PRI Report

Group 4

Nelson Nunes  
 DEI  
Instituto Superior Tecnico  
 [nelson.filipe@tecnico.ulisboa.pt](mailto:nelson.filipe@tecnico.ulisboa.pt)

André Fernandes  
 DEI  
Instituto Superior Tecnico  
[andre.diegues@tecnico.ulisboa.pt](mailto:andre.diegues@tecnico.ulisboa.pt)

Miguel Reis  
 DEI  
Instituto Superior Tecnico  
[miguel.silva.reis@tecnico.ulisboa.pt](mailto:miguel.silva.reis@tecnico.ulisboa.pt)

ABSTRACT

TODO

Introduction:

In this paper we describe our work for the Process and Retrieval Information course. Three tasks were addressed: Ad hoc Search, Text Classification and Named Entity Recognition with statistical analysis. The Datasets used were extracted from the Manifest Project dataset. For both task one and three, the English version was used to better use the existent English vocabulary NLP tools. For the second task, the Portuguese version was used, in this way as we know more about Portuguese Politics, we could better analyze our results.

1 Ad-Hoc Search:

For this problem, we can divide it into three subproblems. Dataset cleaning, document indexing and searching over relevant documents given a query.

In our Problem we were presented by 2 datasets in which we could choose any. For the ad-hoc search we randomly chose *en\_docs.csv.* This dataset presents some problems. The first one is that there are some blank cells. We solved this simply replacing them for a space. Another problem is that some documents are split over many rows. Their *title*, *date*, *manifesto\_id*, *source*, *md5sum\_text*, *id*, are the same. Then the column *pos* identifies their order although they are ordered sequentially over the rows. Having this into account we joined all rows which have the previous attributes equal. Despite the obvious advantage that we can now more easily present the entire documents instead of parts of them, this change greatly reduced the amount of rows in our dataset greatly reducing also the amount of time and memory necessary to index the documents. The cleaned dataset is outputted to the file *en\_docs\_clean.csv.*

To index the document collection, most solutions pass by by defining an *inverted index* of the document collection. An Inverted index is essentially a dictionary that connects a term to its documents-frequency pair. For this, the document collection is ran trough identifying new terms and saving their document occurrence.

Possible solutions for the third subproblem are not so straight forward. Although generally documents that don’t contain any query word are considered irrelevant to the query, how can we present the others by ordering them by *relevance*? This is achieved by a variety of *Scoring Algorithms.* The most popular is *tf-idf,* although *BM25* is gaining popularity.

In the practical part, we used as baseline a tf-idf algorithm based on the version produced in the second lab of the course. This algorithm creates a document per term matrix, in which we transform into an inverted index in memory, at the beginning of the program. Then given a search query it scores the documents according to the terms in the documents using the algorithm.

For comparison purposes we also developed an application of the library *whoosh.* This version indexes the documents in disk, and searches over them having into account both *title* and *text* of the documents. It can use both *BM25* and tf-idf algorithms. By default we chose the former. Whoosh discards words like ‘the’ in search queries because they usually don’t give any context of meaning and are always present in texts. Whoosh also implicitly takes care of punctuation parsing problems. It’s much more precise and robust the our baseline version.

Given the results of the comparison, we decided to default our use to the whoosh version although we can explicitly choose which one to use. More details in section 4.1.

2 Text Classification:

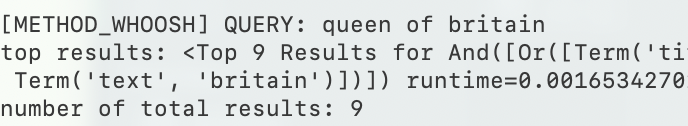
We can divide this task into two subtasks: text preprocessing and method implementation.

For text preprocessing we consider the following strategies: the NLTK Portuguese stemmer, the NLTK Portuguese stopwords and the use of CounterVectorizer or Td-idf Vectorizer from the sklearn library, the use of different min and max frequency, and the number of ngrams extracted.

For method selection we tried a variety of approaches: three different Naïve Bayes implementations (BernoulliNB, MultinomialNB, GaussianNB), SGDClassifier, LinearSVM, a Deep Neural Network (Multi-Layer Perceptron), a Convolutional Neural Network, and a Voting Emsemble Method build with a NaïveBayes, SVM, and RandomForest classifiers).

