Reproducing Benchmark for Complex Answer Retrieval

Rafail Skoulos
Student number: 4847482
r.skoulos@student.tudelft.nl

Achilleas Vlogiaris
Student number: 4875974
a.vlogiaris@student.tudelft.nl

Konstantinos Chronas
Student number: 4923162
k.chronas@student.tudelft.nl

ABSTRACT

The task of retrieving paragraphs which describe accurately a given topic is a challenging task. The TREC Complex Answer Retrieval (TREC CAR) track provides a complete dataset which can be used for this type of tasks. We have reproduced some of the approaches used by F. Nanni et. al [14], in order to measure the performance of several information retrieval approaches for this scenario. We have also added some new ones. We experimented with query expansion techniques (RM1 and RM3), ranking approaches (BM25 and Learning to Rank which we did not manage to complete) and two datasets. The performance of our system, which was measured by several evaluation metrics, was similar (for the same approaches) with the one of the paper we tried reproduce.

INTRODUCTION

Search engines have become a necessary tool for our everyday life by assisting us in finding the information we require; thus search engines need to provide accurate answers to our questions. As artificial intelligent systems have seen increased usage in mobile and voice search, where the screen size and the output length are limited, the need to develop information retrieval models that provide accurate answers to more complex questions is increasing. For example, Amazon's Alexa can provide answers to the user's simple question like "Who is the author of the Game of Thrones books?". Although these systems are designed to answer questions regarding facts and extract information, usually from a knowledge base, they fail to provide valid answers to user's questions that could have more than one answer like "Which are the factors that contributed to the Greece economic crisis?"[17]. The active research to the problem of complex answer retrieval in Information Retrieval systems provides a variety of techniques to tackle this problem, which shows promising results.

According to [10], "Complex answer retrieval (CAR) is the process of retrieving answers to questions that have multifaceted of nuanced answers". The research in information retrieval has developed many approaches for answering queries regarding accurate facts and combining pieces of information from textual sources. However, people are expecting systems to present answers to questions that require complex answers. For example, even a simple query like "Is chocolate healthy?" could be answered by presenting the negative and positive qualities along with evidence and conditions under which the qualities apply. Additionally, when a user is searching for a new topic, information retrieval systems provide the traditional blue links as the answer, instead of pointing out multiple articles or pages that might contain the answer as a summary to the user's query. The main idea behind the goal of a CAR system is the following, when a query, which consists of a topic and aspects of the topic, is given, the system should be able to retrieve information from multiple sources to thoroughly answer the corresponding question. The retrieval of high-quality longer answers is challenging because the choice of a lower rank-cutoff with the same techniques as for short answers is not sufficient, as the information that the user is trying to retrieve might be inside the corpus of the document.

Moreover, in this project, we chose to use as our knowledge base articles from Wikipedia. By looking at the structure of a Wikipedia page, it often contains a complex topic, which is explained in detail under each section. For example, paragraphs that are included in a section such as "Dutch Empire" of the page "History of the Netherlands", give an explanation about one aspect of the domain query "The history of Netherlands". Our primary goal for this project was to retrieve a ranking of passages for each section given an outline. A passage is relevant for a section only if it is included in the original page in the corresponding section. In order to tackle this problem, we followed some of the methods that are mentioned in [11]. Therefore, we tried to rank the queries with two ranking approaches, namely Okapi bm25 and a learning to rank approach. The collection that we chose to experiment with is the TREC track on Complex answer retrieval (TRECCAR), whose primary goal is to retrieve synthesized information rather than documents, for open domain queries and consist of topics, outline, paragraphs that are extracted from the English Wikipedia [3].

Outline: The structure of our report is the following: In section 2 we present an overview of the background research for complex answer retrieval and other related problem of Information Retrieval. Section 3 presents our examined approaches and Section 4 analyzes the data that we used. In section 5 we show the experimental setup that we chose to use and in section 6 we discuss the results that we obtained. Finally, in the last section some conclusions and ideas about future work are presented.

BACKGROUND

One of the first works that has been done with CAR framework is the Wikipedia content generation [3]. The collection fits naturally with this domain because the query, topics and facets often correspond well with the structure of the Wikipedia's article. Furthermore, structure of Wikipedia articles provides an extensive variety of queries, partial content and automatic relevance judgments (paragraphs can be assumed to be relevant to the headings they are under). In this section, we explore background work based on the problems of complex answer retrieval and question answering.

Deep Learning Approaches

Nicula et al.[15] suggested a novel deep learning method for predicting correct answers based on candidate context from Wikipedia using the Lucene search engine. Every paragraph in the English Wikipedia has been indexed and used as a context candidate for questions and answers that correspond to them. The top 5 results including the question and candidate answer, that were obtained by searching the index with the standard BM25 scoring method, are linked and fed into a deep neural network that computes a score for the triple of the question, candidate and context. They tested two neural networks architectures using and evaluated them using two encoders such as bidirectional long-short-term memory network (BiLSTMs) and convolutional neural networks (CNNs).

Nanni et al.[14] presented a survey of the prominent domain and query expansion approaches for the problem of complex answer retrieval. The TREC CAR collection was chosen as the dataset for the experiment and MAP, R-Prec, and MRR were used as the evaluation metrics. The train set was constructed by selecting for each true paragraph under the heading, five paragraphs from different sections of the article and five forms a different article. Besides, the test set contained the combination of all paragraphs included in the article and paragraphs extracted from other articles. The headings of each section and the page title were obtained as the queries. They experiment with three query expansion techniques, including expansion terms (RM1), expansion entities (ent-RM1) and paragraph Rocchio. Also, TF-IDF, GloVe embeddings and RDF2Vec embeddings were used as vector space representations of queries and paragraphs. The ranking approaches they examined were Okapi BM25 with k=1.2 and b=0.75, Cosine similarity and a combination of different baselines with supervised machine learning in a learning-to-rank setting using RankLib. The deep neural network they tested and compared with the other approaches was Duet[13]. The architecture of Duet learns to model the relevance between query and paragraph. The model tries to learn to good representations of the query and paragraphs for matching text and by identifying exact matches of the query terms inside the paragraphs text[14]. The DNN model outperformed all the other approaches that the authors examined.

