SEUPD@CLEF: Team GWCA on Longitudinal Evaluation of IR Systems by Using Query Expansion and Learning To Rank

Notebook for the LongEval Lab at CLEF 2023

Leonardo Bellin[1], Antonino Andrea Carè[1], Marco Martini[1], Maria Teresa Pepaj[1], Matteo Salvalaio[1], Andrea Segala[1], Mariafiore Tognon[1] and Nicola Ferro[1]

[1]*University of Padua, Italy*

Abstract

This paper presents the results and work done by team GWCA (University of Padua) on LongEval CLEF 2023 Lab in Task 1 (retrieval), that aims at evaluating the temporal persistence of information retrieval systems.
After a careful analysis on the dataset given to train the system, the group decided to first try some common techniques to improve performance and then focus on those that produced better experimental results.

Keywords

LongEval 2023, Information Retrieval, Temporal Persistence, GPT query expansion

1. Introduction

This report aims at providing a brief explanation of the Information Retrieval system built for Task 1 of LongEval lab 2023: LongEval Retrieval[1].
The task focuses on evaluating the temporal persistence of information retrieval systems and the goal is to propose a system which can handle changes over time.
The task relies on a large set of data (corpus of pages, queries, user interaction) provided by a commercial search engine (Qwant[2]) and it is designed to reflect the changes of the Web across time, by providing evolving document and query sets.
The queries in the collection were collected from Qwant's users over several months and can thus be expected to reflect the changes in the search preferences of the users.
The main idea of the GWCA group was to initially focus the work on the creation of an effective

CLEF 2023: Conference and Labs of the Evaluation Forum, September 18–21, 2023, Thessaloniki, Greece
leonardo.bellin@studenti.unipd.it (L. Bellin); antoninoandrea.care@studenti.unipd.it (A. A. Carè); marco.martini.7@studenti.unipd.it (M. Martini); mariateresa.pepaj@studenti.unipd.it (M. T. Pepaj); matteo.salvalaio@studenti.unipd.it (M. Salvalaio); andrea.segala.3@studenti.unipd.it (A. Segala); mariafiore.tognon@studenti.unipd.it (M. Tognon); ferro@dei.unipd.it (N. Ferro)
http://www.dei.unipd.it/~ferro/ (N. Ferro)
0000-0001-9219-6239 (N. Ferro)

[1]https://clef-longeval.github.io/tasks/
[2]https://www.qwant.com

IR system and subsequently to support it with improvements aimed at increasing the system's performance and making it persistent as the document collection evolves. In particular, for this last objective, the group focused on *Query Expansion (QE)* techniques based on the use of GPT3.5 Turbo[3] (hereafter only GPT or GPT3.5). The idea is that as web pages evolve over time based on the evolution of language and trends, GPT3.5 is a quite up-to-date model that dates back to 2021, and is hence equipped to deal with current language peculiar expressions and phrases.
The paper is organized as follows: Section 3 describes the approach; Section 4 explains the experimental setup; Section 5 discusses the main findings and results; finally, Section 6 draws some conclusions and outlooks for future work.

2. Related Work

For the implementation of the system, the group relied on a basic structure provided by Professor Ferro during the aforementioned Search Engines course[4]. The project contains basic tools for parsing, indexing and searching that were extensively used in the development. Additionally, it must be pointed out that inspiration for QE with Word2vec was drawn from classes developed by one of the previous years' groups from Search Engines course, specifically Team 6musk[5] [2].

3. Methodology

3.1. Workflow

Once a functioning pipeline was built, with respect to this year's task the main focus became the creation of analysis tools suitable for the LongEval WebSearch French collection, which constituted a basis to integrate effectively QE techniques into the system to improve performance. Specifically, queries were expanded either with synonyms provided by GPT or with word embeddings generated by model Word2vec. Furthermore, the usage of proximity queries as possible expansions was explored and LambdaMART re-ranking algorithm was integrated into the system.
Finally, it is worth mentioning that, given the size of the corpus, multi-thread options were developed to speed up both the indexing and the searching process, achieving a better efficiency. In the following, the reasoning behind the techniques present in the final system is described, as well as the theoretical foundations behind the implemented solutions. Details about the implementation can be found at Section 3.2.

3.1.1. Base System

The basic pipeline has the following three components: a parser, an indexer and a searcher.
As for the parsing, the choice fell upon the provided JSON format for the documents as it is a much more streamlined format than Trec and several libraries are available to manage such

[3] https://platform.openai.com/docs/models
[4] https://bitbucket.org/frrncl/se-unipd/src/master/hello-tipster/
[5] https://bitbucket.org/upd-dei-stud-prj/seupd2122-6musk

documents. As for the indexing phase, instead, documents' bodies and lengths were stored for each document in the collection: specifically, the latter parameter is exploited in the re-ranking phase. Furthermore, since most of the work was thread-safe, a multi-threading approach was implemented, which resulted in much more efficient indexing, as reported in Section 5. Finally, the search phase is carried out in a multi-threading implementation, once again to pursue the efficiency of the IR system.

