SEUPD@CLEF: Team INTSEG on Argument Retrieval for Controversial Questions

Notebook for the Touché Lab on Argument Retrieval at CLEF 2022

Sepide Bahrami[1], Gnana Prakash Goli[1], Andrea Pasin[1], Neemol Rajkumari[1], Mohammad Muzammil Sohail[1], Paria Tahan[1] and Nicola Ferro[1]

[1]*University of Padua, Italy*

Abstract

Search Engines play important roles in helping users to rapidly retrieve relevant information. The technology underlying Search Engines has been improved in the last years, both in terms of hardware capabilities and in terms of software. However, they are still affected by many issues due to the continuously growing amount of data and the various forms in which it comes. In this paper we discuss our solution to the Information Retrieval problem proposed by the CLEF 2022 Touché Task 1.

We first describe in general the considered problem and subsequently present our Information Retrieval System implemented through Apache Lucene illustrating the various phases and methods applied to fulfil the objectives of the task. Eventually, we provide the obtained experimental results and possible explanations for them. In particular, we investigate the reasons for which some methods performed worse than others and describe possible ways to improve the system in the future.

Keywords

Information Retrieval, Information Retrieval System, Search Engine

1. Introduction

Nowadays Search Engines are used by the vast majority of people in their daily routine to retrieve any kind of information in a practical and efficient way. Even though there exist several advanced implementations of them, there are still many big challenges such as the argument retrieval for controversial [1] or comparative questions [2].

This report aims at providing an information retrieval system that has been developed to solve the task "Argument Retrieval for Controversial Questions" provided by the third Touché lab [3] on argument retrieval at CLEF 2022. The ultimate goal of this system is to retrieve a pair of coherent arguments through a huge dataset of online debate portals in response to a query about a controversial topic.

CLEF 2022: Conference and Labs of the Evaluation Forum, September 5–8, 2022, Bologna, Italy

sepide.bahrami@studenti.unipd.it (S. Bahrami); gnanaprakash.goli@studenti.unipd.it (G. P. Goli); andrea.pasin.1@studenti.unipd.it (A. Pasin); neemol.rajkumari@studenti.unipd.it (N. Rajkumari); mohammadmuzammil.sohail@studenti.unipd.it (M. M. Sohail); paria.tahan@studenti.unipd.it (P. Tahan); ferro@dei.unipd.it (N. Ferro)

http://www.dei.unipd.it/~ferro/ (N. Ferro)

0000-0001-9219-6239 (N. Ferro)

The paper is organized as follows: Section 2 describes the previous studies that have been done in this area; Section 3 describes our approach; Section 4 explains our experimental setup; Section 5 discusses our main findings; Section 6 investigates the relevance judgements, finally, Section 7 draws some conclusions; Section 8 outlooks for future works.

2. Related Work

Researchers have conducted a number of studies in the field of information retrieval [4] [5], each one concerning a different application in use. To the extent that it relates to our work as an argument retrieval task for decision making purposes [3], much of the knowledge base is drawn from the work done in the previous year, which was summarized in the overview of Touché lab in 2021 [6].

One critical thing to notice in an information retrieval task is the amount of time needed to perform the task. A number of studies have investigated the effectiveness of document organization in helping a user locate the relevant material among the retrieved documents as quickly as possible. These studies have used a set of clustering algorithms, and various experiments have demonstrated that clustering can be significantly more effective than the traditional ranked list approach [7] like the one we have implemented.

The researchers, who tackled a task similar to ours, believed one premise could be formulated differently [8]. Therefore, the system must avoid retrieving duplicate results and thus rely on some form of clustering. To achieve this, they propose a probabilistic ranking framework for premises based on the idea of TF-IDF that, given a query claim, the system will consider clusters of premises with the same meaning instead of isolated claims and premises.

As the retrieval task involves text documents, different word classes are likely to be significant. We can see the use of part of speech (POS) in order to compute a term weight [9]. Term weights are mathematical computations of how informative words are, and constitute an integral part of the statistical modelling of documents by information retrieval systems. Inspired by this reference, we will assign weights to noun and not-noun classes of words in our proposed methodology.

