SEUPD@CLEF: RAFJAM on Longitudinal Evaluation of Model Performance

Notebook for the LongEval Lab at CLEF 2023

Alvise Bolzonella[1], Riccardo Broetto[1], Marco Gasparini[1], Farhad Sadat[1] and Nicola Ferro[1]

[1]*University of Padua, Italy*

Abstract

This paper is intended to be a report of the work we have done for the CLEF 2023 LongEval Lab, whose main goal is to evaluate and improve performances of IR models along time.
We have implemented a basic retrieval system and then modified and extended it, focusing on different query expansion techniques, involving the use of synonyms and pseudo-relevance feedback.
We will provide a description of our ideas, code and other development details, along with statistical analysis of the runs of our systems on different test collections.

Keywords

Information retrieval, Lucene, query expansion

1. Introduction

The main reason behind LongEval @ CLEF task [1] about temporal persistence of information retrieval systems is that recent research shows that their performances get worse as test data becomes distant in terms of time from train data.

Therefore, participants should develop a model producing satisfying results on the different test sets provided by the organizers, which are collected in three different moments of summer 2022. Every collection consists of a set of documents coming from Qwant's index queries, and of a set of queries related to one or more topics; the choice of these topics is based on Web trends and it is diverse enough. Our group has decided to analyse collections in French, which is the original language of the data sets.

The paper is organized as follows: Section 3 describes our approach; Section 4 explains our experimental setup; Section 5 discusses our main findings; finally, Section 6 draws some conclusions and outlooks for future work.

CLEF 2023: Conference and Labs of the Evaluation Forum, September 18–21, 2023, Thessaloniki, Greece
alvise.bolzonella@studenti.unipd.it (A. Bolzonella); riccardo.broetto@studenti.unipd.it (R. Broetto); marco.gasparini.10@studenti.unipd.it (M. Gasparini); farhad.sadat@studenti.unipd.it (F. Sadat); ferro@dei.unipd.it (N. Ferro)
http://www.dei.unipd.it/~ferro/ (N. Ferro)
0000-0001-9219-6239 (N. Ferro)

2. Related Work

2.1. Project setup and basic retrieval system

We have decided to implement our retrieval system starting from the basic structure of the projects contained in the Search Engines course repository[1], used during our studies for learning the fundamentals about Information Retrieval.
The basic code for the Indexer, Searcher and Analyzer of the project is taken from the hello-tipster example. We have then improved our Analyzer focusing on the hello-analyzer example, in particular we have adopted the StopFilter class from there. We have also looked into FrenchAnalyzer class of Lucene[2], which has suggested us to add an ElisionFilter using its default list of French articles, and to stem words with FrenchLightStemFilter[3]. Finally, as similarity score, after trying different alternatives, we have decided to use *Best Matching 25 (BM25)* because it provides the best results.

2.2. Pseudo-relevance feedback

The general idea for this technique comes from the lectures of the Search Engines course and from some further readings [2]. The different term weighting schemas we have tried to use (see Section 3.2.1) and Zipf's law were presented during the lectures, too.

2.3. Synonym-substitution query expansion

The main idea for this method comes from many papers that discuss on the lexical query expansion [3]. In order to find the most relevant words to substitute with their synonyms, we asked to an artificial intelligence, chatGPT[4], to find the most frequent words in the query list and to provide a list of three synonyms for all this words.

3. Methodology

The procedure to obtain the final runs results consists basically of two main steps, which are the indexing of the documents in the collection, involving parsing operations and using a customised analyser, and the proper searching phase, preceded by parsing and analysis of the queries. The following sections describe our proposed solutions and implementations; they are based on Apache Lucene[5], which is a very popular API for IR. It consists of a lot of packages for various tasks, including analysis of texts in different languages, including French[6].

