University of Glasgow Terrier Team at the TREC 2020 Deep Learning Track

Xiao Wang
University of Glasgow, UK
x.wang.8@research.gla.ac.uk

Yaxiong Wu
University of Glasgow, UK
y.wu.4@research.gla.ac.uk

Xi Wang
University of Glasgow, UK
x.wang.6@research.gla.ac.uk

Craig Macdonald, Iadh Ounis
University of Glasgow, UK
firstname.lastname@glasgow.ac.uk

ABSTRACT

This paper describes our submission to the document ranking task of the TREC 2020 Deep Learning Track. We followed a three-stage architecture: candidate set retrieval, feature calculation and re-ranking using a learning-to-rank technique. In particular, in the feature calculation stage, we leverage the traditional information retrieval document weighting models and the deep contextualised language models to provide the features for the learning-to-rank technique in the final stage. We submitted three runs for the document ranking task: `uogTr31oR`, `uogTrQCBMP` and `uogTrT20` and six baseline runs with no neural re-ranking techniques applied. Among our submitted runs, run `uogTrQCBMP`, which combines query expansion, ColBERT neural ranking and MaxPassage, was the most effective.

1 INTRODUCTION

The University of Glasgow Terrier team participated in the TREC 2020 Deep Learning track, in order to improve the effectiveness and flexibility of our new PyTerrier [9] information retrieval toolkit for an adhoc ranking task on a large corpus of Web documents. PyTerrier leverages Python to allow the easy expression of complex retrieval pipelines. In our participation, we addressed the document ranking task of the Deep Learning Track, but without using the provided initial rankings. We followed a three-stage framework: candidate set retrieval, feature calculation and re-ranking using a learning-to-rank technique. In the first stage, we performed the candidate set retrieval using DPH, DPH with Bo1 query expansion and a T5 [13]-based query expansion technique individually on the index created using Terrier. During the feature calculation, we incorporated traditional retrieval models as well as deep contextualised language models, such as BERT$_{CLS}$ [11], ColBERT [6] as well as `SciBERT` variants, to calculate the feature values. Finally, the LambdaMART [1] learning-to-rank technique was used to obtain the final score of each ranked document based on its various features.

The structure of the remainder of this paper is structured as follows: Section 2 discusses our indexing setup; Section 3 introduces the notions of retrieval pipelines in PyTerrier. Section 4 describes the three-stage architecture of our approach, including the candidate set retrieval followed by the feature calculation and learning-to-rank stage. Both the baseline runs and the submitted runs are detailed in Section 5. Section 6 highlights our results. Concluding remarks follow in Section 7.

2 INDEXING

We used the same indices for MSMARCO as we created for TREC 2019 [15]. In particular, for these indices, we chose not to use the TREC-formatted version of the MSMARCO corpus, but instead, reformatted the CSV files into TREC files such that the URL & title are clearly delineated. Next, we used several indexing configurations:

- Positions: We recorded positional information.
- Fields: We separately recorded the frequencies of terms occurring in different parts of the document. In particular, we recorded the ‘TITLE’, ‘BODY’ and ‘URL’ fields, following our past participations in the TREC Web track [10].
- Stemming & Stopwords: We did not apply stemming nor remove stopwords.

In all cases, we used the standard Terrier indexing configuration to create an inverted index, and a *direct* index to support query expansion and other retrieval techniques, as well as recording the raw text of the URLs, titles and contents of the documents as metadata, to allow deep learning upon these textual representations, as discussed further in Section 4.2 below. For the document contents, we saved only 4KB of plain text. Later experiments show that this limit had no marked impact on effectiveness.

Finally, we added an additional index for the ORCAS queries associated to each document.

3 PYTERRIER RETRIEVAL PIPELINES

All of our experiments and submitted runs for the TREC 2020 Deep Learning track built upon PyTerrier, our new expressive Python bindings for Terrier [9]. In particular, PyTerrier defines all retrieval components (rankers or re-rankers) to take the form of *transformer* objects, which transform one dataframe to another. To create flexible *pipelines* composed of multiple transformers, PyTerrier overloads standard Python operators for transformer objects as follows:

- >> (then): Pass the output of one transformer into another.
- + (linear combination): Combine the retrieval scores of two transformers, ala. CombSUM.
- ~ (cache): Cache (aka. memoize) the outputs of the retrieval transformer to disk, such that subsequent retrieval operations occur faster.
- $*$ (feature union): Consider the outputs of two transformers as separate features for learning-to-rank.

