Group 11

Tiago Mesquita
ist86520

João Barata
ist86450

Carolina Vitorino
ist86394

1 INTRODUCTION

For this second delivery, the goal was to extend the already implemented IR system considering three different approaches : *Clustering*, *Supervised*, and *Graph ranking*. In the following section we will explain the implementation for each approach and also answer the questions mentioned in the proposal. **Important Note**: The efforts for this project were only distributed equally between the group member 86450 and 86520 and therefore this should be discussed in the further demonstration of the project.

2 ANALYSIS

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

2.1 Clustering approach: organizing collections

Forms of unsupervised corpus exploration, such as formal and coherent concept analysis, can be placed to guide IR tasks. In this project, we will consider an alternative approach – clustering – to this very end. The primary goal is to understand the organization of topics in the provided topic collection, as well as the organization of documents in the RCV1 collection. In addition, the clustering system provides a way of describing the documents from a given cluster using the cluster's centroid. The focus of this part is finding a good and interpretable clustering solution.

We tested two different clustering approaches: Hierarchical Clustering - Agglomerative Clustering with complete linkage and Ward linkage, and Partitioning - K-means.

And difference distance metrics that depend on the chosen IR model: cosine (for the vector space model) and Euclidean (for the Boolean model).

2.1.1 What is the (hypothesized) number of topic clusters? And document clusters in the D_{train} collection?

To test the hypothesized number of topic clusters and document clusters in the D_{train} collection we have a clustering function that has the input with the document collection or topic collection, the clustering algorithm that we want to test, as well as the distance criteria we wish to use. Then it calculates the inertia, that tells how far away the points within a cluster are, therefore, a smaller inertia is aimed for. There is also a distortions list that contains the distances between the vector space and the model centers. We use the Knee/Elbow method to help determine the number of clusters in the data set. We tested two approaches, K-means and Clustering with complete linkage, both with the distance cosine.

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

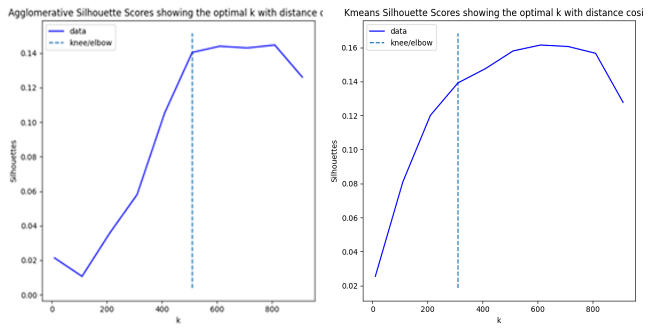

Figure 1: K-means and Agglomerative Clustering, Silhouette showing the optimal k

Analysing the above figures, we can conclude that the optimal number of clusters is 500 for the Agglomerative approach and around 300 for the K-Means approach.

2.1.2 Are the clusters from previous solutions cohesive? And well separated?

The cohesive and separation are both internal measures, cohesion is the sum of the weight of all links within a cluster, meaning how closely related are documents in a cluster. While separation is the sum of the weights between documents in the cluster and documents outside the cluster, how distinct or well separated a cluster is from other clusters. To check if a cluster is cohesive we can do it with the silhouette score, since its value measures how similar a point is to its own cluster(cohesion) compared to other clusters(separation). From the previous analysis we can conclude that since the silhouette scores are not very high, the clusters are well separated.

2.1.3 What the clustering of topic documents, Q, reveals regarding their conceptual organization and independence? Are there highly similar/overlapping topics?

To check if the topic documents are too overlapping we can do it interpreting the silhouette score, a high value indicates that the document is well matched to its own cluster and poorly matched to neighboring clusters. However, since the silhouettes calculated previously are not very high, we can conclude that the documents are not very similar to their own cluster clusters, so there isn't much similarity.

2.1.4 Given a specific cluster of topics, check whether the medoid (a sort of prototype topic for the cluster) adequately represents the remaining topics in the given cluster.

