University of Glasgow Terrier Team at the TREC 2019 Deep Learning Track

Ting Su
University of Glasgow, UK
t.su.2@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

For TREC 2019, we focus on combining deep learning methods with traditional information retrieval methods, by using deep learning scores as an extra feature in the re-ranking process. In particular, we explore the effectiveness of using deep learning techniques based on the state-of-the-art BERT contextual language models, as well as taking into account alternative query reformulations in the re-ranking process. We submitted three official runs to the document ranking task: `uogTrDNN6LM`, `uogTrDSS6pLM`, and `uogTrDSSQE5LM`, where all three runs deploy a deep learning method and the LambdaMART learning-to-rank method. Our results show that `uogTrDNN6LM` is competitive, performing above the TREC median in terms of MAP and NDCG, yet a simple untrained DFR query expansion run was more effective.

1 INTRODUCTION

The University of Glasgow Terrier team participated in the TREC 2019 Deep Learning track, in order to improve the integration between our Terrier.org Information Retrieval (IR) platform [15] and recent deep learning techniques, and to improve the effectiveness of Terrier on a large Web corpora and an adhoc ranking task.

In particular, we participated in the document ranking task of the Deep Learning track, but without using the provided initial rankings. Instead, we indexed the MSMARCO corpus using Terrier v5.2, and then used Terrier to perform the initial *first-phase* retrieval. We also used Terrier to re-rank the candidate results identified in the first-phase, using tools such as CEDR [13] for deep semantic matching and other query expansion techniques (e.g. collection enrichment and axiomatic query expansion [9]). These results were combined using a LambdaMART [5] learning-to-rank technique. In this way, we leveraged our existing Terrier infrastructure, while enhancing it with the integration of new deep learning techniques.

The structure of the remainder of this paper is structured as follows: Section 2 discusses our indexing setup; Section 3 describes our first-phase ranking setup; Section 4 describes the features used in the second (re-ranking) phase, including deep learning; Section 5 describes our submitted runs; Section 6 highlights our results; Concluding remarks follow in Section 7.

2 INDEXING

We indexed the corpus using Terrier v5.2. Firstly, 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 and 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 [18].
- Stemming & Stopwords: We deployed two configurations: Porter's stemmer and removing standard stopwords (denoted SS in our runs); or not applying any treatment - i.e. no stemming, no stopwords removed (denoted NN).

In all cases, we created 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 below.

3 FIRST-PHASE RETRIEVAL

After indexing the corpus with Terrier and the corresponding indexing configurations, we conduct the first-phase retrieval from the document collection.

First, we apply batch retrieval using Terrier on the queries with specific configurations matching indexed corpus with the same retrieval configuration setup. For example, to retrieve documents with the 'Stemming & Stopwords' configured indexing of the given queries, we need to remove stopwords and apply stemming to the terms in the queries.

Next, we use three weighting models to calculate the scores of retrieved documents, rank documents with the obtained scores and then provide the candidate document set for the *second-phase* retrieval. We describe these three weighting models as follows:

- DPH is a divergence from randomness (DFR)-based [22] hyper-geometric weighting model. The 'P' in the DPH model indicates the Popper's normalization [2]. Moreover, DPH is a parameter-free model. This means that DPH can provide document scores without fine-tuning parameters. In particular, DPH is the default weighting model of Terrier v5.2.
- BM25 (i.e. Okapi BM25) is a popular weighting model and has been adopted in many studies [12, 17] to address retrieval-based tasks. Compared to DPH, we note that BM25 can be trained with fine-tuned parameters. Hence, it provides corpus-fitted models.
- PL2 is another DFR-based weighting model. According to Amati [3], the PL2 model can provide good performances for tasks requiring high early precision.

We also tested the use of query expansion. For each query, we select the most informative terms from the top-returned documents to expand the query. In particular, we use three top-returned documents in the experimental setup. Then, we assign each term in

these documents with a weighting score according to the Bo1 term weighting model [1]. After that, we use the top 10 terms with the highest weighting score to expand the query. With these expanded queries, we also rank the retrieved documents by using the three weighting models introduced above, thereby providing another candidate document set.

