

# SEUPD@CLEF: Team JIHUMING on <Short Description>

Notebook for the LongEval Lab at CLEF 2023

Isil Atabek<sup>1</sup>, Huimin Chen<sup>1</sup>, Jesús Moncada-Ramírez<sup>1</sup>, Nicolò Santini<sup>1</sup>,  
Giovanni Zago<sup>1</sup> and Nicola Ferro<sup>1</sup>

<sup>1</sup>University of Padua, Italy

## 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 evaluating the temporal persistence of information retrieval (IR) systems and text classifiers.

The LongEval collection 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

---

*CLEF 2023: Conference and Labs of the Evaluation Forum, September 18–21, 2023, Thessaloniki, Greece*

✉ isil.atabek@studenti.unipd.it (I. Atabek); huimin.chen@studenti.unipd.it (H. Chen);

jesus.moncadaramirez@studenti.unipd.it (J. Moncada-Ramírez); nicolo.santini.1@studenti.unipd.it (N. Santini);


giovanni.zago.3@studenti.unipd.it (G. Zago); ferro@dei.unipd.it (N. Ferro)

🌐 <http://www.dei.unipd.it/~ferro/> (N. Ferro)

🆔 0000-0001-9219-6239 (N. Ferro)

© 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)

evolving document and query sets. The train collection [2] consists of 1,570,734 documents, 672 queries, 98 held-out queries, and 9656 evaluation assignments. The documents were chosen based on queries using the Qwant click model, in addition to random selection from the Qwant index. The 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 test collection [4] for the short-term persistence sub-task was gathered during July 2022, comprising 1,593,376 documents and 882 queries. The test collection for the long-term persistence sub-task was collected in September 2022, containing 1,081,334 documents and 923 queries.

The paper is organized as follows: Section 2 introduces related works; Section 3 briefly describes our approach; Section 4 describes our code in detail; Section 5 explains our experimental setup; Section 6 explains how we selected the systems submitted to the competition among our experiments; Section 7 discusses our main findings; finally, Section 8 draws some conclusions and outlooks for future work.

## 2. Related Work

There are many search engines, using different techniques to enhance retrieval effectiveness, from which we have taken inspiration from.

The BM25 [5] similarity function is widely used information retrieval as it considers term frequencies and document length. This function has demonstrated effectiveness in balancing precision and recall in search results, even if it doesn't consider meta-data document information as other approaches [6] do. Whitespace tokenization has not been the primary focus of research in any study. Anyway, it seems to provide a useful baseline for tokenization; Gow-Smith et al. [7] suggests that allowing tokens to include spaces causes problems, especially in architectures including transformers. Token lowercasing is also a recurring method in information retrieval [8], mainly because it reduces the vocabulary size; it is always applied after more advanced token filters as *WordDelimiterGraphFilter*.

The Terrier [9] stopwords list has been used in plenty of search engines because of the good results it offers working with web documents as blogs [10] or even recommender systems. Another basic information retrieval technique used in our search engine has been stemming. We have relied on the work of Jivani et al. [11] to get an overview of the most adequate stemming techniques for our documents. For the English documents we have chosen a minimal stemmer developed by K. Harman [12]. For the French documents we have also used a minimal stemmer developed by J. Savoy [13].

We have used query expansion [14] in order to broaden the search scope by including additional synonyms related to the original query. These synonyms come from WordNet [15], a popular lexical database that provides semantic relationships between words. We have also

included character N-grams of the English and French versions of the documents. Our experiments on character N-grams have been focus on comparing how the value of  $N$  can affect to the retrieval effectiveness. We took our motivation from the works of Wilson et al. [16], and Goodman [17], who also compared how different N-grams (and other models) can affect to the performance. We tried to refine our results including Named Entity Recognition [18]. This technique has been previously utilized in information retrieval, for example in the food [19] and archaeology [20] domain.

The work from Cai et al. [21] remarks the importance of understanding query temporal dynamics for search result ranking. By considering the temporal patterns of queries and incorporating query temporal dynamics into the ranking process, search engines can deliver more relevant and timely results. An example of this is giving more weight to recent queries or adjusting the ranking based on a certain popularity during specific time periods. In the context of our task, with changing datasets, learning and estimating query temporal dynamics it can be highly relevant.