The medoid is the most central document in the cluster, it is similar to the centroid, the middle of a cluster, but while medoids are always restricted to be members of the data set, centroids are not. We can also calculate the median to check the relevance of each term for the documents in the cluster. We started by calculating the medoid in the interpret function that given a cluster and the document D or topic Q collection described the documents in the cluster considering both median and medoid criteria, and delivers a description in the output. If the medoids are too different from the cluster, the medoid doesn't represent the remaining topics in the cluster.

2.1.5 How are the documents in the target collection organized? Briefly discuss the importance of this information to understand the behavior of the target IR system.
Clustering allows similar objects to be grouped together based on common attributes, with it we can test the collection, which helps us understand how the target IR system works. From the evaluation and interpretation of the target collection we can conclude that the clusters are well separated and not very similar to each other.

2.2 Supervised approach: incorporating relevance feedback

For the classification approach, we've decided to test KNN and MLP as possible classifiers. With the first one, we hope that it is able to capture some of the inner structure behind document relevance, present in the vectorial space. In essence, documents relevant to the same topic, should share same of the terms and likely their scores in frequency metrics, a relation that can be explored by a distance metric. This is the same intuition, behind kmeans clustering, making both approaches deeply intertwined. This model is also the cheaper option provided, making it an excellent candidate for running tests and tuning hyper-parameters. On the other end, there is MLP, potentially the most expensive model, limiting hyper-parameter-tuning, but likely also the most capable. Contrarily to KNN, this model is more than capable in finding relations outside the limits of a distance metric, being limited only by its layer's depth. We hope that this capability allows this model tho achieve significant improvements over KNN and the base IR model.

The relevance classification task is trivial in its approach: train the classifiers with D_{train}, tasked with classifying a document as either relevant or irrelevant for each topic. After training, we test them on D_{test} and evaluate the results as a classification task (no ranking aspects). This type of model is functionally pretty similar with the Boolean retrieval task in the first delivery, where the IR model was tasked with finding the relevant documents, within a selection. Although previously we've only considered the relevant documents for evaluation purposes, as rightfully unretrieved documents (true negatives) weren't evaluated, we now assume them and evaluate accordingly. Furthermore, we only consider assessed documents as opposed to the previous delivery, where the model retrieved from the entire collection (in actuality we still retrieve from the entire collection, but filter the unassessed documents afterwords, but is virtually equal for the Boolean retrieval model). In this conditions, we assume that the comparison between Boolean based and relevance feedback based retrieval is fair.

Ranking, however, is an whole other story. From the 2 suggested approaches, ranking with probabilities felt like the most appropriate: classification models are often more complex than common indexing based IR models, therefore holding a lot of potential for good performance. Not only that, the confidence (probability of the labels) of the classification model is considered, whereas in the other suggested approach, cases where a classifier discards a document with near 100% certainty, or barely above 50%, have the same conclusion. Obviously this could be accounted for, but the solution would likely involve using a confidence threshold, arguably inelegant. Applying Occam's razor, the probability ranking is then the preferred approach.

There is still the issue of comparing with the baseline ranking (first delivery), that is applied over the full eval set of documents (assessed documents for any topic). One way to compare them, is to use the new model to rank the entire eval set. As we quickly found out, the added complexity really shows, namely in the MLP based classifier, making this approach prohibitively slow. We ended up following the same approach we've used for the Boolean retrieval, but with the top-k ($k = 1000$)documents for each topic, rather than the assessed set. From our knowledge, this is the best option: firstly, k is a manageable number of documents, allowing us to quickly run a variety of tests; k is directly retrieved by the baseline IR system, assuring a fair comparison whilst providing a good balance between assessed and unassessed documents; the baseline model also relies on the same vectorizers, assuring that the retrieved documents have interesting features for the classification model; this is an historical [3] and accepted approach, often called *reranking*. This approach is now extremely relevant, as ranking tasks are now dominated by Deep Learning models [1], extremely effective, but equally extremely expensive ranking models (adapted from relevance classification just like in our case), that rely on a simpler IR model, namely BM25, to perform an initial retrieval.

