SEUPD@CLEF: Team 6musk on Argument Retrieval for Controversial Questions by Using Pairs Selection and Query Expansion

Notebook for the Touché Lab on Argument Retrieval at CLEF 2022

Lorenzo Cappellotto[1], Matteo Lando[1], Daniel Lupu[1], Marco Mariotto[1], Riccardo Rosalen[1] and Nicola Ferro[1]

[1]*University of Padua, Italy*

Abstract

This is a report based on the work done for Touché Task 1: Argument Retrieval for Controversial Questions at CLEF 2022 by team 6musk (whose members are all students of the University of Padua). This year's task focuses on the problem of retrieving a couple of sentences to support users who search for arguments to be used in conversations.

Keywords

Touchè 2022, Argument Retrieval, Search Engines, Controversial Questions, Pair of Sentences

1. Introduction

This report aims at providing a brief explanation of the Information Retrieval system built for Touché lab 2022 [1]. We participated in Touché Task 1[1]: Argument Retrieval for Controversial Questions. This year's task focused on the retrieval of pairs of sentences, instead of whole documents, from the collection of arguments used for the previous iteration of the task. The corpus used for the development of the system is a pre-processed version of the args.me corpus [2] (version 2020-04-01) that was also used during the previous year's edition of the Touché task.

The level of complexity added to this year's version of Touché Task 1 required us to think on how to specialize a basic retrieval system for documents to support this new required functionality. After developing a basic system for last year's task, we focused on this year's challenges and on more elaborated ideas to solve them. In a first iteration we decided to divide the system in two distinct phases, the first one to retrieve the most relevant documents for a topic and the second one to select the two most argumentative sentences for each topic and each

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

lorenzo.cappellotto@studenti.unipd.it (L. Cappellotto); matteo.lando@studenti.unipd.it (M. Lando); daniel.lupu@studenti.unipd.it (D. Lupu); marco.mariotto.1@studenti.unipd.it (M. Mariotto); riccardo.rosalen@studenti.unipd.it (R. Rosalen); ferro@dei.unipd.it (N. Ferro)

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

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

CEUR Workshop Proceedings (CEUR-WS.org)

[1]https://webis.de/events/touche-22/shared-task-1.html

document retrieved. Once the full pipeline was in place, we tried to upgrade the second phase to select the two most argumentative sentences for each topic and each document retrieved. Our focus was not to produce a top performing specialized system, but to try to develop general ideas in order to approach sentences selection and query expansion.

The paper is organized as follows: Section 2 introduces related works; Section 3 describes our approach; Section 4 details the implementation process; Section 5 explains our experimental setup; Section 6 discusses our main findings; Section **??** carries out failure analysis; finally, Section 7 draws some conclusions and outlooks for future works.

2. Related Work

On Argument Search and Retrieval

Argumentation is a daily occurrence for taking individual and collaborative decisions. We follow the definition of an argument as proposed by [Walton et al. 2008] to be a conclusion (claim) supported by a set of premises (reason) and to be conveying a stance on a controversial topic [Freeley and Steinberg, 2009]. Sometimes the premise can be left implicit (enthymemes) and the mechanism to draw the conclusion from the premises is informal.

The Web is the most important and extensive source of information, and everyone will rely on Google at some point to fill their lack of knowledge. However, as much as search engines usually provide fast and correct answers for factual information, it is not so straightforward when there are multiple controversial opinions [3]. Also, fake news worsen their effectiveness, forcing users to check the credibility of sources [4] (e.g. the specific website he is visiting). As an example, determining the stance of Twitter users towards messages may constitute an indirect way to identify the truthfulness of discussed rumors [5].

The corpus we have is preprocessed from the argument search engine args.me[2], which clearly identifies sentences (both premises and conclusions) and the premise's stance towards conclusion, but crawling the Web to extract such data is a research field on its own. Automatically detecting argumentative content in natural language text, i.e. argument mining, can help to determine people's opinion on a given topic, why they hold it, and extract insight on public matters, such as politics [6]. More details about argument mining can be found in the survey by Lawrence and Reed [7].

