Flexible retrieval with NMSLIB and FlexNeuART

Leonid Boytsov *
Pittsburgh, PA, USA
leo@boytsov.info

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

Abstract

Our objective is to introduce to the NLP community an existing k-NN search library NMSLIB, a new retrieval toolkit FlexNeuART, as well as their integration capabilities. NMSLIB, while being one the fastest k-NN search libraries, is quite generic and supports a variety of distance/similarity functions. Because the library relies on the distance-based structure-agnostic algorithms, it can be further extended by adding new distances. FlexNeuART is a modular, extendible and flexible toolkit for candidate generation in IR and QA applications, which supports mixing of classic and neural ranking signals. FlexNeuART can *efficiently* retrieve *mixed* dense and sparse representations (with weights learned from training data), which is achieved by extending NMSLIB. In that, other retrieval systems work with purely sparse representations (e.g., Lucene), purely dense representations (e.g., FAISS and Annoy), or only perform mixing at the re-ranking stage.

1 Introduction

Although there has been substantial progress on machine reading tasks using neural models such as BERT (Devlin et al., 2018), these approaches have practical limitations for open-domain challenges, which typically require (1) a retrieval and (2) a re-scoring/re-ranking step to restrict the number of candidate documents. Otherwise, the application of state-of-the-art machine reading models to large document collections would be impractical even with recent efficiency improvements (Khattab and Zaharia, 2020).

The first retrieval stage is commonly referred to as the *candidate generation* (i.e., we generate candidates for re-scoring). Until about 2019, the candidate generation would exclusively rely on a traditional search engine such as Lucene,[1] which indexes occurrences of individual terms, their lemmas or stems (Manning et al., 2010). In that, there are several recent papers where promising results were achieved by generating dense embeddings and using a k-NN search library to retrieve them (Lee et al., 2019; Karpukhin et al., 2020; Xiong et al., 2020). However, these studies typically have at least one of the following flaws: (1) they compare against a weak baseline such as untuned BM25 or (2) they rely on exact k-NN search, thus, totally ignoring practical efficiency-effectiveness and scalability trade-offs related to using k-NN search, see, e.g., §3.3 in Boytsov (2018). FlexNeuART implements some of the most effective non-neural ranking signals: It produced best non-neural runs in the TREC 2019 deep learning challenge (Craswell et al., 2020) and would be a good tool to verify these results.

Furthermore, there is evidence that when dense representations perform well, even better results may be obtained by combining them with traditional sparse-vector models (Seo et al., 2019; Gysel et al., 2018; Karpukhin et al., 2020; Kuzi et al., 2020). It is not straightforward to incorporate

*Work done primarily while at CMU.

[1] https://lucene.apache.org/

Proceedings of Second Workshop for NLP Open Source Software (NLP-OSS), pages 32–43

these representations into existing toolkits, but FlexNeuART supports dense and dense-sparse representations out of the box with the help of NMSLIB (Boytsov and Naidan, 2013a; Naidan et al., 2015a).[2] NMSLIB is an efficient library for k-NN search on CPU, which supports a wide range of similarity functions and data formats. NMSLIB is a commonly used library[3], which was recently adopted by Amazon.[4] Because NMSLIB algorithms are largely distance-agnostic, it is relatively easy to extend the library by adding new distances. In what follows we describe NMSLIB, FlexNeuART, and their integration in more detail. The code is publicly available:

- https://github.com/oaqa/FlexNeuART
- https://github.com/nmslib/nmslib

2 NMSLIB

Non-Metric Space Library (NMSLIB) is an efficient cross-platform similarity search library and a toolkit for evaluation of similarity search methods (Boytsov and Naidan, 2013a; Naidan et al., 2015a), which is the first commonly used library with a principled support for non-metric space searching.[5] NMSLIB is an extendible library, which means that is possible to add new search methods and distance functions. NMSLIB can be used directly in C++ and Python (via Python bindings). In addition, it is also possible to build a query server, which can be used from Java (or other languages supported by Apache Thrift[6]).

k-NN search is a conceptually simple procedure that consists in finding k data set elements that have highest similarity scores (or, alternatively, smallest distances) to another element called *query*. Despite its formulaic simplicity, k-NN search is a notoriously difficult problem, which is hard to do efficiently, i.e., faster than the brute-force scan of the data set, for high dimensional data and/or non-Euclidean distances. In particular, for some data sets exact search methods do not outperform the brute-force search in just a dozen of dimensions (see, e.g., a discussion in § 1 and § 2 of Boytsov 2018).

For sufficiently small data sets and simple similarities, e.g., L_2, the brute-force search can be a feasible solution, especially when the data set fits into a memory of an AI accelerator. In particular, the Facebook library for k-NN search FAISS (Johnson et al., 2017) supports the brute-force search on GPU [7]. However, GPU memory is quite limited compared to the main RAM. For example, the latest A100 GPU has only 40 GB of memory[8] while some commodity servers have 1+ TB of main RAM.

