University of Aveiro

Departamento de Electrónica, Telecomunicações e Informática

Master degree in Informatics Engineering

Assignment nº4

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

Information Retrieval

20th December 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 forth assignment additional classes were developed apart from the ones that were already present in the previous ones. The purpose of this assignment was the development and evaluation of methods for relevance feedback and thesaurus based query expansion. Throughout this report it will be explained the functions and purposes of each of the new developed classes and even though there will be given an explanation for that in here, there is still present on each of them 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 being the majority of the classes that were developed for this assignment present in the packages *queryExpansion*, Figure 2 and *relevanceFeedback*, Figure 3 . Apart from the new developed classes, some previous ones had to be adapted as well in order to integrate the latest developed project with this one.

Interface QueryUpdater

This interface defines an entity that is responsible for either updating the query term weights after the relevance feedback or for making its expansions with similar terms to the terms it already contains based on a thesaurus. This interface defines one method "updateQueries()" that is responsible for the procedures mentioned above.

Class QueryExpansionEngineBuilder

This class extends the class *SearchEngineBuilder* and is responsible for building a SearchEngine with a QueryExpansion *Updater*. This *Updater* is responsible for expanding the original query with similar words for each of the terms it contains based in a thesaurus that is created from the original corpus.

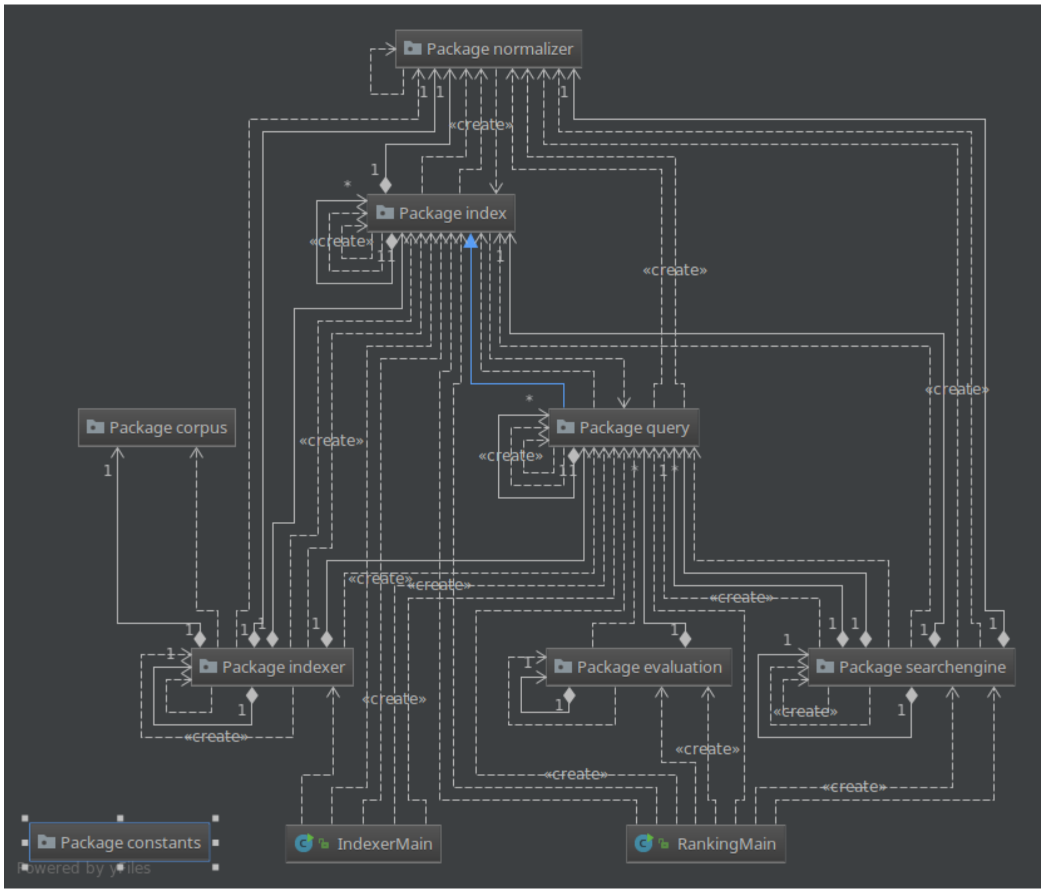

Figure 1: Application's class diagram

Class ImplicitFeedBackEngineBuilder

This class extends the class *SearchEngineBuilder* and is responsible for building a SearchEngine with a ImplicitFeedBack *Updater*. This *Updater* is responsible for updating the query term weights based on the feedback obtained from the top 10 documents that are all implicitly considered as relevant.

