LexBoost: Improving Lexical Document Retrieval with Nearest Neighbors

Hrishikesh Kulkarni
Georgetown University
Washington, DC, USA
first@ir.cs.georgetown.edu

Nazli Goharian
Georgetown University
Washington, DC, USA
first@ir.cs.georgetown.edu

Ophir Frieder
Georgetown University
Washington, DC, USA
first@ir.cs.georgetown.edu

Sean MacAvaney
University of Glasgow
Glasgow, UK
first.last@glasgow.ac.uk

ABSTRACT

Sparse retrieval methods like BM25 are based on lexical overlap, focusing on the surface form of the terms that appear in the query and the document. The use of inverted indices in these methods leads to high retrieval efficiency. On the other hand, dense retrieval methods are based on learned dense vectors and, consequently, are effective but comparatively slow. Since sparse and dense methods approach problems differently and use complementary relevance signals, approximation methods were proposed to balance effectiveness and efficiency. For efficiency, approximation methods like HNSW are frequently used to approximate exhaustive dense retrieval. However, approximation techniques still exhibit considerably higher latency than sparse approaches. We propose LexBoost that first builds a network of dense neighbors (a corpus graph) using a dense retrieval approach while indexing. Then, during retrieval, we consider both a document's lexical relevance scores and its neighbors' scores to rank the documents. In LexBoost this remarkably simple application of the Cluster Hypothesis contributes to stronger ranking effectiveness while contributing little computational overhead (since the corpus graph is constructed offline). The method is robust across the number of neighbors considered, various fusion parameters for determining the scores, and different dataset construction methods. We also show that re-ranking on top of LexBoost outperforms traditional dense re-ranking and leads to results comparable with higher-latency exhaustive dense retrieval.

CCS CONCEPTS

• **Information systems → Retrieval models and ranking**.

KEYWORDS

Lexical Retrieval, Dense Retrieval, Corpus Graph

DocEng '24, August 20–23, 2024, San Jose, CA, USA

ACM ISBN 979-8-4007-1169-5/24/08
https://doi.org/10.1145/3685650.3685658

ACM Reference Format:
Hrishikesh Kulkarni, Nazli Goharian, Ophir Frieder, and Sean MacAvaney. 2024. LexBoost: Improving Lexical Document Retrieval with Nearest Neighbors. In *ACM Symposium on Document Engineering 2024 (DocEng '24), August 20–23, 2024, San Jose, CA, USA*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3685650.3685658

1 INTRODUCTION

Traditionally, lexical relevance methods like BM25 [32] are used for efficient retrieval. These methods consider the surface form of query and document terms and operate using term overlap, ranking documents with more occurrences of important terms ahead of others. Since the surface form of terms are the same regardless of their context within the document, lexical methods tend to miss highly relevant documents due to the vocabulary mismatch problem, limiting their effectiveness in both precision and recall. To overcome this limitation, dense methods were proposed; they go beyond word overlap by learning semantic representations of the documents. These methods utilize deep neural networks to learn the low-dimensional representations and match queries and documents in the low-dimension embedding space [44]. Dense and neural retrieval methods represent documents in vector space of predefined size. They successfully identify semantic closeness between the query and document, significantly boosting retrieval effectiveness.

An ideal information retrieval system would thus capitalize on the efficiency of lexical retrieval methods and the effectiveness of dense retrieval methods. The efficiency limitation of dense retrieval methods has led to a number of approximation approaches that efficiently 'approximate' the results of an exhaustive dense retrieval search, including HNSW [24], IVF [35, 46] and LADR [19]. These methods come close to a dense retriever and do so relatively efficiently. However, using a dense retrieval component still comes with substantially higher latency than lexical retrievers, thus only partially addressing the original problem of simultaneously achieving high effectiveness and efficiency.

We propose LexBoost that goes beyond lexical overlap by utilizing a document's neighborhood in dense retrieval space. Rather than directly combining sparse and dense scores, our approach estimates relevance by combining a document's lexical score with its neighbors' lexical scores. This is an application of the Cluster Hypothesis [17]: we use a dense model to identify document proximity

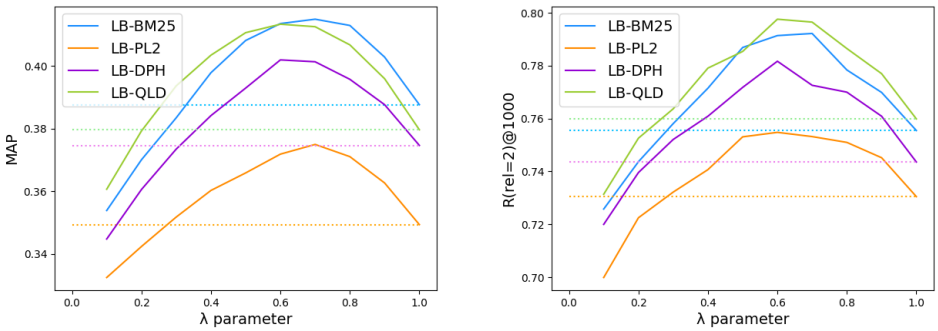

Figure 1: MAP and Recall(rel=2)@1000 for LexBoost on BM25, PL2, DPH, QLD - TREC DL 2019. The faint horizontal lines are respective baselines (i.e., $\lambda = 1$).

offline and a sparse model to estimate the relevance of a document (and its neighbors) online.

Figure 1 presents the improvements of LexBoost when used on top of lexical retrieval methods like BM25 [32], PL2 [3], DPH [3] and QLD [5]. Since the neighborhood is identified offline (i.e., at the indexing time), there is essentially no additional query-time latency overhead when using LexBoost as compared to existing lexical retrievers. The enriched relevance information by effectively using knowledge about the neighborhood and negligible additional latency overheads separates LexBoost from other retrieval approaches. In summary, our contributions are:

- We introduce LexBoost, which results in statistically significant improvements over the state-of-the-art lexical retrieval methods, including BM25.
- Our proposed method LexBoost introduces negligible additional latency overheads in achieving these statistically significant improvements.
- We conducted extensive experimentation using standard benchmark datasets in multiple domains and across a wide range of parameters and demonstrated improvements.

2 RELATED WORK

Information retrieval is the process of matching the query against information items and obtaining the most relevant piece of information. This process involves creation of an index which is an optimized data structure built on top of the information items for faster access [14]. A model (retrieval strategy) is an algorithm along with pertinent data structures which assigns similarity score to every query-document pair [14]. Further, document-document similarity has also been explored for various applications [43]. Some of the foundational models of information retrieval are Boolean model - built on binary relevance, Vector Space model - uses spatial distance as a similarity measure and Probabilistic model - estimates document relevance as a probability [22]. These models have heavily inspired the traditional language/vocabulary driven i.e. lexical information retrieval.

