Information Retrieval Assignment 2 Group 9 Report

PALLAVIT AGGARWAL, SHUNCHENG CAI, ISOBEL MAHON, YATIN MOHAN VAID, and STEFAN HUTANU

In this document we describe our approach to creating an ad-hoc search engine for assignment 2 of CS7IS3. Building on our project from assignment 1, we further improve our search engine by building better search queries,

CCS Concepts: • **Information systems** → *Novelty in information retrieval.*

Additional Key Words and Phrases: information retrieval, indexing, querying, sigir, retrieval models, analyzers, ranking, Financial Times, LA Times

1 INTRODUCTION

The goal of this project was to create and optimise a search engine that could index and search a dataset of news articles. The news articles used are from a variety of sources; the Financial Times Limited, the Federal Register, the Foreign Broadcast Information Service, and the Los Angeles Times. The dataset files are written in Standard Generalized Markup Language.

We leveraged our experience from Assignment 1 to get our base search engine set up quickly and made improvements from there. We met frequently, both online and in person, to discuss our progress, difficulties we were having, and plan what to work on before our next meeting. These meetings helped keep us all on the same page about how the project was going, and let us work together more effectively.

2 RELATED WORK

A quick survey online for techniques to optimise Lucene's retrival process outlines the fundamentals that drive the search like - a) Ranking and scoring algorithms and tweaking the default values to penalize certain aspects of the narrative [1][2][3][4] or employ composite scoring techniques (we discuss further in Section 4 of this report) . b) Query expansion using word disambiguation or thesauri to create synonyms and increase the recall and precision [5]. c) Using SynonymGraphFilter and WordDelimiterGraphFilter along with MultiPhraseQuery [6][7]. d) Using Stemming, Filters and Tokenizers experimentally to understand which one gives us the best result [8]. e) Optimizing Queries and using boosts and relavant words to improve relavance scoring.
The papers mentioned here were helpful in inspiring us to adopt numerous approaches yeilding to a final score of 0.34 MAP score. Some techniques, however, were not fully implemented or included in the report due to lack of documentation online and complexity constraints, but we did try them out to the best our abilities and they did give us newer ideas.

3 SETUP AND INITIAL RESULTS

We began this project with a copy of our first assignment. We compared each of our scores and started adapting the one with the best results. Since the focus of this assignment was to investigate different retrieval models, we were able to re-use a lot of the same code to structure, interact with, and evaluate our new search engine. The next step was to build an index with documents created from the four datasets for this assignment.

Using a library called JSoup, we were able to parse each file from each dataset and pick out the relevant

Authors' address: Pallavit Aggarwal; Shuncheng Cai; Isobel Mahon; Yatin Mohan Vaid; Stefan Hutanu.

XML tags for our search engine. We initially only used `<DOCNO>`, `<TI>`/`<HEADLINE>` (the Title tag is named TI in some datasets and HEADLINE in others), and `<TEXT>` as these fields appeared to be the most relevant for our search engine. We tried using `<DATE>` as well, but found that not all documents are dated, so we couldn't reliably use this information. We put the contents of each of these XML tags into Lucene documents. To create a searchable index, we parse all the documents in the corpus and configure our index to use BM25 Similarity and a Custom Analyzer. For the custom analyzer, we extended the Analyzer class and applied the following filters : (i) EnglishPossessiveFilter, (ii) LowerCaseFilter, (iii) TrimFilter, (iv) StopFilter and (v) PorterStemFilter.

Finally, we created a topic file parser. This parser read each line of the topics file and used a finite state machine to read the topic number, narrative, and description. Then, we apply the same set of pre-processing operations on these queries as applied on the documents during indexing. With the basic index and parser, we used the topic narratives to search the index. However, our results file had an unexpectedly low mean average precision (MAP) of around 2%.

