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

Leonid Boytsov, David Novak, Yury Malkov, Eric Nyberg

October 2016

Abstract

This is a slightly expanded version of our CIKM'2016 talk that presents the eponymous paper. Note that is a only high-level summary and more details can be obtained from our paper and code. The software, instructions, and bootstrapping scripts are published on GitHub.

1 Motivation

The talk is concerned with our ongoing effort to apply *generic* k-nearest-neighbor (k-NN) search algorithms to textual retrieval. Our work is at the intersection of natural language processing (NLP), information retrieval (IR), and k-nearest-neighbor search (k-NN search). It is motivated by our belief that adhoc text retrieval could and should benefit from using generic k-NN search algorithms. When we discussed the idea of substituting the k-NN search for term-based retrieval with several colleagues, they opined that this approach would be *horribly* [sic] slow. We demonstrate that this is not the case, which is one of the central contributions of our paper.

A typical retrieval system relies on a filter-and-refine pipeline, which first generates a list of candidate documents using a simple TF×IDF ranking function and applying more sophisticated similarity at later refinement steps. The filtering step is, in fact, a form of k-NN search, where similarity is the inner product between sparse query and document vectors. Unfortunately, if this *crude* similarity fails to retrieve relevant documents at the filtering step, they cannot be recovered at later refinements steps.

Many filtering errors stem from a vocabulary mismatch. As shown by Zhao and Callan [8], there is a 50% chance that a query term will not appear in a relevant document. Fixing this problem is hard, because, as shown by Furnas and colleagues [3], there is a long tail of synonyms. Yet, only few of them can be efficiently incorporated into query expansion.

Furthermore, it is not clear how to efficiently incorporate sophisticated similarity features such as dense vector document representations (i.e., document and word embeddings) directly into retrieval. Such representations already work

quite well for classification and collaborative filtering [2] and we expect them to be useful in text retrieval too.

1.1 Hardness of k-NN Search

Unfortunately, k-NN search is a hard problem due to the *curse of dimensionality*, which in many cases does not permit an exact and efficient k-NN search on high-dimensional data. In rare cases, a structure of the search problem does permit a reasonably efficient computational shortcut. For example, as pointed out by James Allan, in the case of TF×IDF similarity and short queries, the *sparsity* of the term-document matrix allows us to answer query efficiently using inverted files. However, we think that the curse of dimensionality cannot be fully removed and that computational shortcuts rarely exist. We can nevertheless weaken the curse using the magic of approximation. Approximate search can be substantially more efficient at the expense of missing some relevant answers.

To replace or complement term-based retrieval with k-NN search we need to achieve two challenging objectives. First we need to design a simple similarity that beats baselines such as BM25 by a good margin. Because textual data sets are usually high-dimensional, we have to resort to approximate searching. Therefore, the gains in accuracy achieved by employing a more sophisticated model can be easily invalidated by the inaccuracy of the search procedure. To avoid this *degradation by approximation*, we also need to implement efficient search algorithms that are as accurate as possible.

1.2 Similarity Models for k-NN Search

After some experimentation, we selected the following two models, each of which had a potential to alleviate the problem of the vocabulary mismatch. First, we considered a sparse representation where we combine BM25 scores and log-scores of IBM Model 1. There is prior work showing this combination to be 20% more effective than BM25 alone [6]. Second, we considered dense document representations obtained by averaging individual word embeddings, which are compared using the cosine similarity.

The appeal of word embeddings is twofold:

- They deliver state-of-the art performance in many NLP tasks (while bridging the vocabulary gap);
- There are several algorithms for efficient retrieval of dense vectors.

In the following, I briefly introduce IBM Model 1. I skip the description of averaged word embeddings, because they are well-known. IBM Model 1 is a statistical *lexical* translation model, which is learned from a *parallel* corpus. A good example of a parallel corpus is a set of question answer pairs extracted from a community QA website. Similar to Berger and Lafferty who pioneered this approach in IR, we use Model 1 to quantify a strength of *association* between query terms and terms from respective relevant documents.

