SEUPD@CLEF: Team NEON. A Memoryless Approach To Longitudinal Evaluation

Notebook for the LongEval Lab at CLEF 2023

Simone Bortolin[1], Gioele Ceccon[1], Gil Czaczkes[1], Alessandra Pastore[1], Pietro Renna[1], Giovanni Zerbo[1] and Nicola Ferro[1]

[1]*University of Padua, Italy*

Abstract
This paper presents a possible approach for the LongEval Lab of the CLEF 2023 conference concerning Longitudinal Evaluation of Model Performance. Studies have shown that the performance of Information Retrieval systems decreases as the time gap between the test data and the training data increases. The LongEval Lab focuses on the development of a robust temporal IR system that improves such performance.

Keywords
Information Retrieval, Search Engine, LongEval CLEF 2023

1. Introduction

Information retrieval is nowadays an indispensable tool for almost everyone, helping people to obtain all kind of information in a quick and effective way.

This report describes an Information Retrieval System developed for the Search Engines course at the University of Padua, and it is related to Task 1 of the Longitudinal Evaluation of Model Performance (LongEval) lab for CLEF 2023 [1] conference. As stated on [1], the primary goal for developing this system is to improve the model's performance as the test data are collected further away in time from the train data. The train collection is a large data set based on the Qwant [2] click model and random documents from the Qwant index.

The task is divided into two sub-tasks regarding short-term and long-term persistence to better test the robustness of our system.

The paper is organized as follows:

- Section 3 describes our approach;
- Section 4 explains our experimental setup;
- Section 5 discusses our main findings;
- Section 6 draws some conclusions and outlooks for future work.

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

simone.bortolin.1@studenti.unipd.it (S. Bortolin); gioele.ceccon@studenti.unipd.it (G. Ceccon); gil.czaczkes@studenti.unipd.it (G. Czaczkes); alessandra.pastore.2@studenti.unipd.it (A. Pastore); pietro.renna@studenti.unipd.it (P. Renna); giovanni.zerbo@studenti.unipd.it (G. Zerbo); ferro@dei.unipd.it (N. Ferro)
http://www.dei.unipd.it/~ferro/ (N. Ferro)
0000-0001-9219-6239 (N. Ferro)

2. Related Work

The main motivation for the longitudinal evaluation of IR systems is the need to assess the performance of the systems in real-world scenarios, in which the data is constantly changing. In fact, the data evolves due to the social context and language representation. For this reason, it is of great interest to be able to predict the performance of search engines over a long period of time [3, 4, 5].

Creating a collection resembling such scenario is rather than simple. In the cases of Twitter [6] and Robust [7] collections, the information about the time in which each document was created is available. Then, this information can be used to simulate the changes in the data and to evaluate the system under these changes. Despite this, these collections do not well represent an evolving environment. An interesting case is the TREC-COVID dataset [8]. It captures the evolution of data, however it presents some disadvantages. In particular, it is very specific, since it focuses only on the covid epidemic. Furthermore, the collection is quite small.

Another relevant aspect in evaluating the performance of such systems is the treatment of languages when dealing with a cross-language collection. English and French were part of the CLEF 1999–2002 collections [9, 10, 11, 12].

Later on, English and French were part of the multilingual collections of CLEF eHealth Evaluation Lab [13, 14], that was focused on multilingual information extraction in the medical subject.

For what concerns the word representation, a noteworthy approach is the one presented in [15] (FastText), where each word is represented as a set of character n-grams. For each n-gram is then created its vector representation. This method allows the computation of word representations for words that did not appear in the train data of the model.

Lastly, an exploitable tool is WordNet [16]. Through semantic relations it links words to synonyms. The challenge it presents is polysemy. In fact, choosing between alternative senses of the same word implies choosing between different contexts in which the word can be used.

3. Methodology

The developed IR system is based on Apache Lucene.

To allow for a more in-depth description of the system, this section is organized into separated sub-sections.

Parser: handles the parsing of documents in the collection, provided in the TREC format, as defined in [17].

Analyzer: is responsible for analyzing pieces of text and applying the appropriate filters to standardize the text.

Indexer: takes care of indexing the documents in the index once received from the Parser.

Searcher: handles the search for documents in the index from queries.

3.1. System Architecture

The IR system is based on the latest current version of Apache Lucene (9.5.0). This library represents the state-of-the-art and it is also used by major IR systems, such as those based on Elasticsearch. For this reason, the architecture is based on the four core elements of this library and the main structure is presented in fig. 1.

To test different configurations for the produced model, the system is developed to take in input different parameters and behave differently for each of them, so that it is easier to produce different runs to compare.

