Information Retrieval Group Report
Group 1 - Page One

Mingwei Shi*
mshi@tcd.ie
Trinity College dublin
Dublin, Leinster, Ireland

Brian Whelan†
whelanb8@tcd.ie
Trinity College dublin
Dublin, Leinster, Ireland

Xinyi Li‡
lix13@tcd.ie
Trinity College dublin
Dublin, Leinster, Ireland

Priya Nawal§
nawalp@tcd.ie
Trinity College dublin
Dublin, Leinster, Ireland

ABSTRACT

Search engines play an important role in allowing users to retrieve information. In particular, search engines have been developed to assist users in finding favourable news. In this work, a search engine was built to optimise search amongst four news article corpora. The search engine makes use of TF-IDF query creation, a custom analyzer and the BM25 scoring method. With respect to the partial evaluation file provided, this model achieves a Mean Average Precision (MAP) score of 0.291 and 0.293 respectively under different query generation.

ACM Reference Format:
Mingwei Shi, Brian Whelan, Xinyi Li, and Priya Nawal. 2023. Information Retrieval Group Report Group 1 - Page One. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In the literature, there exists approaches for developing search engines which are targeted at news information[1]. In this paper, four existing news article corpora from different sources were targeted for optimisation, specifically: "Financial Times Limited (1991–1994)", "Federal Register (1994)", "Foreign Broadcast Information Service (1996)", and "Los Angeles Times (1989–1990)". In this work, a search engine is developed to optimise search in these corpora with respect to the MAP. The key components that contribute to the MAP score are the query creation model (i.e. how the queries were created from the original topics to find relevant documents), the analyzer used to index the documents (i.e. how the documents text was parsed, separated and modelled) and the scorer used in querying the index (i.e. the retrieval model used). The remainder of this report provides details on the implementation of these elements as well as their performance.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 METHODOLOGY

In the methodology section, the implementation of the search engine is described starting with an overview followed by details of the query creation, analyzer and scorer used.

2.1 Overview

The search engine is constructed with a number of components (see figure A.1). Most notably, there is a **Reader**, which is responsible for reading and parsing the four corpora into Lucene documents, an **Indexer**, which is responsible for indexing these documents, a **QueryCreator**, which is responsible for creating queries from the original topics, and a **Querier**, which is responsible for searching the index for documents relevant to each query, retrieving the top 1,000 documents.
The Reader uses the "JSoup" library[2] to parse the four corpora into Lucene documents[3], where each document has the same fields regardless of which corpora it originally came. These fields are document number, title and content. Even though each corpus might have more than three fields (e.g. date, page number), we only select three components which are consistent across all four corpora to aid in querying performance.

2.2 Query Creation

Two types of query creation were tested: query creation using the topic title field only and query creation using TF-IDF. The former acted as a simple baseline query creation model, while the latter ended up being our chosen solution. This section outlines the implementation of both of these methods.

2.2.1 Topic Title Model. In this approach, the raw title from each topic was taken and used as a query without any additional enhancements. For example, the query created from the tenth topic using this model is "Schengen agreement".

2.2.2 TF-IDF Model. For the second query generation, we used TF-IDF to select the context-aware terms within the keyword collection to generate the best topic keywords for each query(Note that we use this TF-IDF calculator to compute the term TF-IDF value[1].). Our corollary is that the higher TF-IDF terms, the more

[1] https://gist.github.com/guenodz/d5add59b31114a3a3c66#file-tfidfcalculator-java-L8

informative content it brings about. The rigorous proof is in the appendix section.A.5.
Furthermore, we also count the top 5 context-aware key terms in the whole fifty queries that are selected by removing punctuation, removing stop words, removing non-context-aware terms, and reducing each term into its' root form, which offers the rationale to design a synonymous analyser for a specific term, such as "countries"(see the figure.2.1).

