Neural Information Retrieval 2022: Project Report

Yuqin Zhou
University of Copenhagen
qvk729@alumni.ku.dk

Abstract

This paper uses the NIR2022 dataset to evaluate various search algorithms: probabilistic and language models, divergence from randomness based models and neural retrieval models. In addition, query expansion techniques are used to improve effectiveness. The findings highlight the merit of embeddings, especially the direct use of them to compute similarities between queries and documents. The Cross-Encoder model using segments secured 1th position in the NIR course.

1 Introduction

This project was completed under the guidance of the course, Neural Information Retrieval (NIR), in University of Copenhagen. The goal of this project was to design appropriate strategies for indexing and to explore text retrieval models.

In accordance with the course instructions, the paper is structured as follows: Section 2 describes the NIR2022 dataset and the pre-processing steps. Here, I will explain different indexing approaches and choose an index for the following experiments. In Section 3, the experimental results of various models on the training set are discussed. Section n4 reports the performance of models on unseen queries. Lastly, the limitations of this project and possible directions for improvement are described in Section 5.

2 Data pre-processing and indexing

2.1 The NIR2022 dataset

Before proposing any model, let us briefly look at the NIR dataset provided by the NIR course. As shown in Table 1, the dataset has 250 topics: 200 of them form the training set used in Section 3 and 50 of them are unseen topics which were evaluated in Section 4.

Then, I quickly checked some of documents and topics. These documents consist of newswire articles from four organizations. Although the fields in the documents are stored separately, the four sources have different name conventions for storing fields information, suggesting difficulties to extract the correct corresponding fields like `Title`. This finding discouraged me to use fields information.

Attributes	NIR2022 dataset
Number of Documents	528155
Number of Topics	250
Number of Tokens	269905822
Number of Terms	840517

Table 1: Basic information about NIR2022 dataset.

Certain queries can affect the performance of IR systems. For instance, a query doesn't have judged relevant document label (`qid`: 672). Further, some queries were improperly pre-processed. For instance, the word `U.S.` has been tokenized into two separate characters `u s` in five queries (`qid`: 628, 651, 670, 688, 700). The possessive form `'s` has been tokenized as a single character `s` in two queries (`qid`: 398, 406). Therefore, This observation encouraged me to consider adding information to each query through tools such as query expansion.

2.2 Data pre-preprocessing

The next step was to pre-process the dataset. To reflect the influence of pre-processing tools, I used BM25 with the effective metric Mean Average Precision (MAP) and the efficient metric Mean Response Time (MRT), as shown in Table 2.

From the perspective of effectiveness, the Snowball stemmer outperformed than other stemmers. This result is reasonable as the Snowball is an updated version of the Porter. For instance, the Porter stems *fairly* to *fairli* but it can be properly stemmed to *fair* by the Snowball. However, the Snowball still generated improper word forms, so I tried to perform lemmtization to avoid this problem. Unexpectedly, lemmtization here performed worse

Tools	MAP	MRT
Snowball + Stop	0.2368	19.3702
Snowball stemmer	0.2305	19.5456
Porter stemmer	0.2294	19.4266
Weak Porter stemmer	0.2230	18.6875
Unstemmed	0.2041	17.8731
Lemmatization	0.1931	17.1245

Table 2: Applied BM25 on the NIR2022 dataset that pre-processed differently. Stop means Stop Words Removal

than stemming. This might due to several reasons such as the used lemmatizer. Here, I used the NLTK tools: `TreebankWordTokenizer` `WordNetLemmatizer`. These tools failed to lemmatize many words in the documents such as *tried* and *intended*[1]. As a result, its MAP did not reach the high score I expected.

From the viewpoint of efficiency, Table 2 shows that stemming and stop word removal have little influence on MRT, i.e. the average number of milliseconds to retrieve a query. This is because stemming and stop word removal have been done when indexing documents prior to queries retrieval. Thus, these tools have little influence to MRT after created an indexer (See further discussion of the impact of pre-processing tools on positing lists in Section 2.3).