[1]UniPD Search Engines repository: (https://bitbucket.org/frrncl/se-unipd/src/master/)
[2]Documentation: org.apache.lucene.analysis.fr.FrenchAnalyzer
[3]A class of the org.apache.lucene.analysis.fr library
[4]OpenAI, "ChatGPT", knowledge cutoff: september 2021, avaiable online on https://openai.com
[5]T. A. S. Foundation, Apache lucene, 2022. (org.apache.lucene)
[6]French package:(org.apache.lucene.analysis.fr)

3.1. Processing and indexing the documents

3.1.1. Parsing

Before being stored into the index, documents have to be parsed to correctly obtain their fields. We have inspected the structure of the *.txt* files with the documents: they contain IDs and bodies, marked by different tags, which have suggested us to parse them using Java regular expressions, which easily allow to detect these tags. In our project, class LEParser takes care of processing original documents in this way, storing their content in a fresh instance of the ParsedDocument class. LEParser extends DocumentParser, which is an abstract class providing a general structure for parsers and which is able to iterate over ParsedDocument instances.

3.1.2. Analyzing

This step consists of tokenizing the content of the documents and modifying and filtering the obtained tokens. This procedure is handled in class LEAnalyzer, which has allowed us to analyse the content of the documents in a customized way. It extends the abstract class Analyzer by overriding initReader and createComponents methods. By trying many different combinations and computing related evaluation measures, we have decided the sequence of operations it is supposed to do:

- **tokenization** with StandardTokenizer, provided by Lucene;
- getting all token letters to **lowercase**;
- **deletion** of all tokens representing **stopwords and French articles** (we have tried different lists and chosen one of roughly 500 terms, mostly from French but also from English language);
- **deletion** of single letters words, using a length filter;
- **stemming**, using Lucene FrenchLightStemFilter; we have observed that more aggressive stemming algorithms, even if simpler, provide worse performances;
- **deletion of numeric tokens**, except for integers with more than 3 digits, which often represent years; this filtering step is managed by our NumberFilter.

Some little variations to this schema ("standard analyzing") for the different systems will be briefly described.

3.1.3. Indexing

This is the phase where the documents are actually stored by Lucene IndexWriter. In class DirectoryIndexer, the tree of directories containing the files with the documents is scanned and each document is parsed as described before. Then, its fields are wrapped in a ParsedDocument instance and passed to the writer, which is configured in order to perform our customized analysis. DirectoryIndexer contains a main method, to make it runnable, in order to create the index.

3.2. Searching

This is the part where we spent most of our time, implementing different strategies: the main idea was to improve the quality of the retrieval system modifying the queries.

We started by inspecting the files with the queries, then decided to create a simple parser for the `.tsv` one to get ID and content (namely the "*title*" field) of each query, storing them in a Lucene `QualityQuery` instance. When a new `Searcher` instance is created, this file reader is invoked and, in order to ensure that query contents are by default setting analyzed like the body of the documents, our customized `LEAnalyzer` is passed as a parameter to the constructor.

At searching time, boolean queries are created for each `QualityQuery` instance. Then, each boolean query is searched using BM25 similarity [4]. Resulting scores for top documents are stored, and a textual run file is produced.

This is the standard procedure to search a set of queries in a document index; our corrections are basically *Query Expansion (QE)* techniques which act on the text of queries before building the `BooleanQuery`.
Our three main ideas for query expansions are detailed below, along with a description of their implementation.

3.2.1. Pseudo-relevance feedback

Pseudo-relevance feedback technique [5] is a heuristic way to expand a query with significant words in order to improve search results. The main assumption behind it is that top results coming from the original query can be assumed to be relevant, and most frequent words in them to be therefore significant for the topic.

We have tried different implementations for this technique. As a common programming strategy, they use Lucene term vectors to iterate through the terms of each document, in order to weight them according to their frequency. We have observed that taking into account only occurrences of words in the top documents, and not all the retrieved ones, leads to better performance.