In addition, GPUs are designed primarily for dense-vector manipulations and have poor support for sparse vectors (Hong et al., 2018). When data is very sparse, as in the case of traditional text indices, it is possible to efficiently retrieve data using search toolkits such as Lucene. Yet, for less sparse sets, more complex similarities, and large dense-vector data sets we have to resort to *approximate* k-NN search, which does not have accuracy guarantees.

One particular efficient class of k-NN search methods relies on the construction of neighborhood graphs for data set points (see a recent survey by Shimomura et al. (2020) for a thorough description). Despite initial promising results were published nearly 30 years ago (Arya and Mount, 1993), this approach has only recently become popular due to good performance of NMSLIB and KGraph (Dong et al., 2011)[9].

Specifically, two successive ANN-Benchmarks challenges (Aumüller et al., 2019) were won first by our efficient implementation of the Navigable Small World (NSW) (Malkov et al., 2014) and then by the Hierarchical Navigable Small World (HNSW) contributed to NMSLIB by Yury Malkov (Malkov and Yashunin, 2018). HNSW performance was particularly impressive.

[2] https://github.com/nmslib/nmslib
[3] https://pypistats.org/packages/nmslib
[4] https://amzn.to/3aDCMtC
[5] https://github.com/nmslib/nmslib
[6] https://thrift.apache.org/
[7] https://github.com/facebookresearch/faiss/wiki/Running-on-GPUs
[8] https://www.nvidia.com/en-us/data-center/a100/
[9] https://github.com/aaalgo/kgraph

Unlike many other libraries for k-NN search, NMSLIB focuses on retrieval for generic similarities. The generality is achieved by relying largely on *distance-based* methods: NSW (Malkov et al., 2014), HNSW (Malkov and Yashunin, 2018), NAPP (Tellez et al., 2013; Boytsov et al., 2016), and an extension of the VP-tree (Boytsov and Naidan, 2013b; Boytsov and Nyberg, 2019b). Distance-based methods can only use values of the mutual data point distances, but cannot exploit the structure of the data, e.g., they have no direct access to vector elements or string characters. In addition, NMSLIB has a simple (no compression) implementation of a traditional inverted file, which can be used to carry out an *exact* maximum-inner product search on sparse vectors.

Graph-based retrieval algorithms have been shown to work efficiently for a variety of non-metric and non-symmetric distances (Boytsov and Nyberg, 2019a; Boytsov, 2018; Naidan et al., 2015b). This flexibility permits adding new distances/similarities with little effort (as we do not have to change the retrieval algorithms). However, this needs to be done in C++, which is one limitation. It is desirable to have an API where C++ code could call Python-implemented distances. NMSLIB supports only in-memory indices and with a single exception all indices are static, which is another (current) limitation of the library.

There is a number of data format and distances—a combination which we call a *space*—supported by NMSLIB. A detailed description can be found online[10]. Most importantly, the library supports L_p distances with the norm $\|x\|_p = \left(\sum_{i \in I} |x_i|^p\right)^{1/p}$, the cosine similarity, and the inner product similarity. For all of these, the data can be both fixed-size "dense" and variable-size "sparse" vectors. Sparse vectors can have an unlimited number of non-zero elements and their processing is less efficient compared to dense vectors. On Intel CPUs the processing is speed up using special SIMD operations. In addition, NMSLIB supports the Jaccard similarity, the Levenshtein distance (for ASCII strings), and the number of (more exotic) divergences (including the KL-divergence).

The library has substantial documentation and additional information can be found online[11].

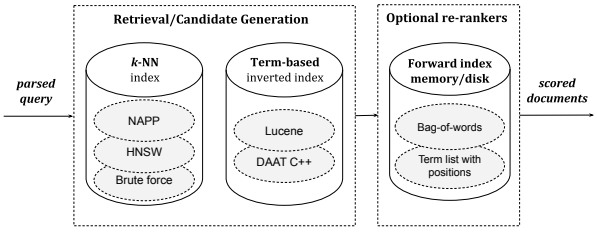

Figure 1: Retrieval Architecture and Workflow Overview

3 FlexNeuART

3.1 Motivation

Flexible classic and NeurAl Retrieval Toolkit, or shortly FlexNeuART (intended pronunciation flex-noo-art) is a modular text retrieval toolkit, which incorporates some of the best classic, i.e., traditional, information retrieval (IR) signals and provides capabilities for integration with recent neural models. This toolkit supports all key stages of the retrieval pipeline, including indexing, generation of training data, training the models, candidate generation, and re-ranking.

FlexNeuART has been under active development for several years and has been used for our own projects, in particular, to investigate applicability of k-NN search for text retrieval (Boytsov et al., 2016). It was also used in recent TREC evaluations (Craswell et al., 2020) as well as to produce strong runs on the MS MARCO document leaderboard.[12] The toolkit is geared towards TREC evaluations: For broader acceptance we would clearly need to implement Python bindings and experimentation code at the Python level.

