University of Aveiro

Departamento de Electrónica, Telecomunicações e Informática

Master degree in Informatics Engineering

Assignment nº3

Authors:
David Ferreira, Nº 72603
Pedro Matos, Nº 73941

Information Retrieval

22nd November 2017

Contents

Summary

In this report not only will be explained the content and purpose of each developed class but the choices that were taken in group during its development in order for the system to be as modular and efficient as possible. For this third assignment additional classes were developed apart from the ones that were already present in the previous ones. The purpose of this assignment was the creation of a weighted tf-idf indexer and ranked retrieval system with further performance analysis such as the measurement of the query throughput, the engine precision, recall, etc. Throughout this report it will be explained the functions and purposes of each developed class and even though there will be given an explanation for that in here, there is still present on each class its corresponding *JavaDoc* as well. It is important to note that this project was adapted so that both developed systems can be executed depending on the arguments given as input by the user, making all the previous and the current developed systems integrated in a single one.

Classes

In this chapter, the classes that were developed for this assignment will be analyzed and described. Figure 1 represents the class diagram of the application, as it can be observed, the application is separated in several packages, the majority of the classes that were developed for this assignment are located in the packages *evaluation*, Figure 2 and *normalizer*. Apart from the new developed classes, some previous ones had to be changed as well in order to integrate the latest developed project with this one.

Interface EfficiencyMetricsWriter

This interface defines an entity that writes the analyzed results from the processing of the engine from memory to disk with a specific format which depends on its implementation. This interface defines one method "saveEfficiencyResults()" which is responsible for creating a file with a given name and writing in it the results that were measured in memory to disk.

Class NormalizedSearchEngineBuilder

This class extends the class *SearchEngineBuilder* and is responsible for building a *SearchEngine* with a *NormalizedProcessor*. The development of this class was necessary for supporting the implementation of the Builder programming pattern.

Class NormalizedProcessor

This class implements the Interface *QueryProcessor* and assumes a td-idf ranking system where all the words in the query are calculated along with the same words appearing in each document based on the lnc.ltc weighting scheme.

Class Vector

This class was created for the appropriate representation of a vector. This Vector objects are used in order to represent both the terms and respective score for each document and the terms and respective score for each query. In other words, this objects represent the term Vectors for each document and for each Query that are further used for calculating the resulting relevance score for each document for each query.

Class Normalizer

As the name suggests this class is responsible for applying the normalization process on both the documents and query Vectors. In the end of the normalization process, both Vectors contain the normalized values that are ready to be calculated in order to obtain the rank of each Document for each query. It is important to note that the normalization process occurs after the TF is applied to each document vector and the TF-IDF is applied to each query vector.

Class NormalizedBuilder

This class extends the abstract class *IndexerBuilder* and represents a builder that constructs a *Normalizer* along with a *ComplexTokenizer* that uses an English Stemmer.

Class EfficiencyMetricsFileWriter

This class implements the interface *EfficiencyMetricsWriter* and is responsible for writing in a file the asked processing analysis results for this assignment: The engine recall, the engine precision, the F-Measure, the mean average precision, the mean precision at rank 10, the mean reciprocal rank, the query throughput and the median query latency.

Class Evaluator

This class is responsible for two different main aspects: the first one is to create a relevance matrix given a relevance gold standard file, which will then be used for the analysis of the engine's performance, and the next one is to calculate all the efficiency measures given a specific threshold and minimum relevance. The minimum relevance is an argument that can be passed as an argument and it defines the minimum relevance needed for a file to be considered relevant in the gold standard file given for the study. The threshold defines the minimum score that a file needs to have in order to be considered relevant in the documents that are scored according to the developed ranking system.