The main variations were about the weighting schema: we have tried with raw total term frequencies, average relative frequencies in the top ranked documents, and with tfidf weighting. We have then looked into evaluation measures for the results, in particular map, and seen that more complicated weighting systems resulted in similar or slightly lower scores. Therefore, Occam principle suggested us to keep the simplest system, which is considering absolute frequencies of terms across the best matching documents retrieved. Fixing a number of words to add to the original query seemed too arbitrary, so we thought to take into account frequency drops of words. By drop we mean a considerable delta from the predicted frequency of a word according to Zipf's law [6], which is an empirical law stating that the number of occurrences of terms and their rank when sorting them by decreasing frequency are roughly inversely proportional.

According to this idea, in our implementation words are added to the original query starting from the most frequent one until a fixed maximum number is reached, or a significant frequency drop is met. This drop happens at k-th-ranked word iff:

$$k \cdot tf_k < tf_1 \cdot kZipf$$

We have also tried some tuning on these two parameters (`kZipf` and the max number of words to be added), but unfortunately we observed that not expanding the query at all provided the best scores. Some additional considerations are left to sections 5 and 6.

3.2.2. Query expansion with synonym substitution

Synonym substitution query expansion [7] is a technique used to enhance the accuracy and relevance of search results. It involves expanding the user's original search query by replacing some of the query terms with their synonyms or related terms. This helps to capture a wider range of results that may not have been included in the original query and can improve the precision of the search results.

In our implementation of this technique we have used, as we explained in section 2.3, an artificial intelligence to create a list of three synonyms for the most frequent words in the query list. Then we create, starting from the original queries, three expanded queries obtained by replacing the words founded in the synonym list with their first, second and third synonym.

Afterwards we compute the average score for all the `ScoreDoc` arrays generated by the four queries and we return to the search method the `ScoreDoc` array of the query that maximize this value.

3.2.3. Pseudo-relevance feedback on synonym-expanded queries

We have also tried to merge the two previously described strategies.
Pseudo-relevance query expansion is applied to the original query and to the three queries obtaining by synonyms replacement, and the results which are kept are the ones coming from the best-performing one, computing scores as described for standard query expansion with synonyms.

3.2.4. Summary of the runs produced by the systems

- **Basic:** standard analyzing, without NumberFilter;
- **SynQE:** standard analyzing and query expansion with synonyms;
- **PseudoRelQE:** standard analyzing and query expansion with pseudo-relevance feedback;
- **AllQE:** standard analyzing without NUmberFilter and query expansion with synonyms combined with pseudo-relevance feedback;

4. Experimental Setup

4.1. Documents

We used three French collections provided by *Conference and Labs of the Evaluation Forum (CLEF)* for The LongEval challenge:

Train collection - Consisted of almost 9 GBs of files, for a total of 1570734 documents.
Short term collection - Consisted of almost of 8 GBs of files, for a total of 1593376 documents.
Long term collection - Consisted of almost of 6 GBs of files, for a total of 1081334 documents.

Train collection contains the documents, the queries and the qrels we used to train the system on, and also some fresh queries to perform in-time testing on the same documents. The others are test collections, containing only documents and queries.

The collections can be found here, along with a brief description:
https://lindat.mff.cuni.cz/repository/xmlui/longeval-train-v2 [8]
https://lindat.mff.cuni.cz/repository/xmlui/longeval-test-collection [9]

4.2. Topics

The topics were also provided by CLEF for the LongEval challenge and were contained in a `.tsv` file that is stored in the */queries* folder of the collections.

4.3. Relevance Judgements and Measures

In order to test the performances of the different analyzers, we used `trec_eval` to compare our results obtained on the train documents with the qrels given by CLEF. We focused our attention mainly on the *Mean Average Precision (MAP)* since it is described as the most relevant parameter to evaluate a run using a single number.