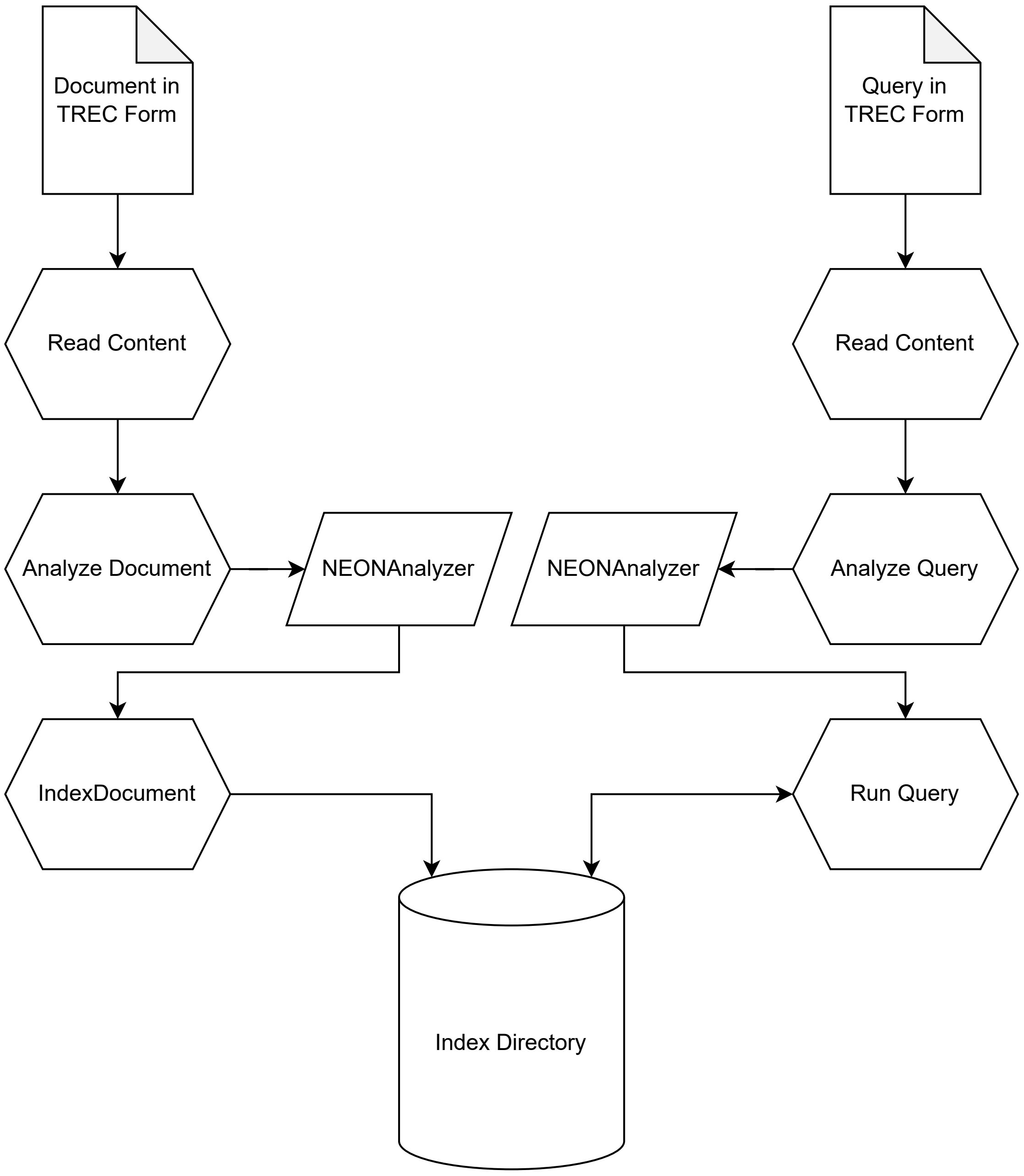


Figure 1: Representation of the system architecture

3.2. Parser

`NEONParser` is the parser developed to manage the parsing of the documents in the LongEval collection. Based on the code analyzed in the Search Engine course, it extends the abstract class `DocumentParser`, an Iterable class used to run through the whole corpus, and reads one document at a time, provided in the TREC format. While reading, it removes the HTML tags and collects the information needed to create the corresponding `ParsedDocument` object, which is made of two fields:

ID: contains the ID of the document.

BODY: contains the textual content of the document.

These objects are then used by the system to create the index.

3.3. Analyzer

The developed Analyzer is called `NEONAnalyzer`, which extends the abstract `Analyzer` provided by the Apache Lucene library. The Tokenizer source is obtained using `StandardTokenizer` from Apache Lucene, and then the returned `TokenStream` is modified by applying both standard and custom filters.

LowerCase Filter: standard lowercase filter provided by the Apache Lucene library. It converts all the tokens into lowercase.

English Possessive Filter: standard filter provided by the Apache Lucene library. This version of the filter is specifically made for the English language and removes the possessives (trailing `'s`).

Accent Filter: custom filter. It removes both accents and diacritics.

Stop Filter: standard filter provided by the Apache Lucene library. It is used to remove all stop words in the Snowball [18] expanded stoplist.

KStem Filter: standard filter provided by the Apache Lucene library. It applies the Krovetz Stemmer that combines both algorithmic and dictionary-based stemming techniques.

PorterStem Filter: standard filter provided by the Apache Lucene library. Transforms the token stream as per the Porter stemming algorithm. It's used when dealing with synonyms, as discussed in Section 3.3.2).

Before choosing to stick with the Krovetz one, other stemmers such as the Lovins Stemmer and the Porter Stemmer were tried.

3.3.1. French and English processing: additional filters

These are additional filters that are used together with the ones of Section 3.3 to form a particular implementation of the system. Since the English dataset is an automatic translation of the French documents, some terms were not fully translated and maintained their original French form (e.g. chalor, challoire). Accordingly, to improve the indexing of these words, a French Filter is incorporated in certain runs.

- **French Minimal Stem Filter:** standard filter provided by the Apache Lucene library. It's designed to perform minimal stemming on French text.
- **Lovins Stem Filter:** Custom filter that implements the Lovins stemmer.

3.3.2. Query expansion: WordNet and WuPalmer similarity

There is a portion of code in `NEONAnalyzer` that is specialized in dealing with query strings. After applying the filters discussed in Section 3.3, to perform query expansion the analyzer applies a POS tagger and then a custom filter that deals with the synonyms.

- **OpenNLP POS Filter:** standard filter provided by the Apache Lucene library. It works by tagging all terms as a normal POS filter, using Machine Learning for the processing of Natural Language.
- **OpenNLP POS Tagging To WordNet POS Tagging Converter:** custom filter that relies on the POS list used in the Penn Treebank Project [19] to detect different types of entities.
- **Synonym Filter and Synonym Add Token Filter:** custom filters that rely on the WordNet database [16] and the Wu & Palmer similarity function [20] to find synonyms up to a given maximum number. In particular, the constant value of 2 was chosen to prevent excessive noise from being added to the queries, which could cause the system to retrieve irrelevant documents.
  These two filters differ in the way they emit the synonyms as tokens:
  - `SynonymFilter` emits for every token it parses a single token, composed by the original word followed by its synonyms (if any) separated by a slash.
  - `SynonymAddTokenFilter` emits the token in input and one separated token for its synonyms (if any), tagged with the string "synonym" to recognize them during the search phase.

  In the end, it was decided to use `SynonymAddTokenFilter` since it allowed an easier integration of the synonyms management into the system.

Created at Princeton University, WordNet is a lexical database for the English language. It is a semantic network consisting of a large collection of words (called "synsets") that are organized based on their meanings and relationships with other words. Each synset contains a group of words that are synonyms of each other and share a common meaning, as well as a definition and examples of how the words can be used in context. WordNet then works by organizing words into hierarchies of semantic relationships, such as hypernyms (words that are more general than a given word) and hyponyms (words that are more specific than a given word). In particular, WordNet consists of four sub-nets, one each for nouns, verbs, adjectives, and adverbs, connecting words from the same Part Of Speech (POS).

In order to use WordNet within the filters, the 3.1 Prolog version of the database has been downloaded and fed to a `SynonymMap` object, a fast hash map used to retrieve synonyms from any specified lowercase word which is provided by the `wordnet` package of Lucene.

To obtain a more precise use of synonyms, it has been decided to retain only the top two words with a score greater than a given threshold value of 0.9 (if specified as input), along with the original token, for the Wu & Palmer similarity function that was implemented.

WuPalmer similarity is a method for measuring the similarity between two words based on their synsets in a semantic network (such as WordNet). The idea behind it is that two words with similar meanings should have synsets that are close to each other in the network. Thus, the method is based on the notion of depth in a semantic network, which is the number of edges that must be traversed to get from one synset to another. The WuPalmer similarity between two synsets is then defined as the depth of their lowest common subsumer (LCS) divided by the sum of their depths in the network. This means that the LCS is the most specific synset that is an ancestor of both senses in the network. The result is a number between 0 and 1, where 1 indicates perfect similarity and 0 indicates no similarity.

To implement the WuPalmer Similarity, the MIT Java Wordnet Interface (JWI) [21] was adopted. JWI is a library used to access and manipulate the WordNet database: it provides a set of classes and methods for accessing WordNet data, including synsets, words, and relationships between them. It also allows to perform various operations on the data, such as searching for words or synsets and retrieving hypernyms and hyponyms. Since the JWI implementation does not accept a Prolog database like the one discussed above, it was downloaded the specific database file, as requested in the JWI documentation.

Furthermore, because WordNet is divided into four subnets - one for each of the previously specified POS tags - it was necessary to:

1. Use `OpenNLPPOSFilter` to attach the POS tag to each token;
2. Use `OpenNLPPOSTaggingToWordNetPOSTaggingConverter` filter to convert the tags into the ones accepted by the JWI;
3. Feed the tokens to `SynonymFilter`/`SynonymAddTokenFilter`.

3.4. Indexer

The developed Indexer (`DirectoryIndexer`) deals with the indexing of the parsed documents by storing their most interesting features, which will be then used during the search process. In particular, for each document, the two extracted fields are the ones created by `NEONParser` (Section 3.2):

- The **ID** field;
- The **BODY** field.

After the tokenization, for each BODY field the following features are saved inside the index:

- For each term in the field, its document frequency and its position;

- The term vector (a distinct data structure from the inverted index that can be accessed during search) contains:
  - Terms frequency;
  - Terms position;
  - Terms payload (if any), which represents any additional data associated with each term, that is the *word2vec* FastText array.