For some of the previous methods, the RAM requirements (specially for bigger ngram max values) were enormous, so we implemented a mini-batch approach using the partial-fit function of the sklearn.

We performed a grid search for these parameters and the results can be view under the folder Results in the Ex2 folder. More explanation of these results in section 4. The script we coded for this testing was classificationTesting.py

After analyzing the results the architecture selected doesn’t use stopwords or stemming, uses td-idf normalized with min frequency of 0.001 and max frequency of 0.5, with n-grams:1,2,3, and the model is a SVM using linear kernel.

3 Named Entity Recognition:

4 Results:

4.1 Ad-Hoc Search:

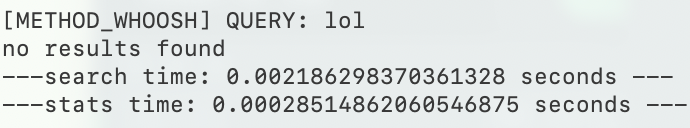
Comparing both approaches using the cleaned dataset, the whoosh approaches take on average much longer to create when comparing to the lab version, although this time is still negligible. This is due to disk access in the whoosh version, and memory only in the lab approach. When we use a bigger dataset (uncleaned data) the whoosh version is 6 times faster than our second approach. The lab version also has peak memory usage of over 2Gb. This is mainly due to optimizations in the whoosh library.

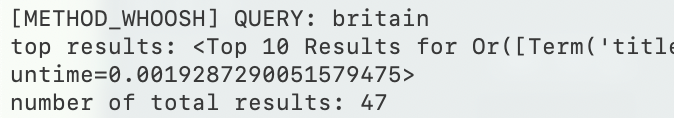
For search time the lab version is generally 5 times faster on the cleaned dataset although irrelevant to the user.

The way statistics are calculated, are different for both approaches, as they are built in different designs. For statistics calculation time, the lab version is generally 2 times slower on the cleaned dataset. Using uncleaned dataset, search times are slighter bigger but irrelevant for the user. Statistics calculation tend to be much higher for the lab version, but acceptable for the whoosh version.

Test values are depicted in more detail in Fig1 in section 5.

For these reasons, the specifications detailed in section 1 and keeping the code generalized for any dataset, we decided that the whoosh version is better and more robust. The following examples are all using whoosh.

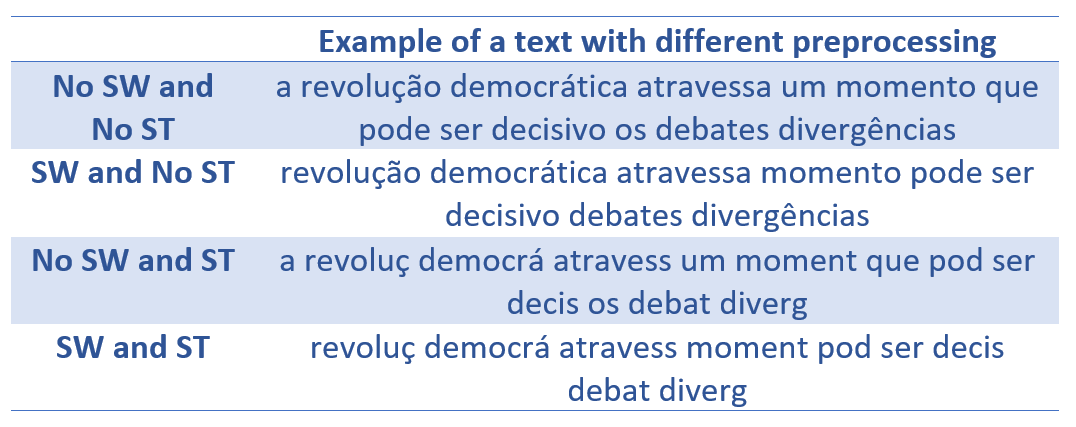
If we search using the query ‘lol’ , we can see that there are no documents found. This is because this sequence of characters is not present in the lexicon of the document collection, i.e there are no documents that use the word ‘lol’. As such, its impossible to calculate the similarity between the query and the documents.

Using the search query ‘britain’, we can see that there are a lot of documents found related with the query. This is to be expected, as the documents are a collection of manifestos from different political parties in Britain.

