Retrieving Solutions to Puzzles & Riddles with both Traditional & New IR Systems

Ellis S Fitzgerald
ellis.fitzgerald@maine.edu
University of Southern Maine
Portland, Maine, USA

Abstract

Using Traditional Information Retrieval Model's such as Okapi BM25 and TF-IDF for searching across the Puzzling Stack Exchange [1] shows BM25 dominance for metrics like MAP, nDCG, and BPref, despite the nature of Puzzle and Riddle questions often lacking descriptive terms or context. The Puzzling Stack Exchange [1] poses as a unique case study and environment where many questions are answered by rephrasing the question, making the retrieval sometimes trivial. There we're tests for BM25 and TF-IDF with stemming the queries and documents but **not** expanding them. With the use of Porter Stemming from Martin Porter [4], BM25 and TF-IDF showed to have stronger results for most metrics. Additionally, a "weak" porter stemming algorithm performed even better as well. However, there was no significance testing done in this article for whether or not stemming provides significantly better results for these traditional models. After using PyTerrier [3] for the BM25 Okapi and TF-IDF implementations, the intention was to use these traditional models as *baselines* to do comparisons and significance testing on modern systems such as Sentence-BERT [5] Bi-Encoder's and Cross-Encoder's. In this paper we used one model of each both trained on Microsoft's MARCO [6] dataset containing 500k real user queries from Bing matched with relevant passages [6]. The Bi-Encoder would do a standard encoding and similarity, while the the Cross-Encoder would re-rank with predictions. This was done with the pretrained models as well as fine-tuned models which was done by splitting our Puzzling Stack Exchange [1] dataset into train, evaluation, and test sets.

CCS Concepts: • **Applied computing** → **Document searching**.

COS470P2, Portland, ME,

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Keywords: Information, Retrieval, Okapi, BM25, TF-IDF, Puzzles, Riddles, Porter, Stemming, SBERT, Neural, Network, BiEncoder, CrossEncoder, MSMARCO

ACM Reference Format:
Ellis S Fitzgerald. 2024. Retrieving Solutions to Puzzles & Riddles with both Traditional & New IR Systems. In *Proceedings of October 22, 2024 (COS470P2).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Puzzles and riddles for many are fun *(or frustrating)* games to play to kill time. For others, they are intensive, often inductive mathematical problems. Few have interpretive answers; many have conclusive answers. The ones we are interested in and the ones you will most likely find in our test collection, the Puzzling Stack Exchange [1], are the latter.

One thing that makes puzzle and riddle information retrieval rather troublesome is that queries intentionally leave out terms that may be related to the solution. Leaving the retrieval process rather complicated. In other words, the terms used between both the question and the answer, or the intersection of both word sets cardinality, can be none *(this becomes less likely if not ignoring stop words).* In the adolescence of this paper, there was a list of information retrieval systems to compare, and for the reason just mentioned, Boolean Retrieval was eliminated almost immediately.

In this paper, we will discuss using two different Traditional Information Retrieval models from the PyTerrier API [3], Okapi BM25, and TF-IDF, for our given topic. We will go over which seems to be superior in the sense that we find more conclusive answers which we assume to be correct given by an outsourced QREL. Given that superiority, we can determine that BM25 or TF-IDF can retrieve more relevant results for our test collection. Afterwards, we will use these baseline results to compare to our Bi-Encoder and Cross-Encoder's results for ranking and reranking.

My use of the PyTerrier API [3] in order to achieve these answers is to be modular enough so that permutations can be made on the fly; such as the stopword set, the stemming algorithm, and tokenization. My hypothesis is that stemming may provide a way to offer context in riddles where there is not any, without the use of any neural language models or processing. In example, if a riddle contains "wordplay," the stemming may be able to offer context between the

Table 1. Okapi BM25 & TF-IDF at Default

Parameter	Value
`Stemming`	None
`Tokenization`	English Terrier Provided
`Stop-words`	Terrier Provided

connection between the word morn and morning, whereas without stemming these terms would be treated completely differently.

Afterwards, once we get to the Neural Networks where language models can be generated and sequences can be converted to vectors to get term relationships, the hypothesis and assumption is that Cross-Encoder will perform better for its re-ranking as most research would suggest.

