Golden Retrievers ARQMath Project

Daniel Castellarin[1], Jameson Toper[1] and Ryan Carnation[1]

[1]*Rochester Institute of Technology, 1 Lomb Memorial Drive, Rochester, NY, 14623, United States*

Abstract

This paper proposes a system for the CLEF 2022 ARQMath-3 lab Task 1 using PyTerrier. The goal of this lab is to improve the effectiveness and efficiency of retrieving answers to mathematical questions. Given a BM25 base retrieval model, the task was to improve its performance both in effectiveness and efficiency. This can be achieved by first analyzing the weaknesses found in the BM25 system and implementing solutions to fill those gaps. The baseline model exhibited poor effectiveness, namely due to the implementation of the index. To fix this, we proposed an experiment which implements a new pre-processing pipeline that utilizes Sequential Dependence Modeling (SDM) to expand queries to analyze the term proximity within documents. We expected that the new pipeline would improve upon the effectiveness of the baseline. Another weakness observed in the baseline model was the absence of a re-ranking model. Through the addition of a re-ranker, specifically KNRM, we expected the effectiveness of the retrieval to improve in terms of the standard ARQMath metrics: nDCG', P'@10, and mAP'. As a result of our experiment, we found that our pre-processing pipeline resulted in lower values of nDCG', P'@10, and mAP' compared to the baseline system. The pre-processing pipeline also exhibited a considerably higher mean response time than that of the baseline system. In terms of re-ranking, BM25 combined with KNRM with the BERT vocabulary was able to outscore baseline BM25 in nDCG', P'@10, and mAP' while lowering the response time. Lastly, BM25 combined with KNRM with the word vector vocabulary performed slightly better than the BERT vocabulary in nDCG' and P'@10 with a slightly lower mAP' value. The word vector implementation also had a faster response time than both of the other systems.

Keywords

ARQMath, BM25, KNRM, SDM, Information Retrieval, PyTerrier, Neural IR

1. Introduction

Prior to the creation of the ARQMath lab, math information retrieval was studied at various conferences. Notable work was performed at NTCIR, where test collections had been created to evaluate the performance of math information retrieval systems. In 2013 at NTCIR-10, teams were tasked with retrieving relevant mathematical formulae from a collection of 100,000 scientific articles for a given query [1]. Later, namely in 2014 and 2016 at NTCIR-11 and NTCIR-12 respectively, an additional corpus was added for teams to retrieve formulae from Wikipedia articles, most of which did not include math in an attempt to fool keyword matching efforts for participating retrieval models[2].

drc3203@rit.edu (D. Castellarin); jxt1009@rit.edu (J. Toper); rdc1996@rit.edu (R. Carnation)
https://github.com/danielcastellarin (D. Castellarin); https://github.com/jxt1009 (J. Toper);
https://github.com/rdc1996/resumeProjects (R. Carnation)
0 (D. Castellarin); 1 (J. Toper); 2 (R. Carnation)

The advent of the ARQMath lab has provided a new opportunity for researchers to develop systems for mathematical information retrieval. The ARQMath lab was introduced to generate math-related test collections, improve evaluation methods, and further explore the capabilities of math retrieval systems[3]. The ARQMath Collection contains sets of answers obtained from years of archived posts on Math StackExchange, a forum popular for the discussion of math-related topics. In 2020, ARQMath-1 introduced two tasks: answer retrieval and formula retrieval. In this paper, we will describe our implementation and testing for a math retrieval system in response to the answer retrieval task.

We utilized a PyTerrier[4] implementation (pt-arqmath) developed by Richard Zanibbi and Behrooz Mansouri as a baseline for our experiments [5]. This code included question and answer pre-processing along with a pre-implemented BM25 baseline retrieval model. However, the baseline had some notable flaws. First, the baseline utilized a standard inverted index with term frequencies stored per document and using stopword removal and porter stemming. We believed that this index was not storing enough information and that the inclusion of word position tracking and Sequential Dependence query expansion would improve the baseline model's effectiveness. Second, the baseline did not utilize any form of re-ranking to attempt to percolate answers with more relevance to the top of the result set. Since re-ranking has become a standard method for improving the effectiveness of retrieval models, we tested whether KNRM could improve the effectiveness of the baseline model.