Finally, moving onto the experiments, we assume that the findings in the first delivery, regarding pre-processing still apply, so in every experiment, simple stemming is used, with removal of the most common stop-words. The first step in order to properly utilize the relevance feedback is to assure the classification models are given the best conditions. In our case, this uncertainty lies not only on the classifiers' hyper-parameters, but also in the vectorizer used to convert the documents. Since both classifiers work decently within default parameters, we decided to test the vectorizers first, separately for each model. We tested the suggested vectorizers (except IDF as it does not vary for each document). For TF and TF-IDF the approach is trivial: this metrics are already meant to be used as a means for translation of documents, into a vectorial sparse representation, and therefore we simply make use of existent implementations, namely *sklearns*'s vectorizers module. BM25, on the other hand, ain't as trivial. BM25 is used as a scoring function, unlike the previous both which rely on a separate function, such as cosine similarity, for retrieving a document score. As such, BM25 does not have a proper vectorial representation. The process we took to address this, is very similar to the normal algorithm, but with 2 key distinctions: Rather than summing the computed scores for each term in a query, we append them to form a vectorial representation. We also consider every single term, rather than only the ones present in the query, allowing us to achieve a sparse vectorial representation, uniform in structure for any query (the same structure of the TF and IDF vectors).

Originally this 1st experiment was meant to find the most promising vectorizer, and with it, tune the hyper-parameters of each classifier. This tuning was performed with *sklearn*'s GridSearchCV which was surprisingly efficient. Given this fact, we opted to test tuning with each vectorizer, making the first experiment obsolete. The tuned parameters were the suggested: for KNN, we varied the number of neighbours between $\{1, 3, 5, 7\}$, which are pretty common values for KNN approaches; distance metrics, specifically euclidean, the common staple of KNN and Manhattan, usually overlooked from its simplicity, but we believe it can be quite powerful in text applications, since it directly corresponds with the frequency of different terms (for TF at least). For MLP we opted to vary the number of layers between 1 and 3 and considered different compositions, with $\{25, 50, 100\}$ numbers of hidden nodes, values we found manageable in terms of running time. The results of this experiments can be found in the following plots:

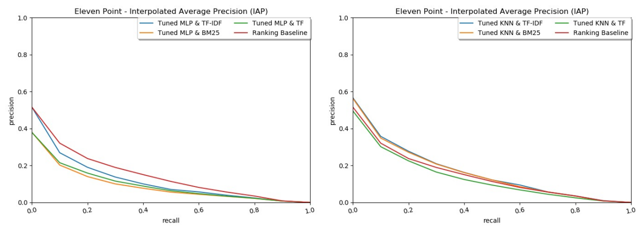

Figure 2: Ranking IAP for a tuned MLP with the 3 vectorizers (LEFT) and for a tuned KNN with the 3 vectorizers (RIGHT)

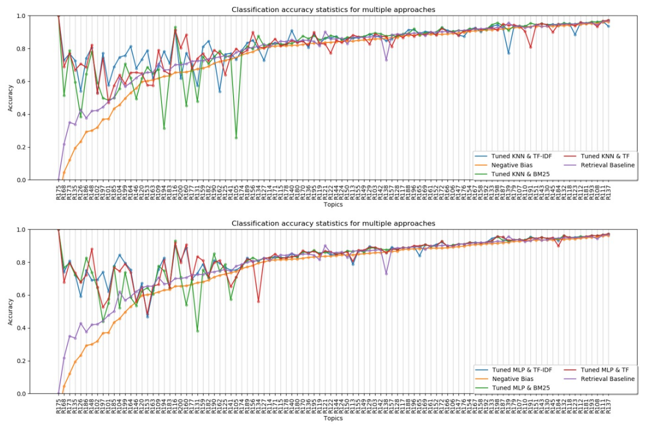

Figure 3: Classification accuracy for a tuned KNN with the 3 vectorizers (UP) and for a tuned MLP with the 3 vectorizers (DOWN)

