

SEUPD@CLEF Task 1: Information retrieval for English and French documents: Team JIHUMING

Jesús Moncada-Ramírez¹, Isil Atabek¹, Huimin Chen¹, Michele Canale¹,
Nicolò Santini¹ and Giovanni Zago¹

Abstract

Our group will propose a novel search engine for the Longitudinal Evaluation of Model Performance (LongEval) task at CLEF 2023 [1]; it will also be the final work of the subject Search Engines at the University of Padova. Our system focuses on the short-term and long-term temporal persistence of the systems' performance, for a collection of both English and French documents. Our approach involves analyzing both English and French versions of the documents using whitespace tokenization, stopword removal and stemming. We generate character N-grams to identify recurring word structures (as prefixes or suffixes) repeated over documents. We also use query expansion with synonyms (in English) and some Natural Language Processing (NLP) techniques as Named Entity Recognition (NER) to further refine our system. The similarity function utilized in our approach is BM25. Our system was developed in Java and primarily utilized the Lucene library. After extensive experiments on these techniques, we came up with five systems that have produced the best results in terms of MAP and NDCG scores.

Keywords

CLEF 2023, Information retrieval, LongEval, English, French, Search Engines

1. Introduction

This report aims at providing a brief explanation of the Information Retrieval system built as a team project during the Search Engine course 22/23 of the master's degree in Computer Engineering and Data Science at the University of Padua, Italy. As a group in this subject, we are participating in the 2023 CLEF LongEval: Longitudinal Evaluation of Model Performance [1]. This annual evaluation campaign focuses on the longitudinal evaluation of model performance in information retrieval and natural language processing.


The LongEval collection [2] relies on a large set of data provided by Qwant (a commercial privacy-focused search engine that was launched in France in 2013). Their idea regarding the dataset (collected in June 2022) was to reflect changes of the Web across time, providing evolving document and query sets. The training collection consists of 672 **queries**, 98 held-out queries, and 9656 evaluation assignments. The **documents** were chosen based on queries using the

"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

✉ jesus.moncadaramirez@studenti.unipd.it (J. Moncada-Ramírez); isil.atabek@studenti.unipd.it (I. Atabek); huimin.chen@studenti.unipd.it (H. Chen); michele.canale.1@studenti.unipd.it (M. Canale); nicolo.santini.1@studenti.unipd.it (N. Santini); giovanni.zago.3@studenti.unipd.it (G. Zago)



© 2023 Copyright for this paper by its authors.
Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

Qwant click model, in addition to random selection from the Qwant index. The training queries are categorized into twenty **topics**, such as: car-related, antivirus-related, employment-related, energy-related, recipe-related, etc. In addition to the original French version, the collection also includes English translations of the documents and queries using the CUBBITT [3] system.

The paper is organized as follows: Section 2 briefly describes our approach; Section 3 describes our code in detail; Section 4 explains our experimental setup; Section 5 discusses our main findings; finally, Section 6 draws some conclusions and outlooks for future work.

2. Methodology

Our final search engine can be divided into the following parts: parsing of the documents and queries, indexing, text processing (analyzers), and run generation (effective search).

Document parsing was performed using the JSON version of the documents. On the other hand, query parsing was based on an XML parser. See Section 3.1 for more details.

In the index, we decided to include four fields: (1) the (processed) English version of the documents, (2) the (processed) French version, (3) character N-grams of both versions concatenated, and (4) some NER information extracted from the French (original) version. As similarity function we have used BM25 [4] as it takes into account both term frequency and document length. See Section 3.3 for more details.

The text added to the fields must first be processed, for this we have developed four different analyzers. The English analyzer is based on whitespace tokenization, breaking of words and numbers based on special characters, lowercasing, applying the Terrier [5] stopword list, query expansion with synonyms based on the WordNet synonym map [6], and stemming. The French analyzer is based on whitespace tokenization, breaking of words and numbers based on special characters, lowercasing, applying a French stopword list [7] and stemming. To generate the character N-grams we consider only the letters of the documents (i.e. we discard numbers and punctuation). To perform NER we apply NLP techniques based on Apache OpenNLP [8]) to the original (French) version of the documents. Specifically, we used NER applied to locations, person names and organizations. See Section 3.2 for more details.