4 EXPERIMENTATION AND IMPROVEMENTS

4.1 Changing MultiFieldQueryParser fields

One of the first things we tried was making the scores outputs binary, as the relevancies in the qrels file were binary, and we weren't sure how exactly trec eval compared the two. At this time, we were getting extremely low scores, which we guessed might be because of the discrepancy between the two ways of expressing relevancy. It later became clear that the very low scores were because the field names we had entered in the MultiFieldQueryParser were incorrect, and making the output scores binary only made our mean precision worse. By correcting this mistake, our MAP score shot up from 2% to 13.87%

4.2 Lucene document fields and weights

Our search engine used Lucene's MultiFieldQueryParser which constructs queries to search multiple fields. We use a BoostMap to selectively boost the importance of fields in the documents. We assigned a weight of 5f to document title and a weight of 17f to the document text. This resulted in the MAP score rising from 13.87% to 14.36%

4.3 Adding Topic title and description in search query

Until now, we had just used topic narrative for constructing our search query. We changed our search query to also include topic title and description. Adding Description to the query resulted in a MAP score of 24.01% and adding Title to this further helped in improving the MAP score from 24.01% to 30.24%

4.4 Boolean "Must" for rarest query term

One thing we tried to ensure only relevant documents were returned was to apply a boolean "Occur.MUST" to the query term with the lowest inverse document frequency. Unfortunately, this brought the MAP score down to 16.14%.

4.5 Designing search queries

We knew that for this project, our queries would have a big impact on the quality of our results. Looking through the topics file, we found that many of the descriptions mentioned what was relevant or not for each topic. Using this knowledge, we wrote code to identify which clauses of the sentences in the topic descriptions referred to relevant or non-relevant terms. We then modified our search query to be a BooleanQuery and set the relevant terms to use Lucene's `BooleanClause.Occur.SHOULD` and the non-relevant terms to use `BooleanClause.Occur.MUST_NOT`.

Unfortunately, this reduced our mean average precision to 23.94%. This puzzled us, since we expected that excluding non-relevant terms would improve the accuracy of our search results. Out of curiosity we tried making our searches use only the relevant terms (and ignoring the non-relevant ones). This improved our mean average precision from our previous score of 30.24% to 31.63%.

4.6 Adding a Kstem filter to our CustomAnalyzer

The KStem filter is an English-specific stemmer which provides a stemming mechanism though which words are transformed to their root form by applying language-specific rules. We added the KStemFilter to our CustomAnalyzer and this helped improve the MAP score from 31.63% to 31.84%.

4.7 Adding a Length filter to our CustomAnalyzer

Length filter will filter out tokens whose CharTermAttribute is either too short (`CharTermAttribute#length()` < `min`) or too long (`CharTermAttribute#length() > max`). We limit the character length to 2-20 characters as a way of filtering out characters that are too long or too short and enhancing the validity of the text. It helped us to gain a boost in the iprec_at_recall index and slightly increased the MAP score from 31.99% to 32.04%.

4.8 Using a different similarity metric

Looking through the Lucene documentation online, we stumbled upon the MultiSimilarity Class which lets us combine multiple similarity functions. We tried using different combination of similarity functions and were able to get a MAP score of 32.71% by combining BM25, LMJelinekMercerSimilarity(0.9f) and Classic Similarity. At a later stage, we tried enhancing this further by experimenting with different values of model parameter λ in LMJelinekMercerSimilarity and using other similarity functions. We found that using a combination of BM25, LMJelinekMercerSimilarity(0.69f) and AxiomaticF2EXP() helped us raise our MAP score from 32.71% to 33.38%

4.9 Using the OpenNlp Lemmatizer

The processing of terms is always an important factor in the performance of a search engine, so we decided to try using a lemmatizer rather than a stemmer, to see if that would improve our engine's accuracy. We used the OpenNlp Lemmatizer provided by Lucene [9] to do this. As this lemmatizer requires more information than our existing analyzer provided, this also meant incorporating the OpenNlp Tokenizer, the OpenNlp Sentence Detector and the OpenNlp POS Filter. The tokenizer was similar to our existing tokenizer, but kept punctuation as tokens on their own, to be used by the sentence detector and POS filter. The sentence detector finds the beginnings and ends of sentences based on their punctuation. The POS Filter, or Part-Of-Sentence Filter adds tags to words specifying what role the word is serving in the sentence, ie. whether the word is a noun, verb, adjective, etc. This Part-Of-Sentence information was used by the lemmatizer to reduce words to their root word, rather than naively chopping off the end of a word as a stemmer would do.