3. Methodology

In our system, the source file includes a parser, indexer, analyzer and a searcher in order to have a robust project structure. The general structure of the system can be divided into several parts. In particular each component has different aims as follows:

1. **Parser**: It parses the corpus CSV file and extracts documents.
2. **Analyzer**: It processes the extracted documents to apply tokenizers and stemmers.
3. **Indexer**: It creates indexable fields based on the parsed documents.
4. **Searcher**: It tries to retrieve relevant documents for each provided topic as a query.

The details of each part are separately reported in sections 3.2, 3.3, 3.4, 3.5.

3.1. Our System in General

Our information retrieval system starts by parsing the collection obtaining the parsed documents. Each parsed document is analyzed and indexed keeping only the necessary information. Documents are indexed in 2 different folders:

- **First Folder**: We store for each document its *context* field and its *sentences* field after having extracted only their *ids*.
- **Second Folder**: We store each sentence found in the *sentences* field with additional data such as its stance retrieved from the *conclusion* and *premises* fields.

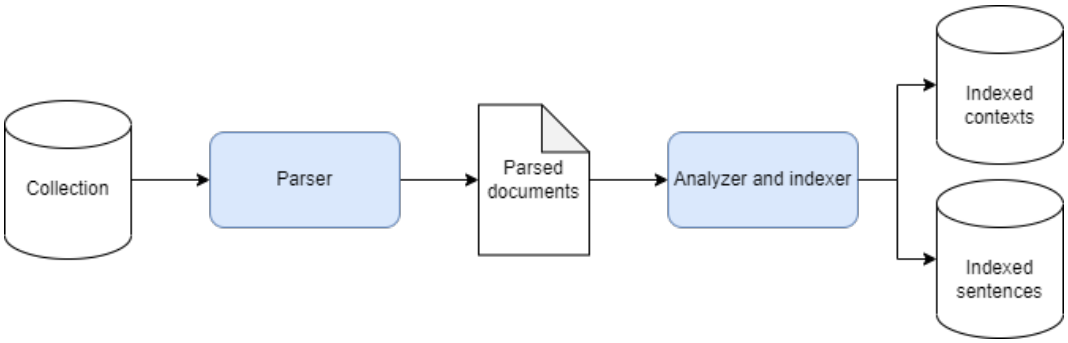


Figure 1: Schema of our system representing the parsing, analyzing and indexing parts

Topics are then parsed to formulate 2 queries for each of them: The first query will be used to retrieve the relevant sentences' *ids* according only to the corresponding contexts while the second query will be used to retrieve relevant *sentences*. Query expansion can be applied to formulate queries also by means of synonyms.

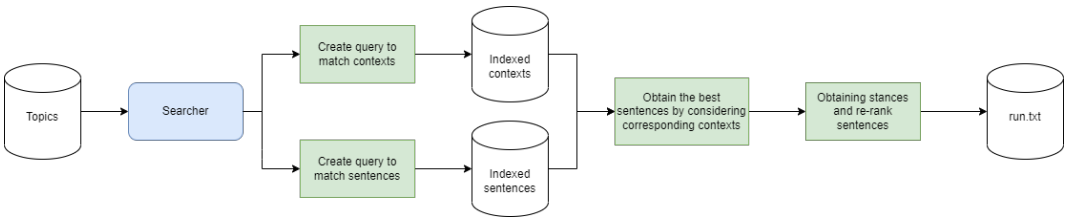


Figure 2: Schema of our system representing the searching part

3.2. Parser

Our utmost goal at this step was to extract as much data as possible and not to lose any meaningful data. To achieve the aimed completeness, our parsed document consists of 5 fields which are document id, hash-table of premise ids and their texts, premise stances, hash-table of conclusion ids and their texts and context.

Document ids were easy to extract while premise stances and hash-tables needed more effort. Our first approach to extract premise stances was done by means of regular expressions to find patterns of 'stance': 'PRO' or 'stance': 'CON' in the *premises* column of the corpus. Moreover,

the same was applied to extract premises/conclusions along with their ids. Here we have used regular expressions that iterates the *sentences* column of the corpus and tries to find patterns of 'sent-id' and 'sent-text' as the starting points and then extracts only the clean id and text of each premise/conclusion. However, in the second approach we decided to simplify the use of regular expressions by combining them with Jackson JSON parser [10]. Finally, we kept the context part for later experiments to check whether a retrieved sentence is ideally the most representative sentence of its corresponding argument or not. Keeping contexts was very tricky because it affects the indexing runtime.