Class ExplicitFeedBackEngineBuilder

This class extends the class *SearchEngineBuilder* and is responsible for building a SearchEngine with a ExplicitFeedBack *Updater*. This *Updater* is re-

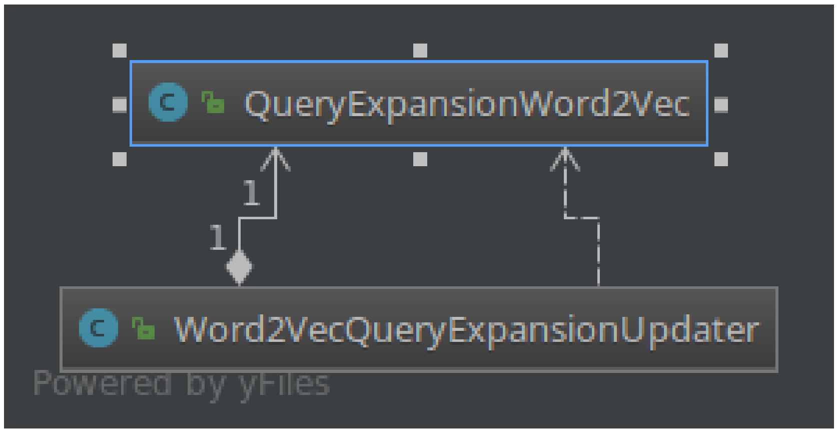

Figure 2: Package queryexpansion class diagram

sponsible for updating the query term weights based on the feedback obtained from the top 10 documents in an explicit way, considering both the relevant and non relevant documents according to the gold standard file.

Class Word2VecQueryExpansionUpdater

This class implements the Interface *QueryUpdater* and is responsible for updating the queries by expanding them. This expansion is based in a thesaurus that is build from the corpus content and results in adding to the query similar terms to the terms it already contains.

Class ImplicitFeedBack

This class implements the Interface *QueryUpdater* and is responsible for updating the weights of the terms of the queries that are subjected to update according to the implicit relevance feedback results.

Class GoldStandardFeedBack

This class implements the Interface *QueryUpdater* and is responsible for updating the weights of the terms of the queries that are subjected to update

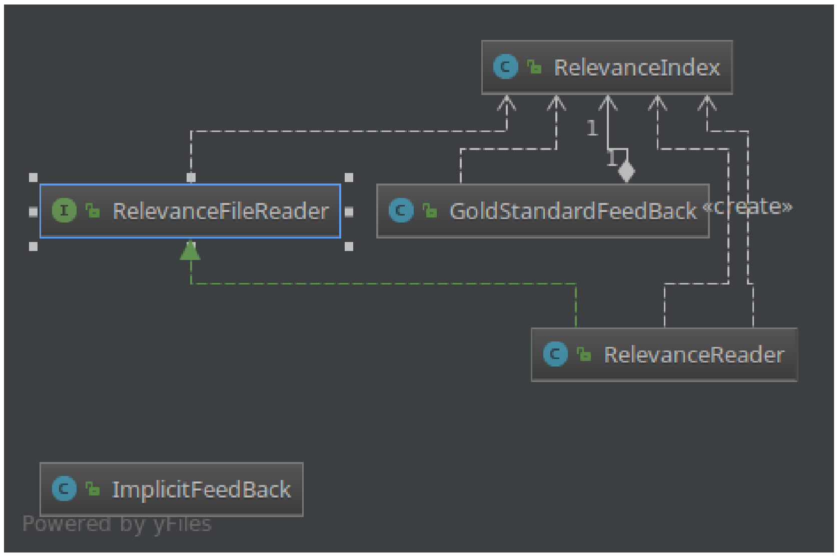

Figure 3: Package relevancefeedback class diagram

according to the explicit relevance feedback results from the relevance scores of the cranfield dataset, also known as the gold standard.