3.5. FastText: Indexer and Query Expansion

FastText is an open-source library designed by Facebook's AI Research (FAIR) used for word embeddings and text classifications [15], needed in order to improve the search for similar words, along with a `FuzzyQuery`[1].

For every token in the indexer, a related vector is added to the payload. This vector is automatically generated by FastText. Then for every token in the query, a `FuzzyQuery` is generated so that the search involves the token itself and similar words at character level (e.g. tax, taxation, taxi). Then the cosine similarity is used to match query vector and indexer vector to find a correlation and update the query score.

The goal of FastText is not to generate synonyms on-the-fly but to search for words correlated at character level.

- It works both at word level and sentence level allowing sentences with the same words but in different order to generate similar vectors (unlike *word2vec* models).
- It uses a hierarchical *softMax* classifier with a binary tree;
- Like *word2vec* models, it uses negative sampling to update the parameter.

Specifically, it has been implemented with the `JFastText` library [22], using the actual last available version (0.9.2) with pre-trained word vectors [23, 24] and using a vector of 50 floats for each word.

This vector is obtained by a resize of the original vector of 300 elements using the tools provided by FastText.

3.6. Searcher

The implemented custom searcher is specialized according to the various ways it is trained: in particular, it branches differently according to the presence or absence of FastText (Section 3.5).

Initially, the method creates a `BooleanQuery` object for each topic, using the `QueryParser` to parse the query.

If FastText is present, a `FuzzyQuery`[1] is created for each word and merged in a single query to be searched together with the payload. Due to efficiency reasons, this does not search for very different words with a similar vector, instead, it should find relatively similar words.

[1]`FuzzyQuery` is a type of query in the Apache Lucene search library that allows you to search for terms that are similar to a specified term based on their Damerau-Levenshtein distance. Damerau-Levenshtein is a measure of the minimum number of single-character edits (insertions, deletions, or substitutions) required to transform one string into another.

If FastText is absent, a query structure is generated based on the combinations of possible words returned by the analyzer. Through a specific procedure that generates all the possible combinations, several queries are then created and merged: one containing all the words and the other containing some of them. The algorithm that computes the combinations of the words is based on the idea that any combination can be represented by a number (in binary form) which indicates whether the element will be present or not.

Example: Let $\text{array} = \{A, B, C\}$. The combination $c(\text{array}, 5)$ is: $5 = 101 \rightarrow \{A, C\}$. So, iterating over the combinations, which are enumerated over the numbers in range $\langle 0, \ldots, 2^{n-1} \rangle$ where n is the size of $array$, the system gets the element corresponding to the number through bit-to-bit operators, which index is represented by the binary representation of the combination.

If WordNet is enabled, synonyms are added to the queries, with a `SynonymQuery` object. Also, if re-ranking is enabled, the method computes a re-ranking score for each document using the Jaccard Similarity measure (Section 3.6.1) and then sorts the documents based on this score. Finally, the method writes the top retrieved documents to a run file.

3.6.1. Re-ranking and Jaccard similarity

The re-ranking method developed is based on the weighted Jaccard similarity measure, which yields a value in $[0, 1]$. The purpose of this implementation was to take into account the term frequency when ranking the documents.

While the Jaccard similarity is a measure of similarity between two sets of items (like the sets of terms corresponding to a query and a document):

$$\mathrm{J}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

the weighted one computes the similarity between a query and a document taking into account the weights of the terms in the documents and in the query, computed using the TF-IDF scheme (with relative frequency):

$$\mathrm{Jw}(Q, D) = \frac{\sum\limits_{t \in Q \cap D} \min\left(w_t^{(Q)}, w_t^{(D)}\right)}{\sum\limits_{t \in Q \cap D} \max\left(w_t^{(Q)}, w_t^{(D)}\right)} \tag{1}$$

where:

- Q is the set of query terms
- D is the set of document terms
- $w_t^{(Q)}$ is the weight of term t in the query
- $w_t^{(D)}$ is the weight of term t in the document

The idea behind Equation (1) is the following:

- The numerator computes the sum of the minimum weights of each term that appears both in the query and the document. This measures the extent to which the query and the document share the smallest weight for each term.

- The denominator computes the sum of the maximum weights of each term that appears both in the query and the document. This measures the overall importance of the terms that appear both in the query and the document.

To implement this method, the Term Vector was stored for each document. Consequently, the terms within a document could be accessed and their TF-IDF weights could be computed by providing a document ID.

4. Experimental Setup

Collection The used experimental collection is provided by the commercial search engine Qwant and is designed to reflect the changes of the Web across time, by providing evolving document and query sets.

In particular, the system was developed over the experimental collection given as training data for Task 1 of the LongEval CLEF 2023 Lab [1]. The training data is divided into three fundamental categories:

- Queries: provided by the users of Qwant, 672 train queries and 98 heldout queries to be used as test set. Both sets are provided in trec and tsv format.
- Corpora: it consists of 1,570,734 documents representing web pages collected in June 2022 using the Qwant click model and the given queries. These documents are provided both in JSON and trec format.
- Relevance Judgments: 9656 corresponding assessments stored in a qrels file.

It's important to point out that all the documents and the queries contained in this collection were originally collected in French and subsequently translated into English using the CUBBITT system. Even if the collection contains both versions, it was decided to work with the English one because of the lack of knowledge of the French language of the team. Given the nature of the translation, when the choice was made it was also taken into account a loss in precision of the system.

Evaluation measures The evaluation tools used during the development of the system are:

- `TRECeval`: to compute the performance measures.
- `Luke`: to inspect the indexes created by Lucene.

The main evaluation measures used to check the system during the various phases of development are:

- MAP (Mean Average Precision): it allowed to understand the behavior and effectiveness of the system.
- Interpolated precision-recall curve: it enabled the comparison of runs produced from different versions of a system.
- nDCG (Normalized Discounted Cumulative Gain): it enabled the measurement of the system's effectiveness while considering the ideal run.

Repository: https://bitbucket.org/upd-dei-stud-prj/seupd2223-neon/src/master/

Hardware used Computers, mainly Windows-based, with 6th to 12th generation processors, 16+ GB RAM and at least 1 TB SSD. No hardware acceleration or computing clusters were used. The run were all performed in human time (less than 24 hours).

5. Results and Discussion

5.1. Training results

In order to test the system and assess the impact of its components over the MAP measure, many runs were produced over the given training collection, with every one of them representing the results obtained by a specific implementation of the system. It's worth noting that the BM25 was used for every run as similarity.

In particular, the following implementations of the system were tried:

- **WordNet:** run provided by the system implementing the query expansion described in Section 3.3.2;
- **Basic:** run provided by the base system, without any further implementation described in Section 3.3;
- **FastText + WordNet:** run provided by the system implementing FastText and the query expansion described in Sections 3.3.2 and 3.5;
- **FastText:** run provided by the system implementing FastText described in Section 3.5;
- **WordNet + Stems:** run provided by the system implementing the query expansion, using the French and English processing described in Sections 3.3.1 and 3.3.2;
- **Stems:** run provided by the system using the French and English processing described in Section 3.3.1;
- **FastText + WordNet + stems:** run provided by the system using the French and English processing described in Sections 3.3.1, 3.3.2 and 3.5, implementing FastText and the query expansion;
- **FastText + stems:** run provided by the system implementing FastText, using the French and English processing described in Sections 3.3.1 and 3.5.

