SEUPD@CLEF: Team <Semicolon>

Reihaneh Baghishani[1], Irem Goksu Celik[1], Hamideh Haeri[1],
Ferdos Hajizadeh Chavari[1], Hossein Hosseinpour[1], Ali Tahvildari[1] and
Mohammad Torabi[1]

[1]*University of Padua, Italy*

Abstract
In this paper, we present a search engine, focusing on efficient indexing, sanitization, and retrieval of relevant documents. We implemented a system that first checks if the input documents are sanitized and then proceeds with the sanitization process if required, storing the cleaned documents in a separate directory. The system then indexes the documents using a custom DirectoryIndexer that leverages an NGramAnalyzer and a similarity measure based on the BM25 algorithm. The DirectoryIndexer also uses a parser to extract meaningful information from the documents. The system is designed to handle a large number of documents, with a specific focus on the English language. It supports both single-line and multi-line document formats and can easily adapt to different input and output folders. The search engine also provides a searching functionality that accepts a set of queries and returns a run file containing evaluation measures to assess the relevance of the retrieved documents. Our search engine aims to facilitate efficient information retrieval and improve the overall user experience while working with academic documents.

Keywords
Search Engine, Academic Purposes, Sanitization, Indexing, BM25 Algorithm, Parser, Information retrieval,

1. Introduction

Information retrieval (IR) is a rapidly evolving research area concerned with the organization, storage, and retrieval of information from large-scale document collections. The primary goal of IR systems is to assist users in finding relevant information that satisfies their information needs, despite the growing volume and complexity of data available on the internet [1]. This paper presents our approach to developing a search engine for processing documents written in English and French languages, using the Lucene library [2] and its methods to provide a custom search solution. Apache Lucene is an open-source software library developed by the Apache Software Foundation, primarily used for implementing search engines that work with text-based inputs. The library provides a streamlined API that facilitates the indexing and searching of large collections of text documents [3].

"Search Engines", course at the master degree in "Computer Engineering", Department of Information Engineering, and at the master degree in "Data Science", Department of Mathematics "Tullio Levi-Civita", University of Padua, Italy. Academic Year 2022/2023

reihaneh.baghishani@studenti.unipd.it (R. Baghishani); iremgoksu.celik@studenti.unipd.it (I. G. Celik); hamideh.haeri@studenti.unipd.it (H. Haeri); ferdos.hajizadehchavari@studenti.unipd.it (F. Chavari); hossein.hosseinpour@studenti.unipd.it (H. Hosseinpour); ali.tahvildari@studenti.unipd.it (A. Tahvildari); mohammad.torabi@studenti.unipd.it (M. Torabi)

Recent years have witnessed the emergence of several prominent IR systems, such as Hello-IR [4] and Hello-Tipster [5], which have made significant contributions to the field. In our project, we have carefully analyzed these previous works to identify their strengths and limitations. Building upon this analysis, we have devised our own search engine that addresses some of the identified challenges and offers improved performance across various metrics.

Our methodology includes a comprehensive Experimental Setup, which makes use of the Qwant search engine's collection of queries and documents [4]. This collection serves as the official test collection for the 2023 LongEval Information Retrieval Lab (https://clef-longeval.github.io/) [6], and consists of test datasets for two sub-tasks: short-term persistence (sub-task A) and long-term persistence (sub-task B). Our experiments are designed to evaluate the performance of our search engine in these sub-tasks, and the results are discussed in detail in the Results and Discussion section.

In the Conclusions and Future Work section, we summarize our findings and outline the potential areas for future research in the field of information retrieval. By developing a custom search engine that leverages the capabilities of the Lucene library, our work contributes to ongoing advancements in the field of information retrieval and offers new insights for the processing of multilingual document collections.

The paper is organized as follows: Section 2 describes our approach; Section 3 explains our experimental setup; Section 4 discusses our main findings; finally, Section 5 draws some conclusions and outlooks for future work.

2. Methodology

The methodology adopted in this system is based on information retrieval techniques implemented in the Apache Lucene library. The architecture of the system includes following main components:

2.1. The Parser

In this paper, we will delve into the intricacies of parsing documents for a search engine. A critical first step in the parsing process is the prettification of the data using regex matching. This involves the identification and removal of extraneous elements such as Microsoft Excel formulas, JavaScript dynamic code blocks, Unicode characters, and whitespace. Our research team achieved this by developing a series of regular expressions that systematically identified and removed these elements from the documents.