The work from Hofmann et al. [22] presents valuable insights into evaluation methodologies for temporal aspects in web search systems. Specifically, the paper explores metrics for evaluating retrieval effectiveness over time, such as precision, recall, F-1 score, and mean average precision (MAP). The paper provides insights into various setups, including time-sliced evaluation, incremental evaluation, and evaluation with simulated temporal queries. These setups can serve as a basis for designing your own experimental setups that align with your specific task requirements. With this motivation we can incorporate evaluation methodologies, metrics, and experimental setups specifically tailored for temporal information retrieval

### 3. 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.

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 [5] as it takes into account both term frequency and document length.

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 [9] stopword list, query

expansion with synonyms based on the WordNet synonym map [23], 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 [24] 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 [25] to the original (French) version of the documents. Specifically, we used NER applied to locations, person names and organizations.

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 [5], but the rest of characteristics depend on the run it is generating. See Section 5 for more details.

## 4. System Architecture

In this section, we address the technical aspects of how our system was developed following the structure (in packages) of the repository [26].

### 4.1. 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.

### 4.2. 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.

#### 4.2.1. 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 [9] stopword list.
7. Apply query expansion with synonyms using `SynonymTokenFilter` from Lucene, which is based on the WordNet synonym map [23]. 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`.

#### 4.2.2. 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:

start=6 Apply a French stopwords list [24].

stbrt=6 Apply a minimal stemming process (in French) using `FrenchMinimalStemFilter` from Lucene.

stcrt=6 Delete empty tokens (`EmptyTokenFilter`).

#### 4.2.3. 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 5 for more details.

#### 4.2.4. NER extracted information

The NER information has been extracted using the Apache OpenNLP [25] 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.

### 4.3. 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.

#### 4.4. 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.



## 4.5. 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.

## 5. Experimental Setup

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

- MAP (Mean Average Precision) and NDCG (Normalized Discounted Cumulative Gain) scores computed for our systems on the train data, in order to select the five systems to be submitted to CLEF.
- MAP (Mean Average Precision) and NDCG (Normalized Discounted Cumulative Gain) scores computed for our systems on the test data, to get a reliable estimation of the final performance.
- Statistical analysis with Two-Way ANOVA using AP (Average Precision) of our five submitted system over all the topics.
- [26, 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 each of the provided collections (train, short-term test, and long-term test). Put simply, certain indexes mentioned in this report incorporate only a few of the characteristics discussed, while others encompass all the characteristics outlined in 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 N-grams generating different versions of indexes with 3-grams, 4-grams and 5-grams. Our motivation was to compare how the size of different character N-grams affect to the effectiveness of our system. 3-grams are able to collecting more specific information about our documents, while 4-grams and 5-grams are more open to the context. An additional functionality of some indexes is query expansion, but as commented, this is only applied to the English body. One index uses Named Entity Recognition which provides not only the search for keywords but also identifying



and extracting specific named entities.

The subsequent indexes are:

- `multilingual_3gram`: both languages of documents, using character 3-grams.
- `multilingual_3gram_synonym`: both languages, character 3-grams, (English) query expansion with synonyms.
- `multilingual_4gram_synonym`: both languages, character 4-grams, (English) query expansion with synonyms.
- `multilingual_5gram_synonym`: both languages, character 5-grams, (English) query expansion with synonyms.
- `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. 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.
- `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, except 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.

For the runs on the **training data**, we computed MAP and NDCG scores with the purpose of selecting five systems to submit to the competition. See Section 6. The results will be commented in the Section 7.

## 6. Runs Submitted to Clef Selection

**Table 1**

MAP and NCDG scores for all runs in training data

Run ID	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

The analysis shows that the highest MAP score (0.1748) is achieved by 10\_fr\_fr\_5gram, followed by 09\_fr\_fr\_4gram (0.1737), 08\_fr\_fr\_3gram (0.1698), 07\_fr\_fr (0.1656), and 12\_fr\_fr\_4gram\_ner (0.1362). The lowest MAP score (0.0360) is obtained by en\_en\_4gram\_ner.

Similarly, the highest NDCG score (0.3285) belongs to 10\_fr\_fr\_5gram, followed by 09\_fr\_fr\_4gram (0.3269), 08\_fr\_fr\_3gram (0.3208), 07\_fr\_fr (0.3135), and 12\_fr\_fr\_4gram\_ner (0.2881). The lowest NDCG score (0.1098) corresponds to en\_en\_4gram\_ner.

Because of this, the five system submitted to CLEF have been (in order of importance): 10\_fr\_fr\_5gram, 09\_fr\_fr\_4gram, 08\_fr\_fr\_3gram, 07\_fr\_fr, 12\_fr\_fr\_4gram\_ner. Following the competition workflow, we created new indexes based on the test data and re-executed this top five runs.