We partitioned 100 and 1000 queries from the provided 367013 MSMARCO training queries to evaluate the effectiveness of various combinations of indexing configurations, weighting models and query expansion. After evaluating the retrieved documents of different retrieval strategies in terms of MAP, P@5, P@10 and Recall@1000, we summarise the three main conclusions in our initial first-phrase retrieval experiments as follows:

- After applying the stemming and stopwords removal configuration to index documents and format queries, we can retrieve documents with a higher precision score compared to other configurations (i.e. (1) without stemming and keeping stopwords, (2) stopwords removed only).
- The DPH weighting model outperforms the other two weighting models (i.e. BM25 and PL2) by providing higher precision scores. Moreover, these three weighting models all achieve similar recall scores, around 90%.
- The query expansion configuration can increase recall, to provide reliable candidate document sets for the second-phrase retrieval.

Our experiments found these conclusions to be consistent while using 100 or 1000 queries sampled from the MSMARCO training set.

4 SECOND-PHASE RETRIEVAL

In this section, we describe our attempts at improving adhoc reranking performance. In particular, we use two techniques: *rewriters*, which allow the calculation of additional query dependent features on the candidate document set obtained from the first set, where the query dependent features use alternative query formulations (Section 4.1); and deep learning techniques for semantic matching (Section 4.2).

4.1 Rewriters

In the past, Terrier has been flexible to allow arbitrary weighting models to be expressed as separate features. This is implemented using the *Fat framework* [16], which caches the postings matching the original query terms for documents entering the top K candidate set. In doing so, it is said to *fatten* the result set with posting information. This means that more query dependent features can be efficiently calculated for these documents without re-traversing the inverted index. Figure 1 provides examples of feature definitions supported by Terrier since version 4.0.

However, a disadvantage of the Fat framework is that the additional query dependent features can only be calculated for the original query terms. Asadi & Lin [4] described an alternative framework, which they called *doc vectors*, whereby the *direct* (a.k.a. *forward*) index data structure is used in the second retrieval phase. This has the advantage that query dependent features can use query terms not present in the original query.

```
SAMPLE #the first pass retrieval score
WMODEL:BM25 #BM25 on the whole document
WMODEL:SingleFieldModel(BM25,0) #BM25 on the title
WMODEL:SingleFieldModel(BM25,1) #BM25 on the body
QI:StaticFeature(OIS,/path/to/inlinks.oos.gz) #inlinks
```

Figure 1: Examples of standard feature definitions in Terrier.

```
WMODEL$original:SingleFieldModel(DPH,0)
    #DPH on title, original query
WMODEL$original:SingleFieldModel(DPH,1)
    #DPH on body, original query
WMODEL$qeBo1:DPH
    #DPH on Bo1 expanded query from top documents
WMODEL$ce:DPH
   #DPH on Bo1 expanded query from top Wikipedia documents
WMODEL$ax:DPH
    #DPH on axiomatic QE terms
WMODEL$prox:org.terrier.matching.models.dependence.pBiL
    #pBiL on MRF proximity terms
```

Figure 2: Examples of feature definitions for groups of query terms obtained from rewriters. Query term tags are denoted by $ after the feature type.

We implemented the doc vectors approach using Terrier, and in particular, build on it to express different rewritten forms of the query as separate features for learning-to-rank. The actions that create different query rewrites are called rewriters - we deployed several rewriters:

- Bo1 QE: Divergence from Randomness query expansion using the Bo1 term weighting model [1] on the top-ranked documents in the first-phase candidate set.
- Collection enrichment (CE): DFR Bo1 Divergence from Randomness query expansion on the top-ranked documents from Wikipedia [14].
- Axiomatic query expansion (axqe) [9, 23].
- Markov Random Fields proximity [19, 21].

In deploying these rewriters, we are able to re-score the documents in the first-phase candidate set using different forms of expanded queries. Each rewriter is a different feature. Different weighting models can be expressed as features on the query terms obtained for each rewriter. Figure 2 provides examples of query dependent features defined on groups of query terms obtained from different rewriters - the $ symbol denotes the *tag* of the query terms forming a given group of terms obtained from a given rewriter.

4.2 Deep Learning

Our use of deep learning techniques initially focused on using MatchZoo [11][1], including adapting it to use different negative sampling strategies.

However, for various reasons, including both overall effectiveness and support for the state-of-the-art BERT language models [8], we changed strategy and opted for using the CEDR toolkit [13][2]

[1] https://github.com/NTMC-Community/MatchZoo
[2] https://github.com/Georgetown-IR-Lab/cedr

Table 1: Feature used in different runs. Note that feature #1 also scores the expanded terms in the case of the SSQE run.