4.4. Tools

We developed and tested our Systems with the following experimental setups:

- Java Version: Open JDK 19.0.2;
- IDE & tools: Intellij IDEA, Apache Lucene, Apache Maven;
- Computer used:
  - CPU: 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz
  - RAM: 16,0 GB LPDDR4x
  - SSD: 512 GB SSD NVMe PCIe

4.5. Bitbucket Repository

Link: https://bitbucket.org/upd-dei-stud-prj/seupd2223-rafjam/src/master/.
We provided four solutions in the "runs" folder, which are the ones described in detail in Section 3 and in particular in 3.2.4.

5. Results and Discussion

5.1. Results on Training Data

Table 1
Scores of different systems on training collection.

Evaluation Measure	Basic	SynQE	PseudoRelQE	AllQE
MAP	0.2026	0.1667	0.1621	0.1548
nDCG	0.3636	0.3103	0.3111	0.2893
nDCG@5	0.1893	0.1564	0.1639	0.1430
P@10	0.1275	0.1051	0.0967	0.0990
Recall	0.8332	0.7319	0.7776	0.7003

This section is intended to provide some numerical results and discussion about the experiments we have conducted on training data while developing our retrieval systems. One possible comparison offered by Table 1 is the one between synonyms and pseudo-relevance query expansions.
While the SynQE System provides a slightly better result in terms of average precision, the PseudoRelQE results in a higher recall. This seems reasonable, as pseudo-relevance starts from a set of documents which have been well-ranked by a previous run, which makes them probably relevant.

As Figure 1 displays, the Basic System offers overall better performances compared to the other systems.
The PseudoRelQE and SynQE we developed offer similar precision, especially at low recall value.
We also tried to combine the two, in the AllQE System, but that resulted in a slight drop in the performances.

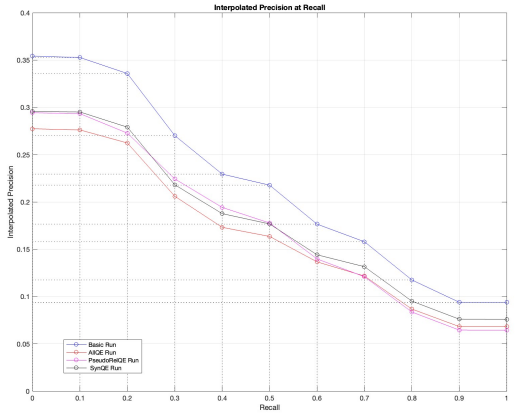


Figure 1: Interpolated Precision at Recall of the four systems.

5.2. Results on Test Data

In this section we provide the results obtained by running our algorithms on each of the three available test collections (heldout, short term, long term). The first one consists of fresh queries to be paired with the same documents of the training collection, while the other ones are made of different topics and documents.

Table 2
Scores of the four systems on heldout test collection.

Evaluation Measure	Basic	SynQE	PseudoRelQE	AllQE
MAP	0.2018	0.1614	0.1971	0.1652
nDCG	0.3740	0.3193	0.3516	0.3209
nDCG@5	0.1811	0.1461	0.1842	0.1437
P@10	0.1337	0.1082	0.1194	0.1071
Recall	0.8473	0.7658	0.7713	0.7506

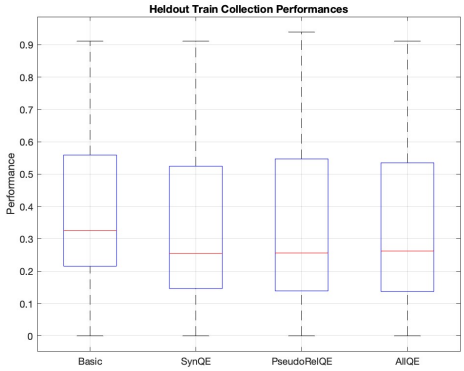
Table 3
Scores of the four systems on short term test collection.

