Anserini at TREC 2018: CENTRE, Common Core, and News Tracks

Peilin Yang and Jimmy Lin[1]
[1] David R. Cheriton School of Computer Science, University of Waterloo
yangpeilyn@gmail.com,jimmylin@uwaterloo.ca

1 INTRODUCTION

Anserini is an open-source information retrieval toolkit built on Lucene [3, 4]. The goal of our effort is to support information retrieval research using the popular open-source Lucene search library by allowing researchers to easily replicate results with modern ranking models on diverse test collections.

Although there are many open-source search engines developed and maintained by academic research groups, most of them are designed primarily to facilitate the publication of research papers, and as such, they often suffer from poor usability, incomplete documentation, and a host of other issues. The growing complexity of modern software ecosystems and the diverse capabilities that are required to build useful end-to-end search applications places academic research groups at a huge disadvantage relative to Lucene. Except for a handful of commercial web search engines that deploy custom infrastructure, Lucene has become the *de facto* platform in industry for building production search applications—used by organizations as diverse as Twitter, Reddit, Bloomberg, and Target. It has an active developer base, diverse features and capabilities, and lies at the center of a vibrant ecosystem. However, Lucene lacks systematic support for information retrieval research—in particular, *ad hoc* experimentation using standard test collections. This is where Anserini comes in: we enable cutting-edge information retrieval research using Lucene.

At TREC 2018, we participated in the CENTRE, Common Core, and News Tracks. Each is described in its own section below. Our development efforts centered around the v0.1.0 release of Anserini, which is based on Lucene 6.3.0 (not the latest release).

2 CENTRE TRACK

To accomplish the objectives discussed in the introduction, we are engaged in an ongoing effort to reproduce effective retrieval ranking models in Anserini, which will provide researchers a solid foundation to build on. For TREC, our primary objective was to reimplement axiomatic semantic term matching [1] in our toolkit. Although the paper of Yang and Hui [2], which applied the technique to web collections, was selected as one of the reproduction targets of the CENTRE Track, we were not particularly interested in optimizing for the track's metrics (i.e., Root Mean Square Error with respect to the original submission). Since the original implementation was in Indri, there will inevitably be differences in document processing (e.g., tokenization, stopwords, stemming, etc.) that, while important, are not interesting from the perspective of reproducibility.

In a recent policy, ACM defines a number of terms related to reproducibility as follows:[1] repeatable ("a researcher can reliably repeat her own computation"), replicable ("an independent group can obtain the same result using the author's own artifacts"), and reproducible ("an independent group can obtain the same result using artifacts which they develop completely independently"). Our work might be characterized as "self-reproducibility"; although one of the authors of the original paper led the Anserini implementation effort, we go beyond repeatability because a completely different codebase was developed from scratch. Because of this, and the fact that we had access to the original code, we did not encounter any challenges during the process. To facilitate future reuse of the Anserini implementation, we have built an extensive regression framework that ensures others will be able to repeat our results.

Although the derivations of the models underlying Yang and Hui [2] are couched within the framework of axiomatic retrieval, the approach is operationalized as query expansion: A "working set" is constructed from an initial ranking, from which expansion terms are computed. These expansion terms are then combined with the original query to yield the final ranking in a second retrieval stage. This procedure means that semantic term matching can be applied to any "base" ranking model. That is, we can use any ranking function to aid in the construction of the working set (i.e., for gathering pseudo-relevant documents), and use the same ranking function (or even a different one) for the expanded query. Our implementation in Anserini makes such explorations easy.

Another feature of our implementation worth mentioning concerns how the working set is populated. In the original formulation, non-relevant documents are sampled from a particular source, and the non-determinism of this sampling process makes runs non-repeatable. We address this by allowing the developer to supply a specific random seed as a configuration parameter. Repeatability can be ensured as long as these seeds are carefully recorded and tracked (just like any other experimental setting).

Yang and Hui [2] described different combinations of target collections and sources for building the working sets. The target collection is what we search against for the initial and expanded query, and the composition of the working set affects the score of the expansion terms. One might think that the target collection is simply the collection for the task, in this case, ClueWeb12. However, this was, in fact, *not* what Yang and Hui did in their TREC 2013 experiments. Due to reasons of computational efficiency, they first created a smaller sub-collection from ClueWeb12, which then served as the target collection on which subsequent experiments were performed. To quote:

> [W]e first use Indri's default language model to retrieve 10,000 top ranked documents for each query and then filter out the documents that have spam score less than -130. The filtered documents are used to build a much smaller index.