2 Methodology

Python-Terrier, a Python wrapper for the open source search engine Terrier, which includes various standardized implementations for Information Retrieval systems and their counterparts such as stop-words, stemming, and tokenization. This is where our BM25 and TF-IDF implementation will come from. My usage takes advantage of the given API by allowing on-the-fly permutation changes with command-line arguments for different indexing and retrieving methods. The reason for such an implementation was purely for convenience and testing. When referring to default parameters, I do not mean PyTerrier run with the least minimal parameters provided, but instead my own definition of “default” as seen in **Table 1**.

There are various parameters to be set while indexing with PyTerrier, but most of it is concerned with high memory issues. Considering that our test collection is limited in size (63,997) and no memory issues were encountered in our runs, this was not introduced in our methodology. It is ensured in the PyTerrier documents that changing those parameters results in the same success rates for the index regardless.

Consistently between all systems and for both Queries and Documents: HTML, unicode, code, and punctuation is either removed or replaced with white space. Stop-words are also considered to be removed, but as mentioned before the set of words differs for the PyTerrier tests. After that, consistently for all systems there are only 100 returned documents for each query.

The model used for the Bi-Encoder and Cross-Encoders **msmarco-distilbert-base-v4** and **cross-encoder/ms-marco-MiniLM-L-6-v2** respectively. As mentioned in the abstract these models we’re both trained on a Microsoft database MS-MARCO [6] from their Bing search engine which contains 500k real user queries matched with relevant passages. This seemed like a good choice out of other models available as others we’re not related to Information Retrieval tasks. The biggest concern is that the "DistilBert" models reduce the tokens in each passage to 512, which on average is 300-400 english words. When calculating the maximum character length (after preprocessing) of answers in our test collection, we find one of length 26,028, posing as a potential problem as our Bi-Encoder is the core to all systems.

For both the Bi-Encoder and the Cross-Encoder the same split of the document collection was used. 90% of the documents was dedicated to training, while the remaining 10% was divided into two 5% portions for evaluation and testing respectively.

Cosine was chose to be the similarity function between different vectors. This means that normalization was performed within the function implementation provided by SBERT [5].

Encoding, ranking, and fine-tuning was done on a single NVIDIA CUDA GPU 4070Ti. The searches (done on the Bi-Encoder) are divided into queries chunks of size 100 and document chunks of size 50,000. Afterwards the predictions are done on pairs returned from the top 100 results for each Query.

For the fine tuning only 1 epoch was used for the Bi-Encoder model due to long training times and because of fears of over-fitting considering the relatively small collection size. Contrastingly the Cross-Encoder was given an epoch count of 4.

The Bi-Encoder and Cross-Encoder were given different evaluators. Bi-Encoder trained with an "InformationRetrievalEvaluator" keeping track of Mean Reciprocal Rank, Normalized Discounted Cumulative Gain, and Mean Average Precision at 1, 5, and 10. The Cross-Encoder used the "CERerankingEvaluator" hence its role being a re-ranker.

The datasets used in training for both Bi-Encoder and Cross-Encoder differed in format but overall were relatively consistent in how they were constructed. Using the external QREL, each Query with Documents matched in its ranking that were deemed relevant were given positive scores, meanwhile everything else was deemed irrelevant. In order to not make the process manual, there was an attempt to make sure there was at least as many negative (documents not found in the ranking) examples as there were positive (documents found in the ranking). Finally, the loss function used in the Bi-Encoder was CoSENTLoss. The Cross-Encoder is technically on version-2 (unlike the Bi-Encoder’s version-3) so the loss function is actually implemented in the "CERerankingEvaluator."

2.1 Experimental Analysis

In my experimental analysis I used TREC standardized evaluation methods within PyTerrier to compare the BM25 and TF-IDF models. Afterwards, all evaluations done are with the Ranx [2] Python library. Given the results of BM25 and TF-IDF we were able to evaluate metrics in batches, compare by hand for averaged values, as well as compare both directly per query retrieval with our QRELs.

