Final Project

Machine Learning Applications

Marta Balairón García

Gracia Estrán Buyo

Marta Almagro Fuello

INDEX

Task 0: Data set creation 3

Task 1: Text Pre-processing and vectorization 5

Text pre-processing 5

text vectorization 6

clasical BoW or TF_IDF representation 6

Word2vec/Fast Text based representation or Doc2Vec vectorization.**Error! Bookmark not defined.**

topic modeling using LDA 7

Task 2: Machine Learning Model 10

Feature Extraction & selection 10

Testament classification 10

Super vector classifier 10

Multinomial naive Bayes classifier 11

Logistic regressor 11

Feature Extraction & selection 11

Book classification 11

Super vector classifier 12

Multinomial naive Bayes classifier 12

validation techniques 12

Task 3: Implementation of dashboard 13

TASK 0: DATA SET CREATION

In order to work with our own dataset, we had to find a collection of many documents that could be interesting to clean and classify. We came up with the Bible as it has many chapters and books with texts that have different contexts depending on if they are from the Old Testament or the New Testament, or different books.

We found in internet a website with all its books and separated depending on the testament that they come from. The website is: https://www.iglesia.net/biblia/libros/

So, to get the information that we need form the web, we first downloaded the whole website using the application *wget* and with the simple command of: *wget https://www.iglesia.net/biblisa/libros/*

In a matter of a few hours, we had all the pages (in the .html extension) of the website stored in our computer. Then we did some cleaning because we just wanted two folders, one with Old Testament books and another one with the new ones. Afterwards we uploaded it to Google drive and developed a function (*get_text*) whose purpose was to go through every word of a given text and return an array with the contents of each chapter from the text. As we have observed in the web pages every chapter starts with the word: *capítulo* and the number of that chapter, so to store the sentences that are in that chapter we will look for the *capítulo* word and save all the following strings until we find again the word *capítulo*. In order words, we keep the index of each time *capítulo* (plus a number) appeared and do a loop that goes from the actual index to the next one, storing all the texts in between those indexes.

Although we are not yet in the text cleaning task, we used these loops to delete some parts of text that should not be kept like for example '\n' and the word *capítulo* at the beginning of each chapter.

Once this function was implemented, we just had to apply it to each of the html pages that we had stored in our folders. We realized again a loop to go through each file and get the text using the *BeautifulSoup* library. Another powerful library used in this task was the *os* library as it was the one that let us iterate through the files of a folder with the function "*listdir(folder_path)".*

In each repetition of that loop, we store a row in the dataset with the book, testament, index chapter, and text of a chapter, which ended being a total of 1184 rows.

We do the loop two times, one for the New Testament and other one for the old one. Finally, we concatenate them (and save it in a pickle file) to form the dataset that we will use continuing this project.

One of the reasons we did not store it by verse instead of chapters is because we observed that many verses where short or they really did not add much information to the whole book so it would be very difficult to manage a classification task with a dataset created of that form.

TASK 1: TEXT PRE-PROCESSING AND VECTORIZATION

This task is divided in two steps, the first one is text pre-processing which consists in the tokenization, homogenization and cleaning of the given texts.

TEXT PRE-PROCESSING

In order to achieve these last goals, we have used the library Spacy and re in the implemented function called *text_preprocessing*. It is important to say that during the creation of the dataset we also removed some tags that came from html document, which it is considered part of the cleaning task.

Apart from importing the library spacy, we had to load the model which is already trained on large corporate texts and it is optimized for various language domains. This model will save us a lot of time because we won't have to train our own models from scratch in order to complete the natural processing tasks.

In our case we loaded the *'es_core_news_md'* model, as what we want is a pre-trained statistical model for the Spanish language. In addition, we disabled the dependency parsing since our goal is to use the pipeline to obtain BoW representation of the documents.

