Group 11

Tiago Mesquita
ist86520

João Barata
ist86450

Carolina Vitorino
ist86394

1 INTRODUCTION

This project intends to deepen our knowledge in the Information Retrieval subject by developing an IR system. In this first delivery, our main focus will be on the development of the basic functionalities of the system, namely, indexing and querying functions, whilst processing the fed topic text with different techniques, (e.g., stemming , lemmatization). The evaluation of our model will be performed with metrics such as mean average precision (MAP) and mean reciprocal rank (MRR). Additionally to the proposed solutions, a fine tuned BM25 [3] model was implemented for ranking purposes. The target data is the RCV1 © corpus, provided by the course faculty. Specifically, D_{test} for ranking documents and R_{test} for evaluation purposes. The following sections detail our approach to the project, we present the results obtained, their analysis and conclude by making an overview of the work. The work was distributed equally throughout the team.

2 APPROACH

This sections details how we approached the development of each functionality of our project.

2.1 Indexing

Before implementing the indexing function, the files need to be extracted. Here, the dimension of this problem became painfully clear, as a i9900k Intel CPU took roughly 8 minutes to complete the extraction, revealing the true size of the data-set of almost 4GB. It became clear that several efficiency considerations needed to be implemented. This extraction step is fully implemented, therefore the *.tar.xz* file can be directly fed to the program.

2.1.1 Parsing. After extraction, the raw data-set becomes available, revealing the 2nd problem. The data comes in a XML format, requiring a parsing step. Here a proper python lib is used [4], filtering only the usable data (*newsitem*, *headline*, *text*, *dateline* and *byline*). This revealed the first opportunity for an efficiency optimization, since only a fraction of the document is used, XML is less space efficient and not natively supported by python. We decided to create automatically create a local json copy, with the usable data. This also doubles as the documents internal representation, D_{test}, a *dict* where each key corresponds to a *itemid*, and the values, a dict with the usable data. This optimization alone shortens the data-set size to 1.1GB, and accelerates parsing to an even greater factor.

2.1.2 Inverted Index. With the text data finally usable, we can focus on the indexing approach. For the Boolean retrieval (which we will address only as retrieval from this point forward) we opted to for an approach with a Boolean index (binary count matrix) paired with a simple TF-IDF matrix, both implemented from sklearn's library [1]. For the ranking we've opted for an implementation with whoosh [2], motivated from lab 3. The reasoning behind these choices will be discussed in sections 2.2 and 2.3 respectively.

2.1.3 Preprocessing and Tokenization. Having separate approaches for retrieval and ranking meant redundant common steps, namely pre-processing and tokenization. Despite this, as we'll see in section 4.2, this decision payed of. Another problem that arose was with the index for retrieval, since sklearn is very protective of its methods, demanding separate analyzers (preprocessor followed by a tokenizer) for both vectorizers (Binary Count and TF-IDF). We managed to circumvent this problem by analyzing the text first, rebuilding it with the processed tokens separated by spaces. This text was then fed to both vectorizers, whose analyzer was a simple *split* function, virtually immediate in execution. This way we could separate the indexing time from the analyzing time. The analyzers used follow whoosh's implementation, allowing them to be used in both models. The indexing in whoosh is very similar, however we make use of the schema functionalities to save separate fields for each usable data. Although the ambit of this project doesn't make use of this detail, if needed, it would easily support search by field, making it a nice addition. Indexing whoosh also reveals an obvious optimization: since it saves the index in memory, we can check if the index already exists, speeding up the next runs. Since we require a saved index for each supported analyzer as well as the subset of the data-set (we used subsets for testing), we opted not to store the documents in the index, as with would quickly explode in size. Instead we opted to save the documents dictionary in the index object, which we found manageable if enough RAM is provided (will be discussed in section 4.2).

2.2 Boolean Querying

In Boolean retrieval we're only interested in token occurrences, so either a token occurs or doesn't, hence the Boolean aspect.