For each system, we performed both a run with re-ranking and one without. Re-ranking is described in Section 3.6.1

Considering Table 1, some considerations are:

- The re-ranking implemented seems to be detrimental to the performances of the system since all the runs score better results without it. Probably, this happens because the method fails to assess and compare the true nature of the documents;
- The basic system scores the best result;
- The systems (with and without re-ranking) implementing both FastText and the query expansion score the worst results.

A comparison between all the different runs is made with the use of the Interpolated Precision-Recall Curves, whose aim is to show how the runs perform in terms of Precision and Recall. It is immediate to notice from both Figures 2 and 3 that precision decreases with an increasing

Table 1
Summary statistics for the runs

	without re-ranking			with re-ranking		
	run id	MAP	time (s)	run id	MAP	time (s)
WordNet	1a	0.1405	30	1ar	0.1291	69
basic	1b	0.1459	12	1br	0.1345	58
FastText + WordNet	2a	0.1253	496	2ar	0.1136	700
FastText	2b	0.1387	383	2br	0.1344	584
WordNet + stems	3a	0.1402	27	3ar	0.1302	69
stems	3b	0.1435	11	3br	0.1336	55
FastText + WordNet + stems	4a	0.1174	566	4ar	0.1066	566
FastText + stems	4b	0.1307	487	4br	0.1192	560

recall, indicating an inverse relationship between the two measures: this means that the higher the recall, the lower the precision.

Considering the different runs in the two graphs, it is worth mentioning that the basic model is always among the top-performing ones. This suggests that the more processing is applied to the input documents, the more the performance of the system decreases. This is contrary to the expected outcome for a system that utilizes filters, identifies synonyms, and utilizes FastText, either in combination or individually, as depicted in the two graphs for both runs with and without re-ranking.

Given the above discussion, it was decided to consider the following runs for the LongEval competition:

- WordNet without re-ranking (call 1a in table 1);
- Basic without re-ranking (call 1b in table 1);
- FastText with re-ranking (call 2br in table 1);
- Stems without re-ranking (call 3b in table 1);
- FastText + stems without re-ranking (call 4b in table 1).