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

	Target Collection	Working Set	Comments
1	CW12Lite2013	CW12Lite2013	directly comparable to UDInfolabWEB1
2	CW12Lite2013	SP13	directly comparable to UDInfolabWEB2
3	CW12Lite2013	Wiki	
4	CW12Lite2018	CW12Lite2018	similar to UDInfolabWEB1, but using Anserini to create sub-collection
5	CW12Lite2018	SP18	similar to UDInfolabWEB2, but using Anserini to create sub-collection + 2018 web snippets
6	CW12Lite2018	Wiki	
7	CW12	CW12	similar to UDInfolabWEB1, but without creating sub-collection first
8	CW12	SP13	similar to UDInfolabWEB2, but without creating sub-collection first
9	CW12	SP18	similar to above, except with 2018 web snippets
10	CW12	Wiki	

Table 1: Different possible combinations of target collections and working sets for axiomatic semantic term matching.

Run	Description	AP	NDCG@20	ERR@20
Anserini-UDInfolabWEB1-1	#7, using BM25 and $\beta = 0.5$	0.1329	0.19754	0.20112
Anserini-UDInfolabWEB1-2	#9, using BM25 and $\beta = 0.5$	0.2172	0.25322	0.23460
Anserini-UDInfolabWEB1-3	#10, using BM25 and $\beta = 0.5$	0.1233	0.18550	0.16617
Anserini-UDInfolabWEB2-1	#1, using QL and $\beta = 1.7$	0.0880	0.15602	0.15770
Anserini-UDInfolabWEB2-2	#4, using QL and $\beta = 1.7$	0.1096	0.16391	0.18595
Anserini-UDInfolabWEB2-3	#6, using BM25 and $\beta = 0.5$	0.1119	0.17599	0.16384

Table 2: Submitted runs to the CENTRE Track.

This reduced sub-collection is referred to as CW12Lite2013. Fortunately, we still have access to the exact docids that comprise this sub-collection, and hence can attempt to replicate results based on it. Using this sub-collection as the target collection, we experimented with three different working sets: using CW12Lite2013 itself, using web snippets from 2013 (which we call SP13),[2] and using a Wikipedia dump from June 20, 2018 (which we call Wiki). These three combinations are shown as the first three rows of Table 1 (rows numbered 1 to 3). Using each combination, we can then apply our implementation of axiomatic semantic term matching from Anserini. In particular, the first two conditions are directly comparable to the UDInfolabWEB1 and UDInfolabWEB2 runs, respectively, expect with our Anserini implementation.

Although first creating a sub-collection was primarily an efficiency optimization made in TREC 2013, we were interested in the effects of sub-collection creation with more modern tools. To create what we call the CW12Lite2018 target collection, we used Lucene's query likelihood with Dirichlet smoothing implementation to fetch the top 10,000 documents (from all 50 queries). We did not, however, perform spam score filtering, as in CW12Lite2013. Using this sub-collection as the target collection, we experimented with the combinations shown in the middle three rows in Table 1 (rows numbered 4 to 6). We also crawled new web snippets in July 2018, which we call SP18.

Finally, we eliminated the sub-collection creation step and used the entire ClueWeb12 collection as the target. With vastly improved hardware since 2013, Lucene makes such experiments practical; we are able to build a single monolithic index over the entire collection (without partitioning) and run experiments with acceptable (albeit non-interactive) latencies. Working set combinations with all of ClueWeb12 as the target collection (CW12) is shown in the four rows on the bottom in Table 1 (rows numbered 7 to 10).

Given these combinations, we have one more variable in our experimental design: the retrieval model for both gathering the initial ranked list and for retrieval using the expanded query. The TREC 2013 experiments used F2Log, but here we additionally tried BM25 and query likelihood (QL) with Dirichlet smoothing.

Out of all the possible experimental conditions, we selected the six shown in Table 2 as our final CENTRE Track submissions. Detailed commands for replicating these runs, including building the index from scratch and generating the run files, are documented in a runbook checked into the Anserini code repository.[3]

We used the newly released qrels file to evaluate the submitted runs and the results are presented in Table 2.

3 COMMON CORE TRACK

We submitted a total of ten runs to the Common Core Track, described as follows:

- anserini_bm25: BM25 baseline.
- anserini_sdm: BM25 + sequential dependence model.
- anserini_rm3: BM25 + RM3.

[2]Fortunately, we have retained the web snippets from 2013.

[3]Available from the main Anserini repo, http://anserini.io; given the impermanence of web links, we strive to keep Anserini documentation up to date and intuitively organized in lieu of providing a specific URL.

