CSE 272 Homework 1 Report: A Search Engine for Ohsumed Dataset

Chris Liu

1. Introduction

In this assignment, I implemented a mini search engine with a variety of classic retrieval methods, including boolean retrieval, TF, TF-IDF, BM25, and pseudo relevance feedback (PRF). I also proposed a new approach using sentence embeddings (SE) obtained by pre-trained sentence Transformer (Reimers & Gurevych, 2020) and a variant SE-PRF, which combines sentence embedding and PRF. The proposed sentence embedding method consistently outperforms all classic models in all `trec_eval` metrics, doubling the previously highest mean average precision (MAP).

The report is structured as follows. I start by describing the software design of the search engine, data structures, libraries, and its strengths and weaknesses. I then introduce the proposed pre-trained sentence embedding method and its detailed implementation. After that, I present detailed experiment results and compare the proposed approach with the classic ones. At the end, I discuss what I learned from this assignment. The code and results are included in https://github.com/chrisliu298/ohsumed-search-engine.

2. Software Architecture

Data IO The documents and queries are read in by two data reading functions. These function accept a file path as their arguments, parse the raw data file, and return a generator which generates a list of `Document` and `Query` objects. The detail of these objects is described below.

Document and Query data structures A single document or query is represented by a `Document` or `Query` object, which is constructed using a Python data class. For documents, a data class object contains eight fields (abstract, author, MEDLINE identifier, MeSH terms, publication type, sequential identifier, source, and title), corresponding to the eight fields in both `ohsumed.88-91` and `ohsumed.87`. A document object also has a `full_text` property, which concatenate the document title, MeSH terms, and the abstract. A query object has three fields (query number, query title, and description), and the `full_text` property simply returns the description field.

Tokenizer The tokenizer consists of a pipeline of preprocessing operations, including splitting, lower case, punctuation filter, stop word filter, and stemming. The stemmer uses standard English stemming protocol, and the stop words come from the Natural Language Toolkit (NLTK) (Bird et al., 2009). Lastly, the empty strings are also removed. The result is represented by a list of individual tokens. Note that this particular tokenizer is only used by the boolean retrieval model, as external tools are used for tokenization operations for other models.

Index The `Indexer` class is the core of the search engine. This class combines both index building and searching operations and some auxiliary functions. During the index building stage, the indexer builds an inverted index (token $\rightarrow$ document IDs), a document index (document ID $\rightarrow$ document objects), a TF matrix, an IDF matrix, and a TF-IDF matrix of all the documents. These matrices are built by the `TfidfVectorizer` from `scikit-learn` (Pedregosa et al., 2011). For TF, TF-IDF, BM25, and relevance feedback, their retrieval process largely follow this procedure: 1) convert the query into a vector; 2) compute the similarity scores between the query and all document vectors in the pre-computed matrix; 3) retrieve the top k documents with the highest similarity scores (here $k = 50$) and rank them by the similarity scores; 4) return all the retrieval results as a list, where each element is a dictionary of the document, its score, and its ranking. For sentence embedding retrieval, we illustrate its difference later in Section 3.

Main program By default, the program will reproduce all results described in this report. More specifically, it builds an index, perform retrievals using all implemented retrieval methods, run `trec_eval`, and write the results. A decorator function is used to log the search duration of every retrieval algorithm. The program in `src/` assumes all data files are placed in `data/` directory and all precomputed sentence embeddings are in `embeddings/`. All results will be written to `results/`, including both the log files (for evaluation) and the evaluation results generated by `trec_eval`, as the evaluation command is executed by the main program. After all evaluations, an additional function extracts the "all" performance at the end of a `trec_eval` file and store in a `.tsv` format.

3. Method

3.1. Sentence embedding model

The backbone model for sentence embedding is a pre-trained MPNet (Song et al., 2020), a Transformer-based language model trained using masked and permuted pre-training. MPNet utilizes incorporates the masked language modeling objective from BERT (Devlin et al., 2018) but, in addition, utilizes the dependencies among the predicted tokens by the permuted language modeling objective introduced in XLNet (Yang et al., 2019). Built upon MPNet, the sentence embedding model is fine-tuned by the self-supervsied contrastive objective on a collection of datasets of more than one billion sentences. Using MPNet as siamese and triplet network structures, it is trained to predict a paired sentence from a set of randomly sampled sentences (Reimers & Gurevych, 2020). Using this model, each document or query is represented by a vector of size 768, which is the sentence embedding dimension of MPNet.