2.2.1 Query Extraction. Since it isn't reasonable for a document to contain all words in a query, we need to filter the searched tokens, in order to maintain relevance, whilst allowing reasonable matching. Therefore the chosen tokens need to hold a lot of relevance to the query, in a sense representing it succinctly. For this task, a metric that makes sense to consider is TF-IDF: The IDF component makes sure to guarantee exclusivity, favoring words that appear in very restrict documents. Although this may seem enough, it may lead to over-specificity, failing to support synonyms, typos and domain-specific language, all found extensively throughout the provided data-set. Here enters the TF component, that values reoccurring words, directly fighting the previous cases.

In our model we transform the topic text with the TF-IDF matrix, providing a score for each word. We then pick the top-k words according to said score.

Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), N. Agmon, M. E. Taylor, E. Elkind, M. Veloso (eds.), May 2019, Montreal, Canada

https://doi.org/doi

2.2.2 Boolean Query. In order to search the extracted top-k tokens along the data-set we finally make use of the stored Boolean matrix: we start by transforming the top-k word vector with the Boolean matrix, which simply generates a sparse vector where the only values at 1 correspond the query-ed tokens. With this vector, finding the occurrences of each word in all documents is simply a matter of computing a dot product of this vector, with the Boolean matrix, returning a vector of number of word occurrences. Finding the matching documents is simply a matter of checking for each position if the occurrences match the provided tolerance ($\text{round}(0.2 \times k)$).

2.3 Ranking

Ranking with the model is pretty straightforward, since the functionality is provided by whoosh. We simply load the index and search a query, for a given scoring function.

2.4 Evaluation

For the evaluation of the IR system, the R_{test} set is used to test the performance on different ranking and retrieval tasks. The metrics implemented were the ones proposed in the project, namely, precision based measures, gain based measures, MRR and BPREF. As mentioned previously, we used a library available in python, but for some of the calculations, i.e., gain based measures and MAP, we decided to use our own functions as after a quick check-up we verified they were more reliable.

3 SYSTEM EVALUATION

To evaluate our model, we've tested a combination of analyzers and scoring functions. Since the full data-set is too big for most testing, we've created a subset, composed of all documents that appear in R_{test}, guarantying that most contain relevance assessments. We found that the size of this subset was pretty manageable and perfect for testing. For all plots presented from here forward, assume they resulted from this subset, unless stated otherwise.

3.1 Evaluated Models

We now present the different considered models to evaluate.

3.1.1 Analyzers. For the analyzers we've decided to test lemmatization and stemming since they often appear as rivaling approaches, the first one as a brute-force approach to tokenization, and the second a more ponderative and therefore expensive.

3.1.2 Scoring. For the scoring functions there wasn't much choice: the classic TF-IDF and the commonly better performant BM25, both supported by whoosh. There is however a small detail, frequently overlooked: BM25 is parameterizable and the values of $k1$ and b can hide a considerable performance boost. To tune the parameter of BM25, we've tested several metrics for both analyzers, in the eval subset. The results can be found on figure 1 for precision at $k = 10$.

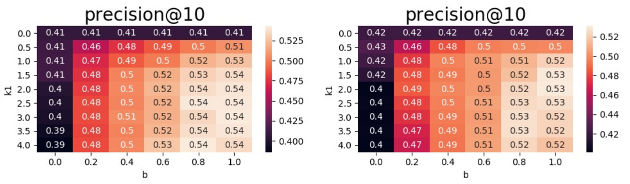

Figure 1: BM25 Precision@10 Heatmap for stemming and lemmatization, respectively

Here it becomes clear that the values $k1$ and b of 2 and 1 are the most effective. This was the case for most other metrics at different k values, showing similar distributions. Not only that, it also shows what we later found out, that stemming outperformed lemmatization, which will be discussed in section 4.7.