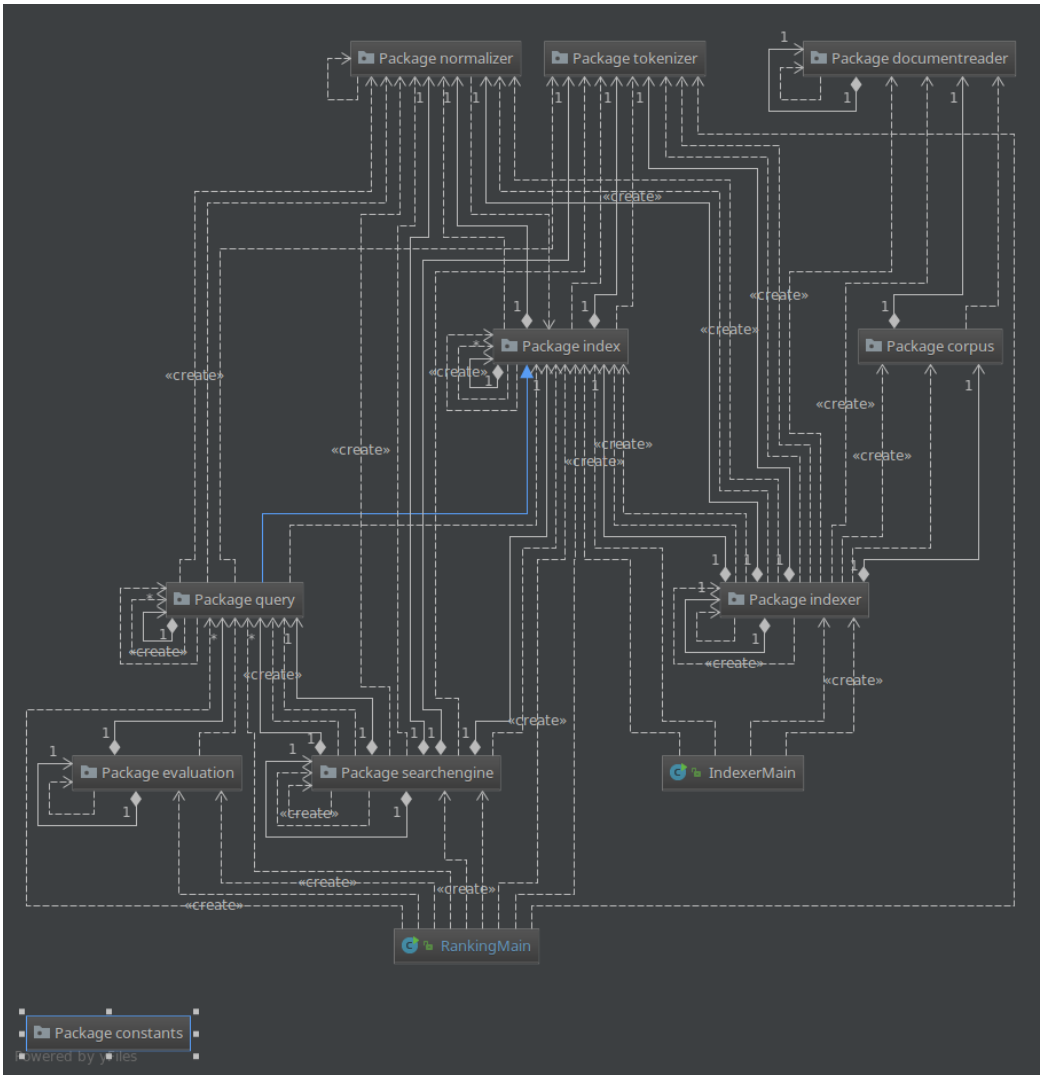

Figure 1: Application's class diagram

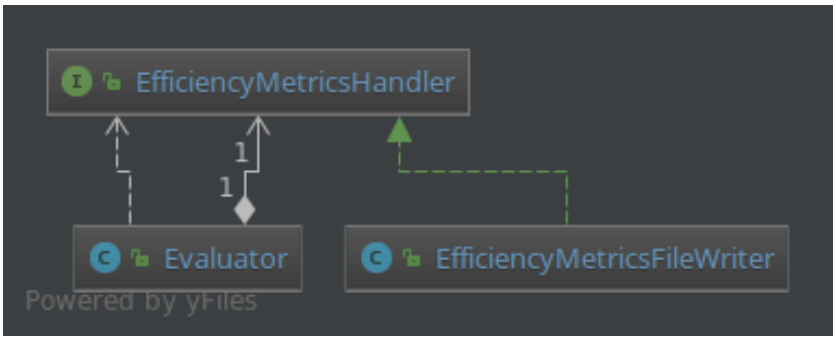

Figure 2: Package evaluation class diagram

Software Design Patterns

Just like the first assignment, in a attempt to improve readability and modularity, the same software design patterns were applied in the development of this part of the application.

Strategy

This pattern consists in creating an interface that describes a certain behavior from certain entities. This way every implementation of the interface is independent and allows interchangeability of modules [3].

Figure 2 shows the application of this pattern with the interface *EfficiencyMetricsHandler*.

Ranking

Unlike the other assignments this application does not use Boolean Retrieval, instead it uses Ranked Retrieval which allows for query language that is more user friendly and solves other issues related with Boolean Retrieval.

tf-idf weighting

The tf-idf weighting system is known as the best weighting scheme in Information Retrieval, the tf-idf weight of a term in a document, $W_{t,d}$, is given by the following formula:

$$W_{t,d} = (1 + \log tf_{t,d}) * \log(N/df_t) \tag{1}$$

Where $tf_{t,d}$ is the frequency of the term t in a document, d, N is the size of the document corpus and df_t is the number of documents that contain the term t.
After calculating the tf-idf weight has been calculated each document and query can be represented as a multi-dimensional vector where each term is an axe of the vector space.

Formalizing vector space proximity

After representing the documents and queries as vectors the next step is to calculate the proximity between them to determine each documents are more relevant to certain queries. This is done by calculating the cosine, $cosine(\vec{q}, \vec{d})$, $\vec{q}$ is the query vector and $\vec{d}$ the document vector.

$$cosine(\vec{q}, \vec{d}) = \frac{\sum_{i=1}^{|v|} q_i d_i}{\sqrt{\sum_{i=1}^{|v|} q_i^2} \sqrt{\sum_{i=1}^{|v|} d_i^2}} \tag{2}$$

Where q_i and d_i are the tf-idf weight of the term i in the query and document, respectively.

Result Scores

Using the relevance scores associated with the corpus that is being used on this application, the Cranfield dataset, some efficiency and evaluation metrics were calculated. This chapter will present the result scores for both the ranking system of assignment nº3 and assignment nº2 were the ranking was made by calculating the frequency of query words in a document and how many query words were in a document. The application allows a user to set threshold that only allows documents that have a certain score or above to be presented to him, so it would be interesting to analyze the results based on that threshold.
All the result are calculated using the *ComplexTokenizer* with Stemming mentioned in the Assignment nº1.

