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

Jesús Moncada-Ramírez[1], Isil Atabek[1], Huimin Chen[1], Michele Canale[1], Nicolò Santini[1] and Giovanni Zago[1]

Abstract

Our group will propose an original and efficient information retrieval system for Longitudinal Evaluation of Model Performance (LongEval) by CLEF2023[1]. The focus is on short term and long term temporal persistence of the systems' performance, for both English and French documents. The aim is to find a model giving good results for longitudinal evolving benchmarks, for the subject Search Engines, University of Padova.

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 masterś degree in Computer Engineering and Data Science at University of Padua, Italy. As a group in this subject, we are participating in the 2023 CLEF LongEval: Longitudinal Evaluation of Model Performance. This is the annual evaluation campaign that 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 was to reflect changes of the Web across time, providing evolving document and query sets. The data was collected in June 2022. The dataset consists of 672 training **queries**, 9656 corresponding evaluation assignments, and 98 held-out queries. The collection's **documents** were chosen based on queries using the Qwant click model, in addition to random selection of documents from the Qwant index. The training queries are categorized into twenty **topics**, including car-related, antivirus-related, employment-related, energy-related, recipe-related, rental car-related, video-related, war-related, gateau-related, police-related, tax-related, solar panel-related, water-related, elderberry-related, curtain-related, land-related, retirement-related, coronavirus-related, beauty-related, and miscellaneous queries. In addition to the original French

"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)

CEUR Workshop Proceedings (CEUR-WS.org)

versions, the collection also includes English translations of the webpages and queries using the CUBBITT system.
Our approach considers both the English and French versions of the documents. We generate character N-grams to identify common word structures (as prefixes or suffixes) repeated over documents. We also use query expansion with synonyms (in English) and the Natural Language Processing (NLP) technique of Named Entity Recognition (NER) to further refine our system. Our system was developed in Java, mainly using the library Lucene.
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

In the index, we decided to include four fields: (1) the English version of the documents, (2) the French version, (3) character N-grams of both versions concatenated, and (4) some NER information extracted using NLP techniques. As similarity function we have used BM25 as it takes into account both term frequency and document length. See Section 3.3 for more details.
To generate all these fields, 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 stopword list, query expansion with synonyms, 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 and stemming. To generate the character N-grams we consider only the letters of the documents (i.e. discarding numbers and punctuation). To perform NER we apply NLP techniques (based on Apache OpenNLP[3]) to the original (French) version of the documents. See Section 3.2 for more details.
To generate the runs we have done some experiments, i.e., we have tried different combinations of the explained techniques during the development of the project. Thus, our searcher will always use BM25, 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.
Furthermore, we will present the main methodology and approaches following the structure in the repository[4].

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. Additionally, because of the large number of documents we are working with, we had to create a **steam** parser, allocating into the main memory one document at a time. Thus, our parser is based on the Java library `Gson`.
The whole parser is made up of:

- `DocumentParser`: an abstract class representing a stream parser for documents in a collection.
- `JsonDocument`: a Java POJO used in the deserialization of JSON documents.
- `ParsedDocument`: Java object that represents a document already parsed. Note that this object contains an identifier and a body.
- `LongEvalParser`: real parser implementing the class `DocumentParser`. Here is where the stream logic is implemented. Objects of this class can be used as iterators returning 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: apply a list of `TokenFilter` to a `TokenStream`.
The final version of the project creates an index with four fields for each document: (1) English version, (2) French version, (3) character N-grams of both versions concatenated, and (4) NER extracted information. 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, 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 `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 found punctuation marks attached to words (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 parts 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].
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`.

3.2.2. French body field

The processing of French documents (in `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 `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 consider the French accent letters. We decided to take this decision after some tests on character N-grams with numbers, which didn't make sense: a lot of meaningless numbers were generated.

4. Delete empty tokens (`EmptyTokenFilter`).

5. Generate character N-grams using `NGramTokenFilter` from Lucene.

We have not fixed the value of N, we have left it open in order to be able to generate different experiments. See Section 4 for more details.

3.2.4. NER extracted information

The NER information has been extracted using the Apache OpenNLP[3] 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

We used the standard Lucene Indexer with the BM25[8] similarity.
We decide to use NGram analyzer with a multilingual indexer, in order to index two versions of the same document, creating documents with unique IDs and bodies from each language.
Ngrams are represented as characters from both English and French version of the documents.
N parameter is set at the analyzer level and class is a field, not stored, in order to minimize space occupation.

3.4. Search

There's a need to be a searcher in order to look for topics/titles desired. The searcher uses the analyzers developed for the different queries: English, French, N-gram and NER analyzers. It also requires the similarity function developed for checking topics and queries that might be similar. The first thing required for the searcher is a directory in which the index can be looked into, in order to make this happen we include an parameter named indexPath. When a query is requested a parser determines which document field and analyzer shall be used. There could be queries requesting multiple documents and involving different combinations of analyzers, so every case must be available for the search. We provide the searcher with an input txt file where the topics to be searched are written, also the number of topics expected to be searched needs to be specified. When the run is completed the searcher creates an output txt file providing all the information regarding the run made, such as runID, documentsID and elapsed time.