Question word	Answer word		Answer word		Answer word		Answer word	
caffeine	coffee	0.074	drink	0.051	tea	0.021	caffiene	0.018
latte	milk	0.052	coffee	0.049	starbucks	0.043	espresso	0.021
oxygen	o2	0.043	air	0.024	gas	0.018	carbon	0.018
contagious	spread	0.024	person	0.022	infection	0.013	virus	0.013
likelihood	likely	0.015	chance	0.015	chances	0.013	odds	0.013
animal	animals	0.110	dog	0.037	dogs	0.019	cat	0.017
study	studying	0.037	school	0.021	studies	0.018	exam	0.013
challenge	like	0.016	challenges	0.015	think	0.012	all	0.010
fat	weight	0.061	eat	0.027	lose	0.024	body	0.023

Table 1: Sample translation probabilities learned from Yahoo Answers

Let us look at the few sample associations learned from Yahoo Answers corpus, which are presented in Table 1. The leftmost column shows question terms. The remaining columns show associated answers with their respective translation probabilities. As we can see here, Model 1 can *successfully* learn various morphological and semantic relationships including synonymy, meronymy, and hypernymy. Look for example at row one. We can see that `caffeine` is associated with words `coffee`, `drink`, and `tea`, as well as with its *incorrect* spelling. Except misspelling, these are all examples of meronymy, i.e., of a part-whole relationship. Consider the second table row. Here, the word `latte` is, somewhat unsurprisingly, associated with `milk`, `coffee`, `starbucks`, and `espresso`. The pair of words `latte` and `coffee` is an example of hypernymy.

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

Computing Model 1 scores involves two steps: retrieving translation probabilities for all pairs of query and document terms and aggregating these probabilities into a single score.

For completeness, we present IBM Model 1 formula in Eq. 1, where Q and D denote query and answer document; q and d denote query and document terms; and $T(q|d)$ denote translation probabilities (which estimate the strength of term associations). Note, however, that exact computation details are *irrelevant* for our discussion. Yet, it is important to understand that retrieving all pairwise probabilities is too expensive for k-NN search to be efficient. Thus, we use simple algorithmic tricks (briefly outlined in the paper) to avoid doing so.

2 k-NN Search Algorithms

As mentioned previously, our second big challenge is to implement search algorithms that answer queries with nearly 100% accuracy. In the beginning of

this project we had high expectations for a class of algorithms called neighborhood or proximity graphs. In a proximity graph, data points are nodes and sufficiently close points are connected by edges. A search procedure is a semi-greedy traversal of the graph. In particular, we used a variant of the proximity graph called SW-graph co-authored by Yury Malkov [4]. For our joint paper, Yury proposed an algorithmic improvement as well as helped identify an inefficiency in handling the priority queue. These contributions reduced retrieval times by nearly an order of magnitude.

Proximity graphs are extremely efficient for high-dimensional Euclidean data sets, where they can outperform both LSH algorithms and multi-tree approaches with the single priority queue [5, 1]. In addition, we also found that proximity graphs work nearly as well for a number of *non-metric* and *non-symmetric* distances such as KL-divergence [5] or Rényi divergence (unpublished experiments). The fact that proximity graphs work for a variety of crazy distances made us think that we have a *unicorn* method, which we could apply it to even more complex similarities.

However, SW-graph worked well only for dense representations based on averaged word embeddings, but not for similarities based on sparse representations. Our recent preliminary experiments show that it is possible to make SW-graph work with BM25, but we do not know how to make SW-graph work well for the combination of BM25 and Model 1.

Leonid also implemented a back up solution based on pivoting (with pivots randomly selected from data points). This method was not terribly slow, but it was not sufficiently fast either. Fortunately, at this point David Novak joined the project and proposed a simple but effective way to generate pivots. He demonstrated that this method was quite efficient for the cosine similarity and Wikipedia sparse TF×IDF vectors. Leonid was able to reproduce David's results and successfully applied his technique to BM25 as well as to the combination of BM25 and Model 1.

