Off the Beaten Path: Let's Replace Term-Based Retrieval with k-NN Search

Leonid Boytsov
Carnegie Mellon University
Pittsburgh, PA, USA
srchvrs@cs.cmu.edu

David Novak
Masaryk University
Brno, Czech Republic
david.novak@fi.muni.cz

Yury Malkov
Institute of Applied Physics RAS
Nizhny Novgorod, Russia
yurymalkov@mail.ru

Eric Nyberg
Carnegie Mellon University
Pittsburgh, PA, USA
ehn@cs.cmu.edu

ABSTRACT

Retrieval pipelines commonly rely on a term-based search to obtain candidate records, which are subsequently re-ranked. Some candidates are missed by this approach, e.g., due to a vocabulary mismatch. We address this issue by replacing the term-based search with a generic k-NN retrieval algorithm, where a similarity function can take into account subtle term associations. While an exact brute-force k-NN search using this similarity function is slow, we demonstrate that an approximate algorithm can be nearly two orders of magnitude faster at the expense of only a small loss in accuracy. A retrieval pipeline using an approximate k-NN search can be more effective and efficient than the term-based pipeline. This opens up new possibilities for designing effective retrieval pipelines. Our software (including data-generating code) and derivative data based on the *Stack Overflow* collection is available online.[1] This revision is a slightly extended version of the respective CIKM'16 paper.

Keywords

k-NN search; IBM Model 1; non-metric spaces; LSH

1. INTRODUCTION

Due to advances in computing, a full-text search has become a ubiquitous information technology. However, this technology still largely relies on memorization of document terms and matching them with the query terms provided by a user.

The full-text search is powered by a *term-based* inverted index: a classic data structure that links document terms—and sometimes phrases—with their locations in a text collection. This way of organizing text data traces back to paper book indices containing alphabetical lists of principal words. In particular, it was used in a 13th century Bible concordance, long before the computer era [22].

[1]https://github.com/oaqa/knn4qa

CIKM'16 , October 24-November 28, 2016, Indianapolis, IN, USA

ISBN 978-1-4503-4073-1/16/10…$15.00

DOI: http://dx.doi.org/10.1145/2983323.2983815

Modern retrieval systems answer queries in a pipeline fashion. First, an term-based inverted index is used to generate a list of candidate documents containing some or all query terms. Second, this list is refined and re-ranked. A few highly-ranked documents are then presented to the user.

Re-ranking may be carried out in several steps, where earlier steps employ cheap ranking functions—such as BM25 [58] or language models [55]—relying solely on term occurrence statistics. A final, aggregation, step typically combines numerous relevance signals generated by upstream components. The aggregation step is often carried out using statistical *learning-to-rank* algorithms [38].

This filter-and-refine approach hinges on the assumption that a term-based search generates a reasonably complete list of candidate documents. However, this assumption is not fully accurate, in particular, because of a *vocabulary gap*, i.e., a mismatch between query and document terms denoting same concepts. The vocabulary gap is a well-known phenomenon. Furnas et al. [24] showed that, given a random concept, there is less than a 20% chance that two randomly selected humans denote this concept using the same term. Zhao and Callan [77] found that a term mismatch ratio—i.e., a rate at which a query term fails to appear in a relevant document—is roughly 50%.

Furthermore, according to Furnas et al. [24], focusing only on a few synonyms is not sufficient to effectively bridge the vocabulary gap. Specifically, it was discovered that, after soliciting 15 synonyms describing a single concept from a panel of subject experts, there was still a 20% chance that a new person coined a previously unseen term. To cope with this problem, Furnas et al. [24] proposed a system of *unlimited term aliases*, where potential synonyms would be interactively explored and presented to the user in a *dialog* mode.

An established *automatic* technique aiming to reduce the vocabulary gap is a *query expansion*. It consists in expanding (and sometimes rewriting) a source query using related terms and/or phrases. For efficiency reasons, traditional query expansion techniques are limited to dozens of expansion terms [12]. Using hundreds or thousands of expansion terms seems to be infeasible within a framework of the term-based inverted index. In contrast, we demonstrate that a system of unlimited term aliases can be successfully implemented within a more generic framework of a *k-nearest neighbor search* (k-NN search).

It has been long recognized that the k-NN search shows a promise to make retrieval a *conceptually* simple optimization procedure [32]. This approach may permit a separation of labor between data scientists, focusing on methods' accuracy, and software engineers, focusing on development of more efficient and/or scalable search approaches. However, the k-NN search proved to be a challenging problem due to the curse of dimensionality. There is empirical and theoretical evidence that this problem cannot be solved both exactly and efficiently in a high-dimensional setting [71, 5, 13, 52]. For some data sets, e.g., in the case of vectors with randomly generated elements, exact methods degenerate to a brute force search for just a dozen of dimensions [71, 5]. Some data sets only "look" high-dimensional, but possess properties of low-dimensional data sets, i.e., they have a low *intrinsic* dimensionality [33, 5, 13]. Unfortunately, textual data seems to be intrinsically high-dimensional. For example, using the definition of Chávez et al. [13], we estimate that the intrinsic dimensionality of Wikipedia TF$\times$IDF vectors is about 2500 in the case of the metric angular distance.

The curse of dimensionality can be partially lifted by using Locality Sensitive Hashing (LSH) techniques [8, 27, 35]. There are numerous modifications of LSH, which differ primarily in how they construct families of locality-sensitive functions [69]. Most of the research focuses on hash functions for well-studied similarities, such as the Euclidean distance and the cosine similarity.

In this paper, however, we explore an effective similarity function *BM25+Model 1*, which is neither metric nor symmetric (see § 2.1.3). We demonstrate that it is possible to carry out an efficient and effective k-NN search (for *BM25+Model 1*) using pivoting techniques. In that, the approximate k-NN search is nearly two orders of magnitude faster than the respective exact brute force search. The k-NN search can be 1.5$\times$ faster than Lucene, while being more effective due to bridging the vocabulary gap.

To ease reproducibility, we make our software (including data-generating code) and derivative data based on the Stack Overflow collection available online.[2]

2. APPROACH

We focus on a task of searching a large collection of answers extracted from a community QA website. The questions and answers are submitted by real people, who also select *best* answers. A question and the respective best answer represent one QA pair. While community QA is an important task on its own, it is used here primarily as a testbed to demonstrate the potential of the k-NN search as a substitute for term-based retrieval. Due to the curse of dimensionality, we have to resort to approximate searching. Note that we need a similarity function that outstrips the baseline method BM25 by a good margin. Otherwise, gains achieved by employing a more sophisticated similarity would be invalidated by the inaccuracy of the search procedure.

One effective way to build such a similarity function is to learn a generative question-answer *translation* model, e.g., IBM Model 1 [10]. However, "…the goal of question-answer translation is to learn associations between question terms and synonymous answer terms, rather than the translation of questions into fluent answers." [57] The idea of using a translation model in retrieval applications was proposed by Berger et al. [4]. It is now widely adopted by the IR and QA communities [18, 62, 57, 74, 63, 23]. Linearly combining BM25 and *logarithms* of IBM Model 1 scores produces a similarity function that is considerably more accurate than BM25 alone (by up to 30% on our data, see Table 3).

Learning IBM Model 1 requires a large monolingual parallel corpus. In that, the community QA data sets seem to be the best publicly available source of such corpora. Note that a monolingual corpus can be built from search engine click-through logs [56]. Yet, such data is not readily available for a broad scientific community. Another advantage of community QA data sets is that they permit a large scale automatic evaluation with sizeable training and testing subsets.

Specifically, we extract QA pairs from the following collections:

- L6 - Yahoo! Answers *Comprehensive* version 1.0 (about 4.4M questions);
- L5 - Yahoo! Answers *Manner* version 2.0 (about 142K questions), which is a *subset* of L6 created by Surdeanu al. [63];
- *Stack Overflow* (about 8.8M answered questions).

Yahoo! Answers collections are available through Yahoo! WebScope and can be requested by researchers from accredited universities.[3] For each question, there is always an answer (and the best answer is always present). The *Stack Overflow* collection is freely available for download.[4] While there are 8.8M answered questions, the best answer is not always selected by an asker. Such questions are discarded leaving us with 6.2M questions.

Each question has a (relatively) short summary of content, which is usually accompanied by a longer description. The question summary concatenated with the description is used as a query with the objective of retrieving the corresponding best answer. The best answer is considered to be the *only* relevant document for the query.