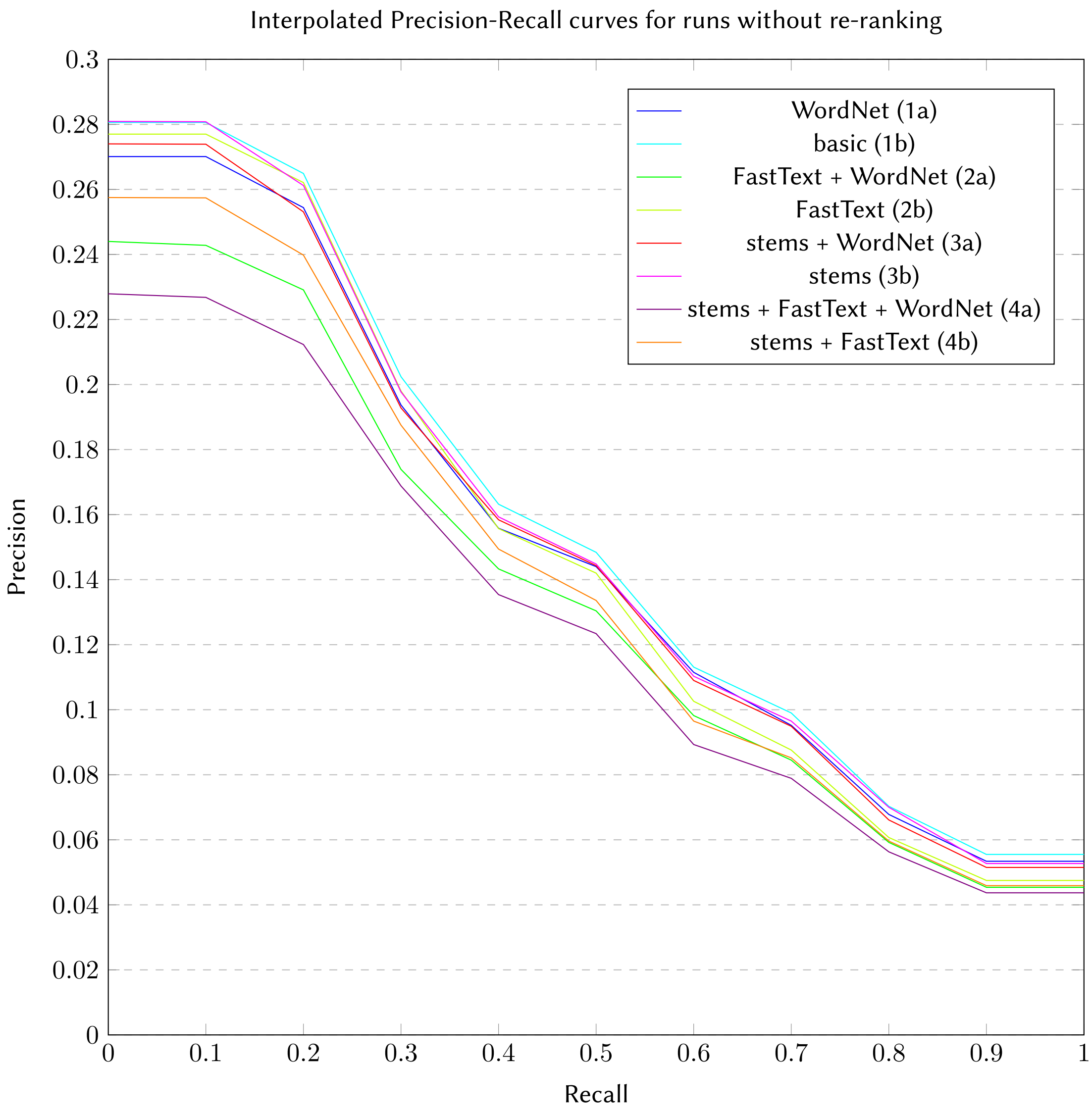


Figure 2: Interpolated Precision-Recall curves for runs without re-ranking

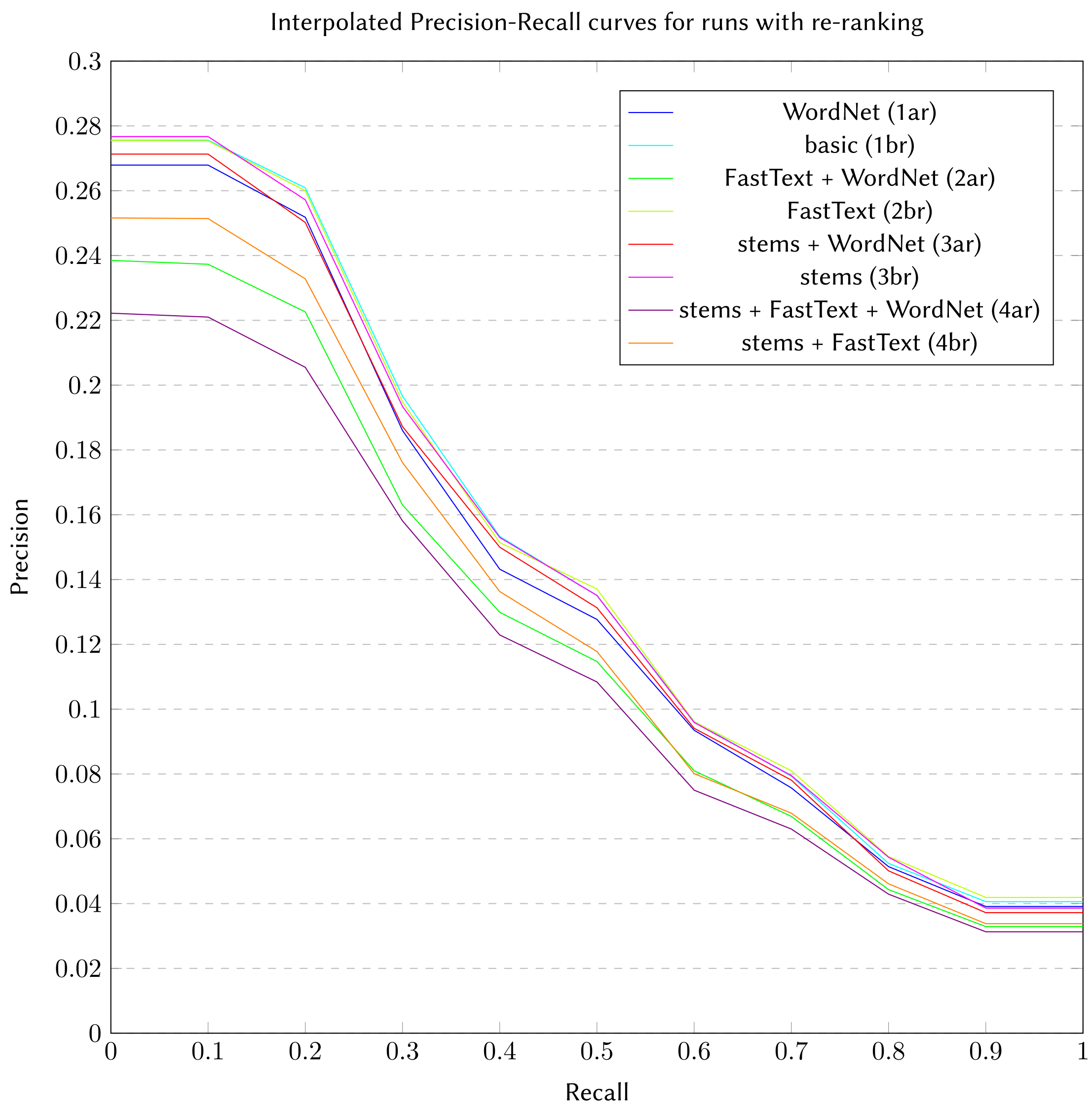


Figure 3: Interpolated Precision-Recall curves for runs with re-ranking

5.2. Test results

The runs selected in Section 5.1 have been tested over the following collections and set of queries provided by the LongEval Lab [1]:

- Collection composed of 1,570,734 documents collected in June 2022 and set, called Heldout, of 98 queries;
- Collection composed of 1,593,376 documents collected in July 2022 and set, called Short-term, of 998 queries;
- Collection composed of 1,081,334 documents collected in September 2022 and set, called Long-term, of 923 queries.

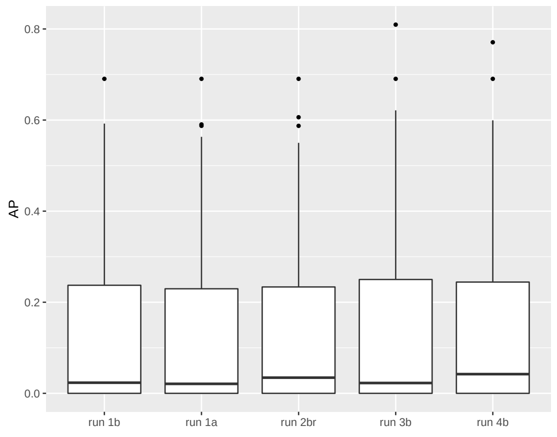
Using the provided relevance judgments, the MAP and nDCG scores are calculated for each run in relation to the three separate test sets. Moreover, for each test set, two groups of boxplots are computed for each set, respectively, by aggregating the Average Precision (AP) values and the nDCG scores of each query within the run. In each group, the runs are shown in descending order of MAP (boxplots group a) and nDCG (boxplots group b). Then, a comparison between the nDCG values of the different runs of the three test sets is performed using the following statistical techniques:

Two-way ANOVA Test: check if the factors **Topic** and **run** are statistically significant, and so, if they are significantly influencing the results. The discussion about the results is done considering a 5% level of significance, and it is performed on both MAP and nDCG measures;

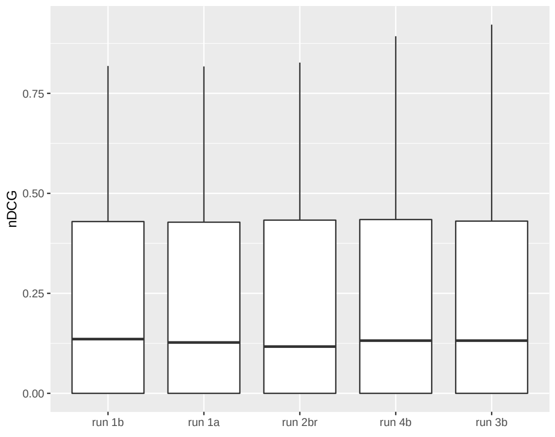
Tukey HSD test for multiple comparison adjustment: it is used to compare different runs performances, for each pair of runs, and it is particularly useful to check which specific groups means differ (or are similar) to each other. It is performed only with respect to the nDCG measure.

Table 2
MAP and nDCG values for the runs tested over the collection of June 2022

	run id	MAP	nDCG
basic	1b	0.1338	0.2269
WordNet	1a	0.1287	0.2201
FastText + re-ranking	2br	0.1279	0.2177
stems	3b	0.1226	0.2017
FastText + stems	4b	0.1213	0.2054



(a) AP



(b) nDCG

Figure 4: Boxplot for Heldout set runs

5.2.1. Heldout: June 2022

Table 2 displays the results obtained by the runs produced by the systems tested with the queries of the Heldout set.

In Figure 4 is reported the MAP value and the nDCG value of each run in a boxplot. Based on the observation, it can be concluded that there is no substantial distinction between the runs.

Some general considerations true for both Figure 4a and Figure 4b are:

- The medians are almost the same for the 5 distributions. In particular, it's evident how similar the median of runs 1a (WordNet without re-ranking) and 1b (Basic without re-ranking) are;
- The distributions are very similar to each other, although run 3b (Stems without re-ranking) and run 4b (FastText + stems without re-ranking) have more extreme outliers.

The Two-way ANOVA test performed for MAP (Table 3) and nDCG (Table 4) suggests that:

- **Topics** are significant for both MAP and nDCG results;
- **Runs** are significant only for the nDCG results.

Table 3
Results of Two-way ANOVA test over MAP of runs and queries of the Heldout set

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Topics	97	13.492	0.13910	27.368	<2e-16
runs	4	0.003	0.00087	0.171	0.953
Residuals	370	1.881	0.00508	-	-

Table 4
Results of Two-way ANOVA test over nDCG of runs and queries of the Heldout set

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Topics	97	29.13	0.3004	7.416e+28	<2e-16
runs	4	0.0	0.0000	2.502e+00	0.042
Residuals	388	<2e-16	<2e-16	-	-

Table 5
MAP and nDCG values for the runs tested over the collection of July 2022

	run id	MAP	nDCG
basic	1b	0.1390	0.2294
stems	3b	0.1384	0.2260
WordNet	1a	0.1356	0.2241
FastText + stems	4b	0.1324	0.2187
FastText + re-ranking	2br	0.1319	0.2219

Furthermore, the Tukey HSD test in Figure 5 is showing that runs are showing similarities between each others. In particular, the graph is highlighting two similar groups of intervals in the pairwise:

- Run 1a is similar to run 1b (WordNet without re-ranking), run 2b (Basic without re-ranking), run 3b (Stems without re-ranking), run 4b (FastText + stems without re-ranking). These pairwise comparisons are the most significant;
- Run 1b and run 2br (FastText with re-ranking) are similar to each other, and they are showing the same behavior when compared with run 3b, run 4b. These pairwise comparisons are less significant.