We conducted some experiments to generate the runs, i.e., we have tried different combinations of the explained techniques. Thus, our searcher will always use BM25 [4], but the rest of characteristics depend on the run it is generating. See Section 4 for more details.

3. System Architecture

In this section, we address the technical aspects of how our system was developed. Some parts of the code we are going to present are inspired in the repositories developed by Professor

Nicola Ferro and presented to us during the Search Engine course. We will follow the structure (in packages) of the repository [9].

3.1. Parsing

To generate an index based on the provided documents, first, we have to parse them; this is, get their text into Java data structures. All this parser package has been developed following the instructions provided by the organizers.

We decided to use the **JSON** version of the documents because they can easily be manipulated and queried using a variety of tools and libraries. In our case, because of the large number of documents we are working with, we had to create a **streaming** parser, that allocates into the main memory only one document at a time. Thus, our parser is based on the Java library Gson including streaming functionalities.

The whole parser is made up of the following classes:

- `DocumentParser`: an abstract class representing a streaming parser. Implements `Iterator` and `Iterable`.
- `JsonDocument`: a Java POJO for the deserialization of JSON documents.
- `ParsedDocument`: represents a document already parsed. Note that this class only contains an identifier and a body.
- `LongEvalParser`: real parser implementing the class `DocumentParser`. Here is where the streaming logic is implemented. Objects of this class can be used as iterators that yield parsed documents.

3.2. Analyzer

To process the already parsed documents' text, we have implemented our own Lucene analyzers. All of them follow the typical workflow: use a `Tokenizer` and a list of `TokenFilter` to a `TokenStream`.

The final version of the project creates an index with four fields for each document. Because of this, we had to create **four different analyzers**. Note that we are not taking into account the first field which is the id, obviously presented in the index, to which no processing is applied. All the following described analyzers use some functionalities from the helper class `AnalyzerUtil` developed by Nicola Ferro. We will explain them independently.

3.2.1. English body field

The processing applied to the English version of the documents (in the class `EnglishAnalyzer`) is the following:

1. Tokenize based on whitespaces.
2. Eliminate some strange characters found in the documents. It is unlikely that a user would perform a query including these characters.
3. Delete punctuation marks at the beginning and end of words. Necessary as we are using the whitespace tokenizer (for example: "address," or "city.").
4. Apply the `WordDelimiterGraphFilter` Lucene filter. It splits words into subwords and performs optional transformations on subword groups. We decided to include the following operations:
 - a) Divide words into different subwords based on the lower/upper case. Example: "PowerShot" converted into tokens "Power" and "Shot".
 - b) Divide numbers into different parts based on special characters located in intermediate positions. Example: "500-42" converted into "500" and "42".
 - c) Concatenate numbers with special characters located in intermediate positions. Example "500-42" converted into "50042".
 - d) Remove the trailing "s" of English possessives. Example: "O'Neil's" converted into "O" and "Neil".
 - e) Always maintain the original terms. Example: the tokens "PowerShot", "500-42", and "O'Neil's" will be maintained.
5. Lowercase all the tokens.
6. Apply the Terrier [5] stopword list.
7. Apply query expansion with synonyms using `SynonymTokenFilter` from Lucene, which is based on the WordNet synonym map [6]. A maximum of 10 synonyms can be added to each term.
8. Apply a minimal stemming process using `EnglishMinimalStemFilter` from Lucene.
9. Delete tokens that may have been left empty because of the previous filters. For this, we created a custom token filter, `EmptyTokenFilter` implementing `FilteringTokenFilter`.

3.2.2. French body field

The processing of French documents (in the class `FrenchAnalyzer`) is identical to the processing of English documents in the first 5 points (excluding the English possessives' removal in 4.d). For this point on, we apply:

6. Apply a French stopword list [7].
7. Apply a minimal stemming process (in French) using `FrenchMinimalStemFilter` from Lucene.
8. Delete empty tokens (`EmptyTokenFilter`).

