Neural Information Retrieval 2022: Final Project

André Oskar Andersen (wpr684)
wpr684@alumni.ku.dk

	Document length	Query length
Min	6	6
Max	1,737,815	35
Mean	2,868.43	20.37

Table 1: Minimum, maximum and average amount of characters in the documents and queries.

1 Introduction

The following paper is my final project in the course *Neural Information Retrieval* taught at University of Copenhagen. The aim of the project is to implement and evaluate search algorithms for news articles. This includes (1) implementing and evaluating various indices, (2) implementing and evaluating various ranking models, and (3) present, analyse and discuss the results. This paper contains no code, but describes the implementation details and covers the three points from above.

2 The Dataset

For the data we will be using the NIR2022 dataset (Retrieval). The dataset is a collection of 528.155 newswire documents and 200 queries. Each document consists of various fields, such as title and abstract. The only preprocessing of the dataset we do is to remove all empty documents, since it is impossible to predict correctly on these document and they just take up space. One could have decided to do other preprocessing, such as tokenization or stemming, however, we have decided not to do so, since the information retrieval framework that we will be using takes care of this. By removing the empty documents we end up with a total of 528.030 documents. Table 1 illustrates the final distribution of the amount of characters in the documents and queries.

The relevance between a document and query is marked as either being *irrelevant*, *relevant* or *highly relevant*. The dataset consists of 247.569 relevance assessments. Of these relevance assessments 883 (0.35%) are marked as "irrelevant", 12.713 (5.14%) are marked as "relevant" and the last 233.973 (94.51%) are marked as "highly relevant".

Name	Stemmer	Stopword Rem.	Blocks
S1	True	True	True
S2	True	True	False
S3	True	False	True
S4	True	False	False
S5	False	True	True
S6	False	True	False
S7	False	False	True
S8	False	False	False

Table 2: The setups of the implemented indices. "Stopwrod Rem." refers to whether or not to remove stopwords. "Blocks" refers to whether or not to create the index with blocks, such that positional information is stored.

3 Indexing

To quickly find relevant documents for a search topic we have indexed the dataset. The indexing has been implemented in Python using PyTerrier (PyTerrier). To find the optimal setup of the index we have implemented the indices illustrated in Table 2. These setups cover all of the setups which are possible by tuning whether or not to use the stemmer, stopword removal and storing positional information. This approach has a rather high development time, however, since the setup of the index can have a major impact on the results we found it worthwhile to spend some extra time on finding the optimal index.

Table 3 presents the results of various measurements of efficiency for the various indices. In the table we can see, that indices `S2`, `S4`, `S6`, and

Name	Index size	Time to make	Mean Response Time
S1	1.9 GB	647 sec.	17.610 sec.
S2	1.4 GB	720 sec.	16.257 sec.
S3	2.4 GB	739 sec.	20.782 sec.
S4	1.5 GB	759 sec.	17.482 sec.
S5	2 GB	826 sec.	16.103 sec.
S6	1.5 GB	728 sec.	15.545 sec.
S7	2.5 GB	968 sec.	40.801 sec.
S8	1.6 GB	907 sec.	34.725 sec.

Table 3: Efficiency evaluation of the index setups from Table 2. "Mean Response Time" is the mean response time it took for a BM25 to receive the results of the queries from the training data, based on 5 runs.

`S8` are the indices which uses the least memory, which is most likely due to the fact, that they not store positional information. We can also see, that the indices which have applied a stemmer to the data generally takes up less space than their non-stemmer counterpart, which comes from the fact, that stemming reduces the vocabulary space and thus reduces the memory usage. We can also see, that the indices which stores positional information actually performs worse, than their counterpart which does not store positional information, which is opposite of what one would assume would be the result. This is probably due to the fact, that the used framework (PyTerrier) is so optimized, such that our data is too small for the advantages of the positional information having an effect. Further, for the indices with positional information probably looks for documents that contains the input query very closely, whereas the indices without positional information probably just looks for documents that have content that is related to the input query, which makes them faster than the indices with positional information. Lastly we can see, that both applying a stemmer and a stopword-remover lowers the response time, which is probably due to the fact, that each term in the query matches with fewer terms in the documents (due to the stemmer) and there are fewer terms in the documents to compare the queries with (due to the stopword remover).

