Shanks Touché Homework

Filippo Berno[a], Andrea Cassetta[a], Alice Codogno[a], Enrico Vicentini[a] and Alberto Piva[a]

[a]*University of Padua, Italy*

Abstract

This paper is the report of the work done for Argument Retrieval CLEF 2021 Touché Task 1 by Shanks team (based in Italy and precisely the members are University's of Padua students). Argument Retrieval CLEF 2021 Touché Task 1 focuses on the problem of retrieving relevant arguments for a given controversial topic, from a focused crawl of online debate portals. After some tests Shanks group has decided to parse the input documents taking only the title, premises and conclusion of the arguments (as well as the stance necessary to understand the arguments' author point of view). After the indexing part of the documents the work is concentrate on how the retrieving and the raking are done. After some tests, we discover that the better results are obtained using a WordNet [1] based query expansion approach and a re-ranking process with two different similarity functions. This report describes in details how the documents parsing work and how the indexing and searching part are developed. The unexpected update of the qrels file did not allow us to re-run all the tests. In the end, however, we also reported the results of the runs obtained from parameter tuning on the new qrels.

Keywords

Argument Retrieval CLEF 2021 Touché Task 1, WordNet synonyms, Re-ranking, BM25, DirichletLM

1. Introduction

In this report, we describe the project developed for the participation by the Shanks group to the CLEF 2021 Touché Task 1. The task focuses on the problem of retrieving relevant arguments for a given controversial topic, from a focused crawl of online debate portals.

Our goal is to develop a Java based information retrieval system that finds and ranks the relevant documents from the args.me corpus dataset composed by over 380.000 arguments crawled from 5 different debate forums [2] for 50 topics (query). The retrieved results need to be relevant for each input topic the system has to elaborate.

This paper is structured as follow: Section 3 is about the solutions that we've taken in consideration to build the retrieval system, Section 4 describes the whole workflow of the program; Section 5 explains our experimental setup including the software, tools and methods used; Section 6 discusses results ; and finally, Section 7 draws some conclusions and outlooks for future work.

"Search Engines", course at the master degree in "Computer Engineering", Department of Information Engineering, University of Padua, Italy. Academic Year 2020/21

filippo.berno@studenti.unipd.it (F. Berno); andrea.cassetta@studenti.unipd.it (A. Cassetta); alice.codogno@studenti.unipd.it (A. Codogno); enrico.vicentini.1@studenti.unipd.it (E. Vicentini); alberto.piva.8@studenti.unipd.it (A. Piva)

2. Related Works

To create our search engine we build upon some source code created by professor Nicola Ferro to show as some toy examples and changed as described in the subsequent sections. We have also read the overview of CLEF 2020 on the Touchè task[3].

3. Initial Attempts

Before going into the details of our final solution, it is useful to describe previous approaches we took into account to solve the problem and why we chose not to explore them further.

3.1. Parsing Documents

Multiple parsers have been developed to parse the documents from the provided collection. The most trivial parser, called $P1$, extracts the *sourceText* and *discussionTitle* elements from the corpus documents. The second parser $P2$ extracts the elements related to the conclusion, the premise, the discussion title, and all the text from the *sourceText* field in between the premise and the conclusion. The third parser $P0$, which at the end we decided to use for our more advanced experiments, extracts only the discussion title, the premise and the conclusion for each document. Table 1 shows how the index statistics are affected by each of these parser.

Table 1
Statistics regarding three indexes generated using the three implemented document parser. The analyzer is always the same. Time Ratio is obtained by dividing the time taken by each parser by the time taken by the fastest parser.

Parser	Term Count	Storage (MB)	Time (seconds)	Time ratio
P0	1078017	195	99	1,00
P1	1196289	1745	507	5,12
P2	1153517	706	249	2,52

3.2. Query Expansion

When we have developed the software to create the index, we have deeply thought about how we could have used the resulting tokens from the analysis of the topic query.

3.2.1. OpenAI GPT-2

In the attempt of expanding the queries we came across the OpenAI GPT-2 model, a Machine Learning algorithm that generates synthetic text samples from an arbitrary input.

The idea was to use this powerful algorithm to make a query expansion, giving as input the topic title to generate a more complete phrase with hopefully new words that could help the searching part. Unfortunately the output of GPT-2 is not always what we expect. For example if we give it as input the tokenized query title, that could be only made of 2 words, the output is a not very useful dialog for our task. Another problem is the structure of the query, in fact since they are all questions, the GPT-2 algorithm generates an answer for them which still is

not what we were interested in. The problem persists also if we remove the question mark at the end of the phrase. In fact, the queries still have a question structure. For these reasons, we have decided to set aside this kind of approach.

3.2.2. Randomly Weighted Synonyms

An approach initially devised for query expansion, but which we later decided not to explore further, was to generate multiple queries for the same topic, each with randomly generated synonym boost values. For each query, the rankings of 1000 documents were then generated, and finally all the rankings were merged into one. The first performances obtained by this method did not encourage us to proceed with the development because there were many possible paths to follow from that point and the search time increased considerably.