3.2 Evaluated Metrics

Varying the analyzer, we obtain the plots present in 2. A bit overwhelming at first, we believe that exposing all this different metrics simultaneously, allows to draw some interesting comparisons. Perhaps what is more apparent is how the precision and recall vary inversely. As we will later see in section 4.5, this inverse relation can be traced all the way to the false positives and negatives. F-β ($\beta = 0.5$ as requested) closely follows precision as expected, after an early controlled boost, showing good performance for lower values of k. MAP also vows for this early performance, starting high and later stabilizing at around 0.35. MRR stabilizes pretty fast ant at roughly 0.75 showing that our model retrieves a relevant document, in average, within the 1st and 2nd positions. BPref show the highest growth, proving that the low results for higher k values are mostly a result of the low number of assessed documents.

Comparing the 2 plots, all metrics are marginally better for the stemming approach, whilst NDCG shows a much clearer difference, growing unstably for the lemmatization approach, likely showing that the relevant documents are ranked in batches, rather than distributed throughout the top p.

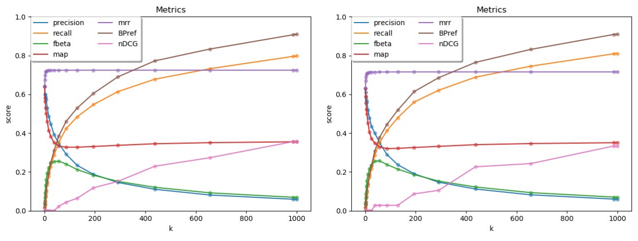

Figure 2: Different evaluation metrics varying k values, for stemming and lemmatization, respectively

4 ANALYSIS

The following section describes our proposed explanations for the "Questions to explore" section of the report.

4.1 Document collection *D* and topic collection *Q* characterization

To analyze how preprocessing affects the terms distribution, we decided to plot a simple histogram for the terms TF-IDF scores. To get them, we simply transform a vector of ones, with the size of the vocabulary, with the TF-IDF vectorizer. Doing this process for processed (with stemming) and unprocessed token resulted in the plots found in figure 3.

Looking at both plots it is clear how the terms distributions were tremendously affected, where stemming seems to introduce some sort of "segmentation" (this effect is even clearer for the full data set, available in the Appendix in figure 9). This way, there are progressively less and less tokens, the higher the TF-IDF score, having a restricted group of keywords (top scores). The non processed on the other hand, has more tokens, the higher the score, leading to a huge amount of "key words" which contradicts the concept itself. Also from the area of the plots, it's clear that the non tokenized has a much more extensive vocabulary. This 2 facts lead us to the conclusion that stemming is able to connect words otherwise separate, increasing IDFs.

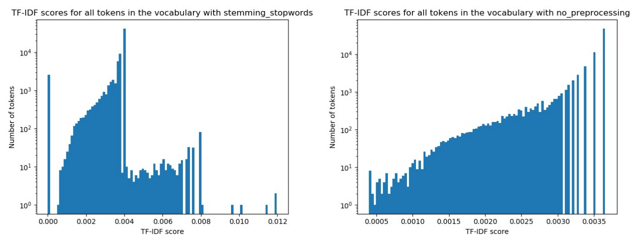

Figure 3: TF-IDF scores for the tokens in the vocabulary with stemming and stopwords (left) and with no pre-processing (right)

In the other plot (figure 4) we can see that most top Q tokens are unique. There is also a good incidence of pairs, triplets and so on, lowering in count exponentially. After the groups of 10, the are no cases, with the exception of some outliers that see extremely high repetitions. This shows that k=10 is inappropriate, leading to the inclusion of really frequent words.

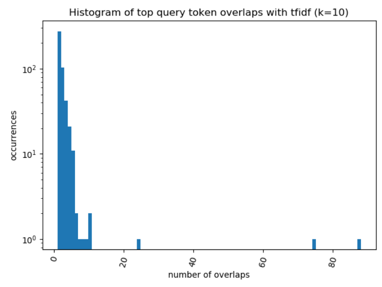

Figure 4: Top query tokens overlaps with tfidf(k=10)

Stages	***Eval**$_{stemming}$*	***Eval**$_{lemmatization}$*	***Full**$_{stemming}$*
	Time/s	Time/s	Time/s
Indexing Boolean	<1	<1	40
Pre-processing Indexing	4	8	358
Indexing TFIDF	< 1	<1	42
Whoosh Indexing	11	14	2035,5
Retrieval	4	4	513
Ranking	68	57	6684