To compute the embeddings, I simply load the model, document text, and query text. The output embeddings will be a matrix of shape (N, 768), where N is the number of documents or queries. The cosine similarity scores are also represented by a matrix of shape (Q, D), where Q and D correspond to the number of queries and documents, respectively.

3.2. Sentence embedding search

Because of the difference between sentence embedding and traditional vector space models, the indexer also reads in pre-computed embeddings of all documents and queries to prevent online encoding (which requires a GPU and slows down the retrieval process). The similarity matrix (from all queries to all documents) is also pre-computed. Therefore, when doing retrieval using sentence embedding, it only requires the similarity matrix to find the indices of the most similar documents and return the scores, rankings, and actual documents based on those indices. However, it could also be feasible to do online query search, as the query encoding and cosine similarity computations are relatively cheap. It takes approximately 24 minutes with an NVIDIA Tesla V100 GPU to encode all document and queries in the `ohsumed.88-91` dataset and compute their cosine similarities. I also combine sentence embedding with PRF (SE-PRF), where the pseudo relevant document are top k vectors that are closest to the query vector. The Rocchio algorithm is implemented to compute a new qeury vector. Unlike the regular sentence embedding, where only the similarity scores are needed, SE-PRF needs the document and query embedding matrices, because PRF requires vector-level operations.

Methods	MAP	P_5	Time (seconds)
Boolean	0.0230	0.1429	0.0067
BM25	0.0882	0.2857	7.3261
TF	0.0758	0.3079	9.0166
TF-IDF	0.1227	0.3556	9.0791
PRF	0.1271	0.3524	17.9212
SE*	0.2532	0.6032	1.4519
SE-PRF*	**0.2698**	**0.6159**	192.1118

Table 1. Mean average precision, precision of the 5 first documents, and the time taken to run all 63 queries. * indicates proposed methods. SE-PRF consistently achieve the best performance, and both the SE and SE-PRF double the previously highest performance. SE can be considered the best method among all due to its strong performance and speed. SE is fast because it leverages the pre-computed similarity matrix so that it no longer need to perform any computation other than sorting. While SE-PRF is the best in terms of performance, it falls shorts on speed due to the expensive operations on vector similarity and its interactions with both the document matrix and the query matrix.

4. Results

All evaluation metrics of the seven implemented methods from the `trec_eval` results are summarized in Figure 1. The mean average precision (MAP), precision of the five first documents (P_5), and the querying time are reported in Table 1. The proposed SE and SE-PRF approaches are the best among all implemented methods due to the superiority in their sentence representations, because the MPNet is fine-tuned to exactly discriminate sentence pairs and on a wide range of datasets to obtain generality. However, general sentence embedding methods are only possible because of the pre-trained models. Therefore, other simple non-neural approaches, like PRF, should be the go-to choice when it is infeasible to use pre-trained models. It is surprising that TF and TF-IDF outperforms BM25 due to their algorithmic similarities. While being the fastest method, due to the nature of exact matching, boolean retrieval model fails to compete with other approaches.

5. Discussions

What I learned I list below things I learned during the completion of this assignment:

1. Implementing all retrieval algorithms strengthened my understanding of the course materials.
2. While being only limited on one dataset, this assignment gave me a sense of what the recipe of a real search engine looks like.
3. I realized that a general sentence embedding model can be effective on in-domain tasks (i.e., in this case, the medical abstract retrieval).

4. I learned the trick from the proposed SE approach, where the query process can take far less time when required information can be pre-computed.

References

Bird, S., Klein, E., and Loper, E. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Reimers, N. and Gurevych, I. Making monolingual sentence embeddings multilingual using knowledge distillation. *arXiv preprint arXiv:2004.09813*, 2020.

Song, K., Tan, X., Qin, T., Lu, J., and Liu, T.-Y. Mpnet: Masked and permuted pre-training for language understanding. *Advances in Neural Information Processing Systems*, 33:16857–16867, 2020.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., and Le, Q. V. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32, 2019.