Through our experiments we noticed that in almost 10 percent of the whole corpus, there exists a conclusion in the *conclusion* column but it does not appear in the *sentences* column. This happens because some conclusion texts are less than 3 words and do not actually contain any stance information. For all these cases we have considered the conclusion as empty in the hash-table.

3.3. Analyzer

We implemented a custom analyzer which extends the abstract class Analyzer provided in Apache Lucene [11]. We made the architecture flexible by parametrizing our analyzer which allows us to call it with different types of filters, tokenizers and stemmers. In this way the analyzer can be modified from the top layer without changing the core code each time. By changing the settings of these parameters it is possible to achieve better results in terms of precision. In addition, it is possible to increase or decrease both the index size and the indexing time required. We allow to choose from the possible parameters:

- **Tokenizer**: Allows choosing from different tokenizers to use such as Whitespace, Letter and Standard.
- **Stemmer**: Allows choosing from different stemmers to use such as none, EnglishMinimal, Porter and Krovetz.
- **Token Length**: Allows specifying the minimum length for a token and its maximum length.
- **Stop-List**: Allows specifying a possible stop-list if used.
- **English Possessive**: Allows specifying if to apply or not the English Possessive Filter.
- **Shingles**: Allows specifying the size of the word n-grams if used.
- **N-grams Size**: Allows specifying the size of n-grams if used.

3.3.1. Custom Stop-List

We decided to analyze the indexed collection and try to create custom stop-lists composed of the most frequent terms in the indexed collection. Thus we parsed, analyzed without any kind of stemming and indexed the collection to inspect it with Luke [12]. Luke is a tool that can be used to inspect the index and obtain some statistics of the indexed files. After having analyzed the entire indexed collection with Luke, we managed to create 2 different stop-lists composed of 100 terms each: one that can be used for the *context* field of the collection and one for the *sentences* field.

3.4. Indexer

The arguments of the collection are stored in the index using four fields provided by the parser. It is possible to avoid indexing some sentences that are recognized as spam. By looking at the document collection, in fact, we found many sentences that contained random characters or words in other languages. Therefore we decided to apply a language model provided by Apache Open NLP [13] to retrieve the language of a sentence: if a sentence is not recognized as being written in English, the sentence is discarded. This resulted in having roughly 300 thousands sentences removed from the indexed documents but this requires additional time when indexing. It is possible to see the details obtained from some of our runs regarding the index size and indexing time based on the parameters used in the analyzer and based on the utilization of the spam method used in the following table:

Tokenizer	Stemmer	Minimum token length	Maximum token length	Stoplist	Spam detection	Time	Total size
WhiteSpace	Krovetz	3	10	Yes	No	527 s	1.01 GB
WhiteSpace	Porter	2	10	Yes	No	559 s	1.00 GB
Standard	English Minimal	2	20	No	No	451 s	1.10 GB
WhiteSpace	English Minimal	2	20	No	Yes	1057 s	1.06 GB

Table 1: Indexing with different settings for the analyzer's parameters

3.5. Searcher

The searcher is the component that is in charge of retrieving, from a given topic, a ranked list of documents from the most to the least relevant. In this case the searcher aims at retrieving the best 1000 pairs of sentences for a given topic. Each couple of sentences should have the same stance that can be either *PRO* in case the pair of sentences is supporting the topic or *CON* vice versa. Our developed searcher integrates different tools:

- **Query Expansion**: This is a technique that allows us to reformulate a query in order to increase the number of matched documents.
- **Term Weighting**: This is a technique consisting of assigning different weights to each term of the query based on its discriminant power on the collection.
- **Document Re-Ranking**: This is a technique that is applied after having retrieved a large set of relevant documents. It involves applying different operations to rank the set of retrieved document once again and keep only a smaller subset composed by the most relevant ones.