All ranking features described in the rest of this paper were expressed as pipelines of transformers using these operators. We refer the reader to [9] for more information about the PyTerrier data model and the flexibility of the operators.

4 ARCHITECTURE

Conceptually, our ranking framework mirrored that of our previous participation, in that it consists of 3 stages:

(1) Candidate retrieval
(2) Feature calculation
(3) Re-ranking using a learning-to-rank technique

In the following, we describe each of these stages in details.

4.1 Candidate Retrieval

In line with past experience from our participation to the TREC 2019 Deep Learning track, we used the DPH hypergeometric model [14] from the Divergence From Randomness (DFR) framework for the first retrieval pass. Indeed, we found DPH to be at least as effective as BM25 without the need for parameter tuning. Similarly, we used the Bo1 query expansion model. Figure 1 presents the heatmaps of retrieval effectiveness including MAP, NDCG and NDCG@10 when varying the number of feedback documents and expansion terms for Bo1 on the TREC 2019 document ranking topics. We found that Bo1 was very effective with its default settings of 3 feedback documents and 10 expansion terms – this is the setting applied in our TREC 2020 participation. Finally, we experimented with a query expansion variant that makes use of T5 [13] for generating an expanded query.

4.2 Feature calculation

We then focused on re-ranking the candidate documents. Each feature was expressed as a PyTerrier retrieval pipeline. Our feature groups encapsulated both classical ranking features, such as statistical retrieval models, and query independent features. We also had a number of BERT-based features.

Retrieval models. We calculated DPH and the number of matching query term features on each field (title, URL, contents). We also calculated a sequential dependence feature [12]. Finally, we applied DPH on smaller passages of the document text, and then took the maximum score.

Query independent features. Field lengths for title, URL and document contents, in terms of the number of indexed tokens.

Query Expansion. Rescoring of the candidate documents by application of the Bo1 query expansion mechanism; similarly, rescoring of the top-documents by application of collection enrichment, i.e. query expansion with the reformulated query obtained from the top-ranked Wikipedia documents.

GLoVe. Representation of document and query in the word2vec space, using the GloVe embeddings. Query document similarity was computed using Word Movers Distance [7], as implemented by GenSim.

BERT-based features. We calculated BERT$_{CLS}$ [11] and ColBERT [6] features[1]. In both cases, we also varied the underlying neural language model between `bert-based-uncased` and `scibert`. For BERT$_{CLS}$, we used the CEDR implementation [8].

Following [3], for long documents, we added features applying MaxPassage on both the BERT$_{CLS}$ and ColBERT neural re-rankers, by composing passaging and maximum score transformers around the BERT transformers. Listing 1 provides salient PyTerrier code for creating a retrieval pipeline encompassing DPH with query expansion and ColBERT MaxPassage, composed using the >> (then) operator. Finally, we devised linear combinations of BERT features, using PyTerrier's + operator.

Listing 1: DPH with QE and ColBERT MaxPassage.

```
# deploy DPH + QE, retrieve the doc metadata
firstpass_dph_qe = pt.BatchRetrieve(index,
                   wmodel="DPH",
                   metadata=["docno", "title", "body"],
                   controls={"qe" : "on", "qemodel" : "Bo1"},
                   num_results=100)

# load an existing fine-tuned ColBERT model
colbert = ColBERTPipeline("dir/colbert.dnn", doc_attr="body")

# compose colbert, using MaxPassage
qe_colbert_100_maxpassage = firstpass_dph_qe >>
    pt.text.sliding(
        length=128,
        stride=64,
        prepend_attr = "title") >>
    colbert >>
    pt.text.max_passage()
```

Entity BERT Feature. We also calculated a variant of BERT$_{CLS}$ using entities.

Table 1 summarises all feature groups and the total number of features in each group. Not all features were applied for all runs. For this reason, the right-hand side of Table 1 has columns for four *feature sets*, which define the features groups used within each feature set. Of note, the "16" and "17o" feature sets do not use deep learned neural ranking features; "17o" and "31o" feature sets both deploy a feature encapsulating the ORCAS query click as a field; finally, for the "20" feature set, we eliminated a subset of the BERT-based features based on feature performance and learner feature importances.

4.3 Learning-to-rank

Following previous years, we made use of a learning-to-rank technique to combine the various retrieval approaches. To do so, all selected features were combined using the $*$ (feature union) operator of PyTerrier.