Touché 2021

Our starting point was the Overview of Touché 2021: Argument Retrieval paper [8], which provided valuable insights on what is argumentation, how we can assess its quality and what retrieval approaches were successful past editions, being this the third one.

We developed our system taking into account two winning techniques: query expansion through Wordnet synonyms/antonyms and DirichletLM as the best similarity function (among BM25, DPH, and TF-IDF). Furthermore, we used last year's qrels to identify the best combinations based on our experiments.

[2] https://www.args.me/index.html

Query Expansion
Query Expansion (QE) is a technique that automatically expands the initial query issued by the user, usually with a human controlled thesaurus. It helps to reduce ambiguity and to increase recall. We added synonyms to the original query in two different ways, namely with WordNet (vocabulary-based) and Word2vec (corpus-based). The interested reader can find more about QE in the comprehensive survey done by Azad and Deepak [9], which provides an overview on core methodologies, use cases and state of the art approaches. Current techniques adopt transformer-based models which show more promising results than classical ones.

On Stance Detection and Sentiment Analysis
This year we face an increased layer of complexity: we have to retrieve two sentences with the same stance, and identify if this last one is "pro" or "con" the given topic. The Webis Group[3] already started tackling the task of predicting if two arguments share the same stance, though the Same Side Stance Classification Shared task [10], which achieved promising results and showed that it's both feasible and there's room for improvement.
Retrieving a relevant pair of sentences is closely related both to stance detection, for which a tutorial can be found in [11][4], and to sentiment analysis. For example, Alshari et al. [12] used a Word2vec model to expand SentiWordNet, a lexical dictionary for sentiment analysis to learn the polarity of words in the corpus, and evaluated their approach on the IMDB with two classifiers (Logistic Regression and SVM). To estimate the polarity of each non-opinion word in the vocabulary, they computed the score of a given word based on the polarity of the closest term present in SentiWordNet [13], with encouraging performance in identifying "positive" and "negative" reviews . Also, current research on stance detection aims at finding the position of a person from a piece of text they produce, focusing on social media portals [14].

3. Methodology

In this section we address how the system was developed starting from examples studied during the Search Engine course. Furthermore, we will highlight which parts we focused more on and, more in details, what we tried to improve the base system.

3.1. Base system, our first steps into IR

As previously stated in the introduction, the system was built in a first iteration to obtain a fully functioning pipeline for retrieving pairs of sentences. In this sense, we spent the first part of the development phase building a basic functioning pipeline to retrieve documents as in last year's edition of Touché Task 1. After concluding the previous step, we turned to this year's challenges and focused on upgrading the base system to retrieve sentences and increasing the performance by trying both off-the-shelf components and custom ones.
The three main components of the basic pipeline are the following:

[3] https://webis.de/
[4] https://dkucuk.github.io/stancedetection/

3.1.1. ToucheParser

Since the corpus was provided as a large .csv file, the first important step was to parse it keeping in mind the memory usage. `ToucheParser` is a class which specializes the abstract `DocumentParser`, an Iterable class used to run through the corpus and to index each document using the `DocumentIndexer`.
In particular, by using a `BufferedReader` and the Jackson `CsvMapper`, each line of the corpus was transformed into a proper fielded instance of `ParsedDocument` which could then be easily used to index the document with `Lucene`[5].

3.1.2. DirectoryIndexer

Once we parsed the lines from the corpus file correctly and created an Iterable class (`ToucheParser`) to obtain a `ParsedDocument` one by one, we proceeded to index all the documents by using the `Lucene` API.
For indexing execution, our class expected to get 365408 documents, previously extracted from `ToucheParser`. As already mentioned, the initially implemented `StandardAnalyzer` belongs to the packet `org.apache.lucene.analysis.Analyzer`. It is a simple analyzer where we specified a `StandardTokenizerFactory` and a `LowerCaseFilterFactory` to make all texts lowercase. In addition, the `BM25Similarity` similarity function has been used as a starting point to match the documents with the topic. (In the next sections it will be shown how we were able to change the analyzer and the similarity function).