Class QueryExpansionWord2Vec

This class is responsible for generating a thesaurus of word associations based on the corpus content. For this, word embeddings are generated from the collection and are used to retrieve the nearest words to a specific one, making it possible to expand the queries terms this way. The first time this module it's executed, a model has to be trained with machine learning, then, the trained model it's exported to a file that is loaded in later executions.

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 [4].
Figure 3 shows the application of this pattern with the interface *RelevanceFileReader*.

Builder

This pattern aims to separate the construction of a complex object from it's representation, an object that aggregates several entities can have many representations depending on the type of entities that its using. This can be solved with *Constructors* for each representation but the complexity of it's creation is still shown. The solution to this problem is to create a protocol that allows the creation of all the representations through a series of steps and define those on an abstract "recipe" in an interface. Any implementation of this interface becomes a possible representation of the complex class. All a client has to do is to ask the builder for it's respective representation of the object [3].
For this assignment, this situation can be applied to the class *SearchEngine*, its attributes are a *QueryUpdater*. This objects can have many implementations, each one giving a different representation to the *SearchEngine*. For

example, giving a *ImplicitFeedBack* object to the *SearchEngine* creates a *SearchEngine* that uses Relevance Feedback with implicit feedback.

Relevance Feedback

The goal of this assignment was to implement a Relevance Feedback system to improve the "quality" of the queries for the sake of better search results. This works by updating the weight of the query terms based on the fact if they belong to a relevant document or not. Terms from relevant documents can also be added to the original query.
To update the weights of the query terms the Rocchio Algorithm is used,

$$\vec{q}_m = \alpha \vec{q}_0 + \frac{\beta}{|D_r|} \sum_{\vec{d}_j \in D_r} \vec{d}_j - \frac{\lambda}{|D_{nr}|} \sum_{\vec{d}_j \in D_{nr}} \vec{d}_j \tag{1}$$

, where $\vec{q}_m$ is the modified vector, $\vec{q}_0$ the original query vector, α, β, λ are weights, D_r the set of known relevant document vectors and D_{nr} the set of known irrelevant document vectors. After the sum of the scores of the relevance documents with the scores of the irrelevant documents, the five terms with most weights are filtered, otherwise the result would be a query with a huge amount of terms, which isn't the intended.

Explicit Feedback

In order to perform a relevance feedback in an explicit way, it was necessary to read the relevance scores of the cranfield dataset contained in the gold standard file. The application considers as positive feedback the documents that are in the set of 10 documents with the higher scores from its search results and have a relevance score associated in the gold standard file. The documents in that set that don't have a score are considered as negative feedback. The α, β, λ values were chosen based on empirical observation and for this method. $\alpha = 1$, $\lambda = 0.5$ and

$$\beta = \frac{1}{s} \tag{2}$$

, where s is the score of a certain document.

Implicit Feedback

As implicit feedback the application uses its own search results to update the query. It considers its 10 documents with the highest score from its search results as positive feedback. In this case there is no negative feedback to consider and the parameters considered were $\alpha = 1$ and $\beta = 0.5$.

Query Expansion

This technique adds useful terms to the queries, for example, synonyms of the terms it already contains. In this case it was used a thesaurus that is generated using machine learning. For each query term 3 are added that represent the ones that are the most similar to that corresponding one.

Result Scores

The Relevance Feedback and Query Expansion methods were compared with the ranking method of the assignment 3.
As a final note its possible to observe with the graphics depicted on the Figures 4, 5, 6 and 7, that the general performance results got slightly better with the relevant feedback methods, being better in its explicit version rather the implicit one, and got worst by using the query expansion method. Even though in this specific case the query expansion method wasn't really helpful, it can be in fact pretty useful in many search engine systems. Its possible to visualize what was just mentioned above by looking at the graphic depicted in the Figure 7 which represents the results comparison between the methods developed for this assignment.