3.2.3. Character N-grams

Character N-grams are created in the analyzer class `NGramAnalyzer`, which performs the following operations:

1. Tokenize based on whitespaces.
2. Lowercase all the tokens.
3. Delete all characters except letters. Note that we also maintain the French accent letters. After conducting tests on character N-grams with numbers, which resulted in many meaningless numbers being generated, we made the decision to avoid including them.
4. Delete empty tokens (`EmptyTokenFilter`).
5. Generate character N-grams using `NGramTokenFilter` from Lucene.

The value of N has not been fixed in order to allow for the generation of different experiments. See Section 4 for more details.

3.2.4. NER extracted information

The NER information has been extracted using the Apache OpenNLP [8] library. As Lucene does not include these functionalities directly, we have used a modified version of a token filter developed by Nicola Ferro based on the mentioned library, (`OpenNLPNERFilter`).

The processing of the tokens in this analyzer (`NERAnalyzer`) is the following:

1. Tokenize using a standard tokenizer, `StandardTokenizer` from Lucene.
2. Apply NER using a tagger model for locations.
3. Apply NER using a tagger model for person names.
4. Apply NER using a tagger model for organizations and companies.

3.3. Index

During the initial stages of the project, we developed an indexer that took into account only one version of the documents (either English or French); this can be found in the class `DirectoryIndexer`. As soon as we decided to consider both versions of the documents, we marked it as `Deprecated` and developed the `MultilingualDirectoryIndexer` class, that is the final version of our indexer.

As previously mentioned, `MultilingualDirectoryIndexer` is used for indexing multilingual documents, but also for retrieving basic statistics about the resulting index's vocabulary. To create an instance of this indexer, several parameters must be provided, such as the paths to the directories where the English and French documents are stored, the path to the directory

where the index will be created, and the expected number of documents to be indexed. It must also receive instances of our custom analyzers, i.e., `EnglishAnalyzer`, `FrenchAnalyzer`, `NGramAnalyzer`, and `NERAnalyzer`, which will be used for tokenizing and processing the text of the documents. In addition, we must provide the similarity function used for indexing (we decided to use BM25) and the size of the RAM buffer used during indexing.

During indexing, `MultilingualDirectoryIndexer` reads the documents from the directory of the English files, the directory of the French files and processes them using the specified analyzers to create an inverted index. Note that for the process to conclude properly, both directories must contain the same number of files, the files must contain the same number of documents and the documents must have the same ID. In other words, at each iteration, the indexer accesses the two versions (English and French) of the same document, to create a single Lucene document in the index.

After indexing, we have used a method to print some statistics about the resulting index's vocabulary. This method prints the total number of unique terms, the total number of terms and a list of terms with their frequency for both English and French vocabularies. This functionality has proven to be highly useful as it provides an overview of the indexed vocabulary. This overview can be utilized for further analysis and optimization of the search system. Another useful feature of the indexer is the ability to estimate the remaining time required for the indexing process. This is particularly valuable as we have observed that these processes can be time-consuming.

3.4. Search

The search package (only made up of the class `Searcher`) is in charge of doing the effective search, i.e., searching in the created indexes for the topics/queries specified. In our case we will search the queries provided from LongEval [2].

The searcher will first apply the explained analyzers to the query title, in order to generate a text that can be matched with the corresponding index fields. This process is performed by relying on the `QueryParser` class from Lucene. After this, the title of the query is searched in the appropriate index fields using the similarity function BM25.

To perform a query, we must specify: the path of the index, the path of the topics file, the number of expected topics, a run descriptor, and the maximum number of documents to be retrieved (1000). To facilitate the execution of all experiments, we have created a menu where users can select the desired run to execute after specifying the required paths and parameters. This menu determines which document fields and analyzers shall be used, depending on the run identifier. It also allows distinguishing between train and test data.

3.5. Topic

To read the queries (in TREC format) we couldn't use the class **TrecTopicsReader** already included in Lucene. This class expects queries to be in a more specific format than the one provided by the organizers. Thus, we developed our LongEval topic reader in the package `topic`, containing the classes `LongEvalTopic` and `LongEvalTopicReader`. Note that we kept the name "TopicReader", but what our `LongEvalTopicReader` does is reading all the queries, not the topics. Thus:

- `LongEvalTopic` is a simple Java POJO representing each query provided by LongEval. Each query has a number (`<num>`) and a title (`<title>`). It is the equivalent of `QualityQuery` when working with **TrecTopicsReader**.
- `LongEvalTopicReader` is the query reader we developed. It considers the query file as an XML file and parses it using the Java XML library.