Considering the tools' performance and their run-time, I used the Snowball stemmer and stop words removal to pre-process the dataset.

2.3 Indexing

To indexing documents, I used PyTerrier to build three indexers and to compare them in terms of size, building time and MRT, as shown in Table 3.

Index	Size	Building	MRT
Pre-processed	379M	5m 48.8s	17.53
Pre-processed$\star$	534M	4m 42.6s	17.77
Pre-processed$*$	907M	6m 43.1s	21.16

Table 3: Efficiency statistics of different indexers. Pre-processed means using the Snowball stemmer and stop words removal. $\star$ means this index was created without prepossessing. $*$ means this index was pre-processed and recorded the term position within the collection.

To explain this table, I need to firstly introduce PyTerrier's index structure. This structure was instantiated by Macdonald and Tonellotto (2020) on two existing Information Retrieval (IR) platforms (Anserini and Terrier). Their official documentation states that a Terrier's index folder contains multiple files including Lexicon, Inverted Index, Document Index, Direct Index, etc. More importantly, Terrier can highly compress an index structure into this folder and reduce its original index size by a factor of 7 (Ounis et al., 2006). Since index optimization will change the order of posting lists, this may also explain why stemming and stop words removal hardly influence MRT when retrieving queries. Considering the other two metrics, the pre-processing step cost more building time but significantly reduced the index size. Based on this observation, I expect that index pruning can also reduce an index size and increase building time.

The third indexer consumed the largest computational resources, since it recorded term location within the collection. This information is useful if we consider phrasal and proximity search. As I did not consider these features in this project, and as the pre-processing step improved the performance of the BM25 shown in Section 2.2, I have chosen the first indexer for the following experiments.

Lastly, I used this indexer to explore the dataset as shown in Table 4 [2]. These statistics can help us choose an efficient way of indexing. For instance, we can pick a search policy such as sequential or binary search based on minimal and maximal positing length. Fortunately, with the help of PyTerrier, I didn't need to worry about such problems in this project.

Length	Min.	Max.	Avg.
Documents	1	184423	302.1
Topics	1	4	2.71
Doc. per topic	207	2992	1244.1
Term frequency	1	1041512	215.8

Table 4: Relevant statistics of documents and topics.

3 Models: training set

After parsed and indexed the dataset by PyTerrier, I compared various models to rank the documents, as shown in Table 5. The evaluation metrics, provided by `pytrec_eval` library based on the

[1]The result can be improved if we provide the Part-of-speech (POS) tag as the second argument to the lemmatizer but this work requires hours to process.

[2]Here, I calculate the lengths by number of tokens instead of characters, and treats the word `u s` as a single word and exclude the word `s` in the topics

`trec_eval` tool, are MAP, nDCG5, nDCG10, nDCG20 and MRT.

Before focus on these models, I need to shortly explain how I organized the experiment part. First, due to limited computational power, I ran various models with default parameters on the training set. For some models, Then, I used query expansion and `5-fold cross-validation` with a grid search algorithm to optimize some models. Their optimal parameter settings were selected based on MAP and their results were the average of `5-fold cross-validation` results[3]. The reasons for optimizing these models are given in the following sections.

3.1 Probabilistic models

A common option to start with is to implement two probabilistic models: TF-IDF and Okapi BM25. These models estimate what is the probability that a user will judge a document relevant to a query (Büttcher et al., 2016). The result shows that TF-IDF is slightly better than Okapi BM25 in terms of all metrics.