Learning To Rank Aproaches

Agarwal, Arvind, et al. [1] first provided a comparison of 5 ranking algorithms, then showed the effectiveness of a cascading approach, where the ranking obtained by one ranker is used as input to the next stage, and evaluated aggregation techniques to combine these algorithms in order to find the best method that improves the ranking of answer candidates for factoid questions. The performance was evaluated on two datasets(the Jeopardy and Medical datasets), and as the evaluation metrics, Persicion@1 and Normalized Discounted Cumulative Gain with k=5 or10 were chosen. The algorithms that were compared for the first experiment were for logistic regression the implementation of WEKA [5], for RankBoost the Coordinate Ascent, and AdaRank from RankLib and for LamdaRank their own implementation. For the second experiment, a pruning technique was considered, where the reranking phase could provide higher precision at high ranks using the top N result from an initial ranking as training data. The baseline Logistic regression was trained on all the data, and the Top N candidates that were determined were used as the training set for all the learning to rank algorithms. For the third experiment, aggregation techniques were examined to check whether a combination of all could produce a more robust accurate ranking. The Supervised Kemeny ranking which extends the Approximate Kemeny aggregation to incorporate weights associated with each input ranking, had the best results. The results showed that the use of multiple rankers and supervised aggregation is better than using an individual ranker. Their experiments were also tested on TREC question answering datasets.

Query Expansion

Query expansion technique reformulates the user's query to improve the retrieval precision in IR operations. The goal of the query expansion is to present a refined query that is a better representation of the information need. The focus of our work is on local approaches to query expansion, which selects terms in order to improve the retrieval performance from the top extracted documents [19]. Relevance feedback, in which the query is reformulated based on documents found relevant as per user's feedback. Rocchio's method[20] was among the first to use relevance feedback. Pseudo-relevance feedback is a popular query expansion technique, where frequent words in top k ranked documents, using a ranking approach (like Cosine similarity), are assumed as relevant [6]. Besides, Language model approaches known as LM rank the documents in the collection by the probability that a query would be observed during the repeated random sampling from the document. Lavrenko et al.[6] proposed a relevance based language model or RM, which is pseudo-relevance feedback approach model, that estimates the probabilities of words in an unknown set of documents relevant to the query. In this project, we take a closer look at RM1 and RM3.

The main focus of our project is the work of Nanni et al. [14]. From the methods that were included in their work, we focused our work in replicating the learning-to-rank method and comparing with the Okapi BM25 as our baseline. We also reproduced query expansion techniques RM1, while we added the RM3 query expansion technique.

EXAMINED APPROACHES

Query Expansion Techniques

In order to improve the performance of our system, we experimented with the two following query expansion models:

Expansion terms(RM1): Uses a pseudo relevance feedback and a relevance model (probabilities of words in the relevant class), in order to extract the feedback terms. More specifically, it searches the index using the provided query, ranks the retrieved documents, it assumes that the top *n* of them are relevant and uses them to estimate the query language model. To estimate this model, it computes a weighted sum of term counts in the feedback documents, by using the query likelihood score of the feedback documents as a weight. Also it assumes that the words in the relevant documents and the query words are sampled identically and independently from a unigram distribution (i.i.d. sampling)[16]. The relevance model RM1, is given by the following formula[8]:

$$P(w|Q) = \sum_{\theta_D \in \Theta} p(\theta_D) p(w|\theta_D) \prod_{i=1}^{|Q|} p(q_i|\theta_D)$$

where $Q = q_1, q_2, \ldots, q_m$ is the query, Θ is the set of smoothed document models for the pseudo feedback documents and D the document we examine. The prior probability on documents $p(\theta_D)$ is often assumed to be uniform so it is skipped. The combination of the extracted feedback terms, are used as the new expanded query. We chose to use the implementation of Galago[1] which is based on a Dirichlet smoothed language model for the feedback run. During our experiments, we have tried many values for the parameters(number of retrieved documents and number of feedback terms) in order to find the ones that give us the better results. The best performance was achieved by 30 feedback terms and 10 retrieved document

Expansion terms(RM3):RM3 is an extension of the RM1, which has shown better results than RM1. In RM3, instead of using the feedback terms as a query, the original query is augmented by the feedback terms at the specified weight. So, it uses a linear interpolation of the original query with the terms selected by RM1. Its formulation is the following:

$$P(w|\theta'_Q)) = (1\alpha)P(w|\theta_Q) + \alpha P(w|Q)$$

where α defines the amount of feedback.

For this query expansion method we used the Galago's implementation, as we did for RM1. RM3 has one extra parameters in comparison with RM1, which is the weight assigned to each of its two components, the feedback terms and the original query. The best performance was achieved by 30 feedback terms, 10 retrieved document, and 0.5 weight for both original query and feedback terms.