While we again used LambaMART [1], we switched from the Jforests [4] implementations we have used in recent years to the Python integrations provided by xgBoost [2] and LightGBM [5]. After some experimentation, we settled on LightGBM as the most effective of the two.

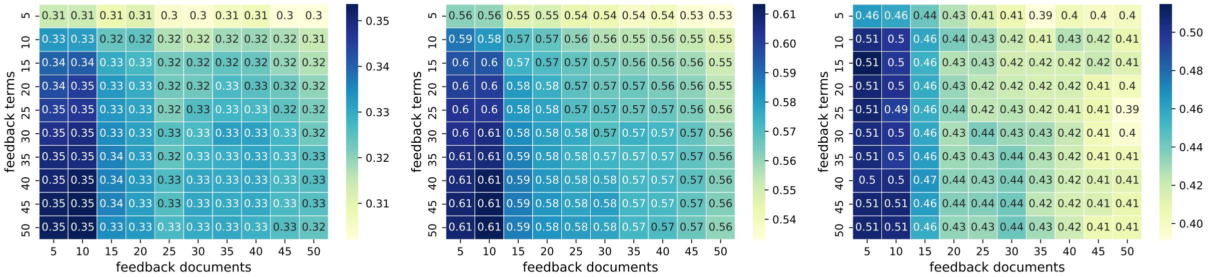
4.4 Experimental Setup Details

We used 1000 queries from the MSMARCO training document ranking task for training the BERT$_{CLS}$ deep learned models and the learning-to-rank configurations. The ColBERT model was trained using the file `triples.train.small.tsv.gz` from the MSMARCO passage ranking task. Validation and run selection were conducted using the TREC 2019 Deep Learning track document ranking task queries and relevance assessments.

When applying MaxPassage, we applied a sliding window of 128 tokens which advanced by 64 tokens each time. The title of the document is prepended to each passage.

All our runs were created using PyTerrier in Jupyter notebooks. PyTerrier is available from https://github.com/terrier-org/pyterrier. Example notebooks for runs submitted to the MSMARCO leaderboard are available at https://github.com/cmacdonald/pyterrier-

[1] As estimated through the special CLS token, explaining the BERT$_{CLS}$ name.



(a) MAP scores **(b) NDCG scores** **(c) NDCG@10 scores**

Figure 1: Query expansion performances with Bo1 for given distinct feedback terms and documents.

Table 1: Feature group descriptions and alignment with feature sets. ✓ denotes that all features in that group (row) were used in that feature set (column), while # denotes the number of features selected from that group.

Feature Group	Description	#	Feature Sets			
			16	17o	20	31o
Retrieval Models	DPH, DPH on individual fields, PL2F, number of matching query terms per field, DFR sequential dependence, DPH MaxPassage	10	✓	✓	✓	✓
QI Features	Field lengths	3	✓	✓	✓	✓
QE Features	Query expansion and collection enrichment scores	2	✓	✓	✓	✓
GloVe	Word Mover Distance	1	✓	✓	✓	✓
BERT-based Features	BERT_{CLS}, ColBERT as well as SciBERT variants, with and without MaxPassage	12			✓	✓
Entity-based BERT Feature	A variant of BERT_{CLS} encapsulating entity knowledge	1				✓
ORCAS	DPH calculated on the ORCAS field	1		✓		✓

msmarco-document-leaderboard-runs. Our integrations of CEDR and ColBERT are available as PyTerrier plugins from https://github.com/cmacdonald/pyterrier_bert.

5 RUNS

We submitted 3 runs to the Document Ranking task. We were also invited by the track organisers to submit baselines runs, which applied no neural re-ranking techniques. We chose to submit 6 baseline runs, which also acted as baselines for our group's main submission runs. In addition, we describe three additional runs on our stemmed index.

5.1 Baseline Runs

Our submitted baseline runs to the document ranking task of the TREC 2019 Deep Learning track contained no neural features, and constituted runs with and without learning-to-rank and query expansion. In particular, our 6 baseline runs are as follows:

- `uogTrBaseDPH`: Applying DPH on our unstemmed index.
- `uogTrBaseDPHQ`: Applying DPH and Bo1 query expansion on our unstemmed index. This corresponds to the retrieval transformer expressed in line 2 of Listing 1.
- `uogTrBaseL16`: Re-ranking the results identified by `uogTrBaseDPH` using the 16 feature set. Re-ranking is performed by LightGBM.
- `uogTrBaseL17o`: As `uogTrBaseL16`, but with an additional feature in the form of a DPH score on the ORCAS field.
- `uogTrBaseQL16`: As `uogTrBaseL16`, but re-ranking the candidate set obtained from `uogTrBaseDPHQ`.
- `uogTrBaseQL17o`: As `uogTrBaseL17o`, but re-ranking the candidate set obtained from `uogTrBaseDPHQ`.