5.2.2. Short-term: July 2022

Table 5 displays the results obtained by the runs produced by the systems tested with the queries of the short-term set.

In Figure 6 is reported the MAP value and the nDCG value of each run in a boxplot. Based on the observation, it shows again that the distributions are almost the same.

In particular, it's possible to make a comparison with what has been shown in Figure 4:

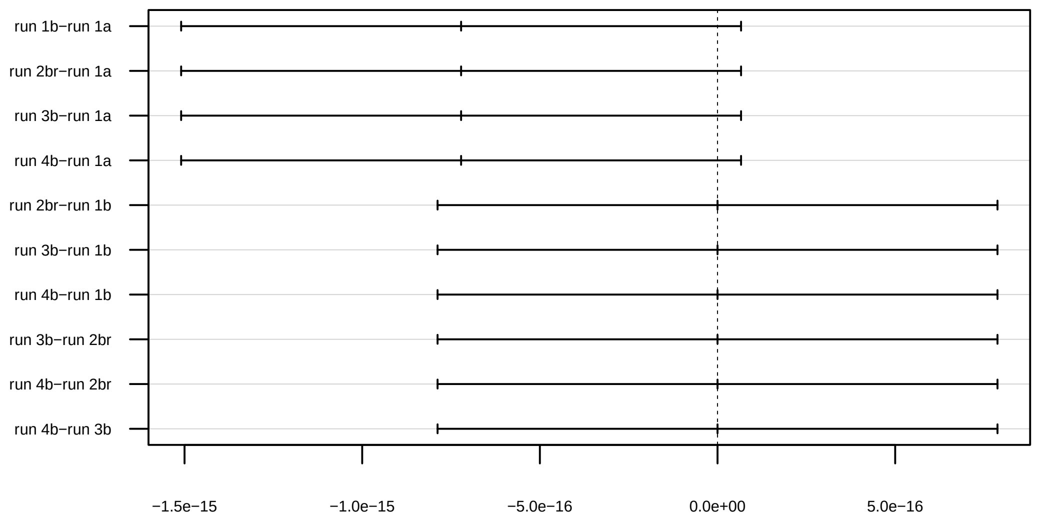
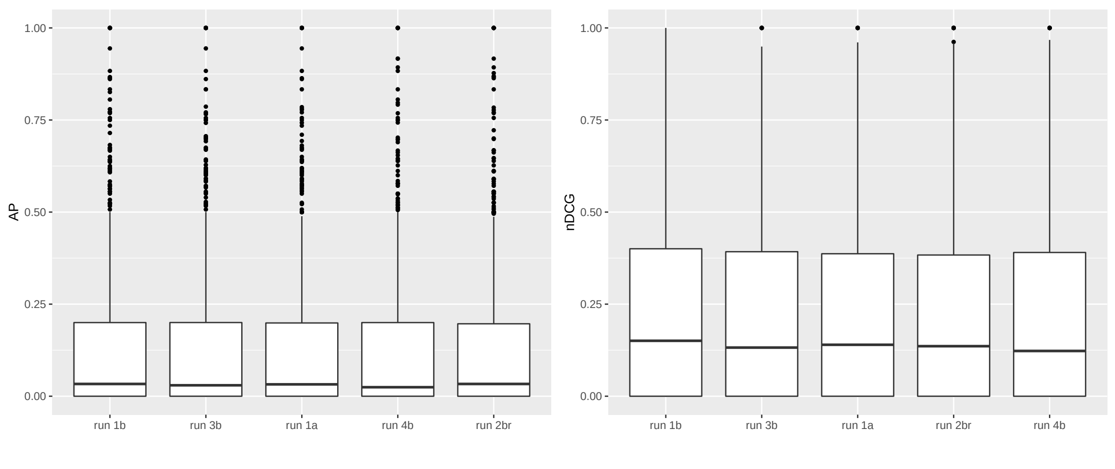


Figure 5: Tukey's results for Heldout set



(a) AP (b) nDCG

Figure 6: Boxplot for Short-Term set runs

- It's even more immediate to notice that the medians are almost equal;
- The runs show overall more outliers characterized by even more extreme values, which makes sense looking at the higher values in Table 5. It's worth noting a concentration of AP outliers in the range between 0.5 and 0.75 for all the different runs.

The Two-way ANOVA test performed for MAP (Table 6) and nDCG (Table 7) suggests that:

- **Topics** are significant for both MAP and nDCG results;

Table 6
Results of Two-way ANOVA test over MAP runs and queries of the Short-term set

	Df	**Sum Sq**	**Mean Sq**	**F value**	**Pr(>F)**
Topics	877	176.17	0.20088	72.506	<2e-16
runs	4	0.04	0.00969	3.497	0.00741
Residuals	3501	9.70	0.00277	-	-

Table 7
Results of Two-way ANOVA test over nDCG of runs and queries of the Short-term set

	Df	**Sum Sq**	**Mean Sq**	**F value**	**Pr(>F)**
Topics	875	281.5	0.3217	1.262e+27	<2e-16
runs	4	0.0	0.0000	6.000e-02	0.993
Residuals	3500	<2e-16	<2e-16	-	-

Table 8
MAP and nDCG values for the runs tested over the collection of September 2022

	run id	MAP	nDCG
basic	1b	0.1478	0.2430
WordNet	1a	0.1446	0.2393
stems	3b	0.1442	0.2387
FastText + re-ranking	2br	0.1351	0.2282
FastText + stems	4b	0.1347	0.2276

- **Runs** are significant only for both MAP and nDCG results.

Furthermore, the Tukey HSD test in Figure 7 is showing that runs are showing similarities between each others. As happened in Section 5.2.1, the graph is highlighting again two similar groups of intervals in the pairwise:

- Run 1a (WordNet without re-ranking) is similar to run 1b (Basic without re-ranking), run 2br (FastText with re-ranking), run 3b (Stems without re-ranking), run 4b (FastText + stems without re-ranking). These pairwise comparisons are the most significant;
- Run 1b and run 2br are similar to each other, and they are showing the same behavior when compared with run 3b, run 4b. These pairwise comparisons are less significant.

5.2.3. Long-term: September 2022

Table 8 displays the results obtained by the runs produced by the systems tested with the queries of the long-term set.

As can be seen in Figure 8, the runs are similar to each other. However, the distributions are slightly different for the 5 runs.