Table 2. BM25 & TF-IDF

System	MAP	nDCG
BM25	**0.45767**	**0.53763**
TF-IDF	0.45297	0.53381

Table 3. BM25 Baseline & TF-IDF, K = 1, 5, 100

System	P@5	P@10	P@100
BM25	**0.47047**	**0.31365**	**0.04139**
TF-IDF	0.46546	0.30926	0.04107
P-Value	**4.39E-05**	**4.74E-09**	**9.03E-06**

Table 4. BM25 with Porter & Weak-Porter

System	MAP	nDCG
BM25 Porter	0.45911	**0.54026**
BM25 Weak-Porter	**0.45917**	0.54013
BM25 Default	0.45767	0.53763
TF-IDF Default	0.45297	0.53381

In my findings, Okapi BM25 was seen as superior at default, see **Table 2** and **Table 3**. We did find there to be statistical significance for the differences mostly in P@10, see **Table 3.** This is to be somewhat expected given its length normalization parameters (left at default given PyTerrier's implementation).

After conducting the comparisons on the default BM25 and TF-IDF we compared the models with Porter Stemming and Weak-Porter Stemming with PyTerrier's provided implementation were done (seen in **table 4**. As mentioned in the beginning of the paper, there was no statistical significance testing done for this part, but it is interesting to see. Bolded numbers showcase greater metric values. BM25 Porter has a better average nDCG while BM25 Weak-Porter has a better MAP. TF-IDF with both stemmers was tested, but was ommitted from the table because of BM25's dominance.

Graphs can be seen for default BM25 and TF-IDF's performance at P@5 for each query in one of our query files in **Figure 1 and 2** respectively. As you can see with the graphs, the distribution of Precision@5 is very similar for both BM25 and TF-IDF, with values ranging from 0-5 it is almost a perfect stair-case. These graphs get more interesting to look at when we consider the Neural Network implementations.

First, let us look at some good and bad results for BM25 and TF-IDF, and then let us discuss why these may have come to be.

In **Figure 3** is demonstrated a highly rated retrieval. **Query 102076** matched with **Answer 102091**. Highly rated means when evaluated in TREC Standard format the Precision@1 =

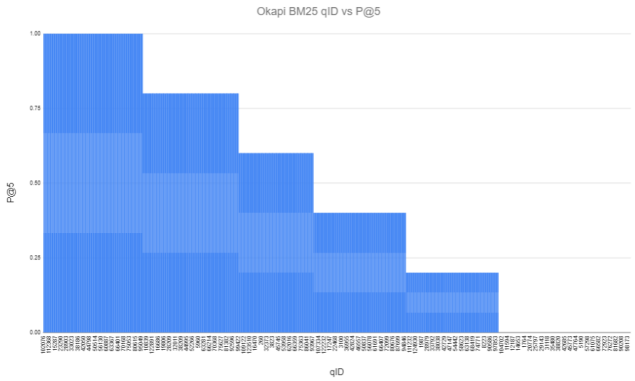

Figure 1. BM25 Default P5 for each query in topics_1.json

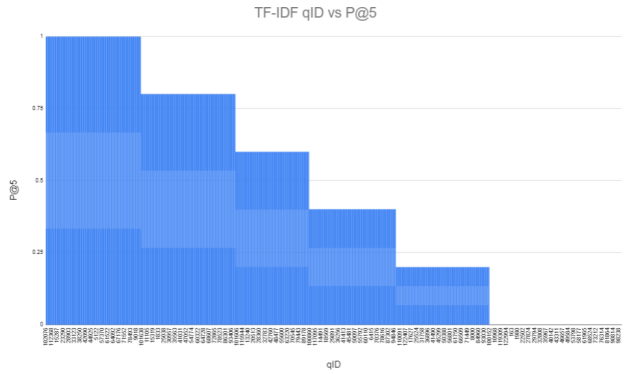

Figure 2. TF-IDF Default P5 for each query in topics_1.json

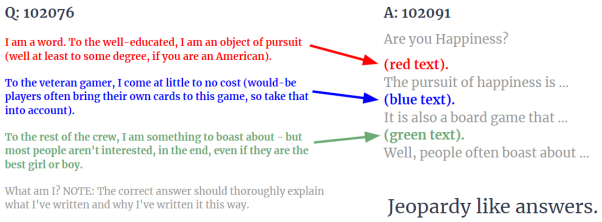

Figure 3. A high rated retrieval, showing repeated phrases within both query and answer.

