Bachelor in Data Science and Engineering - 2020/2021
University Carlos III Madrid
Machine Learning Applications - Group 96

Final Project: Natural Language Processing and Classification of Spanish politicians tweets over the course of 1 month.

Sergio Aizcorbe Pardo
NIA: **100406602**

Bernardo Bouzas García
NIA: **100406634**

Ricardo Chávez Torres
NIA: **100406702**

Manuel Nuño García
NIA: **100406629**

Index

Task 1: Preprocessing, modeling and visualization

Task 2: Classification phase

User's Code Manual

Task 1: Preprocessing, modeling and visualization

Introduction

The aim of this project is to analyse and process **Tweets published by Spanish Congress politicians** during the last month (***07/04/2021 - 07/05/2021***) using **NLP**.

The tweets have been stored in a custom database with the help of **Twitter's API**. The text has been pre-processed and parsed to later obtain their semantic relationship. The analysis offers useful, up-to-date insights and, in addition, the use of this type of text will be **less** likely to contain spelling **mistakes** or abbreviations.

Scraping the data

In order to make use of Twitter's API, we had to justify through an exhaustive form the objective of the analysis and how these results will be used. As we are developing in Python, the **tweepy** library has been used which provides access to the entire Twitter RESTful API methods. Communicating with the API is done through several keys and tokens provided by Twitter and stored in ***config.py***. The main own-implemented main methods would be:

getTwitterByName(name, num_follow, verified): Searches the inputted name, gets all the users in Twitter that match that name and satisfies the required conditions (**minimum number of followers** and **user verification**). Before calling this method, we get the full names of the politicians via the function **wikiParser()**, which requests the html of the Wikipedia page of congressmen of the XIV legislature and returns the names and surnames of each politician and their respective party, using **BeautifulSoup** and **pandas**.

getTweetsByUser(user, last_n_months, since, until) method, which allows us to define a custom **time interval** to scrape over the tweets. We can define that interval using *last_n_months* parameter or selecting the number of days using *since* and *until* (i.e. {since=0, until=20}- last 20 days).

In order to store the information we requested, we have created a SQL database with three tables in which we save **politicians**, **parties** and **tweets**. Each table is represented in a Python model stored in *scrapers/models.py.* In each table we store relevant attributes and metrics (retweets and likes in the case of tweets), as well as a hash to uniquely define them.
The database implementation uses **SQLAlchemy** as its main library.

Parsing the data

Our main parsing function is **tweetsParser**(), which is initialized with a **dictionary** containing **parsing parameters**. Depending on the number of cores, it launches a **Process Pool Executor** to parallelize the whole process (concurrent.futures module). These parsing pipeline can be summarized in **five** different parsing features.

1	Removes the text that is not in Spanish (Catalan, Galician or Euskera) with langdetect.
2	Remove points, commas, symbols, and URLs using Regular Expressions, then lowercase everything.
3	If any specific word refers to a party, replaces the term by the specific party (case - sensitive)
4	When a term begins with @, checks if it belongs to a party/politician and if not, it removes the whole term
5	If remove_hashtag_word = True, removes any word beginning with #. If false, just removes the '#' symbol

NLP preprocessing pipeline

To perform the corpus pre-processing, we have defined a custom **NLPPipeline** class which loads the specific Spanish model of the **spaCy** library, and contains methods to compute and return the respective tokens and lemmas from the parsed tweets.

In this case, we are performing a semantic analysis of the topics discussed on the platform, so we deactivate Spacy's **parser feature**, disabling the syntactic functions and keeping the morphological ones associated only with **Nouns**, **Verbs**, **Adjectives** and **Pronouns**, for POS tagging. NLPPipeline uses a dictionary with parameters to enable or disable these features.

Additionally, we tested the results enabling and disabling the **Named Entity Recognition** feature. We saved the results in 2 different data frames, (*data_ner* and *data_non_ner*), and, after testing both models in advanced steps in the development, we concluded that the feature **performed poorly** and complicated interpretation rather than simply it.