Generally, when an index applies a stemmer, it becomes smaller than its counterpart which does apply a stemmer, which also makes the querying of the index faster. The rating of which index is the most efficient depends on ones requirements. If one does not care about the size of the index and only cares about the speed, then `S6` is the most efficient index, since it is the fastest index in the tests. On the other hand, if one needs the size of the index to be small, then `S2` is the most efficient index, since it is the smallest index. Generally, `S2` and `S6` seems to be the most efficient indices, since these both offer a short response time, as well having a small index size.

4 Ranking Models

To rank the retrieved documents from a index, we implemented and tested two different approaches: (1) Using probabilistic models and (2) using neural information retrieval models. The following section covers the implementation and tuning of these models.

4.1 Probabilistic Models

For the probabilistic ranking models we implement four models:

1. The BM25
2. A language model with Dirichlet Smoothing (LMD)
3. A pipeline which uses pseudo-relevance feedback, which consists of ranking by BM25, then apply query rewriting with RM3, which is then followed by re-ranking the documents with BM25. In PyTerrier we can write this as `BM25 » RM3 » BM25`. This model will be abbreviated as the *RM3-pipeline*
4. A bigger pipeline consisting of first ranking by TF-IDF, then apply query rewriting with RM3, which is then followed by union-joining the rankings of LMD, BM25 and TF-IDF. Lastly, this set-union of documents is passed to a BM25-model, which is used to perform a final round of ranking. In PyTerrier we can

write this as `TF-IDF » RM3 » (LMD | BM25 | TF-IDF) » BM25`. This model will be abbreviated as the *Union-pipeline*.

Our four models are tuned on all of the indices from Table 2 by making use of a 3-fold cross-validation by using a grid-search method implemented in PyTerrier. PyTerrier's implementation of BM25 has three parameters: b, k_1, and k_3, and the Diriclet language model only has μ to tune. For the tuning we let $b \in \{0, 0.5, 1\}$, $k_1, k_2 \in \{0.1, 0.5, 1\}$, and $\mu \in \{l_{avg}, 200, 400, 600, 800, 1,000\}$, where l_{avg} is the average length of the documents stored in a given index.

One could have decided a bigger range of values for the parameters to tune the models on, but due to the long tuning time of these models, and the fact that they will be tuned on all of the indices, we unfortunately had to limit the amount of values for the parameters.

4.2 Neural Information Retrieval Methods

For the neural information retrieval methods we implement three methods:

1. Query expansion using word embeddings: A model which first expands the input query using 50-dimensional GloVe word vectors (Pennington et al., 2014), which is then passed to a BM25-model to rank the documents.

2. Contextualized word embeddings: A model which first ranks the documents by using a BM25-model. We then pick the top K ranked documents returned by the BM25-model and pass them to BERT (MacAvaney et al., 2019) which then re-ranks these K documents.

3. For the last model we would initially have implemented a model which would use a manual term expasion techniques, but, this is unfortunately not possible in PyTerrier. Instead, we decided to use a model which uses the Mono T5 ranking approach (Pradeep et al., 2021a) to rerank all documents retrieved by a BM25-model.

In the first method the GloVe-pipeline has already been pretrained by Gensim, so we will just be using that (Gensim contains various variations of GloVe, we will be using the one called *glove-wiki-gigaword-50*) (Gensim). However, we still have to decide how much the query should be expanded. Since our queries do not contain many words, we will be testing the effects of expanding the queries with $K \in \{1, 2, 3, 4, 5\}$ terms.

For the second method will be training the BERT-model. The fitting is done by using early-stopping, which stops once the validation-loss of the whole pipeline has not increased for 20 epochs. To ensure an unbiased validation-evaluation and testing-evaluation of the model we will split the data into three non-overlapping parts, such that 20% of the data is used for validation (which is only used for the early-stopping), 20% of the data is used for testing (which will be covered in section 5.1.2), and the last 60% of the data is used for training the model. We use the common pick of letting $K = 100$

For the third method we will be using an implementation of Mono T5 which has already been trained on the MS MARCO passage dataset (MSM).

For all methods we will not be tuning the various parameters for the BM25 due to the high parameter search-time, but instead use the optimal setting for the BM25 found in section 4.1.

5 Evaluation

After optimizing our models we can evaluate them. This will be done over two phases: first on the queries we already have and secondly on some unseen queries. The following section covers the results of the evaluation, as well as some analysis of the results.