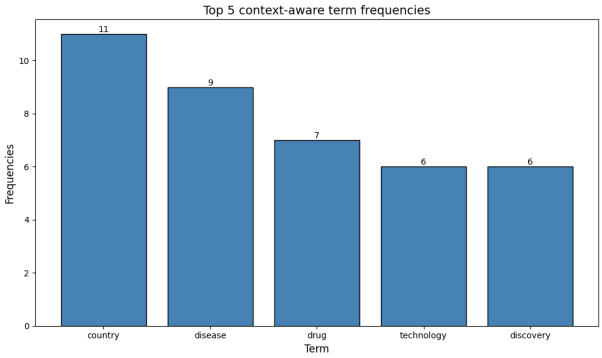

Figure 2.1: Top 5 context-aware key terms from the query corpus

Once we have query collection, we could expand the query by discovering the nearest N (in our setting) documents around the current document in each iteration[4, 5]. Another query expansion trick which was used was synonyms from the top key terms in the query collection. In the Lucene library[3], it sets synonymous words in the same position in the token in order to identify them quickly.
In contrast to the topic title query creation model, the query created from the tenth topic using this model is "border schengen agreement controls".

2.3 Analyser

For indexing, we tested 6 analyzers including two custom analyzers, all of which are described in this section.

2.3.1 StandardAnalyser. This analyser are capable of addressing email address and name features with lowering all the words. Besides, the stop word list for this analyser is 35 words[2].

2.3.2 WhitespaceAnalyzer. This analyser cut the text from document into separate chunks based on spaces.

2.3.3 SimpleAnalyzer. This analyzer cuts text into tokens at every non-letter character, including digits, spaces, hyphens, and apostrophes. Additionally, it removes non-letter characters and converts uppercase letters to lowercase.

2.3.4 EnglishAnalyzer. This analyzer is targeted for English vocabularies.

2.3.5 CustomisedSynonymousAnalyzer. For the customised synonymous analyser, a previous IR assignment's analyser[3] was expanded from only considering "country" to considering the top-5 context-aware terms(see in the figure.2.1).

2.3.6 Generalised Customised Analyser.

(1) **Analyser interpretation:** The analyser extends `StopwordAnalyzerBase` and incorporates logic from both `StandardAnalyzer` and `EnglishAnalyzer`. It offers options for setting maximum and minimum token lengths.
(2) **Stopwords:** The analyzer utilises a predefined set of English stopwords stored in `ENGLISH_STOP_WORDS_SET`. Users have the flexibility to provide their custom set of stopwords during instance creation.
(3) **Tokenization Process:** Tokenization employs a 'standard tokenizer'. Additional filters consist of:
   - `EnglishPossessiveFilter`: Manages English possessives.
   - `LowerCaseFilter`: Converts tokens to lowercase.
   - `StopFilter`: Eliminates stopwords.
   - `LengthFilter`: Filters tokens based on length constraints.
   - `SetKeywordMarkerFilter`: Optionally marks tokens as keywords (excluded from stemming).
   - `PorterStemFilter`: Applies Porter stemming.
(4) **Customization:** Users can define maximum and minimum token lengths. A 'stemExclusionSet' permits specification of tokens exempted from stemming.
(5) **Usage Example:** To utilize this analyzer, instantiate an object and apply it to either a 'Reader' or a 'String' field.
(6) **Potential Improvements:** The code contains commented-out lines associated with 'setMaxTokenLength' and 'KStemFilter'. These features, based on specific use cases, can be uncommented and adjusted.

Overall, this analyzer is designed to provide flexibility and fine-tuning in text analysis for various applications. Users can adjust stop-words, customise token length constraints, and control stemming behaviour to suit specific requirements.

2.4 Scorer

Vector space model. The vector space model[6] scoring algorithm utilises the TF-IDF(Term frequency and inverse document frequency) statistic[7] to assess the relevance of documents by considering both the frequency of query terms within a document and their frequency across all documents. In Lucene, this is referred to as ClassicSimilarity[4].

BM25. The BM25 (Best Matching 25) scoring is a probabilistic relevance model that considers the average document length in the corpus, parts of the TF-IDF statistic[7], and the weights given to query phrases[8].

Bayesian smoothing using Dirichlet priors. This retrieval model was invented by Bayesian Dirichlet Prior[9]. Terms that

[2]https://github.com/apache/lucene-solr/blob/master/lucene/analysis/common/src/java/org/apache/lucene/analysis/en/EnglishAnalyzer.java#L46

[3]https://github.com/tannineo/cs7is3-team4/blob/master/src/main/java/life/tannineo/cs7is3/group4/CustomAnalyzer_Syn_stp.java

[4]https://lucene.apache.org/core/8_7_0/core/org/apache/lucene/search/similarities/ClassicSimilarity.html

exhibit a lower frequency than the language model predicts acquire a negative score in this scoring system. However, the Lucene search engine[3] assigned a zero score for this case for smoothing the results that are greater than 0. In Lucene, there are two versions of this model: LMJelinekMercerSimilarity[5] and LMDirichletSimilarity [6].