3.3. Minimum body length

During the process of documents exploration, useful to detect which field are needed and present in each collection, we notice that in some documents the *sourceText* field were constituted by useless text without a single relevant information about its topic. In order to avoid this kind of documents ending up in the inverted file, we have tried to include, during the indexing process, a check on the length of the *ParsedDocument*'s body. If this field, the one that we have considered as the union of the *conclusion* and *premise* fields, is made up of less than a certain number of token, recurrent aspect that the parsing phase highlights for such instance, we avoid to consider them in the indexing phase.
After doing some tests with different values for the "min body length" (5, 10, 15), we have compared the results of this kind of solution with the results obtained without using it and we have discovered that, taking as example "min body length" equal to 10, the number of retrieved document switches from 48781 to 48764 and the number of relevant document switches from 1263 to 1257 (the other evaluation measures are no so affected by this change).
Considering this result we have decided not to use this kind of document pre-processing to avoid the discarding of some document that could be considered relevant in the qrels file (and so for an user) even if they don't seem like it.

3.4. Re-ranking with discussion ID

One of our primary goals is to improve the final ranking of the documents. With this assumption we tried to improve performance by using re-ranking.

As a first analysis we observed that, in the documents of the dataset, the posts related to the same discussion had the same first part of the document ID. Based on the assumption that only posts from some discussion are relevant to a query, we tried to index those posts as a single document. We finally obtained an index with a discussion-based clustering of documents. In the searching phase, we firstly searched the query in the normal index, retrieving the classic ranking of single documents. Secondly we searched the same query in the second index of discussion clusters, thus obtaining a ranking of discussions. Finally the scores of documents in the first ranking were increased based on the rank of their respective discussion in the second ranking. Unfortunately this approach did not provide the desired results because it assumes

that all posts related to a discussion have the same relevance to the searched topic. In fact, it was found that some posts do not contain any useful information to argue the searched topic but are part of a discussion that is really relevant.

3.5. OpenNLP attempt

Exploring new solutions, we have even tried to implement a version of the program that uses the OpenNLP Machine Learning toolkit in order to see which advantages a tokenization able to distinguish location, personal nouns and so on could provide for the solution.
Following this path we have encountered an error which requires significant changes in the workflow of what we had done up to that point. For this reason we have decided not to keep going on with this branch.

4. Methodology

The goal of this task is to retrieve relevant arguments from online debate portals, given a query on a controversial topic. We have checked the dataset and we have noticed that it is composed by five JSON files. We have also read some documents and we have noticed that the main structure is the same for each one but with some different fields. To use the documents with *Lucene* we had to parse the documents. To do so, we have used the *Jackson library* and we have implemented our parser *P*0 that takes the premises, conclusions, the document title and the stance attribute (PRO or CONS) of the documents.

4.1. Indexing

We have built four different parts starting from the *Lucene* default ones: *ArgsParser*, *ShanksAnalyzer*, *DirectoryIndexer* and the *Searcher*. There is indeed a *ShanksTouche* class which has the main method and allows us to setup parameters for indexing and analyzing parts. When the main method is started, by default, it starts calling the *DirectoryIndexer*'s index method. Inside this, *ArgsParser* is called. It also exploits the *ShanksAnalyzer*. After converting the documents into something that *Lucene* can work on, we focused ourselves on the development of how the indexer is created, in particular on how the analyzer module works. Indeed, it is used both for the indexing phase and search phase. The tokenization starts with a *StandardTokenizer* and then reducing every word to lower case by using a *LowerCaseFilter*. Next we have applied the *EnglishPossessiveFilter* that removes the "'s" particle from the remaining tokens. Finally we have used the stop-words filter, *StopFilter*, that removes words that are frequent in the whole collection that do not bring useful information.

The arguments of the collection are stored in the index using four fields:

- ID, the same ID found for the argument;
- Title, of the argument that can be the "Discussion title" or the "Topic" depending on the source of the argument;

- Body, where we store the premise and conclusion of the argument;
- Stance, which can be PRO or CON.

4.1.1. Custom Stop-List

We have decided to use a custom stop-list to better achieve the project goal. Our stop-list contains 1362 words that are derived from the merging of other stop-lists (e.g. smart and lucene) typically used. To populate our stop-list, we also added the most commonly used terms within the documents in the collection. For example, we have added the words: "http", "people", "debate", "round" etc. Our custom stop-list reduces memory usage by approximately 38% and indexing time by almost 20%.

4.2. Searching

In the searching phase we spent most of our time trying to improve the general quality of the results, experimenting with different strategies like query expansion based on *WordNet* synonyms or defining queries to score differently the fields of the documents. The approach we have chosen to perform involves the use of *BooleanQuery*. In this way it is possible to assign specific weights to every term of the query (boosts).