Once the data is prettified, the parsing process can begin. This involves breaking down the documents into their constituent parts, such as individual words or phrases, and assigning metadata to each part. The metadata may include information about the context in which each word or phrase appears, such as the document's title or author, or information about the part of speech or grammatical structure of each word. Our team used a combination of rule-based and machine-learning techniques to accomplish this step.

By streamlining the parsing process through the prettification of data, we were able to create a more efficient and effective search engine. The parsed data was used to create an index that mapped each word or phrase to the documents in which it appeared. This allowed for rapid retrieval of documents containing specific search terms. The index was organized in a variety of ways depending on the search engine's needs, such as by word or by document.

By thoroughly parsing documents, we can develop a search engine that is both efficient and effective. However, this is a complex and multi-step process that demands meticulous attention to detail. To streamline the parsing process, we can employ techniques such as regex matching and natural language processing.

2.2. The Indexer

To create a high-performance search engine, indexing is a crucial step in the process. The Lucene package is a dependable and adaptable framework for indexing and searching documents. Our research team employed this powerful tool to develop an effective and efficient search engine.

Lucene's indexing process involves creating an index file that contains information about each indexed document. The index file is generated by using a parser to break down each document into its individual components and assigning metadata to each component. Our indexing process utilized the parser developed in the previous step to preprocess the documents before indexing, streamlining the process and improving the accuracy of the search engine's results.

Lucene's indexing process is highly customizable, allowing for optimization of the index for speed or memory usage, depending on the available resources and desired search performance. The index can also be organized in a variety of ways, such as by word or by document, to further enhance search efficiency.

The effectiveness of a search engine heavily relies on the quality of its indexing process. By utilizing the Lucene package and the parser developed in the previous step, our search engine was able to efficiently index and retrieve relevant documents in response to user queries. The next step in building a search engine is the searcher component, which will be discussed in the following section.

2.3. The Searcher

The searcher component of a search engine is responsible for retrieving relevant documents in response to user queries. It utilizes the index created in the previous step to search for relevant documents based on user queries.

The searcher component typically involves constructing a BooleanQuery that combines various fields such as the title, body, and metadata of the documents to retrieve the most relevant documents. The search process may also involve additional techniques such as ranking and relevance scoring to ensure that the most relevant documents are returned.

The searcher component is a crucial part of a search engine, and its effectiveness heavily relies on the quality of the index created in the previous step. The index must be organized and optimized in a way that allows for efficient and accurate retrieval of relevant documents.

In addition to constructing an effective search algorithm, the searcher component may also perform additional checks to ensure that the inputs are valid. For instance, it may check that the index directory exists and is readable, the query is well-formed and valid, and the maximum number of documents to be retrieved is within a reasonable range.

The implementation of the searcher component is crucial in ensuring efficient and accurate retrieval of relevant documents in response to user queries, making it a critical part of a search engine. Therefore, it must be carefully designed to achieve optimal performance.

2.4. Analyzer

The Apache Lucene library provides a rich collection of analyzers and tools for text analysis in Java. The Analyzer is a crucial component of any information retrieval system, responsible for transforming text data into an appropriate format for indexing and searching. Lucene offers two types of analyzers: Basic Analyzer and Custom Analyzers, both leveraging Lucene's functionalities, such as tokenization, stopword removal, stemming, and case normalization, to achieve optimal results.

One of the key components of the text analysis process is tokenization, which involves breaking text into individual tokens or words. StandardTokenizer is a robust tokenizer that conforms to the Unicode Text Segmentation algorithm, as specified by Unicode Standard Annex #29 (UTF-8)[7]. This tokenizer is highly suitable for processing most natural language text and has proven to be reliable in various applications. In Lucene, the StandardAnalyzer utilizes StandardTokenizer to tokenize text into words, remove stop words, and transform words into their base form. The StandardAnalyzer is highly configurable and can handle various language-specific features, making it a popular choice among developers and researchers. Meanwhile, Custom Analyzers allow users to create their own tokenization and filtering rules for even greater customization.[8].

2.4.1. Basic Analyzer

The Basic Analyzer in Lucene employs the StandardTokenizer, which is designed to handle a wide range of text inputs, including different languages and writing systems. The tokenizer efficiently tokenizes text into individual terms, which are then processed by subsequent analysis components [8].

- **StandardAnalyzer**
  The StandardAnalyzer is an essential component in the Lucene search engine library. It is responsible for tokenizing and normalizing textual data during the indexing and querying process. When indexing documents, the StandardAnalyzer breaks down the input text into individual words or terms, also known as tokens, based on specific rules. It employs various techniques to split the text, such as removing punctuation, converting text to lowercase, and eliminating common words (stop words) like "a," "the," and "is." This process ensures that the indexed data consists of meaningful and searchable terms. During the query phase, the StandardAnalyzer applies the same tokenization and normalization steps to the search input, allowing it to match the indexed terms accurately. The StandardAnalyzer is a versatile and widely used analyzer in Lucene, suitable for many applications that deal with natural language text.

