Information Retrieval

2022-2023

Retrieval of Computer Science academic documents applying Neural approaches

Simone Benitozzi - 889407
Mattia Beolchi - 844911
Vincenzo Vommaro Marincola - 878533

January 24, 2023

1 Introduction

In our information retrieval project, we have chosen to work with the "computer science" dataset. In the first part of the project, we focused on analyzing and preprocessing the dataset and then creating an index. This involved cleaning and formatting the data, as well as organizing it in a way that makes it easy to search and retrieve relevant information.

In the second part of the project, we have implemented both base models and neural models for retrieval. Base models such as tfidf, BM25 and Dirichlet Language Model are implemented to retrieve the relevant documents from the dataset. We also implemented Neural Models such as Neural Information Retrieval Model and Neural Ranker for this dataset. These models use machine learning techniques to improve the accuracy and relevance of the search results. Overall, our project aims to enhance the effectiveness of information retrieval from the "computer science" dataset by implementing various retrieval models.

2 Test Collection - Analysis of Queries and Documents

Working with a large dataset, like "computer science" dataset involves to have a lot of queries and relevant documents. Specifically, we are utilizing a benchmark collection of 4,809,684 documents, with 399,565,930 tokens, and a test set of 6497 queries to evaluate the effectiveness of our information retrieval methods.

The test set includes a total of 108,636 relevant documents. This means that on average, there are 16.72 relevant documents per query. Additionally, our queries include a total of 58,881 words, with 6291 of those words being unique. The average number of words per query is 9.06.

After the preprocessing phase, the word cloud in Figure 1 was created. It contains the most common terms in the query corpus. It wasn't possible to create the same for the documents collection, due to its impeditive size and number of tokens.

The Figure 2, on the other hand, shows a more detailed insight on the term occurrence in queries, with a plot representing the terms with at least 150 occurrences.