We added the possibility to formulate queries based on both topics' title and description. After running our system we found that formulating queries using also the topics' description was deteriorating the precision of our system, therefore we decided to consider only the topics' title and to not further implement the retrieval based also on the description field. The searcher first parses the provided topics to extract their fields and then formulates 2 queries:

1. **First Query**: It is used to retrieve relevant sentences' id that can be either premises or conclusions based only on the *context* field of the collection.
2. **Second Query**: It is used to retrieve relevant sentences that can be either premises or conclusions based only on the *sentences* field of the collection.

After obtaining the 2 lists of relevant documents corresponding to each query, we only keep the best 2000 sentences based on the results of the 2 queries. This is shown in the *Algorithm 1*.

Algorithm 1 Filter ranked documents

```
1:  procedure getBest2000RankedSentences
2:      listSentencesFirstQuery ← firstQuery.execute() // sentences from collectionś contexts field
3:      listSentencesSecondQuery ← secondQuery.execute() // sentences from collectionś sentences field
4:      P1 ← weight parameter for contexts
5:      P2 ← weight parameter for sentences
6:      finalListSentences ← empty

7:      ∀ sentence ∈ listSentencesSecondQuery :
8:          if sentence ∈ listSentencesFirstQuery :
9:              sentence.weight=sentence.weight*P2+listSentencesFirstQuery.get(sentence).weight*P1
10:         finalListSentences.add(sentence)

11:     finalListSentences.orderByDescendingWeight()
12:     finalListSentences=finalListSentences.keepOnlyFirst2000()
13:     return finalListSentences
```

3.5.1. Query Expansion

We tried to implement the query expansion part by using the WordNet [14] thesaurus which provides a list of words and their corresponding synonyms. Each topic is processed in a way that when applying query expansion, we try to retrieve the possible synonyms for each term in the topic and consider them when formulating the final query.

Since each term can have different grammatical meaning based on the context (e.g. can be either a noun or a verb) and consequently the list of synonyms can be different, we first use the Apache Open NLP [13] library to mark up each term as corresponding to a particular part of speech (POS). We tested the POS-Tagger with 2 different models provided by Apache Open NLP that are **en-pos-maxent.bin** and **en-pos-perceptron.bin** and we eventually decided to use only the first one because it was generally more accurate.

After having tagged each term, we retrieve from the WordNet dataset the list of synonyms associated with each term based on its tag. This is achieved by keeping an in-memory representation of the dataset through a HashMap. Thus we can obtain search operations in *O(1)*.

The final query is then formulated by means of the Lucene [11] query classes **BooleanQuery**, **BoostQuery**, **TermQuery** and **PhraseQuery** in the following way:

1. Each term of the topic title considering both the original terms of the topic and synonyms is wrapped in a **TermQuery** object. If a synonym is composed of many terms (e.g. one of the synonyms for the term *'search'* is *'look for'*), then we use a **PhraseQuery** instead of a **TermQuery** when creating the second query (the one used to retrieve sentences based only on the topics' sentences field). In fact, we index sentences storing offset values as well, and this allows **PhraseQuery** for multiple terms matching.
2. Each **TermQuery**/**PhraseQuery** object is then wrapped in a **BoostQuery** object that allows to specify a weight. These parameters can be set in our ISearcher class as parameters that depend on the POS tag of the term and whether it is a synonym or an original term of the topic.
3. Each **BoostQuery** object is then added to a final **BooleanQuery** that represents the *OR* of all the added BoostQueries.

3.5.2. Term Weighting

When formulating a query, term weighting can be incredibly useful in emphasizing the importance of some terms over others. In fact, after having inspected the collection, we found that many terms appeared in nearly every document while others were found in only some of them. It is possible to see that terms appearing in almost every document do not have a high discriminant power: a query based only on that term would retrieve almost every document in the collection. We implemented our Searcher with 2 possible Term Weighting measures:

- **TF-IDF**: Over the whole collection. It is a customized version of the normal TF-IDF in which the term frequency TF is calculated over all the terms in the collection and not for the terms in the considered query. We also tried to smooth it by means of the fifth root. Let tot_{tk} be the total number of tokens in the collection, tot_{doc} be the total number of documents in the collection, $|token_i|$ be the number of occurrences of the i-th considered token over the whole collection and $|doc_{token_i}|$ be the number of documents containing at least one occurrence of the i-th considered token. Then the weight of a term according to this measure can be calculated as follows:

$$TFIDF(token_i) = TF(token_i) \cdot IDF(token_i) = \sqrt[5]{\frac{|token_i|}{tot_{tk}}} \cdot (\log \frac{tot_{doc}}{|doc_{token_i}|})$$