Frequency of query words scores

Figure 3, 4 and 5 show the results for Boolean Retrieval system of assignment nº2, this ranking system takes into account the number of occurrences of the queries terms in a certain document.

Query Words in document scores

Figure 6, 7 and 8 show the results for Boolean Retrieval system of assignment nº2, this ranking system takes into account the query words that occur in a certain document, not their frequency.

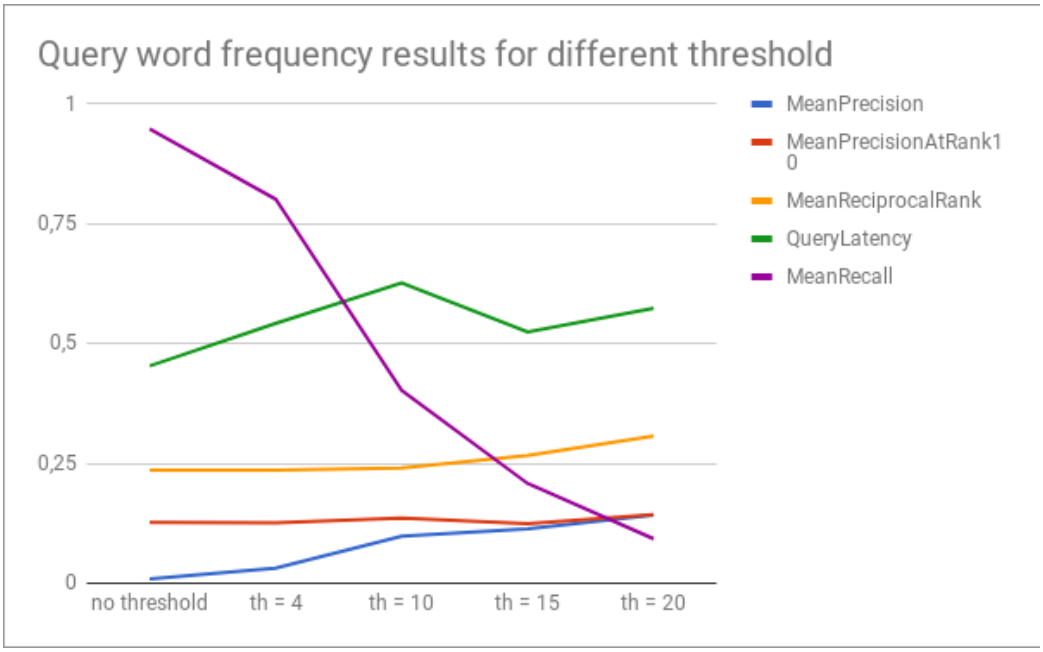

Figure 3: Query word frequency evaluation and efficiency metrics results for different thresholds

Normalized ranking with tf-idf weight scores

Figure 9, 10 and 11 show the results for Ranked Retrieval of the application analyzed in this report, this ranking system uses the raking method explain in .