2.1 Lexical Methods

Traditionally, lexical methods based on word overlap were used [26]. Utilization of inverted index in this sparse retrieval leads to high efficiency due to the term to document mapping. Different methods of weighing and normalization [38] led to a range of Term Frequency Inverse Document Frequency (TFIDF) models. BM25 [32] is one of the most popular and effective formulation in sparse retrieval. BM25 is a bag-of-words retrieval function. Here the documents are ranked based on the query terms appearing in each document, regardless of their proximity within the document. BM25 can be viewed as a non-linear combination of three basic document attributes: term frequency, document frequency, and the length of the document [37]. Document length normalization and query term saturation are the key features in BM25 and hence it did not favor shorter or longer documents and mitigates the impact of excessively high term frequency unlike TFIDF. BM25 is also extended with prescription regarding how to combine additional fields in document description [31].

Divergence From Randomness (DFR) framework was proposed to build probabilistic term weighting schemes [3]. It consists of two divergence functions and one normalization function. Two of the best DFR framework models are PL2 (Poisson-Laplace with second normalization of term frequency) and DPH (hyper-geometric model Popper's normalization) [3]. These methods typically suffer from high vocabulary dependence [27]. Query Linear Combination and Relevant Documents (QLD) uses relevant documents of similar queries for expressing the query as a linear combination of existing queries [5]. Query expansion and pseudo relevance feedback offer resolve to some extent but come with certain latency overhead [8]. KL expansion [47], Rocchio [33], Relevance Modelling [25] and RM3 [1] are popular pseudo relevance feedback methods. Efficiency of lexical methods is exploited by using it as a first-stage document ranker for a short listed input to intricate dense retrieval and large language model based systems [18].

2.2 Dense Methods

Further, to improve the effectiveness of information retrieval systems neural-based approaches for semantic retrieval, such as those

utilizing CNNs and RNNs have been proposed. Here text documents and queries are represented in a continuous vector space. As a next step neural network based similarity measures are used for relevance calculation [16, 34]. Additionally, neural methods utilize learned non-linear representations of text data, resulting in significant retrieval performance improvements [34]. Further, BERT [13], GPT [6] and other transformer architectures [39], have been used to improve the ability of information retrieval systems to attend to important parts of the query and documents for matching.

Thus, dense retrieval methods focus on learning dense representations for documents and work on semantic level. This boosts the effectiveness by a great extent but at the same time an exhaustive search over all document vectors results in compromized efficiency. Mainly, two primary types of dense methods exist: interaction-based where interactions between words in queries are modelled and the other being representation-based where the model learns a single vector representation of the query [30]. TAS-B [15] and TCT-ColBERT-HNP [21] are some of the state-of-the-art result producing methods belonging to this category. Further, multi representations method exemplified by ColBERT require significant memory and pruning methods have been proposed to make it more efficient [2]. Also, better sampling strategies have been proposed to train more effective dense retrieval models [10]. Dense retrieval approaches have also been modified to perform entity-oriented document retrieval [9].

The advent of dense retrieval led to learned vector based pseudo relevance feedback models. Some of the popular ones include ColBERT PRF [42], ColBERT-TCT PRF [21] and ANCE PRF [45]. Models like CEQE (Contextualized Embeddings for Query Expansion) utilize query focused contextualized embedding vectors [28]. Even though term based and vector based pseudo relevance feedback help tackle vocabulary dependence to some extent, they come with latency overheads. This is where LexBoost differentiates from others.

2.3 Approximation Methods

Approximation methods approximate results of exhaustive dense retrieval efficiently for the purpose of enhanced efficiency. Even though they manage to approximate top results they lack in recall [19]. Re-ranking is the most popular approximation method where lexical method like BM25 is used to shortlist top n documents which are then re-ranked using costly dense retrieval methods. Here, the recall limitation is evident and its severity is determined by shortlisting parameter n. Other approximation methods can be classified into tree-based indexing, locality sensitive hashing and product-quantization-based and graph-based methods [20]. Popular approximation methods include HNSW [24], IVF [35, 46], GAR [23], LADR [19] etc. These methods approximate results of a full dense retriever relatively efficiently. But the presence of a dense retriever component still adds additional latency overhead.

2.3.1 Comparison with LADR. LexBoost uses corpus graph to re-rank the documents on the basis of scores assigned to their neighbors in the first-stage lexical retrieval. LADR uses corpus graph to identify additional potentially relevant documents to be re-ranked along with first-stage retrieval results by a dense retrieval method in the re-ranking stage. Hence, LexBoost and LADR both use a corpus graph but for different purposes and at different stages in the multi-stage retrieval pipeline. On the MS MARCO TREC DL 19 dataset [11], LADR results in MAP of 0.50 using seed documents from BM25. In the same setup, it results in MAP of 0.51 when using seed documents from the BM25->LexBoost pipeline. This shows that LexBoost also results in improved effectiveness when used in first-stage retrieval for approximation methods like LADR at virtually no additional latency.

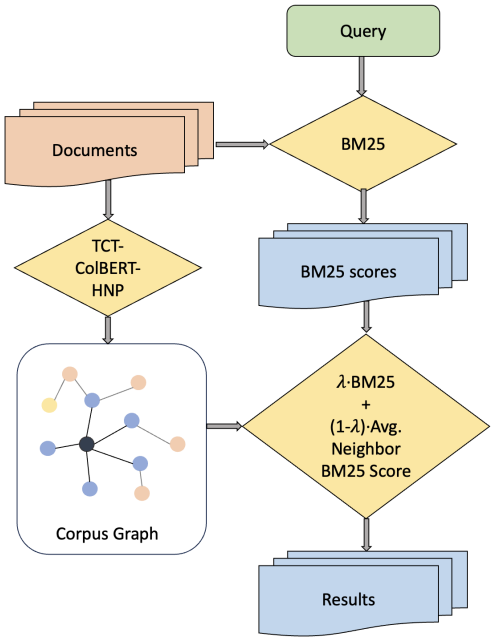

Figure 2: System Architecture

2.4 Hybrid Methods

As sparse and dense retrieval methods provide complementary results - hybrid methods were proposed [7, 29]. In any hybrid method, the lexical method and dense method output individual scores and ranked lists which are combined using various fusion formulations. Convex Combination of lexical and semantic scores and Reciprocal Rank Fusion of individual ranked lists are two of the most popular methods to combine output individual scores and ranked lists [7]. Convex Combination is more robust, agnostic to the choice of score normalization, has only one parameter and outperforms Reciprocal Rank Fusion [7]. Further, Convex Combination is also sample efficient with only a small set of examples required to tune the fusion parameter for the target domain [7].

These hybrid methods have a dense retrieval component and hence have very high latency. Thus, the effectiveness comes with latency overhead and across the approaches in literature high latency remains an important issue to be looked at. The key difference in our proposed method LexBoost is that the statistically significant improvement is delivered without invoking dense method at query time.