- **I-Coefficient**: This is a coefficient that we came up with to calculate the discriminant power of a term. Let tot_{doc} be the total number of documents in the collection, $|token_i|$ be the number of occurrences of the i-th considered token over the whole collection and $|doc_{token_i}|$ be the number of documents containing at least one occurrence of the i-th considered token. Then the weight of a term according to this measure can be calculated as follows:

$$I_{coef}(token_i) = (1 - \frac{|doc_{token_i}|}{tot_{doc}}) \cdot (1 - \frac{|doc_{token_i}|}{2 \cdot |token_i|})$$

3.5.3. Document Re-Ranking

We implemented document re-ranking using IBM Project Debater. This is an AI system developed by IBM that was developed to debate humans on complex topics [15]. It has been in development since 2012 and it is based on a massive dataset. To use project debater it is necessary to have an internet connection since each API call is done through http requests, then IBM servers elaborate the corresponding answer and send back an http response containing it. Using project debater therefore requires almost no computation by the computer that is running our system, but it takes quite some time to obtain an answer for all the topics.

We used the project debater Java library to exploit the following 2 APIs:

- **Pro/Cons API**: This API is used to understand if a sentence most strongly opposes the topic or most strongly supports it. This is used to obtain which of the 2000 retrieved sentences by our system are *PRO* or *CON* a given topic. In fact, the final output of our program needs to have a couple of sentences having same stance.
  This API is used by default in our system; the time taken to obtain the result for 50 topics considering only this part of our system is in the order of 1 hour.
- **Evidence Detection API**: This API is used to determine if a sentence is likely to contain evidence relating to a topic and vice versa. By means of this API we saw a significant improvement in the results, especially considering the best ranked sentences.
  Using this API in our system is optional; the time taken to obtain the result for 50 topics considering only this part of our system is in the order of 4-5 hours.

4. Experimental Setup

Our work is based on the following experimental setups:

- **Repository**: https://bitbucket.org/upd-dei-stud-prj/seupd2122-intseg/src/master/
- **Collection**: The corpora at https://webis.de/events/touche-22/shared-task-1.html
- **Evaluation Measure**: LMDirichlet
- **System Hardware**: We have used different computers to test our system but we report in this document only the results obtained with a machine having the following specifications:
  - CPU: Intel i5-8600k not overclocked
  - GPU: Zotac Nvidia Gtx 1060 AMP 6 GB
  - RAM: 32 GB ddr4 3000 MHz
  - SSD: Samsung 960 evo 1 TB nvme, sequential read: 3,200MB/s, sequential write: 1,900MB/s

5. Results

In order to determine which configuration provided better results, we tested the system with a variety of configurations. Additionally, we also tried adapting it to last year's Touché Task 1 [6].

This has been done by modifying the searcher part so that queries are only applied to contexts and stance detection is not being performed.

Here we report the results of 3 systems with different configurations based on last year's topics and relevance judgements following some important evaluation measures. The 3 systems reported here have been chosen to demonstrate how different configurations lead to variant results. However these 3 systems are different than the one submitted to CLEF [3].

