Which BM25 Do You Mean? A Large-Scale Reproducibility Study of Scoring Variants

Chris Kamphuis[1], Arjen P. de Vries[1], Leonid Boytsov[2], and Jimmy Lin[3](✉)

[1] Radboud University, Nijmegen, The Netherlands
[2] Carnegie Mellon University, Pittsburgh, USA
[3] University of Waterloo, Waterloo, Canada
jimmylin@uwaterloo.ca

Abstract. When researchers speak of BM25, it is not entirely clear which variant they mean, since many tweaks to Robertson et al.'s original formulation have been proposed. When practitioners speak of BM25, they most likely refer to the implementation in the Lucene open-source search library. Does this ambiguity "matter"? We attempt to answer this question with a large-scale reproducibility study of BM25, considering eight variants. Experiments on three newswire collections show that there are no significant effectiveness differences between them, including Lucene's often maligned approximation of document length. As an added benefit, our empirical approach takes advantage of databases for rapid IR prototyping, which validates both the feasibility and methodological advantages claimed in previous work.

Keywords: Scoring functions · Relational databases

1 Introduction

BM25 [8] is perhaps the most well-known scoring function for "bag of words" document retrieval. It is derived from the binary independence relevance model to include within-document term frequency information and document length normalization in the probabilistic framework for IR [7]. Although learning-to-rank approaches and neural ranking models are widely used today, they are typically deployed as part of a multi-stage reranking architecture, over candidate documents supplied by a simple term-matching method using traditional inverted indexes [1]. Often, this is accomplished using BM25, and thus this decades-old scoring function remains a critical component of search applications today.

As many researchers have previously observed, e.g., Trotman et al. [11], the referent of BM25 is quite ambiguous. There are, in fact, many variants of the scoring function: beyond the original version proposed by Robertson et al. [8], many variants exist that include small tweaks by subsequent researchers. Also, researchers using different IR systems report (sometimes quite) different effectiveness measurements for their implementation of BM25, even on the same

© Springer Nature Switzerland AG 2020

test collections; consider for example the results reported in OSIRRC 2019, the open-source IR replicability challenge at SIGIR 2019 [2]. Furthermore, BM25 is parameterized in terms of k_1 and b (plus k_2, k_3 in the original formulation), and researchers often neglect to include the parameter settings in their papers.

Our goal is a large-scale reproducibility study to explore the nuances of different variants of BM25 and their impact on retrieval effectiveness. We include in our study the specifics of the implementation of BM25 in the Lucene open-source search library, a widely-deployed variant "in the real world". Outside of a small number of commercial search engine companies, Lucene—either stand-alone or via higher-level platforms such as Solr and Elasticsearch—has today become the *de facto* foundation for building search applications in industry.

Our approach enlists the aid of relational databases for rapid prototyping, an idea that goes back to the 1990s and was more recently revived by Mühleisen et al. [6]. Adding or revising scoring functions in any search engine requires custom code within some framework for postings traversal, making the exploration of many different scoring functions (as in our study) a tedious and error-prone process. As an alternative, it is possible to "export" the inverted index to a relational database and recast the document ranking problem into a database (specifically, SQL) query. Varying the scoring function, then, corresponds to varying the expression for calculating the score in the SQL query, allowing us to explore different BM25 variants by expressing them declaratively (instead of *programming* imperatively). We view our work as having two contributions:

- We conducted a large-scale reproducibility study of BM25 variants, focusing on the Lucene implementation and variants described by Trotman et al. [11]. Their findings are confirmed: effectiveness differences in IR experiments are unlikely to be the result of the choice of BM25 variant a system implemented.
- From the methodological perspective, our work can be viewed as reproducing and validating the work of Mühleisen et al. [6], the most recent advocate of using databases for rapid IR prototyping.