3.1.2. Analysis and choice of the French collection

Since some of the query translations from French to English were imprecise (see, for example, q06229908: "chateaux a vendre" translated into "knives for sale") and given that the better-performing English analyzer among the explored options was out-performed by the French one, most of the experiments were carried out on the French collection only. The main goal was to build tools for analysis that could deal with the specificity of the French language, such as frequent elisions and diacritical marks. Specifically, by observing the queries, it was possible to notice that Qwant users often wrote their queries without the proper accent marks and that this could affect precision if not correctly managed. Several off-the-shelves tools were tried on the data, as well as a custom made stoplist with words gathered among the most frequent of the French dataset.

3.1.3. Query Expansion

As an attempt to improve performance, given the shortness of the provided web queries, different query expansion techniques [3] were tried out. A brief explanation of the explored approaches follows:

- **Proximity Queries**: given queries were enriched with phrase queries in hope to boost precision. To allow for more flexibility, especially for longer queries, they were turned into proximity queries, with a slop (i.e. an edit distance) heuristically determined on the dataset and dependent on the length of the original query. Such phrases were used as additional boolean clauses, without replacing the single-term clauses derived from the parsing of the original query.
- **GPT generated expansions**: the queries were provided to the neural generative model GPT asking for related words, instead of exact synonyms *per se*, as it was an approach proven to be effective even with earlier models [4]. The model was tuned with parameters from [4], while the number of words added to the queries was heuristically determined. To prevent drifting, the weight of GPT terms was lower than those of the words from the original query.
- **Word2vec word embeddings**: Word2vec neural model [5] was used to to produce semantically similar words for the terms contained in the queries. Such model was trained over 20% of the documents of the collection and it retrieves 4 synonyms per word, provided that they reach a certain similarity threshold (fixed at 60%). The aforementioned numerical values were all heuristically determined.

In an initial phase, Wordnet also considered as a method for query expansion: yet, it was possible to notice that the generated synonyms didn't quite capture the actual meaning of words in the queries they were inserted into. As a consequence, this approach was discarded.

3.1.4. Reranking

In order to push the relevant documents to the top of the ranking, the implementation of a reranking system was attempted after most of the relevant documents were retrieved.
The model used was the `LambdaMART` model, which is a combination of `LambdaRank` and `MART` (Multiple Additive Regression Trees). The difference with the simple MART algorithm which uses gradient boosted decision trees is that `LambdaMART` introduces a cost function to order any ranking situation[6]. Other re-ranking models were also considered, but the choice fell on `LambdaMART`, as its effectiveness in reranking queries taken from query logs of a commercial search engine, just like Qwant, was proven during 2010 Yahoo! Learning To Rank Challenge [6].
In this case, re-ranking models were produced using a 2-fold or a 3-fold approach on the given train `qrels`, with increased performance (see Section 5) but a concrete risk of overfitting.

3.2. Implementation

This section covers the implementation of the main classes developed in the project, together with those that were modified the most with respect to the aforementioned Hello-Tipster baseline. The whole project can be found at the repository linked at Section 4.3. It should be highlighted that execution is delegated to class `Main.java`, whereas project configuration and parameters are managed by `ConfigManager.java`.

3.2.1. JSONLongEvalParser

The Abstract class `DocumentParser` was extended and adapted to JSON.
To accomplish this, the jackson.core[7] and jackson.databind[8] libraries, which are available on GitHub, were extensively used.
With the jackson.core library, a JsonParser using `JSONFactory` was built and the first token, which is expected to be an array was examined. The `jackson.databind` library was then used to put the token into a `JsonNode` and an iterator was utilized to cycle through the elements of the Node.
The Document Creation is handled by the class ParsedDocument. The documents present only two fields given the structure of the corpora:

- `ParsedDocument.FIELDS.ID` which identifies uniquely the document
- `ParsedDocument.FIELDS.BODY` the entire document text

This approach also improved up to 10% the performances reducing indexing time.

[6] https://www.educative.io/answers/what-is-lambdamart
[7] https://github.com/FasterXML/jackson-core
[8] https://github.com/FasterXML/jackson-databind

3.2.2. LongEvalFrenchAnalyzer

This class fully implements the analyzer of the IR system.
It filters the tokens with `ICUFoldingFilter`, which makes French phrases and their encoding more uniform. Afterwards the tokens go through a `StopFilter`, which loads an extended version of the stoplist from Oracle[9]. The extended stoplist contains the most frequent terms of the index found using `VocabularyStatistics`. Lastly, the tokens go through the `FrenchLightStemFilter`, which was found to be the best performing among other stemmers. In an initial phase of the development the group built a `LowerCaseBrandFilter` which lowercased the tokens unless they were in "`brands.txt`" file, which was taken from GitHub[10], and then modified so that each brand has no blank spaces (i.e. it is a single token). However, since queries contained names of companies directly in lowercase letters, this approach was dismissed.
This version of the analyzer was ultimately selected as the best after trying several stop lists and three different French stemmers: `FrenchStemFilter`, as well as `FrenchMinimalStemFilter` and `FrenchLightStemFilter` from `org.apache.lucene.analysis.fr`.
Once the chosen analyzer was finalized, additional techniques were explored.