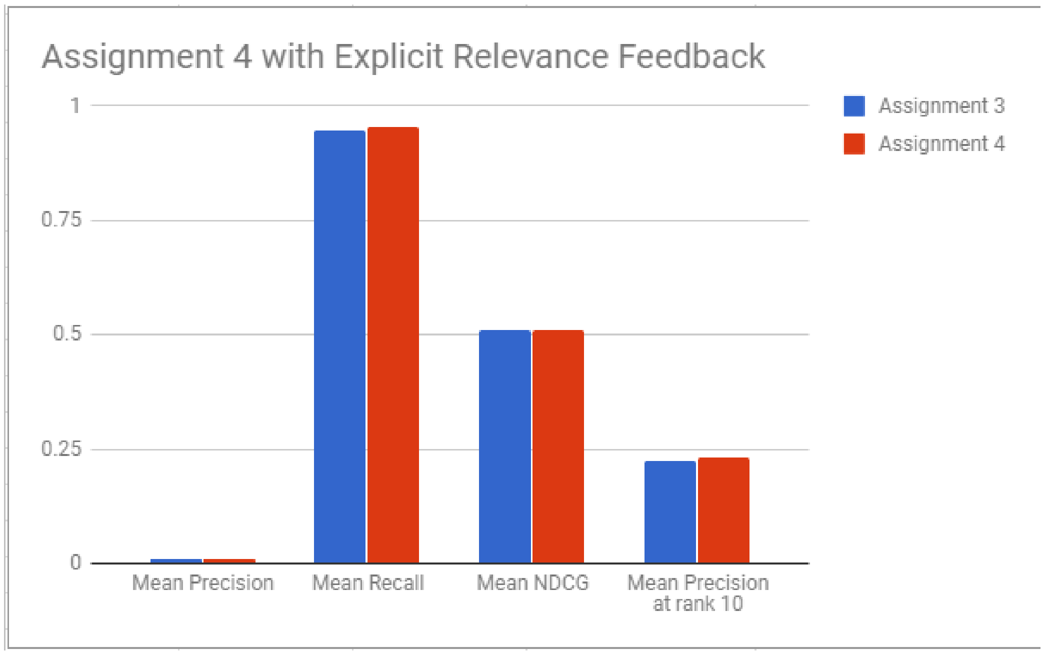

Figure 4: Results of the evaluation metrics for the explicit feedback

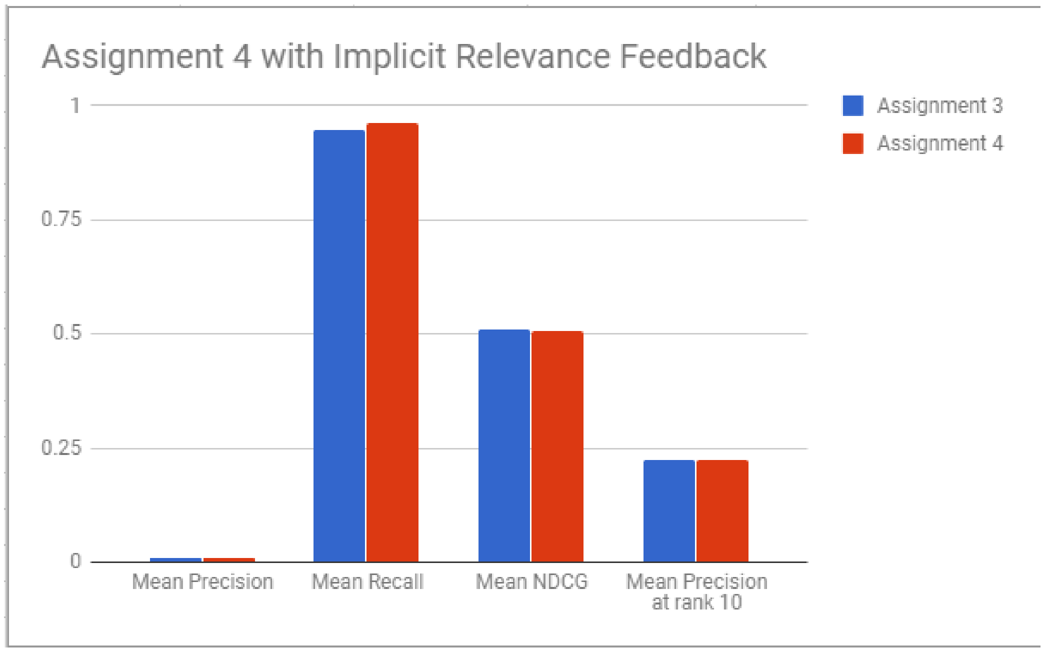

Figure 5: Results of the evaluation metrics for the implicit feedback

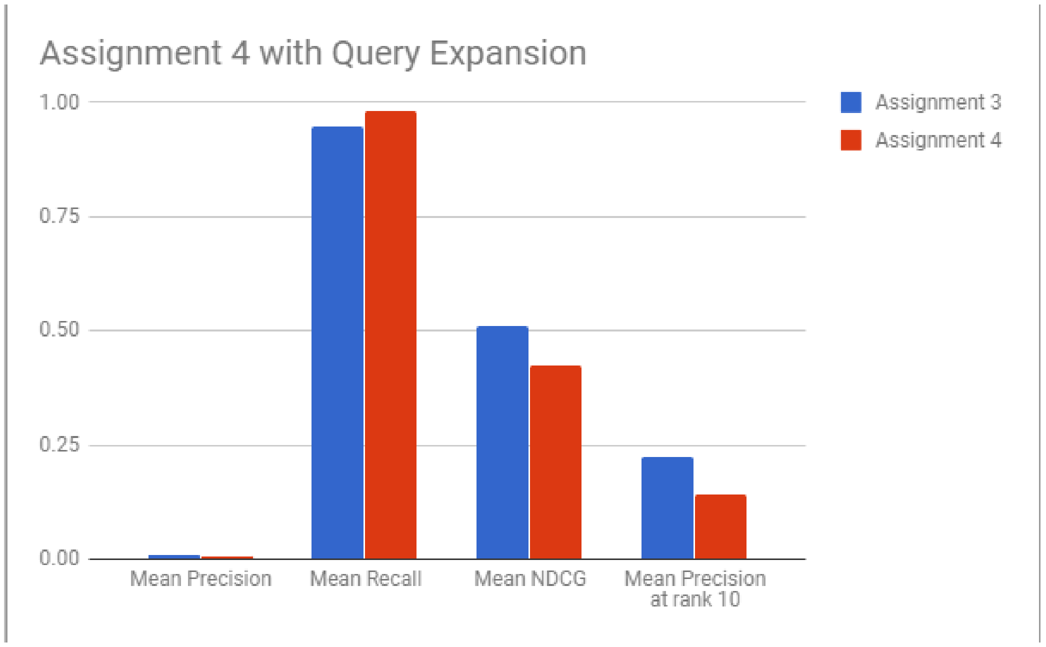

Figure 6: Results of the evaluation metrics for the Query expansion

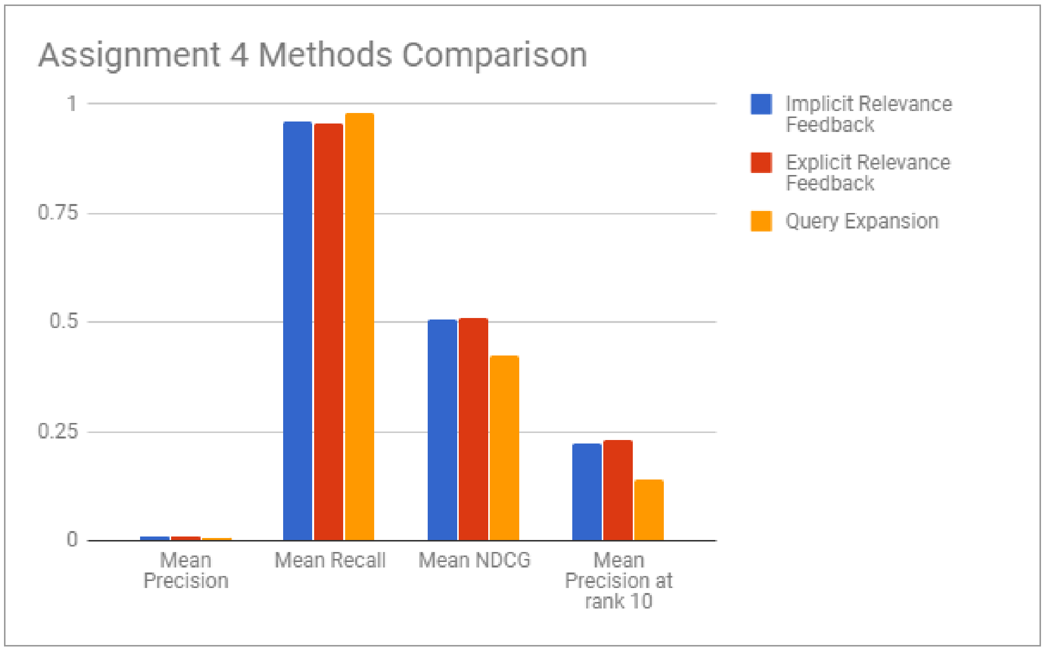

Figure 7: Results of the evaluation metrics for the Relevance Feedback methods and for the Query Expansion

Efficiency Measures

In this chapter, some measures were taken to further analyze the performance of the application. Four 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 measurements were taken by comparing two *TimeStamps* in each Main class. One was used before reading the corpus and another after the index is written into disk for the *IndexerMain* and another two were used as well, one before loading the index that is written into disk and the other one after the queries have been processed, for the *RankingMain*. Figure 8 shows the average run times for 3 consecutive runs, each column represents an average time measure for both main classes using Ranked Retrieval. It's worth noting that the application uses machine learning to perform Query Expansion, each time the *IndexerMain* is executed it takes time to train this machine learning algorithm, this time is not taken into consideration. The high run times for the *RankingMain* with Query Expansion are due to the fact that the application has to load the trained algorithm to expand the queries.

Maximum amount of memory

This measurements were taken using a profiling tool from IDE Netbeans[1]. Figures 9, 10 and 11 show the different memory profiles for the different main classes, with Query Expansion and Relevance Feedback.

