SEUPD@CLEF: Team DARDS - IR system for short and long term retrieval

Notebook for the LongEval Lab at CLEF 2023

Daniel **Carlesso**[1], Riccardo **Gobbo**[1], Simone **Merlo**[1], Angela **Pomaro**[1,2], Diego **Spinosa**[1] and Nicola **Ferro**[1]

[1]University of Padua, Italy
[2]University of Rome La Sapienza, Italy

Abstract

Search Engines represents an important application of Information Retrieval. In particular, a major branch of Search Engines is devoted to web search.

In this document we summarize our work to produce a submission for the CLEF LongEval initiative, primarily concerning web search. The described activity first focuses onto the development of an indexing and searching IR system with the best possible performance based on the provided training data then evaluates its performance on test data coming from different scenarios.

We first introduce the task and related problems. Subsequently we present the retrieval systems that we have used for the program submission.

Afterwards, we discuss the results obtained with the various systems and compare them in the while-time training scope to explain why some systems perform better than others.

Finally, metrics analysis is extended to the additional scenarios LongEval focuses on, along with statistical considerations over the systems' output.

Keywords

CLEF 2023, LongEval, Spam detection, Translation, Synonyms, Reranking, Boosting, Query Expansion, Document Expansion, ANOVA

1. Introduction

Search Engines (SE) are used daily by every person in the world. In particular, the main application of SE is the web search consisting in the retrieval of documents (web pages) in the web. Generally, people expect to provide a sentence as input (a query, representing their interest or information need) and to get back a list of results that are highly related (relevant) to that piece of text.

This document summarizes the information retrieval systems developed by the team DARDS as part of the Search Engines 2022/2023 course, which is held at the University of Padua, in

order to address the task 1 "Retrieval" of the "LongEval" lab proposed by CLEF 2023. The goal of the lab, hence of the systems, is to retrieve the most relevant documents given a query (a short sentence representing what the user would use to look for its information need). Furthermore, the lab aims at understand the time persistence and reliability of the developed systems by testing them using some data sampled at different times.

The paper is organized as follows: Section 2 describes our approach; Section 3 explains our experimental setup; Section 4 discusses our main findings with relative failure analysis; eventually, Section 5 draws some conclusions and outlooks for future work.

2. Methodology

The first goal we intended to achieve was to reach a stable and simple version of a basic indexing and searching system, to be used as a baseline to experiment additional features on.

Afterwards, the workflow split into several lines, each applying a different improvement strategy, such as different analyzing techniques, different filter configurations or different document preprocessing operations. Once a better system was found, it would eventually become the new baseline to run further experiments on. In order to operate in this way, each line would evolve on a different working branch of our git repository, with its performance measured mainly through trec_eval's nDCG and/or MAP metric.

Some tools were also developed to help us analyze what kind of errors the run contained, to perform some pre-processing to the collection documents and to solve some translation problems.

The main programming language that we used to develop our systems is Java, while some tools have been developed using Python. In particular, the core of our work was implemented by exploiting the Lucene java library [1].

We can split our systems in four main components:

- **Parser**: it parses the corpus independently of the language chosen by using the TREC file format.
- **Analyzer**: it processes some text performing tokenization, stopword removal, stemming and other filtering processes.
- **Indexer**: it processes the parsed and analyzed documents fields to creates the index.
- **Searcher**: it processes a given query and exploits the index to retrieve some documents based on the processed query.

In this section we describe the general workflow of our systems (Section 2.1), the developed tools (Sections 2.2 and 2.3), the general components (Sections 2.4, 2.5, 2.6 and 2.7) and the complete systems (Section 2.8).

2.1. General systems flow

All of our systems operate in the following order (a scheme of the system is reported in Figure 1):

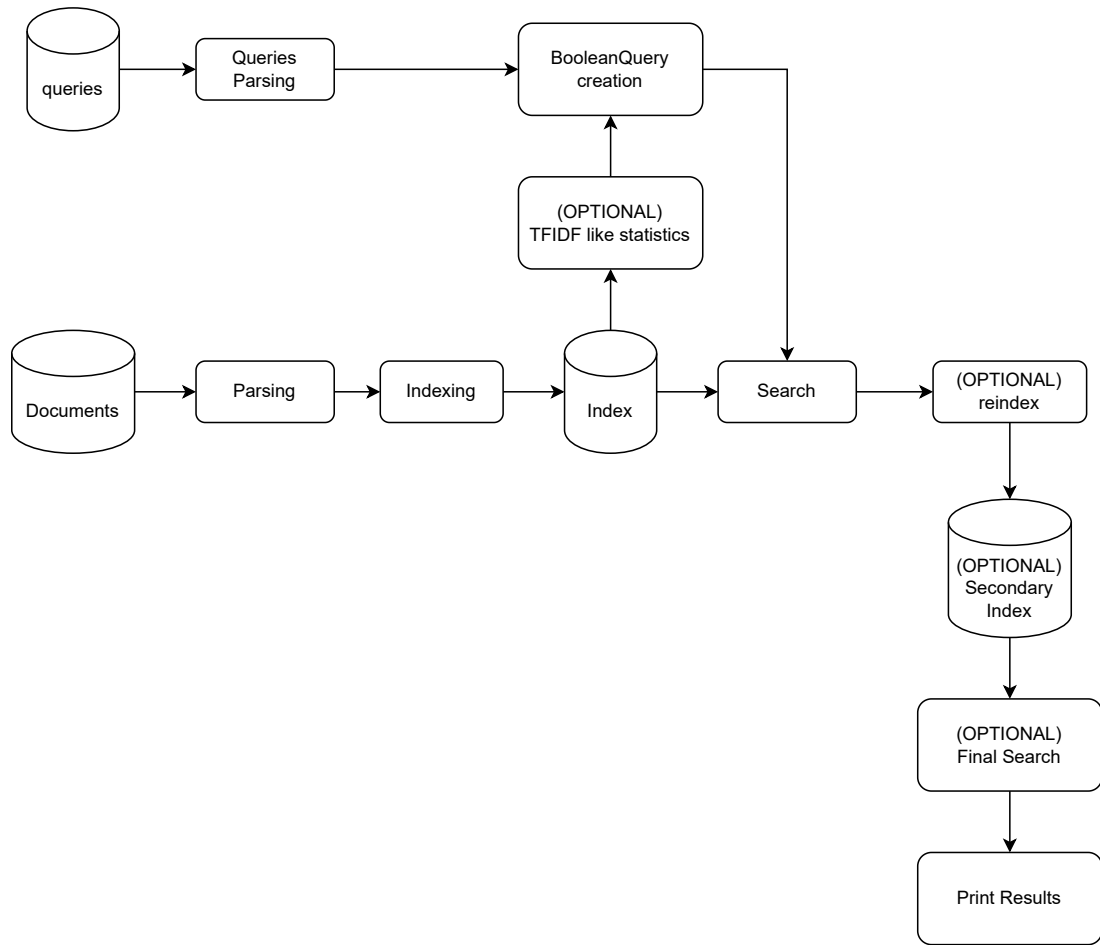1. Apply a pre-processing to the documents of the corpus (optional stage).

Figure 1: General system scheme

2. Parse the documents of the corpus, splitting the content into the appropriate fields, using the parser.
3. Analyze the documents' fields to convert them into a stream of tokens and index them to perform the search (single index folder).
4. Parse and analyze the queries to convert them into a stream of tokens.
5. Perform the search by exploiting the index (one query at the time).
6. Rerank the documents retrieved with the search (optional stage).
7. Print the results of the search into a dedicated text file.

2.2. Pre-processing tools

2.2.1. SPAM parser

By analyzing the results produced with the French-based system (the simplest one) on the French collection, we realized that relevant documents for a specific query were not included in

the first one thousand documents retrieved by the system for that query. Before increasing the system's complexity through new features, while inspecting the retrieved documents, it turned out that several of the non-relevant documents were actually ranked as relevant documents due to the high number of words repetitions inside them. More precisely, excluding documents that were actually not relevant, we noticed that in the analyzed documents the words were not different and the text contained a sensible number of repetitions. This new information drove us to develop term frequency analyses to avoid the retrieval of documents considered "spam" during the search task or, even better, the addition to the Lucene index during the index task, reducing its overall size.

With the term "spam" we denote the type of documents that "hack" the retrieval system through the use of a set of words contained in the user query, that brings the document to the top of the document-ranked list.

