Report Project 1: Information Processing and Retrieval

João Coelho 86448
Marcos Barata 83511
Pedro Gonçalves 83651

31st october 2019

1. Exercise 1

The objective of this exercise was to implement a simple approach for keyword extraction based on TF-IDF. As for the statement of the project, we used the 20NewsGroup dataset to compute the IDF values, and chose a textual document to extract the keyphrases. Our document was about Google acquiring Kaggle (Ex1/original.txt). Our first step was to preprocess all the documents, including the ones in the dataset. Preprocessing techniques used were: lowercasing, removing numbers, stopwords and punctuation. The results of preprocessing were saved in memory, so we don't use more time doing this, and can be found at: Ex1/Dataset/ – the 18846 files composing the 20NewsGroup dataset and Ex1/preprocessed.txt – our original textual document, but preprocessed. After this, we used scikitlearn's CountVectorizer to compute the unigrams, bigrams and trigrams of the preprocessed documents, to get the candidates for keyphrases. From this we were also able to get the term frequency (TF). Then, we proceeded to compute the IDF values for the candidates, using scikitlearn's TfIdfVectorizer. We decided to learn the IDF values for both the train and test sets since we aren't going to get the keywords for the test documents. The textual document was also added to the document collection for which the IDF was learnt, so as to eliminate zero values. We could've decided not to include the document, and use smoothing techniques to eliminate any zero values that could eventually occur, like LaPlace or Good Turing Smoothing. Finally, we multiplied the TF-IDF value by the number of words of the candidate. We also tested multiplying by the number of words, and the result was the same. As for results, we got that for our document, the top 5 keyphrases were: kaggle, google, science machine learning, machine learning competitions, data science machine. We can see that two of them don't really make sense without the removed stop word and (data science **and** machine learning produced 2 trigrams). The others do make some sense in the context, however, something about Google buying Kaggle, which is the main point, wasn't captured. Another problem with this simple approach is that if a word/phrase doesn't appear in any of the 20NewsGroup files, it'll most likely be chosen as a keyword. For this document, we have the example of the word "kaggle", which doesn't present a problem since it can be considered a keyword. However, for another document we tested, which was about how good an Apple laptop was in 2010, the phrase "swiss army knife" was retrieved, since it never occured in any of the 20NewsGroup files, and it doesn't make much sense to be a keyphrase for the document.

2. Exercise 2

This exercise aimed to evaluate the baseline for keyphrase extraction implemented in exercise 1. From the link provided in the statement, we chose the DUC-2001 dataset, which is made of only a test set of 308 documents. We didn't divide the provided set in train/test, because they aren't that many and, since we will compute the IDF to all the documents to eliminate possible zero values, there was no need in doing so. Therefore, we started by parsing the XML files to simple text files (Ex2/Dataset_as_txt). We used the original words instead of their lemmas. Using the lemmas could help since, for example, plurals and singulars of the same word are lemmatized to the same lemma, but it also has some drawbacks, for example, the words Windows (the OS) and window (of an house) are reduced to the same. The generated text files were preprocessed. For this exercise, we decided to keep the stop words, because when we inspected the golden set, many of the keyphrases had stop words on it. (Ex2/Dataset_preprocessed). Then, we parsed the golden-set json (Ex2/test.reader.json), generating one file by document, where each file has one keyphrase by line (Ex2/Keyphrases_golden_set). Golden-Keyphrases with more than 3 words were discarded, since we are only considering candidates up to trigrams. With all the processing done, we computed the IDF values for all the 308 files, and for each one, computed the unigrams, bigrams and trigrams, to select the candidates, so that we can get the top 5 keyphrases, following the method explained in exercise 1. We extracted 10 keyphrases for each document, so that precision differs from precision at 5, and they were saved in memory (Ex2/Keyphrases_experimental). Now, to evaluate this baseline, we needed to compare the retrieved keyphrases to the golden set, for each document. To do that, we used the following measures, as requested:

$$Precision = TP/(TP + FP)$$

$$Recall = TP/(TP + FN)$$

$$F1 = 2PR/(P + R)$$

Individual measures for each document can be found at (Ex2/Metrics). Note that if P + R = 0 for a file, then we consider F1 = 0. The program also prints global average precision and recall for all documents, where the individual precisions(recall) were summed and divided by the total number of files (308).