Combination of multiple search. This retrieval model combines evidence from multiple other scoring methods to attain a single score[10]. In Lucene, it is called "MultiSimilarity"[7].

2.5 Parsers

Parsers are used to read in the four new article corpora as well as the topics. These are discussed in this section.

2.5.1 News Article Corpora Parser. Within the Reader, the documents are parsed. Regardless of which corpus is used, the same fields are selected, namely the document number, title, and the content. This ensures that the same fields are written into the Lucene document[3] which makes it easier for top documents to be retrieved when querying due to this consistency.

2.5.2 Topic Parser. The topic parser is designed to parse the topic file containing 50 topics. Each topic has different sections identified by specific tags such as NUM, TITLE, DESCRIPTION and NARRATIVE. The parser uses the internal enum "Tags" for efficient tag management and regular expressions to extract titles accurately. Once a topic is fully resolved, it is added to the list of topics objects which are then provided to the QueryCreator for query creation.

2.6 Indexing & Querying Optimisations

In this section, some optimisations that were made to the search engine are discussed.

2.6.1 Cache Size & Document Merging. In order to enable indexing to function with a large corpora of documents, some additional configuration was needed. Firstly, the number of documents preserved in cache was set to 100,000 to improve the speed at which indexing takes place[8].
In addition, the 100,000 documents stored in cached are merged into a single chunk periodically. This enables faster querying. Finally, due to issues with heap memory, periodically our program would crash if not enough heap space has been provided. In this specific case, a lock would be held on the index but could not be released due to the program crash, hence the lock file was deleted[9] to enable the lock to be obtained by new runs of the Indexer.

2.6.2 Multi-threading. Due to the size of the datasets, the indexing time was considerable as well as the amount of heap memory occupied as documents were read in from the four corpora. In order to reduce the indexing time, multi-threaded indexing was incorporated whereby each dataset was indexed in its own thread and the results were then merged on completion. The indexing speed was reduced from the initial 1m 40s to 20s. Each data set is allocated to a separate thread for indexing and then merged into a unified index, which ensures the integrity and consistency of the index.

3 EVALUATION

In the evaluation section, the results of running all of the various methods tested are shown, providing a basis for the selection of the query creation mode, analyzer and scorer used in the search engine. These three components are assessed individually in the following sections.
3 and 3 are the two results tables showing the best performing combinations of components. The full versions of these results tables can be found in A.3.

Table 3.1: Results of Analyzer-Scorer combinations using topic title query creation which yielded MAP score > 0.28 (full version in A.1)

runid	map	set_recall
CustomAnalyzer_Syn_stp-BM25Similarity	0.2862	0.7555
EnglishAnalyzer-BM25Similarity+LMDirichletSimilarity	0.2856	0.7653
CustomAnalyzer_Syn_stp-LMJelinekMercerSimilarity	0.2803	0.762
EnglishAnalyzer-LMDirichletSimilarity	0.2826	0.7708
EnglishAnalyzer-BM25Similarity	0.2909	0.7522
GeneralizedCustomAnalyzer-LMDirichletSimilarity	0.2807	0.7645
EnglishAnalyzer-LMJelinekMercerSimilarity	0.2803	0.7604
GeneralizedCustomAnalyzer-BM25Similarity+LMDirichletSimilarity	**0.2933**	**0.771**
GeneralizedCustomAnalyzer-BM25Similarity	0.2917	0,7481
CustomAnalyzer_Syn_stp-BM25Similarity+LMDirichletSimilarity	0.2907	0.7795
CustomAnalyzer_Syn_stp-LMDirichletSimilarity	0.2835	0.7634

[5]https://lucene.apache.org/core/8_8_0/core/org/apache/lucene/search/similarities/LMJelinekMercerSimilarity.html
[6]https://lucene.apache.org/core/8_7_0/core/org/apache/lucene/search/similarities/LMDirichletSimilarity.html
[7]https://lucene.apache.org/core/8_0_0/core/org/apache/lucene/search/similarities/MultiSimilarity.html
[8]http://www.java2s.com/example/java-api/org/apache/lucene/index/indexwriterconfig/setopenmode-1-4.html
[9]https://stackoverflow.com/questions/2341163/why-is-my-lucene-index-getting-locked