The accuracy of a retrieval system is measured using standard IR metrics: a Mean Reciprocal Rank (MRR), a precision at rank one (P@1) and an answer recall measured for the set of top-N ranked documents. P@1—our main evaluation metric—is equal to a fraction of queries where the highest ranked document is a true best answer to the question.

We process collections by removing punctuation, extracting tokens and term lemmas using Stanford CoreNLP [42] (instead, one can use any reasonably accurate tokenizer and lemmatizer). All terms and lemmas are lowercased; stopwords are removed. Note that we keep *both* lemmas and original terms. In *Stack Overflow* we remove all the code (the content marked by the tag `code`).

Each collection is randomly split into several subsets, which include training, two development (*dev1* and *dev2*), and testing subsets. In the case of *Comprehensive* and *Stack Overflow*, there is an additional subset that is used to learn IBM Model 1. The answers from this subset are indexed, but the questions are discared after learning Model 1 (i.e, they are not used for training and testing). In the case of *Manner*, IBM Model 1 is trained on a subset of *Comprehensive* from which we exclude QA pairs that belong to *Manner*. The split of *Manner* mimics the setup Surdeanu et al. [63] and the test set contains 29K queries. Collection statistics is summarized in Table 1.

Collection name	QA pairs total	train	dev1/dev2	test	tran	Terms in question	Terms in answer
Manner	142K	86K	7K/21K	29K	4.2M	13.9	40.6
Comprehensive	4.4M	212K	11K/42K	10K	4.1M	17.8	34.1
Stack Overflow	6.2M	298K	15K/58K	10K	5.8M	48.4	33.1

Table 1: Collection statistics. The column *tran* describes the size of the *BM25+Model 1* training corpus. Note that for *Manner* we use an *external* corpus to train *BM25+Model 1*.

We have implemented multiple retrieval methods and four similarity models (i.e., similarity functions). Retrieval methods, similarity models, and their interactions are summarized in Figure 1. Each

[2]https://github.com/oaqa/knn4qa

[3]https://webscope.sandbox.yahoo.com

[4]We use a dump from https://archive.org/download/stackexchange dated March 10th 2016.

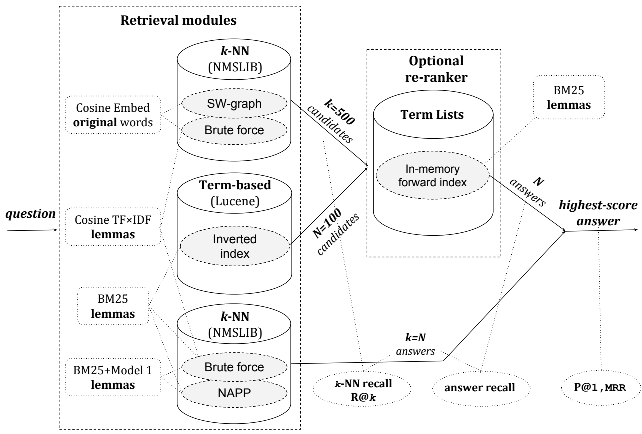


Figure 1: Retrieval pipeline architecture. We illustrate the use of evaluation metrics (inside ovals at the bottom) by dotted lines, which connect ovals with components for which metrics are applied.

retrieval method returns a ranked list of answers, which may be optionally re-ranked. The output is a list of N scored answers. There are two classes of retrieval methods: term-based retrieval supported by Apache Lucene[5] and the k-NN search methods implemented in the Non-Metric Space Library (NMSLIB). NMSLIB is an extendible framework for the k-NN search in generic spaces [7].[6] Similarity models include:

- TF×IDF models: the cosine similarity between TF×IDF vectors (shortly *Cosine TF×IDF*) and *BM25* (§2.1.1);
- The cosine similarity between averaged word embeddings, henceforth, *Cosine Embed* (§2.1.4);
- The linear combination of BM25 and IBM Model 1 scores, henceforth, *BM25+Model 1* (§2.1.3).

In the case of Lucene, we index lemmatized terms and use *BM25* as a similarity model [58]. We have found that Lucene's implementation of BM25 is imperfect (see § 2.1.1 for details), which leads to at least a 10% loss in P@1 for both *Comprehensive* and *Stack Overflow*. To compensate for this drawback, we obtain 100 top-scored documents using Lucene and re-rank them using our own implementation of BM25.

In the case of NMSLIB, we use two indexing methods and the brute force search. The indexing methods are: the Neighborhood APProximation index (*NAPP*) [64] and the Small-World graph (*SW-graph*) [39]. They are discussed in § 2.2. The SW-graph is used only with the *Cosine Embed*; NAPP is applied to both *BM25* and *BM25+Model 1*. We do not create an index for the *Cosine TF×IDF*, but use the brute force search instead. The brute force search is slow, but it is applicable to any similarity model.

Because the *Cosine TF×IDF* and *Cosine Embed* are not very accurate, the output from these models may be further re-ranked using BM25. To this end, we first retrieve 500 candidate records. Next, we discard all but N records with highest BM25 scores. To compare effectiveness of *Cosine TF×IDF* and *BM25*, we also evaluate a variant where the output of *Cosine TF×IDF* is not re-ranked.

To re-rank efficiently, we use a forward index. Given a document identifier, this index allows us to quickly retrieve the list of terms and their in-document frequencies. Following a recent trend in high throughput in-memory database systems [31], we load forward indices into memory. The overall re-ranking time is negligibly small.

Note that the depth of a candidate pool represents a reasonable efficiency-effectiveness trade-off. While increasing the depth of the pool improves the answer recall, it also makes it harder to rank results accurately. Beyond a certain point, increasing the depth leads only to a marginal improvement in P@1 at the expense of disproportionately large computational effort.

2.1 Similarity Models

2.1.1 Cosine TF×IDF and BM25

Cosine TF×IDF and *BM25* are computed for *lemmatized* text. *Cosine TF×IDF* is the classic model where the similarity score is equal to the cosine similarity between TF×IDF vectors [59, 41]. An element i of such a vector is equal to the product of the *unnormalized* term frequency TF_i and the inverse document frequency (IDF). To compute IDF, we use the formula implemented in Lucene:

$$\ln\left(1 + (D - d + 0.5)/(d + 0.5)\right), \tag{1}$$

where D is the number of documents and d is the number of documents containing the term i.

BM25 scores [58] are computed as the sum of term IDFs (Eq. 1) multiplied by respective *normalized* term frequencies. The sum includes only terms appearing in both the query and the answer. We also normalize BM25 scores using the sum of query term IDFs. Normalized frequencies are as follows:

$$\frac{\mathrm{TF}_i \cdot (k_1 + 1)}{\mathrm{TF}_i + k_1 \cdot \left(1 - b + b \cdot |D| \cdot |D|_{\mathrm{avg}}^{-1}\right)}, \tag{2}$$

where k_1 and b are parameters ($k_1 = 1.2$ and $b = 0.75$); $|D|$ is a document length in words; $|D|_{\mathrm{avg}}$ is the average document length. Lucene's implementation of BM25 uses a *lossy* compression for the document length, which results in reduced effectiveness.

2.1.2 IBM Model 1

Computing translation probabilities via IBM Model 1 [10] is one common way to quantify the strength of associations among question and answer terms. The transformed IBM Model 1 scores are used as input to a learning-to-rank algorithm. Specifically, we take the *logarithm* of the translation probability and divide it by the number of query terms.

Let $T(q|a)$ denote a probability that a question term q is a translation of an answer term a. Then, a probability that a question Q is a translation of an answer A is equal to:

$$\begin{aligned} P(Q|A) &= \prod_{q \in Q} P(q|A) \\ P(q|A) &= (1-\lambda)\left[\sum_{a \in A} T(q|a)P(a|A)\right] + \lambda P(q|C) \end{aligned} \tag{3}$$

$T(q|a)$ is a translation probability learned by the GIZA++ toolkit [50] via the EM algorithm; $P(a|A)$ is a probability that a term a is generated by the answer A; $P(q|C)$ is a probability that a term q is generated by the entire collection C; λ is a smoothing parameter. $P(a|A)$ and $P(q|C)$ are computed using the maximum likelihood estimator. For an out-of-vocabulary term q, $P(q|C)$ is set to a small number (10^{-9}). Similar to *BM25* and *Cosine TF×IDF*, computation is based on the *lemmatized* text.