Evaluation Measure	Basic	SynQE	PseudoRelQE	AllQE
MAP	0.2207	0.1876	0.1843	0.1785
nDCG	0.3804	0.3295	0.3355	0.3172
nDCG@5	0.2089	0.1774	0.1732	0.1685
P@10	0.1342	0.1151	0.1166	0.1101
Recall	0.8224	0.7297	0.7811	0.7208

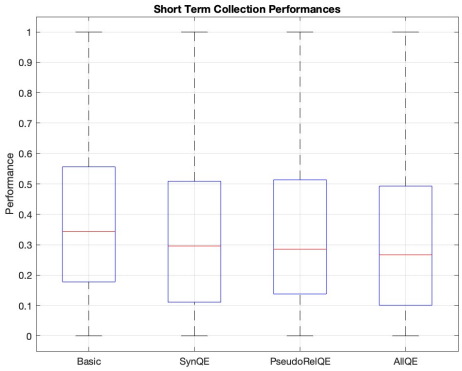
Table 4
Scores of the four systems on long term test collection.

Evaluation Measure	Basic	SynQE	PseudoRelQE	AllQE
MAP	0.2123	0.1719	0.1872	0.1676
nDCG	0.3807	0.3231	0.3490	0.3138
nDCG@5	0.1973	0.1607	0.1754	0.1562
P@10	0.1418	0.1187	0.1244	0.1131
Recall	0.8391	0.7481	0.8026	0.7353

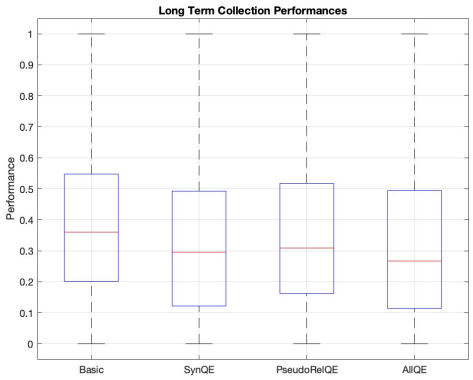
Tables 2, 3 and 4 sum up the performances of the four systems.
By inspecting them we can observe again that Basic system offers superior performances overall in terms of *Normalized Discounted Cumulated Gain (nDCG)* and that, as expected considering QE techniques, PseudoRelQE performs better than the others in terms of recall, as it starts from a set of documents which have already received an high rank on a previous run, making them probably relevant.
The measure that appears to better succeed in differentiating the systems is nDCG, while for example MAP does not show clear patterns.



(a) Heldout collection.



(b) Short term collection.



(c) Long term collection.

Figure 2: Boxplots for nDCG, for the four systems.

The boxplots in Figure 2 confirm that the Basic system offers superior performances overall in terms of nDCG, while we need some more refined tools to make statistically significant observations about the other 3 systems. They also show that the heldout collection presents a bigger variation in results and that they are unbalanced, with a bigger concentration at low performance values. Comparing these with the results obtained on the other two collections, we can assume that the quality of queries and relevance judgements in the heldout collection is worse.

Our following statistical analysis is based on nDCG scores, computed on our runs using the relevance judgements released by CLEF for each test collection.

ANOVA Table

Source	SS	df	MS	F	Prob>F
Systems	0.2037	3	0.06791	6.76	0.0002
Topics	21.4167	97	0.22079	21.97	0
Error	2.924	291	0.01005		
Total	24.5445	391			

(a) Heldout collection.

ANOVA Table

Source	SS	df	MS	F	Prob>F
Systems	2.012	3	0.67068	56.82	0
Topics	208.396	881	0.23655	20.04	0
Error	31.195	2643	0.0118		
Total	241.604	3527			

(b) Short term collection.

ANOVA Table

Source	SS	df	MS	F	Prob>F
Systems	2.487	3	0.829	71.04	0
Topics	187.339	922	0.20319	17.41	0
Error	32.277	2766	0.01167		
Total	222.103	3691			

(c) Long term collection.