Now, to streamline the next steps, the data frame has been updated and saved in *csv* format to continue the analysis in a Jupyter Notebook**, task1_nlp.ipynb.** We have decided to make this transition because, up to now, we have programmed and defined the fixed processes that we knew were necessary; however, the course of the following steps will now be based on an analysis of the different results obtained. In this way, using a Jupyter Notebook gives us additional freedom to test particular cells, print the results of specific approaches and conveniently visualize graphs for decision making.

This Jupyter Notebook contains some of the annotations and displayed graphs. that we will soon display, and, in addition, we will emphasize the why of the decisions and the different metrics that we have been using to study the performance of the outcomes.

Dataset ready and accessible

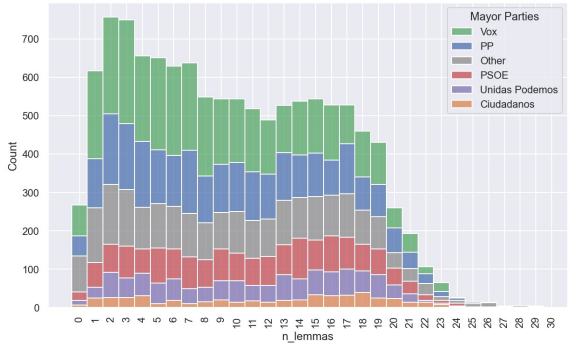
At this point in the project, we have taken special consideration with the sorting task that we must perform later. In order to use the output from this phase, we have splitted the data into a training and testing set. If we instead use our entire dataset in the LDA analysis, and then split it into train and test, the test will be conditioned by the full context of the entire dataset, which could cause **severe overfitting** when training the model. Therefore, the dataset has been splitted into random samples. As the dataset does not contain too many observations, especially after parsing, the chosen ratio to split the dataset has been **80% train - 20% test**.

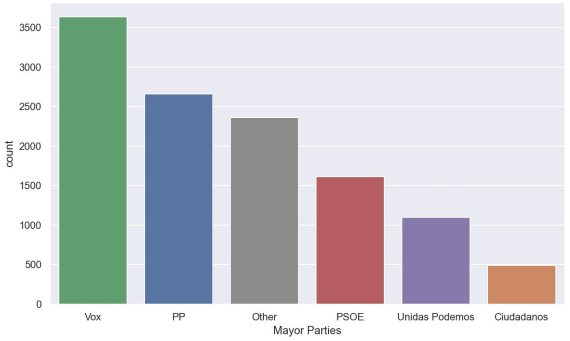
TOTAL DATASET	TRAIN SET (80%, random)	TEST SET (20%)
14812 tweets	11850 tweets	2962 tweets

In this step, a new parsing function has been added, **remove_accents()**. The function eliminates possible duplicates due to misspellings of accents. Now, two words will be the same with or without tilde.

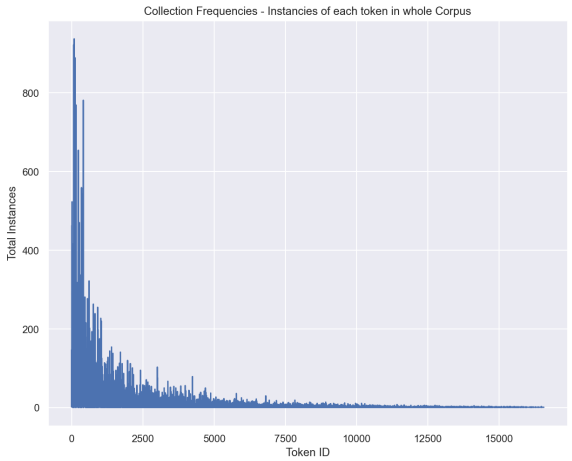
For further convenience in analysis and classification, as minor parties do not represent enough information in the dataset, we have incorporated a **new label** identical to that of parties, but merging the minority parties into a single category.

