

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 an original and efficient information retrieval system for Longitudinal Evaluation of Model Performance (LongEval) by CLEF2023[1]. 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 master's degree in Computer Engineering and Data Science at University of Padua, Italy. Task chosen by the group is CLEF LongEval: Longitudinal Evaluation of Model Performance.

Longeval Websearch collection[2] relies on a large set of data provided by a commercial Qwant (a commercial search engine). The idea is to reflect changes of the Web across time, providing evolving document and query sets.

Our approach uses on N-grams, trying to avoid the problem of the sparsity of the data. Our focus was not to produce a high scoring specialized system, but to try developing general ideas in order to approach sentences selection and query expansion.


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.

"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)

2. Methodology

In this section, we address the technical aspects of how our system was developed. We have use Java and the library Lucene. 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 using the structure in the repository[3].

2.1. Parsing

Participating in the LongEval CLEF Lab 2023 we are provided with a collection of French (original) and English (translated from French) documents. To generate an index based on them, 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.

2.2. 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. We will explain them independently.

2.2.1. English body field

The processing applied to the English version of the documents 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 (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: "Power-Shot" 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 stopword list.
7. Apply query expansion with synonyms using `SynonymTokenFilter` from Lucene.
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`.

2.2.2. French body field

The processing of French documents 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. Apply a minimal stemming process (in French) using `FrenchMinimalStemFilter` from Lucene.
8. Delete empty tokens (`EmptyTokenFilter`).

2.2.3. Character N-grams

Character N-grams are created in an analyzer that 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 3 for more details.

2.2.4. NER extracted information

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

2.3. 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 version of the same document, creating documents with an unique ID 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.

2.4. Search

2.5. Topic

3. Experimental Setup

Describe the experimental setup, i.e.

- used collections
- evaluation measures
- url to git repository and its organization
- hardware used for experiments
- ...

4. Results and Discussion

Provide a summary of the performance on the previous year dataset.

Discuss the results and any relevant issues.

5. 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] Canale, Chen, Ramirez, Santini and Zago, jihuming, <https://bitbucket.org/upd-dei-stud-prj/seupd2223-jihuming/src/master/>, 2023.
- [4] I. Brigadir, Default english stop words from different sources, <https://github.com/igorbrigadir/stopwords>, 2023.
- [5] Princeton University, About WordNet, <https://wordnet.princeton.edu/download/current-version>, 2005.
- [6] D. Harman, How effective is suffixing?, Journal of the American Society for Information Science 42 (1991) 7–15.
- [7] J. Savoy, A stemming procedure and stopword list for general french corpora, J. Am. Soc. Inf. Sci. 50 (1999) 944.
- [8] S. E. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, M. Gatford, Okapi at trec-3, in: Text Retrieval Conference, 1994.