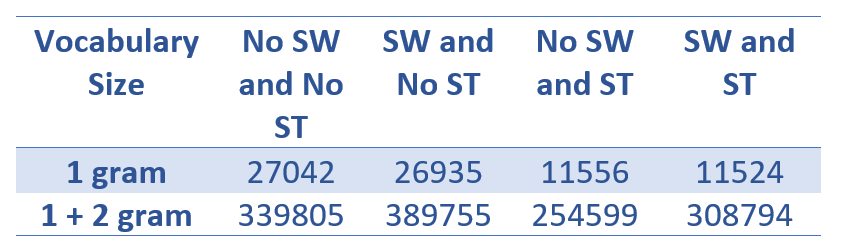
For longer queries a logical ‘and’ is applied for the search between the different keywords. This allows for results which are much closely related to the search query as a whole instead of the individual words. Using the search query ‘queen of britain’ we can see that the number of results has greatly reduced when comparing to the previous query. We only present results which contain both ‘queen’ and ‘britain’. The keyword ‘of’ is discarded like explained in Section 1.1.

4.2 Text Classification:

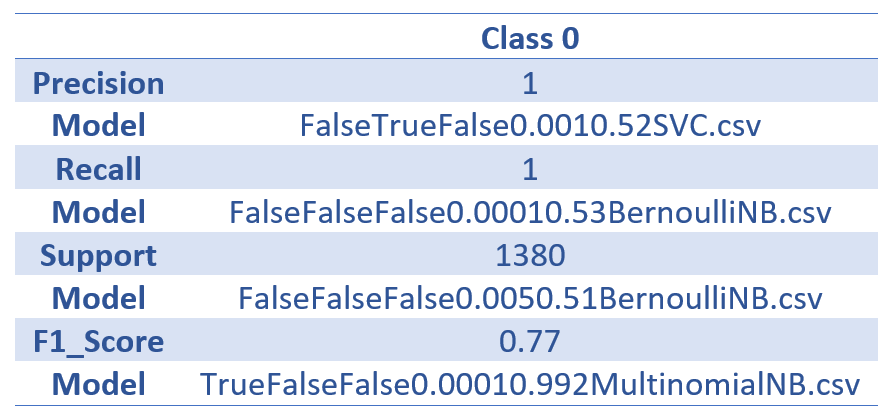
In the following table we show an example of the results of our tested preprocessing, SW means StopWords and ST means Stemming.

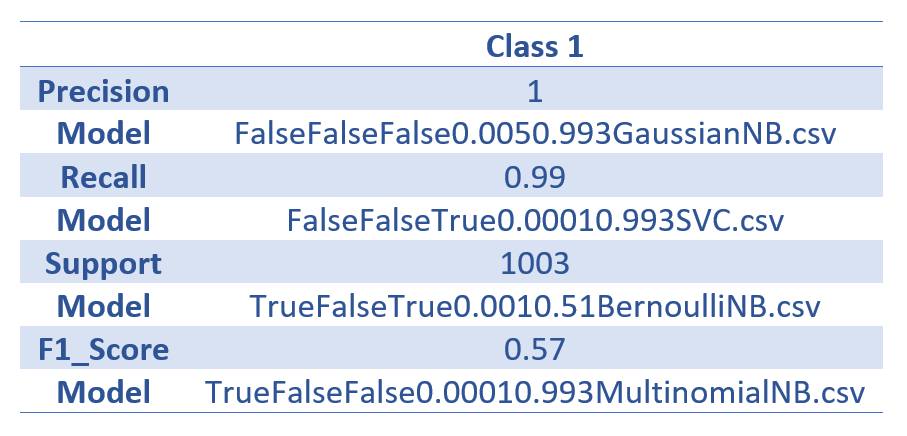


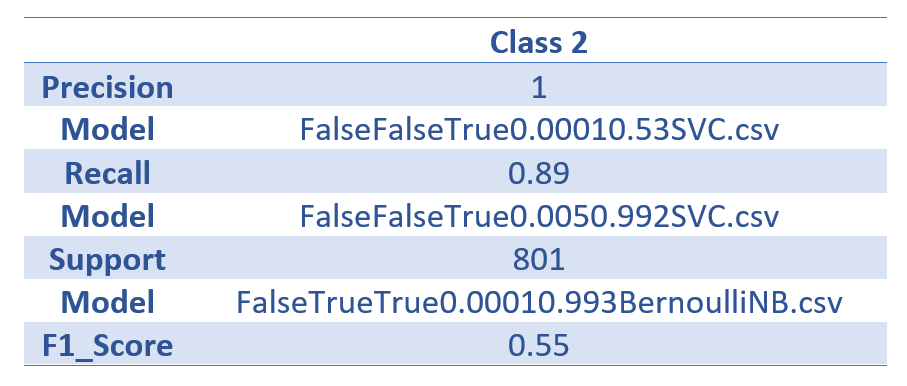
In the following table we show the number of parameters growing with number of ngrams chosen.