2 BM25 Variants

Table 1 summarizes the scoring functions of the BM25 variants we examined:

Robertson et al. [8] is the original formulation of BM25: N is the number of documents in the collection, df_t is the number of documents containing term t, tf_{td} is the term frequency of term t in document d. Document lengths L_d and L_{avg} are the number of tokens in document d and the average number of tokens in a document in the collection, respectively. Finally, k_1 and b are free parameters that can be optimized per collection.[1]

Lucene (default) is the variant implemented in Lucene (as of version 8), which introduces two main differences. First, since the IDF component of

[1] The original publication adds scoring components with constants k_2 and k_3 that are rarely used and thus not considered in our study.

Table 1. Scoring functions of the BM25 variants examined in this work.

Robertson et al.	$\sum_{t \in q} \log\left(\frac{N-df_t+0.5}{df_t+0.5}\right) \cdot \frac{tf_{td}}{k_1 \cdot \left(1-b+b \cdot \left(\frac{L_d}{L_{avg}}\right)\right)+tf_{td}}$
Lucene (default)	$\sum_{t \in q} \log\left(1+\frac{N-df_t+0.5}{df_t+0.5}\right) \cdot \frac{tf_{td}}{k_1 \cdot \left(1-b+b \cdot \left(\frac{L_{dlossy}}{L_{avg}}\right)\right)+tf_{td}}$
Lucene (accurate)	$\sum_{t \in q} \log\left(1+\frac{N-df_t+0.5}{df_t+0.5}\right) \cdot \frac{tf_{td}}{k_1 \cdot \left(1-b+b \cdot \left(\frac{L_d}{L_{avg}}\right)\right)+tf_{td}}$
ATIRE	$\sum_{t \in q} \log\left(\frac{N}{df_t}\right) \cdot \frac{(k_1+1) \cdot tf_{td}}{k_1 \cdot \left(1-b+b \cdot \left(\frac{L_d}{L_{avg}}\right)\right)+tf_{td}}$
BM25L	$\sum_{t \in q} \log\left(\frac{N+1}{df_t+0.5}\right) \cdot \frac{(k_1+1) \cdot (c_{td}+\delta)}{k_1+c_{td}+\delta}$
BM25+	$\sum_{t \in q} \log\left(\frac{N+1}{df_t}\right) \cdot \left(\frac{(k_1+1) \cdot tf_{td}}{k_1 \cdot \left(1-b+b \cdot \left(\frac{L_d}{L_{avg}}\right)\right)+tf_{td}}+\delta\right)$
BM25-adpt	$\sum_{t \in q} G_q^1 \cdot \frac{\left(k_1'+1\right) \cdot tf_{td}}{k_1' \cdot \left(1-b+b \cdot \left(\frac{L_d}{L_{avg}}\right)\right)+tf_{td}}$
$\mathrm{TF}_{l \circ \delta \circ p} \times \mathrm{IDF}$	$\sum_{t \in q} \log\left(\frac{N+1}{df_t}\right) \cdot \left(1+\log\left(1+\log\left(\frac{tf_{td}}{1-b+b \cdot \left(\frac{L_d}{L_{avg}}\right)}+\delta\right)\right)\right)$

Robertson et al. is negative when $df_t > N/2$, Lucene adds a constant one before calculating the log value. Second, the document length used in the scoring function is compressed (in a lossy manner) to a one byte value, denoted L_{dlossy}. With only 256 distinct document lengths, Lucene can pre-compute the value of $k_1 \cdot (1 - b + b \cdot (L_{dlossy}/L_{avg}))$ for each possible length, resulting in fewer computations at query time.

Lucene (accurate) represents our attempt to measure the impact of Lucene's lossy document length encoding. We implemented a variant that uses exact document lengths, but is otherwise identical to the Lucene default.

ATIRE [10] implements the IDF component of BM25 as $\log(N/df_t)$, which also avoids negative values. The TF component is multiplied by $k_1 + 1$ to make it look more like the classic RSJ weight; this has no effect on the resulting ranked list, as all scores are scaled linearly with this factor.