A straightforward but slow approach to compute IBM Model 1 scores involves storing $T(q|a)$ in the form of a sparse hash table. Then, computation of Eq. 3 entails one hash table lookup for each combination of question and answer terms. We can do better by creating an inverted index for each query, which permits retrieving

[5] http://lucene.apache.org

[6] https://github.com/searchivarius/nmslib

Prior art [63]				
	$N = 15$	$N = 25$	$N = 50$	$N = 100$
Recall	0.290	0.328	0.381	0.434
P@1	0.499	0.445	0.385	0.337
MRR	0.642	0.582	0.512	0.453
This paper				
	$N = 10$	$N = 17$	$N = 36$	$N = 72$
Recall	0.293	0.331	0.386	0.438
P@1	0.571 (+14%)	0.511 (+15%)	0.442 (+15%)	0.392 (+16%)
MRR	0.708 (+10%)	0.645 (+11%)	0.570 (+11%)	0.510 (+13%)

Table 2: Comparison of *BM25+Model 1* against prior art on *Manner*. Accuracy is computed for several result set sizes N using the methodology of Surdeanu et al. [63]. Each column corresponds to a different subset of queries.

query-specific entries $T(q|a)$ using the identifier of answer term a as a key. Thus, we need only one lookup per answer term. Identifiers are indexed using an efficient hash table (the class *dense_hash_map* from the package *sparsehash*)[7]. Building such an inverted index is computationally expensive (about 15 ms for each *Comprehensive* and 90 ms for each *Stack Overflow* query). Yet, the cost is amortized over multiple comparisons between the query and data points.

We take several measures to maximize the effectiveness of IBM Model 1. First, we compute translation probabilities on a symmetrized corpus as proposed by Jeon et al. [30]. Formally, for every pair of documents (A, Q) in the parallel corpus, we expand the corpus by adding entry (Q, A).

Second, unlike previous work, which seems to use *complete* translation tables, we discard all translation probabilities $T(q|a)$ below an empirically found threshold of $2.5 \cdot 10^{-3}$. The rationale is that small probabilities are likely to be the result of model overfitting. Pruning of the translation table improves both efficiency and effectiveness. It also reduces memory requirements.

Third, following prior proposals [30, 63], we set $T(w|w)$, a self-translation probability, to an empirically found positive value and rescale probabilities $T(w'|w)$ so that $\sum_{w'} T(w'|w) = 1$.

Fourth, we make an ad hoc decision to use as many QA pairs as possible to train IBM Model 1. A positive impact of this decision has been confirmed by a post hoc assessment.

Finally, we tune parameters on a development set (*dev1* or *dev2*). Rather than evaluating individual performance of IBM Model 1, we aim to maximize performance of the model that linearly combines BM25 and IBM Model 1 scores.

2.1.3 BM25+Model 1

BM25+Model 1 is a linear two-feature model, which includes BM25 and IBM Model 1 scores. Optimal feature weights are obtained via a coordinate ascent with 10 random restarts [43]. The model is trained via RankLib[8] using P@1 as a target optimization metric. To obtain training data, we retrieve $N = 15$ candidates with highest BM25 scores [58] using Lucene. If we do not retrieve a true answer, the query is discarded. Otherwise, we add the true answer to the training pool with the label one (which means relevant), and the remaining retrieved answers with the label zero (which means non-relevant).

To demonstrate that *BM25+Model 1* delivers state of the art performance, we compare our result on *Manner* against a previously published result [63]. We mimic the setup of Surdeanu et al. [63] and use only questions for which a relevant answer is found (but not necessarily ranked number one). We also split the collection in the same proportions as Surdeanu et al. [63].[9] Furthermore, we measure P@1 at various recall levels by varying the result sizes N (which are different from those used by Surdeanu et al [63]).

According to Table 2, our method surpasses the previously published result by 14–26 % in P@1, and by 10–11 % in MRR despite using only two features. This may be explained by two factors. First, we use a 50× larger corpus to train IBM Model 1. Second, the retrieval module Terrier BM25 (employed by Surdeanu et al. [63]) seems to have inferior retrieval performance compared to Lucene. In particular, Lucene achieves a higher recall using a smaller N (see Table 2).

2.1.4 Cosine Embed

Word *embeddings*, also known as distributed word representations, are real-valued vectors associated with words. Word embeddings are usually constructed in an unsupervised manner from large unstructured corpora via artificial neural networks. [14, 44]. Because embeddings can capture syntactic and semantic regularities in language [66, 45], embedding-based similarity can be useful in retrieval and re-ranking tasks [23, 75]. The hope here is that comparing embeddings instead of original words would help to bridge the vocabulary gap.

One popular embedding-based similarity measure is the average cosine similarity computed for all pairs of question and answer terms. The average pairwise cosine similarity is equal to the cosine similarity between averaged word embeddings of questions and answers. In our work we use the cosine similarity between *IDF-weighted* averaged embeddings. Here, we use embeddings of non-lemmatized terms, because this results in a slightly improved performance. We evaluate several sets of pre-trained embeddings to select the most effective ones [44, 51, 73]. We further improve embeddings by retrofitting [21]. In that, Model 1 translation table $T(q|a)$ (see Eq. 3) is used as a relational lexicon.

2.2 Methods of k-NN Search

We employ a k-NN retrieval framework NMSLIB, which provides several implementations of distance based indexing methods [7]. These indexing methods treat data points as unstructured objects, together with a black-box distance function. In that, the indexing and searching process exploit only values of mutual object distances. NMSLIB can be extended by implementing new black-box "distance" functions. In particular, we add an implementation for the similarity functions *BM25*, *Cosine TF×IDF*, *Cosine Embed*, and *BM25+Model 1* (see in §2.1). None of these similarity functions is a metric distance. In particular, in the case of *BM25+Model 1* the "distance" lacks symmetry.

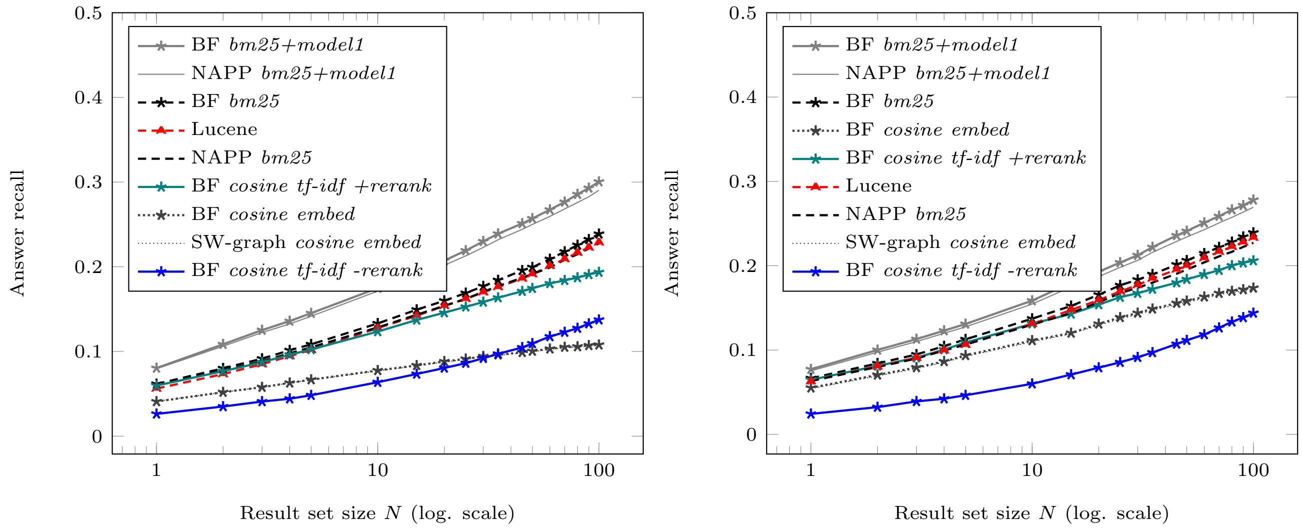
Because exact k-NN search is too slow to be practical, we resort to an *approximate* procedure, which does not necessarily find all k nearest neighbors. The accuracy of the k-NN search is measured using a recall denoted as R@k. R@k is equal to the fraction of true k-nearest neighbors found.

NMSLIB reads contents of the forward index (created by a separate indexing pipeline) into memory and builds an additional *in-memory* index. In this work, we create indices using one of the following method: the Neighborhood APProximation index (*NAPP*) due to Tellez et al. [64] or the proximity graph method called a Small-World graph (*SW-graph*) due to Malkov et al. [39].

[7] The code, originally written by Craig Silverstein, is now hosted at https://github.com/sparsehash/sparsehash

[8] https://sourceforge.net/p/lemur/wiki/RankLib/

[9] The exact split used by Surdeanu et al. [63] is not known. To ensure that the differences are substantial and significant, we also compute 99.9% confidence intervals.