As we know now the three goals of our text pre-processing function these are some of the solutions the we carried out:

1. Tokenization: we convert the text into a doc, which will be a sequence of tokens, which are the ones where we will get most of the information.
2. Delete the numbers' verses (versiculos) from the text: each chapter is divided in verses that are indicated with a number of the form 1:1, 1:2…. We considered that is not important or relevant since we won't be dividing the chapters in verses.
3. Removal of the numbers, punctuation, stop words: they won't be helpful in the following analysis, they are just noise.
4. Homogenization: convert the words to lowercase and to its lemma so we locate the words that have the same meaning and do not treat them like different.
5. Named Entity Recognition (NER): store the entities in a variable for further analysis in the following steps. We will store in other column the labels of them because they will give us more information of each entity.

6. Part-of-Speech (POS) Tagging: store also the types of words because it provides useful information about the grammatical structure of the sentences. Can be used later to understand the meaning of the text.

Once the function is ready, we applied it to each text of the column, adding to the dataset four more columns: *Clean_text*, *Entities*, *Entities_labs* and *Pos_tags.*

Afterwards we created the corpus that we will be using when creating the dictionary and substitute the possible N-grams that we have.

When we created our dictionary we observed that we got 20745 terms which is a good number but a little over fitted for the 'small' number of documents that our corpus has, which is 1184. So we decided to filter it by setting the number of the minimum number of doc to 2 and the maximum proportion of documents that a token appear to 0.5. In fact, now we will have a dictionary with 11798 terms and an average number of tokens per review equal to 215.7 which after testing with other lengths this give us better results.

After this last modification we updated the column *clean_text* with the N-grams located and just the tokens that are in our filtered dictionary.

TEXT VECTORIZATION

The second step of task 1 will consist in text vectorization and analysing which is going to be the best representation for our dataset and classification models.

CLASICAL BOW OR TF_IDF REPRESENTATION

The first analysis is done comparing the Classical BoW and TF-IDF representation.

So, the first thing we do is splitting the dataset into training a testing sets and secondly we create the BoW and TF-IDF models using the Dictionary. We are going to train two classifiers, SVM and Naïve Bayes, which gave really high accuracies when predicting the testaments, as it makes sense because it is know that they have very different characteristic when expressing ideas and topics. However, predicting the books is not as easy, so we get a much worse accuracy. In both cases the highest accuracy is given when using the BoW model, this can be explained because we have a small corpus and TF-IDF is more effective when dealing with larger corpus. Also BoW treats all words as equally important, which can be problematic in cases where common words like "the," "is," and "and" but as we have cleaned before all the chapters is giving the importance to each word depending on the times it appear in the text.

In conclusion we thought that TF-IDF the task requires identifying important keywords or phrases within the documents however given the results and the fact that BoW is often used in text classification tasks because it's easy to interpret we will choose bag of words to vectorise our dataset.

WORD2VEC/FAST TEXT BASED REPRESENTATION OR DOC2VEC VECTORIZATION.

In this section we analyise if it was better to classify the column book depending in the values of each word or doc. We will start by analysing Word2Vec, first, we will convert each word of each sentence of X_train and X_test to its corresponding Word2Vec embedding representation by looking up the embedding of each word in the sentence in the pre-trained Word2Vec model. The pre-trained model we did it by using the sentences of the clean_txt.txt and saving its keyed vectors. Afterwards, in order to get the train and test embedding we will go through every word of the train and test data that we got in the split function and assign it its word vectors. Then we will get the 'average' vector of that sentence (taking into consideration the vectors of all the words in that sentence) and store it in the *train_embeddings* (or *test_embeddings*).

In the case of Doc2Vec, we start by creating a TaggedDocument object, which is a way to represent each document as a tuple of words and a unique identifier, or tag. The words parameter contains the tokenized words in the document, and the tags parameter contains the document identifier. Later we use those TaggedDocument objects to create the Doc2Vec model and, after training them, the document vectors are then appended to the *doc_vectors* list.

Finally, we just split it in train and test and perform a document classification task using the Doc2Vec algorithm and an SVM classifier.

After seeing the results of classifying the class *book,* we came to the conclusion that it makes much more sense to analyse each doc, that represents a chapter, rather than each word, since what really interest us is what the set of words represents, not just the words. So as we will be using TF-IDF and bag of words are techniques for the classification, we will not involve the creation of word embedding. After the SVC prediction we can conclude, that even though is not much difference, the accuracy confirms our appreciation, and that is better to use Doc2Vec techniques.