	Configurations		
Evaluation Measure	**System 1**: Letter Tok. English Stem. No Stop-List POS Tag WordNet Evidence Det. ICoefficient LMDirichlet	**System 2**: Whitespace Tok. Krovetz Stem. Stop-List POS Tag WordNet Evidence Det. ICoefficient LMDirichlet	**System 3**: Whitespace Tok. Porter Stem. No Stop-List POS Tag WordNet TFIDF LMDirichlet
num_ret	43397	47895	48054
num_rel	1818	1818	1818
num_rel_ret	1113	1048	806
map	0.0329	0.0240	0.0197
Rprec	0.0458	0.0269	0.0316
bpref	0.4731	0.4416	0.3645
iprec_at_recall_0.00	0.1695	0.0954	0.1081
iprec_at_recall_0.20	0.0512	0.0408	0.0404
iprec_at_recall_0.40	0.0396	0.0344	0.0240
iprec_at_recall_0.60	0.0279	0.0244	0.0135
iprec_at_recall_0.80	0.0156	0.0118	0.0048
P_5	0.0760	0.0280	0.0240
P_10	0.0640	0.0360	0.0220
P_100	0.0370	0.0238	0.0276
P_1000	0.0223	0.0210	0.0161
recall_5	0.0101	0.0038	0.0030
recall_10	0.0164	0.0101	0.0065
recall_100	0.1063	0.0667	0.0888
recall_1000	0.6134	0.5753	0.4369
ndcg_cut_5	0.0428	0.0187	0.0214
ndcg_cut_10	0.0417	0.0227	0.0188
ndcg_cut_100	0.0715	0.0430	0.0550
ndcg_cut_1000	0.2567	0.2256	0.1830

Table 2: Touché Task 1 2021 [6] Results

We have observed that, in general, the system performs better when configured to not use stop-lists and Porter stemmer. As well, we noticed that using synonyms and POS tags could improve our precision levels, whereas the use of stemmers decreased overall. It is possible to see from the above results that our IR system does not perform very well on the topics from last year. However, we tried to change different parameters to find which configurations were the most effective ones in order to submit our runs to the CLEF [3].

We inspected how the different queries were created. This was in order to understand if there were some problems in our system. However, queries seemed to be formulated in a correct way, by assigning different weights to different terms based on their discriminant power and part of speech tag. We also tried to inspect which topics were obtaining the best results and therefore we tried to see if there were problems with our stop-lists or particular stemmers/tokenizers adapting our system as a consequence.

6. Statistical Analysis

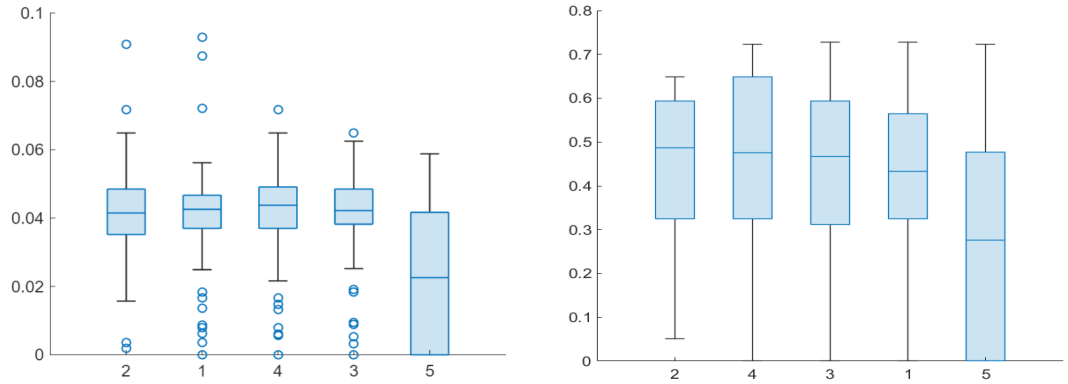
This section contains a summary of the hypothesis testing we did using ANOVA and boxplots for our 5 different runs submitted to CLEF [3] which their configurations are listed in Table 3. The mean value is calculated using two metrics: Average Precision (AP) and Normalized Discounted Cumulative Gain at 10 (nDCG@10).

Run ID	**Stemmer**	**Tokenizer**	**Term Weighting**	**Stoplist**	**POS Tag/Evidence Detection**
1	Krovetz	WhiteSpace	ICoefficient	Yes	No
2	Krovetz	Letter	ICoefficient	No	Yes
3	English Minimal	Letter	ICoefficient	No	Yes
4	Krovetz	WhiteSpace	ICoefficient	Yes	Yes
5	Porter	WhiteSpace	TF-IDF	No	Yes

Table 3: The runs submitted to CLEF [3]

Based on the Average Precision in Figure 3 (a) we can notice that *run 1*, *run 2*, *run 3* and *run 4* all show a similar structure in terms of their median values and interquartile ranges. However, with respect to the length of the whiskers, we can note some differences. Moreover, as shown in *run 1* and *run 4*, both of which use stoplists, there are many outliers (the ones represented as circles). The presence of these outliers leads to deteriorated performance.