5.1 Validation

5.1.1 Probabilistic Models

Table 4 illustrates the resulting evaluation of the probabilistic models described in section 4.1. Looking at the table we can clearly see, that just using a BM25 or LMD in combination with the last half of the indices (indices `S4` to `S8`) generally yield the worst results. On the other hand we can see, that the Union-pipeline in combination with the first half of the indices (indices `S1` to `S4`) yield the best results. The reason for this is most likely due to the fact, that the first half of the indices have applied a stemmer at indexing time as well as applies one at query time, making the matching of words between documents and

Index	Method	Mean response time	MAP	$nDCG_5$	$nDCG_{10}$	$nDCG_{20}$
S1	BM25	0.186 sec.	0.257	0.459	0.441	0.420
S1	LMD	0.165 sec.	0.256	0.451	0.436	0.419
S1	RM3-pipeline	0.374 sec.	0.291	0.447	0.434	0.424
S1	Union-pipeline	0.748 sec.	0.301	0.462	0.451	0.436
S2	BM25	0.152 sec.	0.257	0.459	0.441	0.420
S2	LMD	0.159 sec.	0.256	0.451	0.436	0.419
S2	RM3-pipeline	0.325 sec.	0.291	0.447	0.434	0.424
S2	Union-pipeline	0.715 sec.	0.301	0.462	0.451	0.436
S3	BM25	0.300 sec.	0.253	0.446	0.429	0.411
S3	LMD	0.179 sec.	0.257	0.45	0.432	0.419
S3	RM3-pipeline	0.395 sec.	0.287	0.424	0.422	0.412
S3	Union-pipeline	0.876 sec.	0.296	0.451	0.438	0.422
S4	BM25	0.341 sec.	0.253	0.446	0.429	0.411
S4	LMD	0.162 sec.	0.257	0.45	0.432	0.419
S4	RM3-pipeline	0.337 sec.	0.267	0.424	0.422	0.412
S4	Union-pipeline	0.744 sec.	0.296	0.451	0.438	0.422
S5	BM25	0.239 sec.	0.228	0.432	0.415	0.393
S5	LMD	0.151 sec.	0.229	0.437	0.421	0.396
S5	RM3-pipeline	0.357 sec.	0.278	0.447	0.44	0.426
S5	Union-pipeline	0.736 sec.	0.276	0.436	0.427	0.414
S6	BM25	0.234 sec.	0.228	0.432	0.415	0.393
S6	LMD	0.146 sec.	0.229	0.437	0.421	0.396
S6	RM3-pipeline	0.316 sec.	0.278	0.447	0.44	0.426
S6	Union-pipeline	0.684 sec.	0.276	0.436	0.427	0.414
S7	BM25	0.331 sec.	0.225	0.427	0.409	0.388
S7	LMD	0.345 sec.	0.228	0.436	0.419	0.392
S7	RM3-pipeline	0.674 sec.	0.275	0.446	0.433	0.422
S7	Union-pipeline	0.129 sec.	0.271	0.426	0.423	0.411
S8	BM25	0.290 sec.	0.225	0.427	0.409	0.388
S8	LMD	0.314 sec.	0.228	0.436	0.419	0.392
S8	RM3-pipeline	0.275 sec.	0.275	0.446	0.433	0.422
S8	Union-pipeline	0.994 sec.	0.271	0.426	0.423	0.411

Table 4: Evaluation of the various combinations of indices and probabilistic ranking methods.

queries easier, whereas the last half of the indices have not applied such a stemmer. Further, BM25 and LMD are very simple models that do not capture a lot of information. On the other hand, the Union-pipeline is more complex - by fusing the results of multiple methods (TF-IDF, BM25, and LMD) we essentially combine multiple weak methods into one strong method, since each of the weak methods might capture different information. Thus, by combining the methods we get a more complex model, which can capture more information from various perspectives, which essentially should make better predictions.

We can further see, that whether or not to apply a stopword remover only has a minor effect on the effectiveness of the models, and that whether or not to save the positional information does not have any impact of the effectiveness.

Lastly, we can see, that generally the choice of the index and the choice of model both have a big impact on the result. The more complex a model is the higher is the response time of the model. This makes sense, since a more complex model requires more steps than the simpler models.