3.1.3. BasicSearcher

After checking the integrity of the index using Luke (Lucene Index Toolbox) and verifying all the fields were containing the expected content, we created a basic searcher to retrieve documents. In particular, after opening an `IndexSearcher` and using `ToucheTopicsReader` to read each topic, the searcher constructs a query from the title of the topic only. The searcher matches it with the text and the conclusion of the documents in the index providing the best matching documents for each topic (at most 1000). For the initial searcher, `StandardAnalyzer` and `BM25Similarity` were used as Analyzer and similarity function to be coherent with what we used for indexing.
As it will be shown in the results section, after this part we were able to run the system using last year's topics and relevance judgments to check the base performance for our system.

3.2. Different approaches for Sentence Selection

The second planned iteration of the development required upgrading the searcher(s) to select pairs of sentences in a meaningful way, instead of single documents.
For this phase of the development, we decided to upgrade `BasicSearcher` to trivially retrieve the conclusion and the first premise of the initially selected documents, or, if no conclusion was present, add another premise to the output (e.g., choosing the first two premises of the document). This sparked the idea for two different approaches with a common idea at the core,

[5] https://lucene.apache.org/

doing sentence selection from the document initially matched with the last year's version of the system. In our view, this is beneficial since pairs of sentences from the same document have intuitively higher coherence than arbitrary pairs of sentences from the entire collection.

3.2.1. SentencesSearcher

`SentencesSearcher` is the first of the two new solutions. The development of the class required the generalization of `BasicSearcher` into an abstract class `AbstractSearcher` to follow the DRY principle and avoid repetitions. `AbstractSearcher` was used to implement the shared portion of code and later became the superclass for all the rest of the searchers.
Until this point, the sentences were not correctly indexed to provide fast matching between them and the topics, so it became necessary to index them. To separate the document retrieval from the sentences retrieval we chose to create a new index. As in the case of the searchers we made use of an abstract class called `AbstractDirectoryIndexer` that contained the shared code (DRY principle) for `DirectoryIndexerDocument`, the indexer for the documents, and `DirectoryIndexerSentences`, the index for the sentences.
To sum up, the searcher works by retrieving for each topic the required number of documents using the index constructed by `DirectoryIndexerDocument` and then, for each document, providing the two sentences from that document that best match the topic title using the index constructed by `DirectoryIndexerSentences`.

3.2.2. ConclusionSearcher

After developing `SentencesSearcher`, we decided to push more on the idea of coherence between sentences of the same document. `ConclusionSearcher` works in a similar way to `SentencesSearcher`. For each topic, it retrieves the required number of documents using the index constructed by `DirectoryIndexerDocument` and then, chooses the two sentences by selecting always the first conclusion of the document and the premise (of the same document) which better matches the text of the conclusion. To implement this second step, we constructed a query that finds only the relevant premises in the index created by `DirectoryIndexerSentences` and matches them with the text of the conclusion.

3.3. Increasing performance

Once the full system (Figure 1) for this year edition of Task 1 was completed and different sentence selection methods were explored, we decided to focus on improving the performance of the overall system. The way in which we decided to divide the system was beneficial in experimenting with ideas since we were able to evaluate them using last year relevance judgment. We've tried query expansions because it was listed among the winning approaches of last year and wanted to see its effectiveness against the new topics.
Following are the details of the various off-the-shelf components tried and the custom query expansion techniques used to improve performance.

3.3.1. Trying out off-the-shelf components

To try different off-the-shelf components it was necessary to develop our own analyzer, a subclass of `org.apache.lucene.analysis.Analyzer`.
The main components explored were:

- Stoplists. We experimented with '*glasgow.txt*' and '*smart.txt*' stoplists, respectively composed by 319 and 571 common words.
- Stemming. We experimented with `PorterStemFiler` and `KStemFilter`, versions of the Porter stemmer and the Krovetz stemmer found in the `Lucene` library.
- Character N-grams and Word N-grams (Shingles). Also in this case filters were implemented by `Lucene`, named respectively `NGramTokenFilter` (with 3 characters) and `ShingleFilter` (applied using 3 words).