5.2 Submitted Group Runs

We submitted the following 3 runs:

- `uogTr31oR`: Applying all 31 features, re-ranked using LightGBM. This run re-ranks the candidate sets obtained from `uogTrBaseDPH`. Compared to `uogTrBaseL17o`, it adds neural re-ranking features.
- `uogTrQCBMP`: Applies ColBERT MaxPassage on the candidate sets obtained from `uogTrBaseDPHQ`. This run can be obtained using the code in Listing 1.
- `uogTrT20`: Applies the 20 features set on the candidate set identified by our T5-based query expansion model.

In addition, we describe 3 additional runs that using a stemmed index (but do not apply LTR):

- `uogTrBaseDPHSS`: Applying DPH on our stemmed index.
- `uogTrBaseDPHQSS`: Applying DPH and Bo1 query expansion on our stemmed index.
- `uogTrQCBMPSS`: Applies ColBERT MaxPassage on the candidate sets obtained from `uogTrBaseDPHQSS`.

6 RESULTS & ANALYSIS

Table 2 lists the obtained effectiveness results for all our runs, including our baseline, submitted group runs and additional runs, as well as the TREC per-topic best and median scores across all participating systems, in terms of MAP, P@10[2], NDCG@10, NDCG@100 and MRR.

Firstly, we analyse the performance of the baseline runs. Comparing `uogTrBaseDPH` with `uogTrBaseDPHQ`, it is clear that query expansion benefited all metrics except MRR. For the application

[2] We were not provided with Best or Median numbers for P@10.

of learning-to-rank, the picture is mixed: there are improvements between uogTrBaseDPH and uogTrBaseL16 for all metrics except MRR; between uogTrBaseDPHQ and uogTrBaseQL16, only a small improvement for NDCG@100 is observed. Finally, we see that adding an ORCAS feature made small improvements: comparing uogTrBaseL16 with uogTrBaseL17o, P@10, NDCG@10, NDCG@100 are slightly improved; comparing uogTrBaseQL16 with uogTrBaseQL17o, MAP, P@10, NDCG@10, NDCG@100 and MRR are also improved.

Next, we turn our attention to our submitted group runs. Among these three runs, run uogTrQCBMP – which did not deploy learning-to-rank – was the most effective on all metrics except MRR; this run deployed ColBERT MaxPassage on top of DPH + QE. Compared to the corresponding baseline uogTrBaseDPHQ run, this achieved an improvement of 8.4% in MAP and 14.6% in NDCG@10.

Among the two learning-to-rank runs, uogTrT20 was the most effective for MAP and P@10; on closer examination of the results, we conclude that the promising effectiveness of this approach is due to the reduced learning-to-rank feature set, rather than the new query expansion approach. Comparing uogTr31oR with baseline uogTrBaseDPH, we find that the BERT features result in an uplift of 22.2% in MAP (0.3070 → 0.3722) and 18.8% in NDCG@10 (0.4871 → 0.5476). To illustrate the effectiveness of the deployed features, Figure 2 plots the performances of the features for the TREC 2019 and TREC 2020 topics (numbers above each bar are the rank of that feature among all others). We observe a strong correlation between the performances of the features in these rankings, with the strongest feature (ColBERT MaxPassage, as deployed in run uogTrQCBMP) consistently the most effective. In both figures, the linear combination of SciBERT and BERT results in the 2nd most effective feature.

Finally, we look at the overall reported performances. We note that the TREC Median pseudo-system appears to be very strong this year, and we were disappointed not to exceed it on average (we only exceed it for NDCG@10, 0.5733 vs. 0.5791); we have since investigated and eliminated possible confounding variables such as the stemming configuration, and the 4KB size of the document recorded in the Terrier index used for the BERT models. We continue to investigate ways to improve our performance compared to the TREC median.