2.2.1 *Does the incorporation of relevance feedback significantly impact the performance of the IR system?*

Following the conclusion of the earlier experiments, we tested retrieval for both classifiers with TF-IDF and compare their results in the plots present in figure 4. Unsurprisingly, both tested model manage to improve the accuracy of the IR retrieval model, across most topics. This is particularly noticeable in topics with few related documents, where the baseline retrieval model shows poor performance. This is due to its natural negative bias, present in

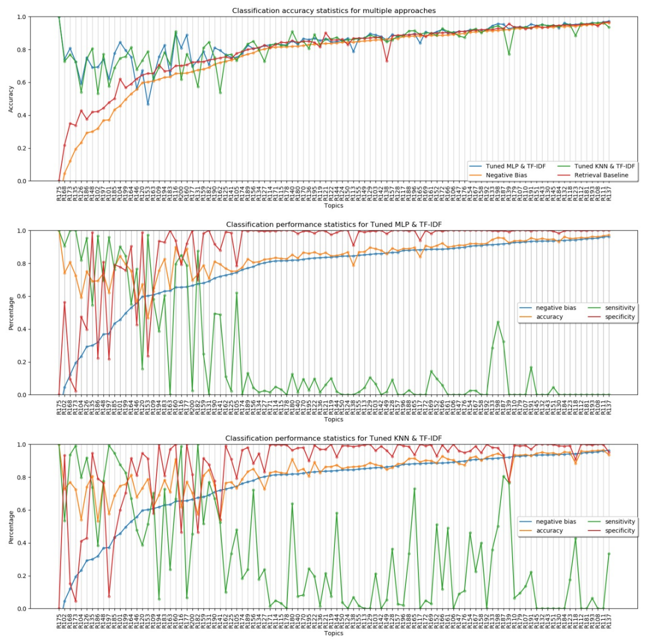

Figure 4: Classification accuracy for a tuned KNN and MLP with TF-IDF (UP), classification statistics for each model individually, MLP (MIDDLE) and KNN (DOWN)

its restrictive conditions for retrieval (only 0.2 tolerance for term matching). This constraint is crucial for ignoring the vast amount of unrelated documents but results in a lot of false negatives. The classification models benefits from the relevance feedback, that is able to convey a decent notion of both relevancy and irrelevancy in the labeled data, resulting in the appreciable improvements. This better relevancy detection is also clear in the sensitivity plots, showing not only a massive raise in the aforementioned positive topics, but even in mostly negative ones, namely KNN.

Since relevancy detection is crucial in any retrieval system, we can confidently say that relevance feedback has a significant impact.

2.2.2 *Are performance improvements approximately uniform across topics? Are there topics substantially harder to classify?*

Here we can also leverage the previous plots (figure 4) to retrieve some conclusions. As already mentioned, the biggest improvements were clearly on the topics with mostly positive documents, given the shortcomings of the baseline model. Despite this, there is a lot of variation in the improvements, reaching very different levels of accuracy (differences of >0.3). In the negative topics, there were mostly improvements, but some topics ended up loosing some accuracy. Here, the variations were much lesser (differences of <0.1). Overall, we can explain this differences by how the topics' text differ from their relevant documents.

2.2.3 *From the tested classification variants (algorithms and parameterizations), which one yields better performance for simple retrieval?*

As previously stated, even though it isn't clear wich model achieves better accuracy, having high variation, KNN achieves better sensitivity, a trait desirable for the IR system. As such we will be focusing only on this classifier, for the sake of brevity. We've tested the model in each of the configurations described in the introduction of this section. The results of this experiment are found in the plots of figure 5, showing the classification accuracy for different topics, for each of the hyper-parametrizations.