Table 3.2: Results of Analyzer-Scorer combinations using TF-IDF query creation which yielded MAP score > 0.25 (full version in A.2)

runid	map	set_recall
GeneralizedCustomAnalyzer-LMJelinekMercerSimilarity	0.2639	0.7199
CustomAnalyzer_Syn_stp-BM25Similarity	**0.291**	**0.7333**
GeneralizedCustomAnalyzer-ClassicSimilarity+BM25Similarity	0.2579	0.7047
CustomAnalyzer_Syn_stp-LMJelinekMercerSimilarity	0.277	0.7338
CustomAnalyzer_Syn_stp-ClassicSimilarity+BM25Similarity	0.2807	0.7302
EnglishAnalyzer-BM25Similarity	0.2741	0.7157
EnglishAnalyzer-ClassicSimilarity+BM25Similarity	0.2681	0.7192
EnglishAnalyzer-LMJelinekMercerSimilarity	0.2816	0.7436
GeneralizedCustomAnalyzer-BM25Similarity+LMDirichletSimilarity	0.2436	0.6731
GeneralizedCustomAnalyzer-BM25Similarity	0.2678	0.712
CustomAnalyzer_Syn_stp-BM25Similarity+LMDirichletSimilarity	0.2586	0.7085

3.1 Query Creation

Irrespective the Analyzer-Scorer combination used, the topic title query creation model appeared to outperform the TF-IDF query creation model based on MAP scores alone. For instance, looking at the EnglishAnalyzer-BM25Similarity combination across both query creation methods, the topic title method yields a MAP of 0.2909 while the TF-IDF approach only achieves 0.2741.
However, this evaluation only uses half of the queries (25 out of 50) and the MAP scores tend to be much more clustered across Analyzer-Scorer combinations for the topic title model with only 15 out of 42 combinations having MAP scores below 0.25 in contrast to 31 out of 42 using the TF-IDF model.
In addition, the query created via the topic title model is usually a subset of that created by the TF-IDF model. For example, for the twenty eighth topic, the topic title model created the query "declining birth rates" while the TF-IDF model created the query "birth rate declining u.s china year". The TF-IDF model generally provides more context in the query. Therefore, it is estimated that the TF-IDF model will perform better when the full evaluation file is used.

3.2 Analyzer

Irrespective of the query creation method used, the WhiteSpaceAnalyzer, SimpleAnalyzer and StandardAnalyzer all performed poorly with respect to the other analyzers used.
Using the topic title query creation method, the EnglishAnalyzer and two custom analyzers performed similarly, with all having MAP scores of $\tilde{0}.29$ when using the BM25 scorer. However, the lack of distinction between scorers here may be related to the query creation model.
In contrast, using the TF-IDF query creation model, while all perform well (MAP scores > 0.26 when using the BM25 scorer), the CustomAnalyzer_Syn_stp provides a noticeable improvement of $\tilde{0}.02$ from both other analyzers.
For these reasons, the CustomAnalyzer_Syn_stp is estimated to perform best along with our TF-IDF query creation approach.

3.3 Scorer

Irrespective of the query creation method used, the ClassicSimilarity performed poorly with respect to the other analyzers used.
Using the topic title query creation method, the BM25Similarity, LMDirichletSimilarity and LMJelinekMercerSimilarity(0.7f) with the EnglishAnalyzer performed well within the MAP score range 0.2803 to 0.2909. Again, the lack of distinction between scorers here may be related to the query creation model.
In contrast, using the TF-IDF query creation model, the BM25 scorer stands out in providing the best MAP scores. Taking the CustomAnalyzer_Syn_stp, the BM25 scorer achieves a MAP score of 0.291 which drops to 0.277 when replaced with the next best LMJelinekMercerSimilarity(0.7f) scorer. Combinations also work well, particularly those containing the BM25 model but they offer no substantial gains and in some case result in a drop in MAP score.
For these reasons, the BM25 scorer is estimated to perform best along with our TF-IDF query creation approach.

4 DISCUSSION

Even though our model has relatively good results in the mean average precision, there is still room for improvement. Phrase search has to be considered, as phrases might convey more meaning than single terms[11–13]. Besides, in order to extract patterns between queries and documents, learning to rank could be considered as this algorithm utilised more implicit and explicit features among query terms and documents[14, 15].