1. This was both TF-IDF and BM25's top result for this query. Knowing the text is not too important, what is important is noticing the pattern of how the answer restates the query almost word for word. Making this one of the trivial results. The word frequencies are probably close to identical, and the lengths are probably similar as well. Resulting in it being the highest candidate for both.

In **Figure 4** is demonstrated a negatively rated retrieval for both BM25 and TF-IDF. **Query 99712** matched with **Answer**

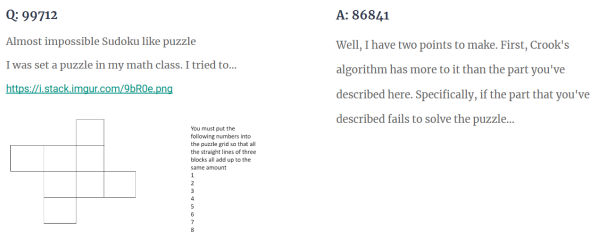

Figure 4. A low rated retrieval. Most of it's query was contained in an image.

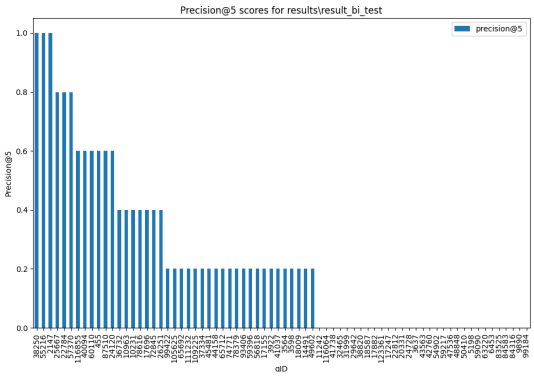

Figure 5. Pretrained Bi-Encoder Results for Precision@5 in all Queries in Test Set

86841. This means that for Precision@5 = 0, meaning the top 5 results resulted in irrelevant passages. The answer provided for this example could be any of the top 5 presented by BM25 and TF-IDF, but I chose to provide what both systems thought was the best. Again, similarly to the highly rated result you do not need to know the specifics to know why this may have happened. What is most important to note is that in this query a link to the website "imgur" is provided. The website links to an image of a sudoku problem. The query text is more like commentary than an actual question. This shows an obvious weakness for our system: it can't embed or encode images. This is something that Neural Networks can do, it is not something we will do in this paper, but shows an obvious weakness to text retrieval when these platforms contain many forms of media.

Now that we have covered our baseline/traditional models (BM25 & TF-IDF) we can go over our Neural Network SBERT [5] Bi-Encoder and Cross-Encoder retrieval. As seen in **Table 5** we can compare both our pretrained and fine-tuned Bi-Encoder and Cross-Encoder. One thing to note is that we do use the same document collection for both the traditional models and the neural networks. However, because we intended to train/fine-tune our neural networks, some queries and answers had to be reserved for training purposes. As mentioned in the methodology section. The split is 90% train, 5% evaluation, and 5% test. The results you see in **Table 5** showcase results from that test set. Because the test set is subset of the document collection used for the traditional models a direct statistical test cannot be approached.

As given in the **table 5** we can see that the pretrained Cross-Encoder for all metrics has statistically significantly better performance compared to the Bi-Encoder except for Mean Reciprocal Rank. For the fine-tuned Bi-Encoder we can see improvements to the Bi-Encoder results across the board, but it is not enough to be statistically significant. This could be both the weakness of the Bi-Encoder, potential over-fitting, or perhaps different training methods could be done (different loss function, evaluator, etc). Finally, the fine-tuned Cross-Encoder has statistically significantly better performance compared to all the other models, even in Mean Reciprocal Rank (unlike pretrained Cross-Encoder). This is encouraging to see, as it shows that not only is re-ranking with the Cross-Encoder improving results, fine-tuning is as well.