## 7. Results and Discussion

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

It's interesting to notice that the cross-language approaches (`en_en_fr_5gram` and `fr_en_fr_5gram`) are out of the five best 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.

### 7.1. Short Term Test Data

### 7.2. Long Term Test Data

### 7.3. Anova 2 Analysis

## 8. 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_fr`, 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 7. 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, Ľ. Kaiser, J. Uszkoreit, O. Bojar, Z. Žabokrtský, Transforming machine translation: a deep learning system reaches news translation quality 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] CLEF2023, LongEval Test Collection, <https://lindat.mff.cuni.cz/repository/xmlui/handle/11234/1-5139>, 2023.
- [5] S. E. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, M. Gatford, Okapi at trec-3, in: *Text Retrieval Conference*, 1994.
- [6] S. Robertson, The Probabilistic Relevance Framework: BM25 and Beyond, *Foundations and Trends® in Information Retrieval* 3 (2009) 333–389. URL: [http://scholar.google.de/scholar.bib?q=info:U4l9kCVIssAJ:scholar.google.com/&output=citation&hl=de&as\\_sdt=2000&as\\_vis=1&ct=citation&cd=1](http://scholar.google.de/scholar.bib?q=info:U4l9kCVIssAJ:scholar.google.com/&output=citation&hl=de&as_sdt=2000&as_vis=1&ct=citation&cd=1).
- [7] E. Gow-Smith, H. T. Madabushi, C. Scarton, A. Villavicencio, Improving tokenisation by alternative treatment of spaces, 2022. *arXiv:2204.04058*.
- [8] C. D. Manning, P. Raghavan, H. Schütze, *Introduction to Information Retrieval*, Cambridge University Press, Cambridge, UK, 2008. URL: <http://nlp.stanford.edu/IR-book/information-retrieval-book.html>.
- [9] 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.
- [10] I. Soboroff, C. Macdonald, I. Ounis, Overview of the trec-2009 blog track, 2010. URL: [https://tsapps.nist.gov/publication/get\\_pdf.cfm?pub\\_id=905208](https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=905208).
- [11] A. G. Jivani, et al., A comparative study of stemming algorithms, *Int. J. Comp. Tech. Appl* 2 (2011) 1930–1938.
- [12] D. K. Harman, How effective is suffixing?, *J. Am. Soc. Inf. Sci.* 42 (1991) 7–15.

- [13] J. Savoy, A stemming procedure and stopword list for general french corpora, *J. Am. Soc. Inf. Sci.* 50 (1999) 944–952.
- [14] E. N. Efthimiadis, Query expansion., *Annual review of information science and technology (ARIST)* 31 (1996) 121–87.
- [15] C. Fellbaum (Ed.), *WordNet: An Electronic Lexical Database, Language, Speech, and Communication*, MIT Press, Cambridge, MA, 1998.
- [16] T. Wilson, S. Raaijmakers, Comparing word, character, and phoneme n-grams for subjective utterance recognition, 2008, pp. 1614–1617. doi:10.21437/Interspeech.2008-270.
- [17] J. Goodman, A bit of progress in language modeling, 2001. arXiv:cs/0108005.
- [18] B. Mohit, Named entity recognition, *Natural language processing of semitic languages* (2014) 221–245.
- [19] G. Popovski, B. K. Seljak, T. Eftimov, A survey of named-entity recognition methods for food information extraction, *IEEE Access* 8 (2020) 31586–31594.
- [20] A. Brandsen, S. Verberne, K. Lambers, M. Wansleeben, Can bert dig it? named entity recognition for information retrieval in the archaeology domain, *Journal on Computing and Cultural Heritage (JOCCH)* 15 (2022) 1–18.
- [21] F. Cai, H. Wang, C. Zhai, Learning to estimate query temporal dynamics for web search, in: *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*, 2014, pp. 373–382.
- [22] K. Hofmann, S. Whiteson, M. de Rijke, Evaluating web search systems considering time, in: *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*, 2014, pp. 321–330.
- [23] Princeton University, About WordNet, <https://wordnet.princeton.edu/download/current-version>, 2005.
- [24] G. Diaz, A. Suriyawongkul, Stopword list in French, <https://github.com/stopwords-iso/stopwords-fr/tree/master>, 2023.
- [25] Apache, Apache OpenNLP, <https://opennlp.apache.org/>, 2023.
- [26] Atabek, Canale, Chen, Moncada-Ramirez, Santini and Zago, JIHUMING, <https://bitbucket.org/upd-dei-stud-prj/seupd2223-jihuming/src/master/>, 2023.