3.2.3. Vocabulary Statistics

To verify the validity of the index and to be able to analyze all the fields, the well-known Luke (Lucene Index Toolbox) tool was initially used, but later, for practical reasons,
a `VocabularyStatistics` class was created, so as to obtain a file of text containing the terms contained in the index and some statistics relating to them. In particular, the class has a method for printing on a text file "`vocabulary.txt`" a dictionary containing all the terms of the index sorted in descending order according to their frequency. Thanks to the analysis of this file, an initial stop list "`oracleFrench.txt`" [11] was integrated by manually adding some very frequent terms, with the idea of reducing the size of the index and slightly improving performance. Later the class was modified so that it could automatically append the most frequent terms to the stop list, but it was found that the previous version of "`oracleFrench.txt`" was already very effective and that further extensions were counterproductive.

3.2.4. Directory Indexer

This class is designed to index each individual document in a corpus after applying an analyzer. The implementation closely follows the one of the course Professor's one, but a few additions were made.
To improve efficiency, parallelization was integrated into this class. During the trial and

[9] https://docs.oracle.com/en/database/oracle/oracle-database/21/ccref/oracle-text-supplied-stoplists.html#GUID-2F4889F1-B3A0-4639-95C6-FFD9C5276245
[10] https://github.com/anna-hope/brandnames/blob/master/brands.txt
[11] https://docs.oracle.com/en/database/oracle/oracle-database/21/ccref/oracle-text-supplied-stoplists.html#GUID-83740606-D05A-49F0-A6C3-0F2290B7DAB0

error process, it was found that the indexing of documents was the most computationally intensive task, especially when dealing with such a large corpus. By using a fixed thread pool `ExecutorService` from the `java.util.concurrent` library package, multithreading solutions were implemented to run tasks asynchronously without exceeding the specified number of threads.
The parsed documents are still processed by the main thread, so assuming one has enough threads to handle the volume of documents, the main bottleneck will be caused by the storage speed.
Overall, this implementation of the class has improved the indexing process, making it faster and more efficient, while still maintaining thread-safety and ensuring the accuracy of the results.

3.2.5. GPT Query Expander

One of the approached techniques to expand queries was the usage of GPT3.5 Turbo. The main goal was to ask the model for a list of words related to the query and to create a .tsv file containing all the query expansions for time efficiency reasons.
In a first attempt, the IR system was fed with both queries and expansions without performing any other processing. However, this resulted in a noticeable drop in performance, with *Mean Average Precision (MAP)* being more than halved. The benefits of Query Expansion became visible only after combination with proximity queries and suitable boosting.
The class `GPTQueryExpander` was implemented in both Java using the `openai-java` library provided by Theo Kanning[12] and Python, but socket timeout errors were experienced in the Java implementation. This led the group to switch to the OpenAI official API[13] in Python, which provided the same results but without any connection problems.
As mentioned before, some model parameters needed to be set: specifically, top_p=0.95, temperature=0.75 and max_tokens=300. The meaning of these parameters is reported below:

- top_p: regulates the diversity of the text produced by the model: higher values lead to results that are similar to the text sequences on which the model has been trained.
- temperature: controls the creativity of the model: higher values lead to more unpredictable results. Topic drift can be observed with really high values.
- max_tokens: GPT was asked for 10 related words, so to be sure they were included a value of 300 was set.

The results provided by GPT were inseted into .tsv files because their format provides simpler processing. The chosen pipeline for query expansion involved reading a query from the query file, asking the model for the expansion returning a numbered list, gathering the contents of the list items into one single-line string, and writing only the expansion in the output file. Characters that may cause problems, such as \n, \r and double quotes, were also removed. Additionally, regular expressions were used to remove numbers and separate the contents of the query expansions with spaces.

[12] https://github.com/TheoKanning/openai-java
[13] https://platform.openai.com/docs/api-reference

3.2.6. Word2Vec Filter

To achieve Word2vec embedding, the `deeplearning4j` library was used and a custom filter that extends the TokenFilter class in Lucene was created.
The Word2VecFilter takes a token and retrieves the four nearest synonyms based on the trained model, with a threshold of 0.60 for the vector similarity.
Furthermore, a duplicate check was added to avoid expanding duplicate terms in a query. The expanded terms are given an increment position of 0, so they are considered as "or" clauses in a Boolean Query.
To train the model, 20% of the corpus was used, but the 20 most frequent terms, as well as stopwords, were removed. The resulting model was designed to have 512 features for the vector representation of the synonyms and a window size of 10 words, as Word2vec models are trained on a window context.