Analysing the overall trends, while it is clear that learning-to-rank models are flexible and can be used to successfully combine effective ranking features, for this task, it may not result in models that are more effective than the most effective constituent features. The high effectiveness of run uogTrQCBMP is testament to this observation. After the TREC submissions, we further conducted additional runs on a stemmed index, namely uogTrBaseDPHSS and uogTrBaseDPHQSS, as well as uogTrQCBMPSS. In general, compared to the unstemmed index, runs using the stemmed index provide higher performance. Among the three additional runs, uogTrQCBMPSS exceeds the TREC median performance on all metrics except MRR demonstrating the effectiveness of the established model and the importance of the proper experimental setting for stemming on this test collection.

Table 2: Performance of submitted and baseline runs.

	MAP	P@10	NDCG@10	NDCG@100	MRR
TREC Best	0.6109	-	0.8019	0.7663	0.9822
TREC Median	0.3902	-	0.5733	0.5859	0.9444
baseline runs					
uogTrBaseDPH	0.3070	0.5089	0.4871	0.4972	0.8415
uogTrBaseDPHQ	0.3461	0.5444	0.5052	0.5345	0.8052
uogTrBaseL16	0.3248	0.5289	0.4964	0.5052	0.8219
uogTrBaseL17o	0.3248	0.5333	0.5120	0.5090	0.7980
uogTrBaseQL16	0.3436	0.5311	0.4998	0.5357	0.7930
uogTrBaseQL17o	0.3530	0.5511	0.5203	0.5399	0.8276
submitted runs					
uogTr31oR	0.3468	0.5622	0.5476	0.5284	**0.8926**
uogTrQCBMP	**0.3752**	**0.6000**	**0.5791**	**0.5673**	0.8722
uogTrT20	0.3692	0.5667	0.5453	0.5296	0.8711
additional runs					
uogTrBaseDPHSS	0.4023	0.5689	0.5145	0.5626	0.8313
uogTrBaseDPHQSS	0.4213	0.5778	0.5173	0.5734	0.8343
uogTrQCBMPSS	0.4066	0.6022	0.5936	0.6050	0.8819

7 CONCLUSIONS

Overall, our participation in the TREC Deep Learning track was a useful activity to refine methods of integration of deep learning techniques as retrieval pipelines in PyTerrier. In terms of effectiveness, our most effective run uogTrQCBMP outperformed the TREC median in terms of NDCG@10 and uogTrQCBMPSS exceeds the TREC median on all metrics except MRR. Moreover, the learning-to-rank runs were not among the most effective, which emphasises the difficulty in learning effective models for adhoc retrieval tasks using training datasets with very few judgements, echoing our findings from TREC 2019.

ACKNOWLEDGEMENTS

The co-authors acknowledge the assistance of Siwei Liu, Sarawoot Kongyoung and Ting Su in supporting our participation.

REFERENCES

[1] Christopher J.C. Burges. 2010. *From RankNet to LambdaRank to LambdaMART: An Overview.* Technical Report MSR-TR-2010-82.

[2] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining.* 785–794.

[3] Zhuyun Dai and Jamie Callan. 2019. Deeper text understanding for IR with contextual neural language modeling. In *Proceedings of SIGIR.* 985–988.

[4] Yasser Ganjisaffar, Rich Caruana, and Cristina Lopes. 2011. Bagging Gradient-Boosted Trees for High Precision, Low Variance Ranking Models. In *Proceedings of SIGIR.* 85–94.

[5] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in neural information processing systems.* 3146–3154.