[10] https://github.com/nmslib/nmslib/blob/master/manual/spaces.md

[11] https://github.com/nmslib/nmslib/tree/master/manual

[12] https://microsoft.github.io/msmarco/#docranking

FlexNeuART was created to fulfill the following needs:

- *Shallow* integration with Lucene and state-of-the-art toolkits for k-NN search (i.e., the candidate generation component should be easy to change);
- Efficient retrieval and efficient re-ranking with basic relevance signals;
- An out-of-the-box support for multi-field document ranking;
- An ease of implementation and/or use of most traditional ranking signals;
- An out-of-the-box support for learning-to-rank (LETOR) and basic experimentation;
- A support for mixed dense-sparse retrieval and/or re-ranking.

Packages most similar to ours in retrieval and LETOR capabilities are Anserini (Yang et al., 2018), Terrier (Ounis et al., 2006), and OpenNIR (MacAvaney, 2020). Anserini and Terrier are Java packages, which were recently enhanced with Python bindings through Pyserini[13] and PyTerrier (Macdonald and Tonellotto, 2020). OpenNIR implements re-ranking code on top of Anserini. These packages are tightly integrated with specific retrieval toolkits, which makes implementation of re-ranking components difficult, as these components need to access retrieval engine internals—which are frequently undocumented—to retrieve stored documents, term statistics, etc. Replacing the core retrieval component becomes problematic as well. In contrast, our system decouples retrieval and re-ranking modules by keeping an *independent* forward index, which enables *plugable* LETOR and IR modules. In addition to this, OpenNIR and Pyserini do not provide API for fusion of relevance signals and none of the toolkits incorporates a lexical translation model (Berger et al., 2000), which can substantially boost accuracy for QA.

[13] https://github.com/castorini/pyserini

```
{
 "DOCNO" : "0",
 "text" : "nfl team represent super bowl 50",
 "text_unlemm" : "nfl teams represented super bowl 50"
}
```

Figure 2: Sample input for question "Which NFL team represented the AFC at Super Bowl 50?"

3.2 System Design and Workflow

The FlexNeuART system—outlined in Figure 1—implements a classic multi-stage retrieval pipeline, where documents flow through a series of "funnels" that discard unpromising candidates using increasingly more complex and accurate ranking components. In that, FlexNeuART supports one intermediate and one final re-ranker (both are optional). The initial ranked set of documents is provided by the so-called *candidate generator* (also known as the *candidate provider*).

FlexNeuART is designed to work with plugable candidate generators and re-rankers. Out-of-the-box it supports Apache Lucene[14] and NMSLIB, which we describe in § 2. NMSLIB works as a standalone multi-threaded server implemented with Apache Thrift.[15] NMSLIB supports an efficient approximate (and in some cases exact) maximum inner-product search on sparse and sparse-dense representations. Sparse-dense retrieval is a recent addition.

Lucene full-text search algorithms rely on classic term-level inverted files, which are stored in compressed formats (so Lucene is quite space-efficient). NMSLIB (see § 2) supports the classic (uncompressed) inverted files with document-at-at-time (DAAT) processing, the brute-force search, the graph-based retrieval algorithms HNSW (Malkov and Yashunin, 2018) and NSW (Malkov et al., 2014), as well the pivoting algorithm NAPP (Tellez et al., 2013; Boytsov et al., 2016).

The indexing and querying pipelines ingest data (queries and documents) in the form of multi-field JSON entries, which are generated by external Java and/or Python

[14] https://lucene.apache.org/
[15] https://thrift.apache.org/

code. Each field can be *parsed* or *raw*. The parsed field contains *white-space* separated tokens while the raw field can keep arbitrary text, which is tokenized directly by re-ranking components. In particular, BERT models rely on their own tokenizers (Devlin et al., 2018).

The core system does not directly incorporate any text processing code, instead, we assume that an external pipeline does all the processing: parsing, tokenization, stopping, and possibly stemming/lemmatization to produce a string of white-space separated tokens. This relieves the indexing code from the need to do complicated parsing and offers extra flexibility in choosing parsing tools.

An example of a two-field input JSON entry for a SQuAD 1.1 (Rajpurkar et al., 2016) question is given in Fig. 2. Document and query entries contain at least two mandatory fields: `DOCNO` and `text`, which represent the document identifier and *indexable* text. Queries and documents may have additional optional fields. For example, HTML documents commonly have a `title` field. In Fig. 2, `text_unlemm` consists of lowercased original words, and `text` contains word lemmas. Stop words are removed from both fields. From our prior TREC experiments we learned that it is beneficial to combine scores obtained for the lemmatized (or stemmed) and the original text (Boytsov and Belova, 2011).