Ranking Approaches

Okapi BM25

Okapi BM25 [18, 19] is a ranking function based on a probabilistic model. The ranking is based on the existence of the query terms in each document, without taking into consideration the correlation between the query terms in a document. It is used by search engines in order to rank the matching documents. Particularly, given a query, the search engines use the Okapi BM25 ranking function to rank the documents in the corpus according to their relevance to this query. Furthermore, Okapi BM25 is used as a baseline approach in many information retrieval tasks. This method has been proved very beneficial, as it has enabled the development of effective weighting functions based on three variables: the *term frequency within documents*, the *term frequency within queries*, and the *document length*. Given a query Q which containing keywords $q_1, q_2, ..., q_n$,the BM25 score of a document D can be computed by the following formula:

[1] lemurproject.org/galago.php

$$score(D,Q) = \sum_{i=1}^{n} IDF(q_i) \cdot \frac{f(q_i,D) \cdot (k_1+1)}{f(q_i,D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{avgdl})}$$

where $f(q_i,D)$ is the term frequency of term q_i frequency in the document D, $|D|$ is the length of the document D in words, and $avgdl$ is the average document length in the corpus. The free parameters k_1 and b, are used to normalize the document length parameter and to control the degree of length normalization respectively. They are usually chosen empirically and their typical values are $k_1 = 1.2$ and $b = 0.75$. As for the term $IDF(q_i)$, it is the probabilistic inverse document frequency (IDF) weight for the query term q_i and it is computed by the following formula:

$$IDF(q_i) = \log \frac{N - n(q_i) + 0.5}{n(q_i)}$$

where N is the number of documents in the corpus, and $n(q_i)$ is the number of documents containing the term q_i

In our experiments, search engine results were ranked by using Galago's implementation of Okapi BM25[2] scoring function. This is formulated by the equation we described above. Galago provides the choice to use Okapi BM25 as the scoring function for the ranking of the search results, by specifying the value of its hyperparameters k_1 and b. It also allows to specify the weight for every single term of the query. We chose to use the values of 1.2 and 0.75 for k and b parameters respectively, as they are the most common ones and the ones used by the paper we reproduce. When it comes to the weight for each term, we used the standard one (1 for each term) as there was no specific reason to consider some terms as more important than others. Finally, porter stemming and stopword removal were applied to both paragraphs and queries, in order to extract the Okapi BM25 scores.

Learning to Rank approach

As it is explained in the introduction, users make a query and for this query the search engine will retrieve a set of results. In our situation these results are paragraphs. The level of relevance of the retrieved paragraphs in respect of a query can vary. The user will choose one of the first results which probably will not be the most relevant result. Further preprocessing procedures can be applied on this matter and a ranking approach has to be implemented in order to provide to user a sufficient feed-back of his query. Various ranking methods can be found in literature and our base line as it is mentioned in the previous sections, is the BM25. A machine learning algorithm in theory can probably overcome problems of this base line approach, which is a TF-IDF-like probabilistic model and is the state-of-the-art scoring-based algorithm for ranking.

On the other hand, TF-IDF-like algorithms could fail in extreme situations, while Yuanhua Lv et al [9] indicate that

[2] https://sourceforge.net/p/lemur/wiki/Galago%20Operators/

BM25 fails when the documents are very long. Furthermore, TF-IDF vector representations has various problems as TF-IDF is based on the bag-of-words (BoW) model, therefore it does not capture position in text, semantics or co-occurrences in different documents. For this reason, TF-IDF is only useful as a lexical level feature and cannot capture semantics (e.g. as compared to topic models, word embeddings). Solutions to these problems can be introduced by a machine learning algorithm. The Learning to Rank machine learning approach [2] has been explored for at least a couple of decades now.

Learning to Rank:
In general when we are referring to learning to rank approached we mean the use of any machine learning approach, in order to rank the results of an Information Retrieval technique. The machine learning algorithms can vary a lot and in order to choose the correct one it is mandatory to take into account the nature of the information that we want to retrieve. As the learning to rank approaches are machine learning approaches, there are two major categories to train a model, namely supervised and unsupervised learning. We chose to use a supervised machine learning method. More practically, we divide the procedure into two parts, the training part and the ranking part (testing part). The training part takes training data as an input. The input is explained practically in the Experimental Section. In a more abstract manner, we could say that, as training data we use specific features that we extracted from the pairs of queries and documents. Finally the learning system constructs a ranking model that is based on this training procedure.

Coordinate ascent:
As we explained before the Learning to Rank approaches are machine approaches. We chose to use the Coordinate ascent as the machine learning algorithm which preforms the ranking to the retrieved list, as we tried to reproduce the paper [14].

Coordinate ascent is an optimization technique for linear machine learning problems and focuses on solving supervise machine learning problems. Through a number of iterations ,the Coordinate ascent algorithm try to optimize a multivariate loss function by implementing a numerous one-dimensional searches to minimize that function. The algorithm performs well when the multivariate function has short ridges but not so well when we use this algorithm to optimize loss function with long ridges, as it will be difficult to converge to the optimized value. Specifically the algorithm operates in cycles, each cycle consists of many iterations and for each cycle the algorithm uses one of the parameters and keeps the remaining parameters as fixed. The mode of Coordinate ascent operation provides an optimization procedure that is insensible to local maximum values.

Lastly as Donald Metzler and W. Bruce Croft [12] indicate, it is not completely guaranteed that the Coordinate ascent approach is able to avoid every global minima to achieve the global maxima but it can optimize a lower bound on MAP metric.

Labeling:
The most generic way to approach this problem is to estimate the relevance of a document with respect to a query, the relevance is expressed with a label, for example a high score label demonstrates the most relevant document in respect of its query. The combination of the relevance score and high quality features will provide to the supervised machine learning algorithm sufficient training data.

