Aldo Nadi at Touché 2022: Argument Retrieval for Comparative Questions

Maria Aba[1], Munzer Azra[1], Marco Gallo[1], Odai Mohammad[1], Ivan Piacere[1], Giacomo Virginio[1] and Nicola Ferro[1]

[1]University of Padua, Italy

Abstract

In this paper we present the information retrieval system we developed for the 2022 Touché @ CLEF Task 2 evaluation campaign. The participation in the task is performed as a student group project conducted in the Search Engines course a.y. 2021/2022 at the Computer Engineering and Data Science master degrees at University of Padua.

This tasks' aim is to create systems that are able to retrieve documents that compare two options, e.g. which is the best pet between a dog and a cat.

Here we describe the architecture of our system, we list the software and hardware resources we made use of, we discuss the results obtained using different configurations and finally we present improvements which could be applied to our system to enhance its performance.

Keywords

Information retrieval, Comparative questions, Lucene

1. Introduction

Before the era of the internet, information storage and retrieval systems were mostly used by professionals for medical research, in libraries, by governmental organizations, and archives. Therefore, access to such information was a hard process especially for non-search experts. Recently, with the fast increase in the number of data and information available online, the importance of search engines grew rapidly. Nowadays, people use search engines to locate and buy goods, choose a vacation destination, select a medical treatment, etc. Search engines transitioned from being searchers' tools for information to tools for building opinions and making major decisions. All of these aspects, when considered together, make retrieval systems a need for impacting the industry and improving the field of information retrieval.

This paper is structured as follows: Section 2 presents related work; Section 3 describes our approach; Section 4 explains our experimental setup; Section 5 discusses our main findings in the model selection process; Section 6 discusses the results and analysis of our runs; finally,

Section 7 draws some conclusions and outlooks for future work.

2. Related Work

The packages mentioned in 3.1 and 3.3 are obtained expanding from a baseline built on the TIPSTER collection, during lessons of the "Search Engines" course, University of Padua. Information about the course are available online: [1]

3. Methodology

The following is the class diagram for our implementation: Figure 1
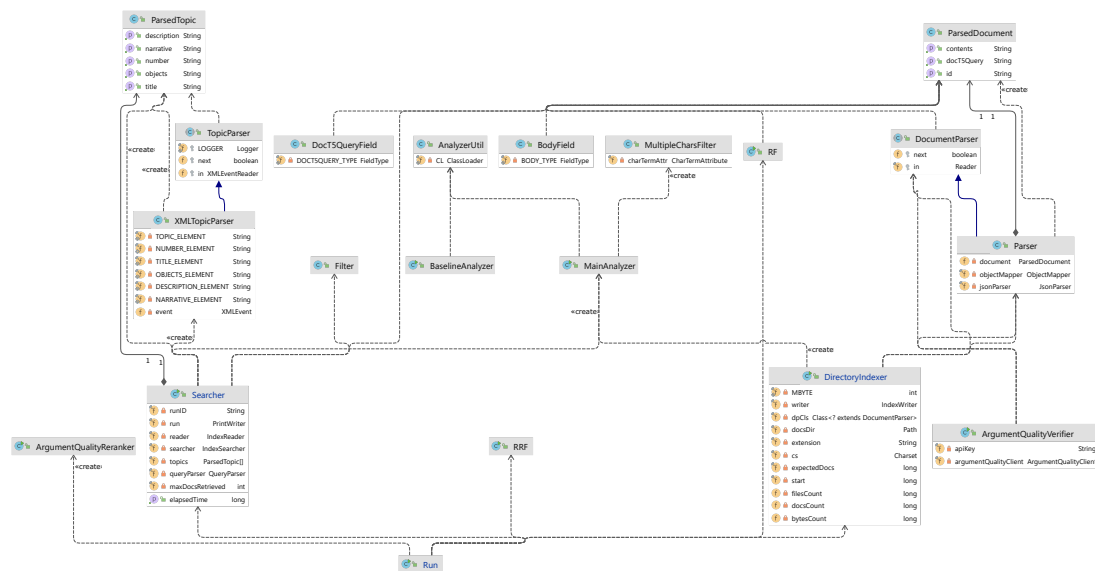
Figure 1: Class diagram of the project

The developed Java system is divided into the following packages, each package representing a stage: Parse, Analyze, Index, Filter, Search, RF, RRF, and Argument Quality.