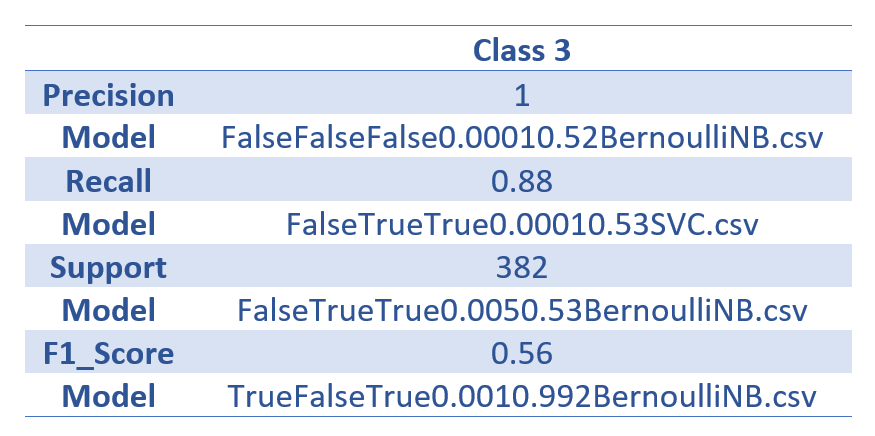


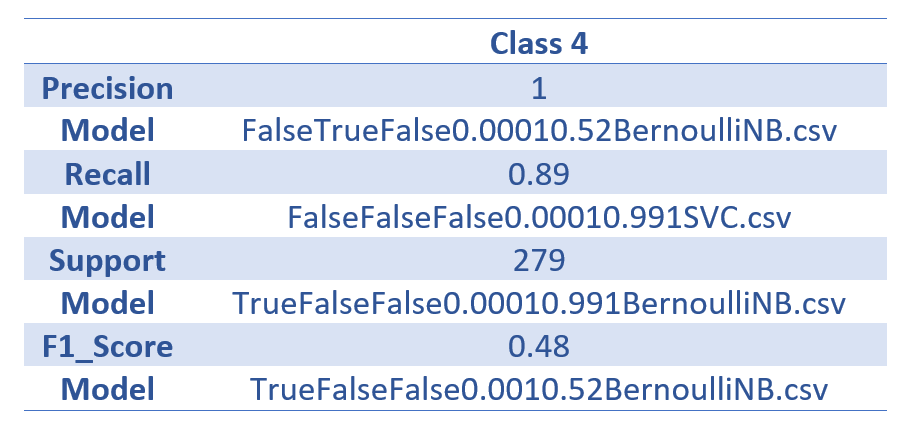
The following tables show some information we extracted from the results of our grid search (576 models tested), grouped by class. The name structure follows the following rule:

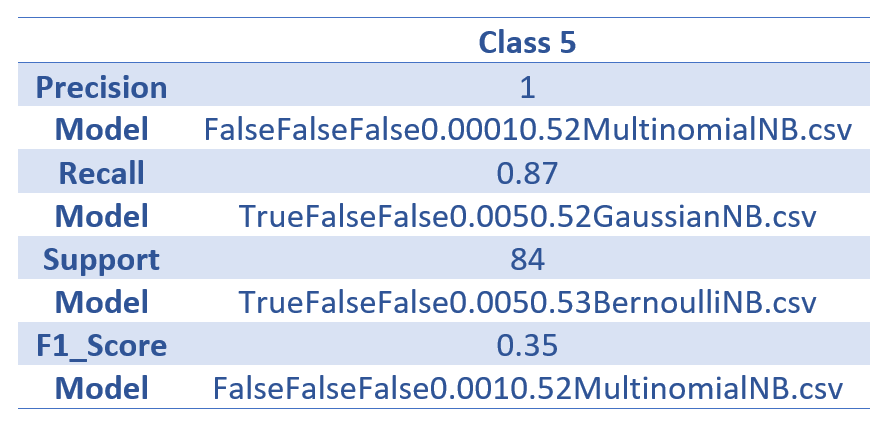
[StopWords:Steeming:tfIdf;minFreq;maxFreq:Ngrams:model.csv] 