TOPIC MODELING USING LDA

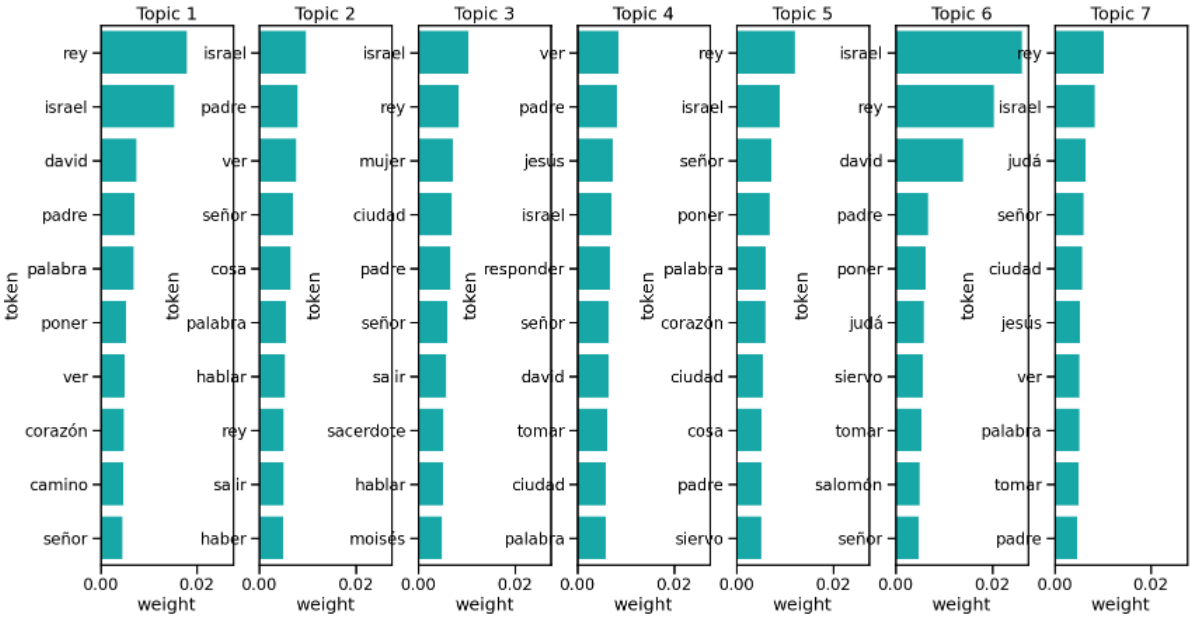
For this task, we are going to perform an analysis of the classification of our texts in different topics. For accomplishing this, we are going to make use of the LDA algorithm. This algorithm is

based on the idea that each of the documents is composed by several topics and each of this topic is composed of a token distribution.