Since the result did not meet my expectation, I checked the official documentation of PyTerrier and the details of two models implementation. The documentation states that TF-IDF here refers to BM11, i.e. TF is given by Robertson's TF and IDF is given by the standard Sparck Jones' IDF (SPARCK JONES, 1972). However, contradicting their official documentation, TF-IDF has two parameters in the code, namely $k_1 = 1.2$ and $b = 0.75$. Compared to TF-IDF, Okapi BM25 has the same default settings for parameters b and k_1, except for an additional parameter $k_3 = 8$. Intuitively, b controls the importance of document length normalization, k_1 adjusts the balance between term frequency and IDF, k_3 is an additional factor applied to term frequencies within queries. Since most of queries in our dataset are short and many terms have single occurrences, so k_3 have little impact on the results. Without tuning parameters, these findings might explain why the BM25 yielded a similar result to the TF-IDF. However, since the BM25 has one more parameter than the TF-IDF, I will combine the BM25 with query expansion in Section 3.4.

[3]Since PyTerrier can only calculate one metric when performing k-fold cross-validation, the tuned models are then evaluated on the entire training set to obtain other metrics

3.2 Language model

As shown in Table 5, I tested the language model with Dirichlet Smoothing (`DirichletLM`). This model tries to answer what is the probability that a term would be entered by a user in order to retrieve a document (Büttcher et al., 2016).

`Dirichlet LM` performed worse than the TF-IDF or the BM25 in terms of all metrics. This may be due to insufficient pre-processing. The model adds $\mu = 2500$ extra tokens to each document based on the distribution of words in the collection. This smoothing requires careful preprocessing of the dataset to reveal the true lexical distribution. However, since I only used a simple stemmer and the removal of stop words as a preprocessing step, this could not guarantee a good result for this language model. Despite its poor performance, I tuned this language model in Section 3.4 due to the project requirement.

3.3 Divergence From Randomness (DFR) based models

To explore more models, an idea intrigues me: the more information word t carries in a document d, the greater the divergence between the frequency of this term t within the document d from within the collection. To develop this idea, the DFR paradigm generalizes Harter's 2-Poisson indexing-model (Harter, 1974), proposing that the term weights are inversely proportional to the probability of term frequencies in document d obtained from the randomness model M. Also, this framework applies the two normalisation based term frequencies and document length.[4]

Further, due to DFR based models are non-parametric, I efficiently evaluated some of them without tuning any parameter,as shown in Appendix A.1. The DPH is the best model among them in terms of all metrics. The reason for this good result still remains unknown to me. It could be that the model properly assumes a hypergeometric term frequency distribution and uses Popper's normalization for this dataset, though this speculation may not be reliable. Since the DFR framework relies on word frequencies, and the word frequencies used here are not close to the real word distribution due to improper preprocessing, as I mentioned in Section 3.2. There

[4]Here, I am trying to give an intuition about this mechanism, and to understand it accurately, this site might be a good choice.

Models	MAP	nDCG5	nDCG10	nDCG20	MRT
TF-IDF	0.2499	0.4621	0.4448	0.4173	16.6747
BM25	0.2483	0.4615	0.4426	0.4151	16.9029
Dirichlet LM	0.2453	0.4514	0.4291	0.4075	**16.5244**
DPH	0.2601	0.4706	0.4555	0.4287	16.8344
Dirichlet LM (Bo2)	0.2346	0.4485	0.4145	0.3828	68.0393
DPH (KL)	0.2984	**0.4890**	**0.4757**	**0.4569**	42.0657
BM25 (Bo2)	**0.3054**	0.4856	0.4721	0.4545	64.5561

Table 5: Performance of probabilistic models, language model and the best DFR models on the training set, as well as the results of the best query expansion technique for these models.

are some studies that support my conjecture, such as Vaidyanathan (2015) showing that the DPH did not achieve the best performance among other DFR models.

Lastly, since the DPH achieved the highest accuracy among all DFR-based models and its non-parametric property, it was chosen to be combined with query expansion in the next section.