Retrieval requires a Lucene or an `NMSLIB` index, each of which can be created *independently*. To support re-ranking, we also need to create *forward* indices. There is one forward index for each data field. For parsed fields, it contains bag-of-word representations of documents (term IDs and frequencies) and (optionally) an ordered sequence of words. For raw fields, the index keeps unmodified text. A forward index is also required to create an `NMSLIB` index.

The `FlexNeuART` system has a configurable re-ranking module, which can combine results from several ranking components. A sample configuration file shown in Fig. 3

```
{"extractors": [
  {"type": "TFIDFSimilarity",
   "params": {
    "indexFieldName": "text",
    "queryFieldName": "text",
    "similType": "bm25",
    "k1": "1.2",
    "b": "0.75"}
  },
  {"type": "avgWordEmbed",
   "params": {
    "indexFieldName": "text_unlemm",
    "queryFieldName": "text_unlemm",
    "queryEmbedFile": "embeds/starspace_unlemm.query",
    "docEmbedFile": "embeds/starspace_unlemm.answer",
    "useIDFWeight": "True",
    "useL2Norm": "True",
    "distType": "l2"}
   }
]}
```

Figure 3: Sample scoring configuration.

contains an array of scoring sub-modules whose parameters are specified via nested dictionaries (in curly brackets). Each description contains the mandatory parameters `type` and `params`. Scoring modules are feature *extractors*, each of which produces one or more numerical feature that can be used by a LETOR component to train a ranking model or to score a candidate document.

The special composite feature extractor reads the configuration file and for each description of the extractor it creates an instance of the feature extractor whose type is defined by `type`. The value of `params` can be arbitrary: parsing and interpreting parameters is delegated to the constructor of the extractor object.

A sample configuration in Fig. 3 defines a BM25 (Robertson, 2004) scorer with parameters $k_1 = 1.2$ and $b = 0.25$ for the index field `text` (and query field `text`) as well as the averaged embedding generator for the fields `text_unlemm`. The latter creates dense query and document representations using StarSpace embeddings (Wu et al., 2018). There are separate sets of embeddings for queries and documents. Word embeddings are weighted using IDFs and subsequently L_2 normalized. Finally, this extractor produces a single feature equal to the L_2 distance between averaged embeddings of the query and the document.

From the forward indices, we can export

```
[
  {
    "experSubdir": "final_exper",
    "candProvAddConfParam" : "exper_desc/lucene.json",
    "extrType": "exper_desc/final_extr.json",
    "extrTypeInterm" : "exper_desc/interm_extr.json",
    "modelInterm" : "exper_desc/classic_ir.model",
    "candQty" : 2000,
    "testOnly": 0,
    "runId" : "sample_run_id"
  }
]
```

Figure 4: Sample experimental configuration.

data to NMSLIB and create an index for k-NN search. This is supported only for inner-product similarities. As discussed in the following subsection § 3.3, there are two scenarios. In the first scenario we export one vector per feature extractor. In particular, we generate a sparse vector for BM25 and a dense vector for the averaged embeddings. Then, `NMSLIB` combines these representations on its own using adjustable weights, which can be tweaked after data is exported. In the second scenario–which is more efficient but less flexible—we create one composite vector per document/query, where individual component weights cannot be changed further after export.

3.3 Scoring Modules

Similarity scores between queries and documents are computed for a pair of query and a document field (typically these are the same fields).[16] Scores from various scorers are then combined into a single score by a learning-to-rank (LETOR) algorithm (Liu et al., 2009). `FlexNeuART` use the LETOR library RankLib from which we use two particularly effective learning algorithms: a coordinate ascent (Metzler and Croft, 2007) and LambdaMART (Burges, 2010). We have found a bug in RankLib implementation of the coordinate ascent: We, thus, use our own, bugfixed, version.

Coordinate ascent produces a linear model. It is most effective when the number of features and/or the number of examples is small. LambdaMART is a boosted tree model, which, in our experience, is effective primarily when the number of features and training examples is quite large.

We provide basic experimentation support. An experiment is described via a JSON descriptor, which defines parameters of the candidate generating, re-ranking, and LETOR algorithms. Some experimentation parameters such as training and testing subsets can also be specified in the command line.

A sample descriptor is shown in Fig. 4. It uses an intermediate re-ranker which re-scores 2000 entries with the highest Lucene scores. A given number of highly scored entries can be further re-scored using the "final" re-ranker. Note that the experimental descriptor references feature-extractor JSONs rather than defining everything in a single configuration file.

Given an experimental descriptor, the training pipeline generates specified features, exports results to a special RankLib format and trains the model. Training of the LETOR model also requires a relevance file (a QREL file in the TREC NIST format), which lists known relevant documents. After training, the respective retrieval system is evaluated on another set of queries. The user can disable model training: This mode is used to tune BM25.