5 CONCLUSION

This project develops a search engine optimised for searching across four news article corpora. It integrates concurrent indexing, complex query generation, efficient document analysis strategies, and precise scoring mechanisms to enhance search accuracy and relevance Through our evaluations, we found that combining the TF-IDF query generation method with the CustomAnalyzer_Syn_stp analyzer and the BM25 scorer yielded the best performance. Although queries based on topic titles performed slightly better in preliminary tests—by a marginal difference of 0.02—the TF-IDF method is expected to demonstrate stronger performance in more comprehensive evaluations. Future research should explore the incorporation of more sophisticated machine-learning techniques to better identify and utilise the subtle patterns between queries and documents.

REFERENCES

[1] A. Gulli, "The anatomy of a news search engine," in *Proceedings of the 14th international conference on World Wide Web*, p. 880, 2005.
[2] J. Hedley, "jsoup," 2009. Last Accessed: 2020-04-17.
[3] T. A. S. Foundation, "Lucene," 2002. Last Accessed: 2020-04-17.
[4] R. Schenkel, A. Broschart, S. Hwang, M. Theobald, and G. Weikum, "Efficient text proximity search," in *String Processing and Information Retrieval: 14th International Symposium, SPIRE 2007 Santiago, Chile, October 29-31, 2007 Proceedings 14*, pp. 287–299, Springer, 2007.
[5] E. M. Voorhees, "Query expansion using lexical-semantic relations," in *SIGIR'94: Proceedings of the Seventeenth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval, organised by Dublin City University*, pp. 61–69, Springer, 1994.
[6] G. Salton, A. Wong, and C.-S. Yang, "A vector space model for automatic indexing," *Communications of the ACM*, vol. 18, no. 11, pp. 613–620, 1975.
[7] K. D. Bollacker, S. Lawrence, and C. L. Giles, "Citeseer: An autonomous web agent for automatic retrieval and identification of interesting publications," in *Proceedings of the second international conference on Autonomous agents*, pp. 116–123, 1998.
[8] S. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford, "Okapi at trec-3. proceedings of the third text retrieval conference (trec 1994)," in *Gazxithersburg, USA*, 1999.
[9] J. Lafferty and C. Zhai, "Document language models, query models, and risk minimization for information retrieval," in *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 111–119, 2001.
[10] E. Fox and J. Shaw, "Combination of multiple searches," *NIST special publication SP*, pp. 243–243, 1994.
[11] W. B. Croft, H. R. Turtle, and D. D. Lewis, "The use of phrases and structured queries in information retrieval," in *Proceedings of the 14th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 32–45, 1991.
[12] D. A. Evans and C. Zhai, "Noun-phrase analysis in unrestricted text for information retrieval," *arXiv preprint cmp-lg/9605019*, 1996.
[13] X. Wang, A. McCallum, and X. Wei, "Topical n-grams: Phrase and topic discovery, with an application to information retrieval," in *Seventh IEEE international conference on data mining (ICDM 2007)*, pp. 697–702, IEEE, 2007.
[14] T.-Y. Liu *et al.*, "Learning to rank for information retrieval," *Foundations and Trends® in Information Retrieval*, vol. 3, no. 3, pp. 225–331, 2009.
[15] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li, "Learning to rank: from pairwise approach to listwise approach," in *Proceedings of the 24th international conference on Machine learning*, pp. 129–136, 2007.
[16] A. Aizawa, "An information-theoretic perspective of tf–idf measures," *Information Processing & Management*, vol. 39, no. 1, pp. 45–65, 2003.
[17] C. E. Shannon, "A mathematical theory of communication," *ACM SIGMOBILE mobile computing and communications review*, vol. 5, no. 1, pp. 3–55, 2001.

A APPENDICES

The appendices contain links to our journal and repository, instructions for running the search engine, the full result tables, an overview diagram, a proof and the group contributions.

A.1 Group Journal

Here is the link for group journal: https://docs.google.com/document/u/2/d/1IwilffetMe_S1e3H-aCRE70p6UBlT73zhW5lR600vec/edit#task=sYtfaqoqpJ6QzSZ7.