3.2.7. ReRanker

A Learning To Rank system has been implemented via the RankLib-2.18[14] library, which generates a model from a training set that can be used to predict scores for unlabeled data.
The model was trained on pairs of document and query IDs, which were retrieved during the searching phase on the training data, along with their relevance assessment from the qrels serving as labels.
The model was trained using both 3-fold and 2-fold cross-validation techniques on all the retrieved pairs.
In order to train the model, the features that were extracted for each document were: the BM25 score and the length of the document in terms of character.
To train the model the following command was used:

```
-train path\_to\_train/train.txt -ranker 6 -kcv 5 \
-kcvmd path\_models/ -kcvmn ca \
-metric2t NDCG@10 -metric2T NDCG@10
```

where:

- `train` is the path to the training set which has the following format:

label	Query ID	BM25 score	document length...	comments
0	qid:q06223196	1:6.693	2:2929 ...	#doc062200206319
0	qid:q06223196	1:6.6602736	2:1582 ...	#doc062201708464
...	...	...	...	...

  in the comment section the document ID was added in order to reconstruct the final run at the end
- `ranker` is the type of algorithm to use, 6 is the code for LambdaMART

[14] https://sourceforge.net/p/lemur/wiki/RankLib/

- `kvc` number of folds for kfold cross validation
- `metric2t` the function used to train the model in this case was NDCG at 10
- `metric2T` the function used to measure the performance on the test set again NDCG at 10

at the end the command will save k number of folds different models. To use the model on the unlabeled data the following command was used:
`-load mymodel.ca -rank run\_with\_features.txt -score predicted\_scores.txt`
where:

- `load` loads the model used
- `rank` means rank the specified set of unlabeled data with the same format as the training set but with fictitious labels
- `score` the file where the predicted scores will be saved.

The predicted scores are in the order of the given test set pairs, in the following format:

query ID	index	Score
qidq06223196	0	-0.8938
qidq06223196	1	-1.1028
...	...	...

so to produce the final run the ReRanker class has to match each predicted score to the query ID and Document ID and then order them in a decreasing way.

3.2.8. Searcher

The final implementation modifies the Hello-Tipster searcher to work in a multi-threaded approach and enriches the original query with different clauses (methods `shortQueryExpander` and `getPhraseQuery`). Each `BooleanQuery` searched through the index can be made up of the following three components:

- the original topic title, parsed with the analyzer. The clauses generated by this parsing were never removed in the runs.
- a Lucene `PhraseQuery`, added as a Boolean clause. As mentioned in Section 3.1.3, such query is obtained from the topic title and is provided with a slop. Empirically, the slop values that lead to optimality upon the training dataset were determined as 5 and 16 respectively for queries shorter and longer than a certain threshold, heuristically fixed at 3 words.
- query-related terms generated by GPT. Such expansions are wrapped inside a `BoostQuery` and are assigned a heuristic weight of 0.1585. The amount of terms added depends on the length of the original query and currently a query is expanded until it is made up of 9 words.

4. Experimental Setup

4.1. Collection

This section is dedicated to briefly describing the datasets used both for training and developing the systems and for testing them.
For the LongEval CLEF 2023 Lab, the experimental collection was provided by Qwant:

- Queries are extracted from Qwant's search logs, based on a set of selected topics. The initial set of extracted queries are filtered to exclude spam and queries returning a small number of documents. The query set was created in French and was automatically translated to English.
- The document collection includes the relevant documents that are selected to be retrieved for each query, by extracting all the documents displayed in SERPs for the queries that were selected. In addition to these documents, potentially non-relevant documents are randomly sampled from Qwant index in order to better represent the nature of a Web test collection.
- The relevance estimates are obtained through automatic collection of user implicit feedback through the usage of a click model, based on Dynamic Bayesian Networks trained on Qwant data. The output of the click model represents an attractiveness probability, which is turned into a 3-level scale score (0 = not relevant, 1 = relevant, 2 = highly relevant). To better contextualize some issues of this task it is necessary to make some observations on the relevance assessments. Assessing the relevance of a document through a click model is a biased estimation method. Indeed the probability that a web page is clicked is not strictly connected to its relevance, and this discrepancy has been noticed during a self-evaluation of the "`train.txt`" file containing the relevance estimates. For example, the document doc062200205332 containing the Wikipedia page of stepladder was judged not relevant for the query q062217010 'escabeau' (stepladder).

4.2. Evaluation Measures

To evaluate the systems, two main evaluation measures were used: MAP and *Normalized Discounted Cumulated Gain (nDCG)*:

- MAP was used since it provides a consistent and easily interpretable result, especially by comparing the Precision-Recall curves.
- pure nDCG scores calculated are used to evaluate a single run, this is because it is consistent with Web search, for which the discount emphasises the ordering of the top results.
- *Relative nDCG Drop (RnD)*, measured by computing the difference between nDCG on within a time heldout test data vs short- or long-term testing sets. This measure captures the goal of the campaign of evaluating the temporal persistence of the systems' performances by assessing the impact of the data changes on the systems' results.