Algorithm 1 LexBoost

Require: q query, D document corpus, λ fusion parameter, n number of neighbors, G Corpus Graph
$R \leftarrow \{\}$
$S \leftarrow \text{LexicalRetrieval}(q, D)$ • retrieve from corpus
for d in S **do**
$N \leftarrow \text{Neighbors}(d, \text{top } n \text{ from } G)$ • get neighbors
$s \leftarrow \lambda \cdot \text{Lookup}(d, S) + \frac{1-\lambda}{n} \cdot \sum_{d_{neigh} \in N} \text{Lookup}(d_{neigh}, S)$
$R[d] \leftarrow s$ • save LexBoost doc score
end for
$R \leftarrow \text{Rank}(R)$ • rank docs

3 LEXBOOST

In LexBoost, we determine the final score of each document based on the lexical score (e.g., BM25) of the document and the lexical scores of its neighbors. The intuition behind the approach is derived from the Cluster Hypothesis [17], which states that documents that are near one another are likely to be relevant to the same query. Consequently, we consider a document to be more relevant if its nearest neighbors are also considered relevant. Since the network of neighbors (i.e., the corpus graph) can be constructed offline, the online costs of applying dense retrieval techniques are mitigated.

Thus, the insights come from dense retrieval method based similarity but without any latency overhead during retrieval. Hence we name our proposed approach LexBoost. We use Convex Combination for fusion of the two scores given its effectiveness [7]. We define a $\lambda \in [0, 1]$ parameter for fusion of the lexical method score and dense method based insight from the corpus graph. λ parameter is tuned using a validation set for optimal results.

Equation 1 shows the scoring formulation of LexBoost. Here, λ weight is given to the lexical score of the document in consideration and $1-\lambda$ weight is given to the mean lexical scores of the neighbors. In Equation 1, $d_1, d_2, ..., d_n \in N$ is the set of n nearest neighbors to the document in the corpus graph. The final ranking of the documents is established using the newly calculated LexBoost score for each document.

$$\text{LexBoost}(q, d, D) = \lambda \cdot \text{score}(q, d) + \frac{1-\lambda}{n} \cdot \sum_{d_{neigh} \in N} \text{score}(q, d_{neigh}) \quad (1)$$

Figure 2 depicts the system architecture of LexBoost. The corpus graph creation using dense retrieval method is done at indexing time and does not cause any latency overhead at retrieval. For every user query, first BM25 scores are obtained for the complete document corpus. Then as formulated in Equation 1, final LexBoost score is assigned to each document as a combination of the document score and mean neighbor score. LexBoost is formally described in Algorithm 1. Here, Lookup() retrieves the precomputed BM25 scores of the neighboring documents. Fusion parameter λ and number of neighbors to be considered n are the two key hyperparameters of the proposed method. The algorithm clearly shows that for each document, its score and neighbor scores are looked up and then final score is calculated using Equation 1.

4 EXPERIMENT

Through extensive experimentation, we address the following research questions:

- **RQ1:** Does the neighborhood of a document in a corpus graph provide comprehensive insights regarding relevance of the document to the query?
- **RQ2:** How does the neighbor score based ranking impact retrieval latency?
- **RQ3:** How does the effectiveness of the proposed method change with increase in the number of neighbors n?
- **RQ4:** How does the effectiveness of the proposed method change with variation in the fusion parameter λ?
- **RQ5:** Does the dataset construction method affect the performance of the proposed method?
- **RQ6:** Can the optimal fusion parameter be determined through training samples?

We have released the code to reproduce our results for respective research questions here[1].

4.1 Datasets and Measures

To validate the significance of the proposed method through experimentation, we use three publicly available benchmark datasets.

- **TREC 2019 Deep Learning (Passage Subtask).** This evaluation query set was made available for TREC 2019 Deep Learning shared task [11]. The document corpus is derived from MS MARCO [4]. It consists of 43 human-evaluated queries with comprehensive labeling using four relevance grades. This benchmark query set has on an average 215 relevance assessments per query.
- **TREC 2020 Deep Learning (Passage Subtask).** This evaluation query set was made available for TREC 2020 Deep Learning shared task [12]. The document corpus is derived from MS MARCO [4]. It consists of 54 queries with human judgments from NIST annotators. This benchmark query set has on an average 211 relevance assessments per query. Similar to TREC DL 2019, this too has relevance judgements on a four point scale.
- **CORD19/TREC-COVID.** Both clinicians and the public has been searching for relevant and reliable information related to COVID-19 since the pandemic. TREC COVID is a pandemic retrieval test collection built to create and test retrieval systems for COVID-19 and similar future events [40]. The document set used for this dataset is COVID-19 Open Research Dataset (CORD-19) [41]. Relevance judgements for the collection of 50 topics are determined by human annotators with biomedical expertise.

4.2 Models and Parameters

We construct the corpus graph by identifying 16 most similar documents using dense retrieval method TCT-ColBERT-HNP [21]. Constructing the corpus graph is a one-time process at indexing-stage.

[1] https://github.com/Georgetown-IR-Lab/LexBoost

The time complexity to build the corpus graph is $O(n^2)$, while the space complexity is $O(n)$. We consider 2, 4, 8 and 16 number of closest neighbors (n) for each document from the corpus graph for experimentation. This choice is ideal as the impact of significant variation in n on LexBoost performance can be studied. Primarily, the lexical retrieval method used to calculate initial score is BM25 [32]. We define λ parameter for fusion between initial BM25 scores and mean neighbor BM25 scores and evaluate for λ from 0 to 1 at regular intervals of 0.05. To evaluate the robustness of the LexBoost we also built corpus graph using TAS-B [15] and used it to identify neighbors.

We determine and tune the value of fusion parameter λ for Convex Combination through validation set given its sample efficiency [7]. We use TREC DL 19 query set as a validation set for determining optimal value of λ parameter. Then, we use this value for LexBoost on TREC DL 20 query set thus validating our optimization approach. We also evaluate LexBoost with a wide range of λ values on all three datasets to understand the impact it has on the nDCG, MAP and Recall evaluations.

4.3 Baselines and Implementation

To comprehensively study the performance improvements by LexBoost we compare it with different baseline methods. BM25 is our primary baseline. Further, we also considered PL2, DPH and QLD lexical retrieval methods as baselines to show that LexBoost is agnostic of lexical retrieval method used with it. As corpus graph construction is performed while indexing, it limits the latency overhead during retrieval. Hence, we can directly compare LexBoost Retrieval results with BM25 output in terms of effectiveness. We also show robustness of LexBoost by running experiments across wide range of parameters listed in the above section. Fusion parameter λ and number of neighbors considered n are the key hyperparameters. Additionally, we also compare LexBoost re-ranking with traditional re-ranking and exhaustive dense retrieval.

5 RESULTS AND ANALYSIS

We now discuss results and their analysis with respect to our research questions.