3.1 Parse, Analyze, Index

These packages are in charge of the database creation and to prepare topics.

The documents in the DocT5Query expanded corpus are parsed, their text field analyzed (with the possibility of using different custom analyzers) and then indexed with the fields: ID, Body and DocT5Query.

The topics are also parsed so that the fields number, title and objects can be used in the search using Lucene, with the latter two also analyzed.

3.2   Filter

This class allows to extract the strings from the object field in the topics and return a Boolean-Query.Builder object, which can be later consumed by the search method by adding it as a MUST clause in the search, to only retrieve documents that present all terms contained in the object fields.

3.3   Search

This packed is responsible responsible for:

1. Calling the Paese and Analyze packages to retrieve and preparing the topics for the search.
2. Defining which type of comparison to perform between topics and documents, that can be chosen by changing the similarity function.
3. Defining how to use topics in the search.
   The topics titles are used search by similarity with a SHOULD clause, it possible to also assign weights to the different fields of the documents among which to search, or to select just one of the two fields (Contents and DocT5Query), and the MUST clause described in the Filter class can be added.
4. Writing the results on a file

3.4   RF

RF is a customized class with the goal of performing a search using explicit relevance feedback to perform query expansion.

RF functions in a similar way to the Searcher class, with the exception of building the query used in the searching using the tokens present in relevant documents, instead of using the terms in title field of the topics file.

The class collects all docID and relevance of relevant documents in the *qrels* file.

The tokens and their frequency in the relevant documents are retrieved by searching the document by docID and iterating through its termvector.

The tokens used in the search are boosted by their frequency in the document multiplied by the square of the relevance score.

Relevance Feedback is standardly based on the Rocchio Algorithm. [1] The formula for the Rocchio Algorithm is:

$$\overrightarrow{Q_m} = \left(a \cdot \overrightarrow{Q_O} \right) + \left(b \cdot \frac{1}{|D_r|} \cdot \sum_{\overrightarrow{D_j} \in D_r} \overrightarrow{D_j} \right) - \left(c \cdot \frac{1}{|D_{nr}|} \cdot \sum_{\overrightarrow{D_k} \in D_{nr}} \overrightarrow{D_k} \right)$$

where $\overrightarrow{Q_m}$ is the modified query vector, $\overrightarrow{Q_O}$ is the original query vector, $\overrightarrow{D_i}$ is the document vector for the ith document, D_r is the set of relevant documents, D_{nr} is the set of non-relevant documents and a, b and c are weight parameters.

In our case the parameters used are 0, 1, 0.

Rocchio algorithm is however defined for working with binary relevance, since this collection uses multi-graded relevance, our version of RF is customized to take into account the different relevance scores used (0 to 3).

The custom formula we used is:

$$\overrightarrow{Q_m} = k_i^2 \cdot \frac{1}{|D_r|} \cdot \sum_{\overrightarrow{D_i} \in D_r} \overrightarrow{D_i}$$

where $\overrightarrow{Q_m}$ is the modified query vector, $\overrightarrow{D_i}$ is the document vector for the ith document, D_r is the set of relevant documents, and k_i is the relevance score of the ith document.

In this work a total of 491 relevant documents have been used to perform Relevance Feedback.

The results of the search are then outputted as a standard run file.

3.5 RRF

This package contains a single class, also called RRF.

RRF.java is a customized class with the goal of performing using Reciprocal Ranking Fusion [2] to fuse the results of different runs in a single one.

RRF takes in imput a directory path and performs RRF using all the runs in .txt documents inside that directory.

For each documents and for each topic the documents and their respective ranking are collected.

Then document receive a new scoring using the RRF formula.

Given a set of documents D and a set of rankings R for the documents, the formula for RRF is:

$$RRF score(d \in D) = \sum_{r \in R} \frac{1}{k + r(d)}$$

where k is a fixed number, in this case k is set to 30.

Then, for each topic, documents are ranked (and ordered) based on their RRF score.

The results of the search are then outputted as a standard run file.

3.6 Argument quality

We decided to make use of IBM Project Debater API.

Project Debater is an AI system used to perform various tasks about debating at a human level. IBM makes freely available, for research purposes, some services based on this system through an API. [3]

We were interested in the argument quality service of the API. It accepts a couple of strings labeled as Sentence and Topic, and it returns a float score in the range 0-1 based on the relevance of the sentence for the topic and on the quality of the sentence as a text, which means how good it is written.