The components were chosen using a heuristic approach. Starting from the baseline model we added one component at a time and selected at each iteration the best performing between the ones available.

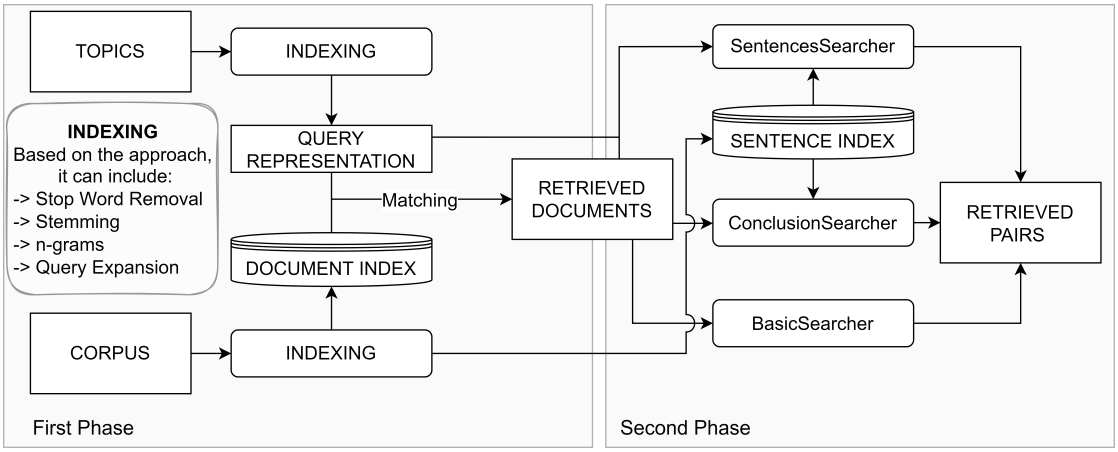

Figure 1: Complete pipeline with the different parts

3.3.2. Query expansion based on WordNet

WordNet is the most popular thesaurus[6]. Different versions can be found online: for convenience reasons which we will explain in 4, we adopted a Prolog version of WordNet 3.1, which can be found at https://github.com/ekaf/wordnet-prolog. WordNet is a network where words are linked to other words which may be semantically related. The new searcher which is implemented in `WordNetSearcher` is an extension of `ConclusionSearcher` in the following sense: the query (topic title) is expanded using all synonyms from WordNet and searched among all documents (using Dirichlet similarity function); for each document among the top 1000 retrieved by the

[6] https://wordnet.princeton.edu/

index searcher, the conclusion is again expanded using at most 2 synonyms for each term and searched among all sentences of this document to find the best possible matching premise (since the other selected sentence is necessarily the conclusion). The value 2 is a heuristic number and can be modified as we please: it is hard to define a precise value for the maximum number of synonyms without qrels.

To speed up the process of searching one can avoid expanding the full conclusion with all terms in WordNet, but instead provide a keep only filter: for instance one can decide to keep only synonyms appearing in the topic description, or in the topic narrative (or both) which likely may contain synonyms of words in the topic title. In the end we opted to fully expand the conclusion, even though the process of searching took almost an hour. See 7 for details on speeding things up or alternative designs.

One final note: we performed the search of the sentences as in `ConclusionSearcher` since we are confident that it is more likely that conclusion will be expanded further than the query title, as in general it is longer (in fact a mixed approach could be tried as well).

3.3.3. Query expansion based on Word2vec

Word2vec [15] is a natural language technique to produce word embeddings, ideally from a large corpus of text. Each word is represented as a vector of numbers and the cosine similarity computes the semantic relationship between words. It's a set of neural network models with one hidden layer trained to predict the features of the surrounding window of a given word (Skip-gram) or the target word from a context window (CBOW, continuous-bag-of-words). CBOW better captures syntactic relationships between words, meanwhile Skip-gram the semantic one. For example, given the term "day" CBOW may retrieve "days", whereas Skip-gram may also retrieve "night", which is semantically close but not syntactically. Skip-gram is less sensitive to high frequency words and is less likely to overfit because it looks at single words each iteration, whereas CBOW trains on a window of words, meaning that it sees frequent words more often. For the previous reasons, we opted for Skip-gram.