Unfortunately, the use of the lemmatizer did not end up improving our mean precision, likely because the use of the lemmatizer prevented us from using a lot of filters by requiring very specific input. The final MAP score with the lemmatizer ended up being 30.09%, down from 32.71%.

4.10 Indexing more fields

So far, our search engine only indexed the `<DOCNO>`, `<TITLE>`, and `<TEXT>` fields. We began wondering whether including some of the other elements of the dataset files would change our search engine precision. Looking through the data, we found that the documents included many other fields such as dates, internal document identifiers, authors, and even the names of the people some documents were written for.

Reading through these documents, we parsed an additional 14 XML elements across the four datasets and re-indexed and re-evaluated our search engine. To our disappointment, not only did this increase our indexing time by 146%, but it also slightly reduced our map score (from 32.71% to 32.70%) so we excluded these fields from our final version. We were confused yet again, since we expected that the new fields we indexed (including headlines, dates, and publishers) would be relevant to many of the topics.

4.11 Query re-adjustment

One of the last changes we made were to the weighting of different parts of the query. To give more importance to the words present in the topic title and description, we experimented with repeating the analyzed topic titles and descriptions in various amounts. After trying a number of different combinations, we found that we were able to improve our search engine's precision by making the query from two concatenated descriptions and four concatenated titles. This improved our MAP value from 33.38% to 34.11%.

5 FINAL RESULTS

The final results of our search engine against the qrels file for phase 1 are in the appendix in Table 2.

6 CHALLENGES

Some of the major challenges faced during the implementation of the project were:

- **Communication issues between team members,** much of this is attributable to the fact that there are members of the group for whom English is not their first language. This has led to a certain amount of misinformation in our communication. There were also delays and information asymmetries in communication between group members, which to some extent reduced the efficiency of the work between group members and the progress of the team project.

- **Problems with code integration,** as it was not decided who would be responsible for integrating the code uploaded to Github, everyone uploaded their own code directly to their new branch. This resulted in the group not being able to tell which branch the latest version of the code was on, which greatly reduced the reuse rate and affected the project's progress. In the end we agreed that when the latest results were available and better than the original results, this version of the code would be uploaded to the master branch as the latest version for use.

- **Problems with improvements in system performance,** we have done a lot of work on indexing, searching and CustomAnalyzer, and the performance of the system has been largely satisfactory. However, the later we got, the harder it was to break through in all areas, and sometimes the system performed worse after many attempts. After a meeting and discussion, we realised that we had to improve the system by starting from the source code, taking into account the actual characteristics of the task and understanding Lucene's own search mechanism.

7 CONCLUSION

Assignment 2 highlighted to us many of the challenges faced by designers of search engines. We worked with a much larger set of documents and needed to search them using less conventional query types. Using our knowledge from assignment 1, we were able to spend more time designing our retrieval model rather than spend time learning how to set up and use Lucene.

We investigated and experimented a lot more with the indexer, document similarity metrics, analyzers, and query formation compared to assignment 1. We are satisfied with our final map score of 34.11%.

A APPENDIX

The final results of our search engine against the qrels file for phase 1 are shown in Table 2.