BM25L [5] builds on the observation that BM25 penalizes longer documents too much compared to shorter ones. The IDF component differs, to avoid negative values. The TF component is reformulated as $((k_1 + 1) \cdot c_{td}) / (k_1 + c_{td})$ with $c_{td} = tf_{td}/(1 - b + b \cdot (L_d/L_{avg}))$. The c_{td} component is further modified by adding a constant δ to it, boosting the score for longer documents. The authors report using $\delta = 0.5$ for highest effectiveness.

BM25+ [4] encodes a general approach for dealing with the issue that ranking functions unfairly prefer shorter documents over longer ones. The proposal is to add a lower-bound bonus when a term appears at least one time in a document. The difference with BM25L is a constant δ to the TF component. The IDF component is again changed to a variant that disallows negative values.

BM25-adpt [3] is an approach that varies k_1 per term (i.e., uses term specific k_1 values). In order to determine the optimal value for k_1, the method starts by identifying the probability of a term occurring at least once in a document as $(df_r + 0.5)/(N + 1)$. The probability of the term occurring one more time is then defined as $(df_{r+1} + 0.5)/(df_r + 1)$. The information gain of a term occurring $r + 1$ instead of r times is defined as $G_q^r = \log_2 ((df_{r+1} + 0.5)/(df_r + 1)) - \log_2 ((df_{tr} + 0.5)/(N + 1))$, where df_r is defined as follows: $|D_{t|c_{td} \geq r-0.5}|$ if $r > 1$, df_t if $r = 1$, and N if $r = 0$ (c_{td} is the same as in BM25L). The information gain is calculated for $r \in \{0, \ldots, T\}$, until $G_q^r > G_q^{r+1}$. The optimal value for k_1 is then determined by finding the value for k_1 that minimizes the equation $k_1^{'} = \mathrm{argmin}_{k_1} \sum_{r=0}^{T} \left(G_q^r/G_q^1 - ((k_1 + 1) \cdot r)/(k_1 + r)\right)^2$. Essentially, this gives a value for k_1 that maximizes information gain for that specific term; $k_1^{'}$ and G_q^1 are then plugged into the BM25-adpt formula.

We found that the optimal value of $k_1^{'}$ is actually not defined for about 90% of the terms. A unique optimal value for $k_1^{'}$ only exists when $r > 1$ while calculating G_q^r. For many terms, especially those with a low df, $G_q^r > G_q^{r+1}$ occurs before $r > 1$. In these cases, picking different values for k_1 has virtually no effect on retrieval effectiveness. For undefined values, we set $k_1^{'}$ to 0.001, the same as Trotman et al. [11].

TF$_{l \circ \delta \circ p}$×**IDF** [9] models the non-linear gain of a term occurring multiple times in a document as $1 + \log(1 + \log(tf_{td}))$. To ensure that terms occurring at least once in a document get boosted, the approach adds a fixed component δ, following BM25+. These parts are combined into the TF component using $tf_{td}/(1 - b + b \cdot (L_d/L_{avg}))$. The same IDF component as in BM25+ is used.

3 Experiments

Our experiments were conducted using Anserini (v0.6.0) on Java 11 to create an initial index, and subsequently using relational databases for rapid prototyping, which we dub "OldDog" after Mühleisen et al. [6]; following that work we use MonetDB as well. Evaluations with Lucene (default) and Lucene (accurate) were performed directly in Anserini; the latter was based on previously-released code that we updated and incorporated into Anserini.[2] The inverted index was exported from Lucene to OldDog, ensuring that all experiments share *exactly* the same document processing pipeline (tokenization, stemming, stopword removal, etc.). While exporting the inverted index, we precalculate all $k_1^{'}$ values for BM25-adpt as suggested by Lv and Zhai [3]. As an additional verification step, we implemented both Lucene (default) and Lucene (accurate) in OldDog and compared results to the output from Anserini. We are able to confirm that the results are the same, setting aside unavoidable differences related to floating point precision. All BM25 variants are then implemented in OldDog as minor variations upon the original SQL query provided in Mühleisen et al. [6]. The term-specific parameter optimization for the adpt variant was already calculated during the

[2] http://searchivarius.org/blog/accurate-bm25-similarity-lucene.