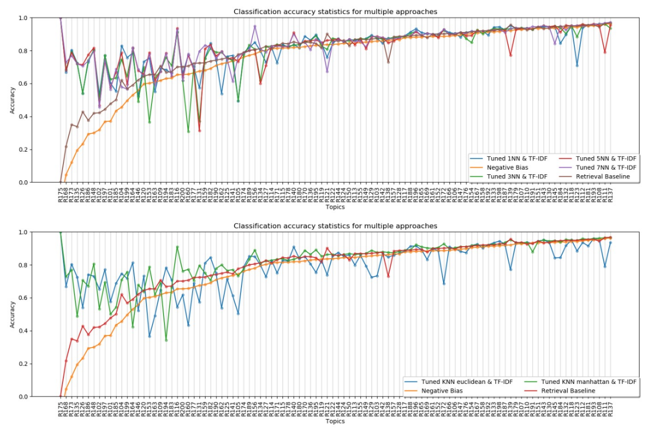

Figure 5: Classification accuracy for different parametrizations of KNN: k-neighbours (UP) and distance metric (DOWN)

Relative to the considered number of neighbours it does not appear to a have a general conclusion: different values of K have in average very similar accuracies, having slight disadvantages on different topics. This is sort of unexpected given the simplicity of the model, but it is pretty welcome, since a dynamic parametrization (for each topic), like GridSearchCV, is able to use it effectively, cancelling eachothers shortcomings. This is also supported by the ranking task, where all the tested variations resulted in similar IAP plots. Relatively to the distance metric, surprisingly, Manhattan seems to take the lead as euclidean has massive accuracy reductions in a lot of topics. We believe the secret here is in it's simplicity, whereas the euclidean distance does not have a proper interpretation in a text sparse vectorial space, Manhattan simple reflects the rate and amount of different topics, maintaining the underlying IR metrics.

2.2.4 *Does the extension of the classification setting towards ranking yield significant performance improvements?*

Finally, we can take a look at the ranking metrics. Here we've seen tf-idf success in classification translating well into a successful ranking, surpassing the other metrics. As such we'll only present plots with this metric, facilitating comparisons. In figure 6 we can see the performance of our ranking baseline and the IAP for our approaches. The results are pretty surprising: KNN is able to effectively improve the ranking, resulting in better precision across the entire range, whilst MLP ends up having an harmful interaction. This for us, is a clear symptom of overfitting, as MLP models are very prone to this. Since the trainning data is very small, particularly in some examples, this is bound to happen. We believe that with enough data and tuning, the MLP would be a viable approach for this problem. Moving on to the KNN, here the results are perhaps a bit tougher to explain. There are however some clues as to what is happening: Looking back at the classification results, the sensitivity was much higher, a the cost of a slight reduction in specificity. This is relevant for 2 reasons: whilst specificity is useful at pushing irrelevant documents to the bottom of the rank, sensitivity is crucial at getting the relevant documents to the earlier positions, arguably much more important. This higher sensitivity is also a possible result of proper fitting, where the model was able to combat the overwhelming unbalance of negative cases. There is also a hidden feature of KNN that we believe was perhaps the most crucial. In a MLP, the confidence (labelling probability) of the model is often a complex value, resulting from the complexity of the model. This essentially assures that every single document has it's one unique score, which guarantees that the new ranking is fully generated (the prior ranking is not preserved). In KNN however, the simplicity of the model transpires to it's probability: the confidence simply reflects the degree of accordance in the neighbours. So for instance, for a $k = 3$, we have only 4 possible probabilities, correspondent with each possible scenario. This means that for the most part, the prior rank is maintained, but it's segmented according to each case. This is, to our knowledge, an effective way of combining relevance feedback with the underlying IR model, where both contribute almost symbiotically.

We also present the other ranking metrics in figure 7, but they mostly just echoed what we've concluded.