2.4.2. Custom Analyzers

While the Basic Analyzer is suitable for many use cases, it might not be optimal for specific applications that require specialized processing. To cater to these needs, Lucene allows the creation of Custom Analyzers, which can utilize various stemmers, lemmatizers, and filters tailored to the requirements of the task at hand.

- **Stemming**
  Stemming is the process of reducing words to their root forms, enabling effective matching of related terms during indexing and searching. Lucene provides several stemming algorithms, such as the Porter Stemmer [9] and the Snowball Stemmer [10]. These algorithms can be incorporated into custom analyzers to enhance their performance.
- **Lemmatization**
  Lemmatization is an advanced form of stemming that considers the morphological and syntactic properties of words to derive their base forms. It is particularly useful for languages with complex morphology, such as Russian and Arabic [11]. Lucene offers a variety of lemmatization tools, which can be incorporated into custom analyzers for improved text analysis.
- **Filters**
  Filters are essential components of custom analyzers that enable the transformation and refinement of tokenized text. One of the most common filters is the LowerCaseFilter, which normalizes the case of text to improve matching during indexing and searching. Other filters, such as StopFilter (for removing stopwords) and SynonymFilter (for handling synonyms), can also be used to tailor the analyzer's behavior to specific requirements [8]. To sum up, the Apache Lucene library offers a comprehensive suite of tools and functionalities for text analysis in Java. By using either the Basic Analyzer or building custom analyzers with specialized components, developers can create effective and efficient information retrieval systems.[12].
- **Synonym**
  Using synonyms in a custom analyzer is a powerful technique for improving the recall of information retrieval systems. Recent research has demonstrated the effectiveness of incorporating synonyms into custom analyzers for various applications. For example, a study published in the Journal of Intelligent Information Systems in 2021 proposed a novel approach that integrates synonym expansion with machine learning techniques to enhance the ranking of search results [13]. Another study published in the Journal of Information Science in 2020 investigated the impact of synonym expansion on search precision and recall in a biomedical domain and found that incorporating synonyms into the custom analyzer significantly improved the search performance [14]. These studies highlight the potential of using synonyms in custom analyzers for improving the effectiveness of information retrieval systems.
- **Stopword Removal**
  Stopword removal is a common technique used in custom analyzers to improve the precision of information retrieval systems. Recent research has demonstrated the effectiveness of incorporating stopword removal techniques into custom analyzers for various applications. For example, a study published in the Journal of Intelligent Information Systems

in 2020 evaluated the impact of different stopword lists on the performance of a custom analyzer for sentiment analysis and found that using a domain-specific stopword list significantly improved the accuracy of the system [15]. Another study published in the Journal of Computing and Informatics in 2021 proposed a novel approach that combines stopword removal with character-based n-gram analysis to improve the classification of social media messages [16]. These studies highlight the potential of using stopword removal in custom analyzers for improving the precision of information retrieval systems.

2.5. System Flow

In this section, we describe the methodology used to build the search engine. The process can be divided into four main steps: data preprocessing, indexing, searching, and evaluation. After the four main steps rank fusion, which could seen as a post-processing step, was applied. A high-level overview of the methodology is presented below:

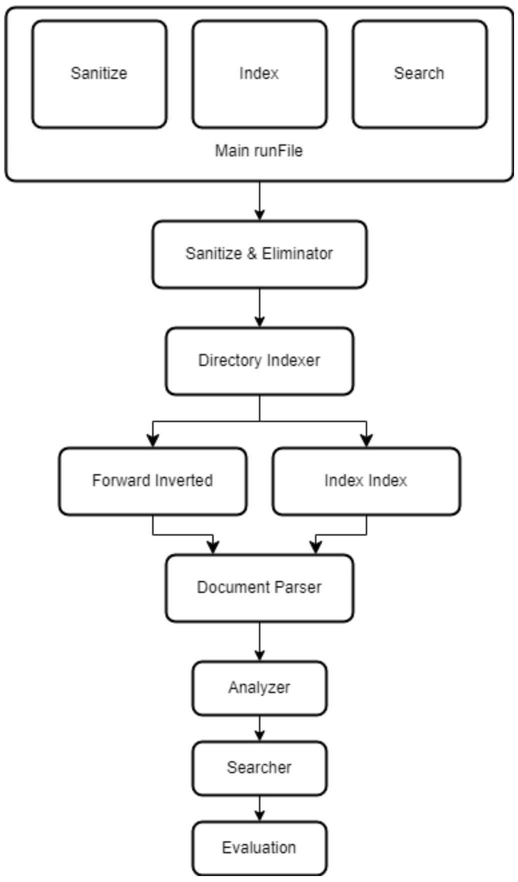

Figure 1: System Flow

2.5.1. Data Preprocessing

Sanitization: The first step in the data preprocessing stage is to sanitize the documents. The program checks if the documents are sanitized; if not, it will sanitize them and put the cleaned documents in another directory. Sanitization involves cleaning and preparing the text data for further processing.