The search query is made of three main clauses. The first one, called *ttQ*, is for searching the topic title in the document title, it is boosted by a user defined parameter called *tBoost*. The second one, called *tbQ*, searches the topic title inside the document body. The third one, called *dbQ*, (which is optional) is used to search the topic description in the document body. For each of the three, the *WordNet* synonyms are boosted with the parameter *sBoost*. In addition to terms, in *ttQ* and in *tbQ*, we also add proximity searches made of all possible pairs of terms contained in the topic title. These proximity searches has a boost defined by the parameter *pBoost*, the proximity distance is given by the parameter *pDist*.

Here follows an example of a query without topic description where $tBoost = 3.5$, $sBoost = 0.0$, $pBoost = 1.75$, $pDist = 15$:

```
((title:insider title:trading title:allowed)^3.5
(title:"insider trading"~15 title:"insider allowed"~15 title:"trading allowed"~15)^1.75
(body:insider body:trading body:allowed)^1.0
(body:"insider trading"~15 body:"insider allowed"~15 body:"trading allowed"~15)^1.75)
```

To provide a more readable and functional implementation there's only one method that takes care of creating the query with the desired parameters. The method's signature is the following:

```
public Query buildQuery(String[] terms, String fieldName,
                            float sBoost, float tBoost, float pBoost, int pDist)
```

Finally, we decided to use and test both *BM25Similarity* and *LMDirichletSimilarity* similarities and then their combination with a *MultiSimilarity* to get the document scores.

4.3. Re-Ranking

During various experiments we have noticed that some measures of similarity and set of parameters favored the precision at the expense of recall and vice versa. With the purpose of combining advantages of both cases, we opted for a re-reranking method which exploits different similarities and query parameters. Our implementation is made of two steps. In the first one we use a query able to obtain a higher recall value when searching the index for relevant documents, whose parameters and similarity are decided based on empirical trials. In the following step we use a second query with better performance in terms of $ndcg$ to re-evaluate the returned documents and so re-ranking them according to the new score. We call *maxRecall* the first query and *maxNdcg* the second one. According to our implementation, this approach turns out to be effective on the 2020 topic set, but at the expense of the time spent in the search phase, which increases quite substantially.

5. Experimental Setup

Our work is based on the following experimental setups:

- Repository: https://bitbucket.org/upd-dei-stud-prj/seupd2021-goldr;
- During the develop and the experimentation we have used our own computer and in the end we have run our code using *Tira*;
- Evaluation measure: BM25, LMDirichlet and a "MULTI" where both were combined;
- *Apache Maven*, *Lucene*;
- Java JDK version 11;
- Version control system *git*.
- $trec\_eval$ tool [4]

The collection is a set of 387,740 arguments crawled from *debatewise.org*, *idebate.org*, *debatepedia.org*, *debate.org* and 48 arguments from Canadian parliament discussions. CLEF provided us with the 50 topics given in the 2020 edition of the contest to train and refine our search engine, topics with a title, a description and the narrative. This last part was not taken into account because it is used by the assessor to judge the relevance of the documents retrieved. Furthermore we developed the source code collaborating through the BitBucket platform.

6. Results and Discussion

In this section we provide graphical and numerical results about the experiments we conducted during the development of the project. We also discuss these results to derive some useful insights.

6.1. Our Baseline

In order to be able to track performance progress during development, we created our own baseline. For each of the three parsers $P0$, $P1$, $P2$ we produced a run using a simple analyzer that uses a *StandardTokenizer*, *LowerCaseFilter* and a *StopFilter* where the stop-list used is the standard one offered by *Lucene*. The similarity used is BM25. Figure 1 compares the three baselines. From these results we decided to develop our approach based on the $P0$ parser. Figures 2 and 3 show how $P0$ compares to the other parser evaluating performance per topic. The choice of $P0$ was motivated not only by the statistics in Table 1 but also by its overall efficacy as shown in the following figures. Figure 1 shows how differently the three approaches can behave. $P0$, that extracts only the discussion title, the premise and the conclusion for each document (with its stance), highlights better performance with respect to the other two retrieving more relevant document across the entire run. Obviously we chose to discard $P1$ because his performances were inferior to the other two as shown in Figure 2. To better understand the behavior of $P2$ versus to $P0$, Figure 3 compare the *per topic average precision*. Approximately 10% of the topic, in particular topic 3, 23, 34 ad 43 to 46 tend to give nicer results with parser $P2$.