(a) *Stack Overflow*

(b) *Comprehensive*

Figure 2: The answer recall at different N. (BF stands for brute-force; *+rerank* and *-rerank* indicate if an optional re-ranker is used).

NAPP is a *pivoting* method that arranges points based on their distances to pivots. This is a filtering method: Candidate points share `numPivotSearch` closest pivots with the query point. The search algorithm employs an inverted index. Unlike term-based indices, however, for each pivot the index keeps references to close data points. More specifically, the pivot should be one of the point's `numPivotIndex` closest pivots. Answering a query requires efficient merging of posting lists. Merging of posting lists represents a substantial overhead.

Tellez et al. [64] use pivots randomly sampled from the data set, but we find that for *sparse* data such as TF×IDF vectors substantially shorter retrieval times—sometimes by orders of magnitude—can be obtained by using a special pivot generation algorithm. Specifically, pivots are generated as pseudo-documents containing K entries sampled from the set of M most frequent words (in our setup $K = 1000$ and $M = 50000$). A more detailed description and analysis of this approach will be presented elsewhere.

During indexing, we have to compute the distances between a data point and every pivot. Because there are thousands of pivots, this operation is quite expensive, especially for *BM25+Model 1*. To optimize computation of Eq. 3, we organize all pivot-specific $T(q|a)$ entries in the form of the inverted index.

A proximity graph is a data structure, where data points are nodes. Sufficiently close nodes, i.e., *neighbors*, are connected by edges. Searching starts from some, e.g., random, point/node and traverses the graph until it stops discovering new points sufficiently close to the query or after visiting a given number of nodes. [2, 60, 26, 25, 16, 70]. Specifically, the SW-graph algorithm (implemented in NMSLIB) keeps a list of `efSearch` points sorted in the order of increasing distance from the query as well as a candidate queue. Traversal proceeds in the best-first manner, by exploring the neighborhood of the candidate that is closest to the query. If a candidate neighbor is closer to the query than the `efSearch`-*th* closest point seen so far, it is added to the candidate queue. Otherwise, the neighbor is discarded. The traversal stops when the candidate queue is exhausted. To improve recall, the algorithm may restart several times. For our data, however, it is more efficient to start from a single point and search using a large-enough value of `efSearch`.

SW-graph works well for dense vectorial data (i.e., embeddings), where it outstrips NAPP by an order of magnitude. SW-graph was found to be much faster [49] than the multi-probe LSH due to Dong et al. [17]. In a public evaluation in May 2016,[10] SW-graph outperformed two efficient popular libraries: FLANN [48] and Annoy[11]. SW-graph was also mostly faster than a novel LSH algorithm [1]. In contrast, NAPP substantially outperforms SW-graph for sparse TF×IDF data, i.e., for models *BM25* and *BM25+Model 1*.

3. MAIN EXPERIMENTS

Experiments are carried out on Amazon EC2 instance *r3.4xlarge*, which has 16 virtual cores and 122 GB of memory. The main retrieval pipeline, which is implemented in Java (1.8.0_11), uses 16 search threads. We use a modified version of NMSLIB 1.5,[12] which operates as a server processing queries via TCP/IP. NMSLIB and its extensions are written in C++ and compiled using GCC 4.8.4 with optimization flags `-O3` and `-march=native`. Lucene version is 4.10.3. The retrieval architecture (see § 2) is outlined in Figure 1.

The collection processing/indexing system is implemented in Java. It employs the framework Apache UIMA and UIMA components from DKPro Core [19].[13] Translation probabilities are computed using GIZA++ toolkit [50] via the EM algorithm (five iterations).[14]

For each *approximate* k-NN pipeline, we execute several runs with different parameters. In the case of SW-graph, we vary the parameter `efSearch`. In the case of NAPP, we vary the number of indexed pivots (parameter `numPivotIndex`) and the number of pivots that should be shared between the query and an answer (`numPivotSearch`). Optimal parameters have been found on a *dev1* set (using a subset of 5K queries).

Retrieval times are measured by a special client application that submits search requests to either Lucene or NMSLIB. In the case

[10]https://github.com/erikbern/ann-benchmarks
[11]https://github.com/spotify/annoy
[12]https://github.com/searchivarius/nmslib/tree/nmslib4a_cikm2016
[13]https://dkpro.github.io/dkpro-core/
[14]https://github.com/moses-smt/giza-pp

Stack Overflow						Comprehensive					
Query time	Speed-up (over BF)	P@1	P@1 loss	Answer recall	Answer recall loss	Query time	Speed-up (over BF)	P@1	P@1 loss	Answer recall	Answer recall loss
BF *BM25+Model 1*											
20.5 s		0.081*		0.300*		8 s		0.077*		0.278*	
NAPP *BM25+Model 1*											
0.84 s	24	0.080*	1.1%	0.290*	3.3%	0.70 s	11	0.075*	2.6%	0.269*	3.1%
0.65 s	32	0.079*	2.2%	0.283*	5.7%	0.30 s	27	0.074*	4.7%	0.261*	6.1%
0.50 s	35	0.078*	3.1%	0.276*	8.2%	0.21 s	38	0.073	5.7%	0.256*	8.1%
0.47 s	44	0.076*	5.8%	0.263*	12.4%	0.18 s	45	0.070	9.7%	0.234	16.0%
0.40 s	52	0.074*	8.2%	0.252*	16.1%	0.09 s	89	0.068	12.4%	0.227*	18.5%
BF *BM25*											
3.5 s		0.062		0.239		1.7 s		0.067		0.239	
NAPP *BM25*											
0.23 s	15	0.060	2.1%	0.228*	4.7%	0.37 s	5	0.063*	5.1%	0.227*	5.0%
0.14 s	25	0.059*	5.0%	0.221*	7.6%	0.18 s	9	0.062*	6.6%	0.222*	7.1%
0.07 s	50	0.057*	7.6%	0.208*	12.9%	0.15 s	11	0.061*	8.4%	0.217*	9.2%
0.06 s	56	0.055*	11.2%	0.195*	18.5%	0.15 s	11	0.060*	9.3%	0.212*	11.2%
Lucene BM25											
0.62 s		0.062		0.229*		0.08 s		0.067		0.233*	
BF *Cosine Embed*											
3.9 s		0.041*		0.108*		2.7 s		0.055*		0.174*	
SW-graph *Cosine Embed*											
0.78 s	5	0.041*	-0.2%	0.107*	0.6%	0.19 s	14	0.054*	1.6%	0.172*	1.0%
0.40 s	10	0.041*	0.5%	0.107*	0.8%	0.09 s	31	0.054*	3.1%	0.170*	1.9%
0.34 s	11	0.041*	0.7%	0.106*	1.3%	0.07 s	37	0.053*	3.6%	0.170*	2.4%
0.13 s	29	0.039*	4.9%	0.102*	5.8%	0.03 s	104	0.050*	10.3%	0.160*	7.8%

Table 3: Efficiency-effectiveness trade-offs of retrieval modules for $N = 100$ (brute force *Cosine TF×IDF* runs are omitted). Statistically significant differences (at level 0.01) from **BF *BM25*** are marked with *. P-values are adjusted for multiple testing via the Bonferroni correction.

of Lucene, we "warm up" the index by executing the whole set of queries twice. Run-times are measured only for the third run.

Effectiveness of *retrieval runs* is measured using an external application, namely, trec_eval 9.0.4.[15] The main experimental results are presented in Figure 2 and Table 3. For Table 3 we compute statistical significance of results using the t-test with a subsequent Bonferroni adjustment for multiple testing. This adjustment for multiple testing consists in multiplying p-values by the total number of runs for $N = 100$ (to save space, some of the runs are not shown in the table). The significance level is 0.01.

In Figure 2 we plot the answer recall (measured for a set of top-N ranked documents) for nine implemented retrieval modules. Note that approximate k-NN methods are represented by their most accurate runs. Exact brute force k-NN runs are plotted using thicker lines (with star marks) of the same style/color as corresponding approximate runs. Their mnemonic names start with the word BF (short for brute force).

The cosine-similarity models are the least effective. The recall of the brute force run BF *Cosine TF×IDF -rerank* is less than half of that for the brute force run BF *BM25*. We can nearly match the performance of *BM25* by adding a BM25-based optional re-ranker (the run BF *Cosine TF×IDF +rerank*). In contrast, the cosine-similarity between averaged word embeddings (e.g., the run BF *Cosine Embed*) is much worse than *BM25* despite using the re-ranker! Somewhat surprisingly, the cosine similarity between TF×IDF vectors *without* re-ranker is sometimes more effective than the cosine similarity between word embeddings whose performance is boosted by the re-ranker (see Panel 2a in Figure 2). This is a discouraging finding given that embedding-based retrieval can be quite efficient (see Table 3). It remains to be verified if better results can be obtained with document embeddings that compute vectorial representations of complete sentences or even documents [28, 36].