Based on our experience with TREC and community QA collections (Boytsov and Naidan, 2013b; Boytsov, 2018), we support the following scoring approaches:

- A proxy scorer that reads scores from one or more standalone scoring servers, which can be implemented in Python or any other language supported by Apache Thrift.[17]. Our system implements neural proxy scorers for CEDR (MacAvaney et al., 2019) and MatchZoo (Fan et al., 2017). We have modified CEDR by providing a better parameterization of the training procedure, adding support for BERT large (Devlin et al., 2018) and multi-GPU training.

[16] There can be multiple scorers for each pair of fields.

[17] `https://thrift.apache.org/`

- The **TF×IDF** similarity BM25 (Robertson, 2004), where logarithms of inverse document term frequencies (IDFs) are multiplied by normalized and smoothed term counts in a document (TFs).
- Sequential dependence model (Metzler and Croft, 2005): our re-implementation is based on the one from `Anserini`.
- BM25-based proximity scorer, which treats ordered and unordered pairs of query terms as a single token. It is similar to the proximity scorer used in our prior work (Boytsov and Belova, 2011).
- **Cosine/L_2 distance** between averaged *word* embeddings. We first train word embeddings for the corpus, then construct a dense vector for a document (or query) by applying TF×IDF weighting to the individual word embeddings and summing them. Then we compare averaged embeddings using the cosine similarity (or L_2 distance).
- **IBM Model 1** is a lexical translation model trained using expectation maximization. We use Model 1 to compute an alignment log-probability between queries and answer documents. Using Model 1 allows us to reduce the vocabulary gap between queries and documents (Berger et al., 2000).
- A proxy query- and document embedder, that produces fixed-size dense vectors for queries and documents. The similarity is the inner product between query and document embeddings. This scorer operates as an Apache Thrift server.
- A BM25-based pseudo-relevance feedback model RM3. Unlike a common approach where RM3 is used for query-expansion, we use it in re-ranking mode (Diaz, 2015).

Although `FlexNeuART` supports complex scoring models, these can be computationally too expensive to be used directly for retrieval (Boytsov et al., 2016; Boytsov, 2018). Instead we should stick to a simple vector-space model, where similarity is computed as the inner product between query and document vectors (Manning et al., 2010). The respective retrieval procedure is a maximum inner-product search (a form of k-NN search). For example both BM25 and the cosine similarity between query and document embeddings belong to this class of scorers.

Under the vector-space framework we need to (1) generate/read a set of field-specific vectors for queries and documents, (2) compute field-specific scores using the inner product between query and document vectors, and (3) aggregate the scores using a linear model. Alternatively, we can create *composite* queries and document vectors, where we concatenate field-specified vectors multiplied by field weights. Then, the overall similarity score is computed as the inner product between composite query and document vectors.

Our system supports both computation scenarios. To this end, all inner-product equivalent scorers should inherit from a specific abstract class and implement the functions to generate respective query and document vectors. This abstraction simplifies generation of sparse and sparse-dense query/document vectors, which can be subsequently indexed by `NMSLIB`.

4 Experiments

We carry out experiments with two objectives: (1) measuring effectiveness of implemented ranking models; (2) demonstrating the value of a well-tuned traditional IR system. We use two recently released MS MARCO collections (Craswell et al., 2020; Nguyen et al., 2016) and a community question answering (CQA) collection Yahoo Answers Manner (Surdeanu et al., 2011). Collection statistics is summarized in Table 1.

MS MARCO has a document and a passage re-ranking task where all queries can be answered using a short text snippet. There are three sets of queries in each task. In addition to one large query set with sparse judgments, there are two small evaluation

	MS MARCO		Yahoo Answers
	documents	passages	
	general statistics		
# of documents	3.2M	8.8M	819.6K
# of doc. lemmas	476.7	30.6	20.1
# of query lemmas	3.2	3.5	11.9
	# of queries		
train/fusion	10K	20K	14.3K
train/modeling	357K	788.7K	100K
development	2500	20K	7034
test	2693	3000	3000
TREC 2019	100	100	
TREC 2020	100	100	
	BITEXT tokens		
# of QA pairs	43.9M	4M	572.8K
# of query tokens	2.7	2.8	12.6
# of doc. tokens	4.3	4.2	20
	BITEXT BERT word pieces		
# of QA pairs	50M	9.5M	572.8K
# of query tokens	6.1	2.8	42.3
# of doc. tokens	9.4	4.3	62.7

Table 1: Data set statistics

candidate generator	MS MARCO documents		MS MARCO passages	
	TREC 2019	develop.	TREC 2019	develop.
BM25	0.647	0.443	0.707	0.452
Tuned system	0.693	0.472	0.739	0.480
Gain	7.08%	6.39%	4.57%	6.08%

Table 2: The effect of using a more effective candidate generator (evaluation metric is NDCG@10). BM25 is tuned for MS MARCO passages, but not documents.

sets from the TREC 2019/2020 deep learning track (Craswell et al., 2020). MS MARCO collections query sets were randomly split into training, development (to tune hyper parameters), and test sets.