The following documents are some examples of "spam". We can notice that the length of the texts is very short compared to the one of an average document:

*"**Tableau** de conversion pouce / inch en Cm mm | **Tableau** de conversion, **Tableau** de conversion de mesure, Scrapbooking à imprimer"*

*"[...] **Quiz** : Cinéma **Quiz** : Histoire de France **Quiz** : Géographie **Quiz** : Histoire **Quiz** : Littérature **Quiz** : Espace **Quiz** : Bien-être **Quiz** : Littérature **Quiz** : Gamer **Quiz** : Formule 1 **Quiz**"[...]*

The methodology that we applied to fulfil this goal takes into account two main factors:

1. The ratio between the most frequent word and the total length of the document, excluding articles and stopwords:
$$\frac{\text{Max freq}}{\text{Doc length}}. \tag{1}$$

   The value obtained is as low as the term occurrences are spread all over the document.

2. The ratio between the number of words in the document and the number of different words used in it:
$$\frac{\text{Doc length}}{\text{Number different words}}. \tag{2}$$

   This value reaches higher values for documents which contain the same few words repeated many times. For this reason, they may not contain any useful information for the user.

In order to combine the two approaches, we can obtain a third ratio by multiplicating the previous ones and comparing the result with a threshold:

$$\frac{\text{Max freq}}{\cancel{\text{Doc length}}} \cdot \frac{\cancel{\text{Doc length}}}{\text{Number different words}} = \frac{\text{Max freq}}{\text{Number different words}} > \text{threshold} \tag{3}$$

During the process, one more threshold is checked: the length of the analyzed word. Since the implemented parser splits the sentences using blank spaces, it may happen that sequence of
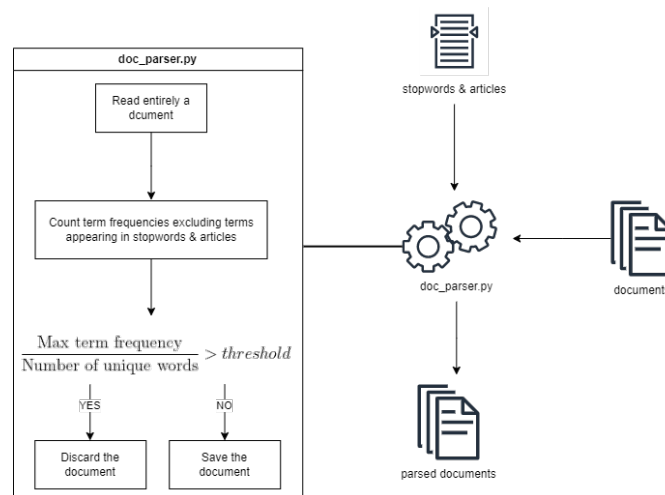
Figure 2: Flow of the documents parser. (docs_filter.py).

characters (e.g. "★★★", "NVO",..) or symbols (e.g. €) cause an erroneous document exclusion. By applying the proposed threshold these errors are avoided.

The entire pre-processing task was implemented in a separate Python file which automatically reads all the files in the French collection and for each file it saves a new one that doesn't contain the documents marked as spam. The overall flow is reported in Figure2.

2.2.2. Synonyms

In order to try to increase the system's performance, a query expansion method was implemented through the use of synonyms. The files containing French synonyms available on the internet were not sufficient and covered only a small part of the entire French dictionary. For this reason, a new collection of words that could be easily used as synonyms of words found in the queries during the search task was needed. To address this need, another Python tool that automatically performs a request to a specialized site (dictionary.reverso.net/french-synonyms/) and retrieves all the related synonyms was created. All the new words found are placed in the same line as the searched word so that the Searcher (2.7) can easily extract the information needed. The overall flow is shown in Figure 3.

2.3. Translation tool

Since the translations of the documents and the queries provided by CLEF resulted to be imprecise, we developed a tool that allows to translate a piece of text from French to English. The tool has been implemented as a script in Google's "Google App Script" platform and then deployed as a web application. This tool must be used as a REST resource.
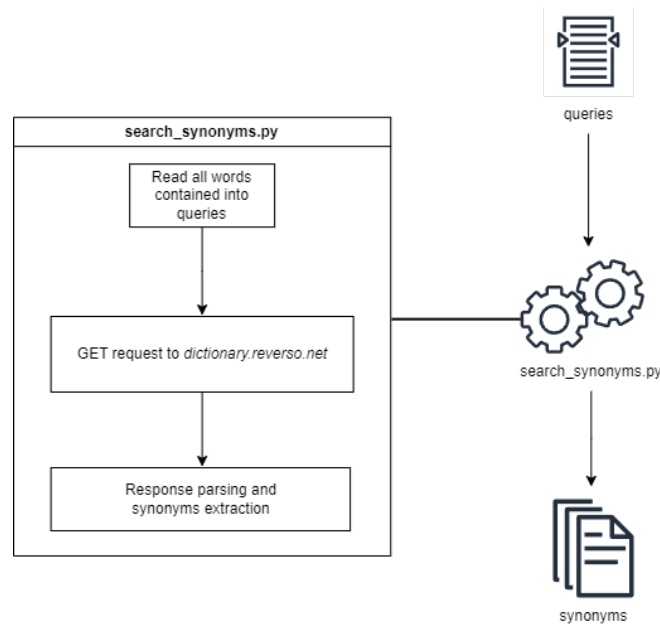
The code used to implement this tool can be found HERE.

Figure 3: Flow of the synonyms searcher. (search_synonyms.py).

2.4. Parser

The parser that we implemented for all of our systems is a basic parser. This component simply reads all the files with a given extension (.txt) present in a directory tree (specified as input) and converts all the **Trec** formatted documents into an instance of document that has two fields: the identifier of the document (ID) and the body of the document (BODY). Some systems in addition to that also read another file (whose path is given as input) that contains pairs "document identifier"-"document url" and add an additional url field (URL) to the instances of the documents based on the identifier. In some systems, we decided to add this field for two reasons: the URL could help decreasing the rank position of the document containing "spam" (see section 2.2) and it could help us in the queries that contain a website url.

Furthermore, we tried also to add another field to the document instances that was meant to contain the document keywords (retrieved exploiting an algorithm called RAKE [2], Rapid Automatic Keyword Extraction), but it required too much computation time so we didn't implement this option.

Eventually, the parsing of the queries uses the **tsv** file format and it is done in the searcher module (see Section 2.7).

2.5. Analyzer

In our systems we implemented customized analyzers instead of using the standard ones proposed by Lucene. This choice was motivated by the fact that this way we could customize the filters pipeline and choose a tokenizer. Tokenizers process the input stream (the document text) and using some predefined rules they split it into tokens: these are the atomic units

composing a document. Unlike words, tokens do not have to be grammatically correct or meaningful, but are suitable for processing and comparison operations. We tried two different tokenizers:

- **StandardTokenizer**: a grammar-based tokenizer constructed with JFlex. It implements the Word Break rules from the Unicode Text Segmentation algorithm, as specified in Unicode Standard Annex #29.
- **WhiteSpaceTokenizer**: it breaks text into terms whenever it encounters a whitespace character.

We started out using StandardTokenizer and we noticed that expressions separated with hyphens were divided into different tokens. To avoid this, we tried a combination of WhiteSpaceTokenizer and PatternReplaceFilter to remove all punctuations/symbols except for hyphens. But then performance decreased and we backtracked.

In addition to the StandardTokenizer we used different filters, some of them shared by both English-based and French-based systems:

French

- **ElisionFilter**: it targets elisions, so removes articles, prepositions and conjunctions placed either in the initial or final part of the token, usually connected by an apostrophe or hyphen. This connection makes it impossible for a generic stopword filter alone to detect and delete those particles.
- **ASCIIFoldingFilter**: it converts alphabetic, numeric, and symbolic Unicode characters which are not in the Basic Latin Unicode block (the first 127 ASCII characters) to their ASCII equivalents, if one exists. This filter was used to solve accents and diacritical marks problem with French.
- **FrenchLightStemFilter**: this stemmer implements the "UniNE" algorithm [3].

Note that for the French case other stemmers (FrenchMinimalStemmer, org.taurus.snowball's FrenchStemmer) and chain of stemmers (multiple stemmers in a single analyzer) have been tested and evaluated but the performance of the FrenchLigthStemmer turned out to be the best.

English

- **NGramFilter**: it generates character n-gram tokens of sizes in the given range. Tokens are ordered by position and then by gram size.
- **ShingleFilter**: it constructs word n-grams, which are token n-grams, from the token stream.
- **PorterStemFilter**: it applies the Porter Stemming Algorithm for English. The results are similar to using the Snowball Porter Stemmer with the language="English" argument. It is coded directly in Java and it is four times faster than the English Snowball stemmer.
- **KStemFilter**: it is an alternative to the Porter Stem Filter for developers looking for a less aggressive stemmer. KStem was written by Bob Krovetz, ported to Lucene by Sergio Guzman-Lara (UMASS Amherst). This stemmer is only appropriate for English language text.