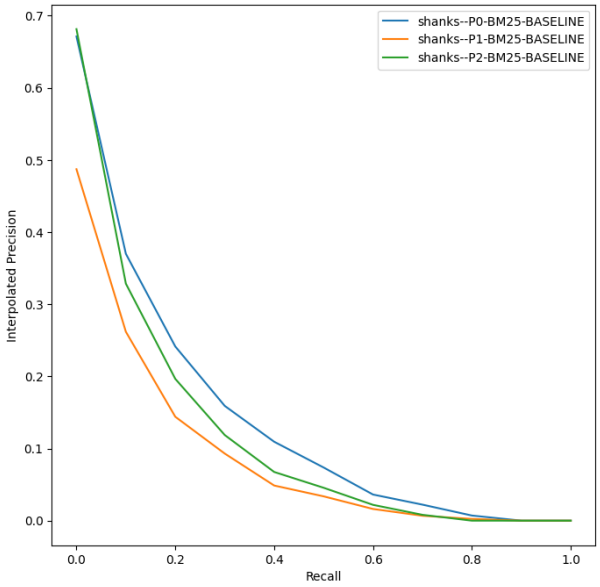

Figure 1: Plot of the Interpolated precision against recall of the three baselines.

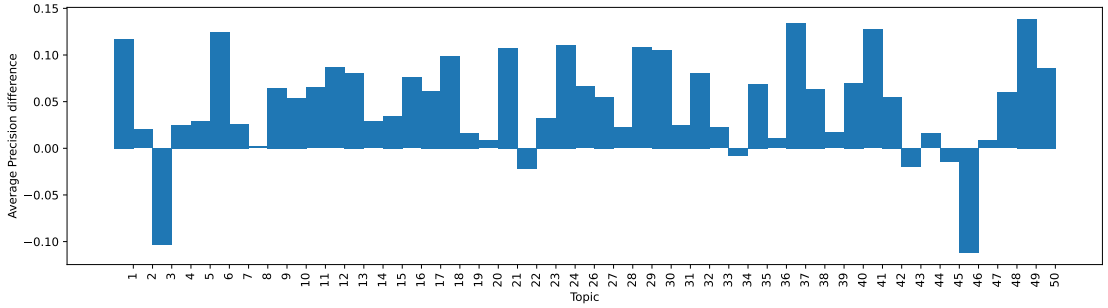

Figure 2: Per topic Average Precision Difference = $AP_{P0} - AP_{P1}$.

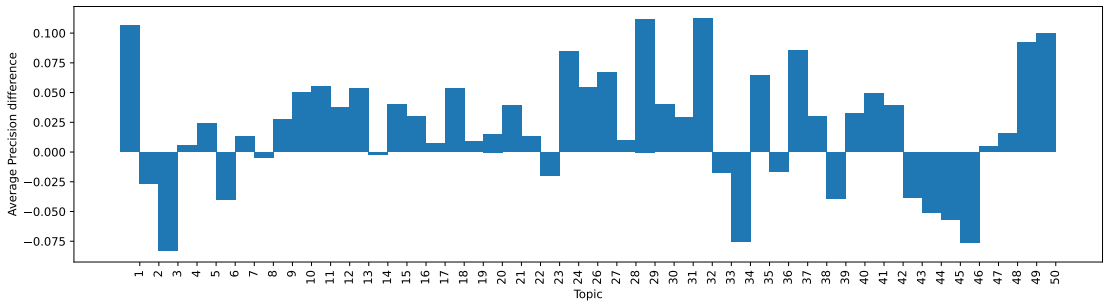

Figure 3: Per topic Average Precision Difference = $AP_{P0} - AP_{P2}$.

6.1.1. Baseline with the custom stop-list

The results obtained by our custom stop list, compared to the one offered by Lucene are comparable.

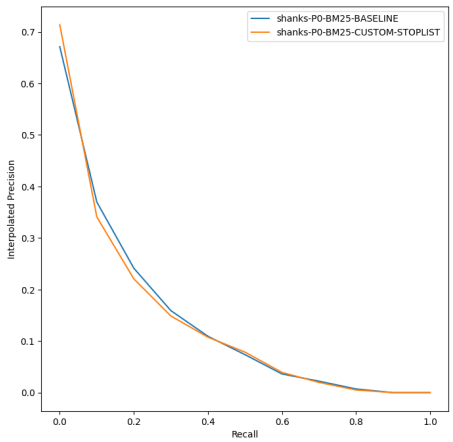

Figure 4: Plot of the Interpolated precision against recall of the two P0 baselines obtained by the Lucene stop-list and the custom stop-list.

6.2. Parameters Tuning

Our ultimate goal is to maximize the average value of *nDGC*@5 across all topics. To find the optimal parameter values, we performed extensive iterative tests, trying all combinations of parameters with values belonging to discrete intervals we defined. The same experiments were conducted using three different similarities : *BM25Similarity* (BM25), *LMDirichletSimilarity* (LMD), *MultiSimilarity* (MULTI). The MultiSimilarity we have used, combines BM25Similarity and LMDirichletSimilarity. The method we developed is governed by five parameters, when using re-ranking they become ten. Out of those ten, five parameters concern the query search in the index which aims to maximize the overall recall, the other five affect the re-ranking process and are chosen to maximize *nDGC*@5.