When considering nDCG@10 in Figure 3 (b), *run 1* and *run 3* have an almost identical structure. Since *run 2* was the one with the highest mean value and had the smallest standard deviation, *run 2* in this case is the one with the best performance.



(a) Using AP **(b)** Using nDCG@10

Figure 3: Boxplots of runs (**X**: Runs in decreasing order of mean performance, **Y**: Performance)

On the other hand, the ANOVA test is used to determine if the means of some groups are equal or not. It assesses whether it is feasible to reject or not reject a null hypothesis H0 (hypothesis that all means are equal between the groups). The following two tables demonstrate that the p-value obtained when considering both AP and nDCG is below the value alpha that we set to 0.05. We can therefore conclude that the null hypothesis is rejected.

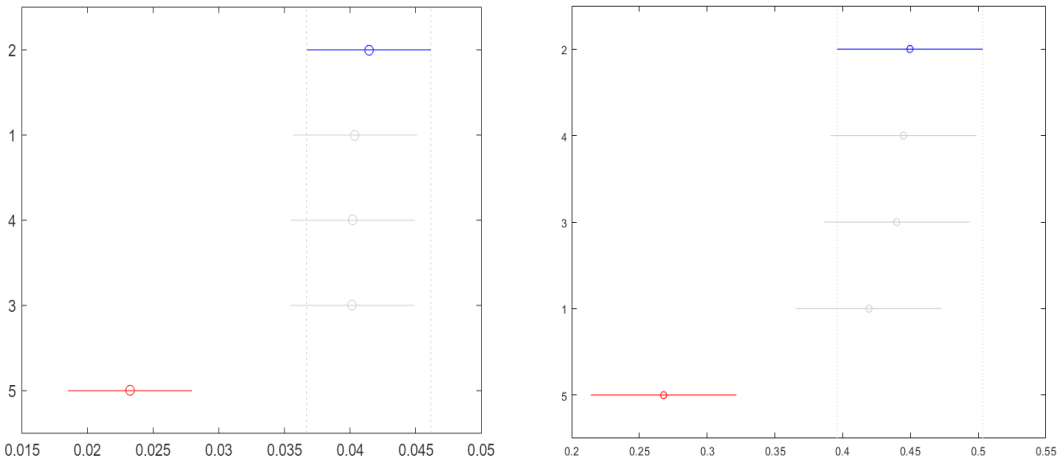
'Source'	**'SS'**	**'df'**	**'MS'**	**'F'**	**'Prob>F'**
'Columns'	0.0120	4	0.0030	10.0253	1.5713e-07
'Error'	0.0736	245	3.0042e-04	[]	[]
'Total'	0.0857	249	[]	[]	[]

Table 4: The one-way ANOVA table using AP

'Source'	**'SS'**	**'df'**	**'MS'**	**'F'**	**'Prob>F'**
'Columns'	1.1896	4	0.2974	7.6717	7.7149e-06
'Error'	9.4976	245	0.0388	[]	[]
'Total'	10.6872	249	[]	[]	[]

Table 5: The one-way ANOVA table using nDCG@10

From the following charts, we can see that *run 5* is quite different from the others.



(a) AP **(b)** nDCG@10

Figure 4: ANOVA tables (The means of runs 2 and 5 in the X-axis are significantly different)

7. Conclusions

In this paper we reviewed the structure of an Information Retrieval system and we proposed a solution to the Information Retrieval problem proposed by the CLEF 2022 Touchè Task 1 [3]. We started by describing each of the different components in our system and further we showed how they interact with each other. We also discussed the results obtained by running our system with various configurations and we highlighted which could have been some of its major issues. We finally observe that there is the need for a more complex evaluation measure. In fact, there may be many combinations of sentences that could be matched together forming couple of relevant sentences having same stance. Therefore before using *trec_eval* [16] to evaluate our system, it could be useful to modify the relevance judgements/quality judgements given by CLEF [3] according to the coherence judgements to account also for different possible combinations of sentences.