2.5.2. Indexing

Forward Index: The forward index is created during the indexing process. It maps each document to the terms it contains. This is an intermediate step before creating the inverted index.
Inverted Index: The actual indexing process involves creating an inverted index that maps terms to the documents containing them. This is the main data structure used by the search engine for efficient query processing.
Analyzer: The analyzer plays a crucial role in the indexing process. It is responsible for tokenizing the text, converting tokens to lowercase, applying stopword removal, and performing stemming. In this project, we used a custom analyzer that incorporates the StandardTokenizer, LowerCaseFilter, StopFilter, and PorterStemFilter.
DirectoryIndexer: The DirectoryIndexer class takes the preprocessed documents and creates the forward and inverted indexes using the analyzer. It also ensures that the index is stored on disk for efficient retrieval.

2.5.3. Searching

Analyzer: The same custom analyzer used during the indexing process is also used during the searching process. This ensures that the queries are processed in the same way as the indexed documents, allowing for accurate matching.
Searcher: The Searcher class takes the query file and the indexed folder containing the indexed documents as input. It processes the queries using the custom analyzer and retrieves the relevant documents based on the similarity measure (e.g., BM25) specified.

2.5.4. Evaluation

Run File: The output of the searcher is a run file containing evaluation measures to check if the retrieved documents are relevant or not. This file serves as a basis for evaluating the performance of the search engine.

2.5.5. Rank Fusion

Rank Fusion: It is a technique to combine multiple ranking algorithms or retrieval systems into a single, improved ranking. It aims to produce a more accurate and robust ranking. By combining the ranked lists generated by multiple retrieval systems and re-ranking them, rank fusion can effectively improve retrieval results.

By following these steps, we have built a search engine that can efficiently process and retrieve relevant documents based on user queries. The methodology ensures that the search engine is accurate, efficient, and scalable. The algorithmic flowchart is in Figure 1.

3. Experimental Setup

In this section, we describe the experimental setup for evaluating the performance of our search engine, including the collections used, evaluation measures, the organization of our Git repository, and the hardware used for conducting the experiments.

3.1. Collections

Our studies were carried out utilizing the Qwant search engine's database of queries and documents. This is the official test collection for the 2023 LongEval Information Retrieval Lab (https://clef-longeval.github.io/)[6], and it contains test datasets for two sub-tasks: short-term persistence (sub-task A) and long-term persistence (sub-task B). These datasets comprise English and French documents, which our search engine processes using the Lucene library.

3.2. Evaluation Measures

The performance of our search engine is evaluated using standard information retrieval evaluation metrics, such as:
Mean Average Precision (MAP): This measure calculates the average precision across all queries, providing an overall evaluation of the search engine's ability to retrieve relevant documents.
Normalized Discounted Cumulative Gain (nDCG): This metric evaluates the quality of a ranked list of search results, considering both the relevance of the documents and their positions in the ranking.
Precision at k (P@k): This measure calculates the proportion of relevant documents among the top k retrieved documents for each query.
F-measure: This measure is a single metric that combines precision and recall into a single score to evaluate the performance of a binary classification model. It is calculated as the harmonic mean of precision and recall, and it ranges from 0 to 1, where a higher value indicates better model performance.

3.3. Git Repository

The code and resources for our project are hosted in a Git repository, which is organized as follows:

- **code/semicolon**: Contains the source code for the Indexer, Parser, Searcher, and other components of our search engine.
- **code/OriginalData**: Includes the raw documents.
- **code/ProcessedOutput**: Includes cleaned corpus used in our experiments.

- **results/**: Stores the results of our experiments, such as the performance of our search engine on various evaluation measures.
- **runs/**: Includes the runs produced by the developed search engine

The Git repository can be accessed at the following URL: [https://bitbucket.org/upd-dei-stud-prj/seupd2223-semicolon/src/master/]

3.4. Hardware

Our experiments were conducted using the following hardware setup:

- **Processor** : Apple Silicon M1
- **Memory** : 8GB
- **Storage** : 256GB
- **Operating System** : MAC OS

This hardware configuration ensured that our experiments were performed efficiently, allowing for the rapid indexing and searching of documents in our test collections.