6.3. Optimal parameters

The optimal set of parameters we found for the *maxRecall* query and for the *maxnDCG* query are available in Tables 2 and 3. The best similarity measure to maximize *nDGC*@5, according to our empirical tests is LMD. To maximize recall, on the other hand, the best measure is MULTI. As it can be seen in Table 2, the best value of *tBoost* for *maxnDCG* is 0. This allowed us to come to the conclusion that considering the title of the discussion (which is the same for all posts in it) can be misleading with respect to the relevance of the content of each post that is part of it. The title, however, is very useful to obtain higher recall. This intuition is what pushed us to abandon the method of re-ranking based on discussion ID, which is described in 3.4.

Table 2
The optimal parameters obtained by training on the 2020 topics (topic description not considered).

Query	Similarity	tBoost	sBoost	pBoost	pDist
maxRecall	MULTI	3,50	0,15	1,75	12
maxnDCG	LMD	0,00	0,05	0,75	15

Table 3
The optimal parameters obtained by training on the 2020 topics (considering the topic description).

Query	Similarity	tBoost	sBoost	pBoost	pDist
maxRecall	MULTI	3,50	0,15	1,75	12
maxnDCG	LMD	0,30	0,05	1,75	15

6.4. maxnDCG and maxRecall

In this section we want to compare the two queries *maxnDCG* and *maxRecall* with the optimal parameter values established by the tests. From the graph in Figure 5 we can see that the precision is significantly higher for *maxnDCG*, while *maxRecall* has a better recall on the whole ranking. From these data we believe to have obtained the desired result from the two queries. In Figures 6, 7, 8 we compare the Average Precision per topic for the three test runs. We can notice how there is not much difference between maxRecall and P0. The same is not true for *maxnDCG*, the latter proves to be better than P0 in almost all topics. This scenario is further

confirmed by Figure 8, which shows the dominance of $AP_{maxnDCG}$ over $AP_{maxRecall}$. The performance can be further compared with the numerical results reported in Table **??**. From these data we can realize the actual trade-off between the two approaches. The advantage in terms of recall for *maxRecall* is significant compared to *maxnDCG*. The same advantage applies in the opposite way for precision and*nDCG*.

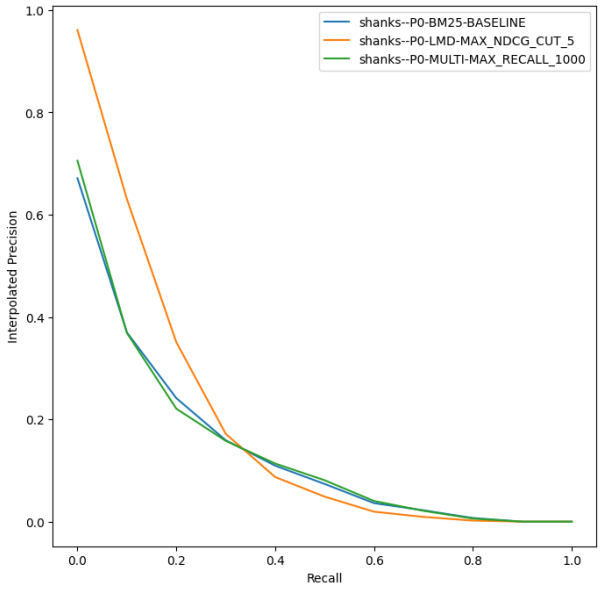

Figure 5: Plot of the Interpolated precision against recall for the *maxnDCG* query versus *maxRecall*.

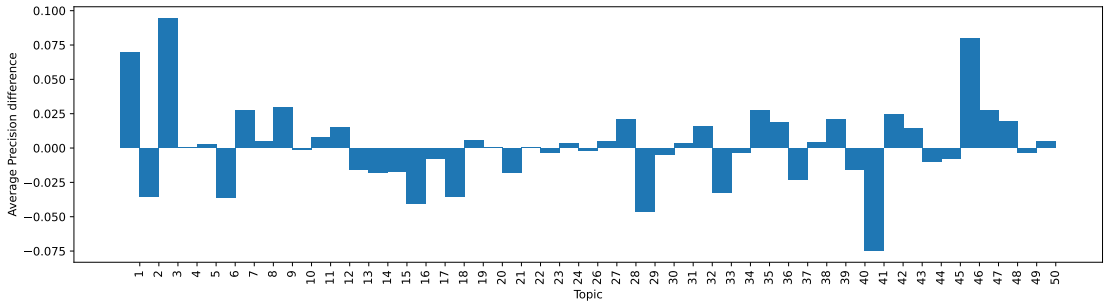

Figure 6: Per topic Average Precision Difference = $AP_{maxRecall} - AP_{P0}$.