This report will only contain MAP and nDCG measures for all the collections. In order to get results for the runs tools like `trec\_eval`[15] and Luke were used in order to easily retrieve information out of the index and check its integrity.

4.3. Repository

To develop the project, the GWCA group made use extensively of git in order to collaborate and organize the files. To facilitate reproducibility, a link to the git repository can be found here.

4.4. Hardware

The hardware used for the runs is:

- OS: GNU/Linux
- CPU: Intel i9 9900k 8c16t 4.7Ghz 16 cores
- RAM: 32GB 2133Mhz
- SSD: Crucial P5 512GB

Note that the hardware used significantly impacts the temporal performance of the systems, and in particular it was possible to notice that even a slight change in the SSD could create a bottleneck and make the indexing phase up to 5 times slower.

5. Results and Discussion

5.1. Runs Description

In this section a short description of each produced run is provided, together with a name that will be later used to show performance results. All the runs were performed with BM25Similarity and make use of the French collection, the filters described in Section 3.2 and stoplist "`oracleFrench.txt`" unless differently specified. Here is a legend of the terms used in the runID to characterize the runs:

- **english**: the run was carried out on the English collection
- **lovins**: the run used Lovins stemmer
- **smartstop**: the run used "`smart.txt`" stoplist for English
- **lightstem**: the run used `LightFrenchStemFilter`
- **stem**: the run used `FrenchStemFilter`
- **minstem**: the run used `FrenchMinimalStemFilter`
- **nostem**: the run didn't use stemming
- **word2vec**: the run used query expansion with Word2vec
- **phrase**: the run used phrase queries as described in Section 3.2.8
- **qexp**: the run used GPT query expansions with weights as described in Section 3.2.8
- **stopexp**: the run used an extended version of "`oracleFrench.txt`" enriched with frequent terms from the dataset
- **rerank2f**: used a reranking model trained with 2-fold cross-validation
- **rerank3f**: used a reranking model trained with 3-fold cross-validation.

[15] https://github.com/usnistgov/trec_eval

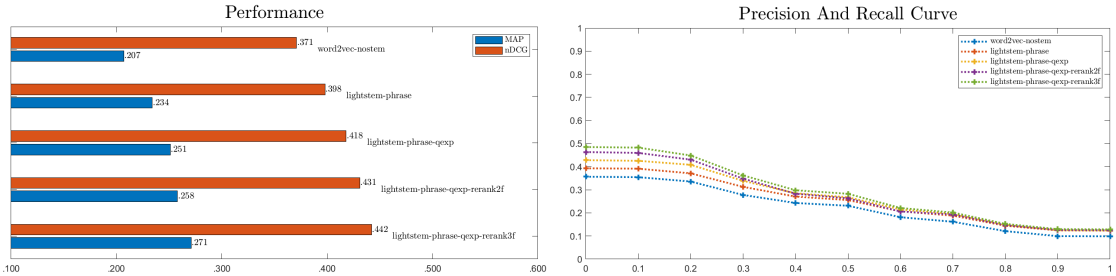
5.2. Results on the training set

In the table below results for some of the carried-out runs are reported: indeed, it needs to be highlighted than several more runs were needed to tune parameters and are hence omitted. As noticeable, performances of the French analyzer are by far superior to those obtained using the English collection. Furthermore, phrase queries seem to succeed at improving Precision and reach their best performances paired with GPT query expansion.
It is necessary to remark that the best results, obtained with the re-ranking algorithm, may be subject to overfitting, and the improvement is not to be expected to be as high on other tests. Furthermore, given the random split between train and validation data, results of the re-ranking process heavily depend on the folds created during the training phase.

The following table presents all the significant results gathered from some of the runs. In bold the names of the 5 submitted systems are highlighted, while bold results highlight the best performance throughout all the experiments, as well as the best without any reranking, since it possibly leads to overfitting.