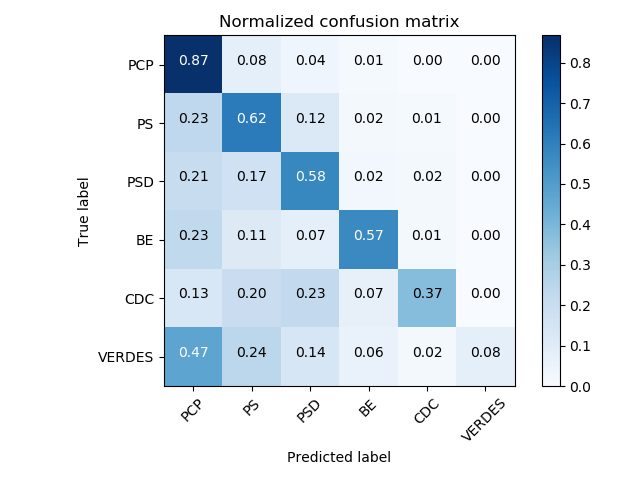




The architecture who achieved better overall results doesn’t use stop words or stemming, uses td-idf normalized with min frequency of 0.001 and max frequency of 0.5, with n-grams:1,2,3, and the model is a SVM using linear kernel.

The following results show its performance.



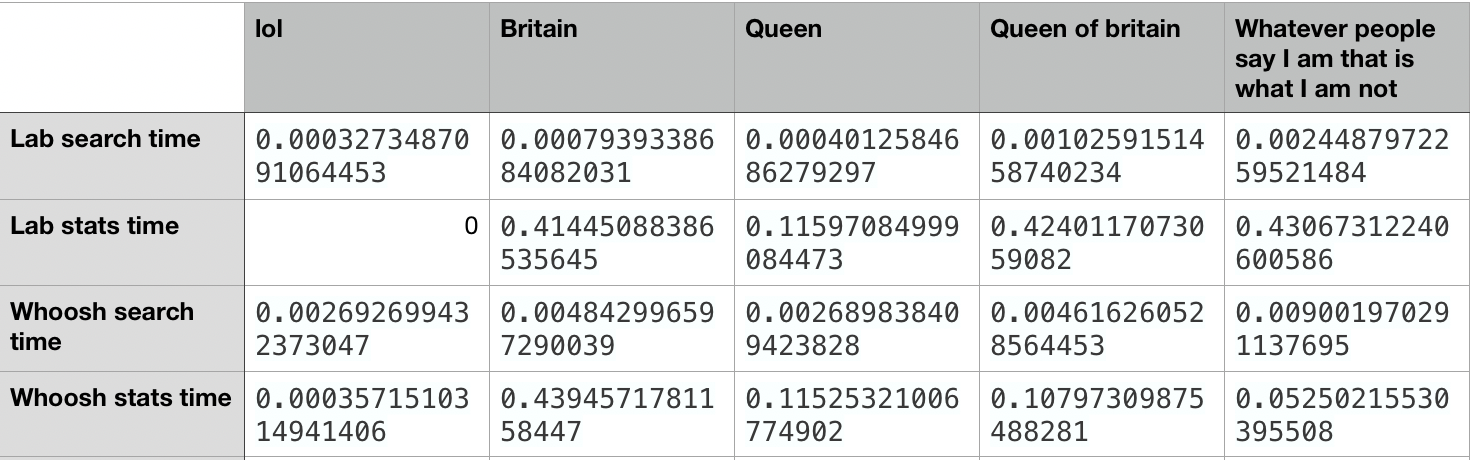


Extra Results: Testing with Neural Networks showed clear overfitting that we weren’t able to fix (always predicted the same class). Voting ensembles with both hard and soft voting achieved less than 0.5 of precision. More than 3 NGRAMS lead to worst results. Not defining a min and max frequency lead to worst results. Stop words and stemming didn’t improve the results.

The performance of SVM seems logic as we think that: text is often linear separable, and text has a lot of features. The Linear SVM is also a good choice because it is computationally cheap.

We understand that better results were possible it the dataset was more balanced, Verdes instances are highly infrequent, while PCP are highly frequent instances.

4.3 Named Entity Recognition:

5 Attachments:

**Fig1** Calculation times per query on the cleaned (smaller) dataset