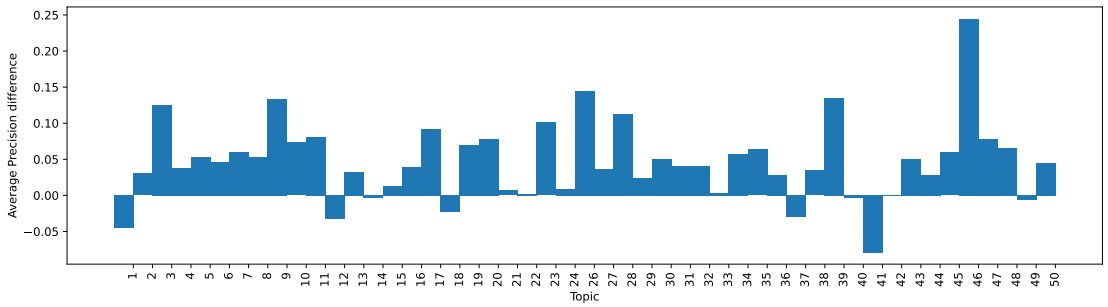

Figure 7: Per topic Average Precision Difference = $AP_{maxnDCG} - AP_{P0}$.

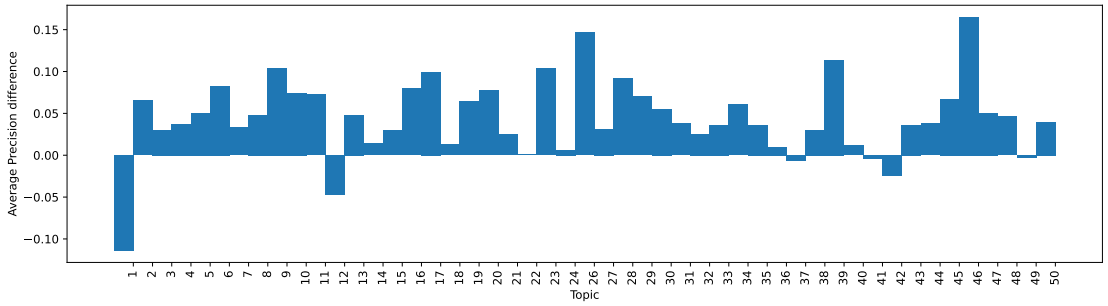

Figure 8: Per topic Average Precision Difference = $AP_{maxnDCG} - AP_{maxRecall}$.

6.5. Re-Ranking Results

Here we compare the *Re-Ranking* approach described in 4.3, with the previous ones. Figure 9 shows how re-ranking based on *maxNdcg* (in red), greatly improves the interpolated precision of *maxRecall* (in green). It is also possible to note that it allows for better performance with respect to *maxNdcg* alone (in orange). Figure 10 shows the Average Precision value obtained by re-ranking for each topic. From it, it is possible to identify the most problematic topics, for which the method developed is not very effective. In particular, the most critical topics are: $\{2, 8, 22, 40, 44\}$. From Figure 11 we can see that the *Re-Ranking* method improves on almost every topic w.r.t the baseline. In Figure 12 it can be seen that the *Re-Ranking* method improves on *maxNdcg* on many topics, except for topic 10 and 20. Figure 13, as predictable, clearly shows the better performance achieved by *Re-Ranking* w.r.t. *maxRecall*, only the first topic is penalized by the re-ranking process.

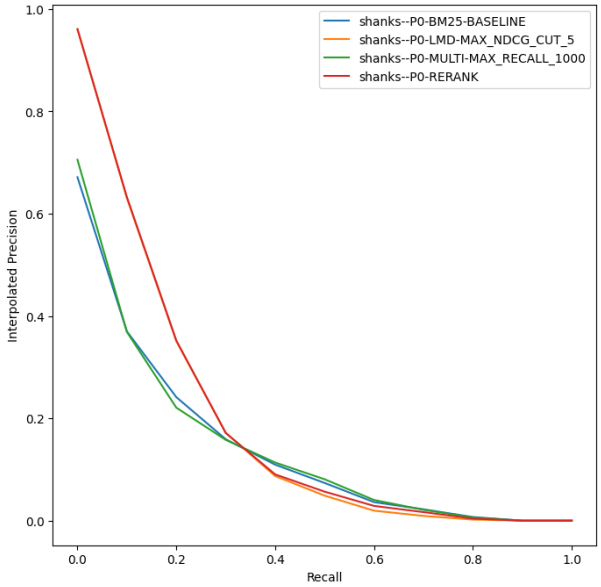

Figure 9: Plot of the Interpolated precision against recall comparing baseline, *maxRecall*, *maxnDCG*, *Re-Ranking*