Note that all BM25-based runs have similar performance. However, *BM25+Model 1* has a recall that is 16% higher in the case of *Comprehensive* and 26% higher in the case of *Stack Overflow*. For *BM25*, it is possible to match the recall of *BM25+Model 1* by increasing N. However, this may increase a load on a downstream re-ranking module. For example, in the case of *Stack Overflow*, *BM25+Model 1* has a nearly 0.2 answer recall for $N = 10$ (Panel 2a in Figure 2). To obtain the same recall level using Lucene, we need to use $N > 20$.

Next, we compare efficiency of k-NN search methods against that of Lucene. Note that Lucene is a strong baseline, which fares well against optimized C++ code, especially for disjunctive queries [68]. Lucene's average retrieval times are equal to 80 ms for *Comprehensive* and 620 ms for *Stack Overflow* (see Table 3). There are at least two factors that contribute to the difference in retrieval times between two collections: (1) questions in *Stack Overflow* have 2.7× as many terms, (2) *Stack Overflow* has 1.4× as many answers (see Table 1).

SW-graph is quite fast for both collections. For example, for *Stack Overflow*, it can answer queries in 340 ms at the expense of

[15] https://github.com/usnistgov/trec_eval

Comprehensive			
BM25+Model 1		*BM25*	
R@1	Reduction in distance comp.	R@1	Reduction in distance comp.
0.982	8.7	0.982	3.7
0.968	61	0.970	20
0.961	142	0.963	48
0.952	246	0.956	98
0.930	434	0.927	226
Stack Overflow			
BM25+Model 1		*BM25*	
R@1	Reduction in distance comp.	R@1	Reduction in distance comp.
0.982	13.4	0.980	157
0.970	39	0.972	208
0.964	64	0.964	260
0.957	97	0.959	287
0.948	137	0.955	315

Table 4: Reduction in the number of the distance computation for two similarity models at approximately equal levels of R@1 (larger reduction is better). Using 5K queries from *dev1* set.

losing only 1.3% answers compared to the brute force search (As it is recently reported by Malkov and Yashunin [40], an improved, hierarchical, variant of SW-graph is even more accurate and/or efficient). In other words, the approximate search is nearly as accurate as the exact one. This is why in Figure 2 the best *approximate* SW-graph run for the model *Cosine Embed* and the run BF *Cosine Embed* are hard to distinguish. However, the model *Cosine Embed* is not very effective. It does not bridge the vocabulary gap and is even worse than *Cosine TF$\times$IDF*.

In the case of *BM25*, NAPP works well for *Stack Overflow*, but not for *Comprehensive*. For example, in the case of *Stack Overflow*, it answers queries in 230 ms while losing only 2.1% in P@1 and 4.7% in the answer recall. This is nearly 2.7$\times$ faster than Lucene and 15$\times$ faster than the brute force search using *BM25*.

For the more complicated model *BM25+Model 1*, NAPP delivers similar speed ups over the brute force search for both collections. However, it is always slower than Lucene in the case of *Comprehensive*. In the case of *Stack Overflow*, NAPP is up to 1.5$\times$ faster than Lucene. For the fastest posted retrieval time of 400 ms it delivers P@1 equal to 0.074 and the answer recall equal to 0.252.

Despite some degradation in comparison to the corresponding exact brute force run, this represents an impressive 19.3% improvement in P@1 and 5.4% improvement in the answer recall compared to the brute force *BM25*. The second slowest *BM25+Model 1* run obtained by NAPP is nearly as efficient as Lucene, but it outperforms *BM25* by 27.4% in P@1 and by 18.4% in recall.

Also note that for *Comprehensive*, NAPP *BM25+Model 1* can be both faster and more accurate than NAPP *BM25*. This is quite surprising given that *BM25+Model 1* is expensive to compute. Specifically, the corresponding brute force run is nearly 5$\times$ slower compared to the brute force run of *BM25*. There are two reasons for why NAPP *BM25+Model 1* can be more faster and accurate than NAPP *BM25*. First, there is a high overhead related to merging pivot posting lists. By varying method's parameters we can reduce the amount of time spent on computation of the distance (at the expense of search accuracy). At some point the time spent on distance computation becomes so small so that the overhead related to processing of posting lists starts to dominate the overall time.

Second, there are differences in filtering effectiveness of the methods. To demonstrate this, we evaluate the reduction in the number of distance computations compared to the brute force search. For example, if an algorithm answers a query by checking only 10% of data points, the reduction in the number of distance computations is 10. Reductions in the number of distance computations are compared for nearly equal values of the k-NN recall R@1, which is equal to the fraction of true nearest neighbors found by the retrieval module (R@1 should not be confused with the answer recall). The results of this comparison are presented in Table 4. For NAPP, the more pivots are indexed, the fewer distance computations are necessary to achieve a given accuracy level. Thus, to make a fair comparison, we index equal number of pivots for both *BM25* and *BM25+Model 1*.

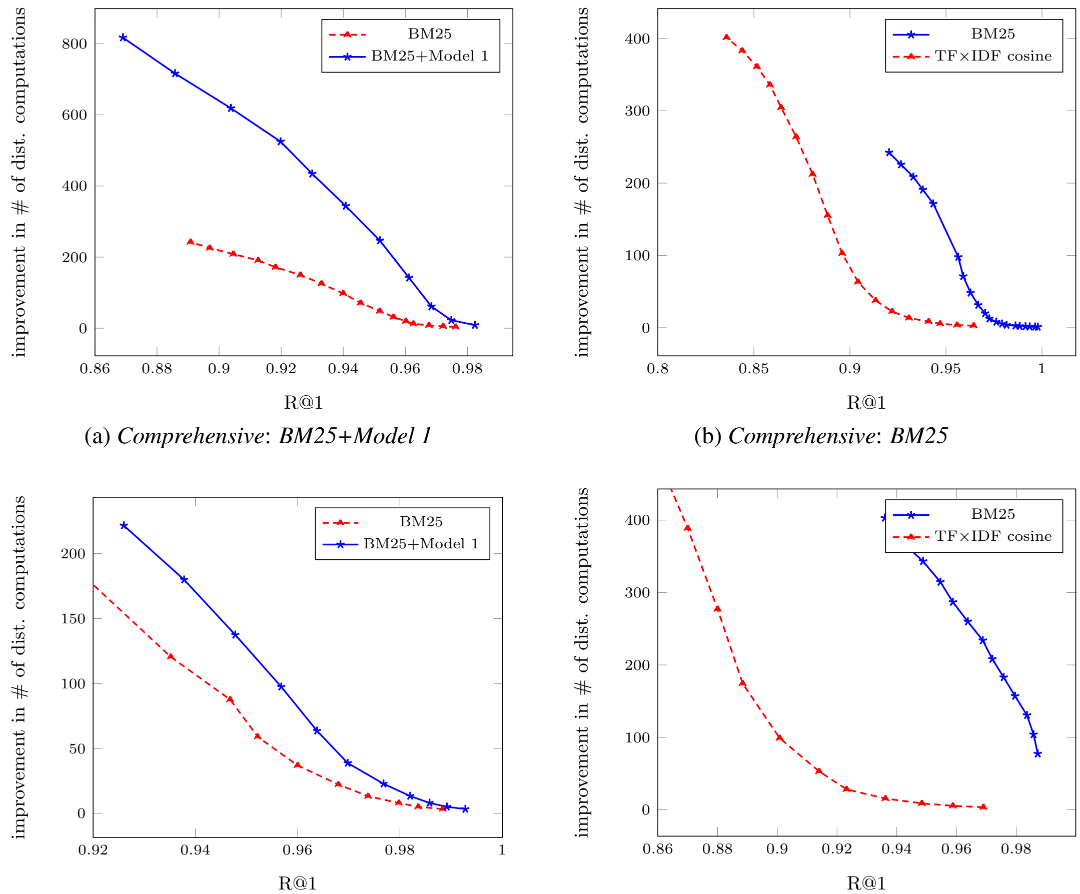
According to Table 4, in the case of *Comprehensive*, it takes 2-3$\times$ fewer distance computations for the model *BM25+Model 1* than for *BM25*. In contrast, in the case of *Stack Overflow*, answering queries for the model *BM25* takes significantly fewer distance computations than for *BM25+Model 1*. Furthermore, the reduction in the number of distance computations for *BM25* on *Stack Overflow* can be two orders of magnitude higher compared to that of *Comprehensive*. What are the possible explanations for these stark differences?