Shared

- **LowerCaseFilter**: it converts any uppercase letter in a token to the equivalent lowercase token. All other characters are left unchanged.
- **StopFilter**: it discards, or stops analysis of, tokens that are on the given stop words list.
- **SynonymGraphFilter**: it maps single- or multi-token synonyms, producing a fully correct graph output. If used for indexing it must be followed by a FlattenGraphFilter to squash tokens on top of one another, because the indexer can't directly consume a graph.
- **HyphenatedWordsFilter**: it reconstructs hyphenated words that have been tokenized as two tokens because of a line break or other intervening whitespace in the field test. If a token ends with a hyphen, it is joined with the following token and the hyphen is discarded.
- **RemoveDuplicatesTokenFilter**: it removes duplicate tokens in the stream. It filters out Tokens at the same position and Term text as the previous token in the stream.
- **NumberFilter**: it removes every token in the token stream that happens to be a number (this filter has been developed by team DARDS).

2.6. Indexer

The indexer's job is to store the tokens, generated by processing every document with a given analyzer, into an inverted index (data structure which maps terms to documents). We implemented two types of indexer, *DirectoryIndexer* and *ReRankDirectoryIndexer*, the first one takes all the documents in a given directory and stores them into the index, the second one does the same but it can also store a list of documents passed to it.

Lucene captures some statistics at indexing time which can then be used to support scoring at query time. To do this it makes use of similarity functions, which set a per-document value for every field in the document. Hence, the similarity determines how Lucene weights terms.

In our systems we tested different similarities:

- **BM25Similarity**: it implements the Okapi BM25 (Best Match, attempt 25).
- **LMDirichletSimilarity**: bayesian smoothing using Dirichlet priors. The formula assigns a negative score to documents that contain the term, but with fewer occurrences than predicted by the collection language model.
- **LMJelinekMercerSimilarity**: language model based on the Jelinek-Mercer smoothing method. The model has a single parameter, λ. The optimal value is around 0.1 for title queries and 0.7 for long queries. Values near zero act score more like a conjunction (coordinate level matching), whereas values near 1 behave the opposite (more like pure disjunction).
- **DFRSimilarity**: it implements the divergence from randomness (DFR) framework. Probabilistic models of IR based on measuring the divergence from randomness. The DFR scoring formula is composed of three separate components: the basic model, the aftereffect and an additional normalization component.
- **ClassicSimilarity**: subclass of TFIDFSimilarity. Lucene combines Boolean model (BM) of Information Retrieval with Vector Space Model (VSM) of Information Retrieval. In VSM, documents and queries are represented as weighted vectors in a multi-dimensional space, where each distinct index term is a dimension, and weights are Tf-idf values. VSM

Similarity	MAP
BM25Similarity	0.1424
LMJelinekMercerSimilarity(0.1F)	0.1249
LMJelinekMercerSimilarity(0.5F)	0.1255
LMJelinekMercerSimilarity(0.8F)	0.1242
DFRSimilarity (most of the configurations)	0.1423
LMDirichletSimilarity	0.1204
IndriDirichletSimilarity	0.0137
ClassicSimilarity	0.0740
BooleanSimilarity	0.0121
AxiomaticF2LOG	0.1358
AxiomaticF2EXP	0.1346

Table 1
Comparison of rerank performance.

score of document d for query q is the Cosine Similarity of the weighted query vectors
V(q) and V(d).

- **BooleanSimilarity**: it gives terms a score that is equal to their query boost.
- **AxiomaticF2LOG**: it is defined as $\sum tfln(\text{term\_doc\_freq}, \text{docLen}) \times \text{IDF(term)}$ whit
  $\text{IDF(term)} = ln\left(\frac{N+1}{df(t)}\right)$; where N equals total number of docs, df corresponds to the
  docs frequency.
- **AxiomaticF2EXP**: it is defined as $\sum tfln(\text{term\_doc\_freq}, \text{docLen}) \times \text{IDF(term)}$ whit
  $\text{IDF(term)} = \left(\frac{N+1}{df(t)}\right)^k$; where N equals total number of docs, df corresponds to the docs
  frequency.
- **IndriDirichletSimilarity**: bayesian smoothing using Dirichlet priors as implemented
  in the Indri Search engine. A larger value for parameter μ produces more smoothing.
  Smoothing is most important for short documents where the probabilities are more
  granular.
- **MultiSimilarity**: particular similarity which implements CombSUM method for combin-
  ing evidence from multiple similarity values [4].

In the large majority of cases the BM25Similarity turned out to be the best option to use (see
Table 1. Note that the values of the following table refer to an initial system that performed the
search on the English corpus provided by CLEF).

2.7. Searcher

The searcher is the other fundamental part of any IR system besides the Indexer. This component
takes care of doing the matching between a document and a topic, giving as a result a ranking,
which marks how relevant a document should be for a certain topic. In our application, topics
are real user queries.

In our systems, the searcher have been customized to first parse the queries from a provided, .tsv formatted file. Then, one query at the time, this component turns the query into a stream of tokens (exploiting a query parser and an analyzer which must be the same used for indexing), converts it into a Lucene Query and searches the index for it. After performing the actual search this tool optionally executes a rerank (see Section 2.7.2) and then prints the results of the overall process into a file.

Note that in some systems the query is not directly turned into a Lucene Query but in the process the terms are boosted (see Section 2.7.1). Furthermore, in the majority of cases the query is performed by considering the BODY field of the document but in some systems the same query is applied also to the URL field.

2.7.1. Boosting

In some of our systems we decided to not to use a plain query but to boost the term according to some statistics. In particular, we computed a boost factor based on a measure that is quite similar to the the TF-IDF weight. The equations used to compute the boost factor for the term i are the following:

$$w_i = \left(\frac{ttf_i}{\sum_j ttf_j} \right)^{0.2} \cdot \left(1 + \log \frac{N+1}{n_i+1} \right),$$
(4)

where with ttf_i we indicate the "total term frequency" (number of occurrences of the term in the entire corpus), with N the number of documents in the corpus and with n_i we denote the number of documents containing the term.

In order to boost the query terms we:

1. Compute from the index an hash map that contains as keys the terms and as values the related weight.
2. Get (exploiting the analyzer), from the original text of the query, the list of terms that will compose our Lucene Query.
3. Create a TermQuery for each of the term.
4. Wrap the TermQuery in a BoostQuery, setting the boost based on the hash map (the boost is set to zero if the term doesn't appear in the index).
5. Merge all the BoostQuery(ies) in a BooleanQuery.

2.7.2. Reranking

After the searcher performed the task, some systems execute a rerank. To implement the reranking we simply consider the document retrieved by the search (hits), we create a new index based only on those documents (or only on the top-N document retrieved) and eventually we repeat the same or a slightly modified query (depending on the system) on the new index.

2.8. Systems

In this subsection we describe the systems that we implemented. The overall system sequence diagram is reported in Figure 1.

2.8.1. BM25FRENCHBASE system

As a baseline for all our work, this IR system for French uses:

Query language	French
Document language	French
Preprocessing	None
Similarity	BM25Similarity
Analyzer	Custom, composed as below
Tokenizer	StandardTokenizer
Filters	ElisionFilter, LowerCaseFilter, StopFilter, ASCIIFoldingFilter, FrenchLightStemFilter
Searcher	Standard searcher

This configuration has been achieved by trying all possible filter configurations while retaining only the better performing ones.

The stoplist used by the StopFilter is a custom stoplist created by merging different French stoplist found online and by adding some terms based on the index created (that has been analyzed using Luke).

The article's list used by the ElisionFilter is a custom list of terms that we created on the basis of French dictionaries.

The ASCIIFoldingFilter was used to avoid problems related to French diacritical marks and accents since not in all queries and/or documents were used properly.

This system has overall good performance on the test data, especially considering its moderate complexity.

2.8.2. BM25FRENCHBOOSTURL system

The BM25FRENCHBOOSTURL system considers a document with three fields: ID,BODY,URL (see Section 2.4). Furthermore this system uses:

Query language	French
Document language	French
Preprocessing	None
Similarity	BM25Similarity
Analyzer	Custom, composed as below
Tokenizer	StandardTokenizer
Filters	ElisionFilter, LowerCaseFilter, StopFilter, ASCIIFoldingFilter, FrenchLightStemFilter
Searcher	Standard searcher + boosting (see Section 2.7.1)

In this system the query is performed on both the BODY field and the URL field of the document but in a slightly different way. In particular, the main boolean query is composed of two sub-queries: one for the BODY field where each term is connected by means of an OR clause, the other for the URL field where term is connected by means of an AND clause. Eventually, the two sub-queries are connected by means of an OR clause to compose the final query.