In particular, some general considerations are:

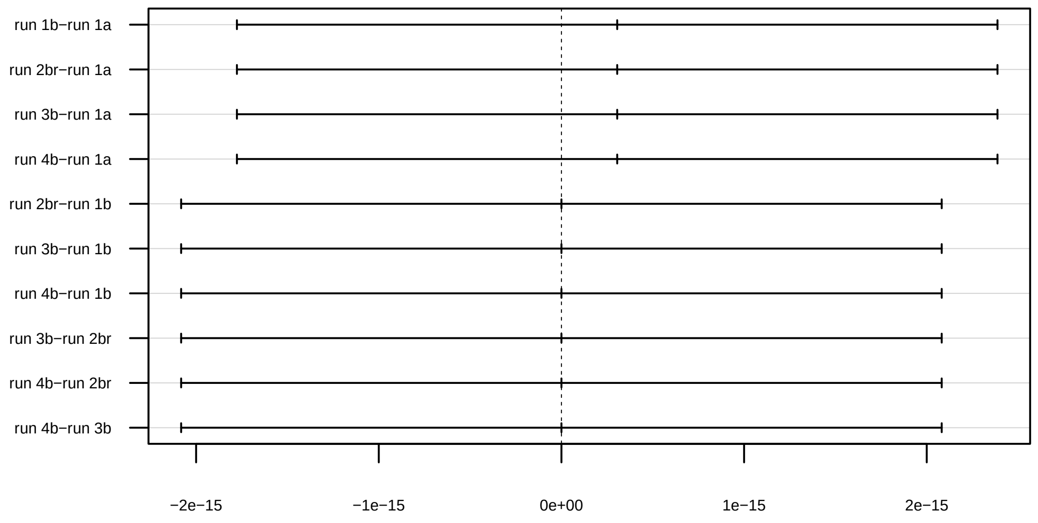
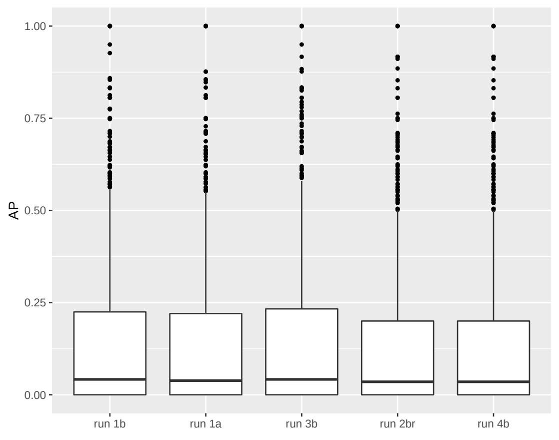
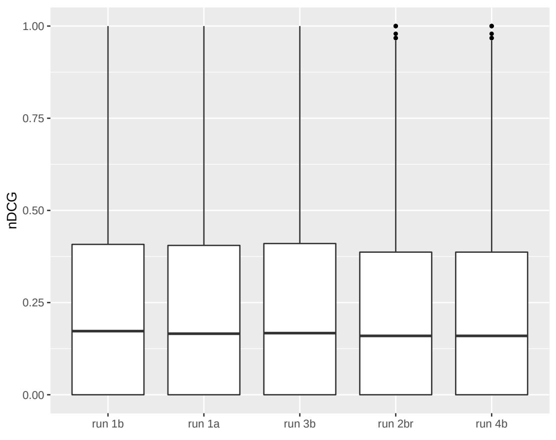


Figure 7: Tukey's results for Short-term set



(a) AP



(b) nDCG

Figure 8: Boxplot for Long-Term set runs

- The median is overall the same for all the runs;
- The outliers are more extreme with respect to what it's shown in Figure 6. Furthermore, looking at Figure 8a it can be noticed that outliers are more concentrated in the range between 0.5 and 0.75.
- Looking at the 3rd quartile, these runs exhibit the highest scores compared to what was observed in the other sets.

Table 9
Results of Two-way ANOVA test over MAP of runs and queries of the Long-term set

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Topics	920	180.77	0.19649	70.98	<2e-16
Runs	4	0.13	0.03333	12.04	1.01e-09
Residuals	3674	10.17	0.00277	-	-

Table 10
Results of Two-way ANOVA test over nDCG of runs and queries of the Short-term set

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Topics	918	301	0.3279	1.123e+27	<2e-16
Runs	4	0.0	0.0000	8.290e-01	0.506
Residuals	3672	<2e-16	<2e-16	-	-

The Two-way ANOVA tests performed for MAP (Table 9) and nDCG (Table 10) suggest that:

- **Topics** are significant for both MAP and nDCG results;
- **Runs** are significant only for the MAP results.

The Tukey HSD test for the Long-Term set in Figure 9 is showing again that runs are similar between each others. The graph is leading to the same conclusions obtained in Section 5.2.1 and Section 5.2.2:

- Run 1a (WordNet without re-ranking) is similar to run 1b, (Basic without re-ranking) run 2br (FastText with re-ranking), run 3b (Stems without re-ranking), run 4b (FastText + stems without re-ranking). These pairwise comparisons are the most significant;
- Run 1b and run 2br are similar to each other, and they are showing the same behavior when compared with run 3b, run 4b. These pairwise comparisons are less significant.

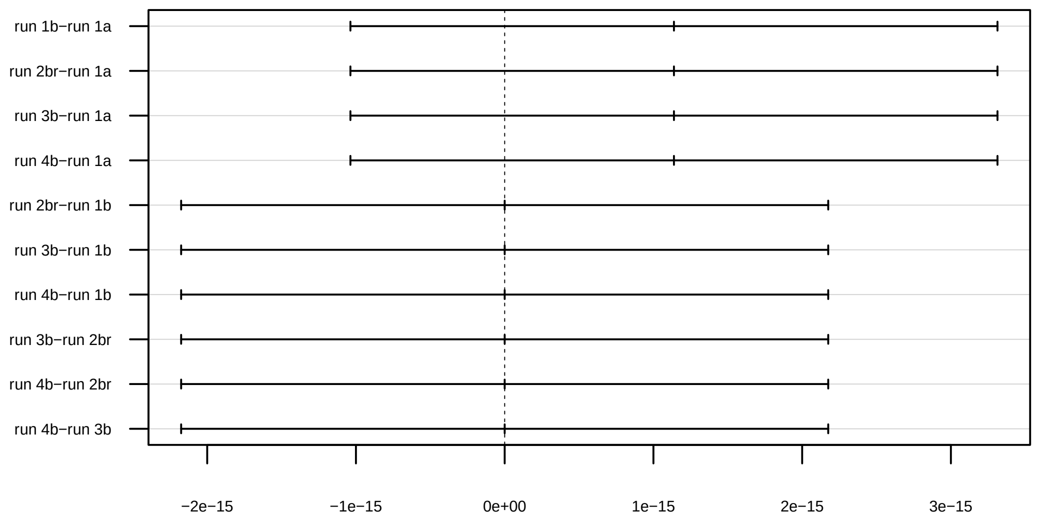


Figure 9: Tukey's results for Long-term set

5.3. Failure Analysis

The main problem in this system is that it finds few relevant results, for example in model 1a it does not find *tax*, even though the search key is *taxation*. In models 1b (Basic without re-ranking), 2br (FastText with re-ranking), 3a (Stems without re-ranking), 4a (FastText + stems without re-ranking) this specific case is solved, despite this, numerous similar cases remain. Limiting the number of relevant documents to 50 (otherwise re-ranking becomes impossible) makes it necessary to compute a score that reflects the real importance of a word or synonym. However, this comes out to be a problem.

Furthermore, it has been seen that comparing the runs on most queries the results are almost the same. For example the runs on query `q062222425` return almost the same order of documents with similar scores, whereas for query `q062214361` they return most of the same documents, often in a slightly different order, and lastly regarding query `q062211403` the top-ranked results are the same among the runs, only those from the 30th onwards change. We can generally classify queries in these 3 cases.

In addition, models 1b, 2a, 3a, 4a compared to model 1a only improve the results for certain cases, however, the improvement is not generalized since they do not change the results.