5.1 RQ1 and RQ2: Insights from corpus graph and impact on retrieval

As evident in Table 1, Table 2 and Table 3, we evaluated results across different datasets, a range of number of nearest neighbors from corpus graph and varying the fusion parameter. We observe statistically significant improvements in retrieval results across all these combinations. Figure 4 shows these improvements graphically for TREC DL 19 and TREC DL 20 query sets. We evaluated MAP, Recall at 1000 and nDCG at 10, 100 and 1000. Using LexBoost on BM25, MAP improved from 0.3877 to 0.415 and from 0.3609 to 0.3933 on TREC DL 19 and 20 respectively. Similarly, using LexBoost on BM25, MAP improved from 0.2525 to 0.2950 on the TREC COVID dataset. Further, we also observe that Recall@1000 improved from 0.7555 to 0.7922 and from 0.8046 to 0.8305 on TREC DL 19 and 20 respectively. Similarly, using LexBoost on BM25, Recall@1000 improved from 0.4429 to 0.5020 on the TREC COVID

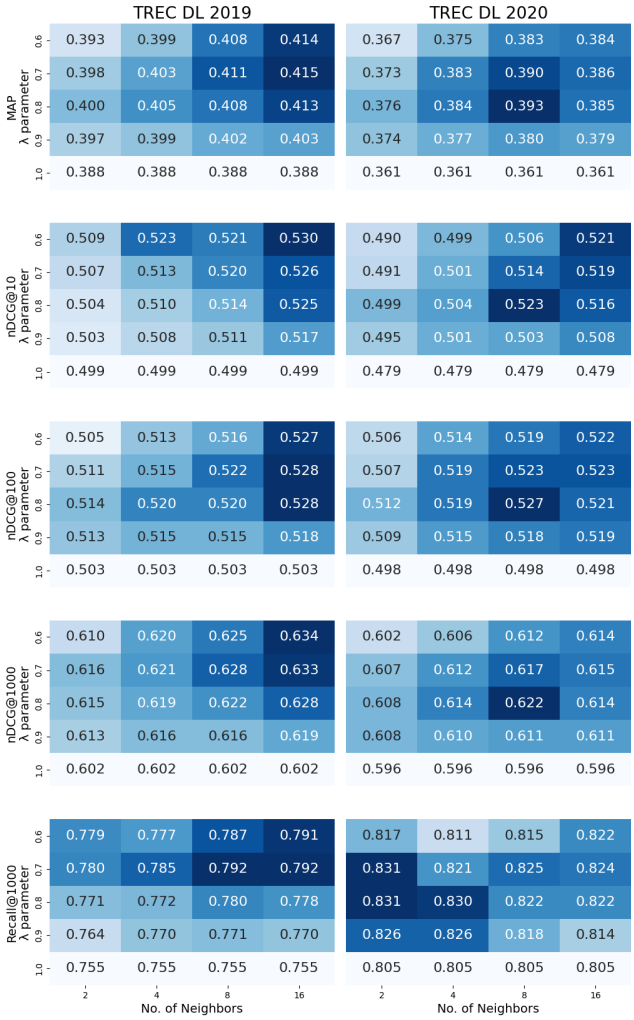

Figure 3: Heat-Maps showing impact of variation in fusion parameter λ and no. of neighbors n on LexBoost.

dataset. Similarly, statistically significant improvements were observed on nDCG@100 and nDCG@1000 with LexBoost across the three datasets under consideration.

To test the applicability of LexBoost, we evaluated it over four popular and effective lexical retrieval methods. As evident in Table 4, LexBoost shows statistically significant improvements over BM25, PL2, DPH and QLD baselines. Hence, we infer that LexBoost is robust and can be applied over a wide range of lexical retrieval methods. Additionally, we created the corpus graph using multiple dense retrieval methods namely: TCT-ColBERT-HNP and TAS-B. LexBoost shows statistically significant improvements using both corpus graphs when evaluated on TREC DL 19 and 20 as evident in Table 5. This shows that LexBoost is robust and agnostic to the dense method used to build the corpus graph. Further, we evaluate re-ranking using LexBoost. Here, we use a dense retrieval method on top of LexBoost to re-rank top 1000 documents. As evident in Table 6, we observe statistically significant improvements over traditional re-ranking pipelines. This experiment was

Table 1: LexBoost on TREC DL 2019. † denotes statistically significant improvement (paired t-test: $p \ddot{Y} 0.05$). Highest values denoted in bold. In LexBoost(k), k is the number of nearest neighbors.

λ parameter	Method	MAP	nDCG@10	nDCG@100	nDCG@1000	R(rel=2)@1000
	BM25	0.3877	0.4989	0.5028	0.6023	0.7555
0.7	LexBoost(2)	0.3977	0.5074	0.5112	0.6158†	0.7802†
	LexBoost(4)	0.4032†	0.5127	0.5151	0.621†	0.7847†
	LexBoost(8)	0.4107†	0.5204	0.5224	0.6278†	0.7918†
	LexBoost(16)	**0.415**†	0.526	0.5283†	**0.6332**†	**0.7922**†
0.8	LexBoost(2)	0.4†	0.504	0.5142	0.6149†	0.7707†
	LexBoost(4)	0.4047†	0.51	0.5204†	0.6189†	0.7724†
	LexBoost(8)	0.4078†	0.5143	0.5199†	0.6215†	0.7796†
	LexBoost(16)	0.413†	**0.5251**†	**0.5284**†	0.6277†	0.7784†
0.9	LexBoost(2)	0.3974†	0.503	0.5126†	0.6131†	0.764
	LexBoost(4)	0.3994†	0.508	0.5146†	0.6162†	0.7696†
	LexBoost(8)	0.4017†	0.5112	0.5146†	0.6159†	0.7705†
	LexBoost(16)	0.4029†	0.5171	0.5184†	0.6188†	0.7699†
0.95	LexBoost(2)	0.3926†	0.4985	0.5086†	0.6088†	0.7611
	LexBoost(4)	0.3946†	0.5015	0.5095†	0.6093†	0.7606
	LexBoost(8)	0.396†	0.5052	0.5118†	0.6116†	0.7634†
	LexBoost(16)	0.3958†	0.5091†	0.5117†	0.6104†	0.7616†

Table 2: LexBoost on TREC DL 2020. † denotes statistically significant improvement (paired t-test: $p \ddot{Y} 0.05$). Highest values denoted in bold. In LexBoost(k), k is the number of nearest neighbors.