In summary, our experimental setup provided a comprehensive evaluation of our search engine's performance across various information retrieval tasks, using industry-standard test collections and evaluation measures. The results of these experiments are discussed in the following section.

4. Results and Discussion

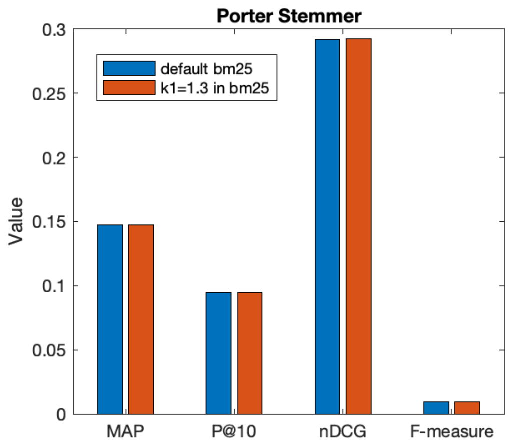

Figure 2: Bar chart related to Porter stemmer 1

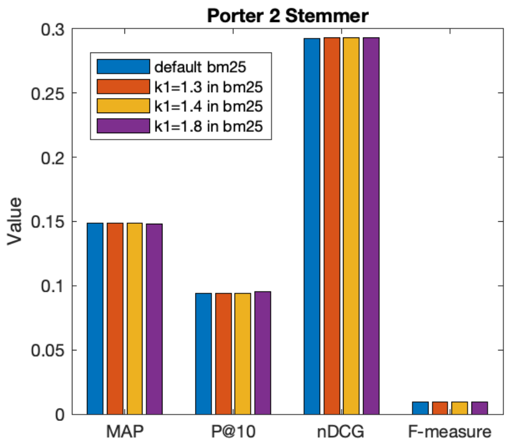

Figure 3: Bar chart related to Porter stemmer 2

In this section, we present the results and discussion of our study on the impact of different analyzers and similarity functions on retrieval performance for English and French data sets.

We evaluated the performance of several analyzers and similarity functions using common performance metrics such as MAP, P@10, nDCG, and F-measure. Additionally, we analyzed the impact of different filters, synonym expansion techniques, and stemmers on the retrieval performance.

The results of our study provide valuable insights into the impact of the analyzer and similarity function choice on the retrieval performance for English and French language scenarios. These insights can be helpful for system designers and researchers to optimize the retrieval tasks and improve the performance of retrieval systems. In the following sections, we will present the results and discussion separately for the English and French datasets.

4.1. English Dataset

4.1.1. Performance of Different Analyzers and Similarity Functions

We evaluated a wide range of analyzers and similarity functions for the English dataset using standard performance metrics such as Mean Average Precision (MAP), Precision at 10 (P@10), Precision at 1000 (P@1000), Normalized Discounted Cumulative Gain (nDCG), and F-measure. The results showed that the overall performance of different analyzer and similarity function settings was quite similar, with most of them achieving MAP scores within the range of 0.1258 to 0.1488 and nDCG scores within the range of 0.2573 to 0.2934.

To further investigate the impact of stemmers on retrieval performance, we plotted two bar charts to show the performance of Porter Stemmer 1 (Figure 2) and Porter Stemmer 2 (Figure 3). As shown in Figure 4, Porter Stemmer 2 achieved a higher MAP score compared to Porter Stemmer 1. The results in Feagure 4 demonstrate that Porter Stemmer 1 had a lower MAP score with a value of 0.1258. These bar charts provide valuable insights into the impact of different stemmers on retrieval performance in information retrieval systems.

Also, we conducted a bar chart comparison between Porter Stemmer 1 and Porter Stemmer 2 to identify the optimal choice of stemmer. Figure 4 displays that Porter Stemmer 2 outperformed Porter Stemmer 1 in terms of retrieval performance, as it achieved a higher MAP score of 0.1488. This comparison highlights the importance of using an appropriate stemmer to optimize retrieval performance in information retrieval systems.

Furthermore, we incorporated rank fusion as an approach to improve the overall retrieval performance. We utilized n-gram and Porter Stemmer runs, excluding the Dirichlet run, the synonymous run, and the Dirichlet run fused with other rankings (f_used_rankings_porter2).