Table 2. Retrieval effectiveness.

	Robust04		Core17		Core18	
	AP	P@30	AP	P@30	AP	P@30
Robertson et al. [8]	.2526	.3086	.2094	.4327	.2465	**.3647**
Lucene (default)	.2531	.3102	.2087	.4293	.2495	.3567
Lucene (accurate)	.2533	.3104	.2094	.4327	.2495	.3593
ATIRE	.2533	.3104	.2094	.4327	.2495	.3593
BM25L	.2542	.3092	.1975	.4253	**.2501**	.3607
BM25+	.2526	.3071	.1931	.4260	.2447	.3513
BM25-adpt	**.2571**	**.3135**	**.2112**	.4133	.2480	.3533
$\mathrm{TF}_{l\circ\delta\circ p}\times\mathrm{IDF}$	.2516	.3084	.1932	**.4340**	.2465	**.3647**

index extraction stage, allowing us to upload the optimal (t, k) pairs and directly use the term-specific k values in the SQL query. The advantage of our experimental methodology is that we did not need to implement a single new ranking function from scratch. All the SQL variants implemented for this paper can be found on GitHub.[3]

The experiments use three TREC newswire test collections: TREC Disks 4 and 5, excluding Congressional Record, with topics and relevance judgments from the TREC 2004 Robust Track (Robust04); the New York Times Annotated Corpus, with topics and relevance judgments from the TREC 2017 Common Core Track (Core17); the TREC Washington Post Corpus, with topics and relevance judgments from the TREC 2018 Common Core Track (Core18). Following standard experimental practice, we assess ranked list output in terms of average precision (AP) and precision at rank 30 (P@30). The parameters shared by all models are set to $k_1 = 0.9$ and $b = 0.4$, Anserini's defaults. The parameter δ is set to the value reported as best in the corresponding source publication. Table 2 presents the effectiveness scores for the implemented retrieval functions on all three test collections.

All experiments were run on a Linux desktop (Fedora 30, Kernel 5.2.18, SELinux enabled) with 4 cores (Intel Xeon CPU E3-1226 v3 @ 3.30 GHz) and 16 GB of main memory; the MonetDB 11.33.11 server was compiled from source using the `--enable-optimize` flag. Table 3 presents the average retrieval time per query in milliseconds (without standard deviation for Anserini, which does not report time per query). MonetDB uses all cores for both inter- and intra-query parallelism, while Anserini is single-threaded.

The observed differences in effectiveness are very small and can be fully attributed to variations in the scoring function; our methodology fixes all other parts of the indexing pipeline (tag cleanup, tokenization, stopwords, etc.). Both an ANOVA and Tukey's HSD show no significant differences between any variant, on all test collections. This confirms the findings of Trotman et al. [11]:

[3] https://github.com/Chriskamphuis/olddog.

Table 3. Average retrieval time per query in ms: Anserini (top) and OldDog (bottom).

	Robust04	Core17	Core18
Lucene (default)	52	111	120
Lucene (accurate)	55	115	123
Robertson et al. [8]	158 ± 25	703 ± 162	331 ± 96
Lucene (default)	157 ± 24	699 ± 154	326 ± 90
Lucene (accurate)	157 ± 24	701 ± 156	324 ± 88
ATIRE	157 ± 24	698 ± 159	331 ± 94
BM25L	158 ± 25	697 ± 160	333 ± 96
BM25+	158 ± 25	700 ± 160	334 ± 96
BM25-adpt	158 ± 24	700 ± 157	330 ± 92
$\mathrm{TF}_{l\circ\delta\circ p}\times\mathrm{IDF}$	158 ± 24	698 ± 158	331 ± 96

effectiveness differences are unlikely an effect of the choice of the BM25 variant. Across the IR literature, we find that differences due to more mundane settings (such as the choice of stopwords) are often larger than the differences we observe here. Although we find no significant improvements over the original Robertson et al. [8] formulation, it might still be worthwhile to use a variant of BM25 that avoids negative ranking scores.