The labels on a query document pair can be estimated in many ways, and a average result can be used as the most objective result. The labeling is of a big importance when we are dealing with learning to rank approaches. The most unbiased labels can provide the most robust learning to rank systems. For example the human labeling is one of the most unbiased judgments about what is relevant and what is not, but the cost of such procedure could be unaffordable.

Feature construction:
The second most important part of learning to rank is the feature construction, the ranking model can be defined as a function of two variables f(x,y) where x is the document, the y is the query and the combination of these two variables is feature a vector. With these function we are able to generalize the ranking model. If we train the model for a specific amount of query-document pairs, it is possible to provide us with sufficient results when we test it on query-document pairs that the system never trained before. The most common and reliable features are BM25 score and PageRank score and the combination of them, while other more simple features can provide the high quality features that are needed.

Evaluation:
The evaluation of a ranking model's performance can be estimated comparing the ranking list of the learning to rank approach with the ground truth list. The ground truth list is based on the ground truth judgments that we mentioned before. This comparison can be expressed with different evaluation measures that used in Information Retrieval field. The most well known measures are the NDC (Normalized Discounted Cumulative Gain), DCG (Discounted Cumulative Gain), MAP (Mean Average Precision), WTA (Winners Take All) and MRR (Mean Reciprocal Rank).

Pairwise approach:
Lastly the learning to rank approach can be divided into 3 main categories as Hang Li [7] indicates, the pointwise approach, the pairwise approach and the listwise approach. In our retrieval problem the approach that is needed is the pairwise approach, as we use a pair that consists of a query and a document. Specifically the pairwise approach creates a preference pair of feature vectors that is used as input to the machine learning algorithm.

DATA SET

The goal of our task is, given a title and an outline of section headings as a query, to retrieve and rank paragraphs from a provided collection. The datasets provided by TREC-CAR contains a plethora of English Wikipedia articles with hierarchical section headings, meaning that their other elements like images and boxes were discarded.

For our experiments we have used the following two data sets:

Test200: This dataset was manually collected by the organizers of TREC CAR. It contains 200 articles and 23000 headings. Also the labels(ground truth) is provided in *QREL* files, which contains the relevance judgment for each paragraph of this data set. Because of its small size, it was used as our *experimental data set*. Furthermore, it was used as the data set where we extracted the query from, for the final evaluation of our retrieval model in the big corpus.

Train data: It contains the half of the Wikipedia articles. The paragraphs in this data set are 7 million. This dataset was used to build the index we used in our final experiments.

The names are the ones given by the organizers of TREC CAR, and they are not indicating the way they were used, except from the learning to rank approach which we discuss below.

Regarding the learning to rank machine learning approach, the dataset is already divided into two parts, the training part and the test part. The ML approach uses the training set to train its optimization function. The ground truth about relevance judgment contained in QREL files, can provide a sufficient amount of training set to use a supervised machine learning algorithm. Also *train data set* is divided into five folds which are intended for cross-validation.

We have only tested our implementation with the *test200* dataset, a smaller dataset which is able to give sufficient unbiased results in respect to the large Paragraph collection. The test200 dataset contains only one-fold, that means the RankLib algorithm has to divide the training into five different folders for cross validation, something that has been explained in detail in the Learning to Rank Section.

EXPERIMENTAL SETUP

The implementation for the system described in this report can be found in: https://github.com/RafailSkoulos17/IR_core_project.git

Tools

Galago Search Engine[3] was used for the biggest part of the work done for this assignment. More specifically, we used it to build the index of our corpus, to make the query expansion and we also used it as a search engine. For these three actions, we used *galago build* and *galago batch-search* commands. Query expansion was done by specifying some extra arguments in *galago batch-search* command.

The text processing needed to extract all the the possible queries and to create the *trectext* file needed to built the corpus was done by using the *Python 3.7* programming language. Especially, except from the standard python packages, we used **trec_car_tools** package[4] which provides some useful development tools for the processing of TREC CAR data sets. Furthermore, Python 3.7 was used to connect the several usages of Galago's Search Engine into one end-to-end component.

Finally, for the evaluation of the results that extracted by our search engine we used **trec_eval**[5] evaluation software. It's a software that is used in the Text Retrieval Conference and gives the values for a plethora of evaluation metrics by comparing the paragraphs returned by our search engine with the relevant ones as they are given by the *qrel* files included in the datasets we used.

More details about the specific way we used Galago Search Engine and Python, will be demonstrated later when the steps we followed on our experiments are described.

Granularities

For our experiments we considered the following three granularities for way the ground truth signal is released[4]:

- **Article**: The paragraph contained anywhere in the page
- **Toplevel**: The paragraph contained in section hierarchy below top level section
- **Hierarchical**: The paragraph is contained in its section

As can be easily observed by the way it is decided where a passage belongs, the task of assigning a relevant paragraph to a query, is easier as we moving from *Article* granularity to the *Hierarchical* one. The reason for this behavior is that moving to this direction, the number of relevant paragraphs is decreasing.

Evaluation metrics

In order to evaluate the performance of proposed information retrieval models, we we used four evaluation metrics, namely mean average precision (MAP), R-Precision (R-Prec), mean reciprocal rank (MRR) and geometric mean average precision (GMAP). Following, we describe the formulation and the meaning of each of them.

- *Mean Average Precision (MAP)*: Given a set of queries, the mean average precision of them is the mean of the average precision scores for each query. More specifically, for a query q_i, the average precision is the mean of the precision values obtained for the set of top k documents retrieved. Subsequently, the mean average precision is obtained by taking the mean of the average precision values over the queries. Its value is given by the following equation[11]:

$$MAP(Q) = \frac{1}{|Q|} \cdot \sum_{i=1}^{|Q|} \frac{1}{m_i} \cdot \sum_{k=1}^{m_i} Precision(R_{ik})$$

  where Q is the query set, q_i is a query of the query set, $\{d_1, \dots d_{m_i}\}$ are the relevant documents for the query q_i, and R_{ik} is the set of topk ranked retrieval documents from the query q_i. MAP is one of the most standard evaluation metrics used in the TREC community as it has been shown to have especially good discrimination and stability[11].