Appendix

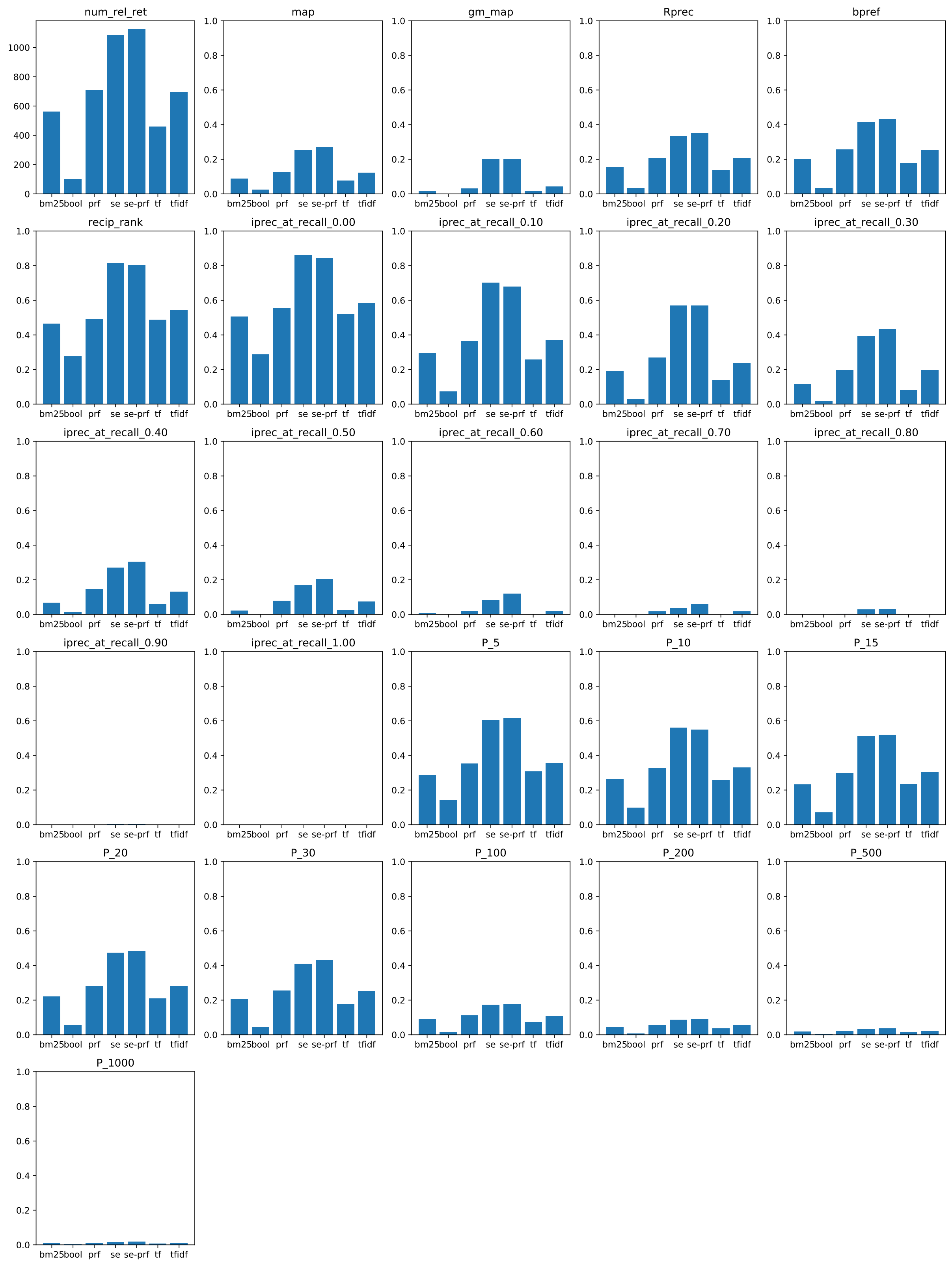

Figure 1. Evaluation results of BM25, boolean retrieval, pseudo relevance feedback, sentence embedding (SE), sentence embedding with pseudo relevance feedback (SE-PRF), TF, and TF-IDF on `ohsumed.88-91`. SE and SE-PRF consistently achieve the best results among all retrieval methods on all metrics.