Furthermore, boosting is performed, as described in Section 2.7.1, in the same way for both the URL and BODY field sub-queries (we tried different combination of boosting, for example to

Table 2

Query translation error examples

Query ID	French	English
q062213307	cuisson gigot agneau	leg leg
q062228	aeroport bordeaux	airport

give to the URL sub-query terms double the weight, but this combination turned out to be the best).

The stoplist used by the StopFilter is a custom stoplist created by merging different French stoplist found online and by adding some terms based on the index created (that has been analyzed using Luke).

The article's list used by the ElisionFilter is a custom list of terms that we created on the basis of French dictionaries.

The ASCIIFoldingFilter was used to avoid problems related to French diacritical marks and accents since not in all queries and/or documents were used properly.

2.8.3. BM25TRANSLATEDQUERIES system

The BM25TRANSLATEDQUERIES system has been developed as an attempt of running an English system with a better query quality: looking at the supplied English queries, multiple translation mistakes and imprecisions were noticeable (see Table 2 for some examples). Therefore, a custom translator was implemented, as defined in Section 2.3.

The translation of the queries is performed in the searcher right after their parsing and before turning them into a token stream.

This system uses:

Query language	French, translated in English
Document language	English
Preprocessing	None
Similarity	BM25Similarity
Analyzer	Custom, composed as below
Tokenizer	StandardTokenizer
Filters	LowerCaseFilter, StopFilter, KStemFilter
Searcher	Standard searcher + query translation

By doing several experiments we noticed that with English the best performing stemmer was the Krovetz stemmer (*KStemFilter*).

The StopFilter does not use a customized stoplist but it uses the standard English stoplist provided by the Lucene class EnglishAnalyzer through the static variable ENGLISH_STOP_WORDS_SET.

Furthermore, to increase the performance, we tried to translate also the documents by using our tool (see Section 2.3) but, since the tool is used as a REST resource and because of the number and length of the documents, this approach resulted to be too demanding in terms of computational time, so we abandoned this idea.

2.8.4. BM25FRENCHRERANK100 system

This system has been developed to evaluate the benefits of introducing reranking mechanics (as defined in Section 2.7.2). In order to achieve a better performance, query boosting (see Section 2.7.1) is also included in the Searcher. In comparison to 2.8.2, this system does not take into account the document's URL but it considers only the BODY and ID fields.

Here follows a more detailed component overview:

Query language	French
Document language	French
Preprocessing	None
Similarity	BM25Similarity
Analyzer	Custom, composed as below
Tokenizer	StandardTokenizer
Filters	ElisionFilter, LowerCaseFilter, StopFilter, ASCIIFoldingFilter, FrenchLightStemFilter
Searcher	Standard searcher + boosting (see Section 2.7.1) + reranking

The rerank is performed as described in Section 2.7.2 and, in this case, the query that is used to repeat the search is the same used for the main search. The query created is a BooleanQuery obtained by connecting the BoostedQuery(ies) (each representing a boosted token of the topic) by means of an OR clause.

Note that this system performs the rerank only for the first (most relevant) 100 document retrieved by the first search (for the CLEF LongEval task the maximum number of retrieved documents is 1000). We have chosen to rerank only 100 documents because by looking at the performance through the Trec_Eval tool we noticed that the recall at the document cut-off 100 was sufficiently high. Nonetheless, we also tried to perform the rerank considering all the retrieved document but the performance resulted to be lower.

Finally, we tried to boost the query performed in the second search in a different way (by recomputing the TF-IDF-like weights considering only the secondary index created in during the reranking step) but the performance decreased (see Table 3).

The stoplist used by the StopFilter is a custom stoplist created by merging different French stoplist found online and by adding some terms based on the index created (that has been analyzed using Luke).
The article's list used by the ElisionFilter is a custom list of terms that we created on the basis of French dictionaries.
The ASCIIFoldingFilter was used to avoid problems related to French diacritical marks and accents since not in all queries and/or documents were used properly.

	rerank100	rerank1000
MAP	0.1960	0.1702
nDCG	0.3657	0.3379
recall	0.8437	0.8437
p@5	0.1533	0.1336

Table 3

Comparison of rerank performance.

THRESHOLD	DOC KEPT	MAP	P@5	R@1000	nDCG
7%	$\simeq 74\%$	0.1462	0.1378	0.4869	0.2564
9%	$\simeq 83\%$	0.1729	0.1508	0.6135	0.3041
11%	$\simeq 88\%$	0.1891	0.1560	0.6869	0.3335
15%	$\simeq 93\%$	0.2067	0.1607	0.7648	0.3623

Table 4

Comparison of spam parsing performance.

2.8.5. BM25FRENCHSPAM system

The system BM25FRENCHSPAM implements entirely the BM25FRENCHBASE (2.8.1) applying documents preprocessing. All documents used as input are first parsed by a "spam" parser implemented in Python, as described in Section 2.2.1, and executed before the BM25FRENCHSPAM analyses. Since the pre-process activity uses a threshold above which a document is excluded, it was necessary to understand the average value ($\frac{\text{Max freq}}{\text{Number different words}}$) of the collection, assuming that most of the documents were valid (i.e. contain meaningful sentences). Using a slightly modified version (not reported) of the already existing parser (Figure 2) it turned out that the average ratio of the test collection corresponded to 6.5%.

In order to assign as correct a value as possible, some test has been done using different thresholds and checking the corresponding performance. More specifically, the considered tested values are 7%, 9%, 11% and 15% and the performance are reported in Table 4.

With this kind of methodology it was expected that, for thresholds below the average value reported above, the performance would have dropped down since the parsing was acting aggressively removing the relevant documents. On the other hand, for bigger thresholds, it was also expected bad performance since the action of the filter would have been not significant. An intermediate value, instead, would have ensured the right balance between the two cases. The heuristic results show an opposite behaviour of the performance from the one described above since the increase in performance as the threshold increases. This behaviour suggests other implementations (as reported in Section 5) of the frequencies analyses possibly weighting the two ratios used during the comparisons with thresholds. With the consideration of the data discussed above, the threshold used during analyses with test data corresponds to 15%.

2.8.6. BM25FRENCHNOENG system

After the first spam removal attempt inspecting the documents we noticed how the majority of spam documents were written in English despite being contained in the French collection. So in this second attempt the DirectoryIndexer class was modified in order to execute a preliminary filtering of the documents to index, discarding the ones written in English. This was accomplished by scanning a copy of the document for English stopwords and another copy for French stopwords. A document is inferred to be containing English content if the amount of English stopwords it contains is at least 1.5 times the amount of French stopwords it contains. Since English words are ubiquitous in all languages, a threshold higher than 1 had to be set.

Query language	French
Document language	French
Analyzer	Same as Baseline
Preprocessing	Custom DirectoryIndexer to exclude English docs

As an example, by looking at a batch of 10 discarded documents, we can what is reported in Table 5.

Over the whole French corpus the pre-processing excludes 10,81% of the documents. The table above shows how the majority of removed documents are not relevant. However, as reported in Table 7, no significant effect (nor improvement, nor worsening) over the baseline system has been recorded. This may be traced back to at least two reasons:

- English documents are seldom picked as relevant for French queries since their tokens do not match. This causes the removal to cause small to none precision improvement.
- Some queries such as the ones in the table above are directly specified in English. This makes English documents effectively relevant for them. This causes the removal to slightly worsen recall.

As already said, this strategy is ineffective on its own but may bring some improvement if paired with other techniques (see Section 5).

Table 5
spam example

document ID	Relevant	For queries
doc062200100031	No	
doc062200100356	No	
doc062200100414	No	
doc062200100463	No	
doc062200100589	No	
doc062200100599	No	
doc062200100704	No	
doc062200100726	Yes	"youtube video converter", "video converter"
doc062200100748	No	
doc062200100757	No	
doc062200100802	No	

2.8.7. BaseSystem system

This system is actually the same as 2.8.3 but doesn't implement the query translation so, as a consequence, it uses the English queries provided by CLEF instead of the French queries. This system is used only as a benchmark for comparison with the system described in Section 2.8.3.

2.8.8. BM25FRENCHDOCEXPANSION system

In this system we implemented the idea to expand the document vocabulary to avoid vocabulary mismatch between queries and documents. To do this we use two different analyzers, one for the documents (index time) and one for the queries (search time). This system is similar to BM25FRENCHBASE (Section 2.8.1), but we added SynonymGraphFilter, FlattenGraphFilter and RemoveDuplicatesFilter to the documents' analyzer. By doing this we can store the documents with their words and the synonyms too, lightening up the search time avoiding query expansion technique.