3.4 Query expansion

Some topics fail to provide enough information for retrieving relevant documents as discussed in Section 2.1. Query expansion is a reasonable tool to mitigate this issue. Here, I selected three models: `Dirichlet LM`, the BM25 and the DPH, and the reasons for this choice are respectively given at the end of Section 3.1, 3.2 and 3.3. For each model, I tested multiple query expansion techniques, displaying their complete results in Appendix A.2.

These query expansion techniques are roughly divided into two categories: association-based and distribution-based methods (Pal et al., 2013). Association-based methods such as RM3 utilize how a candidate term co-occurs with the original query term. On the contrary, distribution-based term selection such as KL compares the distribution of a term in the whole collection and random distribution with the (pseudo) relevant documents. For example, `KL` calculates term weights by this formula (Plachouras et al., 2004):

$$w(t) = P_x \log_2 \frac{P_x}{P_c}$$

where $P_x = \frac{tf_x}{l_x}$, $P_c = \frac{F}{token_c}$. tf_x is the frequency of query term t in the top-k ranked documents. l_x is the total number of tokens in the top-k ranked documents. F is the frequency of query term t in the whole collection. $token_c$ is the total number of tokens in the whole collection.

Within a naïve intuition about these techniques, Table 5 specifies the best combination between the query expansion technique and the three document-weighted models. To obtain a higher accuracy, `Dirichlet LM` and the BM25 here were further tuned [5] using `5-cross validation` with the grid search algorithm.

Considering the interpretation of the results, the factors behind these combinations are still unclear to me and need to be explored in further experiments. Here, I could only provide some simple observation. First, all query expansion techniques significantly reduce the MAP of `Dirchilet LM` as shown in Table 9, suggesting the susceptible of the Dirchilet smoothing. Second, the tuned BM25 outperforms `Dirchilet LM`. This can be explained that BM25 does not rely on word distribution as much as Dirchilet LM does. Obviously, for BM25, we can interpret it as a measure of similarity between queries and documents, while Dirchilet smoothing cannot not be considered in this perspective. Thus, even without a proper pre-processing step, BM25 can demonstrate its power consistently. Third, comparing the combination of the DPH and KL, BM25 (Bo2) may be more aggressive in expanding new query terms, as it obtained the highest MAP but had a lower DCG in the top-5, 10, 20 documents. Also, from the perspective of efficiency, KL was easier to compute and therefore faster to be computed than Bo2.

3.5 Neural Information Retrieval methods

The purpose of this study is to figure out how to utilize query terms to retrieve relevant documents. In the previous section, we used word frequency to represent the meaning of queries. Besides this option, vectors semantics provides a second perspec-

[5] Table 9 and 11 shows the results of un-tuned version of these two models with different query expansion techniques

Models	MAP	nDCG5	nDCG10	nDCG20
Query expansion (GloVe: 5)	0.1432	0.2538	0.2504	0.2414
Query expansion (GloVe: 2)	0.1909	0.3494	0.3338	0.3180
Cross-Encoder	0.2375	0.4578	0.4365	0.4135
Cross-Encoder (segment: 15)	0.2793	0.4842	0.4717	0.4577
Cross-Encoder (segment: 10)	0.3060	0.5188	0.5007	0.4877
Cross-Encoder (segment: 6)	**0.3163**	**0.5296**	**0.5104**	**0.4943**

Table 6: Performance of the models using embeddings on training set. The phrase $GloVe : k$ means using GloVe embeddings to retrieve k similar words for each term in the queries. The number $\#$ in Cross-Encoder (segment: $\#$) means the number of sentences for each segment.

tive: embeddings (Jurafsky and Martin, 2014). Embeddings often outdo word frequency for carrying word meanings. A specific proof is the parallelogram model proposed by Rumelhart and Abrahamson (1973) solving some analogy problems. Under this framework, we can use word embeddings like GloVe to compute a formula $apple + tree - grape$, obtaining the closest answer $vine$[6]. Therefore, we can utilize semantic information in embeddings for retrieving relevant documents.