λ parameter	Method	MAP	nDCG@10	nDCG@100	nDCG@1000	R(rel=2)@1000
	BM25	0.3609	0.4793	0.4984	0.5962	0.8046
0.7	LexBoost(2)	0.3726	0.4914	0.5069	0.6068	0.8311†
	LexBoost(4)	0.3829†	0.5007	0.5185†	0.6123†	0.821†
	LexBoost(8)	0.3897†	0.5135†	0.5229†	0.6175†	0.8247†
	LexBoost(16)	0.3862†	0.5188†	0.523†	0.6152†	0.8241
0.8	LexBoost(2)	0.3757†	0.4992	0.5116†	0.6082†	**0.8305**†
	LexBoost(4)	0.3836†	0.504†	0.5191†	0.6143†	0.8303†
	LexBoost(8)	**0.3933**†	**0.5225**†	**0.5272**†	**0.6218**†	0.8221†
	LexBoost(16)	0.3846†	0.5158†	0.5214†	0.6144†	0.8215†
0.9	LexBoost(2)	0.3738†	0.4946†	0.5095†	0.6082†	0.8263†
	LexBoost(4)	0.377†	0.5009†	0.5146†	0.6104†	0.8264†
	LexBoost(8)	0.3797†	0.5033†	0.5177†	0.6114†	0.8181†
	LexBoost(16)	0.3795†	0.5078†	0.5192†	0.6114†	0.8142†
0.95	LexBoost(2)	0.3695†	0.4899†	0.5075†	0.6044†	0.8171†
	LexBoost(4)	0.3712†	0.4926†	0.5081†	0.6054†	0.8153†
	LexBoost(8)	0.372†	0.4941†	0.5107†	0.6064†	0.8126†
	LexBoost(16)	0.3717†	0.4957†	0.5114†	0.6058†	0.8084

conducted on TREC DL 19 and 20 using TCT-ColBERT-HNP and TAS-B dense retrieval methods. Further, as evident in Figure 5 we compare LexBoost re-ranking with exhaustive dense retrieval. We do not consider exhaustive dense retrieval to be a baseline as it not equivalent to LexBoost re-ranking from latency point of view. In Figure 5, we notice that LexBoost re-ranking outperforms dense retrieval in MAP and show comparable performance in other metrics. This result is important as LexBoost re-ranking is significantly more efficient than exhaustive dense retrieval.

Most importantly, these improvements come with negligible additional latency overheads. The corpus graph is built and neighbors are identified while indexing the documents. Hence, LexBoost enables to use dense retrieval based document similarity insights

Table 3: LexBoost on TREC COVID. † denotes statistically significant improvement (paired t-test: $p \ddot{Y} 0.05$). Highest values denoted in bold. In LexBoost(k), k is the number of nearest neighbors.

λ parameter	Method	MAP	nDCG@10	nDCG@100	nDCG@1000	R(rel=2)@1000
	BM25	0.2525	0.6299	0.4821	0.4191	0.4429
0.4	LexBoost(2)	0.2645†	0.6472	0.4954	0.4319†	0.4584†
	LexBoost(4)	0.2781†	0.6257	0.5035†	0.4471†	0.4810†
	LexBoost(8)	0.2892†	0.6368	0.5162†	0.4578†	0.4915†
	LexBoost(16)	**0.2950**†	0.6408	0.5142†	**0.4654**†	**0.5017**†
0.5	LexBoost(2)	0.2691†	0.6490	0.4989†	0.4368†	0.4637†
	LexBoost(4)	0.2813†	0.6337	0.5075†	0.4494†	0.4815†
	LexBoost(8)	0.2901†	0.6402	0.5186†	0.4588†	0.4909†
	LexBoost(16)	0.2943†	0.6432	0.5192†	0.4630†	0.4963†
0.6	LexBoost(2)	0.2717†	0.6528	0.5022†	0.4402†	0.4673†
	LexBoost(4)	0.2815†	0.6367	0.5061†	0.4478†	0.4776†
	LexBoost(8)	0.2881†	0.6310	0.5177†	0.4552†	0.4852†
	LexBoost(16)	0.2909†	0.6483	**0.5203**†	0.4581†	0.4878†
0.7	LexBoost(2)	0.2718†	0.6483	0.5053†	0.4396†	0.4659†
	LexBoost(4)	0.2790†	0.6442	0.5083†	0.4449†	0.4720†
	LexBoost(8)	0.2833†	0.6444	0.5163†	0.4496†	0.4762†
	LexBoost(16)	0.2846†	0.6445	0.5182†	0.4509†	0.4788†
0.8	LexBoost(2)	0.2691†	0.6496	0.5048†	0.4365†	0.4612†
	LexBoost(4)	0.2735†	0.6444	0.5041†	0.4388†	0.4639†
	LexBoost(8)	0.2754†	0.6409	0.5093†	0.4408†	0.4657†
	LexBoost(16)	0.2760†	0.6465	0.5089†	0.4424†	0.4677†
0.9	LexBoost(2)	0.2632†	0.6539	0.4978†	0.4302†	0.4539†
	LexBoost(4)	0.2651†	**0.6567**†	0.5027†	0.4312†	0.4552†
	LexBoost(8)	0.2654†	0.6382	0.5019†	0.4317†	0.4566†
	LexBoost(16)	0.2654†	0.6448†	0.4990†	0.4320†	0.4568†

effectively during retrieval limiting latency overheads addressing RQ1 and RQ2.

5.2 RQ3: Robustness across number of neighbors considered

As evident in Table 1, Table 2 and Table 3, we evaluated LexBoost considering a range of nearest neighbors (2, 4, 8, 16) of the target document from the corpus graph. Neighbors are the documents in the corpus graph that are most similar to the target document. We observe statistically significant improvements irrespective of number of neighbors selected - with a general trend of higher improvements and better statistical significance with more neighbors in consideration. Thus, the increasing number of neighbors under consideration deliver better insights. Trends can be better understood with the heat-maps shown in Figure 3. The left set of heat-maps is for TREC DL 19 query set while the right set of heat-maps is for TREC DL 20 query set. The heat-maps depict combinations of fusion parameter λ and number of neighbors considered n. For each of the query set we have five heat-maps for the five metrics we are evaluating. Darker the shade of blue, better is the performance of LexBoost for that specific combination in that metric. As evident in Figure 3, the general trend is - with increase in the number of neighbors considered the performance improves. The bottom-most row in each heat-map is for fusion parameter $\lambda = 1$ which is equivalent to the baseline BM25. The significant improvements in performance across the variety of number of neighbors considered is also clearly evident by the drastic change in color between the bottom-most rows and others. This establishes robustness across the number of nearest neighbors considered from the corpus graph for LexBoost - hence addressing RQ3.