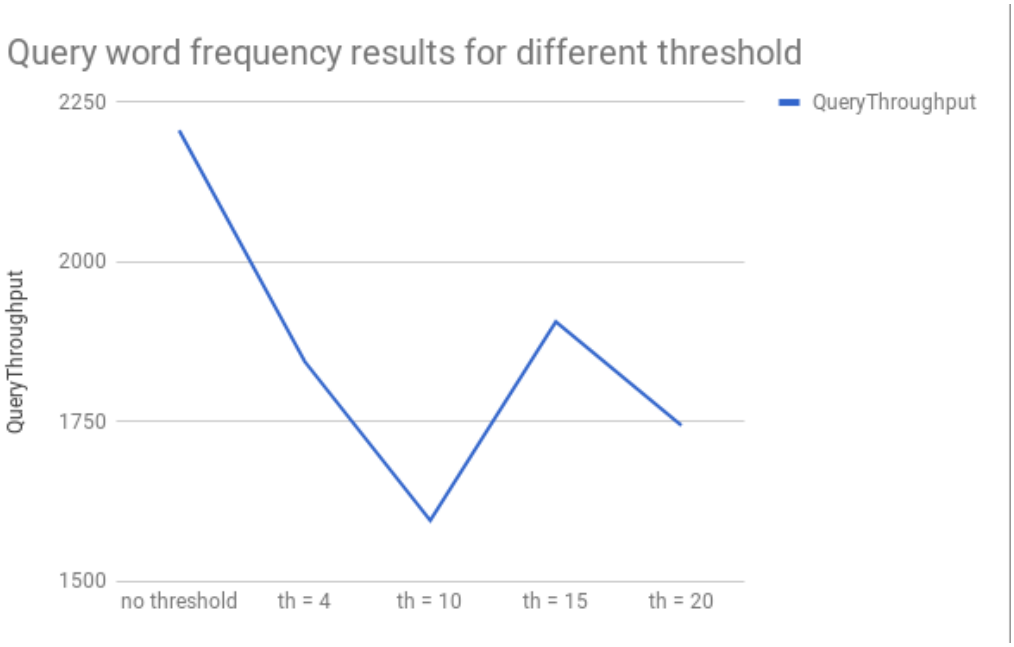

Figure 4: Query word frequency query throughput results for different thresholds

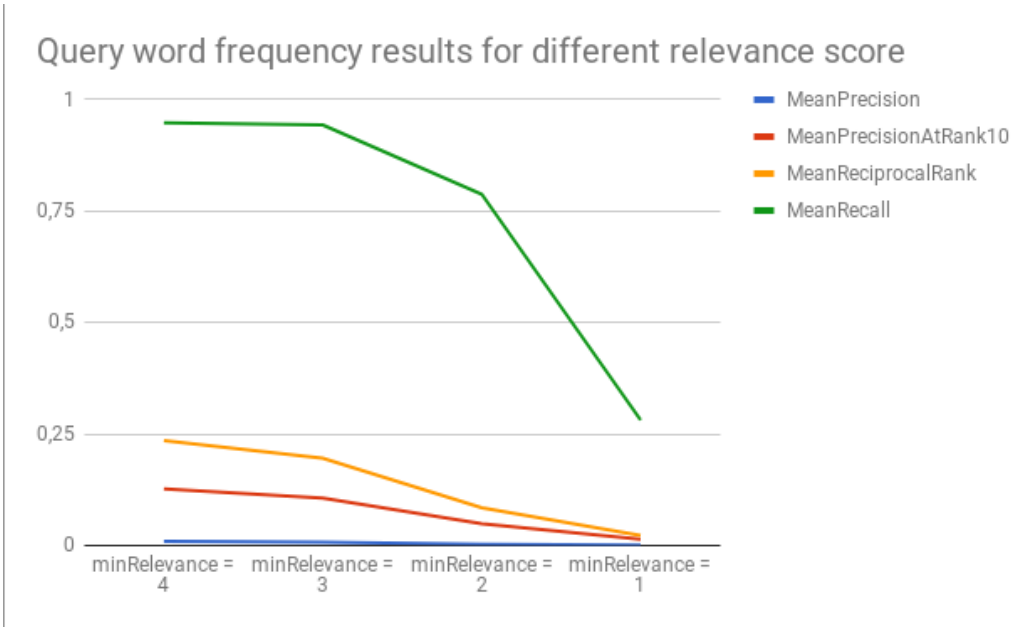

Figure 5: Query word frequency evaluation metrics results for different relevance scores

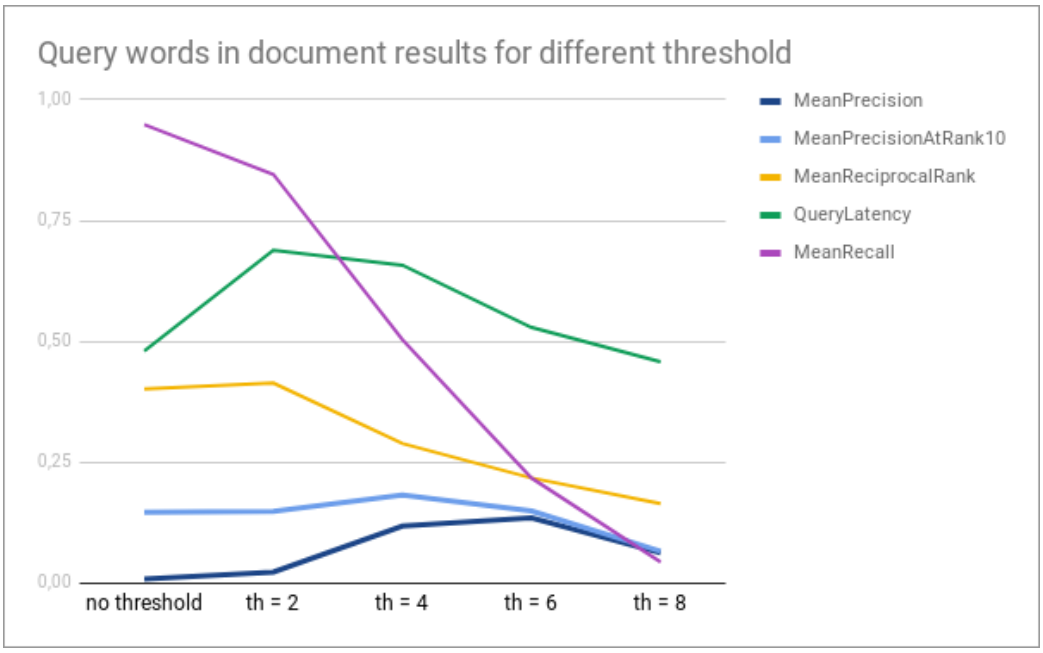

Figure 6: Query words in document evaluation and efficiency metrics results for different thresholds