- *R-Precision (R-Prec)*: R-Precision is the mean (over all queries) of the precision after r documents have been retrieved, where r is number of relevant documents available for query q. So it is clear that for the computation of the R-precision metric, the knowledge of all the documents which are relevant to a query is required. Every query has a different number of relevant documents R, which is used as the cutoff for the calculation of the R-Precision. R-Precision is calculated by the following simple formula:

$$R-Prec = \frac{r}{R}$$

[3]https://sourceforge.net/p/lemur/wiki/Galago/
[4]https://github.com/TREMA-UNH/trec-car-tools
[5]https://github.com/usnistgov/trec_eval

where r is the number of relevant documents in the top-R document retrieved. R-precision also depends on the size of the set of relevant documents, meaning that a perfect system could score 1 for each query, whereas, even a perfect system could score less than 1 if the number of relevant documents in the corpus are less that R. Also, by definition if the number of retrieved document we examine is equal to the number of relevant document in the corpus (R=r), then $\frac{r}{R}$ is not only the R-Precision, but also the R-Recall of this set of results. This measure has been empirically shown to be highly correlated to mean average precision.

- *Mean Reciprocal Rank (MRR)*: MRR is the mean of the reciprocal ranks of the highest ranking relevant document, over all queries. If no relevant documents were returned, then this score is zero. More specifically, the reciprocal rank for results returned by a query, is the reciprocal of the rank of the first relevant document, that is 1 for first place, $\frac{1}{2}$ for second place, $\frac{1}{3}$ for third place and so on. As a result, the mean reciprocal rank metric can be computed by the following formula:

  $$MRR(Q) = \frac{1}{|Q|} \cdot \sum_{i=1}^{|Q|} \frac{1}{rank\, i}$$

  where Q is the query set and $rank_i$ is the rank position of the first relevant document meet at the results of the retrieval for the i-th query.

  It is important to mention that MRR takes into account only the first relevant while the other relevant answers are ignored.

- *Geometric Mean Average Precision (GMAP)*: The GMAP evaluation metric is suitable for cases where the improvement in low-performing queries need to be highlighted. The difference between GMAP and MAP is that the former is the geometric mean of the the per-topic average precision while the latter is their arithmetic mean. GMAP is given by the following equation:

  $$GMAP(Q) = \sqrt[|Q|]{\prod_{i=1}^{|Q|} AP_i}$$

  where Q is the query set, and AP_i is the average precision of the query i.

  Instead of formalize it as the |Q|-th root of the product of the |Q| values, it is usually represented as an arithmetic mean of logs, that is:

  $$GMAP(Q) = \exp \frac{1}{|Q|} \cdot \sum_{i=1}^{|Q|} \log AP_i$$

  As we said above, GMAP is designed to measure the improvement in low-performing queries. So, for example if a run improves the average precision for a query A from 0.06 to 0.08 , while decreasing the one for query B from 0.56 to 0.54, the arithmetic mean will stay unmodified, but the geometric mean will be improved.

Experimental procedure

Building Index

The first step was the building of the corpus index. In order to build it, we first had to process the dataset, which was in *cbor* format, and convert it into a *trectext* file. In particular, we created a trectext file in which every passage of the corpus had the following format:

<DOC>
<DOCNO> Unique paragraph ID **</DOCNO>**
<TEXT>
Paragraph's text
</TEXT>
</DOC>

This format was required by Galago Search engine, in order to be able to recognize each paragraph and its text, so as to build the index file. The build of the index was done by executing the *galago build* command with the proper arguments. The most important of them are the following:

- **fileType**: Specifies the type of the input file used to build the index (trectext in our case)
- **inputPath**: Specifies the path of the input file
- **tokenizer/fields**: Specifies the fields contained in trectext file.

Also the porter stemmer was used, which is the default one in galago build command. All the parameters were specified in a *JSON* file which was used as an argument when we built the index. This procedure was done for both experimental dataset and the big corpus.

Retrieving Relevant Documents

Following, we used this index to retrieve the relevant documents for each of the queries. But first, we had to extract all the possible queries (topics) for all the three different granularities (article, top-level, and hierarchical). For this purpose, we used *Python 3.7* and especially *trec_car_tools* package, the way it has been described in previous section. Then we use the queries that we extracted, to create a JSON file which used as parameter in the document retrieval process from the Galago Search Engine. Particularly, this JSON file contains the parameters that required for the search process, which are the following:

- **index**: Specifies the location of the corpus index
- **requested**: Specifies the maximum number of results to be returned by the search engine
- **scorer**: Specifies the scorer function to be used. According to the one chosen, its parameters may be specifies in other fields.
- **queries**: Contains a list with all the possible queries.

Then, the retrieval of the relevant documents was done by executing the *galago batch-search* command with this JSON file as a parameter. This command returns the search results for all the queries included in JSON file.

The scorer that is used to rank the retrieved documents was Okapi BM25, which functionality was described in previous section. The parameters chosen were $b = 0.75$ and $k_1 = 1.2$. In order to performed a grid search for the value of k_1 parameter in the range $[1.2, 2]$ with step 0.1. Also the number of requested document was set to 20.