#	Feature Set	Features	Feature description
1	SAMPLE	DPH	DPH score between entire document and query
2	WMODEL	SingleFieldModel(DPH,0)	DPH score between doc title and query
3	WMODEL	SingleFieldModel(DPH,1)	DPH score between doc URL and query
4	WMODEL	$qeBo1:DPH	Bo1 Query Expansion: DPH score
5	WMODEL	$prox:pBiL	DFR proximity
6	WMODEL	$ax:DPH	Axiomatic QE: DPH score
7	WMODEL	$ce:DPH	Collection Enrichment QE: DPH score
8	DSM	CEDR-PACRR	CEDR-PACRR on the document level
9	DSM	passage-level CEDR-PACRR	maximum CEDR-PACRR score across all passages

instead. We extended and integrated CEDR into Terrier as follows: We implemented a DocumentScoreModifier in Terrier that allows to obtain the scores of the top-ranked documents from CEDR. In particular, it collects the contents of the documents from the Terrier index, sends these and the query over a HTTP REST-ful connection to a Python-based server serving the CEDR models, which then returns the computed scores of the documents to Terrier. Our extension of Terrier is avaiable from Github[3].

CEDR provides two ranking models: a BERT model, and a PACCR model based on the BERT model. In order to train the CEDR model, we first fine-tune the pre-trained BERT model, which is then used within the CEDR model. For both BERT and PACCR models, we introduced early stopping, which terminates training if there was no validation improvement for 20 iterations.

Additionally, to enhance the effectiveness, and inspired by the recent work of Dai & Callan [7], we adapt CEDR to apply *passaging* to long documents. In particular, by breaking up long documents into shorter passages, effective models are more easily learned. Following Dai & Callan, we break documents into passages of 150 tokens, with a stride of 75 between passages, which are prepended with the title of the document. To obtain the final score of a document, we take the max of the scores of the passages within the document. Our adaptation of CEDR is available from Github[4].

In terms of setup, we trained all CEDR models using 1000 queries (from the MSMARCO training set), ranked to depth 1000, and for validation, we used 200 queries ranked to depth 50. In particular, reducing the rank depth of the validation set significantly reduced the speed of training iterations.

4.3 Learning-to-Rank: Combination of Features

We used the LambdaMART [5] learning-to-rank technique to combine the features described above. We use the Jforests LambdaMART implementation [10][5]. We also tried the Tensorflow TF Ranking package, but at the time of our TREC campaign, the released version did not allow for the prediction of scores using a learned model[6]. Thus, we report unsubmitted runs using TF Ranking in the next sections. Furthermore, we also report effectiveness using a simple linear combination of feature values, with weights optimised to maximise MAP using Simulated Annealing, called Automatic Feature Selection (AFS) [20].

All our learning-to-rank runs were trained on 1000 queries drawn from the MSMARCO training set, with learning validation (e.g. setting number of iterations) using a further 200 queries.

Table 2: Feature used for each run

Run	Features used	Re-ranking method
Submitted runs		
uogTrDNN6LM	1,2,5,6,7,8	LambdaMART
uogTrDSS6pLM	1,2,3,5,8	LambdaMART
uogTrDSSQE5LM	1,2,3,5,8	LambdaMART
Unsubmitted runs		
uogTrDNN	-	-
uogTrDSS	-	-
uogTrDSSQE	-	-
uogTrDNN6AFS	1,2,5,6,7,8	Automatic feature selection
uogTrDSS6pAFS	1,2,3,5,8,9	Automatic feature selection
uogTrDSSQE5AFS	1,2,3,5,8	Automatic feature selection
uogTrDNN6TFR	1,2,5,6,7,8	Tensorflow Ranking
uogTrDSS6pTFR	1,2,3,5,8,9	Tensorflow Ranking
uogTrDSSQE5TFR	1,2,3,5,8	Tensorflow Ranking

5 SUBMITTED AND UNSUBMITTED RUNS

Table 1 lists the names and descriptions of all the features we used in our experiments, for the re-ranking process. Table 2 lists the detailed features we used in each of our runs.

5.1 Submitted Runs

We submitted the following runs to the document ranking task of the TREC 2019 Deep Learning track:

- `uogTrDNN6LM`: In this run, we use the index without prior stemming and stopword removal, denoted NN. We use the DPH model to rank documents, and deploy CEDR-PACRR on the document level to obtain a document's deep learning representation score. We use 8 features in our re-ranking process, as shown in Table 2. The final ranking of documents used a LambdaMART learned model. This method is submitted as our main test run.
- `uogTrDSS6pLM`: In this run, we use the stemmed and stopwords removed index (SS). We deploy DPH in the first pass

[3] https://github.com/terrierteam/terrier-nrr [4] https://github.com/cmacdonald/cedr
[5] https://github.com/yasserg/jforests. [6] This has now been fixed - see https://github.com/tensorflow/ranking/issues/104.

of ranking. In addition to the document level CEDR-PACRR score, we further obtain the maximum CEDR-PACRR scores for each paragraph in the document, as one of the deep learning representation for each document. We use 6 features in our reranking process, as shown in Table 2. The final ranking of documents also used a LambdaMART learned model.