4. Experimental Setup

Our work was initiated based on the experimental setups outlined below.

- Evaluation measures: MAP (Mean Average Precision) and NDCG (Normalized Discounted Cumulative Gain) scores.
- [9, Repository].
- During the development and the experimentation, personal computers were used.
- Java JDK version 17, Apache version 2, Lucene version 9.5, and Maven.

In order to do different run experiments our team has created several indexes from the provided collection during the development of the final version of the project. In other words, the first created indexes only include several of the characteristics explained in this report, while the last indexes correspond to the final version of the project.

All the created indexes are multilingual, which allows us to take full advantage of the (bilingual) data collection. Additionally, we did some experiments with character 3-grams, 4-grams and 5-grams. Our motivation for experimenting with this was to compare how the size of different character N-grams affect to the effectiveness of our system. 3-grams are able to collect more specific information, while 4-grams and 5-grams allow considering bigger structures with more context and information. An additional functionality of some indexes is query expansion, but as commented, this is only applied to the English body. Finally, we created indexes with NER, which provides not only the search for keywords but also identifying and extracting specific named entities.

The subsequent indexes are:

- 2023_04_24_multilingual_3gram: both languages of documents, using character 3-grams.
- 2023_04_29_multilingual_3gram_synonym: both languages, character 3-grams, (English) query expansion with synonyms.
- 2023_05_01_multilingual_4gram_synonym: both languages, character 4-grams, (English) query expansion with synonyms.
- 2023_05_01_multilingual_5gram_synonym: both languages, character 5-grams, (English) query expansion with synonyms.
- 2023_05_05_multilingual_4gram_synonym_ner: both languages, character 4-grams, (English) query expansion with synonyms, NER techniques.

The indexes also can be found in the following [Google Drive](#) folder.

After creating indexes, we were able to conduct multiple runs to evaluate the effectiveness of our system. These runs not only experiment with some of the techniques specified here, but also consider different versions (English or French version) of the queries. With them we can compare and analyze different aspects of our system's performance, such as precision and recall. We then computed the MAP and NDCG scores for each run, which allowed us to further evaluate the performance of our system. The results will be commented in the Section 5. The runs are the following:

- seupd2223-JIHUMING-01_en_en: English topics; using English body field.
- seupd2223-JIHUMING-02_en_en_3gram: English topics; using English body field and 3-gram field.
- seupd2223-JIHUMING-03_en_en_4gram: English topics; using English body field and 4-gram field.
- seupd2223-JIHUMING-04_en_en_5gram: English topics; using English body field and 5-gram field.
- seupd2223-JIHUMING-05_en_en_fr_5gram: English topics; using English and French body fields and 5-gram field.
- seupd2223-JIHUMING-06_en_en_4gram_ner: English topics; using English body field, 4-gram field and NER technique.
- seupd2223-JIHUMING-07_fr_fr: French topics; using French body field.
- seupd2223-JIHUMING-08_fr_fr_3gram: French topics; using French body field and 3-gram field.
- seupd2223-JIHUMING-09_fr_fr_4gram: French topics; using French body field and 4-gram field.

Table 1

MAP and NCDG scores for all runs

Index	Run	MAP Score	NCDG Score
01	en_en	0.0700	0.1614
02	en_en_3gram	0.0704	0.1661
03	en_en_4gram	0.0874	0.2025
04	en_en_5gram	0.1028	0.2288
05	en_en_fr_5gram	0.0669	0.1525
06	en_en_4gram_ner	0.0360	0.1098
07	fr_fr	0.1656	0.3135
08	fr_fr_3gram	0.1698	0.3208
09	fr_fr_4gram	0.1737	0.3269
10	fr_fr_5gram	0.1748	0.3285
11	fr_en_fr_5gram	0.1288	0.2797
12	fr_fr_4gram_ner	0.1362	0.2881

- seupd2223-JIHUMING-10_fr_fr_5gram: French topics; using French body field and 5-gram field.
- seupd2223-JIHUMING-11_fr_en_fr_5gram: French topics; using English and French body fields and 5-gram field.
- seupd2223-JIHUMING-12_fr_fr_4gram_ner: French topics; using French body field, 4-gram field and NER technique.