Since the rest of our system is designed to already score documents based on the relevance to the topic, we now just wanted to evaluate the text quality. In order to do so, for each document

in the collection we decided to send Sentence-Topic pairs in which the Sentence was the body of the document and the Topic was an empty string.

We coded the ArgumentQualityVerifier class which evaluates the written quality of each document by using the API and then saves the scores to a file.

Then we had to use the obtained scores to rerank the results of the search saved in a run file. So we defined the ArgumentQualityReranker class which:

1. loads the quality scores of all the documents from the file into a Map object
2. iterates over the lines of the old run file and for each: multiplies the old score by the one assigned by Project Debater API and saves the object representing the new line to a list
3. sorts the list of new lines by topic number and score and writes them on a new run file

4. Experimental Setup

4.1 Collections

Some of the collections used throughout the process of system development were the ones provided by CLEF for the Touché 2022 edition, accessible from Task 2's site. Those include:

- topics-task2.xml which contains the topics.
- The original version of passages.jsonl which contains the documents.
- DocT5Query expanded version of passages.jsonl[2] which contains the documents expanded with queries generated using DocT5Query. [4]

Other collections are:

- Historical stoplists: lucene, smart and terrier;
- Custom stoplists:
    - Kueristop - Stoplist formed by the 400 most concurrent term in the Contents field of the document collection;
    - Kueristopv2 - Subset of kueristop, obtained by removing from it terms appearing in the Objects field of the topics, except for the very general terms also appearing in lucene stoplist ("in" and "the").
- Sentence quality - file containing, for each document in the document Collection, the pairs of docIds and the score obtained by that document as explained in 3.6.

---

[2]This collection was provided by Team Princess Knight that parteciped in Touche, the corpus can be found at: https://www.tira.io/t/expanded-passages-for-the-touche-22-task-2-argument-retrieval-for-comparative-questions/578

4.2 Evaluation measures

The evaluation measure used is Normalized Discounted Cumulative Gain at depth 5, NDCG@5 in short. [5]

It is the evaluation measure used by Touché to officially evaluate runs.

NDCG@k is calculated as follows:

$$NDCG@k = \frac{DCG@k}{iDCG@k}$$

where

$$DCG@k = \sum_{i=1}^{k} \frac{relevance_i}{log_2(i+1)}$$

and iDCG@k is the ideal DCG@k, meaning the DCG@k for documents ordered by relevance, highest to lowest.

4.3 Git repository

The project's development can be found in the following link to its Git repository [3]

4.4 Hardware

The specifications of the computer used to perform the runs are the following:

OS Windows 10 Home 21H2 x64

CPU AMD Ryzen 5 1600 @ 3.9GHz

RAM 16GB 3000mhz cl16

GPU Nvidia GTX 1060 6GB

HDD 2TB 7200RPM

5. Model Selection

The conventional and ideal approach when evaluating the performance of the runs would have been to use last year's test collection. [6]

However, since we did not have access to last year's corpus we have decided to use this year's test collection to evaluate our systems, using a *qrels* file containing relevance feedback manually performed by us.

The *qrels* file has been built by gathering, for each of the runs performed, the top 5 ranked documents for each topic.