5.1.2 Neural Information Retrieval Methods

Table 5 shows the results of the evaluation of the three neural information retrieval methods

Index	Method	Mean response time	MAP	nDCG$_5$	nDCG$_{10}$	nDCG$_{20}$
S1	GloVe	0.117 sec.	0.167	0.333	0.307	0.295
S1	BERT	1.368 sec.	0.139	0.383	0.337	0.323
S1	Mono T5	12.74 sec.	0.210	0.475	0.435	0.400
S2	GloVe	0.101 sec.	0.167	0.333	0.307	0.295
S2	BERT	1.285 sec.	0.139	0.384	0.337	0.323
S2	Mono T5	12.74 sec.	0.210	0.475	0.435	0.400
S3	GloVe	0.119 sec.	0.165	0.322	0.303	0.290
S3	BERT	1.331 sec.	0.147	0.391	0.367	0.338
S3	Mono T5	12.72 sec.	0.210	0.474	0.434	0.400
S4	GloVe	0.102 sec.	0.165	0.322	0.303	0.290
S4	BERT	1.315 sec.	0.148	0.449	0.403	0.356
S4	Mono T5	12.82 sec.	0.210	0.474	0.434	0.400
S5	GloVe	0.106 sec.	0.150	0.317	0.293	0.278
S5	BERT	1.311 sec.	0.127	0.405	0.358	0.322
S5	Mono T5	12.22 sec.	0.200	0.430	0.430	0.393
S6	GloVe	0.093 sec.	0.150	0.317	0.293	0.278
S6	BERT	1.282 sec.	0.132	0.38	0.354	0.314
S6	Mono T5	12.17 sec.	0.200	0.430	0.430	0.393
S7	GloVe	0.292 sec.	0.149	0.312	0.292	0.274
S7	BERT	1.315 sec.	0.127	0.391	0.361	0.330
S7	Mono T5	12.28 sec.	0.200	0.471	0.434	0.394
S8	GloVe	0.244 sec.	0.149	0.312	0.292	0.274
S8	BERT	1.306 sec.	0.131	0.363	0.341	0.321
S8	Mono T5	12.07 sec.	0.200	0.471	0.434	0.394

Table 5: Evaluation of the various combinations of indices and neural information retrieval methods. *GloVe* shows the evaluation of the GloVe-pipeline for $K = 1$.

described in Section 4.2, where we have chosen *GloVe* to be the pipeline for the GloVe-pipeline, where the queries are expanded with $K = 1$ words, since this setting seems to yield the best results. In fact, the results only worsens as K increases (see Figure 1 in the appendix for the results of the evaluation of tuning K). This trend is due to *semantic drift*, that is, by expanding the query we shift the meaning of the query, even if the newly introduced terms are related the to original query.

Like in the case for the probabilistic models we can also see in this case in Table 5, that storing the positional information has no impact on the accuracy. Likewise, we can also see in this case, that using the first half of the indices generally yield the most accuracte results, whereas the last half of the indices yield the least accurate results.

For all indices the BM25 with Mono T5 yields the best results, whereas the BM25 with BERT yields the worst results. Zhang *et al.* describes, how the finetuning of BERT is unstable, especially in cases where a large model and only a small dataset is used (Zgang et al., 2021). This can explain our observation of BM25 with BERT yielding bad results, since we have, that our BERT-model is rather big and the training dataset only consists of 120 queries. Likewise, Mono T5 is also a rather big model, however, since it has been trained on a larger dataset, the fitting of the parameters is not a problem, which is why it yields great results.

The GloVe-pipeline tends to perform worse than the Mono T5 model, but better than the BERT model, even though it is very similar to just using a BM25-model, which we know from Table 4 should perform better than Mono T5. This is probably due to the issue with semantic drift, which can also happen even when the queries are only expanded with one extra term. Since the queries are rather short (in terms of the amount of terms), only adding 1 term can really shift the semantic of the

Model	MAP	nDCG_5	nDCG_{10}	nDCG_{20}
RM3-pipeline	0.2468	0.4391	0.4096	0.3815
Union-pipeline	0.2400	0.3605	0.3589	0.3469
LMD	0.2316	0.4520	0.4274	0.4026
BM25	0.2308	0.4527	0.4274	0.3946
Mono T5	0.1982	0.4658	0.4414	0.3953
GloVe	0.1837	0.3967	0.3697	0.3467
BERT	0.1599	0.3435	0.3483	0.3639

Table 6: Results of testing the best index-model-combination of each model on 50 unseen queries.

query. Further, due to the low amount of terms in each query it can be difficult to understand the ambiguity of the terms, making it easy to add terms with the incorrect semantic.

5.2 Testing

To measure the effectiveness of the developed models in a more realistic setting, we have tested some the developed methods/models on 50 unseen queries. One could have tested every single developed combination of the various indices and models, however, we found this to be rather redundant, since there does not seem to be a major difference between the effectiveness of some indices and some indices seems to always outperm other indices. Thus, we have decided to test each of the 7 methods that have been developed using index `S1`, since this index seems to be yielding the best results. By doing so we get the results illustrated in Table 6.