Looking at **Figures 5-8** you can see results for both pretrained and fine-tuned encoder models. Notably the distribution for the pretrained model is not as evenly divided compared to TF-IDF and BM25. It is quite visually clear based off the graphs as well as **Tables 2, 3 5** show that the neural networks are not competing as much as they should with BM25 and TF-IDF. Considering the runtime for the Neural Networks as well as the memory space they occupy, the encoders do not seem like a sensible approach on their own.

Last but not least we can take a look at accurate results coming from the encoders both pretrained and fine-tuned in **Figure 9**. Contrastingly the results for all the encoders for the top results differ, unlike BM25 and TF-IDF. Each of these high rated results are queries with Precision@5 = 1. Meaning all the results were deemed relevant. Since we are operating with the Puzzling Stack Exchange [1] most of these queries are riddle related. In these good examples each are riddles with specific key words, such as names. Even more peculiar is that the best result for the fine-tuned Bi-Encoder **(Q: 55216, A: 55235)** and the pretrained Cross-Encoder **(Q: 57370, A: 57385)** are riddles that contain groups of names such as: Riddler, Penguin, Joker, Two-Face; and Peter, Tom, John, Ralph. Since names are not words that are always used in the alphabet and used in conjunction with each other, I believe that the language model being generated benefits these queries because the names are so close to each other. Then finally, the uniform bad result for all the encoders (Q: 99184, A: 109108) was a question wrapped in math containers that were most likely preprocessed to oblivion. The answer was a mathematical proof that was unrelated. In this case, being able to parse those math containers would probably yield better results but becomes more niche.

Table 5. Overall effectiveness of the models. The best results are highlighted in boldface. Superscripts denote significant differences in paired Student's t-test with $p \leq 0.01$.

#	Model	NDCG@5	NDCG@10	P@5	P@10	MAP	BPref	MRR
a	results_bi_test	0.236	0.240	0.225	0.147	0.189	nan	0.463
b	results_ce_test	0.335^{a}	0.337^{a}	0.308^{a}	0.203^{a}	0.268^{a}	nan	0.566
c	results_bi_ft_test	0.271	0.285	0.239	0.169	0.228	nan	0.474
d	results_ce_ft_test	$\mathbf{0.428}^{abc}$	$\mathbf{0.435}^{abc}$	$\mathbf{0.414}^{abc}$	$\mathbf{0.268}^{abc}$	$\mathbf{0.366}^{abc}$	nan	$\mathbf{0.641}^{ac}$

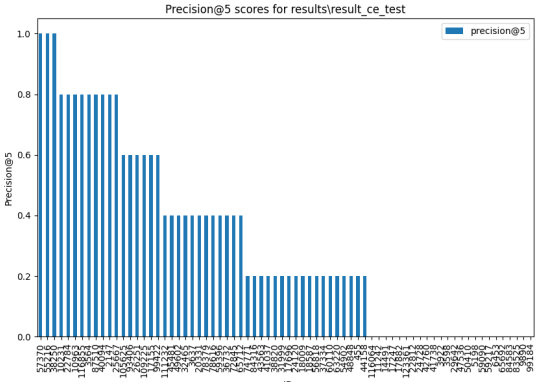

Figure 6. Pretrained Cross-Encoder Results for Precision@5 in all Queries in Test Set

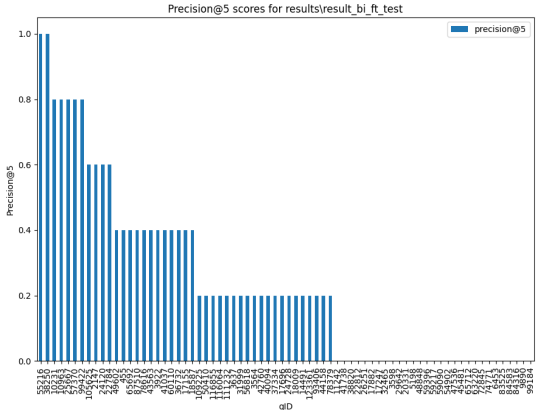

Figure 7. Fine-tuned Bi-Encoder Results for Precision@5 in all Queries in Test Set