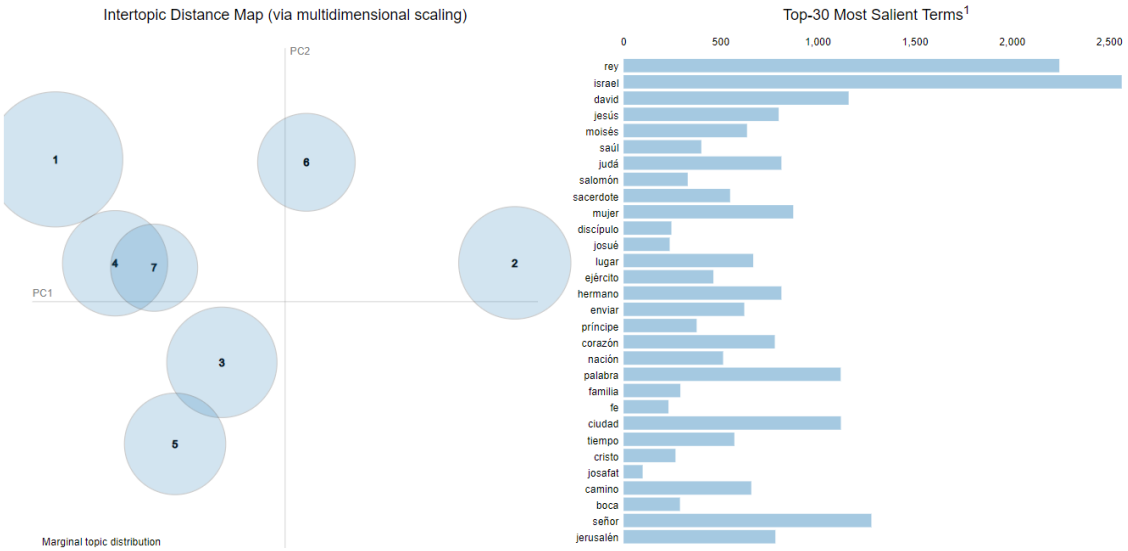
Firstly, we have gotten the corpus BoW representation. We have taken advantage of the method *doc2bow* which turns each of the documents of a corpus into a list of tuples. These tuples are composed by the correspondent index of a token and the frequency with which the token appears in that document.

Furthermore, we must select the number of topics into we want to classify our documents. Therefore, we make use of the coherence models for selecting the number of topics. A coherence model is a technique for evaluating the topics generated. It provides a metric based on some calculations regarding the semantic coherence of each of the topics considering the meaning of each of the tokens compounding those topics. The highest the value of the coherence the better. We decide to test from 5 to 25 topics. Since for each corpus and ldag model the topics are different each time we execute the code we obtain different values but always in a similar range. In this case, we obtain higher values for low number of topics. As we are going to analyse now, the topics are not well differentiated since our database contains documents from a very particular topic which is the bible. Therefore, we aren't interested in generating lots of them.

After some executions, we decide ourselves to analyse a case in which the best number of topics was set to 7. We perform a plot showing the weights of the 10 most relevant tokens of each of the topics:



Moreover, we have also plotted a graph with the library *gensimvis* which shows more graphically the topics distribution. As well it is interactive and shows the weights of each of the tokens at each topic when you click on it:



We want to remark the following aspects about topics with the help of the graphs:

- The most relevant topic (the first) is related to the figure of the king of Israel (David) and its relationship with God. Therefore, the most important tokens are King, Israel and David.
- Topic 2, as we can see in the graph is the most different from the others. It is related to the lessons about faith. Probably, it contains documents from the New Testament since there we can find how Jesus teaches to its disciples.
- The topic 3 is related to the figure of the women in Israel in the context of the bible. Therefore, we can find that some of the most relevant tokens are Israel and Women.
- Finally, we want to refer to the strong relationship between topics 4 and 7. These ones, have references to Jesus, Father and Israel. This could show some lessons about the evangels.

TASK 2: MACHINE LEARNING MODEL

Due to the characteristics of our database, we decided to perform a classification task. We consider two different processes, the first one was predicting the testament of the text and the second was to perform a classification of the book.

FEATURE EXTRACTION & SELECTION

The first step we take was to create our prediction variables. We started by extracting the first variables by applying the text vectorization techniques previously applied and the different text representations. This led us to have a column for the BoW, TF-IDF, Word2Vec and Doc2Vec text representations. Moreover, we create a column for the vectorization of the LDA model obtaining a column for the decomposition by topics of each of the documents.

Finally, we perform different experiments with the different new columns for choosing which one performs better.

TESTAMENT CLASSIFICATION

Firstly, we started by doing the binary classification. We casted the variable testament (Old – New) to binary (0 – 1) and started trying different models:

SUPER VECTOR CLASSIFIER

The first model we tried has been an SVC. For obtaining a good classification, we applied grid search with cross validation for performing hyperparameter tunning for selecting the best combination.

We perform the classification task with all the columns created previously and we obtain different results.