5.3 RQ4: Robustness across variation in fusion parameter λ

The fusion parameter λ decides the role played by neighbors in relevance of the target document to the user query. We evaluated LexBoost for fusion parameter λ values from 0 to 1 at regular intervals of 0.05 as can be seen in Figure 4. Here, the five plots are for five metrics under consideration. Part of LexBoost curve above respective baseline gives the range of λ parameter values leading to improvement in performance. Further, some of the λ values leading to highest improvements are evident in Table 1, Table 2 and Table 3. We evaluated for these values across three datasets

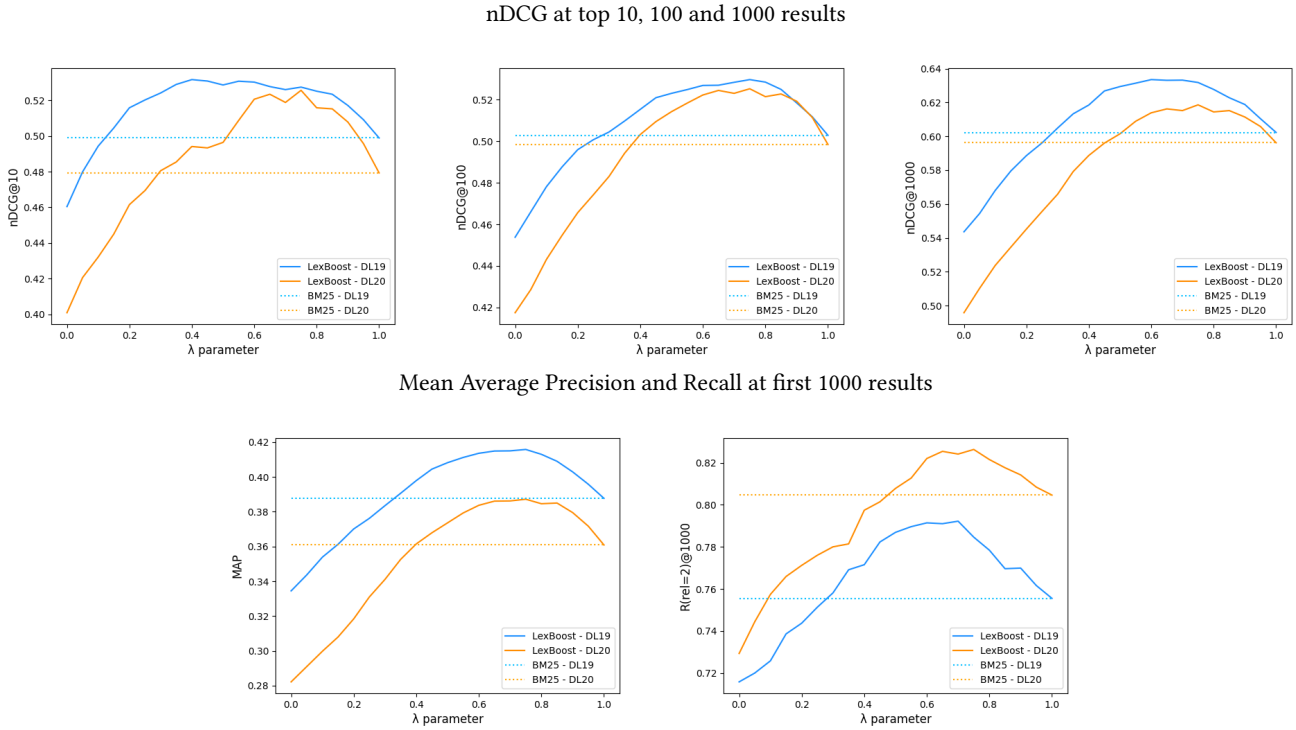

Figure 4: Validation based optimization for determination of fusion parameter λ.

Table 4: LexBoost on MS MARCO TREC DL 20. † denotes statistically significant improvement (paired t-test: $p \ddot{Y} 0.05$). Highest values denoted in bold. In LexBoost(k), k is the number of nearest neighbors. Top fusion parameter λ values determined using TREC DL 19 query set.

Method	λ	MAP	nDCG@1k	R(rel=2)@1k
BM25		0.3609	0.5962	0.8046
LexBoost(16)	0.7	**0.3862**†	**0.6152**†	**0.8241**
	0.8	0.3846†	0.6144†	0.8215†
	0.9	0.3795†	0.6114†	0.8142†
PL2		0.3227	0.5609	0.7772
LexBoost(16)	0.7	**0.3508**†	**0.5874**†	**0.8011**†
	0.8	0.3464†	0.5823†	0.7940
	0.9	0.3363†	0.5747†	0.7939†
DPH		0.3363	0.5704	0.7980
LexBoost(16)	0.7	**0.3645**†	**0.5970**†	**0.8195**†
	0.8	0.3638†	0.5951†	0.8123†
	0.9	0.3532†	0.5855†	0.8052†
QLD		0.3580	0.5870	0.8125
LexBoost(16)	0.7	**0.3987**†	**0.6198**†	**0.8347**†
	0.8	0.3933†	0.6165†	0.8325†
	0.9	0.3790†	0.6045†	0.8258†

Table 5: LexBoost on MS MARCO TREC DL 19 and 20. † denotes statistically significant improvement (paired t-test: $p \ddot{Y} 0.05$). Highest values denoted in bold. In LexBoost(k)(cg), k is the number of nearest neighbors and cg is the method used to construct corpus graph. Corpus graph constructed with HNP: TCT-ColBERT-HNP and TAS: TAS-B.

Method	λ	MAP	nDCG@1k	R(rel=2)@1k
		DL 19		
BM25		0.3877	0.6023	0.7555
LexBoost(16)(HNP)	0.7	**0.4150**†	**0.6332**†	**0.7922**†
	0.8	0.4130†	0.6277†	0.7784†
	0.9	0.4029†	0.6188†	0.7699†
LexBoost(16)(TAS)	0.7	**0.4147**†	**0.6355**†	**0.7896**†
	0.8	0.4123†	0.6272†	0.7766†
	0.9	0.4025†	0.6196†	0.7735†
		DL 20		
BM25		0.3609	0.5962	0.8046
LexBoost(16)(HNP)	0.7	**0.3862**†	**0.6152**†	**0.8241**
	0.8	0.3846†	0.6144†	0.8215†
	0.9	0.3795†	0.6114†	0.8142†
LexBoost(16)(TAS)	0.7	**0.3966**†	**0.6246**†	**0.8284**†
	0.8	0.3892†	0.6190†	0.8267†
	0.9	0.3791†	0.6045†	0.8152†

Table 6: Re-ranking using LexBoost on MS MARCO TREC DL 19 and 20. † denotes statistically significant improvement (paired t-test: p Ÿ 0.05). Highest values denoted in bold. In LexBoost(k)(m), k is the number of nearest neighbors and m is the dense retrieval method used for re-ranking. Corpus graph constructed with TCT-ColBERT-HNP. For m - HNP: TCT-ColBERT-HNP and TAS: TAS-B.