A.2 Running the Search Engine

Instructions on how to run the search engine are available in the README in the repository https://github.com/briantwhelan/page1-search-engine
To run using our Azure VM, use the details found in our Blackboard group journal in the post named 'INSTANCE DETAILS'.
If you run into any issues, accessing the VM or running the code on it, please get in touch at whelanb8@tcd.ie and I will assist. I will turn off the VM at the end of term (15th December) to save credits.

A.3 Full Result Tables

The extended versions of the tables displayed in the evaluation (3) are shown in this section.
A.1 is the full version of 3 displaying the results using the topic title query creation method.
A.2 is the full version of 3 displaying the results using the TF-IDF query creation method.
For the tables displaying more metrics outside of MAP and recall see the GitHub repository.

A.3.1 Notation. We used the following notation to refer different analyser-score combination.

Analysers:

1) $a1$ for Generalised Customised Analyzer.
2) $a2$ for Standard Analyzer.
3) $a3$ for White Space Analyzer.
4) $a4$ for English Analyzer.
5) $a5$ for Simple Analyzer.
6) $a6$ for Customised Analyzer.

Scoring:

1) $s1$ for Classic Similarity.
2) $s2$ for BM25 Similarity.
3) $s3$ for LMDirichlet Similarity.
4) $s4$ for LMJelinekMercer Similarity with λ for 0.7f.
5) $s5$ for MultiSimilarity combining Classic Similarity and BM25 Similarity.
6) $s6$ for MultiSimilarity combining Classic Similarity and LMDirichlet Similarity.
7) $s7$ for MultiSimilarity combining BM25 Similarity and LMDirichlet Similarity.

A.3.2 Results of all 42 Analyser-Scorer combinations using topic title query creation. Here is the table.

A.3.3 Results of all 42 Analyzer-Scorer combinations using topic TF-IDF query creation. Here is the table.

Table A.1: Results of all 42 Analyzer-Scorer combinations using topic title query creation

Analyzer-Scorer	MAP	Recall
*a*4-*s*6	0.2779	0.7815
*a*1-*s*4	0.2788	0.7539
*a*5-*s*1	0.1539	0.5395
*a*3-*s*2	0.2324	0.6012
*a*5-*s*6	0.2579	0.6658
*a*5-*s*3	0.2426	0.6506
*a*4-*s*1	0.1883	0.6747
*a*6-*s*2	0.2862	0.7555
*a*1-*s*5	0.2617	0.7299
*a*4-*s*3	0.2856	0.7653
*a*6-*s*4	0.2803	0.762
*a*5-*s*5	0.2413	0.6454
*a*6-*s*5	0.2635	0.745
*a*3-*s*2	0.1976	0.5802
*a*4-*s*3	0.2826	0.7708
*a*2-*s*1	0.1559	0.5577
*a*4-*s*2	0.2909	0.7522
*a*3-*s*4	0.19	0.5762
*a*1-*s*3	0.2807	0.7645
*a*2-*s*3	0.2645	0.6819
*a*3-*s*5	0.1968	0.5775
*a*3-*s*3	0.201	0.5665
*a*4-*s*5	0.2653	0.736
*a*4-*s*6	0.2803	0.7604
*a*1-*s*7	0.2933	0.771
*a*1-*s*2	0.2917	0.7481
*a*3-*s*1	0.1183	0.4674
*a*2-*s*4	0.2736	0.683
*a*2-*s*5	0.248	0.6624
*a*5-*s*3	0.2627	0.6769
*a*6-*s*7	0.2907	0.7795
*a*6-*s*3	0.2835	0.7634
*a*3-*s*6	0.2109	0.5836
*a*6-*s*1	0.192	0.678
*a*1-*s*1	0.1842	0.6619
*a*5-*s*2	0.2641	0.6651
*a*2-*s*6	0.2604	0.6679
*a*6-*s*6	0.2724	0.7685
*a*2-*s*2	0.2723	0.6747
*a*1-*s*6	0.2753	0.7658
*a*2-*s*4	0.2564	0.677
*a*5-*s*4	0.2774	0.6831