Furthermore, we used the query expansion techniques, as described in "Approaches" Section, in order to investigate their effect in the relevance of the retrieved documents. Each of the methods used, namely RM1 and RM3, were incorporate them in the search process of Galago's Search Engine. This was accomplished by inserting two fields in the JSON file (parameters of search process). These were the following ones:

- **operatorWrap**: We put the value "*rm*" to this field in order to specify that we want to expand the query used for retrieving the documents.
- **relevanceModel**: We used the relevance model we want to used[6].
- **fbDocs**: Specifies the number of documents used for the estimation of the query language model, needed for the query expansion.
- **fbTerm**: Specifies the number of feedback terms that will be used.
- **fbOrigWeight**: This field was used only for RM3. It specifies the weight that will be assigned to the original query. The value $1 - fbOrigWeight$ is the weight assigned to the feedback terms.

Finally, we performed a grid search to specify the optimal values for the *fbDocs*(in range [5, 20] with step 5), *fbTerm*(in range [5,30] with step 5), and *fbOrigWeight*(in range [0.2,0.8] with step 0.1) terms. As it was done before, the grid search was conducted in the experimental data set. The best results were achieved for $fbTerm = 30$, $fbDocs = 10$ and $fbOrigWeight = 0.5$. These parameters were also used for the query expansion on the big corpus.

Here we should mention that because of the limited time and the big computational cost, we experimented only with the hierarchical granularity for the big corpus.Also this agrees with the procedure followed by the paper we reproduce.

Evaluating Results

Finally we used these results, along with the *QREL* files included in the dataset, as arguments in *trec_eval* executable which provides several evaluation metrics for our search results. These metrics contain the ones we examine in this assignment.

RankLib

We used an open source platfrom[7] to apply a Learning to Rank approach. More specifically, to implement the Coordinate ascent ML algorithm, we used the the RankLib, a learning to rank library[8]. The RankLib uses a set of features that includes the *BM25* score, *TF* score, *IDF* score *TF-IDF* score and other documents attributes like document length, anchor length etc. With sufficiently many features and with the fine tuning of them, combined with a large dataset, a Learning to Rank approach can provide good ranking results and outperform the base-line score rankings.

Features that are used for RankLib:

- **Paragraph length:**
- **Paragraph's anchor length:**
- **TF:**
- **IDF:**
- **TF-IDF:**
- **IDF per paragraph:**
- **BM25 score per paragraph:**

Input to the RankLib model: The input to the Ranklib model is a set of rows, each of which contains a set of features as it can be seen below.

1 qid:1 1:1 2:1 3:0 4:0.2 5:0 6:0 7:0 # 1A

where the attributes represent the relevance between the query and the paragraph which represented by binary values (0: non relevant, 1: relevant), the query id, the paragraph length, the anchor length, the *TF* score, the *IDF* score, *TF_IDF* score the BM25 score, the *IDF* per paragraph score and the paragraph id as a comment, respectively.

More specifically feature extraction was the most time-consuming task in order to try to implement a learning to rank approach for the paragraph ranking. We used *Python 3.7* environment with a set of libraries in order to extract the 7 different features. Each *RankLib* input line can be seen in the scheme above and is a pair of a query and its relevant paragraph. That means we had a set of queries and each query was paired with at least one or more paragraphs.

Firstly we used the *trec_car* library to extract valuable information from the training set (the relevant text of each paragraph in respect of a query, the anchor of each paragraph, the id of each paragraph). Then we set a unique id for each query. According to this values we managed to extract the document length of each pair by counting the words at each pairâĂŹs paragraph and then we extracted the number of different anchors that could possible exist in a pair.

Afterwards we calculated the *TF* feature by counting the sum of the frequency of the terms that exist between a query and a paragraph, something that could possible means that for a large query(in terms) it is possible to have more than the correct relevant paragraphs. More precisely we could say that in specific situations where the query is very large, when it is combined with a small paragraph it can produce an insufficient feature. Then the *IDF* score is calculated by applying the equation below, more specifically by counting the number of the documents that contain at least one relevant term in respect of the given query.

$$idf(t,D) = \log \frac{N}{|\{d \in D : t \in d\}|} \tag{1}$$

[6]*org.lemurproject.galago.core.retrieval.prf.RelevanceModel1* for RM1 and *org.lemurproject.galago.core.retrieval.prf.RelevanceModel3* for RM3
[7]https://sourceforge.net
[8]https://sourceforge.net/projects/lemur/

The *IDF* score get multiplied with the *TF* score and provide a *TF-IDF*-like feature, but the multiplication was not applied as a vector to vector dot product but as a two scalar multiplication.

For the *BM25* score we used the *Gensim* library[9]. The *BM25* score is calculated on each pair individually, as we extract a specific score for each query-paragraph pair. We noticed that the *Gensim* library didn't performed well when we gave as input less than three documents. In our situation we only needed to calculate the *BM25* score of the query-paragraph pair which means that the input documents were just two. The scores was consisted mostly of negatives and zeros values. Lastly we used a custom feature that is not indicated in the *LETOR* [10], the *IDF* of every pair without taking into account the other paragraphs.

The results provide us an ouput with the first number to represent the relevance of the query-paragraph pair, the second number represent the query id and the next 7 number represent the features that we explained above. The last number is the paragraph id which is commented. Afterwards, we made a list of pairs where their relevance label is zero, we randomly selected a set of paragraphs and we combined them with a set of queries.We assumed that making the same number of non relevant and relevant pairs should give good results.

Lastly we extract the ground truth to make the *qrel* files. Again, we used the same amount of queries and the same amount of paragraphs, a *qrel* file consist of a query id number, an empty column and lastly the paragraph id which before in the *RankLib* input representation was commented. The format of a qrel file can be seen below.