Method	λ	MAP	nDCG@1k	R(rel=2)@1k
		DL 19		
BM25 » HNP		0.4643	0.6786	0.7555
LexBoost(16)(HNP)	0.7	**0.4832**†	**0.7005**†	**0.7922**†
	0.8	0.4768†	0.6933†	0.7784†
	0.9	0.4728†	0.6887†	0.7699†
BM25 » TAS		0.4888	0.6842	0.7555
LexBoost(16)(TAS)	0.7	**0.5070**†	**0.7057**†	**0.7922**†
	0.8	0.4995	0.6982†	0.7784†
	0.9	0.4971†	0.6946†	0.7699†
		DL 20		
BM25 » HNP		0.4696	0.6854	0.8048
LexBoost(16)(HNP)	0.7	**0.4779**	**0.6967**	**0.8241**
	0.8	0.4763	0.6953†	0.8215†
	0.9	0.4731†	0.6911†	0.8142†
BM25 » TAS		0.4878	0.6912	0.8048
LexBoost(16)(TAS)	0.7	**0.4939**	**0.7023**	**0.8241**
	0.8	0.4933	0.7011†	0.8215†
	0.9	0.4903†	0.6968†	0.8142†

and varying number of nearest neighbors. We observed statistically significant improvements across a wide array of these combinations. The trends are evident in more detail in the heat-maps shown in Figure 3. On the y-axis for each heat-map the fusion parameter λ varies from 0.6 to 1. Darker the shade of blue - higher is the performance of LexBoost for that combination. It is evident that for a wide range of λ values significant improvements are observed. Further, the two different query sets are in sync with respect to the optimal λ value as evident in Figure 3. This shows effectiveness of our proposed method LexBoost for a range of fusion parameter values establishing robustness and hence addressing RQ4.

5.4 RQ5: Robustness across different datasets under consideration

RQ5 is about LexBoost working effectively across different datasets and independent of dataset preparation methods. As evident in Table 1, Table 2 and Table 3, we evaluated LexBoost across multiple datasets with different construction mechanisms. MS MARCO v1 passage ranking dataset was constructed by taking union of top passage lists for a large set of queries [36]. On the other hand, TREC-COVID dataset uses documents from CORD-19 which is a large set of scholarly articles about COVID [40]. In each case

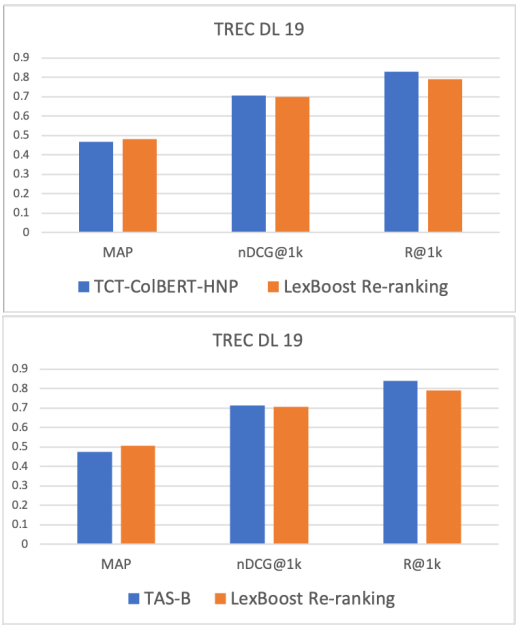

Figure 5: Comparison of LexBoost Re-ranking with exhaustive dense retrieval with TCT-Colbert-HNP and TAS-B on TREC DL 2019 query set.

we found statistically significant improvements for a wide range of combinations of fusion parameter λ and number of neighbors considered. This shows robustness of LexBoost across different datasets, hence addressing RQ5.

5.5 RQ6: Tuning and optimization for optimal fusion parameter λ

Figure 4 shows performance of LexBoost in five metrics, namely nDCG@10, nDCG@100, nDCG@1000, MAP and Recall(rel=2)@1000 for varying fusion parameter λ values. This evaluation is done using the TREC DL 19 and TREC DL 20 query sets. The points for the curve have been plotted for value of λ from 0 to 1 at regular intervals of 0.05. The BM25 baseline is shown by a flat line of the same color. In Figure 4, it is evident that for a wide range of λ values LexBoost leads to a better performance than the BM25 baseline. Further, we also note that both the TREC DL 19 and TREC DL 20 curves are in sync for all the five metrics in the respective graphs. This validates the effective use of training set in determining optimal fusion parameter λ value - addressing RQ6.

6 CONCLUSIONS AND FUTURE DIRECTIONS

We proposed LexBoost method which utilizes location of the target document in the corpus graph to gain valuable semantic insights from the neighboring documents along with their BM25 scores to determine the final score to be assigned. The enrichment resulted through semantic insights contributes to the increase in effectiveness. LexBoost provides a mechanism of effectively utilizing dense retrieval based similarity derived from the corpus graph with virtually no additional latency overheads at query time. This shows

statistically significant improvements in precision and recall. The method is robust across number of neighbors considered, variation in fusion parameter λ and multiple datasets. Overall robustness and improvements with virtually no additional cost makes LexBoost very impactful. LexBoost re-ranking also shows significant improvements over traditional re-ranking results. Further, LexBoost re-ranking shows comparable performance to high-latency exhaustive dense retrieval.

As a future work, LexBoost could be further extended for Cross-Lingual Information Retrieval (CLIR) and Multi-Lingual Information Retrieval (MLIR) settings. Additionally, the proposed approach can be extended to scenario which has user history. Here, a joint document-query graph can be built for stronger insights. We also plan to evaluate LexBoost architecture on more efficiently built graphs which use approximation methods for similarity calculations. We would like leave dynamic (runtime) tuning of fusion parameter λ for the future work.

REFERENCES

[1] Nasreen Abdul-Jaleel et al. 2004. UMass at TREC 2004: Novelty and HARD. *Computer Science Department Faculty Publication Series* 189. (2004).

[2] Antonio Acquavia et al. 2023. Static Pruning for Multi-Representation Dense Retrieval. In *Proceedings of the ACM Symposium on Document Engineering 2023* (Limerick, Ireland) *(DocEng '23)*.

[3] Gianni Amati and Cornelis Joost Van Rijsbergen. 2002. Probabilistic models of information retrieval based on measuring the divergence from randomness. *ACM Trans. Inf. Syst.* 20, 4 (oct 2002), 357–389.

[4] Payal Bajaj et al. 2016. MS MARCO: A Human Generated MAchine Reading COmprehension Dataset. In *InCoCo@NIPS*.

[5] Brian T. Bartell et al. 1998. Optimizing similarity using multi-query relevance feedback. *Journal of the American Society for Information Science* 49, 8 (1998).

[6] Tom Brown et al. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, Vol. 33. Curran Associates, Inc., 1877–1901.

[7] Sebastian Bruch et al. 2023. An Analysis of Fusion Functions for Hybrid Retrieval. *ACM Trans. Inf. Syst.* 42, 1, Article 20 (aug 2023).

[8] Claudio Carpineto and Giovanni Romano. 2012. A Survey of Automatic Query Expansion in Information Retrieval. *ACM Comput. Surv.* 44, 1, Article 1 (jan 2012).