The process of creating the indexes typically took around 1 hour, with the exception of the indexes that included NER, which took approximately 16 hours. On the other hand, generating the runs was a much quicker process, taking consistently less than a minute and a half to complete.

5. Results and Discussion

The analysis shows that the highest MAP score (0.1748) is achieved by fr_fr_5gram, followed by fr_fr_4gram (0.1737) and fr_fr_3gram (0.1698), while the lowest MAP score (0.0360) is obtained by en_en_4gram_ner. Similarly, the highest NDCG score (0.3285) belongs to fr_fr_5gram, followed by fr_fr_4gram (0.3269), whereas the lowest NDCG score (0.1098) corresponds to en_en_4gram_ner.

Results suggest that French queries perform better than their English counterparts, possibly due to the training data's French origin and later translation into English. Moreover, the IR system's effectiveness generally increases with a larger N-gram size, as indicated by the higher scores of en_en_5gram and fr_fr_5gram. Conversely, the inclusion of NER in the indexing process seems to have a negative impact on the scores, as shown by the lower scores of en_en_4gram_ner and fr_fr_4gram_ner. The use of query expansion with synonyms in

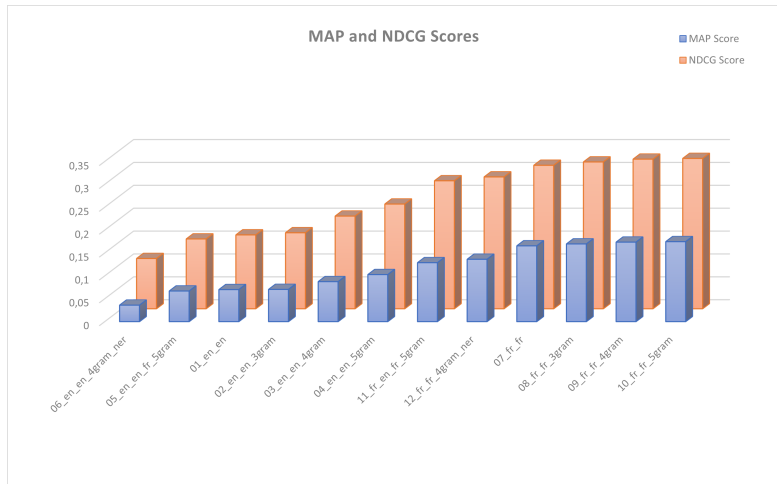


Figure 1: All scores sorted by MAP score

English does not seem to improve the search results to any great extent.

Here we can see a chart ranking of the five best scores, they are the runs that have been presented at CLEF:

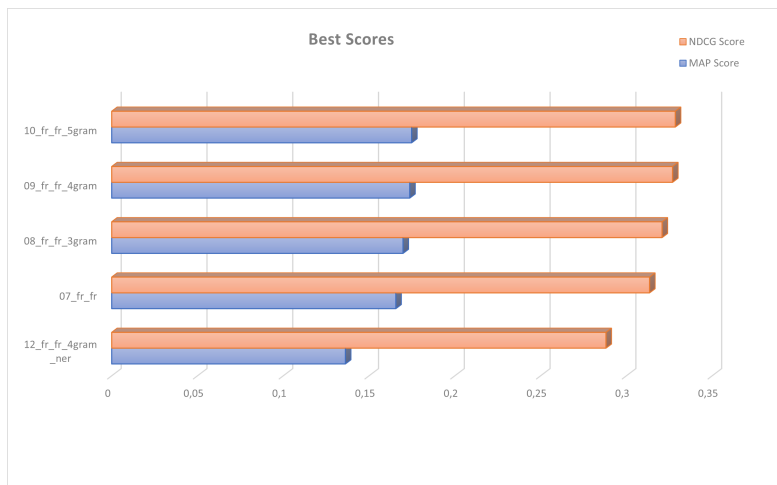


Figure 2: Best MAP and NDCG scores

It's interesting to notice that the cross-language approaches (en_en_fr_5gram and fr_en_fr_5gram) are out of the five bests systems. It turns out that searching for English words in French documents and vice versa messes up the search, lowering the score. Another interesting aspect is that the worst-performing index is the one with named entity recognition in English (en_en_4gram_ner): it combines translated queries and NER, which appears to be the two worst-performing approaches.