3 Experiments

Let us now talk about experiments. Our task is adhoc text retrieval. We use two publicly available community QA data sets: Yahoo Answers and Stack Overflow. We do not use all the answers, but only answers that are considered best answers by the question asker or other community members. In that, questions play the role of queries, while best answers are considered to be *the only* relevant documents. Each collection is as a parallel corpus of question-answer pairs, which allows us to train IBM Model 1. IBM Model 1 is trained on a part of the collection that is *not* used for training, development, and/or testing.

The main results are presented in Figure 2. Here we show efficiency-effectiveness plots for two collections and three similarity models. One model, i.e., BM25 is a baseline mode. Efficiency (on the y-axis) is measured in milliseconds per query. Note the log-scale of y-axis. Effectiveness (on the x-axis) is measured

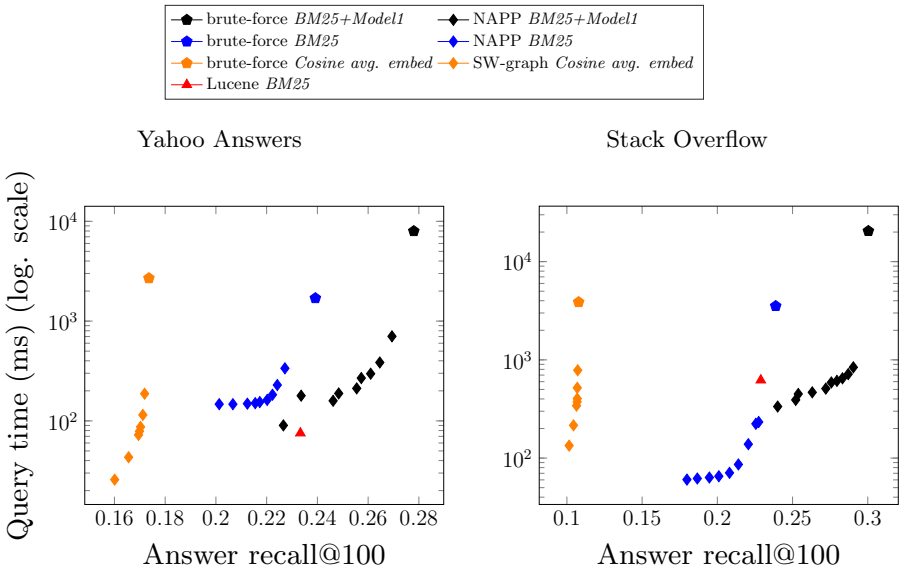

Table 2: Efficiency-effectiveness Trade-Offs of k-NN Search (higher and to the right is better)

as answer recall at rank 100. Lower and to the right is better.

In the case of sparse representations, we use BM25 as well BM25+Model1, which are indexed using the pivoting method NAPP. In the case of dense representations, we use the cosine similarity and averaged word embeddings, which are indexed with SW-graph. Approximate k-NN search runs are represented by diamonds. There are multiple runs corresponding to different similarities and search parameters. Approximate k-NN search is compared against two baselines:

- *Exact* brute force k-NN search, which is represented by pentagons;
- Lucene BM25, which is represented by red triangles. Lucene is a strong baseline, which fares well against optimized C++ code, especially for disjunctive queries [7]. [1]

[1] More exactly we do a bit of re-ranking of Lucene's output to compensate for the deficiency of Lucene's BM25 implementation. See our blog post with details.

First we can see that exact brute-force k-NN search is, indeed, *horribly* slow, but corresponding approximate runs take *substantially* shorter times without losing much in accuracy. In that, SW-graph runs (see these orange diamonds here and here) exhibit the least degradation due to approximation, but the underlying similarity model is not effective. In other words, the averaged embeddings runs are not effective, but this is not a fault of k-NN search.