---

[3]https://bitbucket.org/upd-dei-stud-prj/seupd2122-kueri/src/master/

Table 1
NDCG@5 and setup for single runs

#	NDCG@5	num_q	RF	Stoplist	Filter	Stemmer	Similarity	Weights	Reranking
1	0.3830	50	False	lucene	False	None	BM25	[1,1]	False
2	0.3756	50	False	lucene	False	None	LMD	[1,1]	False
3	0.3313	50	False	lucene	False	None	TFIDF	[1,1]	False
4	0.4140	50	False	smart	False	None	BM25	[1,1]	False
5	0.4258	50	False	terrier	False	None	BM25	[1,1]	False
6	0.4366	50	False	kueristop	False	None	BM25	[1,1]	False
7	0.4548	50	False	kueristopv2	False	None	BM25	[1,1]	False
8	0.4015	48	False	lucene	True	None	BM25	[1,1]	False
9	0.4759	41	False	kueristop	True	None	BM25	[1,1]	False
10	0.4823	48	False	kueristopv2	True	None	BM25	[1,1]	False
11	0.2634	50	False	kueristopv2	False	None	BM25	[0,1]	False
12	0.3654	50	False	kueristopv2	False	None	BM25	[1,0]	False
13	0.4525	50	False	kueristopv2	False	None	BM25	[1,2]	False
14	0.4674	50	False	kueristopv2	False	None	BM25	[2,1]	False
15	0.4873	50	False	kueristopv2	False	Porter	BM25	[1,1]	False
16	0.8549	50	True	kueristopv2	False	False	BM25	[1,1]	False
17	0.8552	50	True	kueristopv2	False	Porter	BM25	[1,1]	False
18	0.5867	48	False	kueristopv2	True	None	BM25	[1,1]	True
19	0.5392	50	False	kueristopv2	False	None	BM25	[2,1]	True
20	0.5714	50	False	kueristopv2	False	Porter	BM25	[1,1]	True
21	0.8606	50	True	kueristopv2	False	False	BM25	[1,1]	True
22	0.8323	50	True	kueristopv2	False	Porter	BM25	[1,1]	True

Table 2
NDCG@5 and setup for rrf runs

#	# fused	NDCG@5	Reranking
23	10,14,15,16,17	0.7521	False
24	10,14,15,16,17	0.7450	True

The runs' performance has been evaluated using *trec_eval*, the key measures considered are *NDCG@5*, the official measure used by CLEF to rank runs, and *num_q*, the number of topics retrieved (since some runs retrieved no documents for some of the topics).

All the runs, their characteristics and key measures are reported in Table 1 and 2. The five runs with their number in bold are the five submitted runs.

All the runs are performed on indexes obtained using Standard tokenizer and Lowercase filter, except for indexes used in runs obtained using Relevance Feedback, which use Letter tokenizer instead; this is because some of the tokens obtained using standard tokenizer were written in a format that caused errors when used as query (e.g. "text:text:text" would be a token that caused errors).

5.1 Retrieval Similarity Model

The runs 1 to 3 compare BM25 (utilizing lucene's default parameters), Dirichlet and TFIDF Similarity as scoring functions, using lucene stoplist. The run using BM25 was the best performer, so we decided to use this Similarity for all the other experiments.

5.2 Stoplists

Runs 1 and 4 to 7 compare different stoplists, in particular we compared lucene, smart and terrier stoplists and our own custom stoplists kueristop and kueristopv2; the results show that among the "generic" stoplists the larger ones have a bigger impact, but custom stoplists bring to even better improvements, with kueristopv2 being the best.

5.3 Filter

We then wanted to assess the impact of filtering the runs by all the terms in the object field. Runs 8, 9 and 10 are performed adding the filter to the setup of runs 1, 6 and 7. Run 9 only retrieved documents for 41 topics, as 9 topics contain, in the ojects field, terms that are in the stoplist (and therefore are in the index); runs 8 and 10 retrieve documents for 48 queries, because lucene and kueristopv2 contain the terms "the" and "in", which again are in the objects field for two queries.

 The runs with filtering have a better *NDCG@5* score compared to runs without, however they retrieve less topics. Retrieving no documents for some topics make us assess these runs as worse performing compared to the ones without filtering. Moreover the improvement in *NDCG@5* score could be caused in part by the lack of these topics, as the system could have worse performance for these topics compared to the others. Despite having worse results when taken singularly, runs using filtering can be used to improve other runs by using *RRF*.

5.4 Field Weight

Runs 11 to 14 use the same setup as the current best performing run, 7, changing the weight of Contents and DocT5Query fields respectively. When searching on a single field (weight 0 on the other field) the score is much worse, increasing the weight of DocT5Query field slightly worsens the score, increasing the weight of Contents field instead improve the score.

5.5 Stemmer

Run 15 adds to the setup of run 7 a stemmer, specifically Porter stemmer; this addition brings to a good improvement in performance.

5.6 RF

Runs 16 and 17 instead are performed using Relevance Feedback, respectively without stemmer and with Porter stemmer; These runs have an NDGC@5 score incredibly higher than the previous ones, this however is due to using the same collection, and in particular the same *qrels*, to obtain the RF runs and to score its performance.

To have a more reliable assessment of performance we could have done the search on a index built removing documents present in the qrels file. However, while this would have prevented the overfitting problem, we still couldn't have directly compared results to other runs; in fact, the documents in the *qrels* file, being the top documents retrieved, should be the most relevant, which mean we should have expected worse results by the runs performed when removing the documents from the collection.