Table 1: Time metrics for the different stages of the eval and full set

Stages	***Eval**$_{stemming}$*	***Eval**$_{lemmatization}$*	***Full**$_{stemming}$*
	Space/mb	Space/mb	Space/mb
Indexing	64.454	66.931	3900,698

Table 2: Space requirements for indexing of the eval and full set stages

4.2 IR system performance

The performance of the developed IR system is summed up in Table 1 and Table 2 with time and space metrics, respectively. Regarding the *Eval* set, we observed that for the Boolean and TFIDF indexing, the values remained equally low. However, when pre-processing is applied, the stemming approach proves to be faster when compared to lemmatization. This is due to the fact that lemmatization requires *WordNet* corpus and a corpus for stop-words. Regarding retrieval and ranking stages, our approach seemed to be efficient having reduced computational times.

When it comes to the *Full* set, the time value is several orders of magnitude larger, this is easily explained by the increased computational effort required by this set. For the *Eval* set the space required was low for both approaches, so the pre-processing seemed to have little to no effect on the space required. Regarding the full set, the space, similarly to the time, is several orders of magnitude larger. This value is also inflated due to our decision of storing the full documents in a dictionary. With sufficient RAM our approach would require less space, however, storing only the document ids in a list format would also be a viable option.

4.3 *k* value impact - Boolean Model

The number of top-*k* terms chosen to tackle this question was 2,4,6 and 8. The plots in Figure 5 do not include 8 terms as the tf-idf scores were close to non-existent in most of the topics. Between 2 and 4 *k* terms the model is able to retrieve most of the documents, whilst for values larger than the aforementioned, it struggles to do so as the toleration threshold is to low for such numbers. We conclude that between 2 and 4, a fixed threshold of 4 *k* even if only slightly, would have a better performance overall.

4.4 Precision or Recall guarantees given a specific *p*

As depicted in the graph 6, the precision and recall scores grow inversely exponential to each other. For low values of p, < 100, the number of relevant documents retrieved compared to the total number of documents is quite high but as the latter increases so does the number of outliers and, therefore, the precision decreases. Opposite to this, recall grows exponentially as p increases because

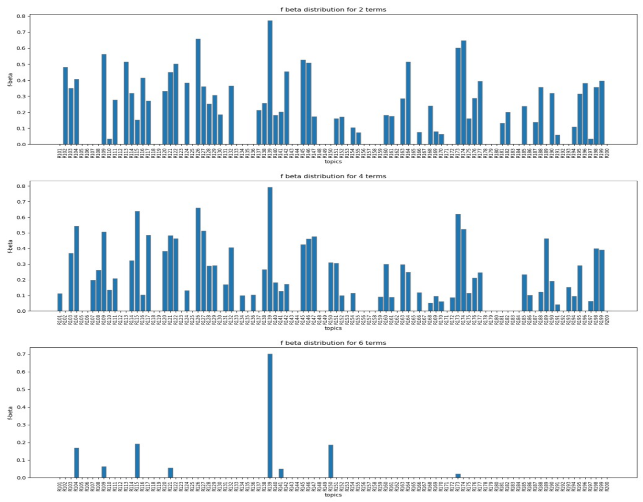

Figure 5: F-β distribution measure for top 2,4,6 terms, respectively

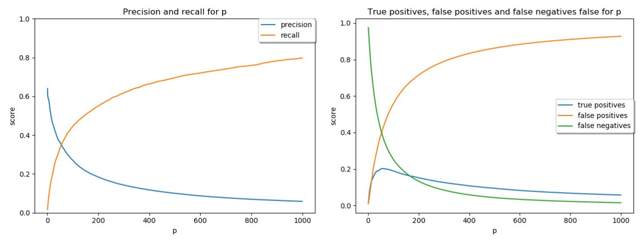

Figure 6: Precision and Recall for p (right), TP,FP and FN for p values

it does not consider the number of irrelevant documents retrieved, consequently, as the total number of documents increases the number of relevant documents available will also increase, resulting in a high recall value.