In general, we focus more on trying multiple approaches, this is why our score has such a big space for improvement. As already said, French queries with bigger N-gram sizes perform better. Instead of relying on single-word matches, the queries could take place with more context, resulting in better search results.

Following the competition workflow, we created the indexes based on the test data and re-executed the top five runs(see Figure 2). These runs will be the ones delivered to CLEF.

6. Conclusions and Future Work

In summary, the IR systems developed in this study followed the Parsing-Analyzer-Index-Search-Topic paradigm and utilized different methodologies, among which the following stand out: processing of English documents based on whitespace tokenization, the TERRIER stopword list, query expansion and stemming; processing of French documents based on whitespace tokenization, a stopword list and stemming; character N-grams of both versions concatenated; and NER information extraction using NLP techniques.

We evaluated the performance of the 12 systems we developed by measuring the effectiveness of runs on the training data, comprising both French and translated English queries and documents. To assess the quality of these runs, we used the MAP and NDCG scores calculated by `trec_eval`. Among these systems, five models performed the best, namely `fr_fr_5gram`, `fr_fr_4gram`, `fr_fr_3gram`, `fr_en_fr_5gram`, and `fr_fr_4gram_ner`, listed in order of their scores from highest to lowest for both MAP and NDCG.

In terms of future work, there are several areas that could be explored to improve the effectiveness of the developed IR systems. Firstly, we could improve indexing methodologies, such as increasing the value of N of N-gram, as we have commented on in Section 5. Secondly, we could explore better NLP techniques to improve the accuracy of the IR systems, as NER turns out not to be very effective.

One last possible future work could be a machine-learning based IR system. Using training data, we could train a model to predict the best N for N-grams, the best analyzer, and the best index for a given query and document. This would be a more dynamic approach to IR systems, as it would be able to adapt to different types of queries and documents.

References

- [1] CLEF2023, LongEval CLEF 2023 Lab, <https://clef-longeval.github.io/>, 2023.
- [2] CLEF2023, LongEval Train Collection, <https://lindat.mff.cuni.cz/repository/xmlui/handle/11234/1-5010>, 2023.
- [3] M. Popel, M. Tomkova, J. Tomek, Ł. Kaiser, J. Uszkoreit, O. Bojar, Z. Žabokrtský, Trans-forming machine translation: a deep learning system reaches news translation qual-

- ity comparable to human professionals, *Nature Communications* 11 (2020) 1–15. URL: <https://www.nature.com/articles/s41467-020-18073-9>. doi:10.1038/s41467-020-18073-9.
- [4] S. E. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, M. Gatford, Okapi at trec-3, in: *Text Retrieval Conference*, 1994.
 - [5] I. Ounis, G. Amati, V. Plachouras, B. He, C. Macdonald, C. Lioma, Terrier: A High Performance and Scalable Information Retrieval Platform, in: M. Beigbeder, W. Buntine, W. G. Yee (Eds.), *Proc. of the ACM SIGIR 2006 Workshop on Open Source Information Retrieval (OSIR 2006)*, 2006.
 - [6] Princeton University, About WordNet, <https://wordnet.princeton.edu/download/current-version>, 2005.
 - [7] G. Diaz, A. Suriyawongkul, Stopword list in French, <https://github.com/stopwords-iso/stopwords-fr/tree/master>, 2023.
 - [8] Apache, Apache OpenNLP, <https://opennlp.apache.org/>, 2023.
 - [9] Atabek, Canale, Chen, Moncada-Ramirez, Santini and Zago, JIHUMING, <https://bitbucket.org/upd-dei-stud-prj/seupd2223-jihuming/src/master/>, 2023.