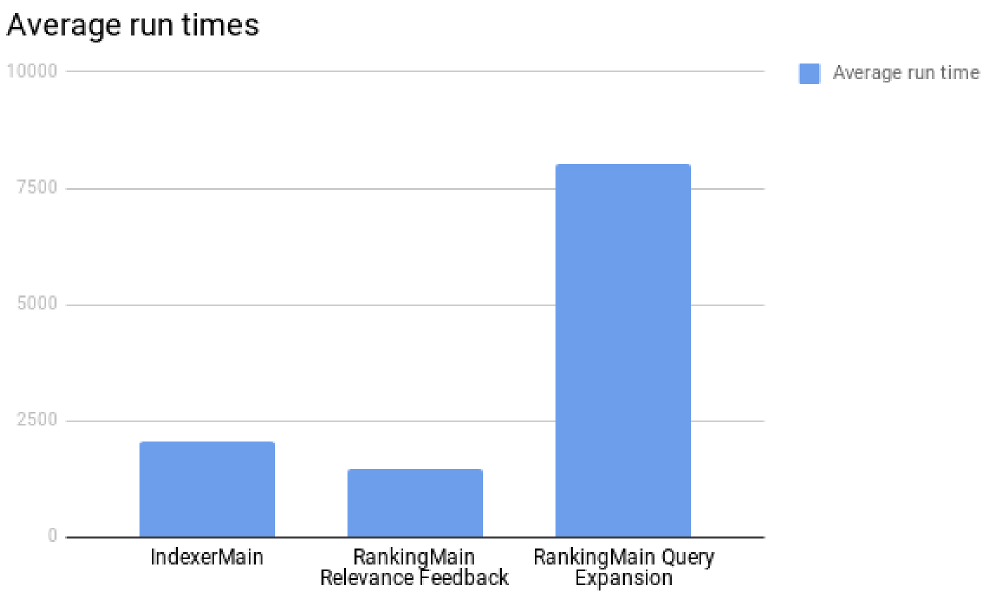

Figure 8: Average run times in milliseconds

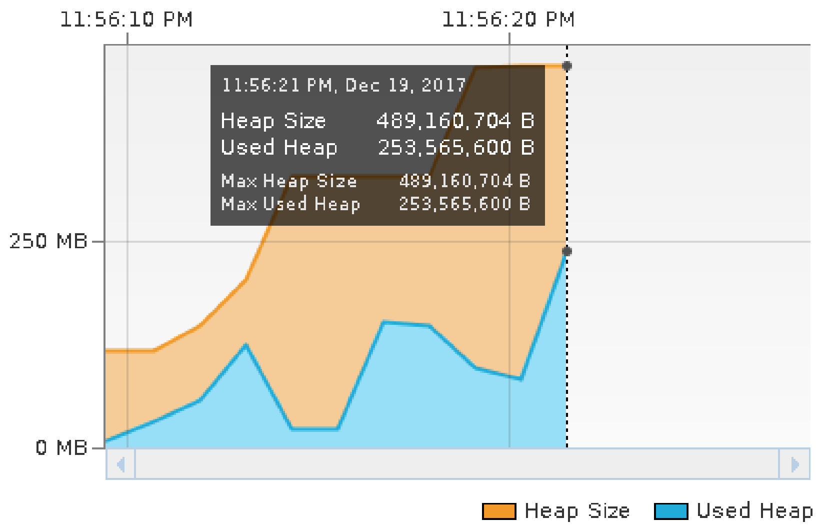

Figure 9: Memory profile for an index built with a *ComplexTokenizer* with stemming and training a word similarity model for query expansion

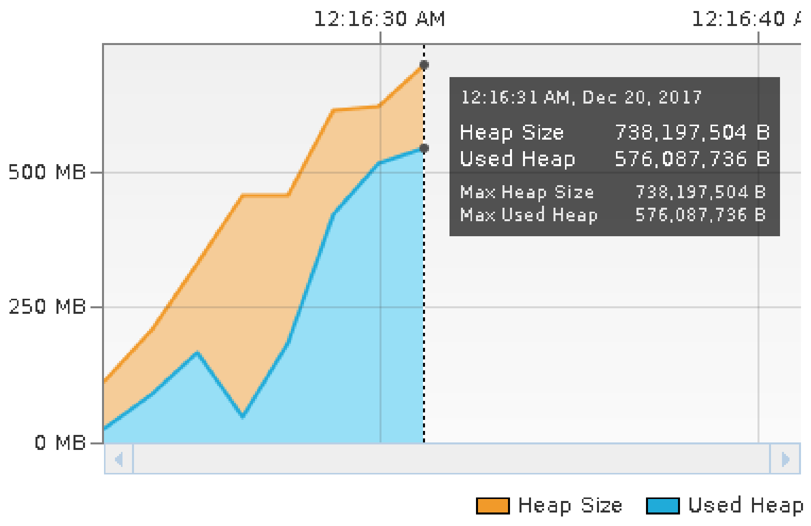