1 QO 1 0

The extraction of the *IDF* feature was extremely time consuming as it needed to search in the whole corpus the terms of the query, something that didn't allow us to extract features for the whole Paragraph data set, we used the test200 dataset. As this dataset was too small, we used the build in *Gesnim's* library function to divide the training set to five different folds in order to perform cross-validation.

The results was not promising as the *MAP* values was exceedingly high. The most sensible assumption to these results was that the extracted features couldn't provide to RankLib the knowledge to distinguish which is a relevant pair and which is not, but trained the ML algorithm to manage every pair as relevant.

Furthermore while we the *RankLib* was running we were able to notice the most used features and some of the features was used more times that other features. That probably indicates that some features predominate of the remaining ones.

[9]https://398-1349775-gh.circle-artifacts.com/0/documentation/html/generated/gensim.summarization.bm25.get_bm25_weights.html

[10]https://www.microsoft.com/en-us/research/project/letor-learning-rank-information-retrieval/

RESULTS

Results for experimental data set

In figure 1, we can observe the performance of our system, for both the three metrics, by using BM25 ranking approach without a query expansion. In comparison with the value of the paper that we reproduce(figure 2), it is clear that the results in all evaluation metrics are close enough.

In our assignment, we chose to measure the performance of our system not only for hierarchical granularity as done in the paper, but also for article and top-level granularity. The results can be seen in figure 1. We can observe that the article granularity performs much better than the others in all the evaluation metrics. This result was quite expected because, as we explained in previous section, article granularity contains more relevant paragraphs for each query. About the performance of the other two granularities, it noticeable that all metrics are close enough. Especially, the top-level one has better *MAP* while hierarchical yields better performance in the other three metrics. For this comparison, we expected top-level to performs better for the same reasons as before.

In addition, we implemented the query expansion techniques *RM1* and *RM3*. The results obtained for these query expansion techniques can be seen in figures 3 and 4. We can notice that RM1 has lead on slightly worse results than the implementation without query expansion. On the other hand RM3 have shown better or the same results for the most of the evaluation metrics and the granularities for our system, and subsequently for the one demonstrated by the paper. The behavior was the expected one, as the RM3 has been shown to provide better results than RM1 as it does an interpolation of the original query and the expansion terms, as it was discussed in Approach Section.

Results for big corpus

In figure 5 we can see the results for the evaluation metrics of our system which were obtained from the big corpus.It can be noticed that the trend of the result is similar to the one of the experimental data set. Namely, the best results for all metric were achieved for article granularity, while the ones for the other two granularities are similar with the hierarchical one being slightly better. But although the trend is similar, the performance for the big corpus is mush worse. In particular, the value of *MAP* ranges from 0.160 to 0.182 and the one of *R-Prec* ranges from 0.252 to 0.131. The other metric has also been reduce significantly. These results were expected as the index has become much bigger than the index of the experimental data set, so it became more difficult to discriminate the relevant paragraphs from the non-relevant ones.

When we compare the performance of our system with the one of the paper (figure 7) we can observe that, although the datasets used have some differences, our results are slightly better. More specifically, the MAP has increased by 0.023, the *R-Prec* by 0.018 and the *MRR* by 0.033.

Finally, the implementation of the query expansion techniques *RM1* and *RM3* has not improved the results, as can be observed by figure 6. To be more specific, *RM1* has much worse

	Our Results			
	MAP	*R-Prec*	*MRR*	*GMAP*
Article	0.554	0.5703	0.993	0.4776
Toplevel	0.332	0.2744	0.472	0.0882
Hierarchical	0.346	0.2662	0.451	0.0722

Table 1. Evaluation metrics for each of the 3 granularities for experimental data set, by using BM25 scoring and no query expansion

	Paper Results		
	MAP	*R-Prec*	*MRR*
Hierarchical	0.320	0.232	0.409

Table 2. Evaluation metrics for hierarchical granularity for experimental data set which is used in the paper, by using BM25 scoring and no query expansion

performance than the implementation without the query expansion, something that is clearly depicted by the values of all the evaluation metrics. As for the *RM3*, in contrast to the experimental data set, in the big corpus it has not improved the retrieval results but it has shortly worsen them.

CONCLUSIONS AND FUTURE WORK

Conclusions

In this assignment we tried to reproduce the work done from Federico Nanni et al.[14]. We experimented with some approaches of this paper, but have also tried some new ones(*RM3* query expansion, article and top-level granularities). Unfortunately, we were not able to complete the implementation of Learning to Rank approach, so we end up with only the *Okapi BM25* as ranking approach, which consists a disadvantage of our implementation. When the results achieved are considered, we were able to reproduce the results of the paper for the Okapi BM25 ranking approach, meaning that the difference between the two results is almost negligible, and especially in the most of the evaluation metrics we has achieved better results.