4. Implementation

4.1. Query expansion based on WordNet

The package `org.apache.lucene.wordnet` provided by `Lucene` allows for adding a synonym filter from a WordNet-like database in Prolog. It loads the database in a `SynonymMap` which is a fast hash map used to retrieve synonyms from any specified lowercase word. After creating this map, we can add a `SynonymTokenFilter` to our analyzer, passing the former to the constructor. We explicitly made an analyzer called `WordNetQueryExpander` which does other things among setting up a synonym filter. In this order, the first step consists in setting up a lower-case filter, then a stop filter, and finally the synonym filter. The constructor of the analyzer accepts a parameter `set` which enables us to keep only synonyms words contained in `set`: in this sense this has been used to speed up the process of searching, as said before. The second step applies this keep only filter if `set` is not null, and then applies the Krovetz Stemmer.

We tried two combinations of filters for this analyzer (the synonym filter is always applied clearly):

- a combination of lowercase filter, stop filter (Glasgow list), synonym filter, Krovetz stemmer (the default)
- a combination of lowercase filter and synonym filter

both were tried using BM-25 similarity and Dirichlet similarity (see 6 for a discussion of the final results).

4.2. Query expansion based on Word2vec

Word2Vec generates synonyms after training on the corpus, which we preprocessed to remove noise and improve performance (training and searching time). The training data for the word2vec model was built removing all duplicates, i.e. documents with the same "sourceId" sub-field, resulting in 58962 documents. We filtered from "args_processed_04_01.csv" the "sourceText" sub-field, replacing with a whitespace all words with three or more equal consecutive characters, words that contained numbers and the substring "xa0". All terms have been turned lowercase and only ones between 3 and 14 characters were kept.

The model was built using the deeplearning4j library with the following:

```
Word2Vec vec = new Word2Vec.Builder().stopWords(list).minWordFrequency(20)
.layerSize(512).windowSize(10).iterate(iter).tokenizerFactory(t).build();
```

- `stopWords(list)` is a stopword list based on '*smart.txt*' plus approximately 50 of the more common terms from the context field.
- `minWordFrequency(20)` forces the removal from the vocabulary of words with less than 20 repetitions.
- `tokenizerFactory(new StandardTokenizer())` is the default tokenizer of the deeplearning4j library.
- `windowSize` is the number of words used to compute the numeric vector of a given word. Since we already preprocessed "SourceText" and removed many stop words we kept a relatively low value (10).
- `layerSize` is the number of features in the word vector. Default values are around 100, we used 512 to try to improve the quality, with no apparent downsides. Printing out synonyms for query terms we saw a closer semantic meaning with higher values of this parameter (512 vs 256).

Word2Vec run description In the first phase, query expansion was performed on the topic title, using two stoplists ('*glasgow.txt*' and '*smart.txt*') but no stemming because, as can be seen from row 1 and rows 2 & 3 (student-students) in table 1, we are kind of doing lemmatization. Also antonyms may be added (row 6)). During searching time, only synonyms with a similarity higher than a heuristically found value (0.53 in range 0-1) were added, and at most 5 per term of the original query. For the second phase, we've used the conclusion field from the retrieved

document as input for another query to find the most similar premise in the document (as in `ConclusionSearcher`). The Krovetz stemmer has been applied and no stoplist.

Table 1
Example of synonyms/antonyms generated with Word2vec

	Query Term	Synonym	Similarity
1	teachers	teacher	0.76
2	teachers	student	0.66
3	teachers	students	0.66
4	teachers	classroom	0.65
5	teachers	schools	0.61
6	legal	illegal	0.60

5. Experimental Setup

This section contains more details on which hardware was used to run the experiments, alongside which tools and measures were used to evaluate the performance.
The hardware used to create the indices, search the topics and evaluating the performance is:

- OS: MacOS Big Sur
- CPU: AMD Ryzen 7 3800X 3.9 Ghz
- RAM: 16GB DDR4 3200MHZ
- Storage: SSD M.2 PCIe NVMe 512GB