Figure 10: Memory profile for the *RankingMain* with Query Expansion

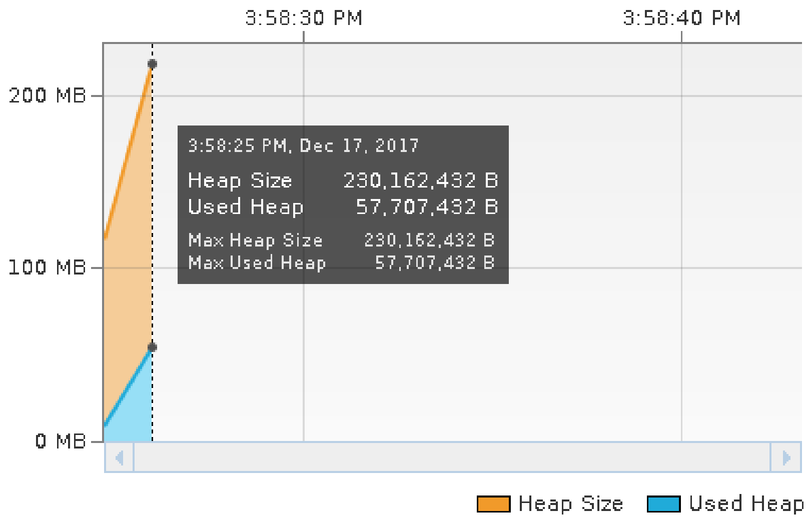

Figure 11: Memory profile for the *RankingMain* with Relevance Feedback

	Boole l
IndexerMain	1,9MB
RankingMain with Relevance Feedback	3,1MB
RankingMain with Query Expansion	3,8MB