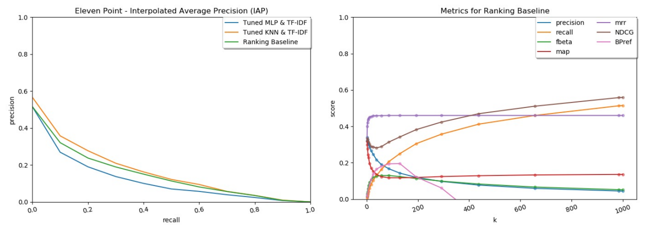

Figure 6: Ranking IAP of the best KNN and MLP classifiers (LEFT) and Ranking metrics for the baseline (RIGHT)

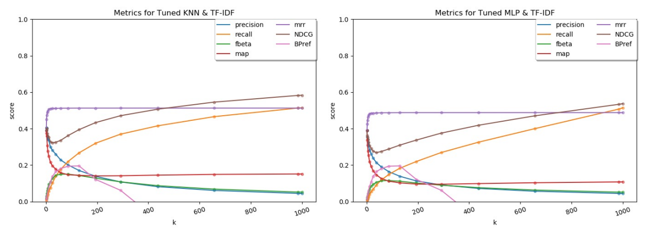

Figure 7: Ranking metrics for the best KNN (LEFT) and MLP (RIGHT) classifiers

2.2.5 *Does the incorporation of additional features aid the behavior of the target IR system?*

Here the experiments were trivial: given the vectorial nature of the IR models, simply append them to test their interaction. We've tested all possible combinations, resulting in the plots in figures

8, for ranking, and 9, for classification. Without entering into much detail, in general the results of the composed models at best matched the best of their intervenients, usually resulting in an overall score reduction. This was expected, as the 3 IR metric have the exact same basis, TF, and as such don't really had much information content to the model.

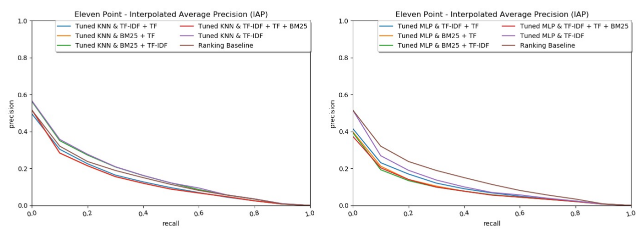

Figure 8: Ranking IAP for a tuned KNN (LEFT) and for a tuned MLP (RIGHT) extension for multiple IR models

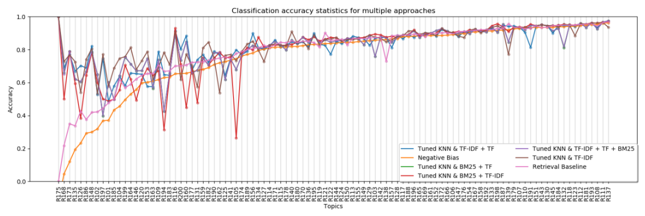

Figure 9: Classification accuracy performance forKNN with the extensions for multiple retrieval models

2.3 Graph ranking approach: document centrality

The graph-based ranking system consisted mainly of methods based on representing documents as a graph, where nodes correspond to documents, and where edges encode relationships between documents. For instance, an edge between two nodes can encode the similarity between the vector representations of two documents using the cosine measure. The graph was built aided mostly by the *NetworkX*[1] package. For the project we had to firstly implement an unweighted version of the PageRank algorithm, in which we decided to use the existing implementation from the aforementioned package, with a small twist. In order to meet the constraints of the project, namely the number of iterations we changed the source code of the package slightly. Our thought process behind this was to have an existing solid base of the algorithm but also flexibility to change it to meet the project goals.

The graph construction consisted of several phases: vectorizing the data using the tfidf-vectorizer from *sklearn*, constructing the similarity matrix with a desired pairwise similarity metric (e.g., cosine similarity), and constructing the graph directly using the similarity matrix between the documents, this similarity matrix acted like an adjacency matrix where the entries represented the weight(similarity) between nodes(documents). The values that did not meet the threshold were assigned with zero and therefore were not added as edges in the graph construction.