Run	AP	NDCG	P@10	Pool
anserini_bm25	0.2284	0.5064	0.4500	Y
anserini_sdm	0.2364	0.5127	0.4860	Y
anserini_rm3	0.2680	0.5422	0.4680	Y
anserini_ax	0.2734	**0.5582**	**0.4960**	Y
anserini_ax17	0.2059	0.4942	0.4060	Y
anserini_ql	0.2294	0.5059	0.4660	N
anserini_qlsdm	0.2326	0.5071	0.4740	N
anserini_qlrm3	0.2501	0.5359	0.4660	N
anserini_qlax	**0.2749**	0.5484	0.4780	N
anserini_qlax17	0.2039	0.4875	0.4280	N
Median (Auto)	0.1745	0.4483	0.3340	-
Max (Auto)	0.3958	0.6849	0.7280	-
Median (All)	0.2112	0.4925	0.4140	-
Max (All)	0.5353	0.7792	0.8340	-

Table 3: Anserini results in the Common Core Track, compared to summary statistics provided by NIST.

- `anserini_ax`: BM25 + axiomatic semantic term matching, with working set from the Washington Post collection itself.
- `anserini_ax17`: BM25 + axiomatic semantic term matching, with working set from the New York Times collection used in Core17.
- `anserini_ql`: QL baseline.
- `anserini_qlsdm`: QL + sequential dependence model.
- `anserini_qlrm3`: QL + RM3.
- `anserini_qlax`: QL + axiomatic semantic term matching, with working set from the Washington Post collection itself.
- `anserini_qlax17`: QL + axiomatic semantic term matching, with working set from the New York Times collection used in Core17.

A detailed runbook checked into our Anserini code repository describes how these runs can be replicated.[4]

The effectiveness of our submitted runs in terms of standard retrieval metrics are shown in Table 3. The final column indicates whether or not the run contributed to the judgment pools. The final four rows in the table show mean of per topic median and max scores across two different categories of runs: automatic runs (Auto) and all runs (All). These values are based on summary statistics provided by NIST.

4 NEWS TRACK

We participated in the background linking task of the News Track, submitting five runs. Our main focus was to explore different approaches to constructing a keyword query from a query article.

- `anserini_1000w`: Each term in the query article is assigned a weight equal to its *tf-idf* score. The query is constructed by selecting up to 1000 terms based on these scores. The selected terms comprise a weighted query for searching the collection using BM25 to generate the final ranked list.
- `anserini_nsdm`: We selected the first 1000 terms from the query article and used those as the "query" to search the collection using the sequential dependence model with BM25. Due to limits on the number of clauses in a Lucene query, we were not able to use the entire document as the query.
- `anserini_nax`: We ran BM25 with axiomatic semantic term matching using the query article as the "query". That is, we constructed an initial query using up to the first 1000 terms from the query article *without* weights. BM25 is used to perform an initial retrieval, and then axiomatic semantic term matching is used to select up to 20 expansion terms. The expanded query (original terms from the query article and the expansion terms) is used to search the collection with BM25 to produce the final ranked list, with $\beta = 0.4$ to control the importance of the semantic term match weights.
- `anserini_sdmp`: Instead of treating the entire query article as a single query, we selected the five longest paragraphs and formed five individual queries, each comprising up to 1000 terms (from the paragraph). These were used as input to the sequential dependence model to query the collection using BM25. The final ranking is generated by selecting results of the paragraph queries in a round-robin fashion.
- `anserini_axp`: Similar to `anserini_sdmp`, but we used BM25 with axiomatic semantic term matching for each of the individual paragraph queries. We selected up to 20 expansion terms (each), with $\beta = 0.4$ to control the importance of the semantic term match weights.

All of our background linking runs contained a post-processing step to eliminate duplicate articles. Results from the evaluation are shown in Table 4. It appears that our simplest approach of selecting terms from the query article to form a weighted query is the most effective in terms of mean NDCG@5.

	NDCG@5	
Run	mean	median
anserini_1000w	**0.3529**	0.3433
anserini_nsdm	0.3347	**0.3456**
anserini_nax	0.2948	0.2402
anserini_sdmp	0.3271	0.3435
anserini_axp	0.3030	0.2832

Table 4: Anserini results in the background linking task of the News Track, compared to summary statistics provided by NIST.

REFERENCES

[1] Hui Fang and ChengXiang Zhai. 2006. Semantic Term Matching in Axiomatic Approaches to Information Retrieval. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2006).* Seattle, Washington, 115–122.

[2] Peilin Yang and Hui Fang. 2013. Evaluating the Effectiveness of Axiomatic Approaches in Web Track. In *Proceedings of the Twenty-Second Text REtrieval Conference (TREC 2013).* Gaithersburg, Maryland.

[3] Peilin Yang, Hui Fang, and Jimmy Lin. 2017. Anserini: Enabling the Use of Lucene for Information Retrieval Research. In *Proceedings of the 40th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2017).* Tokyo, Japan, 1253–1256.

[4] Peilin Yang, Hui Fang, and Jimmy Lin. 2018. Anserini: Reproducible Ranking Baselines Using Lucene. *Journal of Data and Information Quality* 10, 4 (2018), Article 16.

[4]See similar note with respect to the CENTRE runbook.