In summary, in contrast to recall, given a specific small value for p the IR system provides better a precision score.

4.5 p value to use w.r.t to user preferences

Considering the values plotted in Figure 6, intuitively, to minimize false negatives the number of documents should increase. This happens because by having more documents, the number of relevant ones will consequently decrease and so it will be less likely to obtain a false negative. On the same note, to minimize false positives, the reverse applies, by having a low value of p, the number of relevant documents ratio to the total number will be higher and thus it will be less likely to obtain a false positive. Lastly, for true positives, as it was previously mentioned, for values of $p \approx 50$, we will obtain a higher true positive score, due to the increased ratio of relevant to total number of documents.

To sum up, a small value of p to minimize false positives and maximize true positives and a high value of p to minimize false negatives.

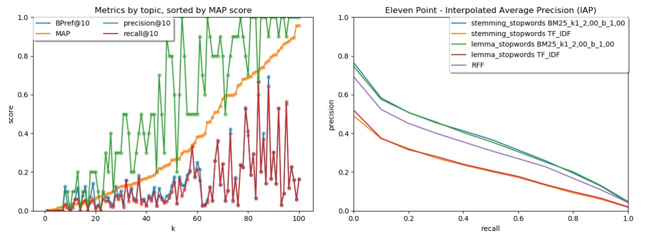

Figure 7: Metrics by topic sorted by MAP (right), IAP for the different models, using different text processing and scoring techniques(left)

4.6 Performance variation across topics in Q

In order to better see how the performance varies across the topics, we've decided to plot a common approach (7 (left)). In it, each topic is scored according to MAP, a metric known to be stable. From this metric we can then sort the topics by their respective scores, and plot other metrics. For the others, we decided the classics, precision and recall, essential to understand any minor variations, along with BPref, known to better evaluate models with insufficient assessments, like this case. The MAP alone shows how huge variance of performance, plotting an almost linear graph. Interestingly, precision and recall roughly follow the MAP. This means that the model is able to quickly retrieve most relevant documents in the first results for some topics, whilst failing to do so in others, likely to have less assessments.

4.7 Impact of different text processing and scoring options

Observing the graph in Figure 7, it is clear that BM25 outperforms TFIDF on stemming and lemmatization. Moreover, we once again observe that the difference in performance in between both text pre-processing approaches is practically null. Consequently, RRF is usually beneficial when the ensemble models both behave similarly well or complement each other. This is not observable in our models as TFIDF besides having a lower performance compared to BM25, also does not compensate the flaws in the latter, i.e., (does not retrieve documents BM25 is unable to).

5 CONCLUSION

In summary, our approaches to this project proved to be efficient, as the time values depicted in the previous section. Moreover, the IR system proved to have coherent behavior as our plots were easily explained based on our theoretical knowledge from the course. Lastly, we identified some points that could be improved, as the space required for the indexing, using a different data structure as list with the ID's could probably lead to faster look-ups.

REFERENCES

[1] [n. d.]. learn. ([n. d.]). https://scikit-learn.org/stable/index.html
[2] [n. d.]. Whoosh. ([n. d.]). https://pypi.org/project/Whoosh/
[3] Giambattista Amati. 2009. *BM25*. Springer US, Boston, MA, 257–260. https://doi.org/10.1007/978-0-387-39940-9_921
[4] Leonard Richardson. [n. d.]. Beautiful Soup. ([n. d.]). https://www.crummy.com/software/BeautifulSoup/

A APPENDIX

The following section shows the plots obtained from running with the full set, as they were not used for the experimental analysis we've decided to include them in an appendix section.