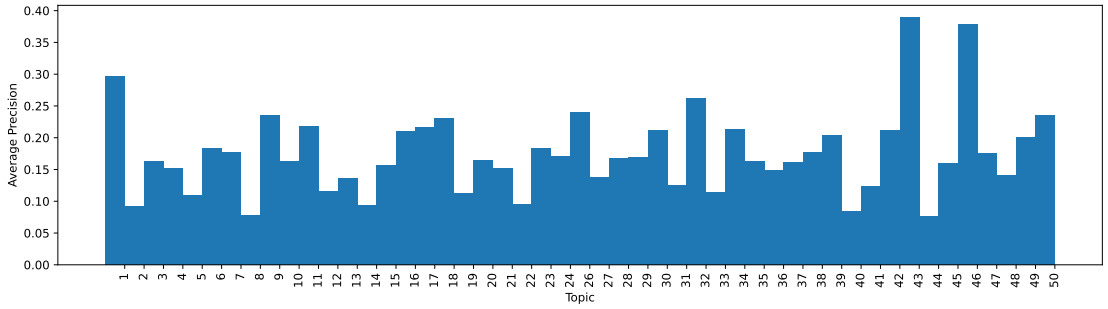

Figure 10: Per topic Average Precision for the Re-Ranking approach.

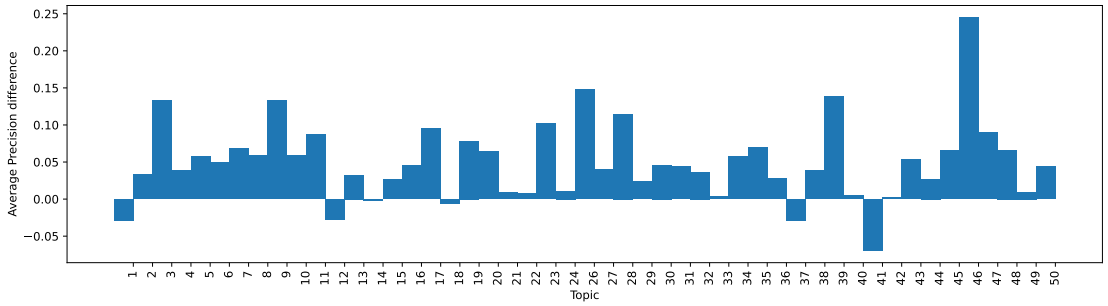

Figure 11: Per topic Average Precision Difference = $AP_{re-ranking} - AP_{P0}$.

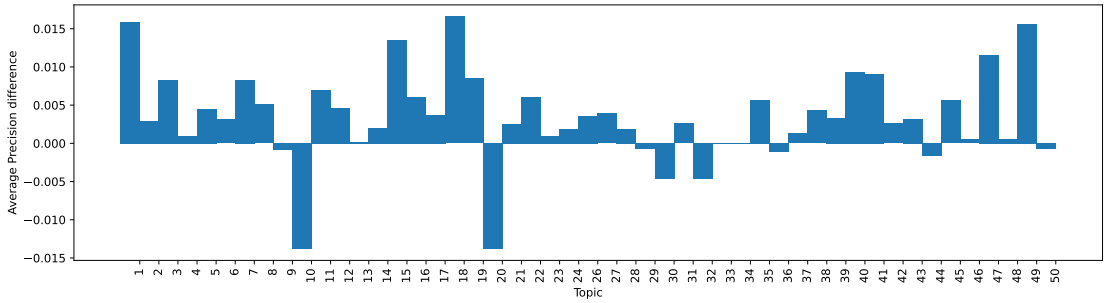

Figure 12: Per topic Average Precision Difference = $AP_{re-ranking} - AP_{maxnDCG}$.

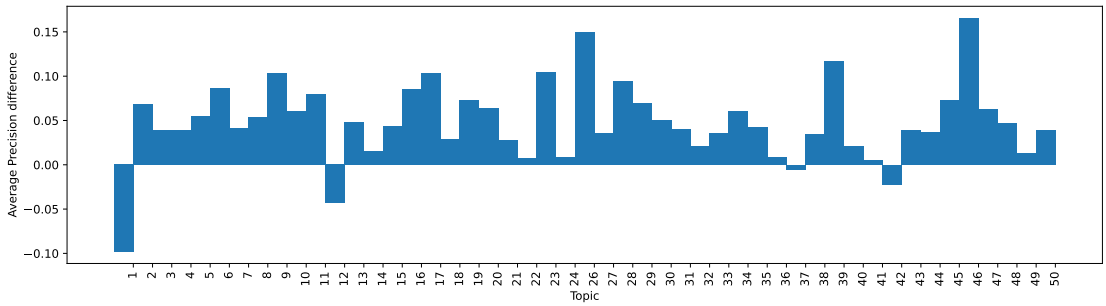

Figure 13: Per topic Average Precision Difference = $AP_{re-ranking} - AP_{maxRecall}$.

Table 4
Some numerical results for comparing performance of *Re-Ranking*, *maxnDCG*, *maxRecall* and the baseline.