Yahoo Answers Manner has a large number of paired question-answer pairs. We include it in our experiments, because Model 1 was shown to be effective for CQA data in the past (Jeon et al., 2005; Riezler et al., 2007; Surdeanu et al., 2011; Xue et al., 2008). It was randomly split into the training and evaluation sets.

Document text is processed using Spacy 2.2.3 (Honnibal and Montani, 2017) to extract tokens and lemmas. The frequently occurred tokens and lemmas are filtered out using Indri's list of stopwords (Strohman et al., 2005), which is expanded to include a few contractions such as "n't" and "'ll". Lemmas are indexed using Lucene 7.6. In the case of MS MARCO documents, entries come in the HTML format. We extract HTML `body` and `title` (and store/index them separately).

In additional to traditional tokenizers, we also use the BERT tokenizer from the HuggingFace Transformers library (Wolf et al., 2019). This tokenizer can split a single word into several sub-word pieces (Wu et al., 2016). The stopword list is not applied to BERT tokens.

Training Model 1, which is a translation model, requires a parallel corpus where queries are paired with respective relevant documents. The parallel corpus is also known as a *bitext*. In the case of MS MARCO collections documents are much longer than queries, which makes it impossible to compute translation probabilities using standard alignment tools (Och and Ney, 2003).[18] Hence, for each pair of query q and its relevant document d, we first split d into multiple short chunks d_1, d_2, ... d_n. Then, we replace the pair (q, d) with a set of pairs $\{(q, d_i)\}$.

We evaluate performance of several models and their combinations. Each model name is abbreviated as `X (Y)`, where `X` is a type of the model (see §3.3 for details) and `Y` is a type of the text field. Specifically, we index original tokens, lemmas, as well as BERT tokens extracted from the main document text. For MS MARCO documents, which come in HTML format, we also extract tokens and lemmas from the `title` field.

First, we evaluate performance of the *tuned* `BM25 (lemmas)`. Second, we evaluate fusion models that combine `BM25 (lemmas)` with BM25, proximity, and Model 1 scores (see §3.3) computed for various fields. Note that our fusion models are linear. Third, we evaluate collection-specific combinations of manually-selected models: Except for minor changes these are the fusion models that we used in our TREC 2019 and 2020 submissions.

All models were trained and/or tuned using training and development sets listed in

[18] https://github.com/moses-smt/mgiza/

	MS MARCO documents			MS MARCO passages			Yahoo Answers
	test	TREC 2019	TREC 2020	test	TREC 2019	TREC 2020	test
	MRR	NDCG@10	NDCG@10	MRR	NDCG@10	NDCG@10	NDCG@10
BM25 (lemmas)	0.270	0.544	0.524	0.256	0.522	0.516	0.152
BM25 (lemmas)+BM25 (BERT tokens)	0.283	0.528	0.537	0.270	0.518	0.525	0.159
BM25 (lemmas)+BM25 (tokens)	0.274	0.544	0.523	0.265	0.517	0.521	0.157
BM25 (lemmas)+BM25 (title tokens)	0.294	0.550	0.527				
BM25 (lemmas)+proximity (lemmas)	0.282	0.559	0.524	0.257	0.538	0.523	
BM25 (lemmas)+proximity (tokens)	0.284	0.560	0.531	0.265	0.534	0.524	
BM25 (lemmas)+Model1 (tokens)	0.283	0.548	0.535	0.274	0.522	0.567	0.160
BM25 (lemmas)+Model1 (BERT tokens)	0.284	0.557	0.525	0.271	0.517	0.509	0.175
best combination	0.310	0.565	0.542	0.290	0.558	0.560	

Table 3: Evaluation of various fusion models.

Table 1. For TREC 2019 and 2020 query sets (as well as for Yahoo Answers Manner), the evaluation metric is NDCG@10 (Järvelin and Kekäläinen, 2002), which the main metric in the TREC deep learning track (Craswell et al., 2020). For subsets of MS MARCO collections, we use the mean reciprocal rank (MRR) as suggested by Craswell et al. (2020).

From the experiments in Table 3, we can see that for all large query sets the fusion models outperform `BM25 (lemmas)`. In particular, the best MS MARCO fusion models are 13-15% better than `BM25 (lemmas)`. In the case of Yahoo Answers Manner, combining `BM25 (lemmas)` with Model 1 scores computed for BERT tokens also boost performance by about 15%. For small TREC 2019 and 2020 query sets the gains are marginal. However, our fusion models are still better than `BM25 (lemmas)` by 4-8%.

We further compare the accuracy of the BERT-based re-ranker (Nogueira and Cho, 2019) applied to the output of the tuned traditional IR system with the accuracy of the *same* BERT-based re-ranker applied to the output of Lucene (with a BM25 scorer). The BERT scorer is used to re-rank 150 documents: Further increasing the number of candidates degraded performance on the TREC 2019 test set.