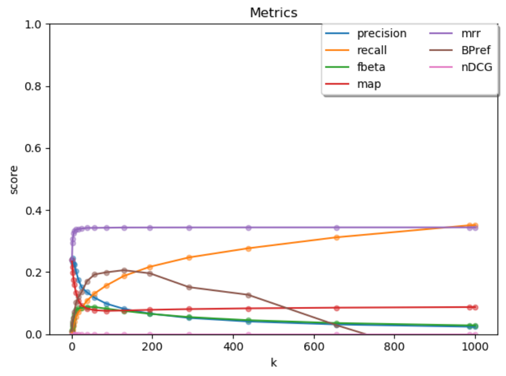

Figure 8: Different evaluation metrics varying k values

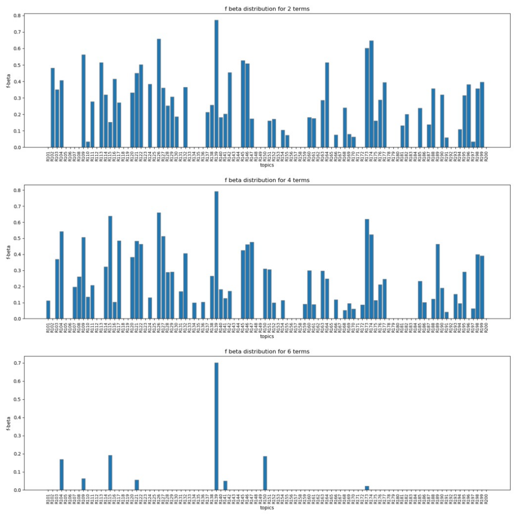

Figure 11: F-β distribution measure for top 2,4,6 terms, respectively

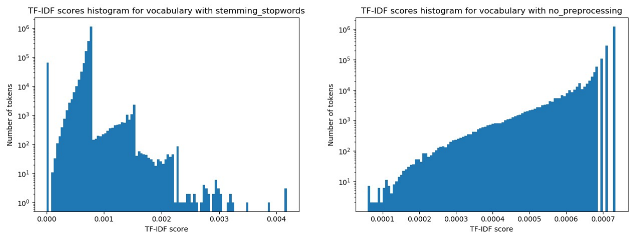

Figure 9: TF-IDF scores for the tokens in the vocabulary with stemming and stopwords and with no preprocessing,respectively

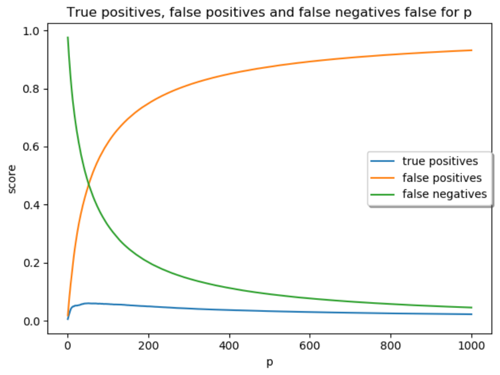

Figure 12: TP,FP and FN for p

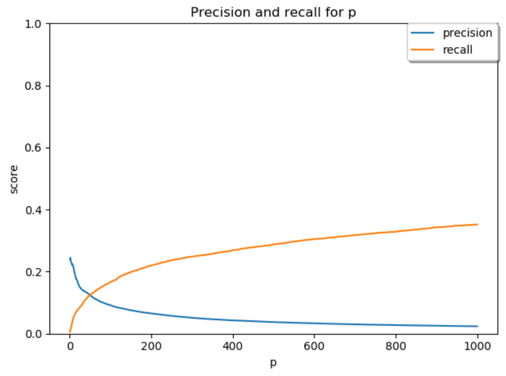

Figure 10: Precision and recall for p

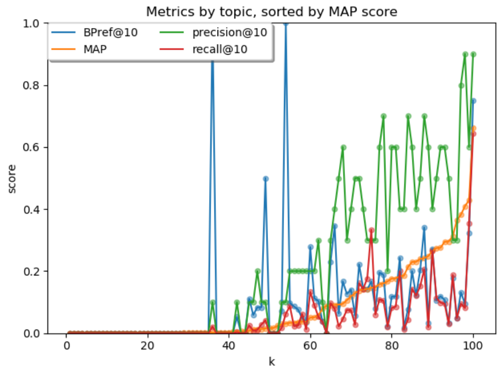

Figure 13: Metrics by topic sorted by MAP