Table 1
Performance of the runs

Run	MAP	nDCG	P@10	R@1000
seupd2223-gwca-lightstem-phrase-qexp-rerank3f	**0.2710**	**0.4420**	**0.1461**	**0.8698**
seupd2223-gwca-lightstem-phrase-qexp-rerank2f	0.2582	0.4312	0.1424	0.8698
seupd2223-gwca-lightstem-phrase-qexp	**0.2506**	**0.4183**	**0.1488**	**0.8698**
seupd2223-gwca-stem-phrase-qexp	0.2484	0.4159	0.1476	0.8674
seupd2223-gwca-minstem-phrase-qexp	0.2433	0.4096	0.1463	0.8605
seupd2223-gwca-lightstem-phrase-qexp-stopexp	0.2431	0.4091	0.1443	0.8543
seupd2223-gwca-lightstem-phrase	0.2340	0.3982	0.1430	0.8556
seupd2223-gwca-nostem-word2vec-rerank2f	0.2184	0.3885	0.1211	0.8313
seupd2223-gwca-lightstem	0.2150	0.3822	0.1348	0.8515
seupd2223-gwca-minstem	0.2131	0.3788	0.1335	0.8400
seupd2223-gwca-stem	0.2115	0.3782	0.1341	0.8469
seupd2223-gwca-word2vec-nostem	0.2070	0.3708	0.1296	0.8313
seupd2223-gwca-lightstem-qexp	0.2048	0.3758	0.1268	0.8595
seupd2223-gwca-lightstem-word2vec-rerank2f	0.1989	0.3705	0.1082	0.8444
seupd2223-gwca-word2vec-lightstem	0.1917	0.3593	0.1254	0.8444
seupd2223-gwca-english-lovins-smartstop	0.1568	0.2862	0.0942	0.6796
Baselines				
BM25 Terrier (fr) reranked by T5	0.2175	0.3650	0.1329	0.7695
BM25 Terrier (en) reranked by T5	0.2074	0.3582	0.1292	0.7695



(a) MAP and nDCG results

(b) Precision and Recall Curve

Figure 1: Graphical visualization for performance of the systems submitted to CLEF

5.2.1. Efficiency of multi-threaded system

Below information about the time improvement derived from multithreaded indexing and searching system is provided. Reported values are means over 5 executions of run seupd2223-gwca-lightstem-phrase-qexp.
It is noteworthy that these times were the ones obtained the setup described in Section 4, but trying the same procedure on another machine, with 20 cores, the time dropped further to 81s, suggesting that this procedure can be highly parallelized. As for Searching in this particular computer, the performance didn't improve at all as a single core was fast enough to process any queue on the fly.

Table 2
Time performance of the runs

	16 threads	**1 thread**
Indexing	96.6 s	469.8 s
Searching	8 s	7.4 s

5.3. Results on Heldout Data and Test Collections

In this section, results for the runs submitted to CLEF are provided: raw performance metrics are paired with statistical analysis so as to better compare the systems and the effectiveness of the different implemented strategies.
Specifically, names of the runs are based on the conventions defined in Section 5.1, whereas in boxplots and multiple comparisons each run is identified by a number from one to five, reported beside its name in Tables 3, 5 and 7.
It needs to be remarked that statistical testing (i.e., two-way ANOVA and Multiple Comparisons with Tukey's HSD test) was carried out only on nDCG, as the chosen evaluation measure for LongEval Lab, rather than on all the reported metrics.
Finally, all tests were performed with a significance level $\alpha = 0.05$.

Table 3
Performance of the runs of Heldout data

	Run	MAP	nDCG	P@10	R@1000
1	seupd2223-gwca-lightstem-phrase-qexp	0.2524	0.4294	0.1531	0.8908
3	seupd2223-gwca-lightstem-phrase	0.2303	0.4052	0.1418	0.9034
2	seupd2223-gwca-lightstem-phrase-qexp-rerank2f	0.2302	0.4059	0.1347	0.8908
4	seupd2223-gwca-lightstem-phrase-qexp-rerank3f	0.2099	0.3872	0.1378	0.8908
5	seupd2223-gwca-word2vec-nostem	0.2083	0.3843	0.1337	0.8720

5.3.1. Training Heldout Data

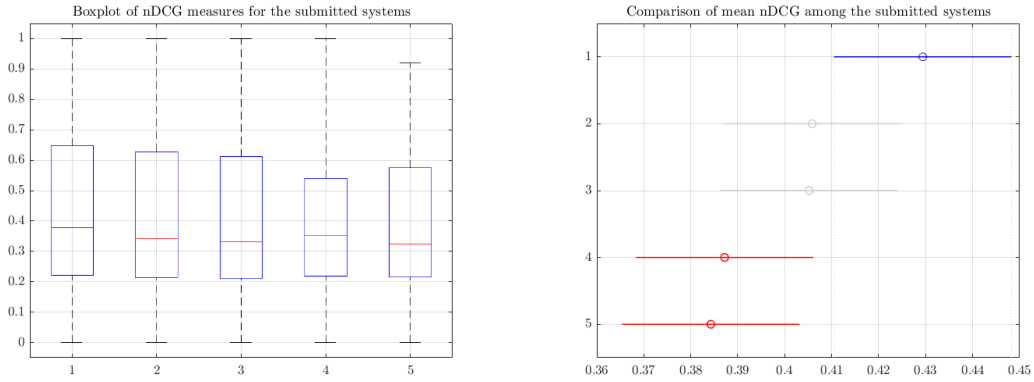
As far as the training heldout data are concerned, it is noticeable from Table 3 that the re-ranking system did not succeed in producing a general model, also suitable for non-assessed documents, as the run without re-ranking performs significantly better than the -seemingly overfit- 3-fold trained model.
The absence of outliers in the boxplot Figure 2a, as well as the width of whiskers, can possibly be interpreted as the reflection of the great variability in performance dependent on the specific topics, also assessed by the correspondent high value for MS in Table 4. Here, small p-values are evidence against the null hypothesis of the runs being all equivalent. This is noticeable from Figure 2b, where the best performing run is highlighted in blue and is statistically different from the red ones. Apparently, as for this dataset, GPT generated query expansions did not provide a statistically significant improvement in nDCG.