Table A.2: Results of all 42 Analyzer-Scorer combinations using TF-IDF creation

Analyzer-Scorer	MAP	Recall
*a*4-*s*6	0.2243	0.6773
*a*1-*s*4	0.2639	0.7199
*a*5-*s*1	0.1624	0.6098
*a*3-*s*2	0.1106	0.4018
*a*5-*s*6	0.2114	0.6441
*a*5-*s*3	0.2223	0.6492
*a*4-*s*1	0.1845	0.6474
*a*6-*s*2	0.291	0.7333
*a*1-*s*5	0.2579	0.7047
*a*4-*s*3	0.246	0.6787
*a*6-*s*4	0.277	0.7338
*a*5-*s*5	0.2395	0.6849
*a*6-*s*5	0.2807	0.7302
*a*3-*s*2	0.1215	0.4298
*a*4-*s*3	0.2111	0.6665
*a*2-*s*1	0.1692	0.5881
*a*4-*s*2	0.2741	0.7157
*a*3-*s*4	0.1246	0.4485
*a*1-*s*3	0.2135	0.6616
*a*2-*s*3	0.1965	0.6002
*a*3-*s*5	0.1271	0.4495
*a*3-*s*3	0.0994	0.4105
*a*4-*s*5	0.2681	0.7192
*a*4-*s*6	0.2816	0.7436
*a*1-*s*7	0.2436	0.6731
*a*1-*s*2	0.2678	0.712
*a*3-*s*1	0.0757	0.4023
*a*2-*s*4	0.2233	0.6547
*a*2-*s*5	0.2377	0.6802
*a*5-*s*3	0.1926	0.6004
*a*6-*s*7	0.2586	0.7085
*a*6-*s*3	0.2038	0.6697
*a*3-*s*6	0.1091	0.4064
*a*6-*s*1	0.1894	0.6559
*a*1-*s*1	0.1866	0.6192
*a*5-*s*2	0.2308	0.6635
*a*2-*s*6	0.2012	0.6408
*a*6-*s*6	0.2379	0.6967
*a*2-*s*2	0.2338	0.6733
*a*1-*s*6	0.2161	0.6675
*a*2-*s*4	0.2261	0.6586
*a*5-*s*4	0.2169	0.6575

A.4 The overview diagram

A.5 TF-IDF query generation proof

The general mechanism is as follows:
The weight of every keyword in a query should express the information it brings about[16]. The intuitive approach to utilising the information amount of every term is as its' weight. From information theory, information entropy is a measure of the uncertainty of specific information[17]. By applying this theory, we

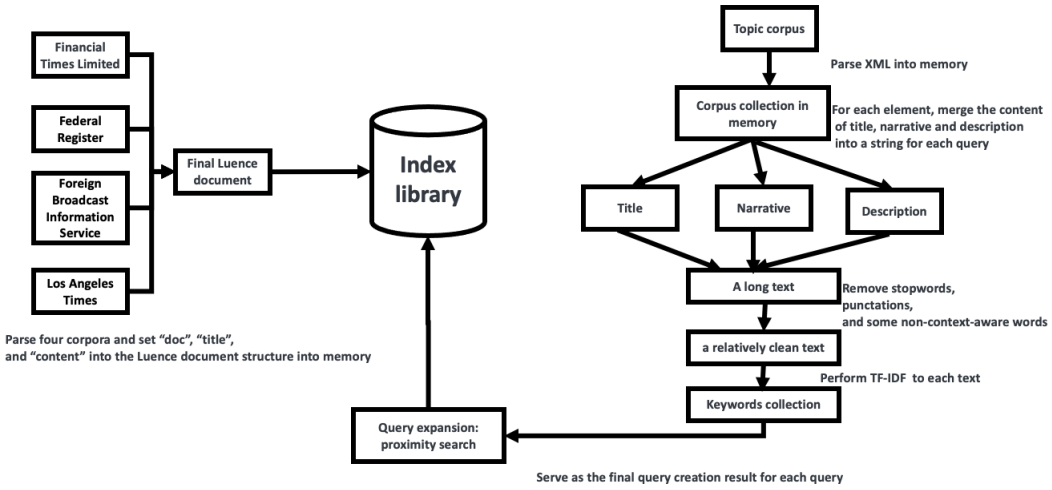

Figure A.1: The four corpora are parsed into Lucene documents which are then indexed. The original topics go through a series of steps to create a number of queries. The queries search the index for relevant documents and retrieve the top 1,000 documents for each query.

could derive the following formula:

$$I(w) = -P(w)\log P(w) = -\frac{TF(w)}{N}\log\frac{TF(w)}{w} \quad (1)$$

The N is the size of the whole corpus; this constant and the negative sign could be ignored, as these does not impact the complexity of this equation.1. For each query document, in our context, the string consisting of "title", "desc", and "narr" is fixed after removing punctuation, stop words and non-context-aware vocabulary.