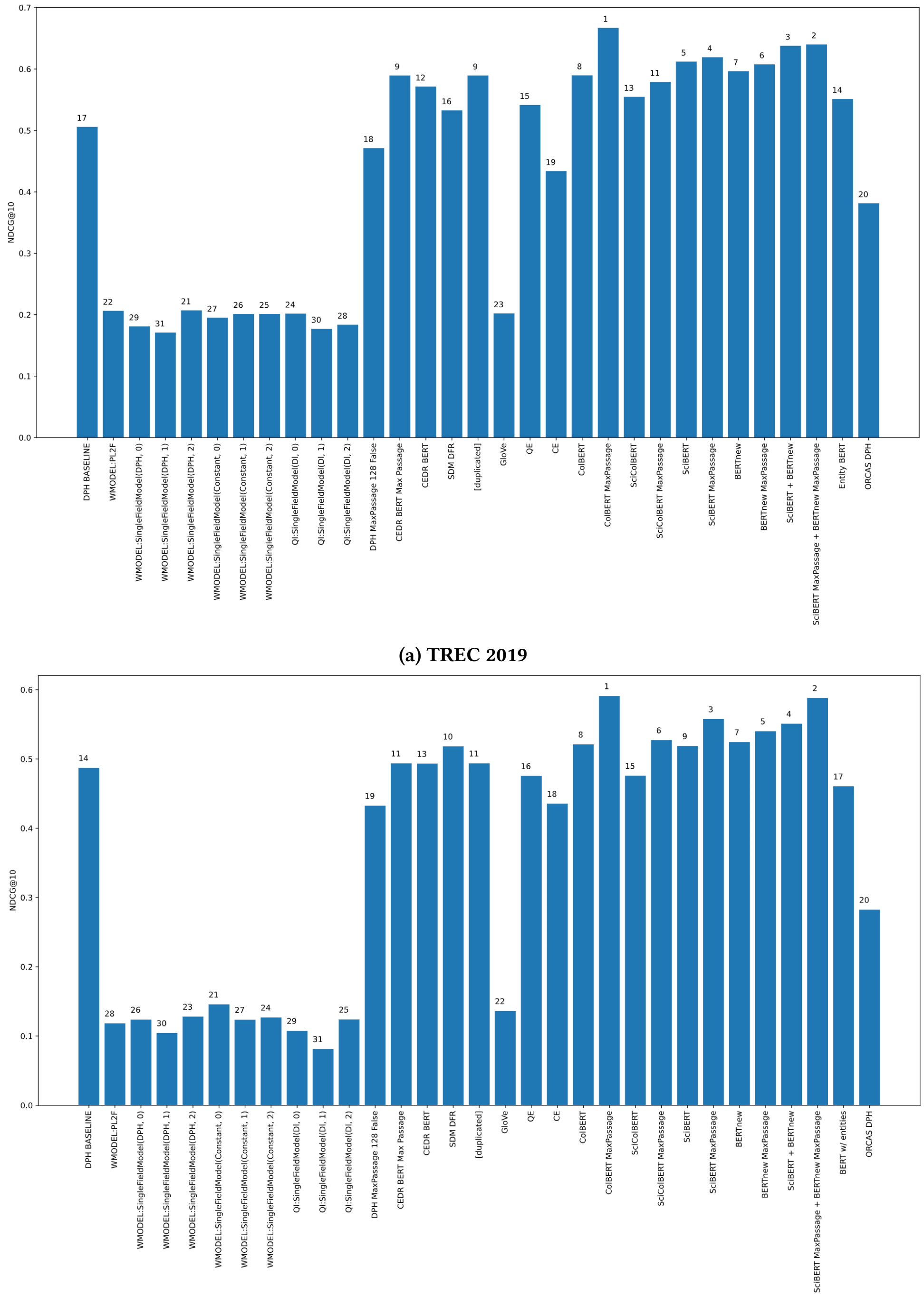
[6] Omar Khattab and Matei Zaharia. 2020. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. *arXiv preprint arXiv:2004.12832* (2020).

[7] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. 2015. From word embeddings to document distances. In *International conference on machine learning.* 957–966.

[8] Sean MacAvaney, Andrew Yates, Arman Cohan, and Nazli Goharian. 2019. CEDR: Contextualized embeddings for document ranking. In *Proceedings of SIGIR.* 1101–1104.

[9] Craig Macdonald and Nicola Tonellotto. 2020. Declarative Experimentation in Information Retrieval using PyTerrier. In *Proceedings of the 2020 ACM SIGIR on International Conference on Theory of Information Retrieval.* 161–168.

[10] Richard McCreadie, Craig Macdonald, Iadh Ounis, Jie Peng, and Rodrygo L. T. Santos. 2009. University of Glasgow at TREC 2009: Experiments with Terrier. In *Proceedings of TREC 2009.* 177–185.



(a) TREC 2019

(b) TREC 2020

Figure 2: NDCG@10 feature performances for features in the 31o feature set - these features are constituents of `uogTr31oR`.

[11] Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage Re-ranking with BERT. *arXiv preprint arXiv:1901.04085* (2019).

[12] Jie Peng, Craig Macdonald, Ben He, Vassilis Plachouras, and Iadh Ounis. 2007. Incorporating term dependency in the DFR framework. In *Proceedings of SIGIR*. 843–844.

[13] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research* 21, 140 (2020), 1–67.

[14] Stephen E Robertson and Steve Walker. 1994. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *Proceedings of SIGIR*. 232–241.

[15] Ting Su, Xi Wang, Craig Macdonald, and Iadh Ounis. 2019. University of Glasgow Terrier Team at the TREC 2019 Deep Learning Track. In *Proceedings of TREC*.