Regarding the implementation details, to answer the following questions, a 0.5 threshold *th* was fixed. The similarity metric used was the cosine similarity:

$$\cos\varphi(A,B) = \frac{A \cdot B}{|A| \cdot |B|} \tag{1}$$

Using the available implementation on *sklearn*[2], the values oscillate between 0 (the least similar) and 1 (most similar), with this in mind we aimed to remove the lowest similarity edges but at the same time keeping a decent amount of connections (a high *th* value would remove a large amount of nodes).However, as we will further demonstrate, this was not the *th* value that achieved the best performance.

For each of the queries, given a topic q the judged docs (from D_{test} eval set) for that specific topic were used as D to create the graph. Essentially, a graph was constructed with the documents for each query and ranked with the PageRank algorithm.

Further extensions as the usage of prior probabilities were implemented and will be detailed in the following questions.

2.3.1 *Does the graph ranking method based on document similarity aid IR?*

For this question, the approach was to plot a graph (see Figure 10) in order to compare the performance of both systems (baseline and graph IR). Firstly, it is important to notice that in this case the document set used was the $top-k$ documents retrieved from the baseline system where $k = 1000$. Furthermore, the approach taken to build the graph was similar to what was previously mentioned. On another note, besides plotting the "Vanilla" version of the PageRank, a priors vector was used, which values consisted of a normalized ranking score for each document/query. The differences between the performances are seen below.

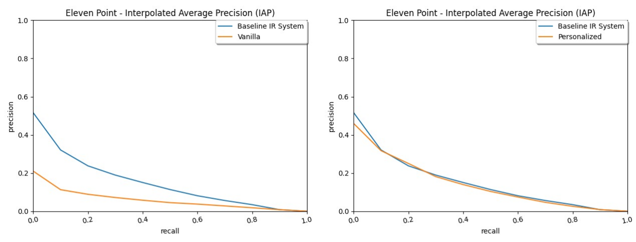

Figure 10: Baseline and Vanilla PageRank performance (left), Baseline a Personalized PageRank performance (right)

We can thus observe that even if only slightly on the personalized version of the PageRank, the baseline IR system outperforms the graph approach on both plots. Remembering the PageRank thesis "A page is important if it is pointed to by other pages.", in this case a document would be important if it had a large amount of links, so the higher the similarity score a document had with his neighbours, the higher the number of edges, ultimately increasing the importance of that document. We hypothesize that, even though we believe the ranking scores produced could indeed

[1]https://networkx.org/.

[2]https://scikit-learn.org/stable/

be better with the PageRank algorithm, as the only metric we are assessing is the binary relevance ("relevant" or "not relevant) of the document, another quality metric (e.g., the order of the retrieved documents) would be more suitable for an algorithm like PageRank which is primarily used in web-search engines. Lastly, we would also like to point out that a different threshold that the one used (0.5) could possibly yield a better result for the PageRank.

2.3.2 *How does ranking performance vary with θ?*

In this question, the approach taken was to plot a precision-recall curve like previously, but with different th (varying from 0 to 1 with 0.1 interval), whilst also showing the performance of the MLP classifier from the previous classification section as a baseline (see Figure 11 below).

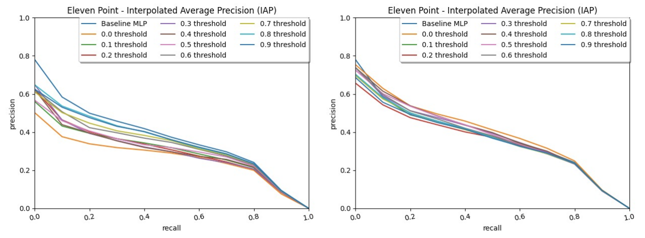

Figure 11: Vanilla PageRank performance analysis with θ variation (left), Personalized PageRank performance analysis with θ variation (right)

It is observable that the baseline classifier outperforms (slightly in the personalized PageRank) each iteration of the th. For the Vanilla PageRank, bigger thresholds resulted on a better performance and lower thresholds on a worse performance. Having a low threshold would increase the number of low similarity documents connections and therefore the number of edges, consequently decreasing the performance of the PageRank, whilst a high threshold would keep only the very similar documents connected, removing most of the connections and ultimately, increasing the PageRank performance.