We think that pivoting methods are effective only if comparing a query and an answer with the same pivot provides a meaningful information regarding their proximity. In the case of a simple *BM25* model, this is only possible if the pivot, the query, and the answer have at least one common term. Such an overlap is much more likely in the case of *Stack Overflow* where questions are nearly 3$\times$ longer compared to *Comprehensive*. In contrast, for the model *BM25+Model 1* information regarding proximity of answers and queries may be obtained if pivots, queries, and answers share only related but not necessarily identical terms. Thus, using *BM25+Model 1* is more advantageous in the case of short queries (e.g., in the case of *Comprehensive*).

To further illustrate importance of using the right function to compute distance to pivots, we evaluate filtering effectivness in two scenarious: (1) when the distance to pivots is computed using an original distance function and (2) when the distance to pivots is computed using a different, i.e., proxy function. For each scenarios, we use two models: *BM25+Model 1* and *BM25*. In the case of *BM25+Model 1*, the proxy distance is *BM25*. In the case of *BM25*, the proxy distance is *Cosine TF$\times$IDF*. The results are presented in Figure 3 where the curves corresponding to the original distance are blue and the curves corresponding to the proxy distance are red.

Panels 3a and 3c show us what happens if the distance to pivots is computed using cheap *BM25* instead of expensive *BM25+Model 1*. We can see that resorting to using the proxy distance makes us check more candidate documents to achieve the same level of recall. In other words relying on the proxy distance has a negative effect on filtering effectiveness. In turn, this can drastically reduce overall search efficiency.

The difference is bigger for Panel 3a, which corresponds to the collection *Comprehensive*. A likely explanation of this difference is the above-described disparity in query lengths between two collections. In the case of *Stack Overflow* queries are long and there is a bigger overlap between queries and answer documents. This is why the similarity function that relies on a pure lexical match (in this case *BM25*) allows us to find answers rather effectively. In the case of *Comprehensive* a lexical overlap between queries and answer documents is less likely, which can be, nevertheless, remedied by enhancing *BM25* model with Model 1 scores. However, when Model 1 scores are excluded—by using the proxy distance function to compute distance to pivots—this exclusion has a larger negative effect for *Comprehensive* than for *Stack Overflow*.



(a) *Comprehensive*: *BM25+Model 1*

(b) *Comprehensive*: *BM25*

(c) *Stack Overflow* : *BM25+Model 1*

(d) *Stack Overflow* : *BM25*

Figure 3: Filtering effectiveness of NAPP for original and a proxy distance function (when computing distances to pivots). The curves for the original distance are blue while proxy distance curves are red. Filtering effectiveness is measured using via reduction in the number of distance computations (larger is better). The left column has data for the distance *BM25+Model 1* and the right column has data for *BM25*. Using 5K queries from *dev1* set.

Panels 3b and 3d show us what happens if the distance to pivots is computed using *Cosine TF×IDF* instead of *BM25*. In this case, such a replacement leads to a much larger performance deterioration than removal of Model 1 scores. This is not surprising: As we can see from Figure 2, there is a much bigger gap in effectiveness between *BM25* and *Cosine TF×IDF* than between *BM25+Model 1* and *BM25*. Thus, replacing *BM25* with *Cosine TF×IDF* has also a larger negative effective on filtering effectiveness (than replacing *BM25+Model 1* with *BM25*).

4. DISCUSSION AND RELATED WORK

The k-NN search is an extensively studied area. For a detailed discussion the reader is addressed to the surveys of metric [13] and non-metric [61] access methods, as well to the recent survey of hashing techniques [69].[16]

The k-NN search is a popular technique in IR and NLP, where the following two approaches are typically used. The first approach relies on a term-based inverted index in retrieving documents that share common terms with the query. These documents are further re-ranked using some similarity function. Dynamic and static pruning can be used to improve efficiency, sometimes at the expense of decreased recall [67, 11, 15]. This approach supports arbitrary similarity functions, but it suffers from the problem of the vocabulary mismatch [4, 63, 23].

The second approach involves carrying out the k-NN search via LSH [53, 65, 46, 37]. It is most appropriate for the cosine similarity. For example, Li et al. [37] propose the following two-stage scheme to the task of finding thematically similar documents. In the first step they retrieve candidates using LSH. Next, these candidate are re-ranked using the Hamming distance between quantized TF×IDF vectors. Li et al. [37] find that their approach is up to 30× faster than the classic term-based index while sometimes being equally accurate.

Petrović et al. [53] applied a hybrid of LSH and the term-based index to the task of the streaming First Story Detection (FSD). The LSH keeps a large chunk of sufficiently recent documents, while the term-based index keeps a small subset of recently added documents. They report their system to be substantially faster than the state-of-

[16]In addition to the k-NN search, hashing techniques are often used for near-duplicate detection [41], which consists in finding objects with a high degree of similarity. It is a related but distinct problem, which is often solved by applying high-precision low-recall techniques [41]. However, these techniques are not applicable to broader search tasks such as finding answers relevant to a given question.

the-art system—which relies on the classic term-based index—while being similarly effective. In a follow up work, Petrović et al. [54] incorporate term associations into the similarity function. Their solution relies on an approximation for the kernelized cosine similarity. The associations are obtained from an external paraphrasing database. Moran et al. [46] use the same method as Petrović et al. [54], but find synonyms via the k-NN search in the space of word embeddings (which works better for Twitter data). Moran et al. [46] as well as Petrović et al. [54] calculate performance using an aggregated metric designed specifically for the FSD task. Unfortunately, they do not report performance gains using standard IR metrics such as precision and recall.

Most importantly, as shown in the literature (see [72] and references therein), similarity functions based on the cosine similarity are not especially effective. In particular, compared to *BM25*, our implementation of the TF×IDF cosine similarity finds 2× *fewer* answers for any given rank N(see Figure 2). It is possible to improve answer recall by increasing N. However, this has effect on search module' performance. In particular, if we retrieve top-N entries using an approximate k-NN algorithm, as N increases, accuracy or the efficiency of the search decreases. Simply speaking, it is easier to carry out an accurate 1-NN search than an accurate 500-NN search.

One notable exception is a recent paper by Brokos et al. [9] who, in contrast to our findings, learned that the cosine-similarity between averaged word embeddings is an effective model for retrieving Pubmed abstracts. However, they do not compare against standard IR baselines such as BM25, which makes an interpretation of their finding difficult.

We argue that instead of relying on the cheap cosine similarity it may be better to employ an expensive but more accurate similarity function. The exact brute force search using this function would be expensive, but the cost could be reduced by applying an approximate search method for generic—i.e., not necessarily metric—spaces.

A common approach to non-metric space indexing involves projecting data to a low-dimensional Euclidean space. The goal is to find a mapping without a large distortion of the original similarity measure. Jacobs et al. [29] review projection methods and argue that such a coercion is often against the nature of a similarity measure, which can be, e.g., intrinsically non-symmetric.

Among other factors, the lack of symmetry prevents us from using the kernelized LSH [34, 47]. The only LSH variant that might be directly applicable in our case is the Distance-Based Hashing (DBH) [3], which uses randomly selected pivots to project points to a one-dimensional space via FastMap [20]. The space is further binarized so that approximately one half of data points are mapped to one, and the other half is mapped to zero.

While a detailed comparison of pivoting approaches to DBH is out the scope of the paper, we hypothesize that performance of DBH—like performance of NAPP—depends on the choice of pivots. In the case of NAPP, we have found that composing pivots from randomly selected terms allows us to achieve *substantially* better performance than selecting pivots randomly. Thus, engineering pivots to support effective searching in a non-metric space seems to be an important research area. Results obtained from this area will likely benefit both DBH and NAPP.

Proximity graphs (see § 2.2) is another promising class of distance-based methods, which are shown to be useful in non-metric spaces [49]. In this work we employ the SW-graph [39], which works quite well for dense vector spaces. However, it has been less useful for *BM25* and *BM25+Model 1*. We have not been able to understand what causes the lack of performance, but this remains an important research question as well.

Finally, we want to highlight the relationship of our approach to indexing automatically learned features for QA [76]. Yao et al. propose to automatically learn associations between a question type and various linguistic annotations such as named entities and POS tags [76]. For example, for a question "Who is the president of the United States" an answer sentence contains a person name (a named entity). Given a training corpus, we can automatically learn associations and exploit them to guide the retrieval process. Technically, this requires indexing linguistic annotations and carrying out a query expansion by adding expected annotations to the query.