[9] Shubham Chatterjee et al. 2024. DREQ: Document Re-ranking Using Entity-Based Query Understanding. In *Advances in Information Retrieval: 46th European Conference on Information Retrieval, ECIR 2024, Glasgow, UK, March 24–28, 2024.*

[10] Nachshon Cohen et al. 2024. InDi: Informative and Diverse Sampling for Dense Retrieval. In *Advances in Information Retrieval: 46th European Conference on Information Retrieval, ECIR 2024, Glasgow, UK, March 24–28, 2024.*

[11] Nick Craswell et al. 2020. Overview of the TREC 2019 deep learning track. https://doi.org/10.48550/ARXIV.2003.07820

[12] Nick Craswell et al. 2021. Overview of the TREC 2020 deep learning track. https://doi.org/10.48550/ARXIV.2102.07662

[13] Jacob Devlin et al. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics)*. Minneapolis, Minnesota, 4171–4186.

[14] Ayse Goker and John Davies. 2009. Information Retrieval: Searching in the 21st Century. (10 2009). https://doi.org/10.1002/9780470033647

[15] Sebastian Hofstätter et al. 2021. Efficiently Teaching an Effective Dense Retriever with Balanced Topic Aware Sampling. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Virtual Event, Canada) *(SIGIR '21)*. 113–122.

[16] Po-Sen Huang et al. 2013. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management* (San Francisco, California, USA) *(CIKM '13)*. 2333–2338.

[17] N. Jardine and C. J. van Rijsbergen. 1971. The use of hierarchic clustering in information retrieval. *Inf. Storage Retr.* 7 (1971), 217–240.

[18] Hrishikesh Kulkarni et al. 2023. Genetic Generative Information Retrieval. In *Proceedings of the ACM Symposium on Document Engineering 2023* (Limerick, Ireland) *(DocEng '23)*. Article 8, 4 pages.

[19] Hrishikesh Kulkarni et al. 2023. Lexically-Accelerated Dense Retrieval. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Taipei, Taiwan) *(SIGIR '23)*. 152–162.

[20] Haitao Li et al. 2023. Constructing Tree-based Index for Efficient and Effective Dense Retrieval. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Taipei, Taiwan) *(SIGIR '23)*. 131–140.

[21] Sheng-Chieh Lin et al. 2021. In-Batch Negatives for Knowledge Distillation with Tightly-Coupled Teachers for Dense Retrieval. In *Proceedings of the 6th Workshop on Representation Learning for NLP (RepL4NLP-2021)*. Association for Computational Linguistics, Online, 163–173.

[22] Bing Liu. 2007. Web usage mining. *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data* (2007), 449–483.

[23] Sean MacAvaney et al. 2022. Adaptive Re-Ranking with a Corpus Graph. In *31st ACM International Conference on Information and Knowledge Management*.

[24] Yu A. Malkov and D. A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42, 4 (2020).

[25] Donald Metzler and W. Bruce Croft. 2005. A Markov random field model for term dependencies. In *Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*.

[26] Tomas Mikolov et al. 2013. Efficient Estimation of Word Representations in Vector Space. In *ICLR*.

[27] Bhaskar Mitra and Nick Craswell. 2018. An Introduction to Neural Information Retrieval. *Foundations and Trends® in Information Retrieval* 13, 1 (December 2018), 1–126.

[28] Shahrzad Naseri et al. 2021. CEQE: Contextualized Embeddings for Query Expansion. In *Advances in Information Retrieval: 43rd European Conference on IR Research, ECIR 2021*. Springer-Verlag, Berlin, Heidelberg, 467–482.

[29] Sameh Neji et al. 2021. HyRa: An Effective Hybrid Ranking Model. *Procedia Comput. Sci.* 192, C (Jan 2021), 1111–1120.

[30] Liang Pang et al. 2016. A Study of MatchPyramid Models on Ad-hoc Retrieval. https://doi.org/10.48550/ARXIV.1606.04648

[31] Stephen Robertson et al. 2004. Simple BM25 extension to multiple weighted fields. In *Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management* (Washington, D.C., USA) *(CIKM '04)*. 42–49.

[32] Stephen E. Robertson and Hugo Zaragoza. 2009. The Probabilistic Relevance Framework: BM25 and Beyond. *Found. Trends Inf. Retr.* 3 (2009), 333–389.

[33] J. J. Rocchio. 1971. Relevance feedback in information retrieval. In *The Smart retrieval system - experiments in automatic document processing*, G. Salton (Ed.). Englewood Cliffs, NJ: Prentice-Hall, 313–323.

[34] Yelong Shen et al. 2014. Learning semantic representations using convolutional neural networks for web search. In *Proceedings of the 23rd International Conference on World Wide Web* (Seoul, Korea) *(WWW '14 Companion)*. 373–374.

[35] Sivic and Zisserman. 2003. Video Google: a text retrieval approach to object matching in videos. In *Proceedings Ninth IEEE International Conference on Computer Vision*. 1470–1477 vol.2.

[36] Ian Soboroff. 2021. Overview of TREC 2021. In *30th Text REtrieval Conference. Gaithersburg, Maryland.*

[37] Krysta M. Svore and Christopher J.C. Burges. 2009. A machine learning approach for improved BM25 retrieval. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management* (Hong Kong, China) *(CIKM '09)*. 4 pages.

[38] Peter D. Turney and Patrick Pantel. 2010. From Frequency to Meaning: Vector Space Models of Semantics. *J. Artif. Int. Res.* 37, 1 (jan 2010), 141–188.

[39] Ashish Vaswani et al. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, Vol. 30.

[40] Ellen Voorhees et al. 2021. TREC-COVID: Constructing a Pandemic Information Retrieval Test Collection. *SIGIR Forum* 54, 1, Article 1 (feb 2021), 12 pages.

[41] Lucy Lu Wang et al. 2020. CORD-19: The COVID-19 Open Research Dataset. In *Proceedings of the 1st Workshop on NLP for COVID-19 at ACL 2020*. Online.

[42] Xiao Wang et al. 2023. ColBERT-PRF: Semantic Pseudo-Relevance Feedback for Dense Passage and Document Retrieval. *ACM Trans. Web* 17, 1, Article 3 (jan 2023), 39 pages.

[43] Kyle Williams et al. 2014. SimSeerX: a similar document search engine. In *Proceedings of the 2014 ACM Symposium on Document Engineering* (Fort Collins, Colorado, USA) *(DocEng '14)*. 143–146.

[44] Lee Xiong et al. 2020. Approximate Nearest Neighbor Negative Contrastive Learning for Dense Text Retrieval. In *ICLR*.

[45] HongChien Yu et al. 2021. Improving Query Representations for Dense Retrieval with Pseudo Relevance Feedback. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management* (Virtual Event, Queensland, Australia) *(CIKM '21)*. 3592–3596.

[46] Jiangbo Yuan et al. 2012. Efficient Mining of Repetitions in Large-Scale TV Streams with Product Quantization Hashing. In *Computer Vision - ECCV 2012. Workshops and Demonstrations - Florence, Italy.*, Vol. 7583. Springer, 271–280.

[47] Chengxiang Zhai and John Lafferty. 2001. Model-based feedback in the language modeling approach to information retrieval. In *Proceedings of the 10th International Conference on Information and Knowledge Management (CIKM '01)*.