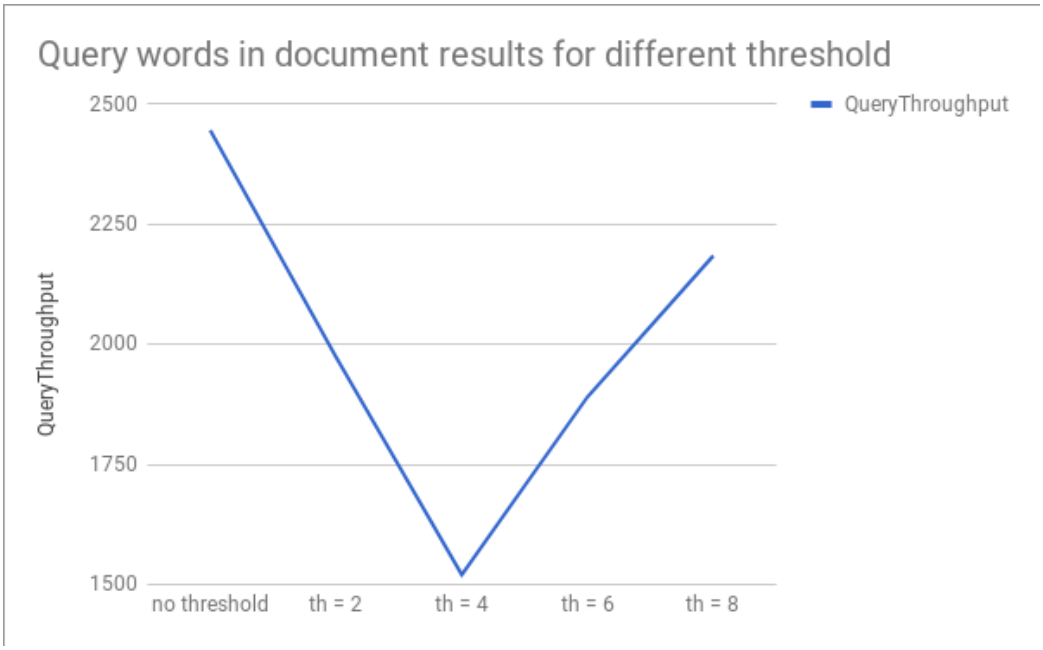

Figure 7: Query words in document query throughput results for different thresholds

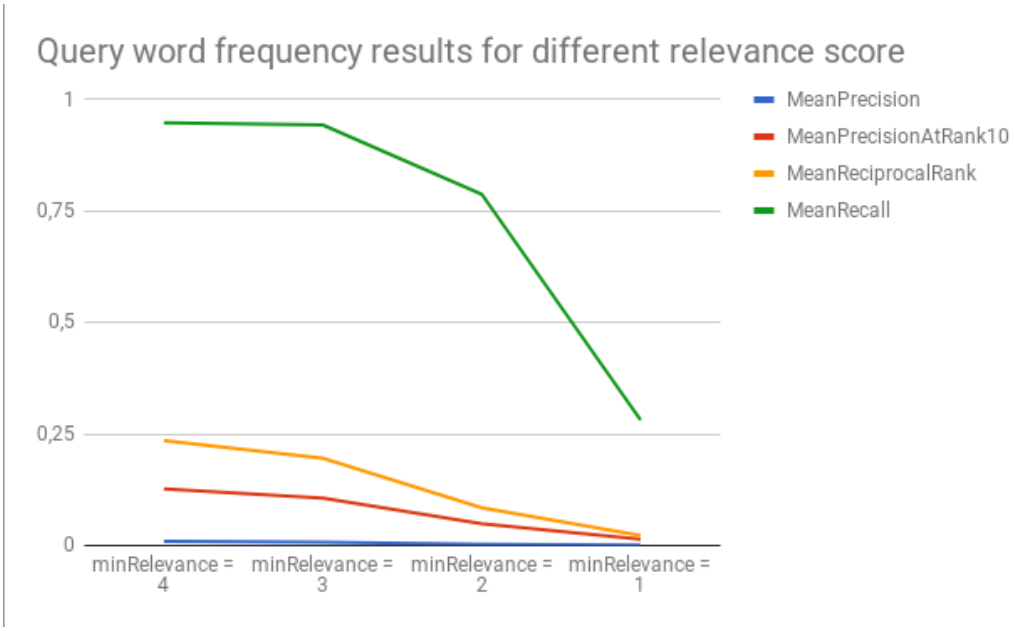

Figure 8: Query words in document evaluation metrics results for different relevance scores

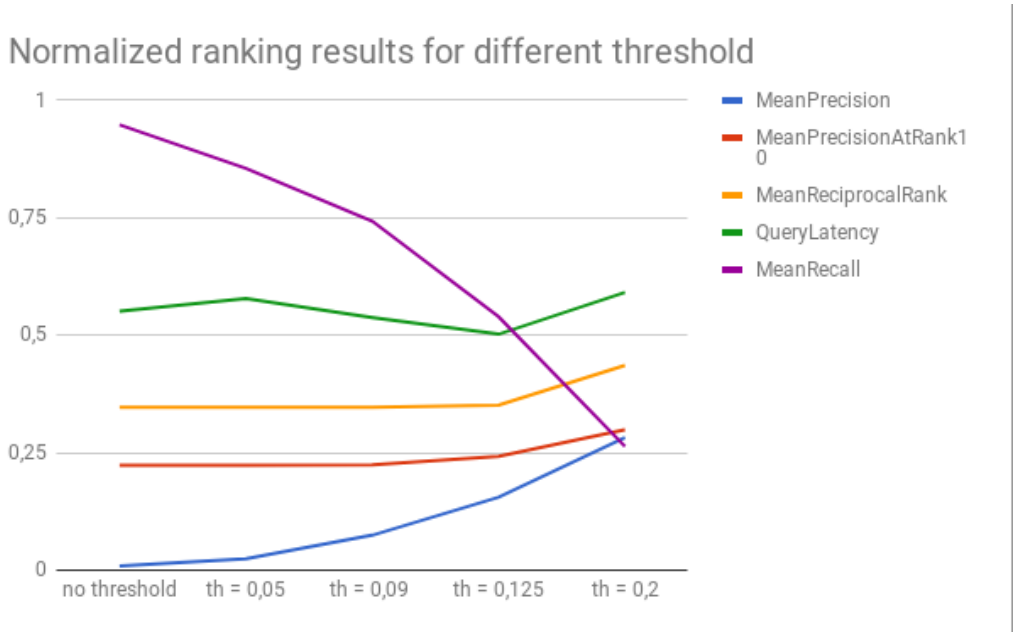

Figure 9: Query words in document evaluation and efficiency metrics results for different thresholds

