.4$, $\beta = 0.6$. Bold indicates the best results w.r.t. each base model.

		TREC DL 2019			TREC DL 2020		
Model	PRF	MAP	nDCG@100	Recall@1000	MAP	nDCG@100	Recall@1000
BM25	-	0.3773	0.5018	0.7389	0.2856	0.4902	0.7863
BM25	RM3	0.4270	0.5286	**0.7882**	0.3019	0.5077	**0.8217**
BM25 + BERT-base	-	**0.4827**	**0.6426**	0.7389	**0.4926**	**0.6473**	0.7863
ANCE [22]	Original	0.3710	0.5540	0.7554	0.4076	0.5679	0.7764
	Average PRF	**0.4247**	**0.5937**	0.7739	**0.4325**	0.5793	0.7909
	Rocchio	0.4211	0.5928	**0.7825**	0.4315	**0.5800**	**0.7957**
TCT-ColBERT V1 [13]	Original	0.3906	0.5730	0.7916	0.4290	0.5826	0.8181
	Average PRF	0.4336	0.6119	0.8230	**0.4725**	**0.6101**	**0.8667**
	Rocchio	**0.4463**	**0.6143**	**0.8393**	0.4625	0.6056	0.8576
TCT-ColBERT V2 [14]	Original	0.4269	0.6129	0.8342	0.4717	0.6200	0.8407
	Average PRF	**0.4766**	**0.6487**	**0.8574**	0.4701	0.6209	0.8739
	Rocchio	0.4709	0.6435	0.8496	**0.4819**	**0.6324**	**0.8760**
DistillBERT KD [4]	Original	0.3759	0.5765	0.6853	0.3909	0.5728	0.6893
	Average PRF	0.4362	**0.6217**	0.7180	0.3955	0.5755	**0.7279**
	Rocchio	**0.4378**	0.6189	**0.7291**	**0.3990**	**0.5760**	0.7222
DistillBERT Balanced [5]	Original	0.4761	0.6360	0.7826	0.4755	0.6346	0.8009
	Average PRF	0.5057	0.6526	0.8054	**0.4873**	0.6449	**0.8392**
	Rocchio	**0.5249**	**0.6684**	**0.8352**	0.4846	**0.6470**	0.8262
SBERT [18]	Original	0.4060	0.5985	0.7872	0.4124	0.5734	0.7937
	Average PRF	0.4354	**0.6149**	0.7937	0.4258	0.5781	0.8169
	Rocchio	**0.4371**	**0.6149**	**0.7941**	**0.4342**	**0.5851**	**0.8226**
ADORE [24]	Original	0.4188	0.5946	0.7759	0.4418	0.5949	0.8151
	Average PRF	0.4672	**0.6263**	0.7890	0.4706	0.6176	0.8323
	Rocchio PRF	**0.4760**	0.6193	**0.8251**	**0.4760**	**0.6193**	0.8251

4 EMPIRICAL VALIDATION

4.1 Experimental Setup

To validate the correct implementation of the DR-based PRF methods in Pyserini, we replicate the experiments of Li et al. [9] performed on the TREC DL 2019 [1] and TREC DL 2020 [2] collections. Both use the MS MARCO Passage Ranking [16] dataset. As discussed in Li et al. [9], the original MS MARCO passage ranking dev queries and judgements are not suitable for evaluating PRF methods as there is only one judged relevant passage per query on average. Furthermore, often this relevant passage is part of the PRF signal. Because of this, we are not include results on this dataset in the main part of the paper. Nevertheless, we evaluate the methods on the dev queries of the MS MARCO dataset for completeness and report these results in Appendix 5.

Training is not required for our PRF methods: it can directly be applied to any dense retriever. We considere the following DR models, which are readily available in Pyserini: ANCE [22], TCT-COlBERT V1 [13], TCT-COlBERT V2 [14], DistillBERT KD [4], DistillBERT Balanced [5], SBERT [18] and ADORE [24]. Of these, Li et al. [9] only experimented with ANCE.

Any of these models can be directly used with the two classes `SimpleDenseSearcher` and `DenseVectorPrf` within Pyserini: each individual DR uses a different on-the-fly `QueryEncoder` within `SimpleDenseSearcher` to encode the query text at inference time. PRF parameters are not tuned and are shared across all DRs models. For comparison as baselines, we also include BM25, BM25+RM3, and BM25 + BERT-base [17] along with the DR results without PRF.

Effectiveness of all methods is measured with MAP, nDCG@100, and Recall@1000, according to common TREC DL practice. The experiments were executed on a 2019 MacBook Pro with 2.4 GHz 8-Core Intel Core i9 CPU and 64GB memory; no GPU was used at inference time, the batch size was set to 64, with 12 threads. The dense indexes were pre-built offline using Faiss [6].

4.2 Results

Experiment results for TREC 2019 and 2020 are reported in Table 1. Results for DR-based PRF methods can be replicated using Code Samples 3 and 4 by replacing the parameters corresponding to topics and encoders.