Table 4
Two-way ANOVA results for nDCG on Heldout data

Source	Sum of Squares	Degrees of freedom	Mean Squares	F	p-values
Systems	0.1282	4	0.0320	3.4136	0.0093
Topics	25.4205	97	0.2621	27.9137	1.2256e-127
Error	3.6427	388	0.0094	[]	[]
Total	29.1914	489	[]	[]	[]

5.3.2. Short-Term Persistence

As far as the Short-Term LongEval Test collection is concerned, it is noticeable from Table 5 that the pre-trained reranking models perform worse than all the other runs, in terms of MAP. Similarly to the results on Heldout data, F-values and p-values are respectively quite high and quite low, which acts as a further evidence against the null hypothesis. This is noticeable from Figure 3b, where the best performing run is highlighted in blue and is statistically different from the red ones. Once again, query expansion via GPT3.5 does not seem to add a statistically significant impact on performance, if compared to run 2, i.e. the one with phrase queries only.



(a) Boxplot for nDCG

(b) Multiple comparisons of systems' nDCG

Figure 2: Graphs obtained from statistical analysis of Heldout data

Table 5
Performance of the runs of Short-Term Test set

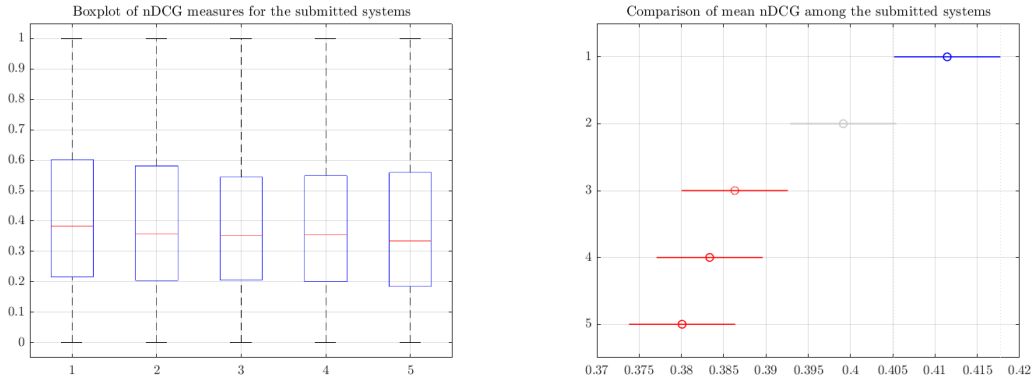
	Run	MAP	nDCG	P@10	R@1000
1	seupd2223-gwca-lightstem-phrase-qexp	0.2475	0.4114	0.1476	0.8543
2	seupd2223-gwca-lightstem-phrase	0.2375	0.3992	0.1437	0.8445
5	seupd2223-gwca-word2vec-nostem	0.2205	0.3801	0.1334	0.8315
3	seupd2223-gwca-lightstem-phrase-qexp-rerank3f	0.2160	0.3863	0.1325	0.8543
4	seupd2223-gwca-lightstem-phrase-qexp-rerank2f	0.2117	0.3833	0.1317	0.8543

Table 6
Two-way ANOVA results for nDCG on Short-Term LongEval Collection

Source	Sum of Squares	Degrees of freedom	Mean Squares	F	p-values
Systems	0.5982	4	0.1496	15.9982	5.4916e-13
Topics	244.0679	881	0.2770	29.6346	0
Error	32.9437	3524	0.0093	[]	[]
Total	277.6098	4409	[]	[]	[]

5.3.3. Long-Term Persistence

As for the Long-Term Persistence task, while the null hypothesis is once again discredited by the small p-values, differently from before, query expansion with GPT seems to significantly improve nDCG with respect to the baseline set by the run with phrase queries only, as visible in Figure 4b. Once again, Word2vec synonyms perform significantly worse in terms of nDCG than the two best runs and the pre-trained re-ranking model end up being harmful for performance.