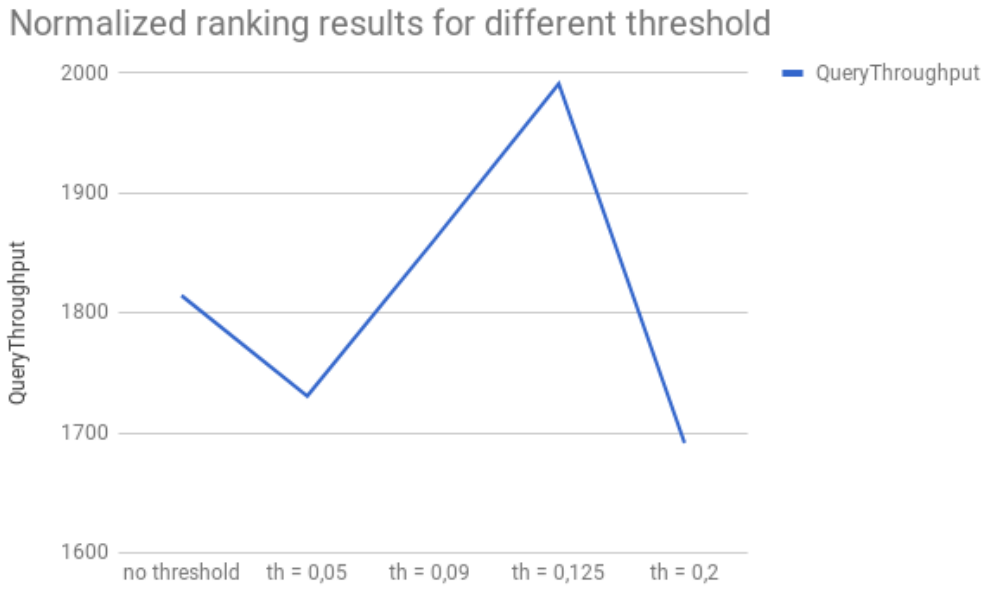

Figure 10: Query words in document query throughput results for different thresholds

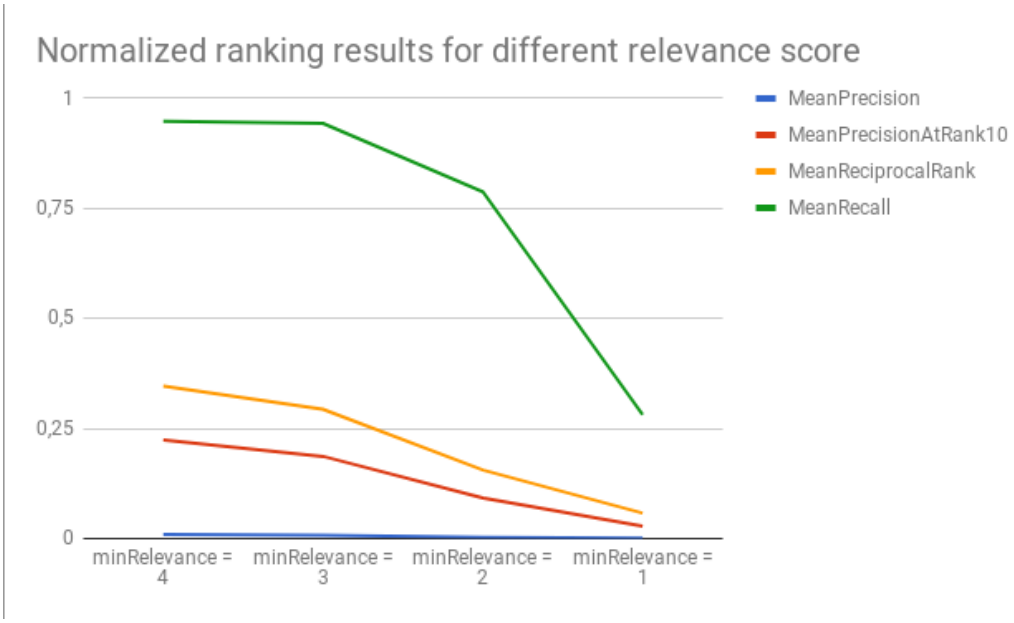

Figure 11: Query words in document evaluation metrics results for different relevance scores

Efficiency Measures

In this chapter, some measures were took to analyze the performance of the application. Three measures were chosen to evaluate the system, indexing time, querying time, max amount of memory used and query result space on disk. These measures were took in a machine with a Quad-core, Intel Core i7-2630QM, 2.9 GHz[2].

Querying and Indexing Time

This measure was took by comparing two *TimeStamps*, before reading the corpus and after the index is written on disk for the *IndexerMain*, before loading the index that is written on disk and after the queries have been processed for the *RankingMain*. Figure 12 shows the average run times for 3 consecutive runs, each column represents an average time measure for both main classes using Boolean and Ranked Retrieval.

Maximum amount of memory

This measures were taken using a profiling tool from IDE Netbeans[1]. Figures 13 and 14 show the different memory profiles for the different main classes.

Result space in disk

This measure was took by checking the space occupied by the result files for the different indexes. The occupied disk space can be observed in Table 1.