Figure 3: ANOVA tables for nDCG.

In order to better study the difference between runs with different systems and on different queries, we have conducted a two-way *ANalysis Of VAriance (ANOVA)* test on nDCG performances of our runs, considering a significance of 0.05.
We want therefore to reject two null hypotheses, for each collection:

- the one stating that each system is expected to provide the same mean nDCG:

$$H_0 : \mu_{Basic} = \mu_{PseudoRelQE} = \mu_{SynQE} = \mu_{AllQE}$$

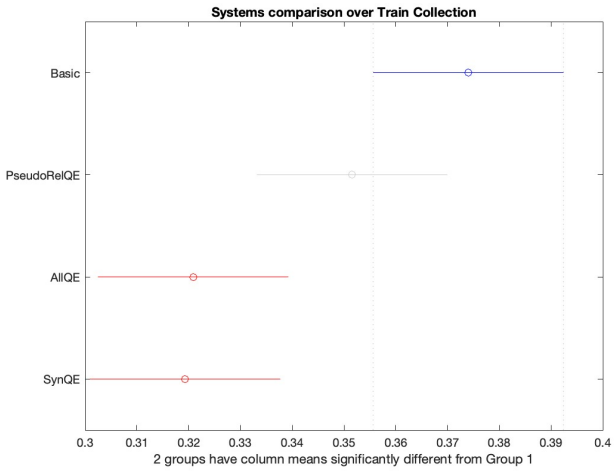
- the one stating that each topic is expected to get the same mean nDCG across the systems:

$$H_0 : \mu_i = const, \forall i \in topics$$

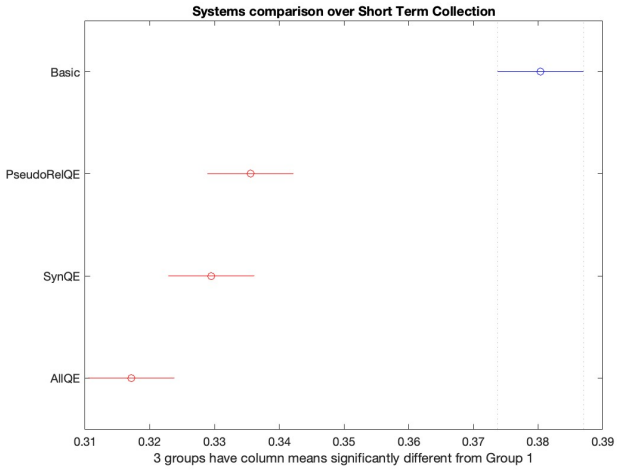
The three tables in Figure 3 represents the results of the analysis.
The F statistic is computed as:

$$F_{stat} = \frac{MS_{source}}{MS_{error}}$$

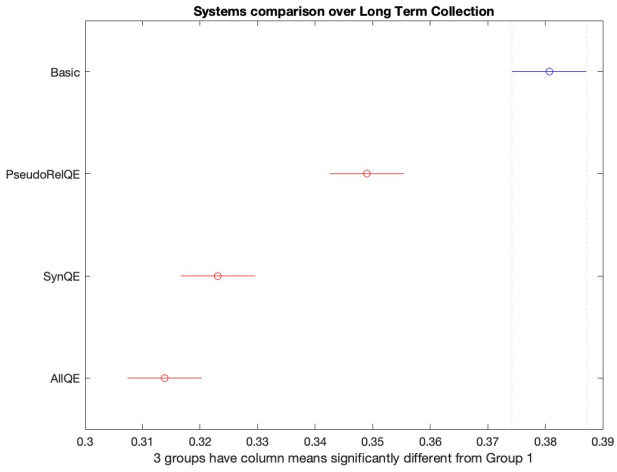
where $source$ can be either Systems or Topics. As the p-values are all much lower than 5%, both null hypotheses are rejected for every collection. This means that the performances of the four systems are significantly different from a statistical point of view, and that there are also consistent differences between performances on different topics.