More importantly, these plots seem to *support* our conjecture that incorporation of sophisticated similarity into retrieval can be useful. Look at the left pane: blue diamonds correspond to BM25; black diamonds correspond to BM25 + Model 1. In both cases, the search method is NAPP. BM25 + Model 1 is computationally more expensive than BM25: The corresponding brute-force k-NN search is, in fact, an order of magnitude slower! Yet, black diamond runs are roughly as efficient as blue diamond runs, while being more effective. Look at the right pane. Here, black diamond runs can outperform Lucene—which uses a simpler BM25 similarity—in both accuracy and speed.

4 Discussion and conclusion

We have finally come to a conclusion and I want to discuss our positive and negative results. First of all, generic k-NN search is not horribly slow. In fact, it can be comparably efficient to Lucene, which is our practicality threshold. Our efficient k-NN search relies on pivoting. In that, pivoting based on more effective (but slower-to-compute) similarity can be more efficient than pivoting based on a simpler and cheaper BM25 similarity. We believe that these results support our main conclusions:

- Generic k-NN search can be efficient *and* accurate in a text retrieval task;
- It may be, *indeed*, beneficial to employ sophisticated similarity at an early retrieval stage.

Thus, text retrieval can and should benefit from using generic k-NN search algorithms.

Although Lucene is fast [7], we know that C++ code implementing a similar algorithm is still faster. Unfortunately, we do not have an apple-to-apple comparison results right now. Yet, we know that C++ code that does not use compression and does not compute BM25[2] is nearly 2× fast compared to Lucene. Likely, such a code would be about 30% faster compared to our fastest black diamond runs (but 10% less effective). It remains to be seen if our method can be further optimized.

It is a great pity but proximity graphs seem to have limitations. They work great for a variety of crazy non-metric distance functions as well as for word embeddings. Unfortunately, word embeddings are not effective on their own.

We also admit that we yet have no example of k-NN search finding lots of additional documents compared to a filter-and-refine approach (based on

[2]because it simulates computation via inner product with precomputed vector values.

TF×IDF). Even though this is not shown in plots, a nearly effective solution can be obtained by simply using a larger pool of candidate entries. This is not related to the accuracy of k-NN search, because the same is true for exact brute force search using BM25+Model 1. We hypothesize that the classic TF×IDF search may be effective here for two reasons:

- Our collections are small;
- *Long* queries answered in a *fully* disjunctive mode are relatively unaffected by the vocabulary mismatch, because there is a good chance that both query and relevant documents share at least one common term.

In our work we explored three types of similarities: two are based on sparse representations and two are based on dense ones. It does not mean that our methods are limited to these similarities only. In contrast, we think that k-NN search can work for a variety of complex similarities (going beyond bridging the vocabulary gap) and data representations. It seems to us that sparse document representations work well for adhoc search and are hard to beat in this domain. We also know that dense representations work well for, collaborative filtering, where sparse bag-of-words representations do not perform well [2]. If we can substantially improve dense representations and build better similarity models for them, the hybrid models may become viable in the future.

References

[1] E. Bernhardsson. Benchmarks of approximate nearest neighbor libraries in python. `https://github.com/erikbern/ann-benchmarks` Using single-thread results from June 2016.

[2] E. Bernhardsson. Collaborative filtering at spotify. NYC machine learning meetup. Jan 17, 2013. `http://www.slideshare.net/erikbern/collaborative-filtering-at-spotify-16182818` Last checked October 2016.

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

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

[5] B. Naidan, L. Boytsov, and E. Nyberg. Permutation search methods are efficient, yet faster search is possible. *PVLDB*, 8(12):1618–1629, 2015.

[6] 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.

[7] S. Vigna. Quasi-succinct indices. In *Proceedings of WSDM*, pages 83–92. ACM, 2013.

[8] L. Zhao and J. Callan. Term necessity prediction. In *Proceedings of the 19th ACM Conference on Information and Knowledge Management, CIKM 2010, Toronto, Ontario, Canada, October 26-30, 2010*, pages 259–268. ACM, 2010.