We first examine the results obtained using the ANCE dense retriever, as this setting corresponds to that studied by Li et al. [9] in the context of PRF. We confirm we were able to reproduce these results with our new Pyserini dense PRF feature – and that the use of PRF improves ANCE's effectiveness. In addition, we tested Li et al. [9]'s PRF methods using our new Pyserini PRF feature and the other DRs made available by the toolkit. Similar to ANCE, we found that PRF improves the DRs effectiveness: this is in both TREC DL datasets, except Average with TCT-ColBERT V2 on TREC DL 2020 MAP. Recall@1000 was improved the most by PRF: this is desirable, as commonly DRs form the first stage of a multi-stage retrieval pipeline – and high recall is required for a first stage method. Remarkably, (1) these results were obtaining without tuning the PRF parameters, (2) PRF improved the effectiveness also of strong DRs such as DistillBERT Balanced, which, with PRF, can now outperform the BM25 + BERT-base two-stage reranker across all measures on TREC DL 2019, and for Recall@1000 on TREC DL 2020.

As for query latency, with PRF the time consumed for a single query (e.g., ANCE with PRF: 395ms) is slightly lower than twice the original retrieval time (ANCE: 209ms). The second round of retrieval (i.e. the PRF) is slightly faster than the first because it does not need to perform the query encoding: the query generated by the PRF is a vector. We note this fairly low latency (comparable, if not lower than BM25+RM3 for long feedback inputs) makes this PRF framework efficient and effective – and it can serve as a "free boost" to all DR models. Although we only demonstrate this PRF framework with default parameter settings, researchers can easily conduct grid search, or other hyper-parameter optimisation methods, to identify the best settings for their context; this can be achieved by simply modify the hyper-parameter values in the command lines.

5 CONCLUSIONS

Reproducing the results reported in research papers is often a hard task; an even harder task is integrating existing methods with newly proposed ones. In this paper we have provided a new feature for Pyserini that aims to make it easier to reproduce results and integrate new methods in the context of recent advances in Pseudo-Relevance Feedback with Dense Retrievers. We have further demonstrated this new feature by implementing two recently proposed DR-based PRF methods [9] and reproducing their results. In addition, we exploited the extensive set of DR methods and datasets already available in Pyserini to further extend the empirical evaluation of the PRF methods to DRs and datasets not studied in the original work of Li et al. [9]. The empirical results provide a novel finding: these PRF methods deliver improved effectiveness over non-PRF runs regardless of the DR model used or the dataset studied. This means that the PRF feature showcased here provides a "free boost" to any DR – just like RM3 often does for BM25. Finally, the implemented PRF framework is easily adaptable and extendable, allowing others to integrate and study alternative DR-based PRF methods, thus lowering the barriers for reproducing existing and investigating new methods in this context.

Acknowledgements.

This research was supported in part by the Grain Research and Development Corporation project AgAsk (UOQ2003-009RTX) and the Natural Sciences and Engineering Research Council (NSERC) of Canada. Computational resources were provided by The University of Queensland, Compute Ontario and Compute Canada.