The development of the system was based on the experimental collections created for this year and last year editions of Touché Task 1. To evaluation tools used during the development are trec_eval[7] to compute the measures and Luke, a GUI that lets you look inside the index `Lucene` creates, to check indices health and coherence.
The evaluation measures used to check the system during the various phases of the development are:

- nDCG_cut_5 or Normalized Discounted Cumulated Gain at cut 5: main metric used for evaluating the systems in Touché Task 1.
- MAP or Mean Average Precision: with a single number it gives an overall view of the system's performance, and offers a different perspective w.r.t nDCG_cut_5 since it's a binary relevance function. Plus, the more qrels available, the more accurate and we had the ground truth of previous two editions.

The git repository containing the source code of the system is available at the link seupd2122-6musk.

[7] https://github.com/usnistgov/trec_eval/

6. Results

6.1. Performance on This Year's Relevance Judgments

This year another class of judgments was given besides quality and relevance ones: in order to establish if a pair of sentences do not contradict each other, Touché organizers supplied us with coherence qrels. We want to understand whether our searching strategies are different or not: comparing performance scores between runs always requires statistical analysis; to do so, we will compare the distributions of NDCG values among the 5 runs we submitted to CLEF for each topic title. The runs we chose based on last year performance are:

1. seupd2122-6musk-kstem-stop-shingle3 (uses 3.2.1 and off the shelf components described in 3.3.1)
2. seupd2122-6musk-stop-kstem-basic (described in 3.1.3)
3. seupd2122-6musk-stop-kstem-concsearch (described in 3.2.2)
4. seupd2122-6musk-stop-wordnet-kstem-dirichlet (described in 3.3.2)
5. seupd2122-6musk-word2vec-sentences-kstem (combines searcher from 3.2.1 and expansion technique described in 3.3.3)

We have the following results for NDCG on each type of judgment:

Table 2
nDCG@5 for all judgments

Run	nDCG@5 qual	nDCG@5 rel	nDCG@5 coh
seupd2122-6musk-kstem-stop-shingle3	0.7258	**0.6378**	**0.3699**
seupd2122-6musk-stop-kstem-basic	0.2876	0.1767	0.1962
seupd2122-6musk-stop-kstem-concsearch	0.7244	0.5881	0.3415
seupd2122-6musk-stop-wordnet-kstem-dirichlet	**0.7299**	0.6055	0.3622
seupd2122-6musk-word2vec-sentences-kstem	0.7183	0.5822	0.3374

This table only shows means computed by the trec_eval tool over all topics; to get a better idea over the whole data, boxplots are really useful:

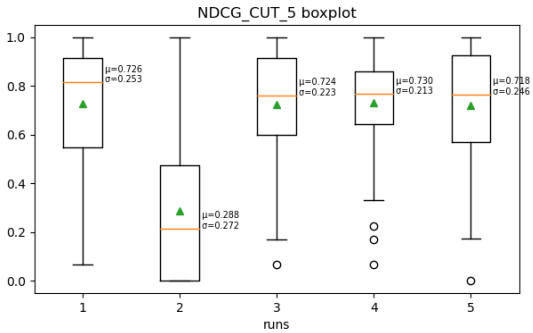

Figure 2: nDCG@5 for quality qrels

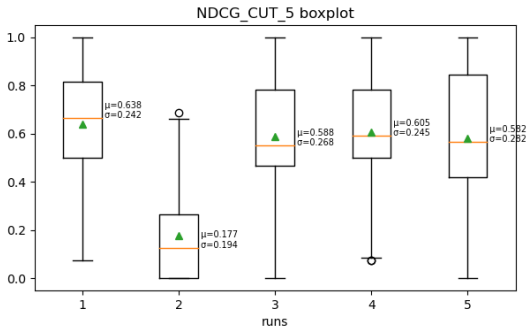

Figure 3: nDCG@5 for relevance qrels

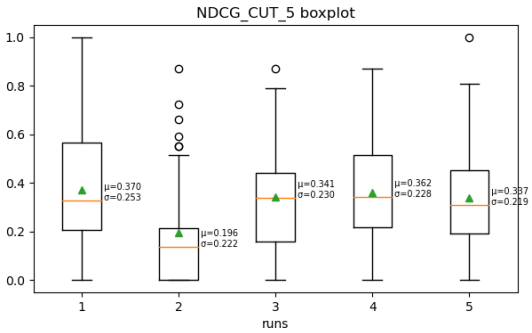

Figure 4: nDCG@5 for coherence qrels

Informally we can say that runs 1, 3, 4, 5 perform all better than run 2, which intuitively makes sense, since the core of the searcher is very basic. Runs 3, 4, 5 are almost equal, in particular run 4 shows a lower interquartile range. To verify these statements we will apply both ANOVA one way and pairwise Student's t-tests.

qrels	F stat	p_value
quality	31.85	2.3e-21
relevance	29.38	5.8e-20
coherence	4.68	0.001

Table 3
F statistic and p_values

Table 3 confirms that for each qrels file, the means of the NDCG distributions are not all equal among different runs, as boxplots anticipated (as p_values are less than 0.01 we reject the null hypothesis that all means are equal). To better understand pairwise differences, we show the p_values after computing pairwise t-tests:

	1	2	3	4	5
1	nan	0.0000	0.9776	0.9305	0.8830
2	nan	nan	0.0000	0.0000	0.0000
3	nan	nan	nan	0.9012	0.8982
4	nan	nan	nan	nan	0.8040
5	nan	nan	nan	nan	nan

Figure 5: p_values pairwise t-tests, quality qrels

	1	2	3	4	5
1	nan	0.0000	0.3380	0.5126	0.2975
2	nan	nan	0.0000	0.0000	0.0000
3	nan	nan	nan	0.7380	0.9153
4	nan	nan	nan	nan	0.6629
5	nan	nan	nan	nan	nan

Figure 6: p_values pairwise t-tests, relevance qrels

	1	2	3	4	5
1	nan	0.0005	0.5625	0.8745	0.4986
2	nan	nan	0.0020	0.0004	0.0020
3	nan	nan	nan	0.6556	0.9292
4	nan	nan	nan	nan	0.5845
5	nan	nan	nan	nan	nan

Figure 7: p_values pairwise t-tests, coherence qrels

So run 2 is in every case worse than all the other runs since the p_values are < 0.01, as we suspected. We can't inference further differences between runs from the tables. Note that run 2 and run 3 use the same analyzer, whereas the difference relies only on the type of searcher. So we say for sure that searcher 3.2.2 makes a very substantial impact on the final result w.r.t. 3.1.3. We can also point out that even if run 1 and run 5 use very different first phases (and 3.2.1 as second phase of the system), they were not statistically different. Perhaps this suggests that another interesting test to execute would be to run the same first phase as in run 2 and run 3, but in combination with sentences searcher. This would have enabled us to see the difference in performance given by only 3.2.1 with respect to the other 2 sentence selection methods.

7. Conclusions and Future Work

Comparing our results on document retrieval with last year's overview paper [8] allowed us to ensure that the system we were developing had a satisfying enough performance, and we could proceed with sentence selection and retrieval. With the release of qrels, we will be able to discern which of the multiple approaches worked better and to fine-tune parameters (e.g., similarity functions, word2vec training parameters and WordNet max synonyms).

An evaluation measure for Touché 2022 Task 1 is the coherence between sentences; we may want to further improve it by means of character n-grams, word n-grams, and skip-grams that overcome data sparsity. Since sentences, especially conclusions, are often short phrases and should be meaningful claims, finding matching text between them may lead to an increase of syntactic similarity, and consequently in coherence. For this reason, we are interested to see how the just mentioned techniques could perform.

In the run file we have to print the stance w.r.t the query and, as can be seen from the args.me search engine's API[8], the one we have available in the premises field is towards conclusion. So, we don't have an immediate information if a retrieved argument is "pro" or "con" the original query, and we may try to address this issue through sentiment analysis. It's the application of natural language processing (NLP) to understand if the explicit or implicit opinion in a sentence is positive, negative or neutral. Through the use of *Part of Speech (PoS)* tagging, for example ApacheNLP [9], and human-validated sentiment lexicon we can define a polarity score for the premises and compare it with the stance towards conclusion to derive the final returned stance. Since "each sentence in the pair must ideally be the most representative/most important of its corresponding argument", sentiment analysis may also lead to an improvement in this regard.