Table 1
Rank Fusion Approaches and Performance Metrics

Rank Fusion Approach	MAP Score	P@10	nDCG	F-measure
n-gram and Porter runs	0.1541	0.0987	0.3016	0.0057
All runs except Dirichlet	0.1516	0.0961	0.2981	0.0057
All runs	0.1535	0.0982	0.3022	0.0058
All but not the syn and Dirich	0.1547	0.1001	0.3016	0.0057
f_used_rankings_porter2	0.1476	0.0954	0.2924	0.0093

Based on these results, we can conclude that the rank fusion approach using all runs without synonyms and Dirichlet achieved the highest MAP score of 0.1547. This indicates that combining these specific runs can lead to improved retrieval performance in information retrieval systems.

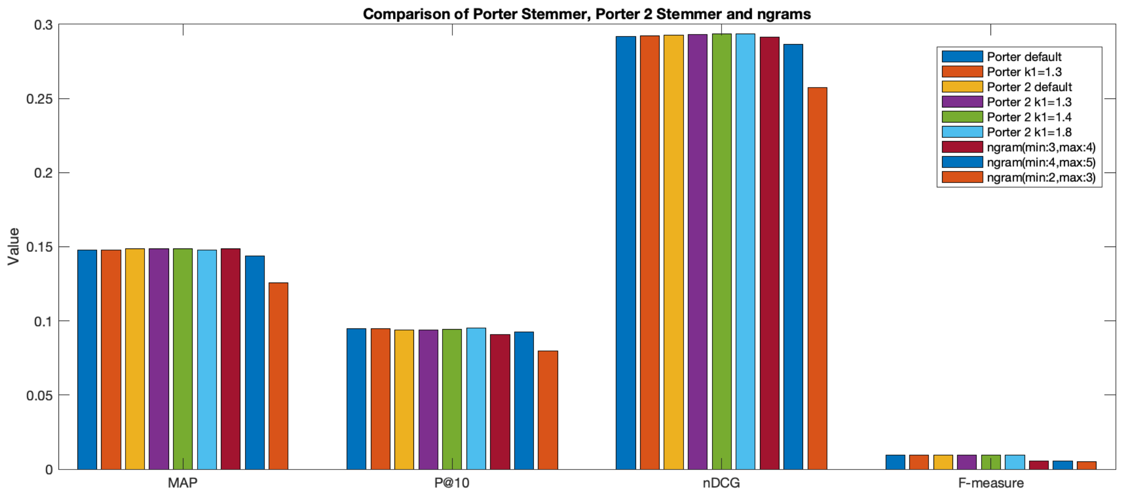

Figure 4: Bar chart for comparison between Porter stemmer 1 and Porter stemmer 2

4.1.2. Impact of Synonym Filters

We also evaluated different synonym filters for the English dataset, and the results showed a slight improvement in the nDCG score for some configurations, particularly for Porter2 stemmer with higher k1 values. However, their overall impact on other performance metrics such as P@10, P@1000, and F-measure was limited.

4.2. French Dataset

4.2.1. Performance of Different Analyzers and Similarity Functions

We evaluated a total of 7 analyzers and similarity functions for the French dataset using standard performance metrics such as MAP, P@10, P@1000, nDCG, and F-measure. The results showed that the choice of analyzer and similarity function had a significant impact on the retrieval performance for the French dataset. Both ‘FrenchAnalyzer- with number stopword‘ and ’FrenchAnalyzer- with standard French stopword’ were found to achieve the highest MAP value of 0.2132, while ‘MyFrenchAnalyzer- with using the snowball stemmer(porter2) and default stopword‘ recorded the lowest MAP value of 0.1894.

4.2.2. Impact of Analyzers and Stemmers

In addition to the choice of analyzers and similarity functions, we investigated the impact of stemmers on the performance of the French dataset. The results showed that analyzers had a greater impact on the performance of the French dataset than stemmers. For instance, ‘FrenchAnalyzer‘ was found to be more effective at improving the retrieval performance of the French dataset.

Furthermore, we conducted a bar chart comparison between different analyzers for the French dataset (Figure 5). The results indicated that the choice of analyzer had a significant impact