Fig. 1. Interpolated Precision vs Recall for Each Similarity Scoring Approach

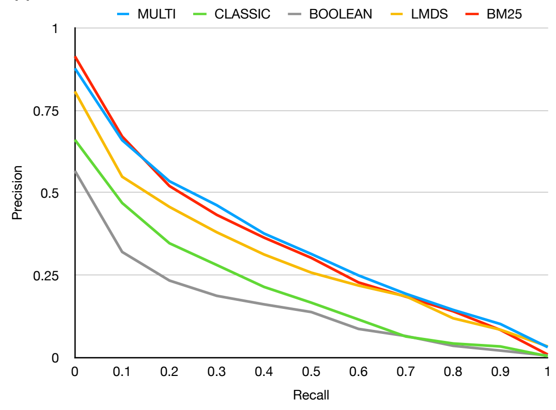

Fig. 2. Precision of the first N documents for Each Similarity Scoring Approach

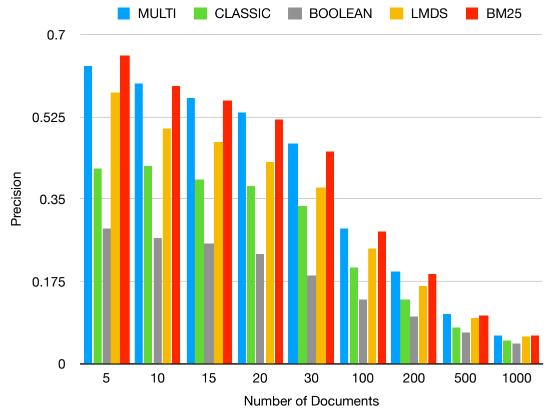

Table 1. Final Model Outputs for Each Similarity Scoring Approach

	MULTI	CLASSIC	BOOL	LMDS	BM25
map	0.3411	0.1951	0.1402	0.2884	0.3306
gmmap	0.2767	0.1468	0.0704	0.2097	0.2676
Rprec	0.3793	0.2697	0.1913	0.3322	0.3653
bpref	0.3365	0.2357	0.1605	0.2858	0.3253
recrnk	0.8433	0.5862	0.5131	0.7792	0.8700

Table 2. Full trec_eval results for final model

Metric	Value
num_q	25
num_ret	25000
num_rel	2132
num_rel_ret	1512
map	0.3411
gm_map	0.2767
Rprec	0.3793
bpref	0.3365
recip_rank	0.8433
iprec_at_recall_0.00	0.8777
iprec_at_recall_0.10	0.6597
iprec_at_recall_0.20	0.5351
iprec_at_recall_0.30	0.4624
iprec_at_recall_0.40	0.3765
iprec_at_recall_0.50	0.3143
iprec_at_recall_0.60	0.2492
iprec_at_recall_0.70	0.1936
iprec_at_recall_0.80	0.1447
iprec_at_recall_0.90	0.1018
iprec_at_recall_1.00	0.0315
P_5	0.6320
P_10	0.5960
P_15	0.5653
P_20	0.5340
P_30	0.4680
P_100	0.2876
P_200	0.1970
P_500	0.1062
P_1000	0.0605

REFERENCES

[1] F.B. Dian Paskalis and M.L. Khodra. "Word sense disambiguation in information retrieval using query expansion". In: *Proceedings of the 2011 International Conference on Electrical Engineering and Informatics*. 2011, pp. 1–6. DOI: 10.1109/ICEEI.2011.6021532.

[2] *IRRAATTREC2012: Divergencefromindependence(dfi) - NIST*. June 2012. URL: https://trec.nist.gov/pubs/trec21/papers/irra.web.nb.pdf.

[3] Elastic. *Configuring a similarity*. URL: https://github.com/elastic/elasticsearch/blob/main/docs/reference/index-modules/similarity.asciidoc.

[4] Edward Kai Dang, Robert Wing Luk, and James Allan. "A comparison between term-independence retrieval models for ad hoc retrieval". In: *ACM Transactions on Information Systems* 40.3 (2022), pp. 1–37. DOI: 10.1145/3483612.

[5] F.B. Dian Paskalis and M.L. Khodra. "Word sense disambiguation in information retrieval using query expansion". In: *Proceedings of the 2011 International Conference on Electrical Engineering and Informatics* (2011). DOI: 10.1109/iceei.2011.6021532.

[6] *WordDelimiterGraphFilter Summary*. Sept. 2017. URL: https://lucene.apache.org/core/6_6_1/analyzers-common/org/apache/lucene/analysis/miscellaneous/WordDelimiterGraphFilter.html.

[7] *SynonymGraphFilter Summary*. Sept. 2017. URL: https://lucene.apache.org/core/6_4_1/analyzers-common/org/apache/lucene/analysis/synonym/SynonymGraphFilter.html.

[8] A Arslan and O Yilmazel. "Quality benchmarking relational databases and Lucene in the trec4 adhoc task environment". In: *Proceedings of the International Multiconference on Computer Science and Information Technology* (2010). DOI: 10.1109/imcsit.2010.5679643.

[9] Apache Foundation. "Lucene OpenNLP Lemmatizer Documentation". In: (2017). URL: https://lucene.apache.org/core/8_0_0/analyzers-opennlp/index.html.

Received 8 December 2022