For the Personalized version, the priors used were the classification probabilities obtained from the MLP classifier. In this version, the threshold variation did not seem to have a large effect on the performance, because of the introduction of weighted connections and priors probabilities on the documents. Curiously, we can actually observe that in this case the "best" performance was achieved with a 0.0 threshold, reinforcing what we said previously. The presence of priors and weights diminishes the importance of th values, because besides each node having a pre-existing importance factor(prior) even if we keep low similarity documents connected their connection will be weighted as almost insignificant when computing the PageRank formula.

2.3.3 *Upon the analysis of the graph, which are the documents with higher graph centrality score?*

For this question the full D_{test} was used to construct the graph. We then took the top-k centralities and their respective nodes(document ID's). What we found is that, for $k = 10$ the centrality scores spawned from 0.13 to 0.09. The highest centrality node (document) was '218339' which we found out it was about "VW under pressure to reach deal on GM suit", curiously the second document "FOCUS-VW, Opel square off again in Lopez fight", which related to the same subject.

The documents with higher centrality were the ones that related to this subject concerning the law-suite against Volkswagen from General Motor, which we hypothesize that would have been a big deal at that time. We also found that the average centrality of the graph was 0.01, in which we can conclude that there is a large disparity of "importance" throughout the document "network".

2.3.4 *Does the inclusion of non-uniform prior probabilities yield performance improvements?*

For this question we firstly plot a precision-recall curve to compare the performance of both approaches. The priors vector used in this case consisted of the probabilities scores obtained from the MLP classifier, similarly to the previous question. We can observe from the plot below (see Figure 12) that the Personalized PageRank outperforms the vanilla version.

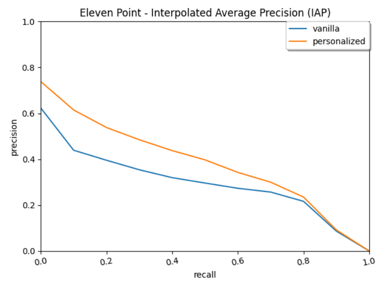

Figure 12: Vanilla and Personalized PageRank comparison

We hypothesize that the introduction of prior knowledge in each of the document improves the performance of the graph IR system as it guides the PageRank to assign a low score to low "relevance" documents, essentially filtering out the insignificant nodes. Without any kind of prior knowledge, the algorithm simply assumes all the documents have the same importance (uniform probability), having only the number connections as a way to assess the ranking of a node. We conclude that the addition of priors has indeed been a way of improving the PageRank algorithm as it has been studied in several research papers [2].

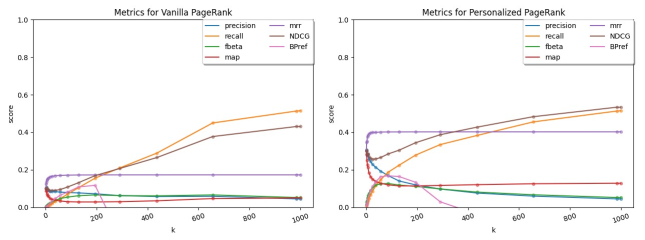

Figure 13: Vanilla PageRank performance (left), Personalized PageRank performance (right)

The plot above (see Figure 13) shows a detailed version of the performance comparison between the two PageRank approaches.

REFERENCES

[1] Jimmy Lin, Rodrigo Nogueira, and Andrew Yates. 2020. Pretrained Transformers for Text Ranking: BERT and Beyond. (2020). arXiv:cs.IR/2010.06467

[2] Qi Liu, Biao Xiang, Nicholas Jing Yuan, Enhong Chen, Hui Xiong, Yi Zheng, and Yu Yang. 2017. An Influence Propagation View of PageRank. *ACM Trans. Knowl. Discov. Data* 11, 3, Article 30 (March 2017), 30 pages. https://doi.org/10.1145/3046941

[3] R. F. Simmons. 1965. Answering English questions by computer: A survey. (1965).