8. Discussions and Future Work

Our IR system is far from perfect and there are many aspects that can be improved to make it more effective. Extracting data that can be used to formulate queries from the topics' description field may improve performances. This must be done in a way that only the relevant terms are actually added to the query because some terms may introduce noise and decrease precision.

Another improvement that can be done is to parse contexts in a better way to index only the fields that can actually be useful. In this way, the index size can be decreased and the precision of the system may be improved by formulating queries to more fields.

In addition, another feature that could be implemented is to apply query expansion by retrieving some terms from documents that resulted the most relevant from a first search phase and formulate a new query accordingly. In this way we could create a second query containing more terms and that would likely increase the recall (similar approach to pseudo-relevance feedback).

Lastly machine learning and in particular systems like BERT's [17] could be used to further improve the results when doing re-ranking and stance detection. Machine learning is, in fact, a field in which many researchers are interested and therefore new well performing solutions will be proposed. Thus integrating those solutions in future IR systems will probably be a key factor to further improve their effectiveness.

References

[1] Touchè@CLEF, Touché task 1: Argument retrieval for controversial questions, https://webis.de/events/touche-22/shared-task-1.html, 2022.

[2] Touchè@CLEF, Touché task 2: Argument retrieval for comparative questions, https://webis.de/events/touche-22/shared-task-2.html, 2022.

[3] A. Bondarenko, M. Fröbe, J. Kiesel, S. Syed, T. Gurcke, M. Beloucif, A. Panchenko, C. Biemann, B. Stein, H. Wachsmuth, M. Potthast, M. Hagen, Overview of Touché 2022: Argument Retrieval, in: Experimental IR Meets Multilinguality, Multimodality, and Interaction. 13th International Conference of the CLEF Association (CLEF 2022), Lecture Notes in Computer Science, Springer, Berlin Heidelberg New York, 2022, p. to appear.

[4] A. Leuski, Evaluating document clustering for interactive information retrieval, in: Proceedings of the tenth international conference on Information and knowledge management, 2001, pp. 33–40.

[5] A. Singhal, et al., Modern information retrieval: A brief overview, IEEE Data Eng. Bull. 24 (2001) 35–43.

[6] A. Bondarenko, L. Gienapp, M. Fröbe, M. Beloucif, Y. Ajjour, A. Panchenko, C. Biemann, B. Stein, H. Wachsmuth, M. Potthast, M. Hagen, Overview of Touché 2021: Argument Retrieval, 2021, pp. 450–467. doi:10.1007/978-3-030-85251-1_28.

[7] A. Leuski, Evaluating document clustering for interactive information retrieval, in: Proceedings of the Tenth International Conference on Information and Knowledge Management, CIKM '01, Association for Computing Machinery, New York, NY, USA, 2001, p. 33–40. URL: https://doi.org/10.1145/502585.502592. doi:10.1145/502585.502592.

[8] L. Dumani, P. J. Neumann, R. Schenkel, A framework for argument retrieval, in: J. M. Jose, E. Yilmaz, J. Magalhães, P. Castells, N. Ferro, M. J. Silva, F. Martins (Eds.), Advances in Information Retrieval, Springer International Publishing, Cham, 2020, pp. 431–445.

[9] C. Lioma, C. van Rijsbergen, Part of speech n-grams and information retrieval, Revue française de linguistique appliquée, XIII, 2008. URL: https://doi.org/10.3917/rfla.131.0009.

[10] FasterXML, Jackson json parser, https://github.com/FasterXML/jackson, 2022.

[11] Apache, Apache lucene, https://lucene.apache.org/, 2022.

[12] Luke, Luke, http://www.getopt.org/luke/, 2022.

[13] Apache, Apache open nlp, https://opennlp.apache.org/, 2022.

[14] Wordnet: A lexical database for english, https://wordnet.princeton.edu/download, 2005. Accessed: 2022-04-25.
[15] IBM, Ibm project debater, https://research.ibm.com/interactive/project-debater/how-it-works/, 2022.
[16] N. I. of Standards, Technology, trec_eval, https://trec.nist.gov/trec_eval/, 2022.
[17] J. Devlin, M.-W. Chang, K. Lee, K. N. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, 2018. URL: https://arxiv.org/abs/1810.04805.