As a further development we would like to retrieve sentences from different documents to encourage diversity, as suggested by Touché's organizers. To ensure coherence between sentences and query text, we may want to identify keywords, weight them and try synonym expansion with terms from the description or narrative field of the topic file.

[8] https://www.args.me/api-en.html
[9] https://opennlp.apache.org/

References

[1] 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.

[2] Y. Ajjour, H. Wachsmuth, J. Kiesel, M. Potthast, M. Hagen, B. Stein, Data Acquisition for Argument Search: The args.me corpus, in: C. Benzmüller, H. Stuckenschmidt (Eds.), 42nd German Conference on Artificial Intelligence (KI 2019), Springer, Berlin Heidelberg New York, 2019, pp. 48–59. doi:10.1007/978-3-030-30179-8_4.

[3] M. S. Shahshahani, J. Kamps, Argument retrieval from web, in: International Conference of the Cross-Language Evaluation Forum for European Languages, Springer, 2020, pp. 75–81.

[4] M. Samadi, P. Talukdar, M. Veloso, M. Blum, Claimeval: Integrated and flexible framework for claim evaluation using credibility of sources, in: Thirtieth AAAI Conference on Artificial Intelligence, 2016.

[5] G. Gorrell, E. Kochkina, M. Liakata, A. Aker, A. Zubiaga, K. Bontcheva, L. Derczynski, SemEval-2019 task 7: RumourEval, determining rumour veracity and support for rumours, in: Proceedings of the 13th International Workshop on Semantic Evaluation, Association for Computational Linguistics, Minneapolis, Minnesota, USA, 2019, pp. 845–854. URL: https://aclanthology.org/S19-2147. doi:10.18653/v1/S19-2147.

[6] H. Hüning, L. Mechtenberg, S. Wang, Detecting arguments and their positions in experimental communication data, Available at SSRN 4052402 (2022).

[7] J. Lawrence, C. Reed, Argument mining: A survey, Computational Linguistics 45 (2020) 765–818.

[8] 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, in: K. Candan, B. Ionescu, L. Goeuriot, H. Müller, A. Joly, M. Maistro, F. Piroi, G. Faggioli, N. Ferro (Eds.), Experimental IR Meets Multilinguality, Multimodality, and Interaction. 12th International Conference of the CLEF Association (CLEF 2021), volume 12880 of *Lecture Notes in Computer Science*, Springer, Berlin Heidelberg New York, 2021, pp. 450–467. URL: https://link.springer.com/chapter/10.1007/978-3-030-85251-1_28. doi:10.1007/978-3-030-85251-1_28.

[9] H. K. Azad, A. Deepak, Query expansion techniques for information retrieval: a survey, Information Processing & Management 56 (2019) 1698–1735.

[10] B. Stein, Y. Ajjour, R. El Baff, K. Al-Khatib, P. Cimiano, H. Wachsmuth, Same side stance classification, Preprint (2021).

[11] D. Küçük, F. Can, A tutorial on stance detection, in: Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining, 2022, pp. 1626–1628.

[12] E. M. Alshari, A. Azman, S. Doraisamy, N. Mustapha, M. Alksher, Senti2vec: An effective feature extraction technique for sentiment analysis based on word2vec, Malaysian Journal of Computer Science 33 (2020) 240–251.

[13] S. Baccianella, A. Esuli, F. Sebastiani, Sentiwordnet 3.0: An enhanced lexical resource

for sentiment analysis and opinion mining, in: Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10), 2010.
[14] A. ALDayel, W. Magdy, Stance detection on social media: State of the art and trends, Information Processing & Management 58 (2021) 102597.
[15] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, 2013. URL: https://arxiv.org/abs/1301.3781. doi:10.48550/ARXIV.1301.3781.