Comparing Lucene (default) and Lucene (accurate), we find negligible differences in effectiveness. However, the differences in retrieval time are also negligible, which calls into question the motivation behind the original length approximation. Currently, the similarity function and thus the document length encoding are defined at index time. Storing exact document lengths would allow for different ranking functions to be swapped at query time more easily, as no information would be discarded at index time. Accurate document lengths might additionally benefit downstream modules that depend on Lucene. We therefore suggest that Lucene might benefit from storing exact document lengths.

4 Conclusions

In summary, this work describes a double reproducibility study—we methodologically validate the usefulness of databases for IR prototyping claimed by Mühleisen et al. [6] and performed a large-scale study of BM25 to confirm the findings of Trotman et al. [11]. Returning to our original motivating question regarding the multitude of BM25 variants: "Does it matter?", we conclude that the answer appears to be "no, it does not".

Acknowledgements. This work is part of the research program Commit2Data with project number 628.011.001, which is (partly) financed by the NWO. Additional support was provided by the Natural Sciences and Engineering Research Council (NSERC) of Canada.

References

1. Asadi, N., Lin, J.: Effectiveness/efficiency tradeoffs for candidate generation in multi-stage retrieval architectures. In: Proceedings of the 36th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2013), pp. 997–1000, Dublin (2013)
2. Clancy, R., Ferro, N., Hauff, C., Lin, J., Sakai, T., Wu, Z.Z.: Overview of the 2019 Open-Source IR Replicability Challenge (OSIRRC 2019). In: CEUR Workshop Proceedings of the Open-Source IR Replicability Challenge (OSIRRC 2019) at SIGIR 2009, vol. 2409, Paris (2019)
3. Lv, Y., Zhai, C.: Adaptive term frequency normalization for BM25. In: Proceedings of the 20th ACM International Conference on Information and Knowledge Management (CIKM 2011), pp. 1985–1988, Glasgow (2011)
4. Lv, Y., Zhai, C.: Lower-bounding term frequency normalization. In: Proceedings of the 20th ACM International Conference on Information and Knowledge Management (CIKM 2011), pp. 7–16, Glasgow (2011)
5. Lv, Y., Zhai, C.: When documents are very long, BM25 fails! In: Proceeding of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2011), pp. 1103–1104, Beijing (2011)
6. Mühleisen, H., Samar, T., Lin, J., de Vries, A.: Old dogs are great at new tricks: column stores for IR prototyping. In: Proceedings of the 37th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2014), pp. 863–866, Gold Coast (2014)
7. Robertson, S., Zaragoza, H.: The probabilistic relevance framework: BM25 and beyond. Found. Trends Inf. Retrieval **3**(4), 333–389 (2009)
8. Robertson, S.E., Walker, S., Jones, S., Hancock-Beaulieu, M., Gatford, M.: Okapi at TREC-3. In: Proceedings of the 3rd Text Retrieval Conference (TREC-3), pp. 109–126, Gaithersburg (1994)
9. Rousseau, F., Vazirgiannis, M.: Composition of TF normalizations: new insights on scoring functions for ad hoc IR. In: Proceedings of the 36th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2013), pp. 917–920, Dublin (2013)
10. Trotman, A., Jia, X.F., Crane, M.: Towards an efficient and effective search engine. In: SIGIR 2012 Workshop on Open Source Information Retrieval, pp. 40–47, Portland (2012)
11. Trotman, A., Puurula, A., Burgess, B.: Improvements to BM25 and language models examined. In: Proceedings of the 2014 Australasian Document Computing Symposium (ADCS 2014), pp. 58–66, Melbourne (2014)