3.5.1 Using embeddings to expand queries

My first attempt was to use the pre-trained embeddings to retrieve k similar words to the ones in the original query. I used the python package `Genism` to download 100-dimensional GloVe embeddings and retrieve 5 or 2 similar words for each term in the query (Rehurek and Sojka, 2011). The choice of GloVe was mainly because it consistently outperformed `word2vec` for the same corpus, vocabulary, window size, and training time (Pennington et al., 2014).

As shown in Table 6, the outcome of this attempt is poor and the model's MAP increases as its k decreases. This finding suggests that the quality of new expanded query terms is terrible. There are many possible reasons, such as the failure of GloVe to capture the meaning of polysemous words. However, I think the most important factor is that embeddings, as meaningful units, can only carry most of the lexical relations when embeddings are used as high-dimensional vectors. Any slight modification of their vectors could completely change embeddings meaning, let alone expand queries by words rather than new embeddings.

A more reliable alternative would be to directly compute the similarity between embeddings and documents. Therefore, in the next section, I used Sentence-BERT to achieve this goal (Reimers and Gurevych, 2019).

3.5.2 Using embeddings to re-rank documents

In this section, inspired by the teaching assistant in the NIR course, I first retrieved 1000 relevant documents from the entire collection using the BM25 (Bo2) as shown in Table 5. Then, the documents returned by the first stage were reordered using Sentence Transformers. The model has been trained on the dataset MS MARCO and the architecture I used was, Cross-Encoder, due to its higher performance than Bi-Encoder (Reimers and Gurevych, 2019) [7].

The result of this pre-trained model is the third one in Table 6. Although the result largely improves compared to using embedding to extend queries, pre-trained embeddings without fine-tuning still drag down the model's performance to a large extent. Thus, if we have sufficient computational resources, we can train the Cross-Encoder on the NIR dataset to improve its performance. However, if these resources are not available, we need to find another way to mitigate the impact of the lack of fine-tuning.

In this case, a method proposed by Nogueira et al. (2020) may address this issue. The method firstly segment each document into passages by applying a sliding window of k sentences with a stride of b. For each passage and query, a model such as Sentence-BERT will calculate a relevance score and the highest score is selected as the relevance probability of the document. In this way, a score can be obtained with fewer pre-trained embeddings, largely reducing noises especially when users only need one or two paragraphs of each document.

As shown in Table 6, the results show that relying on 6 sentences in the document is a good choice to retrieve queries on the NIR dataset.

[6]Note that this result requires to exclude the input words and their morphological variants such as $apples$

[7]The official explanation of the difference between Bi-Encoder and Cross-Encoder can be found here.

Models	MAP	nDCG5	nDCG10	nDCG20	Rank
Cross-Encoder (segment: 6)	**0.3144**	**0.5335**	0.5095	**0.4954**	**1**
Cross-Encoder (segment: 10)	0.3060	0.5333	**0.5140**	0.4818	2
Cross-Encoder (segment: 15)	0.2738	0.4987	0.4543	0.4428	19
Cross-Encoder	0.2493	0.4538	0.4410	0.4163	38
BM25 (Bo2)	0.2491	0.4439	0.4113	0.3998	41
DPH (KL)	0.2377	0.4206	0.4017	0.3744	62

Table 7: Performance of the models on unseen topics. The number # in Cross-Encoder (segment: #) means the number of sentences for each segment. Rank refers to the ranking of the model's MAP among all models submitted in the NIR course.

4 Models: unseen queries

In the previous sections, the BM25 (Bo2) and DPH (KL) have achieved relevant good performance compared to other classical methods. The runs of these two models and Cross-Encoder have been submitted to see how they perform in unseen queries, as displayed in Table 7.