We need to explore this formula further to identify the relationship between information amount and TF-IDF. Besides, for each query document, we could denote the number of terms in this query document as M. Based on this description, we derive the following formula: $M = \frac{N}{D} = \frac{\sum_w TF(w)}{D}$,where N is the number of the whole collection and D is the size of the document.

A key term once appears in a document, and we denote a weight for each term in the collection as t for this keyword without loss of generalisation as each term weight might vary. Once a particular term appears,no matter how many times, we could derive this: c(w) = $\frac{TF(w)}{D(w)}$. where c(w) is the occurrence of term in the document,which could be represented by the ratio of the value of term frequency of this term to the occurrence of the whole document. It is noticed that this value is less than the size of the document, indicating $c(w) < M$ as the M is the number of total terms.

Based on the above explanation, we expanded the formula 1 into the following equation:

$$\begin{aligned} I(w) &= TF(w)\log\frac{N}{TF(w)} \\ &= TF(w)\log\left(\frac{MD}{t\cdot c(w)\cdot D(w)}\right) \\ &= TF(w)\log\left(\frac{D}{D(w)}\cdot\frac{M}{t\cdot c(w)}\right) \\ &= TF(w)\log\left(\frac{D}{D(w)}\right) + TF(w)\log\left(\frac{M}{t\cdot c(w)}\right) \\ &= TF\text{-}IDF(w) + TF(w)\log\left(\frac{M}{t\cdot c(w)}\right) \end{aligned}$$

Then we transform the equation, we could derive this final formula, $TF - IDF(w) = I(w) - TF(w)\log(\frac{M}{t*c(w)})$.

Since c(w) is always less then M,so $\log\frac{M}{t*c(w)}$ is greater than 0. Based on this fact, $\frac{M}{t*c(W)}$ is greater than 0; put this non-zero value into the Logarithm function. We could derive a nearly constant value when the w increase over 10.

Then we could simplify the final formula: $TF - IDF(w) = I(w) - TF(w) * C$ where C is a constant value. We could regard it as a linear relationship ,such as y = x-b.

From this formula, the more information this term brings, the more TF-IDF value, indicating that TF-IDF is a reasonable and legitimate approach to ranking keywords. Based on this theoretical finding, we selected the top 11 TF-IDFs as the same TF-IDF might consist of several terms.

A.6 Group Contributions

In this section, the individual group contributions to the project are noted. In addition contributions can be viewed in both our Blackboard journal and in our GitHub repository

Mingwei Shi:

1. Wrote the whole model for phase one, including query creation, indexing construction, and basic model construction
2. Adapted the existing customised analyser to extend new key terms functionality.
3. Wrote the methodology, introduction, appendix table discussion, and conclusion of the research paper.
4. I worked as the team leader, allocating the task to each member and explaining how this project worked.

Brian Whelan:

1. Created infrastructure around basic indexing and querying to allow running of multiple Analyzer-Scorer combinations with results being evaluated against the qrels file using trec_eval and then being output into a markdown table for easy comparison (all handled in a single run bash script)
2. Read in and parsed 3 of the datasets and handled issues with JVM heap memory due to size of dataset and index when indexing (increased size of our VM and gave ~4GB of RAM to the JVM)
3. Wrote the evaluation section of the report as well as proofread and refactored the abstract, introduction and methodology
4. Scheduled calendar meetings, organised meetings document and created and managed GitHub repository

Xinyi Li:

1. Created and coded topic parsers, and refactored basic indexing, querying and search engine code (although not adopted).
2. Add MultiSimilarity scorers which combines two scorers and use different analyzers and scorers combinations to run to find and assess optimal results.
3. Adjust the index and search engine classes to add multithreaded indexing, which greatly speeds up the time required for indexing (from 1m40s to 20s).
4. Wrote the dataset, topic parser, multithread index and conclusion in the report.

Priya Nawal:

1. Created GeneralisedCustomAnalyzer to get the best map accuracy possible.
2. Ran analyzer with different scorers and obtained results.
3. Wrote a report for GeneralisedCustomAnalyzer.