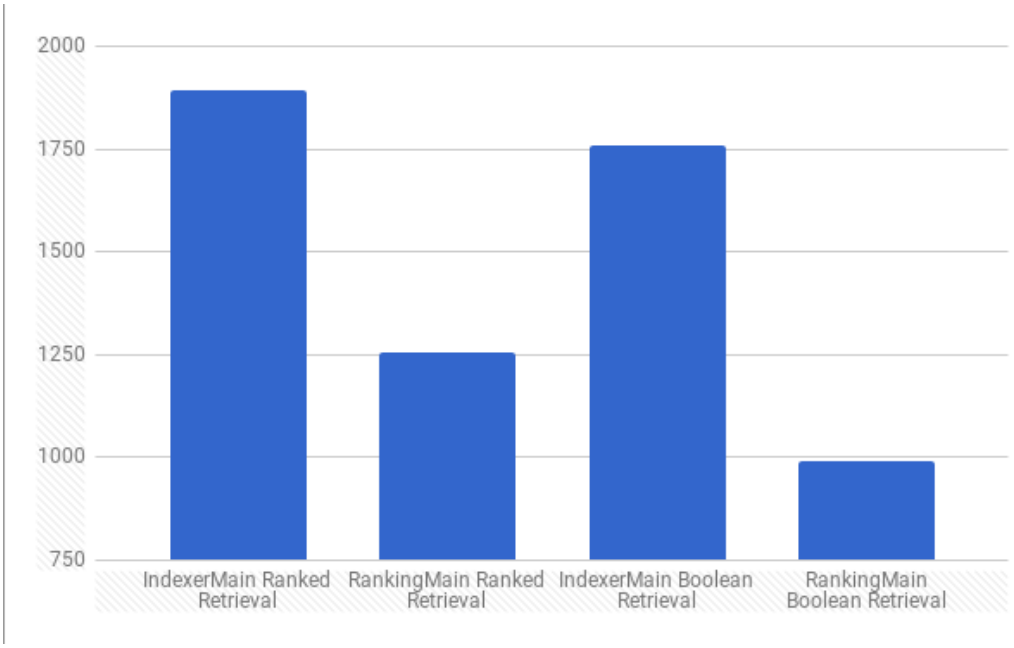

Figure 12: Average run times

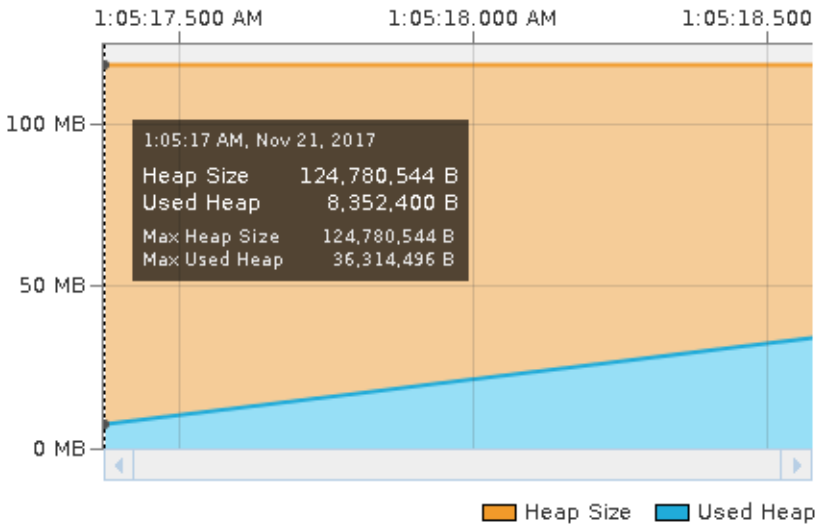

Figure 13: Memory profile for an index built with a *ComplexTokenizer* with stemming

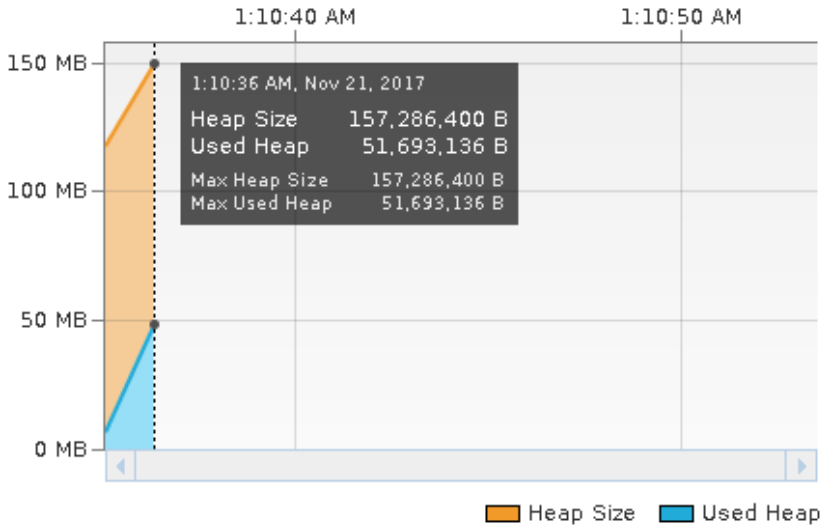

Figure 14: Memory profile for an index built with a *ComplexTokenizer* without stemming