Our pre-processing experiment changed from what we initially planned. Originally, we wanted to test the effect of using docT5query to complement the implementation of the baseline indexing. DocT5Query [6] is a document expansion strategy that can generate potential queries for a given document using the document's contents. We planned to use the model pre-trained on the MS-MARCO dataset to generate these queries, append them to the end of each answer document, and observe the results. However, the indexing operation was taking too long to compute, so we had to change experiments due to time constraints. Our alternative experiment tested the effectiveness of using query expansion via the Sequential Dependence Model, where documents will be scored higher for query terms occurring in close proximity[7]. Word position data would be added to the original index to allow proximity calculations to be performed.

Our approach to re-ranking was to implement the KNRM neural re-ranker. We did so using two different built-in vocabularies that it offers, Word-Vec_Hash and BERT. Without knowing which one was better for the given task, we tested each and compared their results to the baseline BM25 model. KNRM was trained using the ARQMath-2020 dataset and validated using the ARQMath-2021 dataset.

The results of the pre-processing experiment did not turn out as hypothesized. The overall effectiveness of the model decreased after implementing SDM. The efficiency also got considerably worse with our new pre-processing implementation. The re-ranking experiments had more success, with the results showing that the trained KNRM model with the vocabulary of WordVec_Hash is capable of outperforming every other implementation. The vocabularies made little difference in the results, but Word-Vec_Hash consistently slightly outperformed BERT.

2. Baseline Model (BM25)

Prior to designing our own experiments, we had to identify the capabilities of a baseline model. Specifically, the baseline we analyzed was a first-stage BM25 ranking that covered all ARQMath-1 topics (2020). This baseline was implemented using PyTerrier, a Terrier Python framework that facilitates the creation of information retrieval experiments, developed by Craig Macdonald and Nicola Tonellotto [4]. PyTerrier is traditionally used for text-based search engines, so this implementation (developed by Richard Zanibbi and Behrooz Mansouri) was specially designed to handle math formulas. In Table 1, we can see the results of this baseline test. There are a few reasons for the failure of the baseline system.

Table 1
Effectiveness results of the baseline model (BM25) run on all topics from ARQMath-1. Effectiveness measured with nDCG', P'@10, and mAP'

Name	nDCG'	P'@10	mAP'	MRT
BM25	0.105284	0.07013	0.043515	1335.21363

The most glaring flaw relates to how the data is indexed for this implementation. The questions and answers in the ARQMath collection contain both text and formulas. Math is made up of LaTeX formulas, which contain a lot of punctuation. However, PyTerrier has a default feature that removes punctuation from documents. This has been standard practice for many IR systems because it simplifies index generation, and the index uses less space when it does not account for punctuation. In order to process text and formulas together, the punctuation contained in LaTeX formulas is translated to an English representation. One example of this remapping format is a pound sign, "#", which maps to the English word "hash". This operation causes formulas to lose some of their meaning because they have been broken up into separate terms. In the following, we show an example of how queries are transformed.

Original query: how to solve $x^2 + 17x - 4xy + 6y - y^2 = 10$?
Its translation: how to solve colon x power 2 plus 17x minus 4xy plus 6y minus y power 2 equals 10 question

The idea that formulas can lose their meaning when their contents are spread across multiple index terms is compounded by the fact that this BM25 model weights terms independently of each other when ranking documents. With this approach, the contextual meaning of each term is lost. Each term in a formula will be less indicative of the topic the formula represents. When searching, the model will be less effective at returning topically relevant results for similar formulas than if it had access to contextual information. An example of a potential fix would be to track word location or proximity in the index. The baseline model could also benefit from other procedures that account for contextual information in the ARQMath Questions and

Answers. For example, the addition of neural rankers could help analyze contextual information later in the pipeline to improve previous results.

Another weakness of the current implementation is that it does not have a re-ranker. Although the current model is simple in its design, we believe its effectiveness could benefit from additional re-ranking stages. Re-rankers can considerably improve the effectiveness of existing retrieval models by taking the top-*k* documents and re-ranking them, usually by looking at information ignored by previous ranking stages. By looking at fewer documents than the steps before it, re-rankers can afford to perform more complex computations that would be too costly to run across larger portions of the collection. As stated previously, the re-ranker could account for the lack of contextual analysis in the baseline model. Regardless, the effectiveness of the true re-ranker would be dependent completely on the first-stage ranker's ability to find relevant documents within the top-*k*.

A technique that could help increase the number of relevant documents in the top-1000 would be query expansion. The baseline model does utilize a Porter Stemmer by default when tokenizing, but its inclusion has the potential to provide as many issues as benefits. The Porter Stemmer is known to improve recall[8], but could reduce precision by introducing terms that are not related to the user's information need. Otherwise, the baseline model does not use any other query expansion techniques. We believe that after adding position data to the index, there would be an option to use that information as part of a query expansion step to potentially improve the baseline model's effectiveness.