For efficiency reasons, this works well only if we can find few strong associations for a query. To demonstrate that this is not true in the case of the vocabulary gap, we compute effectiveness of *BM25+Model 1* for varying sizes of the translation table. Specifically, we prune all the entries $T(q|a)$ below a threshold. In addition, we estimate the average number of non-zero translation entries $T(q|a)$ associated with a *single* query term. As a reference point we also include data for *BM25*. We present results only for *Comprehensive*, because results for *Stack Overflow* are analogous.

Minimum translation probability	P@1	Number of associated terms
BM25	0.065	N/A
0.1	0.066 (+2.6%)	1700
0.05	0.070 (+8.6%)	3800
0.025	0.073 (+12.5%)	6200
0.005	0.077 (+19.3%)	12000
0.0025	0.079 (+21.6%)	15000

Table 5: Average number of terms associated with a query term at various performance levels of *BM25+Model 1* (estimated on *dev2*). The first row represents a *BM25* run.

According to Table 5, outperforming *BM25* by about 20% requires to keep more than 10K associations per query term (on average). This number is so high because frequent words, which tend to appear in queries and text, are associated with many less frequent words (i.e., respective translation probabilities are non-zero).

If we keep only translation entries with high (≥ 0.1) probabilities, the improvement over BM25 is merely 2.6%. Yet, we still have to keep nearly 2K associations per query term! This further corroborates the finding of Furnas et al. [24] that accurate retrieval requires using a large number of term aliases, which is hard to implement using term-based indices. Yet, it is possible to do within a framework of the k-NN search.

That said, the proposed methods are likely have limitations as well. For example, for both data sets employed in our experiments, the queries are quite long. It is not yet clear if k-NN can be applied to shorter ad hoc queries, which are frequently submitted to Web search engines.

5. CONCLUSION

In this paper we attempt to replace the classic term-based retrieval with the k-NN search. To this end, we train a linguistically motivated *non-metric* and *non-symmetric* similarity function: a weighted combination of BM25 scores and IBM Model 1 log-scores. Then, we demonstrate that it is possible to carry out an *efficient* and *effective* approximate k-NN search using this function.

An exact brute-force k-NN search using this similarity function is slow. Yet, an approximate algorithm can be nearly two orders of magnitude faster at the expense of only a small loss in accuracy. A retrieval pipeline using an approximate k-NN search can be sometimes both faster and more accurate compared to the term-based Lucene pipeline (see Table 3). The success of our approach stems from the novel combination of existing methods and new algorithmic tricks to compute IBM Model 1 efficiently.

While the k-NN search has been previously applied to IR and NLP problems [53, 65, 37, 54, 46, 9], the previous work focuses largely on the cosine similarity and LSH methods (see § 4 for a discussion). This is the first successful attempt to apply a *generic* k-NN search algorithm to a similarity function as challenging as a combination of BM25 and IBM Model 1. In that, we find that the cosine similarity alone (in particular, the cosine similarity between averaged word embeddings) lacks a lot in effectiveness (see Figure 2).

The focus of our study is on techniques that bridge the vocabulary gap. Yet, our methods are generic in the sense that they can be used to model various types of semantic and syntactic mismatch [6, 76]. This opens up new possibilities for designing effective retrieval pipelines.

Our software (including data-generating code) and derivative data based on the *Stack Overflow* collection is available online.[17]

Acknowledgements

Leonid Boytsov is supported by the Open Advancement of Question Answering Systems (OAQA) group.[18]; David Novak is supported by the Czech Research Foundation project P103/12/G084; Yury Malkov is supported by the Russian Foundation for Basic Research (project No. 16-31-60104 mol_a_dk).

We also thank Di Wang for helping with a Lucene baseline; Chris Dyer for a discussion of IBM Model 1 efficiency; Yoav Goldberg, Manaal Faruqui, Chenyan Xiong, Ruey-Cheng Chen for discussions related to word embeddings; Michael Denkowski for a discussion on approximating alignment scores.

6. REFERENCES

[1] A. Andoni, P. Indyk, T. Laarhoven, I. Razenshteyn, and L. Schmidt. Practical and optimal lsh for angular distance. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 1225–1233. Curran Associates, Inc., 2015.

[2] S. Arya and D. M. Mount. Approximate nearest neighbor queries in fixed dimensions. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, pages 271–280. Society for Industrial and Applied Mathematics, 1993.

[3] V. Athitsos, M. Potamias, P. Papapetrou, and G. Kollios. Nearest neighbor retrieval using distance-based hashing. In *2008 IEEE 24th International Conference on Data Engineering*, pages 327–336. IEEE, 2008.

[4] A. Berger, R. Caruana, D. Cohn, D. Freitag, and V. Mittal. Bridging the lexical chasm: Statistical approaches to answer-finding. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '00, pages 192–199, New York, NY, USA, 2000. ACM.

[5] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In *Database theory–ICDT'99*, pages 217–235. Springer, 1999.

[6] M. W. Bilotti, P. Ogilvie, J. Callan, and E. Nyberg. Structured retrieval for question answering. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 351–358. ACM, 2007.

[7] L. Boytsov and B. Naidan. Engineering efficient and effective non-metric space library. In *Similarity Search and Applications*, pages 280–293. Springer, 2013.

[8] A. Z. Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings*, pages 21–29. IEEE, 1997.

[9] G.-I. Brokos, P. Malakasiotis, and I. Androutsopoulos. Using centroids of word embeddings and word moverâĂŹs distance for biomedical document retrieval in question answering. In *Proceedings of the BIONLP 2016 Workshop*, 2016.

[10] P. F. Brown, V. J. D. Pietra, S. A. D. Pietra, and R. L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311, 1993.

[11] D. Carmel, D. Cohen, R. Fagin, E. Farchi, M. Herscovici, Y. S. Maarek, and A. Soffer. Static index pruning for information retrieval systems. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '01, pages 43–50, New York, NY, USA, 2001. ACM.

[12] C. Carpineto and G. Romano. A survey of automatic query expansion in information retrieval. *ACM Comput. Surv.*, 44(1):1:1–1:50, Jan. 2012.

[13] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín. Searching in metric spaces. *ACM Comput. Surv.*, 33(3):273–321, 2001.

[14] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 12:2493–2537, Nov. 2011.

[15] S. Ding and T. Suel. Faster top-k document retrieval using block-max indexes. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, pages 993–1002, New York, NY, USA, 2011. ACM.

[16] W. Dong, C. Moses, and K. Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th international conference on WWW*, pages 577–586. ACM, 2011.

[17] W. Dong, Z. Wang, W. Josephson, M. Charikar, and K. Li. Modeling lsh for performance tuning. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM '08, pages 669–678, New York, NY, USA, 2008. ACM.

[18] A. Echihabi and D. Marcu. A noisy-channel approach to question answering. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, ACL '03, pages 16–23, Stroudsburg, PA, USA, 2003. ACL.

[19] R. Eckart de Castilho and I. Gurevych. A broad-coverage collection of portable nlp components for building shareable analysis pipelines. In *Proceedings of the Workshop on Open Infrastructures and Analysis Frameworks for HLT*, pages 1–11, Dublin, Ireland, August 2014. ACL and Dublin City University.

[20] C. Faloutsos and K.-I. Lin. Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and

[17] https://github.com/oaqa/knn4qa

[18] https://oaqa.github.io/

multimedia datasets. *SIGMOD Rec.*, 24(2):163–174, May 1995.

[21] M. Faruqui, J. Dodge, S. K. Jauhar, C. Dyer, E. Hovy, and N. A. Smith. Retrofitting word vectors to semantic lexicons. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1606–1615, Denver, Colorado, May–June 2015. ACL.

[22] J. F. Fenlon. *The Catholic Encyclopedia*, volume 4. 1913. http://en.wikisource.org/wiki/Catholic_Encyclopedia_%281913%29/Concordances_of_the_Bible [Last Checked March 2016].

[23] D. Fried, P. Jansen, G. Hahn-Powell, M. Surdeanu, and P. Clark. Higher-order lexical semantic models for non-factoid answer reranking. *Transactions of the ACL*, 3:197–210, 2015.

[24] G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais. The vocabulary problem in human-system communication. *Communications of the ACM*, 30(11):964–971, 1987.