The performance of the classical IR method is largely degraded for unseen queries. One possible explanation for this is the overfitting of the models on the training set. However, for the DPH (KL), I did not tune any parameter except for the choice of query expansion technique. Thus, this explanation is at least not a major factor. To get more clues, I looked at the test queries and found that five queries were improperly pre-processed, a higher percentage compared to the training queries (`qid`: 339, 391, 409, 443, 634)[8]. Also, I assume that a improper pre-processed query will perform poorly and below average. Based on this assumption, I believe that the test queries provided less information for retrieving documents than the training queries did.

Compared to classical IR methods, the Cross-Encoder is incredibly robust, whether I segment documents or not. This result may be benefit from the use of pre-trained models without fine-tuning. Embeddings often can capture meaning-bearing units such as the morphemes $-est$ or $-er$, which may ensure a stable result when encountering improperly pre-processed.

For using k sentences form documents to compare with per query, this decision helped me to obtain the best result. However, the value k depends mainly on the dataset, and I think this value may have poor generalization power.

[8]For poorly pre-processed queries, there were approximately 7 queries out of 150 training queries and 6 queries out of 50 test queries.

5 Conclusion

The flaws or possible improvement in this study can be roughly divided into two categories: the pre-processed dataset and the used models

At the beginning, the improper pre-processed queries mentioned in Section 2.1 cause unnecessary difficulties. More importantly, the previous sections demonstrates the importance of providing a true lexical distribution for our models such as `Dirichlet LM`. Therefore, a lemmatizer using POS tag should be considered in the following research. In addition, due to the different document structure, I did not put effort into extracting the correct field information from the documents, which can be explored in the future.

Considering the models, this paper compromises a lot to computational resources. For instance, with many parameters in query expansion techniques or document-weighted models, I often fixed several choices to come up with the best method, and dispensed with the rest. However, it is possible to improve on the best single method by combining the results of several, without necessarily identifying which is best (Büttcher et al., 2016). Thus, fusion mechanism should be helpful in this direction. Lastly, in order to make embeddings more suitable for the NIR dataset, the transfer learning paradigm should be complete, i.e., using the fine-tuning technique (Jurafsky and Martin, 2014).

References

Gianni Amati, Edgardo Ambrosi, Marco Bianchi, Carlo Gaibisso, and Giorgio Gambosi. 2007. Fub, iasi-cnr and university of tor vergata at trec 2007 blog track. In *Proceedings of the 16th Text REtrieval Conference (TREC-2007)*.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text*

with the natural language toolkit. " O'Reilly Media, Inc.".

Stefan Büttcher, Clarke Charles L A., and Gordon V. Cormack. 2016. *Information retrieval: Implementing and evaluating search engines*. MIT Press.

Donna Harman. 1992. Evaluation issues in information retrieval. *Information Processing and Management*, 28(4):439 – –440.

Stephen Paul Harter. 1974. *A probabilistic approach to automatic keyword indexing*. Ph.D. thesis, University of Chicago.

Kalervo J"arvelin and Jaana Kek"al"ainen. 2002. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446.

Dan Jurafsky and James H. Martin. 2014. *Speech and language processing*. Prentice Hall, Pearson Education International.

Craig Macdonald and Nicola Tonellotto. 2020. Declarative experimentation ininformation retrieval using pyterrier. In *Proceedings of ICTIR 2020*.

Rodrigo Nogueira, Zhiying Jiang, and Jimmy Lin. 2020. Document ranking with a pretrained sequence-to-sequence model.

I. Ounis, G. Amati, V. Plachouras, B. He, C. Macdonald, and C. Lioma. 2006. Terrier: A High Performance and Scalable Information Retrieval Platform. In *Proceedings of ACM SIGIR'06 Workshop on Open Source Information Retrieval (OSIR 2006)*.

Dipasree Pal, Mandar Mitra, and Kalyankumar Datta. 2013. Query expansion using term distribution and term association.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Vassilis Plachouras, Ben He, and Iadh Ounis. 2004. University of glasgow at TREC 2004: Experiments in web, robust, and terabyte tracks with terrier. In *Proceedings of the Thirteenth Text REtrieval Conference, TREC 2004, Gaithersburg, Maryland, USA, November 16-19, 2004*, volume 500-261 of *NIST Special Publication*. National Institute of Standards and Technology (NIST).