5.7   RRF

The first *rrf* run is obtained fusing a mixture of well performing and slightly different runs: 10, 14, 15, 16 and 17. It presents a very good NDCG@5 score, but since it uses *RF* runs the score is not reliable as these runs also may contain overfitting.

5.8   Reranking

Runs 18 to 22 and the second *rrf* run are obtained by applying reranking to the runs above (10, 14, 15, 16 and 17 and their fusion). Reranking has been performed by multiplying documents' scores in the runs with their respective Argument quality score, as described in 3.6.

Comparing to their non-reranked respectives we can see that results on RF and rrf runs are mixed, but again not the most reliable because of previous overfitting; on the other three runs instead reranking offers a really great improvement in performance.

6.   Results and Analysis

6.1   Results

The runs' performance are evaluated on this year's relevance qrels, provided by CLEF, built performing top-5 pooling on the runs delivered by all participants. [7]

In table 3 we show, for each run, the NDCG@5 score obtained during model selection and the respective NDCG@5 score obtained with CLEF's relevance qrels.

The scores are close to the ones obtained in model selection and all choices done are confirmed by the final results.

The only runs that differ much from the ones in the model selection are, as expected due to the mentioned overfitting, the ones that use Relevance Feedback. These runs still have a better score than other run obtained before reranking, but they don't differ from them as much as they did earlier.

The runs obtained through RRF also suffer a decrease in score, also due to the previous partial overfitting deriving from RF runs.

As expected reranking improved the performance of all runs, included RF and RRF runs.

The only result hinted in the model selection that we didn't expect to turn out to be true was that Porter stemmer slightly worsened the score when applied to RF runs.

The best performing run is run 24, the reranked RRF run.

Table 3
Model and CLEF's qrels scores

#	selection NDCG@5	Final NDCG@5
1	0.3830	0.3828
2	0.3756	0.3688
3	0.3313	0.2937
4	0.4140	0.4497
5	0.4258	0.4461
6	0.4366	0.4746
7	0.4548	0.4896
8	0.4015	0.4376
9	0.4759	0.5226
10	0.4823	0.5042
11	0.2634	0.2289
12	0.3654	0.4088
13	0.4525	0.4535
14	0.4674	0.4939
15	0.4873	0.5466
16	0.8549	0.6098
17	0.8552	0.6036
18	0.5867	0.5812
19	0.5392	0.5772
20	0.5714	0.6362
21	0.8606	0.6954
22	0.8323	0.6669
23	0.7521	0.6681
24	0.7450	0.7089

6.2 Statistical Analysis

All the following statistical analysis have been obtained using CLEF's relevance qrels.

In the analysis, to produce better results, when a run retrieves no documents for a topic the NDCG@5 score is set to 0, while earlier that topic was not considered in the average NDCG@5 in the run, therefore results are worse than to the ones observed earlier for those runs (8, 9, 10, 18).

First we wanted to check if the runs were significantly different between them, in order to do so we have used Tukey's HSD test [8], with $\alpha = 0.05$: Figure 2

In particular run 24 is highlighted, to show how it significantly differs from all runs not using RF or reranking (runs 1 to 15)

Then we produced a boxplot graph showing, for each run, the NDCG@5 score of each topic on the Y axis, ordered by average NDCG@5: Figure 3

All runs have a very similar interquartile range and all runs, except run 19, have a score equal to 0 for at least one topic.