Query language	French
Document language	French
Preprocessing	None
Similarity	BM25Similarity
Analyzer	Custom, composed as below
Tokenizer	StandardTokenizer
Filters	ElisionFilter, LowerCaseFilter, StopFilter, SynonymGraphFilter, FlattenGraphFilter, ASCIIFoldingFilter, FrenchLightStemFilter, RemoveDuplicatesTokenFilter
Searcher	Standard searcher

2.8.9. BM25FRENCHQUERYEXPANSION system

This system is similar to the previous one but instead of expanding the documents we expand the queries. It exploits the same idea of having two analyzers, one for the documents and one for the queries. This time we added SynonymGraphFilter to the queries' analyzer because sometimes users do not formulate queries using the best terms, therefore using synonyms can improve the quality of search results. This is less cost-effective than document expansion because it makes the search heavier.

Query language	French
Document language	French
Preprocessing	None
Similarity	BM25Similarity
Analyzer	Custom, composed as below
Tokenizer	StandardTokenizer
Filters	ElisionFilter, LowerCaseFilter, StopFilter, SynonymGraphFilter, ASCIIFoldingFilter, FrenchLightStemFilter
Searcher	Standard searcher

3. Experimental Setup

Overall, our experimental setup was composed by:

- **Used collections**: In our experiments we both used the French and English corpora provided by the CLEF LongEval organizers, which include 1570734 documents and 672 queries for each language. However, the French collection was favoured because of the translation errors the English corpus contains (see section 4 for a closer look).

- **Evaluation measures**: The general metric we used for evaluating the various systems was the nDCG. However, more specific recall and precision metrics had also to be taken into account to guide the improvement efforts. The tool used to evaluate the performance once the run files were created was Trec_Eval (version 9.0.7). The ground-truth considered for the evaluation was the one provided directly by CLEF organizer along with the collection.

- **Url to git repository**: https://bitbucket.org/upd-dei-stud-prj/seupd2223-dards/src/master/

- **Organization of git repository**: The organization of the git repository is accurately given in the README.md file. However, the main directories are:
  - code: which contains some sub-directories each corresponding to one of the systems and some additional run files.
  - runs: which contains the runs submitted to CLEF for the evaluation (properly organized in zipped directories having a name that corresponds to the related system).

- **Hardware used for experiments**: All the indexing, searching and eventual pre-processing work has mostly been carried out on different commercial, mid-end machines. Overall, extensive testing (especially employing advanced NLP and POS-tagging techniques) has been limited by the low computing power at our disposal that made certain processes unsustainably long to complete. We report some of the used hardware:
  - Machine 1:
    * CPU: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
    * RAM: 16,0 GB (DDR4)
    * GPU: NVIDIA GeForce GTX 1050
    * HDD: Seagate Mobile HDD ST1000LM035
  - Machine 2:
    * CPU: Intel(R) Core(TM) i5-2400S CPU
    * RAM: 8,0 GB (DDR3)
    * GPU: NVIDIA GeForce GTX 750Ti
    * HDD: 500GB 7200rpm hard drive
  - Machine 3:
    * CPU: AMD A8-7410 APU
    * RAM: 8,0 GB (DDR3)
    * GPU: Radeon R5 Graphics
    * SSD: Baititon 480GB SSD

Table 6
Non-uniform translation example

Item ID	French	English
q0622311	bourse de l emploi public	Public Employment Exchange
doc062200210641	"Sélectionnée par Emploi Public"	"Selected by Public servant"

- **Scripts and tools employed**: A tool was developed to help in the error analysis process. *tellme* is a small C program that automatically analyzes the run file, comparing it with the qrels and producing a list of all the retrieval errors. Thresholds for both precision and recall errors can be set in order to show only the most prominent errors while a verbose mode will also print to screen the content of the related query and document, aiding error inspection. You can find the tool's C source here.

4. Results

In this section we first discuss the performance on training data and how these were used to select the system to submit to the LongEval lab. After that we analyze the performance on test data (with some discussions) and report the hypotesis testing and failure analysis.

4.1. Performance on Training Data and Discussion

As aforementioned, it can be noted how work on the French corpus has been largely favoured. Due to the fact that the English document and query set were obtained by translation of the French ones, the former batch contains a lot of translation-induced noise which strongly interferes with the goal of setting up an effective IR system. This was already seen from Table 2.

Another related issue is document and queries not being translated homogeneously. As an example, in Table 6 we report a couple consisting of a query and a highly relevant document and their translations. Such a translation, while semantically correct, makes it harder for the IR system to correctly match the document to the query.

Moreover, by comparing our base French and base English systems we observed consistently worse performance on the latter (see Table 7), while employing for the two systems essentially the same techniques (adapted to the respective languages).

Overall, the best system for all metrics is the *BM25FRENCHBOOSTURL*. This particularly good performance can be due to the URL tokenizing system helping with queries expressing *ad hoc search* and *known item search*. In this scope, keywords contained inside the URL are good indicators of page content: most of the time, while looking for a particular service or a website, the best match contains its name in the page URL, since those are more likely to be official websites and sources. Furthermore, considering the fact that some documents were rich of repetitions and useless symbols, considering the URLs in the search phase allowed us to distinguish more between real relevant documents and outliers.

Another thing to observe is the *BM25TRANSLATEDQUERIES* performance in comparison to *BaseSystem*. When compared to base French systems, the results highlight how difficult query

Table 7
Training-time systems performance overview

System name	Language	nDCG	MAP	Recall@1000	CLEF-submitted?
BM25FRENCHBOOSTURL	**French**	**0.3815**	**0.2152**	**0.8421**	**Yes**
BM25FRENCHBASE	French	0.3812	0.2146	0.8451	Yes
BM25FRENCHNOENG	French	0.3799	0.2139	0.8397	No
BM25FRENCHRERANK100	French	0.3657	0.1960	0.8437	Yes
BM25FRENCHDOCEXPANSION	French	0.3650	0.2014	0.8176	No
BM25FRENCHSPAM	French	0.3623	0.2067	0.7648	Yes
BM25FRENCHQUERYEXPANSION	French	0.3567	0.1941	0.8202	No
BM25TRANSLATEDQUERIES	Both	0.3037	0.1523	0.7437	Yes
BaseSystem (base English system)	English	0.2944	0.1505	0.7022	No

translation is, and the different results between the two denote how using a different translator may substantially influence output. This difference in results confirms how translation adds significant noise that interferes with the system IR-wise improvement process.

The *BM25FRENCHRERANK100* system instead doesn't work as expected. In particular we were expecting to increase the nDCG by reranking the first 100 documents but this metric actually decreases. This can be due to the fact that by recreating the index based only on the top-100 documents the terms statistic changes and the new statistics favors the documents that are not considered relevant in the ground-truth.

To implement the reranking we thought about using some machine learning and/or deep learning techniques (Learn To Rank, LTR) but we were worried about the fact that since we had only a small number of relevance feedback for the training topics, then the resulting system would overfit on the training data.

We chose the systems to be submitted to CLEF based on the previous consideration and by evaluating also the trade-off between performance and novelty we brought (in the last column of Table 7 we marked which systems were submitted).

Figure 4 shows the interpolated precision-recall curve for the two best performing systems.

4.2. Performance on Test Data

Table 8 reports the results computed for our CLEF-submitted systems over the LongEval test datasets. Each metric group represents a different indexing scenario, whether on same-time (of training data), short term evolution or long term evolution.

The results in Table 8 highlight how the time-wise evolution of corpora (and queries) did not affect the system's effectiveness in any relevant way. This can be traced back to two main reasons:

- The systems rely on language-specific components only for a minimal part. Moreover, such components are not largely affected by language change over time: unless way longer timespans are observed, stopwords are unlikely to change and filters working on syntax and morphology are expected to keep their behavior.
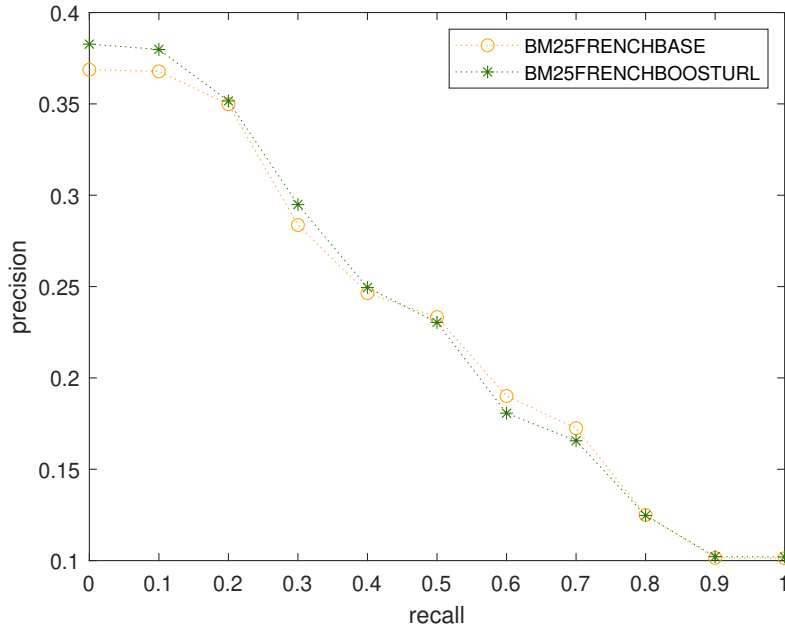
Figure 4: Interpolated precision-recall curve