Besides this values, we also calculated the precision at 5 (relevant keywords on the top5) and mean average precision (the average of the evolution of the precision as new keywords are extracted, for a document) Our global average results for this baseline were: $Precision = 5.7\,\%; Recall = 7.8\,\%; F1 = 6.6\,\%; P@5 = 5.9\,\%; MAV = 6,2\,\%$ They are low, and this can be explained by the problems presented at exercise 1. We also identified other problem while exploring the results. Take for example that the golden set for file X contains the phrase "computer networks". We noted that our system retrieved both "computer" and "networks", but not the whole phrase. We tried to tune the multiplying by the candidate's length feature, also tried adding the scaling factor, but the results barely changed, so we stood with what we explained at exercise 1. Also, as explained, we kept the stop words, so, another problems, is that single stop words will be retrieved by this model as keyphrases, when they are not, because their TF is very high. But as we explained, we decided to keep them because most of the keyphrases have stop words on it, and since we are going to try to improve this result by using candidate selection, single stop words will not be considered on the next exercise.

3. Exercise 3

On this exercise we tried to improve the obtained in 2. We implemented BM25 as per the formula provided in the statement. Also, we considered noun-phrase only candidates, i.e., candidates that match the grammar: {(<JJ >* <NN.*>+ <IN>)? <JJ>* <NN.*>+}. POS tagging was done using NLTK tagger. We tested in the same conditions in exercise 2. Our function that extracts the candidates was inspired by the one that can be found here. Global average results with BM25 and candidate selection: $Precision = 15.7\,\%; Recall = 16,9\,\%; F1 = 16.0\,\%; P@5 = 15.6\,\%; MAV = 15.4\,\%$ Since candidates are now being properly selected, single stop words are no longer being considered, therefore they aren't extracted as relevant keywords, however, they can appear in the middle of a trigram for example, which can be correctly retrieved. Our documents aren't very long, and because of that BM25 is a good evaluation method. To further improve this results, we tried:

1- Combining the BM25 score with the length of the candidate: We tried multiple linear combinations for this, none of which presented better results than what we previously got.

2- Using a less restrictive grammar: For some documents, the provided grammar returns less than 10 candidates, which means we won't be able to retrieve at least 10 keyphrases, as we did for exercise 2. Therefore, for those documents, we considered the general definition of a noun-phrase: an optional determiner followed by zero or more adjectives and a noun, proper noun or pronoun: {<DT>? <JJ>* (<NN$\|$<NP>$\|$<PRN>)+}.

Candidates matching this regexp were selected for documents where the first grammar returns less than 10 candidates. This returned values a little bit below the ones we started with:

$Precision = 15.2\,\%; Recall = 16.5\,\%; F1 = 15.8\,\%; P@5 = 15.2\,\%; MAV = 14.9\,\%$

Which probably means that for the phrases selected by the second grammar are false positives, in our case.

Given that we couldn't improve the result with our suggestions, running the code provided in Ex3/exercise-3.py will return the values using BM25 and candidate selection with the first grammar (the alternatives are commented). Folders in Ex3/ have the same items explained in exercise 2, such as metrics, extracted keyphrases, etc.

4. Exercise 4

For this exercise we implemented a supervised approach, using the Perceptron algorithm. We divided our set (preprocessed the same way as for exercise 3) into test (10 %) and train set (90 %), can be found at Ex4/ Test_Set_Preprocessed and Ex4/ Train_Set_Preprocessed, respectively. Also, we divided the global golden set to the respective train/test golden set, which can be found at Ex4/Goldenset_train and Ex4/Goldenset_test, respectively. We started by using the base features presented in the statement, which yielded the following vectorization for a candidate: cand = (TFIDF_score, number_of_words, position_in_document). For the TFIDF score, we learned the values for the whole dataset. Then, for each document in the training set, we vectorized the candidates. We considered candidates all unigrams, bigrams, and trigrams, except single stop words. Also, we computed the list of expected result for each candidate (0 or 1), taking into account the golden keyphrases for the training set. When all the candidates of all documents (train) were vectorized, we trained the scikitlearn Perceptron. We went for a large number of epochs (10000) and a low learning rate (0.01). After training, the candidates of the documents in the test set were vectorized, so that the perceptron could predict whether or not it was a keyphrase. Running this first scenario gave the following global average results for all documents in the test set:

$Precision = 2.1\,\%; Recall = 0.8\,\%; F1 = 1.1\,\%; P@5 = 1.2\,\%; MAV = 3\,\%$

There was a clear problem in this model: For all the documents in the train set, there are 425363 candidates, where only 1927 are actual keyphrases. We could've tried two things: Undersampling or reducing the candidates as we did in exercise 3. We started by reducing the candidates to noun-phrase only. Unfortunately, our implementation for this had a bug in which the same keyphrase was being selected multiple times, so we can't conclude anything here. Undersampling would probably be the best approach to get good results here, as the class distribution was very imbalanced, but for time reasons we couldn't do so. Also, we would like to test with other features other than the ones proposed but again, for time issues, we weren't able to do so.