Following these results we also got interested in finding out the difference by topics, to see what type of topic we had a poor performance on: Figure 4
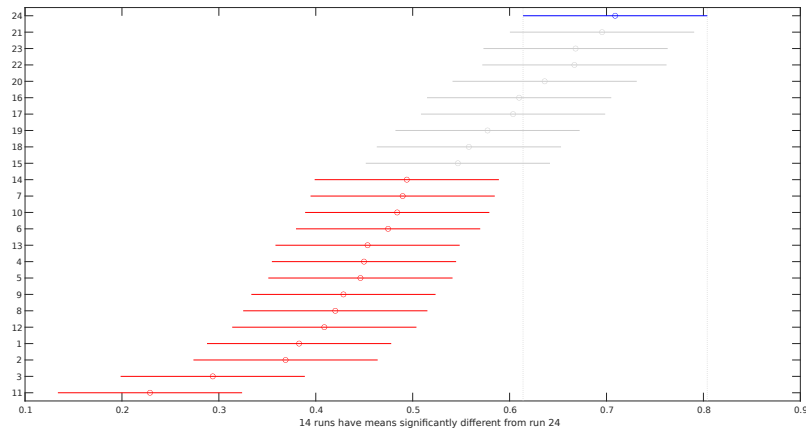
Figure 2: Multiple comparison of Tukey's HSD test
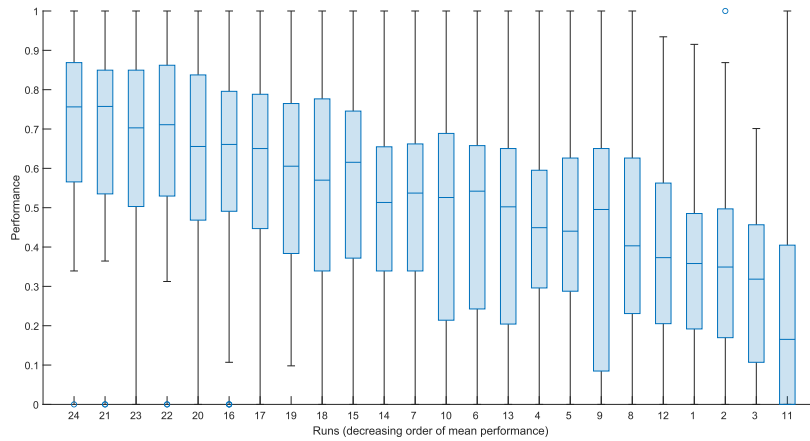

Figure 3: Run's Boxplot ordered by NDCG@5

The run got their worse results on topics 43, 86 and 77, which are respectively:

- Should I prefer a Leica camera over Nikon for portrait photographs?
- I am planning to buy sneakers: Which are better, Adidas or Nike?
- Is it healthier to bake than to fry food?

The problem we found in particular with these topics is, for the first two, that a lot of documents retrieved were ads, and for the last one that a lot of documents retrieved were just recipes that bake or fry food.

Due to these results we believe in future work it might be useful to add to the search keywords or shingles offering comparison (e.g. "versus", "compared to", "against"), since comparison between two items is intrinsic to the task.
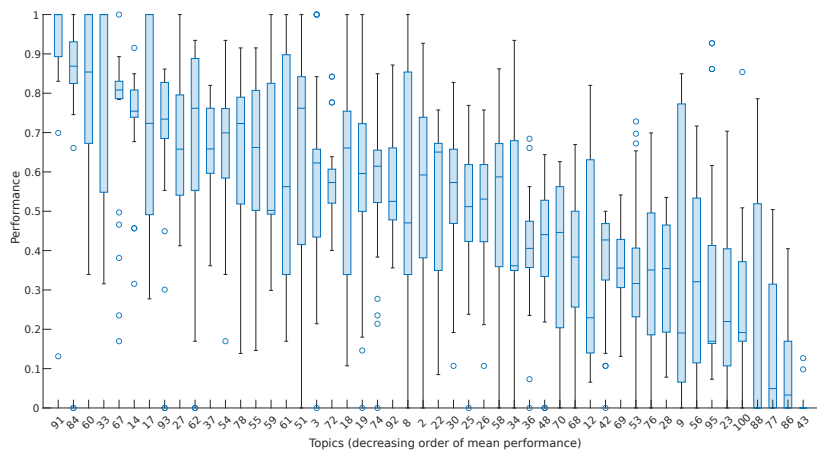
Figure 4: Topics's Boxplot ordered by NDCG@5