Looking at Table 6 we can see, that the neural based models perform the worst, whereas the probabilistic methods performs the best. Like the case for the validation data, this is probably due to the high amount of tuneable parameters for the neural based models, making it difficult to fit the models with such a small training dataset.

Comparing Table 6 with Table 4 and 5 we can further see, that the probabilistic methods tends to perform worse on the testing set than on the validation set, whereas the neural based models tend to perform better on the testing set (except for Mono T5). The fact that the probabilistic models performs worse on the testing data is most likely due to a mistake we made when training and validating the probabilistic models. For these models we did not split the dataset into three batches - one for tuning, one for validation, and one for testing - but instead just tuned and tested the models on overlapping data, making the evaluation biased, as explained by Adrian Tam (Tam). On the other hand, for the BERT-pipeline we did split the dataset into non-overlapping batches. Thus, the reason behind the increase of performance for this model, as well as for GloVe, is probably due to some differences in the validation and testing data, for instance the testing data being less ambigious. This can probably also explain the decrease of the performance of the Mono T5 method, which, unlike the BERT-pipeline, was already pretrained on some other dataset.

5.3 Future Work

If we were to work further with this project, it would be ideal to first off perform the same tuning on a bigger dataset, such as the MS MARCO dataset (MSM), to see how this would affect the results. In combination with this one could use transfer learning to fine-adjust the weights of a pretrained model, such as Mono T5, to see whether this would yield a better result. Further, one could combine the Mono T5 with the Duo T5 as it was initially intended to be used (Pradeep et al., 2021b). Lastly, one could do a wider hyperparameter optimization, such as grid search, for the parameters of the probabilistic method to see if this would yield a better result.

6 Conclusion

Through this paper we have successfully designed, implemented and evaluated various indices and ranking models. We have analysed the reasons behind the results and presented future work one could perform if they were to continue on with this project.

References

Ms marco. https://microsoft.github.io/msmarco/. Accessed: 5.06.22.

Gensim. Word2vec embeddings. https://radimrehurek.com/gensim/models/word2vec.html?highlight=glove%20wiki%20gigaword. Accessed: 02.06.22.

Sean MacAvaney, Andrew Yates, Arman Cohan, and Nazli Goharian. 2019. Cedr: Contextualized embeddings for document ranking. Technical report, IRLab, Georgetown University and Max Planck Institute for Informatics and Allen Institute for Artificial Intelligence.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. Technical report, Computer Science Department, Stanford University.

Ronak Pradeep, Rodrigo Nogueira, and Jimmy Lin. 2021a. The expando-mono-duo design pattern for text ranking with pretrained sequence-to-sequence models. Technical report, David R. Cheriton School of Computer Science, University of Waterloo.

Ronak Pradeep, Rodrigo Nogueira, and Jimmy Lin. 2021b. The expando-mono-duo design pattern for text ranking with pretrained sequence-to-sequence models. Technical report, University of Waterloo.

PyTerrier. Welcome to pyterrier's documentation. https://pyterrier.readthedocs.io/en/latest/. Accessed: 5.05.22.

Neural Information Retrieval. Neural information retrievel dataset. https://absalon.instructure.com/courses/56884/files/folder/project. Accessed: 5.05.22.

Adrian Tam. Training-validation-test split and cross-validation done right. https://machinelearningmastery.com/training-validation-test-split-and-cross-validation-done-right/. Accessed: 13.06.22.

Tianyi Zgang, Felix Wu, Arzoo Katiyar, Kilian Q. Weinberger, and Yoav Artzi. 2021. Revisiting few-sample bert fine-tuning. Technical report, ASAPP Inc. and Stanford University and Penn State University and Cornell University.

A Appendix

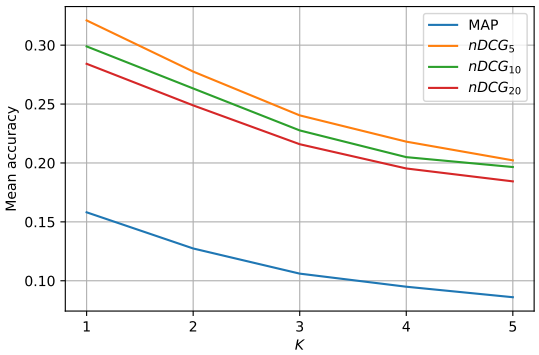

Figure 1: The effect on the mean accuracy of the K-parameter of the GloVe-method.