[25] K. Hajebi, Y. Abbasi-Yadkori, H. Shahbazi, and H. Zhang. Fast approximate nearest-neighbor search with k-nearest neighbor graph. In *Proceedings of the 22d International Joint Conference on Artificial Intelligence*, IJCAI'11, pages 1312–1317. AAAI Press, 2011.

[26] M. E. Houle and J. Sakuma. Fast approximate similarity search in extremely high-dimensional data sets. In *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, pages 619–630. IEEE, 2005.

[27] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the 30th annual ACM symposium on Theory of computing*, pages 604–613. ACM, 1998.

[28] M. Iyyer, J. Boyd-Graber, L. Claudino, R. Socher, and H. Daumé III. A neural network for factoid question answering over paragraphs. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 633–644, Doha, Qatar, October 2014. ACL.

[29] D. Jacobs, D. Weinshall, and Y. Gdalyahu. Classification with nonmetric distances: Image retrieval and class representation. *Pattern Analysis and Machine Intelligence*, 22(6):583–600, 2000.

[30] J. Jeon, W. B. Croft, and J. H. Lee. Finding similar questions in large question and answer archives. In *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, CIKM '05, pages 84–90, New York, NY, USA, 2005. ACM.

[31] R. Kallman, H. Kimura, J. Natkins, A. Pavlo, A. Rasin, S. Zdonik, E. P. C. Jones, S. Madden, M. Stonebraker, Y. Zhang, J. Hugg, and D. J. Abadi. H-store: A high-performance, distributed main memory transaction processing system. *Proc. VLDB Endow.*, 1(2):1496–1499, Aug. 2008.

[32] D. Konopnicki and O. Shmueli. Database-inspired search. In *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB '05, pages 2–12. VLDB Endowment, 2005.

[33] F. Korn, B.-U. Pagel, and C. Faloutsos. On the "dimensionality curse" and the "self-similarity blessing". *Knowledge and Data Engineering, IEEE Transactions on*, 13(1):96–111, 2001.

[34] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2130–2137. IEEE, 2009.

[35] E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. In *Proceedings of the 30th annual ACM symposium on Theory of computing*, STOC '98, pages 614–623. ACM, 1998.

[36] Q. Le and T. Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1188–1196, 2014.

[37] H. Li, W. Liu, and H. Ji. Two-stage hashing for fast document retrieval. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 495–500, Baltimore, Maryland, June 2014. ACL.

[38] T.-Y. Liu. Learning to rank for information retrieval. *Found. Trends Inf. Retr.*, 3(3):225–331, Mar. 2009.

[39] Y. Malkov, A. Ponomarenko, A. Logvinov, and V. Krylov. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, 45:61–68, 2014.

[40] Y. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. *ArXiv e-prints*, Mar. 2016.

[41] C. D. Manning, P. Raghavan, H. Schütze, et al. *Introduction to information retrieval*, volume 1. Cambridge University Press, 2008.

[42] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky. The Stanford CoreNLP natural language processing toolkit. In *ACL System Demonstrations*, pages 55–60, 2014.

[43] D. Metzler and W. Bruce Croft. Linear feature-based models for information retrieval. *Information Retrieval*, 10(3):257–274, 2007.

[44] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[45] T. Mikolov, W.-t. Yih, and G. Zweig. Linguistic regularities in continuous space word representations. In *HLT-NAACL*, pages 746–751, 2013.

[46] S. Moran, R. McCreadie, C. Macdonald, and I. Ounis. Enhancing first story detection using word embeddings. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16. ACM, 2016.

[47] Y. Mu and S. Yan. Non-metric locality-sensitive hashing. In *AAAI*, 2010.

[48] M. Muja and D. G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36(11):2227–2240, 2014.

[49] B. Naidan, L. Boytsov, and E. Nyberg. Permutation search methods are efficient, yet faster search is possible. *Proceedings of the VLDB Endowment*, 8(12):1618–1629, 2015.

[50] F. J. Och and H. Ney. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51, 2003.

[51] J. Pennington, R. Socher, and C. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language*

Processing (EMNLP), pages 1532–1543, Doha, Qatar, October 2014. ACL.

[52] V. Pestov. Lower bounds on performance of metric tree indexing schemes for exact similarity search in high dimensions. *Algorithmica*, 66(2):310–328, 2012.

[53] S. Petrović, M. Osborne, and V. Lavrenko. Streaming first story detection with application to twitter. In *NAACL'10*, pages 181–189. Association for Computational Linguistics, 2010.

[54] S. Petrović, M. Osborne, and V. Lavrenko. Using paraphrases for improving first story detection in news and twitter. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, NAACL HLT '12, pages 338–346, Stroudsburg, PA, USA, 2012. ACL.

[55] J. M. Ponte and W. B. Croft. A language modeling approach to information retrieval. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 275–281. ACM, 1998.

[56] S. Riezler and Y. Liu. Query rewriting using monolingual statistical machine translation. *Computational Linguistics*, 36(3):569–582, 2010.

[57] S. Riezler, A. Vasserman, I. Tsochantaridis, V. Mittal, and Y. Liu. Statistical machine translation for query expansion in answer retrieval. In *Proceedings of the 45th Annual Meeting of the ACL*, pages 464–471, Prague, Czech Republic, June 2007. ACL.

[58] S. Robertson. Understanding inverse document frequency: On theoretical arguments for IDF. *Journal of Documentation*, 60:503–520, 2004.

[59] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986.

[60] T. B. Sebastian and B. B. Kimia. Metric-based shape retrieval in large databases. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, volume 3, pages 291–296. IEEE, 2002.

[61] T. Skopal. On fast non-metric similarity search by metric access methods. In *Advances in Database Technology-EDBT 2006*, pages 718–736. Springer, 2006.

[62] R. Soricut and E. Brill. Automatic question answering using the web: Beyond the factoid. *Information Retrieval*, 9(2):191–206, 2006.

[63] M. Surdeanu, M. Ciaramita, and H. Zaragoza. Learning to rank answers to non-factoid questions from web collections. *Computational Linguistics*, 37(2):351–383, 2011.

[64] E. S. Tellez, E. Chávez, and G. Navarro. Succinct nearest neighbor search. *Information Systems*, 38(7):1019–1030, 2013.

[65] F. Ture, T. Elsayed, and J. Lin. No free lunch: Brute force vs. locality-sensitive hashing for cross-lingual pairwise similarity. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, pages 943–952, New York, NY, USA, 2011. ACM.

[66] P. D. Turney. Human-level performance on word analogy questions by latent relational analysis. *arXiv preprint cs/0412024*, 2004.

[67] H. Turtle and J. Flood. Query evaluation: Strategies and optimizations. *Inf. Process. Manage.*, 31(6):831–850, Nov. 1995.

[68] S. Vigna. Quasi-succinct indices. In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, WSDM '13, pages 83–92, New York, NY, USA, 2013. ACM.

[69] J. Wang, H. T. Shen, J. Song, and J. Ji. Hashing for similarity search: A survey. *arXiv preprint arXiv:1408.2927*, 2014.

[70] J. Wang, J. Wang, G. Zeng, R. Gan, S. Li, and B. Guo. Fast neighborhood graph search using cartesian concatenation. In *Multimedia Data Mining & Analytics*, pages 397–417. Springer, 2015.

[71] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, volume 98, pages 194–205, 1998.

[72] J. S. Whissell and C. L. Clarke. Effective measures for inter-document similarity. In *Proceedings of the 22Nd ACM International Conference on Information & Knowledge Management*, CIKM '13, pages 1361–1370, New York, NY, USA, 2013. ACM.

[73] J. Wieting, M. Bansal, K. Gimpel, and K. Livescu. Towards universal paraphrastic sentence embeddings. *CoRR*, abs/1511.08198, 2015.

[74] X. Xue, J. Jeon, and W. B. Croft. Retrieval models for question and answer archives. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '08, pages 475–482. ACM, 2008.

[75] L. Yang, Q. Ai, D. Spina, R.-C. Chen, L. Pang, W. B. Croft, J. Guo, and F. Scholer. *Advances in Information Retrieval: 38th European Conference on IR Research, ECIR 2016, Padua, Italy, March 20-23, 2016. Proceedings*, chapter Beyond Factoid QA: Effective Methods for Non-factoid Answer Sentence Retrieval, pages 115–128. Springer International Publishing, Cham, 2016.

[76] X. Yao, B. Van Durme, and P. Clark. Automatic coupling of answer extraction and information retrieval. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 159–165, Sofia, Bulgaria, August 2013. ACL.

[77] L. Zhao and J. Callan. Term necessity prediction. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 259–268. ACM, 2010.