3. Text, Formula, and Query Processing

Having identified key weaknesses within the baseline BM25 implementation, we seek a method which would improve retrieval pre-processing. Out of the flaws we found with the baseline's pre-processing, we wanted to investigate how the inclusion of query expansion would impact the baseline retrieval's effectiveness. As briefly mentioned before, the baseline implementation only uses a standard inverted index for ranking, which contains plain-text keywords and term frequencies per document as postings. As seen in Section 2, using the current index to rank ARQMath answer posts is not very effective. We believe that allowing the retrieval model to analyze words used in proximity to each other would help improve its effectiveness. Specifically, we believe there is an opportunity to capture more information in the index that would help distinguish relevant answers from the rest of the collection. We propose an experiment which adds word position data to the index and applies the Sequential Dependence Model of query expansion to the base pipeline. We expected that with these additions, the baseline model's retrieval effectiveness would improve (exemplified through increased nDCG', P'@10, and mAP' scores).

The ultimate goal of this experiment was to see the effects of query expansion via Sequential Dependence on retrieval effectiveness. The Sequential Dependence Model (SDM) [7] was introduced by Metzler and Croft and used to model term dependencies of adjacent query terms. In PyTerrier's implementation, there are three ways in which the rewritten query is used to score documents. The first matches query bi-grams for exact matches within documents. The second looks for adjacent query terms found in unordered windows of size eight in documents. The last

Table 2
Results of the baseline model coupled with Sequential Dependence query expansion run on all ARQMath-1 topics, alongside the baseline BM25 system. Effectiveness measured with nDCG', P'@10, and mAP'

Name	nDCG'	P@10'	mAP'	MRT
BM25	0.1054	0.0701	0.0435	3019.19
SDM-BM25	0.0881	0.0675	0.0372	23158.35

Table 3
Results of the baseline model coupled with Sequential Dependence query expansion run on all ARQMath-2 topics, alongside the baseline BM25 system. Effectiveness measured with nDCG', P'@10, and mAP'

Name	nDCG'	P@10'	mAP'	MRT
BM25	0.0783	0.0282	0.0166	3248.79
SDM-BM25	0.0541	0.0239	0.0111	20584.70

looks for all query terms found in unordered windows of size twelve in documents (weighted lower than the other methods). We believed that by allowing the retrieval model to boost the ranking of documents which have query terms appearing close to each other, the retrieval model would recognize document contexts and, we hope, improve retrieval effectiveness.

First, we generated a new index for the ARQMath Collection so that it would contain word position data. We then did some testing to understand how SDM utilized Terrier's underlying query language, the Match Op Query Language[9]. After a few test queries, we noticed that queries containing Unicode characters would crash when passed through the SDM transformer. We had to implement additional translations to prevent these problems from occurring in our actual experiments. After accounting for these issues, we run our new retrieval pipeline (the SDM query expansion transformer followed by the BM25 baseline retrieval model) alongside the baseline model on all topics for ARQMath-1 and then for ARQMath-2.

The results of this experiment (seen in Tables 2 and 3) indicate that our hypothesis was incorrect. Looking at our results, all the effectiveness metrics for our improved pre-processing pipeline demonstrate a decrease in performance compared to the baseline when querying on ARQMath-1 and ARQMath-2 topics. Additionally, we observe a large decrease in efficiency compared to the baseline model. Although response time was not a metric that we were originally interested in optimizing, we found it pertinent to investigate the possible causes of both decreased efficiency and effectiveness.

We believe that the worse runtime can be attributed to how the baseline model and SDM combine to accentuate the length of ARQMath's question posts. ARQMath Task 1 queries are often quite long by themselves. For example, an ARQMath-1 query reads: "Suppose that $f : [a, b] \longrightarrow R$ is continuous and that $f([a, b]) \subset [a, b]$. Prove that there exists a point $c \in [a, b]$ satisfying $f(c) = c$." As mentioned in Section 2, the baseline model translates punctuation to plain text, usually a single term per punctuation. This will bloat the query with more terms. After this, SDM will expand the query even more in an effort to find adjacent query terms both in ordered pairs and in unordered windows using the index. The index would have to be checked against all these word pairs, and this would increase the runtime dramatically, especially for longer queries.