- For the BoW, TF-IDF and word2vec vectorizations we obtained extremely high accuracies. This made as suspect that our model was completely overfitted. We also think that it is possible that it is easy for the model to get high accuracies as the label predicted is binary and the differences between Old and New testament may be very big and the amount of documents with which it trains is also high.
- For the doc2vec and LDA decompositions, we obtain considerably good accuracies of around 80%. These results were satisfactory enough.

MULTINOMIAL NAIVE BAYES CLASSIFIER

The second experiment carried out was using a multinomial naïve bayes classifier. This is a popular model in text classification, and it is quite fast not needing big amounts of data.

In this case we just tried this model with BoW, TF-IDF and LDA decompositions and the obtained results were satisfactory excluding the BoW representation which led to an accuracy of 98% showing an incredible overfitting.

LOGISTIC REGRESSOR

Finally, we consider the possibility of predicting the testament with a logistic regression. As it is known, this is a regression model which fits perfectly with our dataset since it is thought for binary classification tasks.

We obtain the best of the results, excluding again the BoW vectorization which stayed overfitted.

As a conclusion of this classification task, we observe how our models predicted considerably good the testament. This could be due to the good differentiation between the used words of the Old Testament and the New one. These differences are of course influenced by the time difference between the events and the strong difference of the treated topics.

FEATURE EXTRACTION & SELECTION

The first step we take was to create our prediction variables. We started by extracting the first variables by applying the text vectorization techniques previously applied and the different text representations. This led us to have a column for the BoW, TF-IDF, Word2Vec and Doc2Vec text representations. Moreover, we create a column for the vectorization of the LDA model obtaining a column for the decomposition by topics of each of the documents.

Finally, we perform different experiments with the different new columns for choosing which one performs better.

BOOK CLASSIFICATION

In this case we opted for classifying the book. This target variable is much more complicated for being classified since it has much more values and there are not that many differences between the different books. However, we tried again different models:

SUPER VECTOR CLASSIFIER

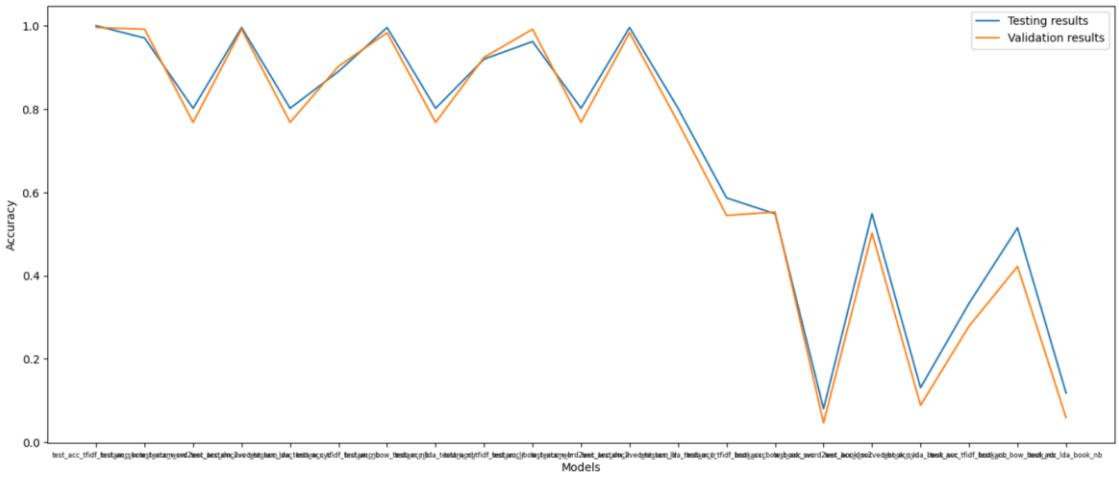
When applying this model, we obtained very different accuracies. On the one hand, we obtained some accuracies of around 50% or 60% with vectorizations like TF-IDF, BoW or Doc2Vec. On the other hand, we obtained some insignificant accuracies of around 10% when predicting using the LDA decomposition or the word2vec. This could be related to the mentioned issue of the small differences between topics and used words between books.