By mistake we used the same BM25 parameters for both passages and documents. As a result, MS MARCO documents candidate generator was suboptimal (passage retrieval did use the properly tuned BM25 scorer). However, we refrained from correcting this error to illustrate how a good fusion model can produce a strong ranker via a combination of suboptimal weak rankers.

Indeed, as we can see from Table 2, there is a substantial 4.5-7% loss in accuracy by re-ranking the output of BM25 compared to re-ranking the output of the well-tuned traditional pipeline. This degradation occurs in all four experiments.

5 Conclusion and Future Work

We present to the NLP community an existing k-NN search library `NMSLIB`, a new retrieval toolkit `FlexNeuART`, as well as their integration capabilities, which enable efficient retrieval of sparse and sparse-dense document representations. `FlexNeuART` implements a variety of effective traditional relevance signals, which we plan to use for a fairer comparison with recent neural retrieval systems based on representing queries and documents via fixed-size dense vectors.

6 Acknowledgements

This work was done primarily while Leonid Boytsov was a PhD student at CMU where he was supported by the NSF grant #1618159. We thank Sean MacAvaney for making CEDR (MacAvaney et al., 2019) publicly available and Igor Brigadir for suggesting to experiment with indexing of BERT word pieces.

References

Sunil Arya and David M Mount. 1993. Approximate nearest neighbor queries in fixed dimensions. In *Proceedings of the fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 271–280.

Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. 2019. ANN-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems*.

Adam L. Berger, Rich Caruana, David Cohn, Dayne Freitag, and Vibhu O. Mittal. 2000. Bridging the lexical chasm: statistical approaches to answer-finding. In *SIGIR 2000: Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, July 24-28, 2000, Athens, Greece*, pages 192–199.

Leonid Boytsov. 2018. *Efficient and Accurate Non-Metric k-NN Search with Applications to Text Matching*. Ph.D. thesis, Carnegie Mellon University.

Leonid Boytsov and Anna Belova. 2011. Evaluating learning-to-rank methods in the web track adhoc task. In *TREC*.

Leonid Boytsov and Bilegsaikhan Naidan. 2013a. Engineering efficient and effective non-metric space library. In *Proceedings of SISAP 2013*, pages 280–293. Springer.

Leonid Boytsov and Bilegsaikhan Naidan. 2013b. Learning to prune in metric and non-metric spaces. In *Advances in Neural Information Processing Systems*, pages 1574–1582.

Leonid Boytsov, David Novak, Yury Malkov, and Eric Nyberg. 2016. Off the beaten path: Let's replace term-based retrieval with k-NN search. In *Proceedings of CIKM 2016*, pages 1099–1108. ACM.

Leonid Boytsov and Eric Nyberg. 2019a. Accurate and fast retrieval for complex non-metric data via neighborhood graphs. In *International Conference on Similarity Search and Applications*, pages 128–142. Springer.

Leonid Boytsov and Eric Nyberg. 2019b. Pruning algorithms for low-dimensional non-metric k-nn search: A case study. In *Similarity Search and Applications*.

Christopher JC Burges. 2010. From RankNet to LambdaRank to LambdaMart: An overview. Microsoft Technical Report MSR-TR-2010-82.

Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, and Ellen M Voorhees. 2020. Overview of the trec 2019 deep learning track. *arXiv preprint arXiv:2003.07820*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Fernando Diaz. 2015. Condensed list relevance models. In *Proceedings of the 2015 International Conference on The Theory of Information Retrieval*, pages 313–316.

Wei Dong, Charikar Moses, and Kai Li. 2011. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th international conference on World wide web*, pages 577–586.

Yixing Fan, Liang Pang, JianPeng Hou, Jiafeng Guo, Yanyan Lan, and Xueqi Cheng. 2017. Matchzoo: A toolkit for deep text matching. *arXiv preprint arXiv:1707.07270*.

Christophe Van Gysel, Maarten De Rijke, and Evangelos Kanoulas. 2018. Neural vector spaces for unsupervised information retrieval. *ACM Transactions on Information Systems (TOIS)*, 36(4):38.

Changwan Hong, Aravind Sukumaran-Rajam, Bortik Bandyopadhyay, Jinsung Kim, Süreyya Emre Kurt, Israt Nisa, Shivani Sabhlok, Ümit V Çatalyürek, Srinivasan Parthasarathy, and P Sadayappan. 2018. Efficient sparse-matrix multi-vector product on gpus. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, pages 66–79.

Matthew Honnibal and Ines Montani. 2017. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear.

Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446.

Jiwoon Jeon, W. Bruce Croft, and Joon Ho Lee. 2005. Finding similar questions in large question and answer archives. In *Proceedings of the 2005 ACM CIKM International Conference on Information and Knowledge Management, Bremen, Germany, October 31 - November 5, 2005*, pages 84–90.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*.

Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*.

Omar Khattab and Matei Zaharia. 2020. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, page 39–48, New York, NY, USA. Association for Computing Machinery.

Saar Kuzi, Mingyang Zhang, Cheng Li, Michael Bendersky, and Marc Najork. 2020. Leveraging semantic and lexical matching to improve the recall of document retrieval systems: A hybrid approach. *arXiv preprint arXiv:2010.01195*.

Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. 2019. Latent retrieval for weakly supervised open domain question answering. *arXiv preprint arXiv:1906.00300*.

Tie-Yan Liu et al. 2009. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225–331.

Sean MacAvaney. 2020. OpenNIR: A complete neural ad-hoc ranking pipeline. In *WSDM 2020*.

Sean MacAvaney, Andrew Yates, Arman Cohan, and Nazli Goharian. 2019. Cedr: Contextualized embeddings for document ranking. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1101–1104.

Craig Macdonald and Nicola Tonellotto. 2020. Declarative experimentation in information retrieval using pyterrier. *arXiv preprint arXiv:2007.14271*.

Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. 2014. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, 45:61–68.

Yury A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*.

Christopher Manning, Prabhakar Raghavan, and Hinrich Schütze. 2010. Introduction to information retrieval. *Natural Language Engineering*, 16(1):100–103.

Donald Metzler and W Bruce Croft. 2005. A markov random field model for term dependencies. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 472–479.

Donald Metzler and W. Bruce Croft. 2007. Linear feature-based models for information retrieval. *Inf. Retr.*, 10(3):257–274.

Bilegsaikhan Naidan, Leonid Boytsov, Yury Malkov, and David Novak. 2015a. Non-metric space library manual. *arXiv preprint arXiv:1508.05470*.

Bilegsaikhan Naidan, Leonid Boytsov, and Eric Nyberg. 2015b. Permutation search methods are efficient, yet faster search is possible. *Proceedings of the VLDB Endowment*, 8(12).

Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. Ms marco: A human generated machine reading comprehension dataset.

Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage re-ranking with bert. *arXiv preprint arXiv:1901.04085*.

Franz Josef Och and Hermann Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51.

Iadh Ounis, Gianni Amati, Vassilis Plachouras, Ben He, Craig Macdonald, and Christina Lioma. 2006. Terrier: A high performance and scalable information retrieval platform. In *Proceedings of the OSIR Workshop*, pages 18–25.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100, 000+ questions for machine comprehension of text. In *Proceedings of EMNLP 2016*, pages 2383–2392.

Stefan Riezler, Alexander Vasserman, Ioannis Tsochantaridis, Vibhu O. Mittal, and Yi Liu. 2007. Statistical machine translation for query expansion in answer retrieval. In *ACL 2007, Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*.

Stephen Robertson. 2004. Understanding inverse document frequency: on theoretical arguments for IDF. *Journal of Documentation*, 60(5):503–520.

Minjoon Seo, Jinhyuk Lee, Tom Kwiatkowski, Ankur P Parikh, Ali Farhadi, and Hannaneh Hajishirzi. 2019. Real-time open-domain question answering with dense-sparse phrase index. *arXiv preprint arXiv:1906.05807*.

Larissa C Shimomura, Rafael Seidi Oyamada, Marcos R Vieira, and Daniel S Kaster. 2020. A survey on graph-based methods for similarity searches in metric spaces. *Information Systems*, page 101507.

Trevor Strohman, Donald Metzler, Howard Turtle, and W Bruce Croft. 2005. Indri: A language-model based search engine for complex queries. http://ciir.cs.umass.edu/pubfiles/ir-407.pdf [Last Checked Apr 2017].

Mihai Surdeanu, Massimiliano Ciaramita, and Hugo Zaragoza. 2011. Learning to rank answers to non-factoid questions from web collections. *Computational Linguistics*, 37(2):351–383.

Eric Sadit Tellez, Edgar Chávez, and Gonzalo Navarro. 2013. Succinct nearest neighbor search. *Inf. Syst.*, 38(7):1019–1030.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Ledell Yu Wu, Adam Fisch, Sumit Chopra, Keith Adams, Antoine Bordes, and Jason Weston. 2018. Starspace: Embed all the things! In *Proceedings of AAAI 2018*.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144.

Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul Bennett, Junaid Ahmed, and Arnold Overwijk. 2020. Approximate nearest neighbor negative contrastive learning for dense text retrieval. *arXiv preprint arXiv:2007.00808*.

Xiaobing Xue, Jiwoon Jeon, and W. Bruce Croft. 2008. 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 2008, Singapore, July 20-24, 2008*, pages 475–482.

Peilin Yang, Hui Fang, and Jimmy Lin. 2018. Anserini: Reproducible ranking baselines using Lucene. *J. Data and Information Quality*, 10(4):16:1–16:20.