Figure 1: Queries corpus word cloud

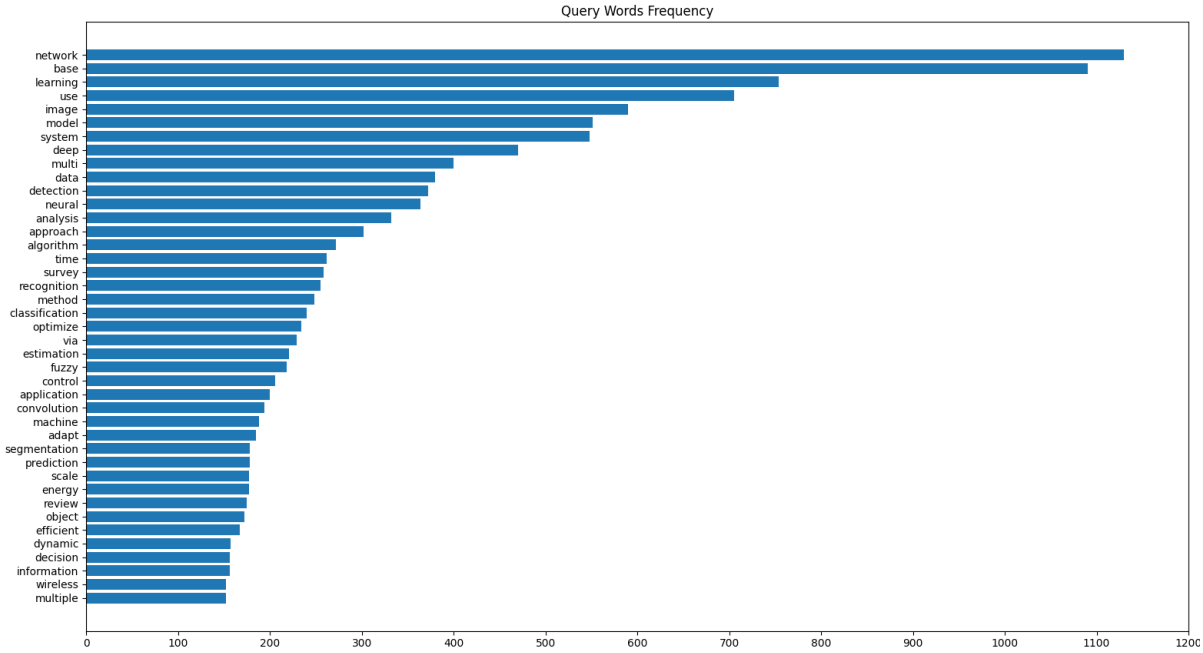

Figure 2: Queries corpus frequence plot

2.1 Data Preprocessing

In the Data Preprocessing step we have done some operation in order to have documents more easy to retrieve. Documents and queries were preprocessed and standardized with the techniques that use regex and NLTK library:

- Lower case folding
- Strip acronyms
- Remove punctuation
- Normalization
- Remove links and html tags

- Remove extra whitespace
- Remove stop words of English vocabulary
- Stemming

Since we had to index and execute queries on both title and text without distinction, in the preprocessing phase of the documents we decided to preprocess both fields and concatenate them together in a single field called *"contents"*. As for the tokenization we have decided to work with unigrams, since we won't need adjacency and proximity operators in the queries and retrieval methods.

Example of *preprocessing*:

Before pre-processing

"{*'title'*: 'Perfect Reconstruction AM-FM Image Models', *'text'*: 'For the first time, we present an AM-FM image model that, in addition to being remarkably consistent with human visual perception, also provides perfect reconstruction of the image and is thus suitable for synthesis as well as analysis applications. We employ a non-separable coiflet-based wavelet filterbank with channels that are both orientation selective and jointly localized. The analysis responses define the AM-FM image components, which we demodulate analytically using a new, high-quality 2-D phase unwrapping algorithm coupled with a spline-based phase model. The amplitude and frequency modulations obtained with this approach correspond remarkably well with human visual perception of the salient image structures, suggesting that this invertible model could form the basis for a general theory of image processing in the modulation domain.'}"

After pre-processing

"{*'contents'*: 'perfect reconstruction amfm image model first time present amfm image model addition remarkably consistent human visual perception also provide perfect reconstruction image thus suitable synthesis well analysis application employ nonseparable coifletbase wavelet filterbank channel orientation selective joint localize analysis response define amfm image component demodulate analytic use new highqual 2d phase unwrapp algorithm couple splinebase phase model amplitude frequency modulation obtain approach correspond remarkably well human visual perception salient image structure suggest invert model could form basis general theory image process modulation domain'}"

At first it was thought not to use preprocessing for queries. Having analyzed them empirically, there seemed to be no need. During the testing process, however, a problem arose with a symbol that raised an exception. We therefore decided to preprocess the queries as well. We used the same functions as document preprocessing.

3 Search Engine - Basic Search

After analyzing and preprocessing all the data the following steps are implemented:

- Data Indexing
- Base Models Implementation
- Neural Implementation

3.1 Data Indexing

The Indexing phase has been executed using the PyTerrier library. The full original collection has been indexed just to extract more efficiently some corpus and tokens statistics (described in the dataset analysis section), while only the preprocessed collection indexing will be used for search pipelines and retrieval models later on.

The index for the preprocessed collection has given 2,225,016 terms, with 253,719,768 postings , with 1 field (*contents*) and 415,050,364 tokens.

3.2 Base Models

For the first phase of the Results analysis 4 different base models have been chosen:

- *Tf*: Term Frequency
- *Tf-Idf*: Term Frequency - Inverse Document Frequency
- *BM25*: bag-of-words retrieval function that ranks a set of documents based on the query terms appearing in each document, regardless of their proximity within the document.
- *LM Dirichlet Similarity*: Dirichlet Prior Smoothing

While the metrics employed to compare the obtained results are the following:

- *P*recision@10: Precision up to rank 10 result
- *R*ecall@10: Recall up to rank 10 result
- *n*DCG@10: Normalized Discounted Cumulative Gain up to rank 10 result
- *A*P@100: Average Precision up to rank 100 result
- RR@100: Mean Reciprocal Rank up to rank 100 result