RUN	num_rel_ret	map	P_5	recall_1000	nDCG	nDCG@5
P0-RERANK	1263	0.1750	0.6490	0.6631	0.4979	0.5495
P0-LMD-MAX_nDCG@5	1103	0.1717	0.6490	0.5803	0.4764	0.5495
P0-MULTI-MAX_RECALL_1000	1263	0.1276	0.4082	0.6631	0.4283	0.3117
P0-BM25-BASELINE	1250	0.1256	0.3796	0.6563	0.4216	0.2871

6.6. RUN Submission

The five run we decided to submit are the following:

- run-1 : *Re-Ranking* approach.
- run-2 : like run-1 but proximity searches areonly with pairs of subsequent tokens.
- run-3 : *maxnDCG* query with LMDirichletSimilarity
- run-4 : *maxnDCG* query with MultiSimilarity
- run-5 : *maxRecall* query with MultiSimilarity

Table 5 shows the numerical results we obtained for each run. Out of all, the first run is the best one overall.

Table 5
Numerical statistics from the trec-evaluations of the 5 run on the 2020 topic set.

RUN	num_rel_ret	map	P_5	recall_1000	nDCG	nDCG@5
shanks-run-1	1263	0.1750	0.6490	0.6631	0.4979	0.5495
shanks-run-2	1255	0.1735	0.6408	0.6588	0.4960	0.5432
shanks-run-3	1103	0.1717	0.6490	0.5803	0.4764	0.5495
shanks-run-4	1199	0.1497	0.4816	0.6312	0.4536	0.3866
shanks-run-5	1263	0.1276	0.4082	0.6631	0.4283	0.3117

SUBMISSION UPDATE:

NEW EXPERIMENTAL RESULTS AFTER A NEW CORRECTED VERSION OF THE QRELS WAS RELEASED

All the previous results are based on an incorrect version of the .qrels file for the 2020 topics. Since we only knew about the new corrected version when the deadline was approaching, we could not recreate all the graphs and comparisons in 6. However, we managed to find the new optimal parameter sets and repeat the five run.

The new data is provided in Tables 6 and 7.

Table 6
The optimal parameters obtained by training on the 2020 topics (WITH CORRECTED QRELS).

Query	Similarity	tBoost	sBoost	pBoost	pDist
maxRecall	*MULTI*	0,3	0,2	0,75	12
maxnDCG	*LMD*	0,15	0,05	0,75	17

Table 7
Numerical statistics from the 5 run on the 2020 topic set (WITH CORRECTED QRELS).

RUN	num_rel_ret	map	P_5	recall_1000	nDCG	nDCG@5
shanks-run-1	*795*	0.3146	0.6245	0.8705	0.6521	0.6407
shanks-run-2	*790*	0.3126	0.5959	0.8671	0.6521	0.6213
shanks-run-3	770	0.3141	0.6245	0.8502	0.6479	0.6407
shanks-run-4	788	0.2546	0.4327	0.865	0.5839	0.4391
shanks-run-5	795	0.2565	0.4449	0.8705	0.588	0.4513

7. Conclusions and Future Work

At the end of this experiment we have discovered that the changes we made, lead to obtain fairly good improvement in respect to the starting baseline [Table 4] of our information retrieval system. All the statistics have undergone a significant increase, as an example we can see the improvement as follows: MAP +39.3%, P5 +68.8%, nDCG@5 +91.4%.

Comparing our results with the last year ones we have noticed that they follows their trend for the nDCG@5 parameter, but looking at the applied strategies they seem to be fairly different. Our results are in line with those of last year.

The whole process we followed highlighted a lot of possible expansions that with more time could be implemented and explored. As an example it could be interesting the application of some sort of location word detection, personal nouns recognition and compound words discernment and an improvement or even the addition of a new ranking phase that allows the insertion of some kind of argument quality analysis. The possibility of implementing a different approach based on some kind of machine learning model remains open. As we stated in the initial attempt section 3 we dropped the idea of using GPT-2 because it returned us unsatisfying results, so in the future we can go more into details of this modern tool and improve in this way the performance of our solution.

References

[1] C. Fellbaum, WordNet: An Electronic Lexical Database, Bradford Books, 1998.

[2] Y. Ajjour, H. Wachsmuth, J. Kiesel, M. Potthast, M. Hagen, B. Stein, args.me corpus, 2020. URL: https://doi.org/10.5281/zenodo.3734893. doi:10.5281/zenodo.3734893.

[3] A. Bondarenko, M. Fröbe, M. Beloucif, L. Gienapp, Y. Ajjour, A. Panchenko, C. Biemann, B. Stein, H. Wachsmuth, M. Potthast, M. Hagen, Overview of Touché 2020: Argument Retrieval, CEUR-WS 2696 (2020).

[4] C. L. A. Clarke, N. Craswell, I. Soboroff, Overview of the TREC 2009 Web Track, in: E. M. Voorhees, L. P. Buckland (Eds.), The Eighteenth Text REtrieval Conference Proceedings (TREC 2009), National Institute of Standards and Technology (NIST), Special Publication 500-278, Washington, USA, 2010.