In regards to the lower effectiveness metrics, we believe this can be attributed to the nature of analyzing the proximity of math formula terms in documents using wide proximity windows. Before formulating our experiment, we overlooked the importance of the order of terms in mathematical formulae. For example, one excerpt from a translated ARQMath query reads:

> "open brace n backslash log underscore open brace 3 close brace n close brace n close brace backslash right close parenthesis question."

With this query, our improved pipeline could find and give similar scores to documents that contain sequences such as 'open brace 3 close brace n', 'open brace n backslash log', and 'log underscore open brace 3 close brace', all representing different meanings within the context of their original posts. Because of this behavior, looking for unordered co-occurrences of terms in sliding windows within answer posts can be detrimental when attempting to distinguish posts which are relevant to the given query.

4. Result Re-ranking

Looking at the current method in which BM25 is implemented, one aspect it lacks is result re-ranking. As discussed above, there are pre-processing steps which can be added before BM25 to help enhance the input data to the scoring algorithm. In a similar manner, the output can be further processed to correlate the retrieved documents with the query text.

With this in mind, we propose an experiment to evaluate neural re-ranking pipelines. To gain more insight into the most effective model, we plan to leverage the KNRM neural model and compare two of its vocabulary options. Additionally, we expect that the neural models will produce better effectiveness metrics due to their deeper contextualization of language. All models will be trained on the ARQMath-1 (2020) dataset. For evaluation, all models will run on all ARQMath-2 (2021) topics.

4.1. KNRM

The KNRM utilizes a translation matrix to combine the query and document terms which is passed into a set of kernels to learn the term weights. Since this model will be trained on the ARQMath dataset, it will be able to gain deeper contextual connections between the query terms and the content of the document. This is crucial for a dataset like ARQMath which does not follow normal language model conventions. The ability to adapt to the corpus of the dataset is a large advantage over using a model which was pre-trained on a dataset like MS-MARCO.

The corpus of ARQMath is also a benefit for this re-ranking method as math topics use similar symbols and terminology; at least compared to something like MS-MARCO which needs to understand millions of documents. The benefit of the neural model in this case is that it does not focus on individual works but instead the connection between words. This is crucial as the LaTeX translation splits symbols into multiple English terms. If a model were to only look term-at-a-time, it would miss out on obvious connections and typical usage patterns within the corpus. Thus, the KNRM model is a great fit for this task.

4.2. Experimental Design

To perform this experiment, there are three pipelines created in parallel. One is the normal BM25 run that passes through the prime transformer. Two instances of KNRM are created; one with the BERT vocabulary and one with the WordVec_Hash vocabulary. The BM25 pipeline is then piped into both of these models to create their separate experiments. Using the ARQMath-1 dataset, both of these models are asked to train and then validate using the ARQMath-2 validation topics. This helps the neural models to generalize content from the respective documents, as they have to evaluate both content it has seen and content it has not seen before.

4.3. Vocabularies

These two vocabularies are intriguing to evaluate with this experimental setup. The WordVec_Hash vocabulary model is based on Google's Word2Vec. This is intriguing because it can strongly generalize words and help us understand which terms are similar or related to it. An example from Google's documentation demonstrates this ability:

vector('king') - vector('man') + vector('woman') is close to vector('queen')[10]

This is precisely in line with the processes desired to comprehend the math language and will be an interesting point of comparison with the BM25 baseline. The second vocabulary is based on BERT and attempts to break sentences and words down into subterms and use those to understand what parts of a document are most important. It is also very flexible and only requires a bit of training to adapt to a new corpus:

...a pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks.[11]

This also appears to be a great match for the task at hand, though potentially not as strong as the WordVec_Hash vocabulary. Since BERT splits document terms into subterms, it may over-generalize the meaning and underestimate the value of the terms.

4.4. Re-Ranking Results

Number	Name	nDCG'	P'@10	mAP'	MRT
0	BM25	0.0780	0.0281	0.0164	842.49
1	BM25 » KNRM-BERT	0.0772	0.0309	0.0173	8451.29
2	BM25 » KNRM-WORDVEC-HASH	0.0801	0.0394	0.0182	1031.64