Comparing the base results, in the Table 1, the top model in terms of performances appears to be *TF-IDF*, with *BM25* just below. *Dirichlet LM* doesn't come close to the 2 previous result, while *Tf*, as expected, is the worst model out of the 4.

	name	P@10	R@10	nDCG@10	AP@100	RR@100
0	TF_IDF	0.166384	0.106992	0.190944	0.099761	0.441662
2	BM25	0.165584	0.106653	0.189841	0.099566	0.438675
3	DirichletLM	0.141188	0.090756	0.160028	0.081381	0.377222
1	Tf	0.006326	0.003843	0.006085	0.002842	0.023874

Table 1: Base Models results comparison

From now on, the reranking and enhancement approaches will be applied to the top-2 models (*TF-IDF* and *BM25*), since it's unlikely that the other models could improve in such a way to result in better performances than those 2.

3.3 Query Expansion and Rewriting

The first attempt to enhance the base results, was to apply some query rewriting techniques, like *KL (Kullback–Leibler) Divergence* and *Query Tokeniser*. The Table 2 shows the results obtained for the *KL Divergence*, which results in an improvement for both the base results of *TF-IDF* and *BM25*.

	name	P@10	R@10	nDCG@10	AP@100	RR@100
0	TF-IDF » KL	0.175050	0.112700	0.197272	0.109632	0.436601
0	BM25 » KL	0.174065	0.112141	0.196082	0.109301	0.434015
0	TF_IDF	0.166384	0.106992	0.190944	0.099761	0.441662
2	BM25	0.165584	0.106653	0.189841	0.099566	0.438675

Table 2: KL Divergence results comparison

The *Query Tokeniser* approach doesn't change much: there's only a really slight improvement on the *Average Precision* and the *Mean Reciprocal Rank*

	name	P@10	R@10	nDCG@10	AP@100	RR@100
0	TF_IDF	0.166384	0.106992	0.190944	0.099761	0.441662
0	TF-IDF » TOKENISER	0.166384	0.106992	0.190944	0.099762	0.441666
2	BM25	0.165584	0.106653	0.189841	0.099566	0.438675
0	BM25 » TOKENISER	0.165584	0.106653	0.189841	0.099568	0.438679

Table 3: Query Tokeniser results comparison

4 Search Engine - Advanced Search

For the Neural Approach, related to the task B of our project, we tried different methodologies, but not all of them were executed because of computational environment limitation.

The first reranker used was *KNRM*: the first attempt was to use it in combination with a *word2vec* vocabulary, but both the *wordvec_unk* and *wordvec_hash* were too heavy for the RAM limits offered by Colab, so we had to use it in combination with a *Bert* vocabulary. The Table 4 shows the obtained results, compared to the base models, whose performances were not improved.

	name	P@10	R@10	nDCG@10	AP@100	RR@100
0	TF_IDF	0.166384	0.106992	0.190944	0.099761	0.441662
2	BM25	0.165584	0.106653	0.189841	0.099566	0.438675
0	BM25 » KNRM	0.032276	0.019442	0.031387	0.033564	0.107038
0	TF_IDF » KNRM	0.031953	0.019286	0.031709	0.033814	0.109683

Table 4: KNRM reranker results comparison

The second attempt was to use a *Vanilla-Bert* reranker, but also in this case, the results were not as good as the base models.

	name	P@10	R@10	nDCG@10	AP@100	RR@100
0	TF_IDF	0.166384	0.106992	0.190944	0.099761	0.441662
2	BM25	0.165584	0.106653	0.189841	0.099566	0.438675
0	BM25 » V-BERT	0.064445	0.040387	0.062490	0.046706	0.168578
0	TF_IDF » V-BERT	0.064260	0.040323	0.062253	0.046603	0.168213

Table 5: Vanilla-BERT reranker results comparison

The Table 6 shows a comparison between the 2 previously described neural rerankers. We can see how in both cases *BM25* performs better than *TF-IDF*, but still not as good as the corresponding base model. So we will try to improve that by creating a new pipeline which weights the *BM25* and *Vanilla-Bert* combination.