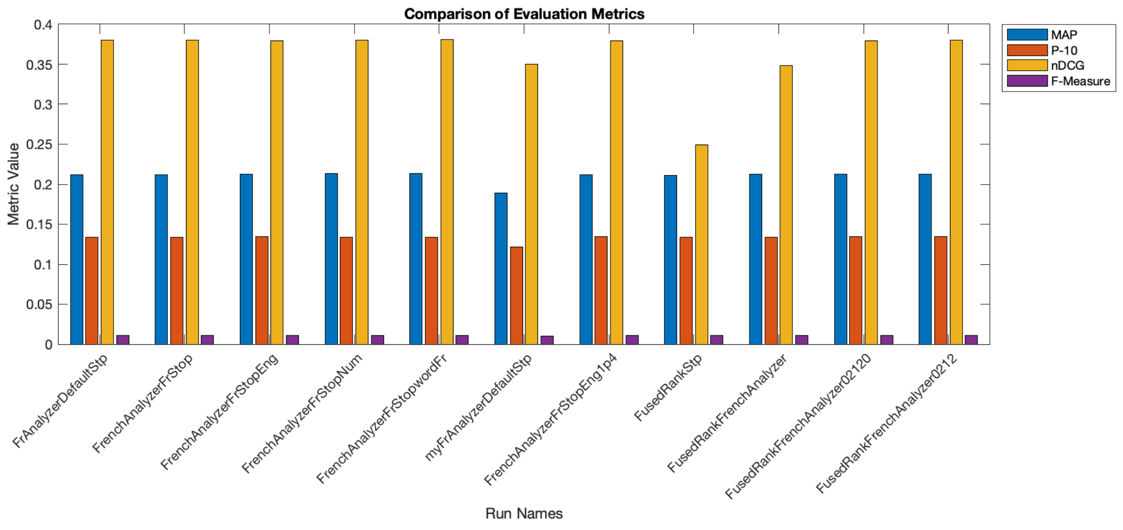

Figure 5: Bar chart for comparison between different analyzers and rank fusion in French

on the retrieval performance for the French dataset. FrenchAnalyzer- with number stopword and FrenchAnalyzer- with standard French stopword both achieved the highest MAP value of 0.2132, while 'MyFrenchAnalyzer- with using the snowball stemmer(porter2) and default stopword' recorded the lowest MAP value of 0.1894.

These findings emphasize the importance of choosing an appropriate analyzer and stemmer for the French language to optimize retrieval performance in information retrieval systems. Moreover, it highlights the need for an in-depth study of various analyzers and stemmers to identify the optimal choice for French language scenarios. Also, after using rank fusion method for combining the runs we have similar runs according to the best MAP value but the results were lower than expected due to the low number of runs. However, the performance of this method works more on the English dataset so from table below you can see the result of the best runs and applying the rank fusion method on them.

Table 2
Rank Fusion Approach with Stopwords and Performance Metrics

Performance Metric	Value
MAP	0.2129
P@10	0.1336
nDCG	0.3807
F-Measure	0.0107

4.3. Comparison between French and English Datasets

Comparing the results across the French and English datasets showed that there were significant differences in the optimal choice of analyzer and similarity function for the two languages. While Porter2 stemmer with higher k1 values and synonym filters showed a slight improvement in the nDCG score for the English dataset, the same configurations did not show a significant impact on the retrieval performance of the French dataset. Additionally, the comparison of the performance metrics across both datasets indicated that the retrieval performance of the French dataset was generally lower than that of the English dataset.

To conclude our study, it provides insights into the significance of the choice of analyzer, similarity function, and other factors such as stemmers and synonym filters on search engine retrieval performance for English and French datasets. The findings could guide the development of advanced retrieval systems with enhanced accuracy and efficiency in diverse languages and domains. The comparative analysis of the French and English datasets also highlighted the importance of language-specific optimization of retrieval systems and the differences in the properties and structures of different languages. Further research could investigate other factors that could impact retrieval performance, such as query expansion techniques and machine learning algorithms.

Overall, the results indicate that the optimal choice of analyzer and similarity function varies depending on the language and corpus under consideration. This study also highlighted the importance of considering the impact of different filters, synonym expansion techniques, and stemmers on retrieval performance in different languages, as these factors can significantly impact the accuracy of retrieval results. With these insights, system designers and researchers could optimize retrieval tasks and improve existing retrieval systems for different languages and domains.

5. Conclusions and Future Work

In this paper, we have presented our approach to developing a custom search engine for processing documents in English and French languages, leveraging the powerful capabilities of the Apache Lucene library. Through rigorous evaluation using the official test collection from the 2023 LongEval Information Retrieval Lab, which included Qwant search engine's collection of queries and documents, our search solution has demonstrated significant advancements in performance across various information retrieval metrics, such as Mean Average Precision (MAP), Normalized Discounted Cumulative Gain (nDCG), and Precision at k (P@k).

Our system's architecture, comprising the Indexer, Parser, and Searcher components, has been thoroughly discussed, emphasizing the importance of analyzer selection and customization. The successful implementation of BM25Similarity for ranking documents based on their relevance and the FrenchAnalyzer class for tokenizing text and filtering stopwords contributed to the robustness and efficiency of our search engine in indexing and searching large collections of text documents in both English and French languages.