5.4. Discussion

Overall the system presented in this document implements some interesting techniques, for example, a lot of filters were implemented in order to make it possible to compare them and select the ones achieving better results for the runs. According to the results, the system that achieves better performance is the one that uses only essential filters. Nevertheless, some queries that do not achieve any result with the essential system, gain some improvements using specific filters. Despite that, the overall system performs better without these specific filters.

As already noticed, most of the runs returns the same documents with similar scores. Furthermore, as can be seen in Tables 2, 5 and 8, the MAP and nDCG values in each table are almost the same. It is important to highlight that the values for both these measures increase over time, with the highest scores being observed for the September's collection. This last consideration leads to the conclusion that the developed system well performs with data further away in time from the training data, that was the final goal the system was intended to achieve.

6. Conclusions and Future Work

This system presents many flaws, but taking it as a starting point, there are several ways to improve it:

- Develop a re-ranking method that better suits the system
- Integrate in a more comprehensive way the synonyms generated through the query expansion
- Implement a more refined tokenization system (for example, one that deals with hyphens)
- Transform it into a 3-way re-ranking system: i.e. a first query that searches many more documents, a second query for selection and re-ranking, and then finally a re-ranking similar to the one we implemented

Other interesting features that could be developed are specific filters for relevant cases that do not worsen the overall performance, based on the ones already developed.
Secondly, since WordNet has not been updated in the last 10 years, FastText could be an alternative to it in order to generate synonyms on the fly. Developing these mentioned features would surely increase the performance of this system.
Furthermore, working with the French collection should give the system a boost to its precision, overcoming the translation issues, and it is also important to highlight that this dataset and the LongEval task would require the use of a machine learning statistic model able to find more suitable correspondences.

References

[1] R. Alkhalifa, I. Bilal, H. Borkakoty, J. Camacho-Collados, R. Deveaud, A. El-Ebshihy, L. Espinosa-Anke, G. Gonzalez-Saez, P. Galuscakova, L. Goeuriot, E. Kochkina, M. Liakata, D. Loureiro, H. T. Madabushi, P. Mulhem, F. Piroi, M. Popel, C. Servan, A. Zubiaga, Overview of the clef-2023 longeval lab on longitudinal evaluation of model performance, in: Experimental IR Meets Multilinguality, Multimodality, and Interaction. Proceedings of the Fourteenth International Conference of the CLEF Association (CLEF 2023), Lecture Notes in Computer Science (LNCS), Springer, Thessaliniki, Greece, 2023.

[2] Qwant, Qwant, https://www.qwant.com, 2023.

[3] P. Galuščáková, R. Deveaud, G. Gonzalez-Saez, P. Mulhem, L. Goeuriot, F. Piroi, M. Popel, Longeval-retrieval: French-english dynamic test collection for continuous web search evaluation, 2023. arXiv:2303.03229.

[4] R. Alkhalifa, E. Kochkina, A. Zubiaga, Building for tomorrow: Assessing the temporal persistence of text classifiers, Information Processing & Management 60 (2023) 103200.

[5] R. Alkhalifa, I. Bilal, H. Borkakoty, J. Camacho-Collados, R. Deveaud, A. El-Ebshihy, L. Espinosa-Anke, G. Gonzalez-Saez, P. Galuščáková, L. Goeuriot, E. Kochkina, M. Liakata, D. Loureiro, H. Tayyar Madabushi, P. Mulhem, F. Piroi, M. Popel, C. Servan, A. Zubiaga, Longeval: Longitudinal evaluation of model performance at clef 2023, in: Advances in Information Retrieval: 45th European Conference on Information Retrieval, ECIR 2023, Dublin, Ireland, April 2–6, 2023, Proceedings, Part III, Springer-Verlag, Berlin, Heidelberg, 2023, pp. 499–505.

[6] R. Sequiera, J. Lin, Finally, a downloadable test collection of tweets, in: Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR ’17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 1225–1228.

[7] E. M. Voorhees, The trec 2005 robust track, SIGIR Forum 40 (2006) 41–48.

[8] E. Voorhees, T. Alam, S. Bedrick, D. Demner-Fushman, W. R. Hersh, K. Lo, K. Roberts, I. Soboroff, L. L. Wang, Trec-covid: Constructing a pandemic information retrieval test collection, SIGIR Forum 54 (2021).

[9] E. M. Voorhees, D. K. Harman, Overview of the seventh text retrieval conference (trec-7) [on-line], 1999.

[10] M. Braschler, Clef 2000 – overview of results, 2001, pp. 9–26.

[11] M. Braschler, Clef 2001 – overview of results, in: C. Peters, M. Braschler, J. Gonzalo, M. Kluck (Eds.), Evaluation of Cross-Language Information Retrieval Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2002, pp. 9–26.

[12] M. Braschler, Clef 2002 – overview of results, in: C. Peters, M. Braschler, J. Gonzalo, M. Kluck (Eds.), Advances in Cross-Language Information Retrieval, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, pp. 9–27.

[13] L. Kelly, L. Goeuriot, H. Suominen, A. Neveol, J. Palotti, G. Zuccon, Overview of the clef ehealth evaluation lab 2016, in: N. Fuhr, K. Balog, N. Ferro, B. Larsen, P. Quaresma, T. Goncalves, C. Macdonald, L. Cappellato (Eds.), Experimental IR Meets Multilinguality, Multimodality, and Interaction: 7th International Conference of the CLEF Association, CLEF 2016, Proceedings [Lecture Notes in Computer Science, Volume 9822], Springer, Switzerland, 2016, pp. 255–266. URL: https://eprints.qut.edu.au/98864/.

[14] L. Kelly, H. Suominen, L. Goeuriot, M. Neves, E. Kanoulas, D. Li, L. Azzopardi, R. Spijker, G. Zuccon, H. Scells, J. Palotti, Overview of the clef ehealth evaluation lab 2019, in: F. Crestani, M. Braschler, J. Savoy, A. Rauber, H. Müller, D. E. Losada, G. Heinatz Bürki, L. Cappellato, N. Ferro (Eds.), Experimental IR Meets Multilinguality, Multimodality, and Interaction, Springer International Publishing, Cham, 2019, pp. 322–339.

[15] P. Bojanowski, E. Grave, A. Joulin, T. Mikolov, Enriching word vectors with subword information, arXiv preprint arXiv:1607.04606 (2016).

[16] Princeton University, WordNet database, https://wordnet.princeton.edu/, 2000-2023.

[17] J. Palotti, TRECtools, https://github.com/joaopalotti/trectools, 2023.

[18] I. Brigadir, J. Nothman, igorbrigadir/stopwords: First release, 2019.

[19] M. P. Marcus, B. Santorini, M. A. Marcinkiewicz, Penn Treebank P.O.S. Tags — ling.upenn.edu, https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html, 1989-1992.

[20] Z. Wu, M. Palmer, Java Libraries for Accessing the Princeton Wordnet: Comparison and Evaluation, https://aclanthology.org/P94-1019/, 1994.

[21] Finlayson, M. Alan, Java Libraries for Accessing the Princeton Wordnet: Comparison and Evaluation, http://projects.csail.mit.edu/jwi/, 2014.

[22] K. Vinh, S. Carsten, L. Théo, Jfasttext: Java interface for fasttext, https://github.com/stroppycow/JFastText, 2022.

[23] T. Mikolov, E. Grave, P. Bojanowski, C. Puhrsch, A. Joulin, Advances in pre-training distributed word representations, in: Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018), 2018.

[24] E. Grave, P. Bojanowski, P. Gupta, A. Joulin, T. Mikolov, Learning word vectors for 157 languages, in: Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018), 2018.