Table 8
Test-time systems performance overview (nDCG)

System name	Within Time		Short Term		Long Term	
	nDCG	Recall@1000	nDCG	Recall@1000	nDCG	Recall@1000
BM25FRENCHBOOSTURL	**0.3859**	**0.8708**	**0.3866**	**0.8323**	**0.3945**	**0.8556**
BM25FRENCHBASE	0.3843	0.8684	0.3924	0.8375	0.3916	0.8531
BM25FRENCHRERANK100	0.3755	0.8644	0.3756	0.8365	0.3758	0.8531
BM25FRENCHSPAM	0.3605	0.7874	0.368	0.7686	0.3643	0.7773
BM25TRANSLATEDQUERIES	0.3072	0.7317	0.3051	0.7225	0.3189	0.7482

- At design time, during error analysis, changes were implemented targeting frequent problems that affected documents of various nature and content. Problems affecting specific categories of documents with specific content were not considered of primary interest as a means to avoid overfitting when given potentially different content to index.

4.3. Hypothesis Testing & Failure Analysis

To compare the submitted systems for each of the three case studies (withing time, short term and long term) we first plotted the box plots to have a visual representation of the systems, their mean and their variance. After that, we computed the two-way *ANalysis Of VAriance (ANOVA)* considering as hypothesis the null hypothesis $H_0 : \mu_x = \mu_y$ (where x and y represent two
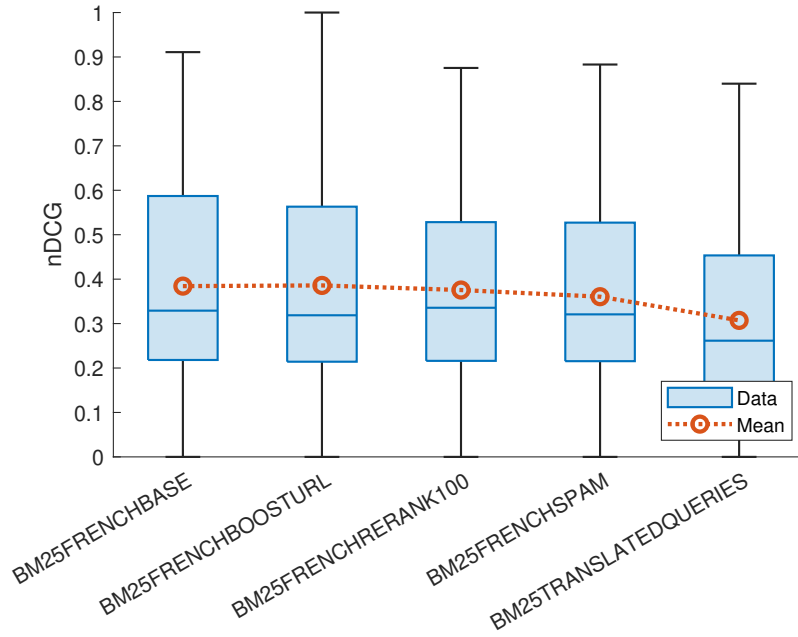
Figure 5: Box Plots Within Time

generic systems to be compared). The reason why we considered two-way ANOVA and not one-way ANOVA is that two-way ANOVA considers both the topics and the systems effect while the one-way ANOVA considers only the systems effect. The threshold that we used to consider the systems different was $\alpha = 0.05$ (5%). The results of the various ANOVA tests have been reported in Table 9, Table 10 and Table 11, where with SS we mean "Sum of Squares", with df we mean "degrees of freedom", with MS we mean "Mean Squares", with F we indicate the F-Test value and with prob>F we indicate the P-Value.

Eventually, we computed the Tukey HSD Test to understand which of the systems were actually different (in Figure 6, Figure 8 and Figure 10 the system highlighted in blue is the one considered for the comparison, the systems in red are the systems that are considered different, while the systems in grey are the systems that are considered similar). In all cases, the metric selected for the comparison has been the nDCG.

4.3.1. Within Time

From the Box plots reported in Figure 5 we can see that the systems, in this case, appear to be quite similar. This is not confirmed by the output of ANOVA reported in Table 9 since the P-Value result to be lower than the considered threshold. Nonetheless, the results can be considered reliable since the system contribution is much higher than the error contribution. The ANOVA results highlight significant differences between the systems we have analyzed. This difference may be mainly due to the presence of an English-based system (*BM25TRANSLATEDQUERIES*) along French-based ones.
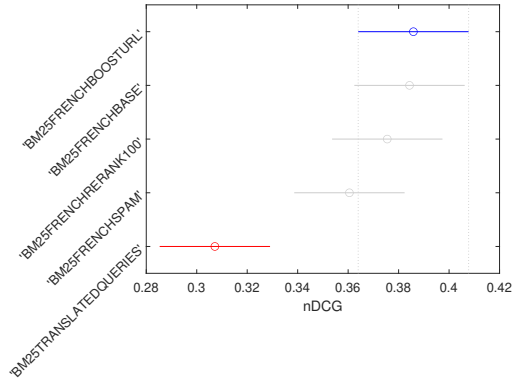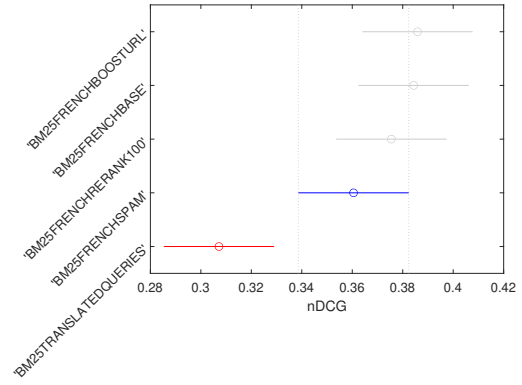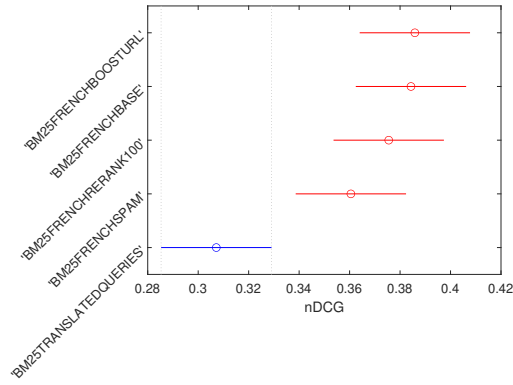
(a) BM25FRENCHBOOSTURL
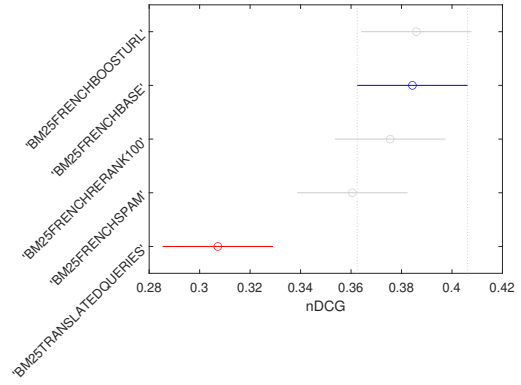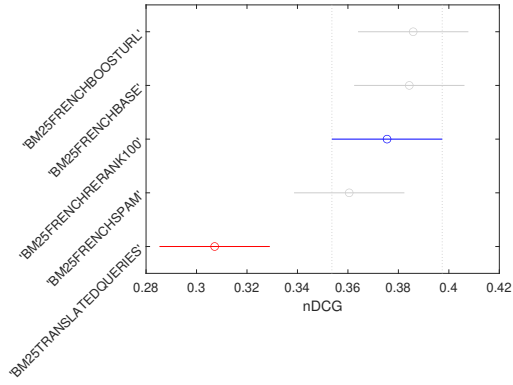
(b) BM25FRENCHBASE