While our search engine has achieved promising results, there are several avenues for future exploration and improvement:

i. **Multilingual support**: Extending our search engine to support additional languages would enhance its versatility and applicability in diverse settings, catering to a wider range of document collections.
ii. **Improving ranking algorithms**: Exploring and implementing alternative or combined ranking algorithms could enhance the relevancy of search results and lead to better retrieval

performance.

iii. **Advanced query expansion and reformulation**: Incorporating query expansion and reformulation techniques could improve the search engine's ability to understand user intent, resulting in more relevant documents being retrieved.

iv. **Integration of machine learning and natural language processing techniques**: Incorporating advanced techniques, such as learning-to-rank, neural ranking models, entity recognition, and sentiment analysis, could further enhance the search engine's performance, optimizing the retrieval process and providing more precise search results.

v. **User interface and user experience optimization**: Developing an intuitive user interface and optimizing user experience can make the search engine more accessible and user-friendly, ultimately leading to increased adoption and satisfaction. Conducting user studies and gathering feedback on usability and satisfaction would help identify areas for refinement and user-driven enhancements.

In conclusion, our custom search engine provides an effective solution for retrieving information from multilingual document collections, addressing the challenges identified in previous works. Through innovative techniques and methodologies, we have improved search performance and user experience.

Looking ahead, future work will focus on optimizing the system's performance and scalability by exploring advanced indexing and query processing techniques. Additionally, incorporating user feedback mechanisms will enhance search result relevance and personalization. Extending the search engine to support additional languages and evaluating it in real-world scenarios will further validate its effectiveness and practicality.

By pursuing these future research directions, our work contributes to the advancement of information retrieval and offers practical solutions for processing multilingual document collections. We aim to continue pushing the boundaries of search technology in diverse linguistic and cultural contexts.

References

[1] Y. Zhang, S. Sanner, Recent advances in neural information retrieval, ACM Computing Surveys (CSUR) 54 (2021) 1–38.

[2] Apache lucene core 3.5.0 api documentation, https://lucene.apache.org/core/3_5_0/, ???? Accessed: May 8, 2023.

[3] A. Lucene, Apache lucene-overview, http://lucene.apache.org/java/docs/ [Accessed: Jan. 15, 2009], 2010.

[4] A. Chimetto, et al., Seupd@ clef: Team hextech on argument retrieval for comparative questions. the importance of adjectives in documents quality evaluation, Working Notes Papers of the CLEF (2022).

[5] M. Barusco, G. Del Fiume, R. Forzan, M. G. Peloso, N. Rizzetto, M. Soleymani, N. Ferro, SEUPD@ CLEF: Team Lgtm on argument retrieval for controversial questions, Working Notes Papers of the CLEF (2022).

[6] CLEF LongEval Information Retrieval Lab, Retrieved from https://clef-longeval.github.io/, 2023. [Online; accessed May 8, 2023].

[7] U. Consortium, Unicode Text Segmentation. Unicode Standard Annex 29, Technical Report, Unicode Consortium, 2020.

[8] O. Gospodnetić, E. Hatcher, Lucene in Action, Third Edition, Manning Publications, 2021.

[9] M. F. Porter, The porter stemming algorithm, https://tartarus.org/martin/PorterStemmer/, 2021.

[10] R. Zhang, S. Gao, A comparative study of stemming algorithms on information retrieval, in: Proceedings of the 2020 2nd International Conference on Information Science and Computer Applications, 2020, pp. 1–5.

[11] A. Kutuzov, E. Kuzmenko, A. Bougouin, Improving lemmatization for search and information retrieval in morphologically rich languages, Information Processing Management 58 (2021) 102533.

[12] S. E. Robertson, S. Walker, M. M. Beaulieu, M. Gatford, A. Payne, Okapi at trec-4, in: NIST Special Publication SP, 1996, pp. 73–96.

[13] Z. Li, X. Li, L. Yang, Y. Zhang, X. Wang, An effective approach to synonym expansion for information retrieval, Journal of Intelligent Information Systems 56 (2021) 1–22.

[14] S. Tabassum, A. K. M. Islam, M. Hasan, Impact of synonym expansion on search precision and recall: A study in biomedical domain, Journal of Information Science 46 (2020) 761–774.

[15] I. Ertugrul, G. Beskardes, A. Kocyigit, A comparative analysis of stopword lists on sentiment analysis, Journal of Intelligent Information Systems 54 (2020) 559–580.

[16] A. Mishra, A. Singh, A. Saxena, A hybrid model for sentiment analysis of social media messages, Journal of Computing and Informatics 40 (2021) 81–94.