- `uogTrDSSQE5LM`: The difference between this run and the `uogTrDSS6pLM` run, is that we additionally deploy QE during the first pass ranking. Furthermore, similarly to the `uogTrDNN6LM` run, we use the document-level CEDR-PACRR scores *only* for each document, as the deep learning representation for each document. Similar to the previous two runs, the final ranking of documents also uses a LambdaMART learned model.

5.2 Unsubmitted Runs

For each of the aforementioned submitted runs, we report the effectiveness of the first phase retrieval approach for each of the submitted runs (i.e. NN, SS and SSQE) as unsubmitted runs. We also report performances using two different learning-to-rank techniques, namely, AFS and Tensorflow Ranking (TF Ranking), using the same feature sets as the submitted runs. Finally, later in Section 6, we also report the effectiveness of the individual feature sets of the submitted runs.

6 RESULTS & ANALYSIS

Table 3 lists the obtained effectiveness results for our submitted and unsubmitted runs, as well as the TREC per-topic best and median scores across all participating systems, in terms of MAP, P@10, NDCG@10 and NDCG@1000[7]. We also report the effectiveness of the top 100 results obtained from an Indri QueryLikelihood run, as provided by the track organisers for the re-ranking task (denoted TREC provided top 100). All evaluation metrics are calculated using the official qrels, as judged by NIST assessors. Firstly, on analysing the table, we note that one of our submitted runs, `uogTrDNN6LM` exceeds the median performance for MAP and NDCG; It also outperforms the TREC provided top 100 for all four measures; Our other two runs are below the median for the relevant measures (P@10, MAP & NDCG@1000).

Next, we analyse the performance of our unsubmitted runs. First, it is apparent that a simple application of Terrier's standard query expansion, i.e. `uogTrDSSQE`, and without any learning (deep or otherwise), would have outperformed all of our submitted runs for MAP & P@10. Moreover, on inspection of the track overview paper [6], we note that it would have outperformed the highest 'traditional' submitted run among all participants (`srchvrs_run1`) by 7% for NDCG@10 (0.5610 → 0.6007).

Next, we note that while the alternative learning-to-rank techniques, TF Ranking and AFS, could both marginally enhance P@10 on the 50 topics, they could not enhance the high MAP of `uogTrDNN6LM`. In contrast, it is clear that the low effectiveness of `uogTrDSS6pLM` and `uogTrDSSQE5LM` is unexpected, as the same features re-ranked using AFS or TF Ranking would have resulted in dramatically increased effectiveness.

To analyse further the effectiveness of the various features, Table 4 reports the effectiveness of the various features evaluated using the TREC official qrels. On analysing the table, we note the following observations: Firstly, the initial rankings obtained from DPH are more effective than the other features, across all three settings (6 out of 9 measurements in Table 4). However, we do observe that P@10 can be enhanced in two cases by the BERT-based CEDR models. Between the two CEDR models, we lack a fair comparison, but we believe that the passage-based CEDR model (feature #9) appears to be more effective than the document-level model (feature #8) in comparison to the various first-pass retrieval settings (feature #1).

Table 3: Performance of submitted and unofficial runs.

	MAP	P@10	NDCG@10	NDCG@1000
TREC Best	0.5150	0.8800	-	0.7428
TREC Median	0.2989	0.6907	-	0.5394
TREC provided top 100	0.2367	0.5977	0.5147	0.4303
`uogTrDNN6LM`	0.3158	0.6744	0.6060	**0.5771**
`uogTrDSS6pLM`	0.1250	0.5372	0.6333	0.3918
`uogTrDSSQE5LM`	0.1240	0.5442	0.6381	0.4207
Unsubmitted runs				
`uogTrDNN`	0.2943	0.5837	0.5059	0.5482
`uogTrDSS`	0.2875	0.6116	0.5462	0.5014
`uogTrDSSQE`	**0.3434**	0.6930	0.6007	0.5549
`uogTrDNN6AFS`	0.2649	0.6512	0.5689	0.5544
`uogTrDSS6pAFS`	0.2976	**0.6953**	0.6290	0.5176
`uogTrDSSQE5AFS`	0.2943	0.5837	0.5058	0.5481
`uogTrDNN6TFR`	0.2836	0.6605	0.5979	0.5627
`uogTrDSS6pTFR`	0.2823	**0.6953**	0.6246	0.5103
`uogTrDSSQE5TFR`	0.2365	0.6535	0.5934	0.5393

