Course Project: Part 1

Group 9: Joel Almeida (81609), Matilde Gonçalves (82091), Rita Ramos (86274)

I. Exercise 1

In the first exercise, we started by dividing the document into its sentences using the nltk library. We decided to use nltk.sent_tokenize(), instead of using a trained model for portuguese segmentation, since the segmentation of sentences in english is similar to portuguese. Indeed, when we tried with nltk.data.load('tokenizers/punkt/portuguese.pickle'), our results were not affected and, thereby, we agree to use the default nltk.sent_tokenize. It should be noted that, for each sentence segmented by the nklt, we made sure that it was not composed by just punctuation, thereby sentences such as "(...)" were not considered.

After, we represented the sentences as vectors by means of scikit-learn and numpy libraries. First, we counted the terms for each sentence by using CountVectorizer(), which were then represented in a numpy array (Rows=sentences,Columns=terms, Values=counts). Secondly, having this numpy with the terms' counts, we easily computed the normalized TF by dividing the rows with the highest number of that row (most frequent term of that sentence). Then, we computed the inverse sentence frequency (ISF), also by using the numpy of the terms' counts. For each term (column), we computed the log of the size of the numpy (number of sentences) divided by the sum of the counts that were different to zero (number of sentences that had that term). Finally, we were able to represent the sentences by multiplying the normalized TFs with the ISF.

We then computed the terms' counts for the doc, by summing up the previous numpy with the counts of terms for each sentence. Giving an example, for two sentences with 5 terms as [1,0,2,1,5], [1,2,0,0,0], the result would be [2,2,2,1,5]. Having these terms' counts for the doc, we computed the normalized terms frequencies as we did for the sentences. After, the doc was represented as a vector by multiplying those TFs with the previous computed ISFs.

Finally, we computed the cosine similarity between each sentence and the doc, returning the top 3-highest scored sentences, as well as the summary for the user (those 3 sentences by the order of their appearance).

II. Exercise 2

In the second exercise, we iterate over the TeMario dataset and, for each file, we applied the approach of exercise 1 (representing the sentences and the doc as vectors). In this iteration, we were also saving all the docs contents and the sentences of each doc, in order to then apply the simplest approach of exercise 2. Indeed, when the iteration was over, we were able to apply the second alternative. In this second approach, since we had to calculate the IDFs over the documents collection, we started by representing all the docs into vectors and only then the sentences.

For the doc representation, we used the CountVectorizer (as in exercise 1) and then computed the respective TFs, as well as, the IDFs, using the log of the number of docs divided by the docs that contain the respective term. We then use the vocabulary of all these docs (vocabulary_ of the CountVectorizer) and the IDFs to represent the sentences as vectors. Therefore, for each sentence, we used the CountVectorizer with the given vocabulary, and then calculated the TFs and multiplied by the given IDFs.

After we calculated, for each document, the cosine similarity of each approach and evaluate it. The evaluation involved the precision, recall and the AP (we first calculated AP with interpolation, but we then decided to just sum the precision of the recall points divided by the number of relevant sentences). In the end, for each approach, we computed the mean of the Precision and the Recall, using it to calculate the F-measure, and we also computed the mean of the AP (MAP).

As expected, the first approach was better than the simplest approach of exercise 2, as it can be seen in the following results:

Table I: Results of Ex1 and Ex2

	Exercise 1	Exercise 2
Precision	0.4639999999999999	0.42399999999999993
Recall	0.25963301649331055	0.234116199976494
F1	0.33295805168394976	0.30166486949745014
MAP	0.18577918459536102	0.163760558858353

III. Exercise 3

For this exercise, we used the attribute ngram_range in CountVectorizer to automatically count the unigrams and bigrams for all the sentences of the document. After, we added the noun phrases to this initial counting matrix. For that, we started by tagging the words of each sentence, using UnigramTagger trainer based in Floresta from NLTK, in order to learn the tags for different words in portuguese. When iterating through the document sentences, for each word we used the tagger to get its tag and used the simplify_tag function[1] to get the part of the tag that matters to us. We were then able to find the noun phrases, using the regular expression given in the project document, reformulated with the following correspondences of the portuguese tag's names: JJ -> (prp, conj-s, conj-c); NN -> (adj); NN -> (n, prop). For each tagged sentence in the document, we found all the contained noun phrases, considering just the noun phrases with more than 2 words (to avoid repetitions with the previous uni and bigrams terms). We initialized a vector to store those counts for all the document sentences. We put this vector in a dictionary and increased the counting each time we found the same noun phrase in another sentence. After the counting is done, we added each vector, transposed, to the initial counting matrix (with unigrams and bigrams).

Having the numpy array with counts of the unigrams, bigrams and noun phrases, we were able to represent the sentences with the BM25, calculating the inverse sentence frequencies and then computing the score, with k = 1.2 and b = 0.75. For representing the doc, we summed up the previous numpy with the counts of terms for each sentence (as in exercise 1), and computed the BM25 score with the previous calculated isfs.

Besides what was requested in the project document, we considered relevant to also remove stopwords, besides of having all terms in lower case. These improvements increased significantly our results. In addition, we attempt to remove the rarest words, as in lab3, but this resulted in a slightly worse performance overall, since each document has not many repeated terms (mosts terms were removed). We also attempted to remove the most frequent words (90%) of the doc, but this did not change our results, since, as mentioned, the document has not many repeated terms. After, we attempted to just use portuguese stems for the unigrams and bigrams, but it also lead to a worse result.

After implementing all the described above, we tried as well the same algorithm with both the Hidden Markov Model and Perceptron trainers and we verified a higher precision increase with HMM, although without a substantial change.

Finally, we played around with the K and B parameters, having the best result with a k=2.0 (see Table II). As it can be seen, we had a slightly better performance than exercise 1.

Table II: Results of Ex1 and Ex3

	Exercise 1	Exercise 3
Precision	0.46399999999999999	0.46400000000000002
Recall	0.25963301649331055	0.2592205302867068
F1	0.33295805168394976	0.33261867166671816
MAP	0.18577918459536102	0.18600864095055272

IV. Exercício 4

In this exercise, we selected the TF-IDF weights for the representation of the docs and its sentences (1^{o} exercise). We iterate for each of the docs, and having the numpy with the doc sentences, we easily computed the cosine similarity with the doc vector, as we did in the previous exercises. And we then computed the similarity with the previous selected sentences (if any). Having now the numpy of sentences with the corresponding MMR, we can select the sentence that has the highest score (argmax). We save this new sentence to the doc summary, and eliminate it from the set of doc sentences, so that it will not be chosen next time.

Finally, we iterated over all the docs summaries and evaluate it, as we did in the previous exercises, along with the approach of the top five sentences.

It should be noted that we use the lambda value 1/5, since the second part of the MMR equation (similarity with the 5 summaries) will have more weight than the first part of the term. However, the results were actually better with the lambda value equal to 0 (same result of exercise 1), which means there was no advantage in comparing the sentences with the summaries, in this case.

Table III: Results of Ex4

	Exercise 4 - Alternative MMR	Exercise 4 - Alternative Top 5
Precision	0.44800000000000006	0.26399999999999997
Recall	0.249260971871266	0.1495511555438026
F1	0.3203073738621481	0.19093891787896147
MAP	0.1827508986928104	0.054559285499359005

[1]http://www.nltk.org/howto/portuguese_en.html