Table 1: Disk space occupied by the index file and the result files in disk

Result space in disk

This measurement 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.

How to run

Since the developed application consists in two main classes, both can be executed, and for doing so the project needs to be imported to an IDE that supports maven integration. The recommended initial 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* from a document corpus and to save it into disk. It takes arguments like the directory of the corpus, the scoring system which consist on how the document will be scored, the file which will contain the full corpus content in the case that a word query expansion method is to be used, etc.
The other main class, *RankingMain*, loads the index from disk, processes a file that contains queries and answers them producing a results file stored by the scores of each document. Some of its arguments are the file that contains the queries, a file that contains relevance scores for the document corpus, the possible types of relevance feedback methods to be used, the query expansion usage, etc.

Conclusion

Observing Figure 12 it's possible to see that two additional main blocks were developed in comparison to the last assignment ones. First, there was a need of implementing a *Word2Vec Query Expansion* module so that the queries could be expanded according to a thesaurus that is automatically generated from the full corpus content. This allows the creation of word embeddings from the collection that are used to verify the similarity between words in order to expand the original queries. The other main block that was built was the *Relevance Feedback* module which is responsible for updating the weights of the query terms according to the results of the relevance feedback, which can be either made in an implicit way considering the top 10 documents to be relevant, or by explicitly checking the gold standard file and take into account both the relevant and the non relevant files. After this additional processing phases, in case that the user wishes to perform them, measurements of the new evaluation performance are made and 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 in general if a relevance feedback method is used. That cant be said about the developed query expansion method since the only thing that got severely higher was the recall of the produced results, that is due to how small the training set is being impossible to actually train the neural network for proper similarity estimation, but even though in this specific case it wasn't really helpful, there's systems where this methodology can be pretty useful.

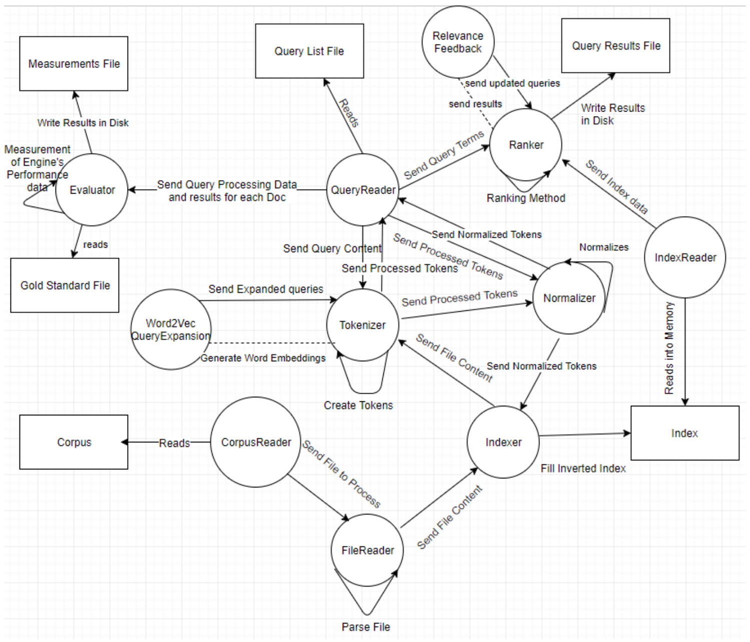

Figure 12: Dataflow diagram of the application

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^{TM} 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 - Builder Pattern*. 2017. URL: `https://www.tutorialspoint.com/design_pattern/builder_pattern.htm` (visited on 22/10/2017).

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