Radim Rehurek and Petr Sojka. 2011. Gensim–python framework for vector space modelling. *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, 3(2).

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks.

J. J. Rocchio. 1971. Relevance feedback in information retrieval. In G. Salton, editor, *The Smart retrieval system - experiments in automatic document processing*, pages 313–323. Englewood Cliffs, NJ: Prentice-Hall.

David E Rumelhart and Adele A Abrahamson. 1973. A model for analogical reasoning. *Cognitive Psychology*, 5(1):1–28.

KAREN SPARCK JONES. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1):11–21.

Rekha Vaidyanathan. 2015. A study on retrieval models and query expansion using prf. *International Journal of Scientific and Engineering Research*, 6.

Cornelis J Van Rijsbergen. 1979. Information retrieval.

A Complete experimental results

A.1 Divergence From Randomness based models

Models	MAP	nDCG5	nDCG10	nDCG20
DPH	**0.2601**	**0.4706**	**0.4555**	**0.4287**
InL2	0.2528	0.4565	0.4420	0.4173
In_expB2	0.2522	0.4604	0.4472	0.4207
BB2	0.2521	0.4600	0.4451	0.4191
IFB2	0.2508	0.4593	0.4440	0.4179
In_expC2	0.2471	0.4672	0.4472	0.4186
PL2	0.2369	0.4596	0.4371	0.4084

Table 8: Performance of certain DFR based models provided by PyTerrier. The name convention for the models follows the official PyTerrier documentation.

A.2 Query Expansion

Due to limited computational resources, the models at this stage used the default hyperparameters in PyTerrier. For retrieval models, such as the BM25, their formulas and default parameters can be found here. For query expansion models, such as Bo1, Bo2, their information is displayed here.

For the results of using the three models (`Dirichlet LM`, the BM25 and the DPH) with different query expansion models are shown in the following tables.

Models	MAP	nDCG5	nDCG10	nDCG20
Bo1	0.2144	**0.4346**	**0.3958**	**0.3638**
BA	0.2129	0.4272	0.3919	0.3595
KLComplete	0.2128	0.4261	0.3919	0.3593
KL	0.2127	0.4275	0.3910	0.3585
KLCorrect	0.2126	0.4275	0.3910	0.3583
RM3	0.2073	0.3980	0.3711	0.3446
Information	0.1535	0.3716	0.3203	0.2907

Table 9: Performance of using `DirichletLM` with different query expansion models (default parameters).

Models	MAP	nDCG5	nDCG10	nDCG20
KL	**0.2984**	**0.4890**	**0.4757**	0.4569
Bo1	0.2978	0.4866	0.4733	**0.4580**
RM3	0.2927	0.4791	0.4704	0.4537
KLCorrect	0.2887	0.4799	0.4718	0.4521
Bo2	0.2877	0.4776	0.4537	0.4386
KLComplete	0.2663	0.4665	0.4355	0.4119
BA	0.2661	0.4703	0.4366	0.4119
Information	0.0172	0.0765	0.0654	0.0587

Table 10: Performance of using the DPH with different query expansion models (default parameters).

Models	MAP	nDCG5	nDCG10	nDCG20
Bo2	**0.2977**	**0.4941**	**0.4706**	**0.4542**
KL	0.2896	0.4856	0.4648	0.4504
Bo1	0.2895	0.4837	0.4689	0.4512
RM3	0.2839	0.4733	0.4619	0.4423
KLCorrect	0.2807	0.4808	0.4624	0.4430
BA	0.2765	0.4829	0.4557	0.4296
KLComplete	0.2761	0.4830	0.4555	0.4311
Information	0.0232	0.1295	0.1025	0.0857

Table 11: Performance of using the BM25 with different query expansion models (default parameters).