	name	P@10	R@10	nDCG@10	AP@100	RR@100
0	BM25 » V-BERT	0.064445	0.040387	0.062490	0.046706	0.168578
0	TF_IDF » V-BERT	0.064260	0.040323	0.062253	0.046603	0.168213
0	BM25 » KNRM	0.032276	0.019442	0.031387	0.033564	0.107038
0	TF_IDF » KNRM	0.031953	0.019286	0.031709	0.033814	0.109683

Table 6: Neural rerankers results comparison

We tried 2 weighting approaches: a (0.9*BM25 + 0.1*VBERT) and a (0.75*BM25 + 0.25*VBERT). As the Table 7 shows, the first approach managed to improve the results of the base *BM25* model, but not the *TF-IDF* ones.

	name	P@10	R@10	nDCG@10	AP@100	RR@100
0	TF_IDF	0.166384	0.106992	0.190944	0.099761	0.441662
0	BM25 » 0.9*BM25 + 0.1*VBERT	0.165877	0.106900	0.190214	0.099817	0.438986
2	BM25	0.165584	0.106653	0.189841	0.099566	0.438675
0	BM25 » 0.75*BM25 + 0.25*VBERT	0.164784	0.106148	0.189365	0.099013	0.440613
0	BM25 » V-BERT	0.064445	0.040387	0.062490	0.046706	0.168578
0	TF_IDF » V-BERT	0.064260	0.040323	0.062253	0.046603	0.168213
0	BM25 » KNRM	0.032276	0.019442	0.031387	0.033564	0.107038
0	TF_IDF » KNRM	0.031953	0.019286	0.031709	0.033814	0.109683

Table 7: Weighted rerankers results comparison

4.1 Other Neural Attempts

As previously stated, we also tried to implement pipelines that were not executable by Google Colab.

First we tried to use a pre-tuned *Bert* model, but since the *OpenNir* library isn't that popular, we didn't manage to find a checkpoint for a dataset related to our problem, and the only other possibility was to perform a fine-tuning from scratch, which wasn't a viable solution due to the size of the collection and of our train set.

After that we tried to apply an Automatic Ranking Learning technique, using a *Random Forest Regressor*, but even after reducing the train set size from 550.000 to 50.000 (1/11th), the fit phase stopped after 6 hours of execution due to RAM exceeding.

5 Conclusions

As it's possible to see in Table 8 the best model is *TF-ID* after applying *KL Divergence*, a query rewriting technique. All the measures are the best ones in relation to the others. Alternatively using *BM25*, applying *KL Divergence* as well, could be a good choice which slightly deviates from the previous.

In conclusion, having in disposal a better and more powerful environment can certainly lead to more and better experiments, especially on the neural implementation side.

	name	P@10	R@10	nDCG@10	AP@100	RR@100
0	TF-IDF » KL	0.175050	0.112700	0.197272	0.109632	0.436601
0	BM25 » KL	0.174065	0.112141	0.196082	0.109301	0.434015
0	TF_IDF	0.166384	0.106992	0.190944	0.099761	0.441662
0	TF-IDF » TOKENISER	0.166384	0.106992	0.190944	0.099762	0.441666
0	BM25 » 0.9*BM25 + 0.1*VBERT	0.165877	0.106900	0.190214	0.099817	0.438986
2	BM25	0.165584	0.106653	0.189841	0.099566	0.438675
0	BM25 » TOKENISER	0.165584	0.106653	0.189841	0.099568	0.438679
0	BM25 » 0.75*BM25 + 0.25*VBERT	0.164784	0.106148	0.189365	0.099013	0.440613
3	DirichletLM	0.141188	0.090756	0.160028	0.081381	0.377222
0	BM25 » V-BERT	0.064445	0.040387	0.062490	0.046706	0.168578
0	TF_IDF » V-BERT	0.064260	0.040323	0.062253	0.046603	0.168213
0	BM25 » KNRM	0.032276	0.019442	0.031387	0.033564	0.107038
0	TF_IDF » KNRM	0.031953	0.019286	0.031709	0.033814	0.109683
1	Tf	0.006326	0.003843	0.006085	0.002842	0.023874

Table 8: Final results comparison