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` is actually doing 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 a 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
- [4, Repository]
- During the development and the experimentation, personal computers were used
- Java JDK version 17, Apache version 2, Lucene version 9.5, Maven

Our team has created several indexes from the collection which was provided by LongEval. Indexes were created to be used in experiments with the runs, and they were created in order to enable fast and effective search and retrieval. All of the created indexes are multilingual. Which gives the ability to use, understand, communicate in more than one language since we have two languages in our project to work with. In other words they are proficient in multiple languages. Additionally indexes were created with different implementations of character 3-grams, character 4-grams and character 5-grams in order to capture different aspects of the language. Indexes which implementation of character 3-grams are based on groups of three consecutive characters in each document allowing for some contextual understanding of word variations and misspellings. 4-gram indexes capture four consecutive characters and additional information about word and phrase structure. They were used to provide even greater context around words and to improve query performance and precision. 5-gram indexes, which use groups of five consecutive characters were used to capture even more context around words to aim further improvements in query performance. Additional functionality of some of the indexes is having query expansion with synonyms, which provides the improvement of the search results by including related words or phrases that are equivalent in meaning to the

Table 1
MAP and NDCE scores for all runs

Index	Run	MAP Score	NCDG Score
01	en_en	0.0700	0.1614
02	en_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

user's original query. One index(2023_05_05_multilingual_4gram_synonym_ner) uses Named Entity Recognition which provides not only the search for keywords but also identifying and extracting specific named entities. By implementing this to the index, the accuracy and relevance is improved. The subsequent indexes are:

- 2023_04_24_multilingual_3gram
- 2023_04_29_multilingual_3gram_synonym
- 2023_05_01_multilingual_4gram_synonym
- 2023_05_01_multilingual_5gram_synonym
- 2023_05_05_multilingual_4gram_synonym_ner

The indexes also can be found in the following Google Drive folder.

5. Results and Discussion

The dataset comprises MAP and NDCG scores for twelve runs of an Information Retrieval (IR) system with different language models and n-gram sizes for indexing. MAP measures how effectively the IR system returns relevant results for a given query, while NDCG indicates the ranking quality of the results.
The analysis shows that the highest MAP score (0.1748) is achieved by fr_fr_5gram, followed fr_fr_4gram (0.1737) and fr_fr_3gram (0.698), while the lowest MAP score (0.0360) is obtained by en_en_4gram_ner. Similarly, the highest NDCG score (0.3208) belongs to fr_fr_4gram_ner, followed by fr_fr_5gram (0.3285) and 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

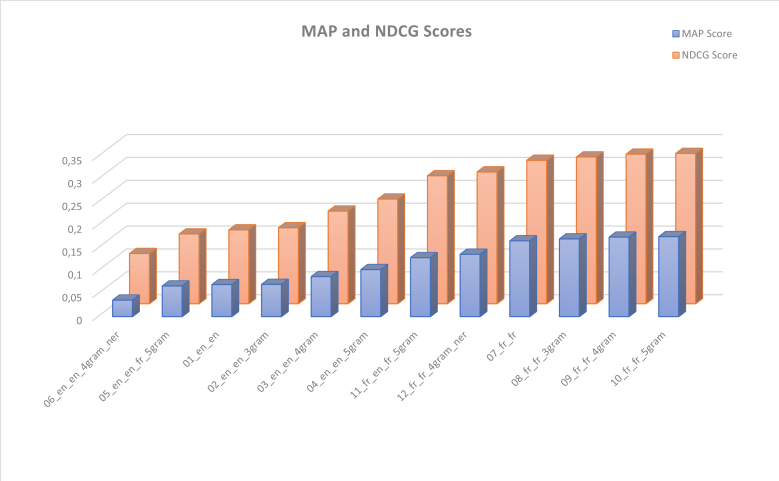

Figure 1: All scores sorted by MAP score

due to the training data's origin in French 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 named entity recognition (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.
Here is a chart ranking of the five best scores, there are indexes that have been presented at CLEF: It's interesting to notice that a cross-language approach (fr_en_fr_5gram) is just out of

Figure 2: Best MAP and NDCG scores

the five bests performing indexes, even though it has English indexes. We suppose this is due to the fact that most of the score comes from the French queries, which are the most relevant ones. On the other hand, the worst-performing index is the one with named entity recognition

(en_en_4gram_ner): it combines translated queries and named entity recognition, 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: queries weren't made by single word match, instead the more context was given, the better the results were.

6. Conclusions and Future Work

Provide a summary of what are the main achievements and findings.

Discuss future work, e.g. what you may try next and/or how your approach could be further developed.

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] Apache, Apache OpenNLP, https://opennlp.apache.org/, 2023.
[4] Atabek, Canale, Chen, Moncada-Ramirez, Santini and Zago, JIHUMING, https://bitbucket.org/upd-dei-stud-prj/seupd2223-jihuming/src/master/, 2023.
[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] S. E. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, M. Gatford, Okapi at trec-3, in: Text Retrieval Conference, 1994.