Finally, we compare and contrast the results in Table 4 with those results that we obtained before the runs were submitted. In particular, using an internal "test" set of 200 topics drawn from the MSMARCO training query set (separated from our internal training and validation sets), we compare and contrast the effectiveness of the features in Table 4 (evaluated using the final official evaluation qrels) with that obtained on the internal test set. The resulting scatterplots, for MAP and NDCG@10, are shown in Figure 3. In particular, for MAP, Figure 3a shows that a fairly weak overall correlation can be observed between effectiveness on our internal test set (sampled from the MSMARCO training queries) and on the final official evaluation qrels. For example, the correlation for the DNN6 setting is only Spearman's $\rho = 0.18$, although other settings exhibit higher correlations. On the other hand, the stronger correlation observed for NDCG@10 (Figure 3b) suggests that the CEDR PACRR features are clearly the most effective features for NDCG@10, on both the training and official qrels. Overall, we conclude that there are significant differences between the labels for the MSMARCO training queries and the final official TREC qrels, which would result in differing conclusions for feature selection and learning-to-rank for MAP. Indeed, on inspection of the various relevance assessments, we found that the training queries had very few relevant documents (1.04 on average) compared to the average of 153.4 for the official TREC qrels.

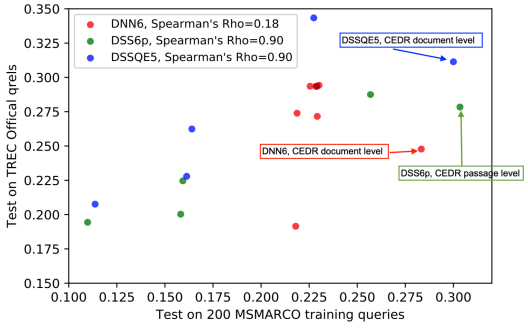
7 CONCLUSIONS

Overall, our participation in the TREC Deep Learning track was a useful activity to increase our understanding about the effective integration of deep learning techniques into Terrier, as well as how different query formulations (obtained from *rewriters*) can be integrated within a learning-to-rank setting. In terms of effectiveness,

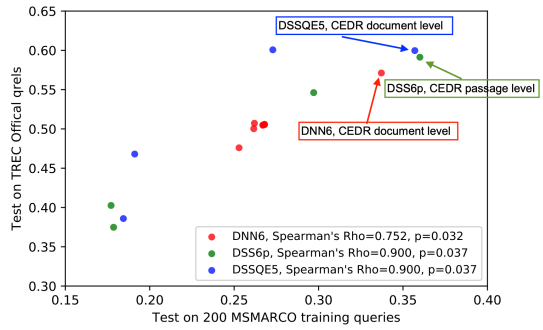
[7] No TREC Best and Median results were provided for NDCG@10.

Table 4: Performance of features within each first phase & feature set setting.

Feature ↓	First Phase & Feature set								
	NN6			SS6p			SSQE5		
	MAP	P@10	NDCG@1000	MAP	P@10	NDCG@1000	MAP	P@10	NDCG@1000
1	**0.2943**	0.5837	**0.5482**	**0.2857**	0.6116	0.5014	**0.3434**	**0.6930**	**0.5549**
2	0.1915	0.5744	0.4939	0.1944	0.4233	0.4037	0.2076	0.4349	0.4347
3	0.2716	0.5837	0.5358	0.2003	0.4356	0.4239	0.2279	0.4698	0.4639
4	0.2935	0.5837	0.5475	-	-	-	-	-	-
5	0.2936	0.5860	0.5474	0.2246	0.4977	0.4390	0.2624	0.5791	0.4975
6	0.2739	0.5442	0.5319	-	-	-	-	-	-
7	0.2935	0.5837	0.5475	-	-	-	-	-	-
8	0.2478	**0.6302**	0.5432	-	-	-	0.3114	0.6814	0.5434
9	-	-	-	0.2784	**0.6605**	**0.5022**	-	-	-



(a) Scatterplot for MAP



(b) Scatterplot for NDCG@10

Figure 3: Scatterplot showing the performances of the features in Table 4, as obtained on from the MSMARCO training queries (x-axis), and the official TREC qrels (y-axis).

our most effective run `uogTrDNN6LM` outperformed the TREC median, but we observed that a simple DFR-based query expansion run could be more effective. Moreover, some of our learning-to-rank runs significantly diverged from their expected performances. This emphasises the difficulty in learning effective models for adhoc retrieval tasks using training datasets with very few judgements.