4.5. Result Discussion

Looking at the results, we do see improvements from both KNRM neural models re-ranking the baseline BM25 run. As anticipated, the results using the BERT vocabulary are better than the basic BM25 model, but it does not perform as well as WordVec_Hash. This does make sense, as the BERT methodology is extremely fine-grain in the way it processes and breaks down the text to comprehend and contextualize it. WordVec_Hash on the other hand, takes a 'bigger picture' approach to training its model, and it benefits from focusing more on how document and query terms relate to one another. The most significant metric from the results is the P'@10 of WordVec_Hash, as it increased the most out of any of the runs conducted in all of our experiments. It is also interesting to see the MRT of the BERT vocabulary being a nearly 10-fold increase in duration, due to the significant number of iterations required for KNRM to break down each of the document terms and try to understand its context. These are still not perfect results; these two approaches are after all aimed at natural language processing. This corpus is far from natural language, so training can only go so far. Especially when you consider KNRM is re-ranking the output of BM25; they have to work with the results they are given after that it has concluded its own processing. Work would need to be done earlier in the processing pipeline to allow for larger improvements at this stage.

5. Conclusion

In this paper, we walked through our PyTerrier implementation of an information retrieval system based around the ARQMath collection. Provided with the baseline BM25 engine, we made additions that we hypothesized to improve the system through the measures of nDCG', P'@10, mAP' and MRT. Our first addition came from the realization that the BM25 baseline did not handle any query expansion and only considers term frequency. Because of this we decided it would be beneficial to implement Sequential Dependence query expansion which would cause the retrieval system to score documents higher for containing query terms in close proximity. Doing so resulted in considerably lower values for nDCG', P'@10, and mAP' and a much higher response time. We determined the decreased efficiency of our pre-processing pipeline was due to how the already long ARQMath queries were expanded upon to create very long runtime operations, and we determined the lower effectiveness metrics was caused by the contextual analysis of our query expansion not being strict enough to handle the structure of math formulae. We additionally implemented the neural model KNRM as a re-ranker, which we thought would greatly boost our effectiveness metrics. We measured the results of both KNRM's Word-Vec_Hash and BERT-based vocabulary; the Word-Vec_Hash vocabulary had the best measurements in every category, with BERT coming close in second. Both were able to slightly outperform BM25 in both effectiveness.

If we were to continue this experiment, there are a few adjustments that we would make to attempt a more successful outcome. For our pre-processing experiment, we would be interested in experimenting with size of the sliding windows implemented by SDM, along with whether the terms in the windows should be ordered. We propose that using am exhaustive grid search to find the best window size for viewing term proximity would heighten our effectiveness results. We would also make some adjustments to the neural re-ranker by training it at different depths and comparing its performance with other neural re-rankers to see which one handles the ARQMath Collection best.

References

[1] I. O. Akiko Aizawa, Michael Kohlhase, Ntcir-10 math pilot task overview, 2013. URL: https://research.nii.ac.jp/ntcir/workshop/OnlineProceedings10/pdf/NTCIR/MATH/04-NTCIR10-MATH-KohlhaseM_intro.pdf.

[2] M. K. I. O. G. T. K. D. Richard Zanibbi, Akiko Aizawa, Ntcir-12 mathir task overview, 2015. URL: https://www.cs.rit.edu/~rlaz/files/ntcir12-mathir.pdf.

[3] D. O. R. Z. Behrooz Mansouri1, Anurag Agarwal1, Finding old answers to new math questions: The arqmath lab at clef 2020, 2020. URL: https://www.cs.rit.edu/~rlaz/files/ARQMATH_Lab_overview_.pdf.

[4] C. Macdonald, N. Tonellotto, Declarative experimentation in information retrieval using pyterrier, 2020. URL: https://pyterrier.readthedocs.io/en/latest/index.html.

[5] R. Z. Behrooz Mansouri1, pt-arqmath, 2022. URL: https://gitlab.com/dprl/pt-arqmath.

[6] J. L. Rodrigo Nogueira, Document expansion by query prediction, 2019. URL: https://github.com/castorini/docTTTTTquery.

[7] W. B. C. Donald Metzler, A markov random field model for term dependencies, 2005. URL: http://web.cs.ucla.edu/~yzsun/classes/2014Spring_CS7280/Papers/Probabilistic_Models/A%20Markov%20Random%20Field%20Model%20for%20Term%20Dependencies.pdf.

[8] M. Porter, The porter stemming algorithm, 2006. URL: https://tartarus.org/martin/PorterStemmer/.

[9] R. M. Craig Macdonald, Query language, 2020. URL: https://github.com/terrier-org/terrier-core/blob/5.x/doc/querylanguage.md.

[10] Google code archive - word2vec, ???? URL: https://code.google.com/archive/p/word2vec/.

[11] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, arXiv, 2018. URL: https://arxiv.org/abs/1810.04805. doi:10.48550/ARXIV.1810.04805.