(c) BM25FRENCHRERANK100
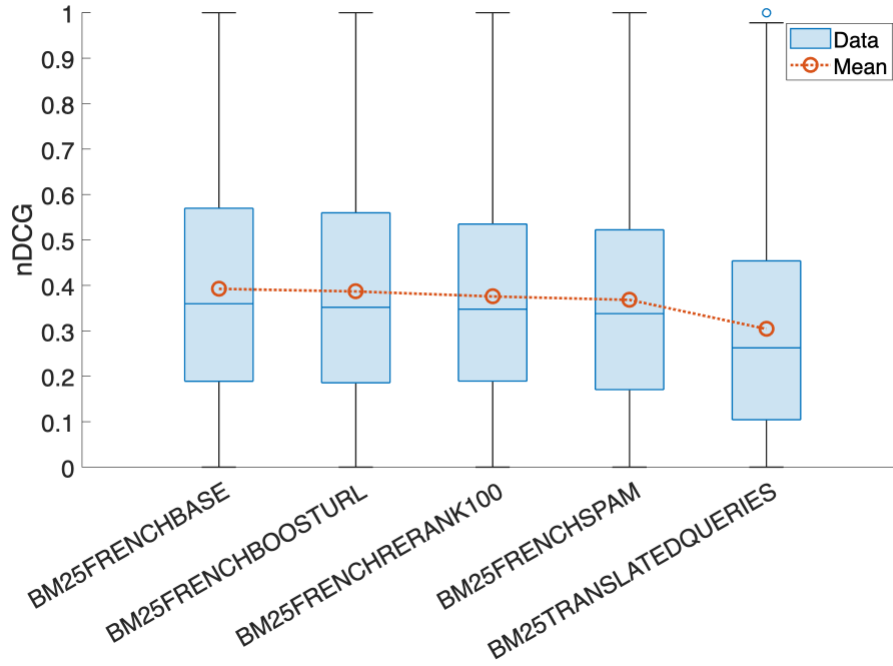

(d) BM25FRENCHSPAM

(e) BM25TRANSLATEDQUERIES

Figure 6: Tukey HSD Test Within Time

Table 9
Two-Way ANOVA Within Time (based on nDCG)

Source	SS	df	MS	F	prob>F
Columns (systems)	0.4170	4	0.1042	8.2753	2.0570e-06
Rows (topics)	21.6586	97	0.2233	17.7255	1.1527e-97
Error	4.8876	388	0.0126		
Total	26.9631	489			

Figure 7: Box Plots Short Term

Moreover, by looking at the Tukey HSD results (reported in Figure 6) the only system that appears different from the others is the *BM25TRANSLATEDQUERIES* system. This could be due to the fact that that system is the only one using the English corpus and the performance results to be quite different from the others.

We can conclude that the systems are quite similar but the *BM25TRANSLATEDQUERIES* system turns out to be the only one showing relevant differences.

4.3.2. Short Term

By looking at the box plot reported in Figure 7 we see that the systems have pretty much the same performance, only *BM25TRANSLATEDQUERIES* differs a bit from the others. Furthermore we can see that the extension of the whiskers in almost all the systems goes from 0 to 1, this is

Table 10
Two-Way ANOVA Short Time (based on nDCG)

Source	SS	df	MS	F	prob>F
Columns (systems)	4.4582	4	1.1145	87.3582	7.1783e-71
Rows (topics)	231.3659	881	0.2626	20.5839	0
Error	44.9605	3524	0.0128		
Total	280.7846	4409			

due to the fact that those systems performs very well for some queries (and reach the maximum value of nDCG) but very poorly for some other queries. A deeper analysis of the results revealed that the queries performing well were very specific while the ones performing badly were either or generic queries or containing terms that can be considered spam (see Section 2.2.1 for more details).

The two-way ANOVA (whose results are reported in Table 10) does reveal relevant differences among the systems since the P-Value result to be lower than the considered threshold. Furthermore, the results of ANOVA can be considered reliable since the system contribution is much higher than the error contribution.

By looking at the Tukey HSD results (reported in Figure 8) we noticed that there is a clear difference between *BM25TRANSLATEDQUERIES* and the other systems, this is due to the fact that this system is based on the English collection. Taken the best system's test (*BM25FRENCHBASE*'s), we see that the *BM25FRENCHBASE* system is significantly different from all systems other than the *BM25FRENCHBOOSTURL* system, this happens because they are basically the same except that the *BM25FRENCHBOOSTURL* system uses an additional field: the URL.

4.3.3. Long Term

From the Box plots reported in Figure 9 we can see that the systems, even in this case, appear to be quite similar. Furthermore we can see that the extension of the whiskers in almost all the systems goes from 0 to 1, this is due to the fact that those systems performs very well for some queries (and reach the maximum value of nDCG) but very poorly for some other queries. A deeper analysis of the results revealed that the queries performing well were very specific while the ones performing badly were either or generic queries or containing terms that can be considered spam (see Section 2.2.1 for more details).

The two-way ANOVA (whose results are reported in Table 11) does reveal relevant differences among the systems since the P-Value result to be lower than the considered threshold. Furthermore, the results of ANOVA can be considered reliable since the system contribution is much higher than the error contribution.

By looking at the Tukey HSD results (reported in Figure 10) we noticed that the two best performing systems (*BM25FRENCHBOOSTURL* and *BM25FRENCHBASE*) happen to be quite similar between each other but different from the other systems; this happens because the *BM25FRENCHBOOSTURL* system is actually the same as the *BM25FRENCHBASE* system except that is uses one additional field: the URL. Furthermore, also the *BM25FRENCHRERANK100* system and the *BM25FRENCHSPAM* turn out to be similar while the *BM25TRANSLATEDQUERIES* system
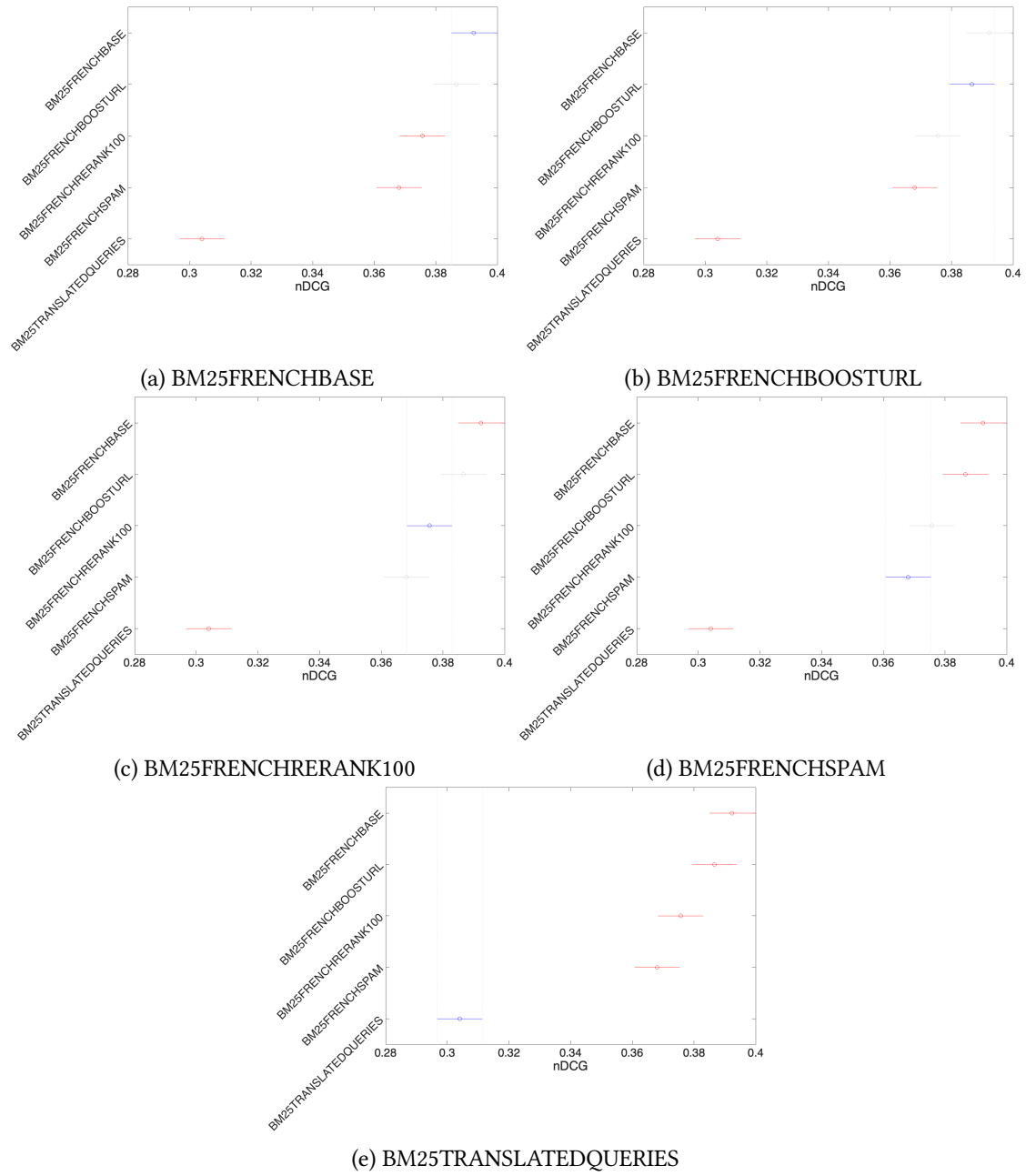
(a) BM25FRENCHBASE