We also decided to check the difference in performance in different topics between our best run and the third and second best, again to check the reason for the dip in performance in specific topics: Figure 5 Figure 6
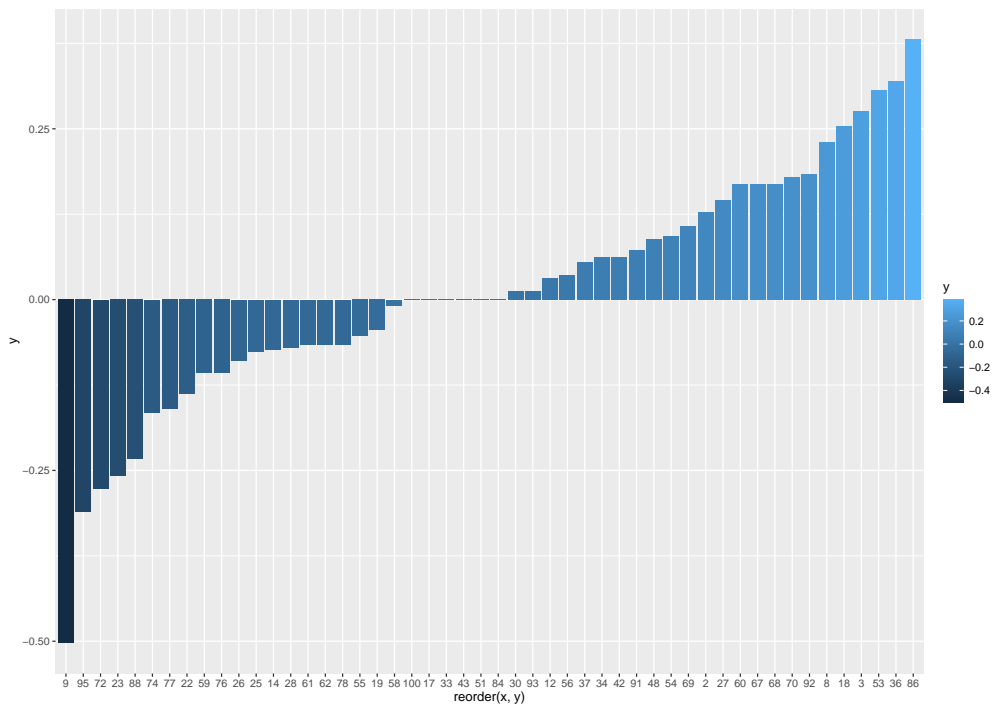

Figure 5: Difference in performance by topic in runs 24 and 21

When comparing run 24 to run 21 the performance is noticeably worse, with a difference of

0.5, for topic 9: "Why is Linux better than Windows?"

This top is however is one of the worse performing topic among all runs and at the same time has the largest interquartile range.

Going more in depth we find, rather than the weaknesses of run 24, a proof to the strength of Relevance Feedback: in fact, in this specific topic in general the documents retrieved often display people talking about one of the two objects, Relevance Feedback runs instead excel because they look for keywords, that are very often used when comparing the two objects, like for example "price", "safety", "open" and "source".
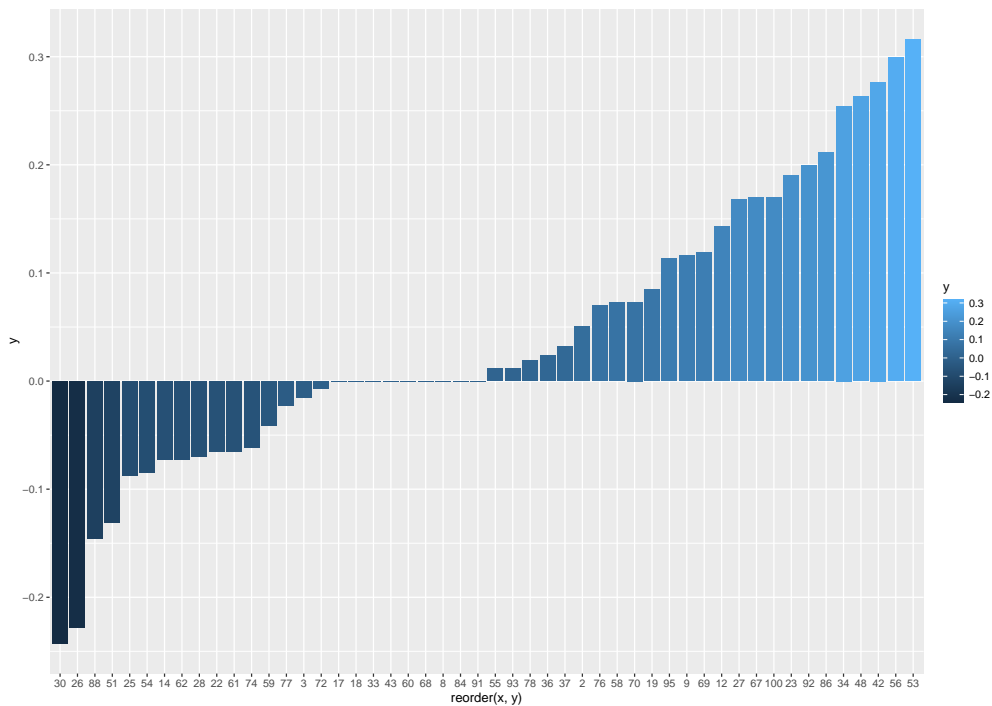