MULTINOMIAL NAIVE BAYES CLASSIFIER

The second experiment carried out was using a multinomial naïve bayes classifier. Again, we just tried this model with BoW, TF-IDF and LDA decompositions and the obtained results were again around 50% of accuracy besides the LDA which obtained an insignificant accuracy.

As a conclusion of this classification task, we observe how our models poorly predicted this variable. In our opinion, we consider that this is due to two factors. On the one hand, there are a lot of different possible values for our target variable. On the other hand, there are not strong differences between the used words of one book and other. However, it surprised us, how the worst feature for predicting this variable was the LDA topic decompositions. We didn't expect this because we though that the treated topics in each book will be very different.

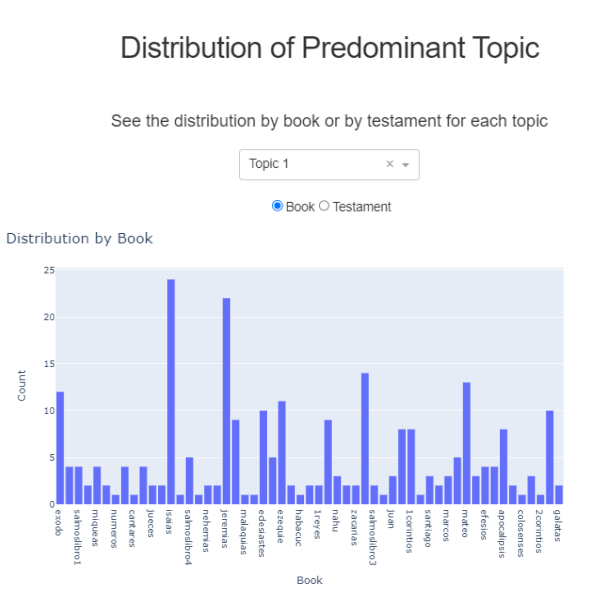
VALIDATION TECHNIQUES



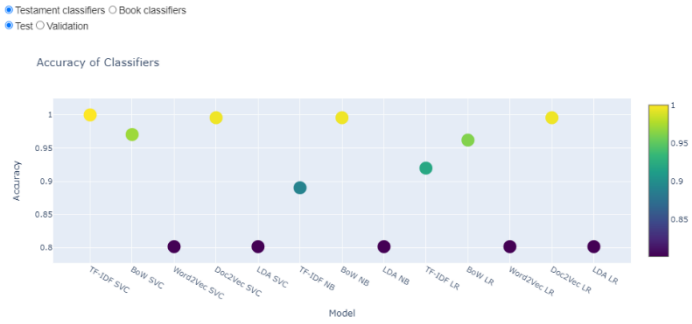
We have performed validation, testing and training to make sure we were not having any overfitting problem and as we can see in the plot above, the results are closely similar for validation and testing. We have to highlight the drastic decline of the accuracies we can see when we used the LDA topic decompositions for the book prediction.

TASK 3: IMPLEMENTATION OF DASHBOARD

Our first dashboard is a figure related to the predominant LDA topic of each document. The goal is to easily visualize the distribution of the topics in the documents. We have implemented a dropdown so that the user can select the topic to see its distribution. In addition, the user can select via a radio button to see the plot with by books or by testaments. This way, the chart is interactive and the user can get different plots from one unique chart. In this case we decided to analyse the distribution when the number of topics is set to 5. With this dashboard we can easily see the distribution in the different books or in the testaments.



The second dashboard creates an interactive scatterplot which shows the accuracy of the different models we used in the classification task. The plot can be filtered based on whether the classifiers were trained to predict the testament or the book and to see either the test or the validation results. The scatter plot shows the accuracy of each classifier as a point and the color of each point represents the accuracy score, with larger and darker points indicating higher accuracy.



The third dashboard plots a wordcloud based on the predicted books that were correctly classified by the different models in the classification task. The user can select the model wordcloud to display with a radio button. For this we previously stored the predictions for different models in different files, and the callback function selects the appropriate dataset and updates the graph. With it, we can easily see which books were correctly predicted most of the times.