(b) BM25FRENCHBOOSTURL

(c) BM25FRENCHRERANK100

(d) BM25FRENCHSPAM

(e) BM25TRANSLATEDQUERIES

Figure 8: Tukey HSD Test Short Term

still looks different and (as in the previous case studies) this is due to the fact that this last system is based on the English collection.
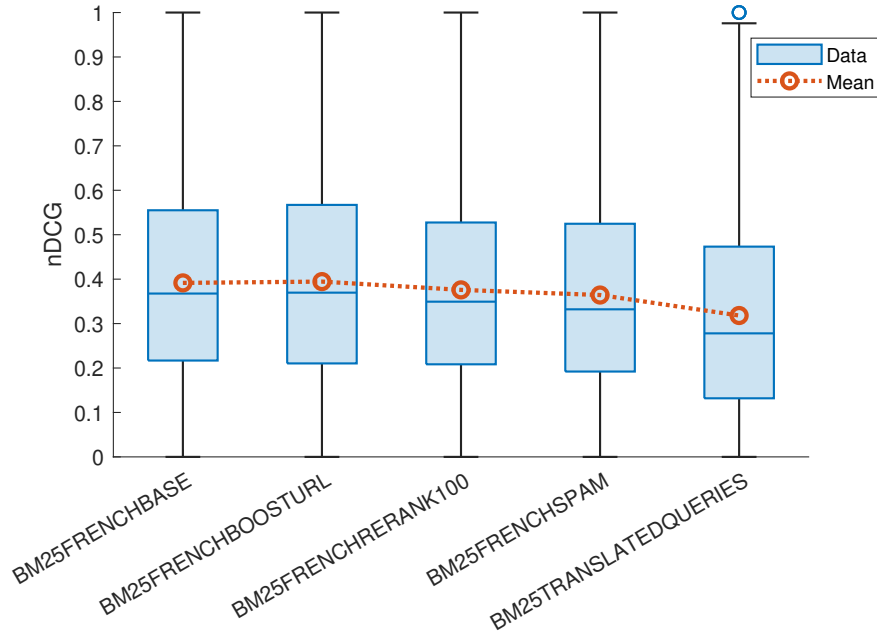
Figure 9: Box Plots Long Term

Table 11
Two-Way ANOVA Long Time (based on nDCG)

Source	SS	df	MS	F	prob>F
Columns (systems)	3.5172	4	0.8793	79.2206	1.4254e-64
Rows (topics)	218.3013	992	0.2368	21.3319	0
Error	40.9342	3688	0.0111		
Total	262.7527	4614			

5. Conclusions and Future Work

We have been able to reach good results in terms of nDCG and in particular of recall, in particular with the system *BM25FRENCHBOOSTURL*. We have developed a variety of system that use multiple different techniques to retrieve the documents always by paying attention to not create systems that are too overfitted to our training data (especially because the goal of the CLEF LongEval task was to evaluate the stability of the systems over time).

The English translations of the original French documents and queries were done by machine translators and their quality was not particularly good. Our translation tool demonstrated that machine translation could have a not negligible impact on IR.
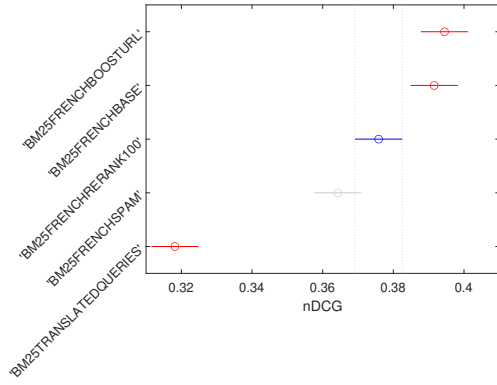
For what concerns spam detection, after these experiments we can assert that the models we have experimented with are indeed too simple to adequately distinguish between spam and non-spam documents. For instance, a more complex algorithm that takes into account
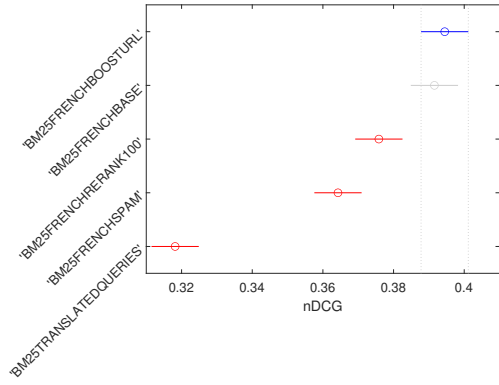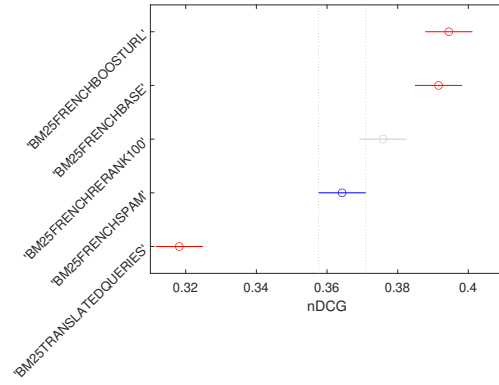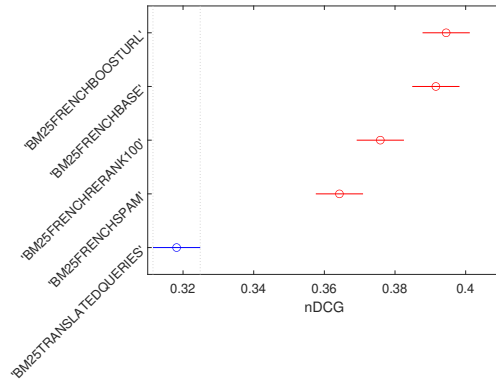
(a) BM25FRENCHBOOSTURL
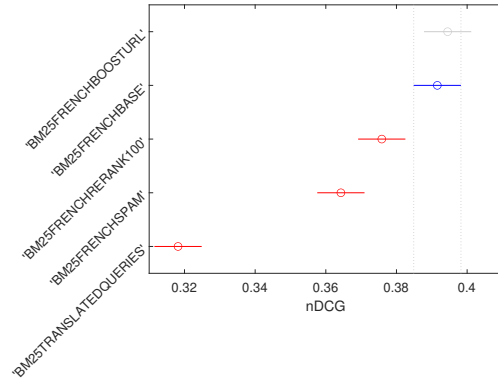
(b) BM25FRENCHBASE

(c) BM25FRENCHRERANK100

(d) BM25FRENCHSPAM

(e) BM25TRANSLATEDQUERIES

Figure 10: Tukey HSD Test Long Term

document length, word frequency distribution, word variance, sentence shape and language simultaneously may be able to target spam way more effectively. Moreover, with such an implementation, machine learning techniques may help in finding the optimal thresholds and weights for each document feature.

5.1. Future work

To improve our systems we will work on these aspects:

- **Document translation**: try to improve the documents translations by improving our translation tool (and turning it in something that allow us to execute the translation of the documents BODY field in a computationally feasible time).
- **System combination**: develop further the systems that use the English language, trying to increase the performance. After that combine the English and French systems results trying to do a multilingual search.
- **Learn To Rank (LTR)**: try to implement some LTR (and feature extraction) technique considering more relevance feedback to avoid overfitting.
- **Machine learning for spam detection**: look for correlation between various document features to infer with limited uncertainty whether a document contains spam.

References

[1] lucene.apache.org, Lucene, 1999. URL: https://lucene.apache.org.

[2] S. Rose, D. Engel, N. Cramer, W. Cowley, Automatic Keyword Extraction from Individual Documents, 2010, pp. 1 – 20. doi:10.1002/9780470689646.ch1.

[3] J. Savoy, Light stemming approaches for the french, portuguese, german and hungarian languages, in: Proceedings of the 2006 ACM Symposium on Applied Computing, SAC '06, Association for Computing Machinery, New York, NY, USA, 2006, p. 1031–1035. URL: https://doi.org/10.1145/1141277.1141523. doi:10.1145/1141277.1141523.

[4] E. A. Fox, J. A. Shaw, Combination of multiple searches, in: D. K. Harman (Ed.), Proceedings of The Second Text REtrieval Conference, TREC 1993, Gaithersburg, Maryland, USA, August 31 - September 2, 1993, volume 500-215 of *NIST Special Publication*, National Institute of Standards and Technology (NIST), 1993, pp. 243–252. URL: http://trec.nist.gov/pubs/trec2/papers/ps/vpi.ps.