Through the process of reproducing the paper we have chosen, we have to make much research about the methods proposed by the paper and also other ones that may be effective for out task. This helped us to better understand many of the components of the Information Retrieval field and the ways we can address the several problems that can occur. Also, the fact that the paper was not completely reproducible, meaning that it do not describe in details the procedure they followed, has made

	Our Results			
	MAP	*R-Prec*	*MRR*	*GMAP*
Article	0.477	0.495	0.992	0.390
Toplevel	0.297	0.249	0.423	0.041
Hierarchical	0.294	0.226	0.387	0.028

Table 3. Evaluation metrics for each of the 3 granularities for our system for the experimental data set, by using BM25 scoring and RM1 query expansion

	Our Results			
	MAP	*R-Prec*	*MRR*	*GMAP*
Article	0.620	0.627	0.992	0.554
Toplevel	0.339	0.266	0.462	0.110
Hierarchical	0.340	0.246	0.433	0.085

Table 4. Evaluation metrics for each of the 3 granularities for our system for the experimental data set, by using BM25 scoring and RM3 query expansion

	Our Corpus Results			
	MAP	*R-Prec*	*MRR*	*GMAP*
Article	0.182	0.252	0.624	0.0468
Toplevel	0.160	0.131	0.263	0.0028
Hierarchical	0.173	0.136	0.249	0.0020

Table 5. Evaluation metrics for each of the 3 granularities for our system for the whole coprus data set, by using BM25 scoring and no query expansion

	Our Corpus Results			
	MAP	*R-Prec*	*MRR*	*GMAP*
RM1	0.100	0.084	0.153	0.0002
RM3	0.159	0.124	0.233	0.0028

Table 6. Evaluation metrics for each of the hierarchical granularity for our system for the whole coprus data set, by using BM25 scoring and RM1 and RM3

	Paper Corpus Results		
	MAP	*R-Prec*	*MRR*
Hierarchical	0.150	0.118	0.216

Table 7. Evaluation metrics for hierarchical granularity for the whole corpus which is used in the paper, by using BM25 scoring and no query expansion

us search more by ourselves and obtain a better image of the examined approaches. Furthermore, the implementation phase of the assignment was also beneficial for us. We learn how to use some common tools used for information retrieval tasks and in general how such a task can be faced accurately.

Future work

An important improvement is the completion of the Learning to Rank approach which has been left incomplete but seems promising. This approach has also achieved very good results for the paper we reproduce. Also, a neural network approach would be a interesting idea as there have been a rapid progress in that field the last years, and especially in their usage in information retrieval tasks. Finally the Rocchio query expansion technique may have yield some improvement.

REFERENCES

1. Arvind Agarwal, Hema Raghavan, Karthik Subbian, Prem Melville, Richard D Lawrence, David C Gondek, and James Fan. 2012. Learning to rank for robust question answering. In *Proceedings of the 21st ACM international conference on Information and knowledge management*. ACM, 833–842.
2. Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to Rank: From Pairwise Approach to Listwise Approach. In *Proceedings of the 24th International Conference on Machine Learning (ICML '07)*. ACM, New York, NY, USA, 129–136. DOI: http://dx.doi.org/10.1145/1273496.1273513
3. Laura Dietz, Manisha Verma, Filip Radlinski, and Nick Craswell. 2017a. TREC Complex Answer Retrieval Overview. In *TREC*.
4. Laura Dietz, Manisha Verma, Filip Radlinski, and Nick Craswell. 2017b. TREC complex answer retrieval overview. In *Proceedings of TREC*.
5. Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. 2009. The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter* 11, 1 (2009), 10–18.
6. Victor Lavrenko and W Bruce Croft. 2017. Relevance-based language models. In *ACM SIGIR Forum*, Vol. 51. ACM, 260–267.
7. Hang LI. 2011. A Short Introduction to Learning to Rank. *IEICE Transactions on Information and Systems* E94.D, 10 (2011), 1854–1862. DOI: http://dx.doi.org/10.1587/transinf.E94.D.1854
8. Yuanhua Lv and ChengXiang Zhai. 2010. Positional relevance model for pseudo-relevance feedback. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*. ACM, 579–586.
9. Yuanhua Lv and ChengXiang Zhai. 2011. When Documents Are Very Long, BM25 Fails!. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '11)*. ACM, New York, NY, USA, 1103–1104. DOI: http://dx.doi.org/10.1145/2009916.2010070
10. Sean MacAvaney, Andrew Yates, Arman Cohan, Luca Soldaini, Kai Hui, Nazli Goharian, and Ophir Frieder. 2018. Characterizing question facets for complex answer retrieval. *arXiv preprint arXiv:1805.00791* (2018).
11. Christopher Manning, Prabhakar Raghavan, and Hinrich Schütze. 2010. Introduction to information retrieval. *Natural Language Engineering* 16, 1 (2010), 100–103.
12. Donald Metzler and W. Bruce Croft. 2007. Linear feature-based models for information retrieval. *Information Retrieval* 10, 3 (01 Jun 2007), 257–274. DOI: http://dx.doi.org/10.1007/s10791-006-9019-z
13. Bhaskar Mitra, Fernando Diaz, and Nick Craswell. 2017. Learning to match using local and distributed representations of text for web search. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1291–1299.
14. Federico Nanni, Bhaskar Mitra, Matt Magnusson, and Laura Dietz. 2017. Benchmark for complex answer retrieval. In *Proceedings of the ACM SIGIR international conference on theory of information retrieval*. ACM, 293–296.
15. Bogdan Nicula, Stefan Ruseti, and Traian Rebedea. 2018. Improving Deep Learning for Multiple Choice Question Answering with Candidate Contexts. In *European Conference on Information Retrieval*. Springer, 678–683.
16. Javier Parapar López. 2013. Relevance-based language models: new estimations and applications. (2013).
17. George-Sebastian Pirtoaca, Traian Rebedea, and Stefan Ruseti. 2018. Improving Retrieval-Based Question Answering with Deep Inference Models. *arXiv preprint arXiv:1812.02971* (2018).
18. Stephen E Robertson and Steve Walker. 1994. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *SIGIRâĂŹ94*. Springer, 232–241.
19. Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, and others. 1995. Okapi at TREC-3. *Nist Special Publication Sp* 109 (1995), 109.
20. Joseph John Rocchio. 1971. Relevance feedback in information retrieval. *The SMART retrieval system: experiments in automatic document processing* (1971), 313–323.