	Boolean Retrieval	Ranking Retrieval
IndexerMain	712,1KB	1,9MB
RankingMain	2,9MB	2,9MB

Table 1: Disk space occupied by the result files of different indexes

How to run

Since the application now consists of two main classes both can be executed, the application can be used by importing the project to an IDE or by following the instructions of the README file that consist on using the command line to manually compile the code and run it. For both cases the recommended approach is to run the main Classes with the '-h' tag to check the USAGE print with detailed information about each argument that is needed to properly run each program.
The main class *IndexerMain* is used to create an *InvertedIndex* form a document corpus and saves it on disk, it takes arguments like the directory of the corpus, the scoring system, which consist on how the document will the scored, by counting the number of occurrences of a certain term of by applying the *tf* and normalization scores.
The other main class, *RankingMain*, loads the index from disk, processes a file that contains queries and answer them producing a result file stored by the scored of each document. Some of its arguments are the file that contains the queries, a file that contains relevance scores for the document corpus, etc.

Conclusion

Observing Figure 15 it's possible to see that two additional main blocks were developed in comparison to the last assignments. First there was a need of implementing a *Normalizer* so that the vectors that contain the terms and their respective weights for both the queries and the documents could be normalized. These normalizing processes are essential for further calculation of the ranking of each document for each query. The other main block that was built was the *Evaluator* block which is responsible for reading the gold standard relevance file and for building a data structure that represent for each query each possible relevance and for each relevance of each query the list of documents that belong to that relevance. With that data structure and the results of the ranking evaluation it is possible to make measurements of the engine's performance data. After the measurements are done the results are written to a file in disk. As a final note, comparing both results from each assignment, its pretty clear that better results are produced from a tf-idf rank method rather than a simple word and term frequency ranking.

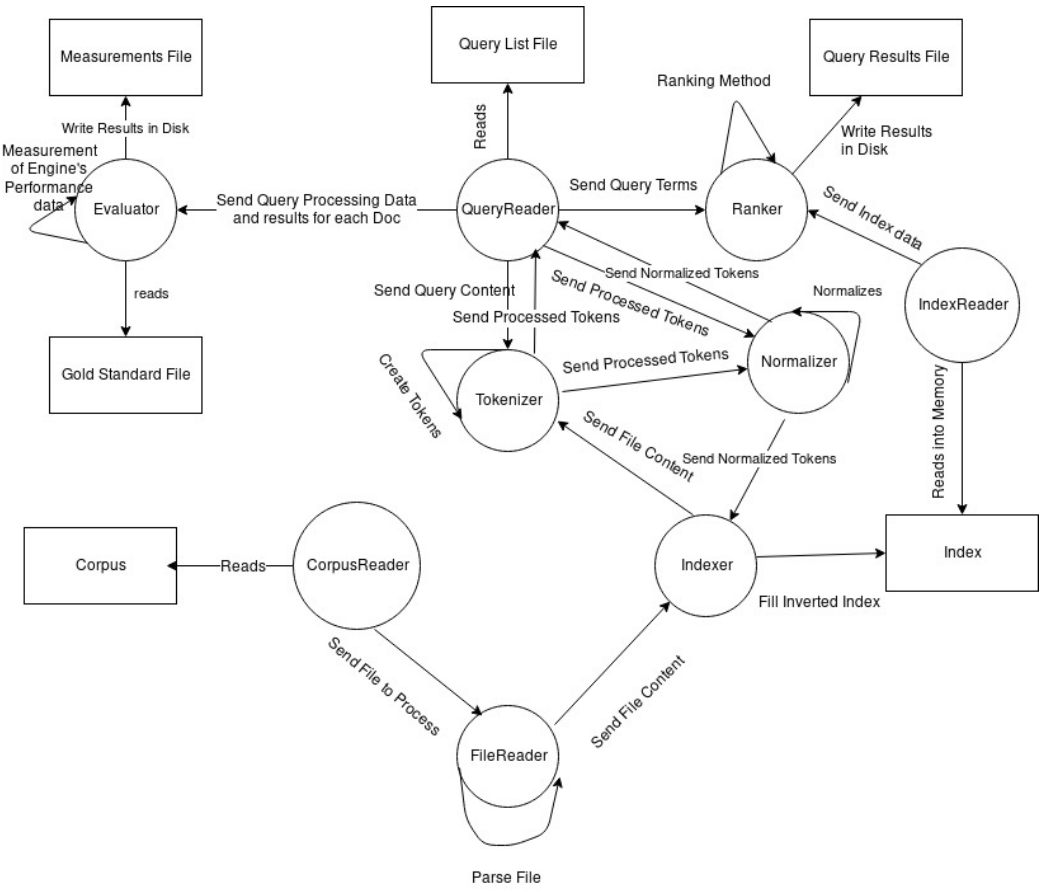

Figure 15: Dataflow diagram of the application

References

[1] Oracle Corporation and/or its affiliates. *Profiler*. 2014. URL: https://profiler.netbeans.org/ (visited on 22/10/2017).

[2] Intel Corporation. *Intel® Core™ i7-2630QM Processor*. 2014. URL: https://ark.intel.com/products/52219/Intel-Core-i7-2630QM-Processor-6M-Cache-up-to-2_90-GHz (visited on 22/10/2017).

[3] ToturialsPoint. *Design Patterns Strategy Pattern*. 2017. URL: https://www.tutorialspoint.com/design_pattern/pdf/strategy_pattern.pdf (visited on 22/10/2017).