(a) Heldout collection.



(b) Short term collection.



(c) Long term collection.

Figure 4: Multiple comparisons of the nDCG scores of the 4 systems with Tukey HSD.

Finally, we have plotted the graphs representing the results of Tukey's *Honestly Significant Difference (HSD)* multiple comparison tests on each collection. They allow to visualize pairwise comparisons between systems, and therefore to make more precise observations.
One important point is that we can confirm that the Basic system has better performances than the others under nDCG: the test shows that it's significantly comparable only with PseudoRelQE in the heldout collection, but the fact that it has few queries makes confidence intervals considerably larger than the ones in the other test collections, and so more easily overlapping.
Another possible consideration is that PseudoRelQE seems to be always the second best option, although the test shows statistical evidence only in the long term collection.

6. Conclusions and Future Work

The first consideration to do is about both query expansion methods we have implemented: we have unfortunately observed that this heuristic operations don't lead to an improvement in terms of precision. Therefore, we haven't performed an organic refinement of parameters (in particular the maximum number of words and `kZipf`) because the best option was to not add words.

One more satisfying observation is about pseudo-relevance query expansion: counting word occurrences only in few top-ranked documents produces better results. That's what we expected, because documents ranked after the first ones in the original searching results, which are probably the most relevant, are less likely to contain words which are significant for the topic. In our implementation, using only the top-ranked document leads to the best performance in terms of mean average precision.

One potential improvement could be the parameter tuning which was mentioned before, and could be performed after finding a more effective term weighting model.

References

[1] P. Galuščáková, G. Gonzalez-Saez, P. Mulhem, P. Goeuriot, F. Piroi, M. Popel, LongEval-Retrieval: French-English Dynamic Test Collection for Continuous Web Search Evaluation, arXiv.org, Information Retrieval (cs.IR) arXiv:2303.03229 (2023).

[2] P. R. Christopher D. Manning, H. Schütze, Introduction to Information Retrieval, 1st ed., Cambridge University Press, 2008.

[3] H. K. Azad, A. Deepak, Query Expansion Techniques for Information Retrieval: a Survey, arXiv.org, Information Retrieval (cs.IR) arXiv:1708.00247 (2017).

[4] S. E. Robertson, U. Zaragoza, The Probabilistic Relevance Framework: BM25 and Beyond, Foundations and Trends in Information Retrieval (FnTIR) 3 (2009) 333–389.

[5] P. Arora, J. Foster, G. J. F. Jones, Query Expansion for Sentence Retrieval Using Pseudo Relevance Feedback and Word Embedding, in: G. J. F. Jones, S. Lawless, J. Gonzalo, L. Kelly, L. Goeuriot, T. Mandl, L. Cappellato, N. Ferro (Eds.), Experimental IR Meets Multilinguality, Multimodality, and Interaction. Proceedings of the Eighth International Conference of the CLEF Association (CLEF 2017), Lecture Notes in Computer Science (LNCS) 10456, Springer, Heidelberg, Germany, 2017, pp. 97–103.

[6] G. K. Zipf, Human Behavior and the Principle of Least Effort, Addison-Wesley, Cambridge (MA), USA, 1949.

[7] H. Imran, A. Sharan, Thesaurus and query expansion, Department of Computer Science, Jamia Hamdard, New Delhi, India, 2009.

[8] P. Galuščáková, R. Devaud, G. Gonzalez-Saez, P. Mulhem, L. Goeuriot, F. Piroi, M. Popel, LongEval train collection, 2023. URL: http://hdl.handle.net/11234/1-5010, LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

[9] P. Galuščáková, R. Devaud, G. Gonzalez-Saez, P. Mulhem, L. Goeuriot, F. Piroi, M. Popel, LongEval test collection, 2023. URL: http://hdl.handle.net/11234/1-5139, LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.