(a) Boxplot for nDCG

(b) Multiple comparisons of systems' nDCG

Figure 3: Graphs obtained from statistical analysis of nDCG in LongEval Short-Term Test sub-collection

Table 7
Performance of the runs of Long-Term Test set

	Run	MAP	nDCG	P@10	R@1000
1	seupd2223-gwca-lightstem-phrase-qexp	0.2453	0.4161	0.1581	0.8640
2	seupd2223-gwca-lightstem-phrase	0.2297	0.3988	0.1547	0.8570
5	seupd2223-gwca-word2vec-nostem	0.2179	0.3844	0.1460	0.8372
3	seupd2223-gwca-lightstem-phrase-qexp-rerank3f	0.2168	0.3942	0.1469	0.8640
4	seupd2223-gwca-lightstem-phrase-qexp-rerank2f	0.2131	0.3905	0.1410	0.8640

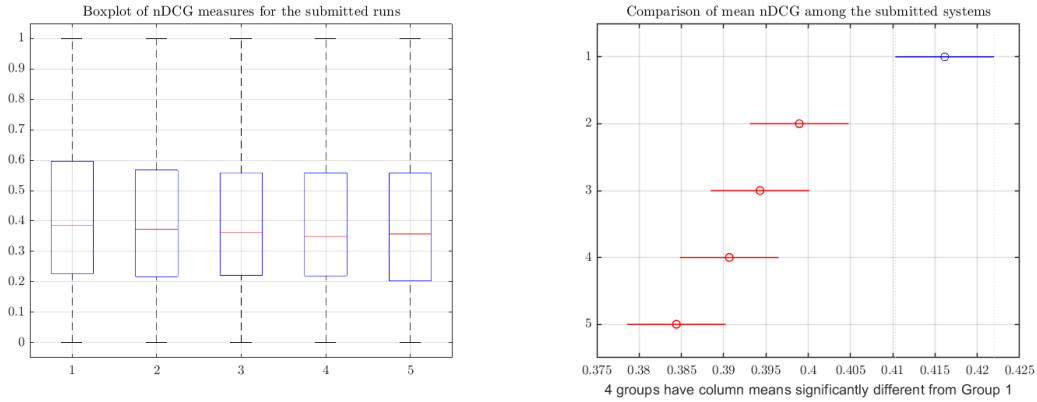
Table 8
Two-way ANOVA results for nDCG on Long-Term LongEval Test sub-collection

Source	Sum of Squares	Degrees of freedom	Mean Squares	F	p-values
Systems	0.5310	4	0.1328	15.7221	9.1886e-13
Topics	237.1325	921	0.2575	30.4909	0
Error	31.1087	3684	0.0084	[]	[]
Total	268.7722	4609	[]	[]	[]

6. Conclusions and Future Work

6.1. Conclusions

To summarize, results upon the training set seem to suggest that query expansion can actually improve performance, but only if paired with techniques that improve precision, such as proximity queries. This was further confirmed by results upon the test collections. Furthermore, among the explored sources to gather synonyms for QE, GPT3.5 Turbo seems to provide the best related words, while Word2vec doesn't perform as well, possibly because of the very noisy documents the corpus was made up of. Finally, it is worth mentioning that LambdaMART with the 2-fold cross validation appears to increase performances only on the dataset the model is



(a) Boxplot for nDCG

(b) Multiple comparisons of systems' nDCG

Figure 4: Graphs obtained from statistical analysis of nDCG in LongEval Long-Term Test sub-collection

trained upon, but possibly not as substantially as expected.

6.2. Future Work

In a possible future work, some more attention should be put into the Learning To Rank system, trying to use more features to better capture the relationship between relevance and documents. This could make the pre-trained model more general and suitable for several different collections. Indeed, in this work very few features were used, so the model turned out to be quite inaccurate in a general context.
A possible margin of improvement may come from a more careful tuning of parameters for GPT3.5, as here some of them were taken for granted as they were reported in Claveau [4].
Implementing a weighted score, incorporating standard scores such as BM25 and Dirichlet score and the one predicted by the re-ranking model, could also be explored: getting a fusion of different systems could indeed possibly mitigate the effect of reranking and overfitting.
Finally, different options for parameters of the Word2vec model, which may have some impact on performance, could be explored, but special attention should be dedicated to make the generated synonyms less noisy, which seemed to be the problem with this specific trained model.

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, Thessaloniki, Greece, 2023.

[2] L. Cappellotto, M. Lando, D. Lupu, M. Mariotto, R. Rosalen, N. Ferro, SEUPD@CLEF: Team 6musk on Argument Retrieval for Controversial Questions by Using Pairs Selection and Query Expansion, in: Conference and Labs of the Evaluation Forum https://ceur-ws.org/Vol-3180/paper-254.pdf, 2022.

[3] C. Carpineto, G. Romano, A Survey of Automatic Query Expansion in Information Retrieval, ACM Computing Surveys 44 (2012).

[4] V. Claveau, Query expansion with artificially generated texts, arXiv preprint arXiv:2012.08787 (2020).

[5] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, J. Dean, Distributed representations of words and phrases and their compositionality, in: NIPS'13: Proceedings of the 26th International Conference on Neural Information Processing Systems, volume 2, 2013.

[6] C. J. Burges, From RankNet to LambdaRank to LambdaMART: An Overview, Learning 11 (2010) 81.