2.2 Conclusion

For the traditional Information Retrieval systems/models, we can see given by our experimental analysis that regardless of the permutation or preprocessing methods, the Okapi BM25 implementation performs better. There are some cases where TF-IDF performed better for some averaged metrics, however none of them we're deemed to be statistically significant unlike BM25. In our experimental analysis section we were

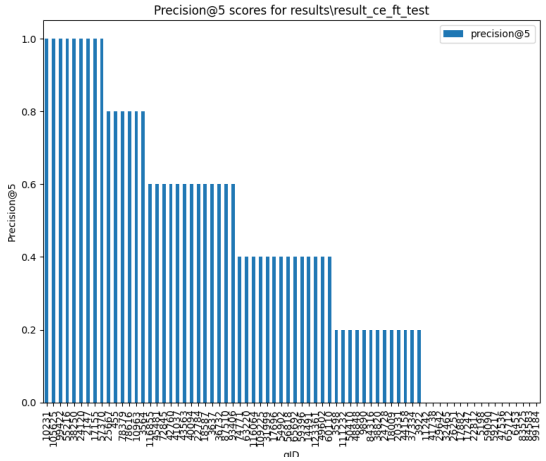

Figure 8. Fine-tuned Cross-Encoder Results for Precision@5 in all Queries in Test Set

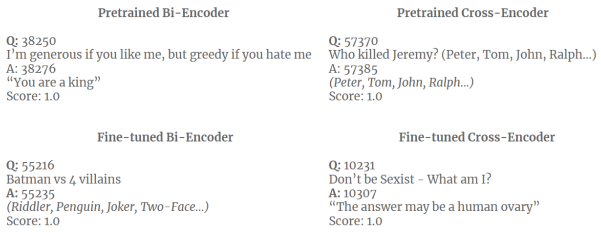

Figure 9. High rated retrievals for Pretrained/Fine-tuned Bi/Cross-Encoder's

able to prove there is statistical significance for precision at 10, 100, and 5 for BM25's dominance. Additionally, we found how PyTerrier's implementation of the Porter stemmer and Weak-Porter stemmer did provide stronger results for both TF-IDF and BM25. However, there was no statistical significance test done in these experiments. Finally, with the use of SBERT Bi-Encoder and Cross-Encoder we were able to divide our test collection into train, evaluation, and test triplets so we could encode, rank, and re-rank using neural networks. In the end, we found that the fine-tuned

Cross-Encoder was significantly more stronger in all metrics for retrieval. Showcasing promising results, but still not as performant nor efficient as BM25. Different training methods should be evaluated to potentially maximize the neural networks ability and minimize overfitting.

Acknowledgments

PyTerrier, for ironically failing to provide search results on their document page. Thank you to all of their authors for providing better IR code than I could ever produce.

References

[1] [n. d.]. Puzzling Stack Exchange. https://puzzling.stackexchange.com/ Accessed: 2024-10-18.

[2] Elias Bassani. 2022. ranx: A Blazing-Fast Python Library for Ranking Evaluation and Comparison. In *Advances in Information Retrieval - 44th European Conference on IR Research, ECIR 2022, Stavanger, Norway, April 10-14, 2022, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 13186)*, Matthias Hagen, Suzan Verberne, Craig Macdonald, Christin Seifert, Krisztian Balog, Kjetil Nørvåg, and Vinay Setty (Eds.). Springer, 259–264. https://doi.org/10.1007/978-3-030-99739-7_30

[3] Craig Macdonald and Nicola Tonellotto. 2020. Declarative Experimentation in Information Retrieval using PyTerrier. In *Proceedings of ICTIR 2020*.

[4] Martin Porter. 2006. Porter Stemming Algorithm. https://tartarus.org/martin/PorterStemmer/

[5] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. http://arxiv.org/abs/1908.10084

[6] Nils Reimers and Iryna Gurevych. 2021. The Curse of Dense Low-Dimensional Information Retrieval for Large Index Sizes. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. Association for Computational Linguistics, Online, 605–611. https://arxiv.org/abs/2012.14210

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009