ACKNOWLEDGEMENTS

The co-authors acknowledge the assistance of Zaiqiao Meng in deep learning, as well as insights from Richard McCreadie and Jeff Dalton in supporting our participation.

REFERENCES

[1] Gianni Amati. 2003. *Probabilistic Models for Information Retrieval based on Divergence from Randomness.* Ph.D. Dissertation. Department of Computing Science, University of Glasgow.
[2] Gianni Amati, Giuseppe Amodeo, Marco Bianchi, Carlo Gaibisso, and Giorgio Gambosi. 2008. FUB, IASI-CNR and University of Tor Vergata at TREC 2008 Blog Track.. In *Proceedings of TREC 2008.* 248–254.
[3] Gianni Amati and C.J. van Rijsbergen. 2002. Probabilistic models of information retrieval based on measuring the divergence from randomness. *ACM Transactions on Information Systems (TOIS)* 20, 4 (2002), 357–389.
[4] Nima Asadi and Jimmy J. Lin. 2013. Document vector representations for feature extraction in multi-stage document ranking. *Inf. Retr.* 16, 6 (2013), 747–768.
[5] Christopher J.C. Burges. 2010. *From RankNet to LambdaRank to LambdaMART: An Overview.* Technical Report MSR-TR-2010-82.
[6] Nick Craswell, Bhaskar Mitra, Daniel Campos, and Emine Yilmaz. 2020. Overview of the TREC 2019 Deep Learning Track. In *Proceedings of TREC 2019.*
[7] Zhuyun Dai and Jamie Callan. 2019. Deeper Text Understanding for IR with Contextual Neural Language Modeling. In *Proceedings of SIGIR.* 985–988.
[8] 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).
[9] Hui Fang and ChengXiang Zhai. 2006. Semantic Term Matching in Axiomatic Approaches to Information Retrieval. In *Proceedings of SIGIR 2006.* 115–122.
[10] 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.
[11] Jiafeng Guo, Yixing Fan, Xiang Ji, and Xueqi Cheng. 2019. MatchZoo: A Learning, Practicing, and Developing System for Neural Text Matching. In *Proceedings of SIGIR.* 1297–1300.
[12] Ben He and Iadh Ounis. 2005. Term frequency normalisation tuning for BM25 and DFR models. In *Proceedings of ECIR.* 200–214.
[13] Sean MacAvaney, Andrew Yates, Arman Cohan, and Nazli Goharian. 2019. CEDR: Contextualized Embeddings for Document Ranking. In *Proceedings of SIGIR.*
[14] Craig Macdonald, Ben He, Vassilis Plachouras, and Iadh Ounis. 2005. University of Glasgow at TREC 2005: Experiments in Terabyte and Enterprise tracks with Terrier. In *Proceedings of TREC 2005.* 181–193.
[15] Craig Macdonald, Richard McCreadie, Rodrygo Santos, and Iadh Ounis. 2012. From Puppy to Maturity: Experiences in Developing Terrier. In *Proceedings of the SIGIR Workshop on Open Source Information Retrieval.*
[16] Craig Macdonald, Rodrygo Santos, Iadh Ounis, and Ben He. 2013. About Learning Models with Multiple Query Dependent Features. *Transactions on Information Systems* (2013).
[17] Craig Macdonald and Nicola Tonellotto. 2017. Upper Bound Approximation for BlockMaxWand. In *Proceedings of SIGIR.* 273–276.
[18] 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.
[19] Donald Metzler and W. Bruce Croft. 2005. A Markov random field model for term dependencies. In *Proceedings of SIGIR.* 472–479.
[20] Donald A. Metzler. 2007. Automatic feature selection in the markov random field model for information retrieval. In *Proceedings of SIGIR.* 253–262.
[21] 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.
[22] 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.
[23] Peilin Yang and Jimmy Lin. 2019. Reproducing and Generalizing Semantic Term Matching in Axiomatic Information Retrieval. In *Proceedings of ECIR.* 369–381.