Figure 6: Difference in performance by topic in runs 24 and 23

The comparison between run 24 and 23 is particularly interesting, since the first is the reranked version of the second. A noticeable dip in performance from run 23 to run 24 can be seen in topics 30 and 26:

- Should I buy an Xbox or a PlayStation?
- Which is a better vehicle: BMW or Audi?

We decided to go in depth to find out the reason of the worse performance in topic 30 (Xbox vs. Playstation) by manually checking the top-5 document retrieved for each and their relevance score in qrels.

The main reason for the difference is that most of the relevant document for run 23 are formed mainly by short ads (in the format "item on sale - price"), but also contained a very short

phrase that was relevant to the topic. These document when reranking suffer a big penalty due to their low sentence quality.

Instead, in run 24 we found a document that is uniquely an ad (which was unexpected as we thought that reranking with sentence quality was a very good way to push ads down in the ranking), however this document was a well written ad, consisting of a company advertising their business that sold consoles, therefore it's sentence quality score is high.

A problem we found in this in depth analysis is that some documents (e.g. clueweb12-1810wb-39-31830___3, clueweb12-1808wb-28-21892___11) were poorly scored, since these documents were comprised of only ads, containing no relevant information at all to the topic, but still were scored as partially relevant. Finding these two blatant mistakes in only 8 documents manually checked (two document are present in both runs) raises concerns on the reliability of the relevance scores delivered by CLEF.

7. Conclusions and Future Work

We managed to effectively select the search engine model, that offered results close to the ones obtained with the official qrels, except for the already expected difference due to overfittin in RF.

We managed to improve substantially the performance of the runs compare to the initial lucene baseline, with an increase in score of over 85% when considering our best performing run.

The greatest impact comes from relevance feedback, but reranking and a stoplist customized to our corpus also offered noticeable improvements.

This is remarkable also because, due to the lack of access to last year's corpus, it wasn't possible for us to perform any fine-tuning.

Having access to such test collections would allow us for example to fine tune BM25 parameters, the field weights, the boosts for terms in RF, we could experiment with many more stoplists and stemmers. As an example, a run implementing Porter stemmer (or a different stemmer), fine tuned weights with the Contents field having more weight than the DocT5Query one would probably best all the other single runs, but the extra time it took us to also manually assess documents proved to be a strong limiting factor in the expansion of our experiments.

In future works it would be interesting to, as mentioned, add to the search terms used to compare objects, and experiment with other "classic" method, for example using shingles, but mostly with machine learning and deeplearning techniques, that have become the standard in the last decade of information retrieval.

It would also be interesting having the chance to tackle a similarly built task, but with the change to work with data in formats different than full-text, with the addition of metadata (for example in this case, since the corpus was created crawling the web, having access to metadata from the webpages would have presented new opportunities, like individuating ads).

References

[1] J. J. Rocchio, Relevance feeback in information retrieval (1965). URL: http://sigir.org/files/museum/pub-08/XXIII-1.pdf.

[2] G. V. Cormack, C. L. A. Clarke, S. Büttcher, Reciprocal rank fusion outperforms condorcet and individual rank learning methods (2009). URL: https://plg.uwaterloo.ca/~gvcormac/cormacksigir09-rrf.pdf.

[3] Project debater for academic use, n.d. URL: https://early-access-program.debater.res.ibm.com/academic_use.

[4] R. F. Nogueira, W. Yang, J. Lin, K. Cho, Document expansion by query prediction, CoRR abs/1904.08375 (2019). URL: http://arxiv.org/abs/1904.08375. arXiv:1904.08375.

[5] K. Järvelin, J. Kekäläinen, Cumulated gain-based evaluation of IR techniques, ACM Trans. Inf. Syst. 20 (2002) 422–446. URL: http://doi.acm.org/10.1145/582415.582418. doi:10.1145/582415.582418.

[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, 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.

[7] 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.

[8] Tuckey's method, n.d. URL: https://www.itl.nist.gov/div898/handbook/prc/section4/prc471.htm.