REFERENCES

[1] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, and Ellen M Voorhees. 2020. Overview of the TREC 2019 Deep Learning Track. In *TREC*.
[2] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, and Ellen M Voorhees. 2021. Overview of the TREC 2020 Deep Learning Track. In *TREC*.
[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805* (2018).
[4] Sebastian Hofstätter, Sophia Althammer, Michael Schröder, Mete Sertkan, and Allan Hanbury. 2020. Improving efficient neural ranking models with cross-architecture knowledge distillation. *arXiv preprint arXiv:2010.02666* (2020).
[5] Sebastian Hofstätter, Sheng-Chieh Lin, Jheng-Hong Yang, Jimmy Lin, and Allan Hanbury. 2021. Efficiently Teaching an Effective Dense Retriever with Balanced Topic Aware Sampling. In *SIGIR*.
[6] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data* (2019).
[7] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In *EMNLP*.
[8] Hang Li, Ahmed Mourad, Bevan Koopman, and Guido Zuccon. 2022. How Does Feedback Signal Quality Impact Effectiveness of Pseudo Relevance Feedback for Passage Retrieval. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '22)*. Association for Computing Machinery, New York, NY, USA, 2154–2158. https://doi.org/10.1145/3477495.3531822
[9] Hang Li, Ahmed Mourad, Shengyao Zhuang, Bevan Koopman, and Guido Zuccon. 2021. Pseudo Relevance Feedback with Deep Language Models and Dense Retrievers: Successes and Pitfalls. *arXiv preprint arXiv:2108.11044* (2021).
[10] Hang Li, Shengyao Zhuang, Ahmed Mourad, Xueguang Ma, Jimmy Lin, and Guido Zuccon. 2022. Improving Query Representations for Dense Retrieval with Pseudo Relevance Feedback: A Reproducibility Study. In *European Conference on Information Retrieval*. Springer, 599–612.
[11] Jimmy Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, and Rodrigo Nogueira. 2021. Pyserini: A Python Toolkit for Reproducible Information Retrieval Research with Sparse and Dense Representations. In *SIGIR*.
[12] Jimmy Lin, Rodrigo Nogueira, and Andrew Yates. 2020. Pretrained transformers for text ranking: Bert and beyond. *arXiv preprint arXiv:2010.06467* (2020).
[13] Sheng-Chieh Lin, Jheng-Hong Yang, and Jimmy Lin. 2020. Distilling dense representations for ranking using tightly-coupled teachers. *arXiv preprint arXiv:2010.11386* (2020).
[14] Sheng-Chieh Lin, Jheng-Hong Yang, and Jimmy Lin. 2021. In-batch negatives for knowledge distillation with tightly-coupled teachers for dense retrieval. In *RepL4NLP-2021*. 163–173.
[15] Yuanhua Lv and ChengXiang Zhai. 2009. A comparative study of methods for estimating query language models with pseudo feedback. In *CIKM*. 1895–1898.
[16] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. MS MARCO: A Human Generated Machine Reading Comprehension Dataset. In *Workshop on Cognitive Computing at NIPS*.
[17] Rodrigo Nogueira, Wei Yang, Kyunghyun Cho, and Jimmy Lin. 2019. Multi-stage document ranking with bert. *arXiv preprint arXiv:1910.14424* (2019).
[18] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *EMNLP-IJCNLP*.
[19] J.J. Rocchio. 1971. Relevance Feedback in Information Retrieval. In *The SMART Retrieval System - Experiments in Automatic Document Processing*. 313–323.
[20] Xiao Wang, Craig Macdonald, Nicola Tonellotto, and Iadh Ounis. 2021. Pseudo-Relevance Feedback for Multiple Representation Dense Retrieval. In *ICTIR*.
[21] Xiao Wang, Craig Macdonald, Nicola Tonellotto, and Iadh Ounis. 2022. ColBERT-PRF: Semantic Pseudo-Relevance Feedback for Dense Passage and Document Retrieval. *ACM Transactions on the Web* (2022).
[22] Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul N Bennett, Junaid Ahmed, and Arnold Overwijk. 2020. Approximate Nearest Neighbor Negative Contrastive Learning for Dense Text Retrieval. In *ICLR*.
[23] HongChien Yu, Chenyan Xiong, and Jamie Callan. 2021. Improving Query Representations for Dense Retrieval with Pseudo Relevance Feedback. In *CIKM*.
[24] Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Jiafeng Guo, Min Zhang, and Shaoping Ma. 2021. Optimizing dense retrieval model training with hard negatives. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1503–1512.

APPENDIX A

In Table 2 we report the results obtained by PRF models on the dev queries of the MS MARCO passage ranking dataset. We note that the characteristics of the relevance judgements of this dataset make results on this dataset not particularly insightful for evaluating

Table 2: Comparison between DR with and without PRF on the dev queries set of the MS MARCO dataset. All PRF parameters are not tuned: PRF depth =3 for average; for Rocchio $\alpha = 0.4$, $\beta = 0.6$, PRF depth = 5. Bold indicates the best results w.r.t. each base model.

Model	Method	MAP	nDCG@10	nDCG@100	Recall@1000
ANCE	Original	0.3362	0.4457	0.9587	0.3302
	Average PRF	0.3133	0.4247	0.9490	0.3073
	Rocchio PRF	0.3115	0.4250	0.9545	0.3048
TCT-ColBERT V1	Original	0.3416	0.4514	0.9640	0.3350
	Average PRF	0.2882	0.4014	0.9452	0.2816
	Rocchio PRF	0.2809	0.3988	0.9543	0.2740
TCT-ColBERT V2 HN+	Original	0.3644	0.4750	0.9695	0.3590
	Average PRF	0.3183	0.4325	0.9585	0.2995
	Rocchio PRF	0.3190	0.4360	0.9659	0.2933
DistillBERT KD	Original	0.3309	0.4391	0.9553	0.3250
	Average PRF	0.2830	0.3940	0.9325	0.2470
	Rocchio PRF	0.2787	0.3937	0.9432	0.2716
DistillBERT Balanced	Original	0.3515	0.4651	0.9771	0.3443
	Average PRF	0.2979	0.4151	0.9613	0.2630
	Rocchio PRF	0.2969	0.4178	0.9702	0.2897
SBERT	Original	0.3373	0.4453	0.9558	0.3314
	Average PRF	0.3094	0.4183	0.9446	0.3035
	Rocchio PRF	0.3034	0.4157	0.9529	0.2974
ADORE	Original	0.3523	0.4637	0.9688	0.3466
	Average PRF	0.3188	0.4330	0.9583	0.3127
	Rocchio PRF	0.3209	0.4376	0.9669	0.3145

PRF methods. This is because this dataset contains on average only one judged relevant passage per query, and this this relevant passage is often part of the PRF signal. This characteristic of the dataset may explain why on the dev queries of the MS MARCO passage ranking dataset PRF methods do not provide improvements in effectiveness compared to their corresponding dense retrievers – instead, they actually often provide sensible losses. This is unlike the main results reported in Section 4.2 which were obtained on the TREC DL collections. These collections use the same passages of MS MARCO, along with queries from the same logs used to created the queries in MS MARCO [1, 2], but consider a larger pool of relevance assessments for each query.