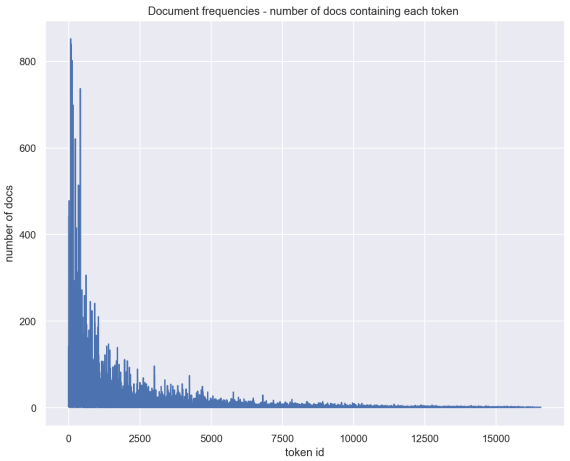
Cleaning out lemmas





The first thing we do is to study the distribution of lemmas over the total number of documents (tweets). With an average of 10 lemmas per tweet, a mode of 2 and a median of 9, we find a distribution with certain right-skewed lines. We observe a clear dominance of VOX over the rest of the parties, followed by PP. We observe how these two parties alone bear the maximum weight of right-wing representation this month on Twitter. In other words, 2 right-wing parties tweet the same relevant information as all the left-wing parties together, including PSOE, Unidas Podemos and most of the small parties. Inspecting the data, we decided to set the **minimum number of lemmas per tweet to 3**, and we are left with 10209 tweets with 3 or more lemmas that will provide **more and better context.**





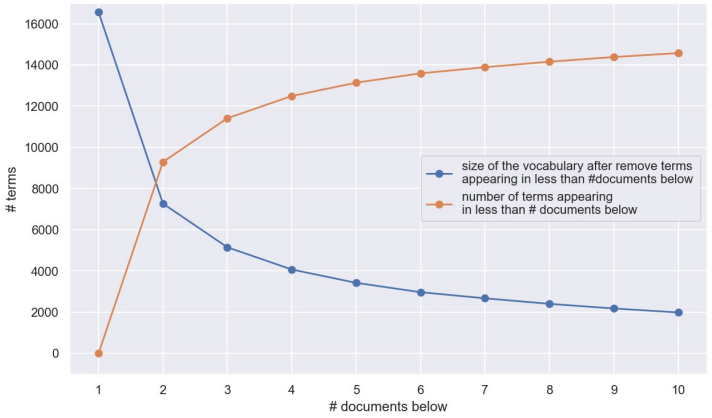
Next, we created the **token dictionary** to deepen our study of the terms, by means of the **Collection and Document Frequencies**. Nearly identical distributions are observed here, albeit with slight differences. Our discovery is that there are certain tokens that appear significantly more often in the whole corpus than others. Moreover, these same terms appear in a larger number of documents than the rest. Here we observe the first 9, with their number of occurrences.

madrid	gobierno	el	vox	pp	españa	psoe	gracias	año	libertad
937	922	889	781	769	654	559	543	525	470

We concluded that, when dealing with tweets whose content is very small, it is normal that the **most repeated terms** appear **less diluted**. Among these terms, there are some that can add value to the identification of the topic, such as "**madrid**" or " **libertad**" in the Madrid 2021 election campaign. Others that are too generic, obviously, should be and will be eliminated, such as words that should not even be here, like "**el**", or the names of the **parties**.

Technical considerations aside, if we study the number of lemmas that appear in the top, we again observe a dominance. Right-wing parties, in general, are **mentioned more often** than left-wing parties, either by themselves or by the opposition parties.
However, we should note that the NLP pipeline would be detecting "Ciudadanos" and "Unidas Podemos" as individual generic words. On the basis of the previously stated conclusion, a maximum proportion of documents has not been established. The repeating terms will be manually eliminated later in the stopwords removal process.

In addition, due to the chosen time interval, **Elections of Madrid 2021** is the topic that stands out the most, together with other latest news such as those related to the Covid crisis, climate change laws approval and threats in letters to two left-wing politicians.



However, we certainly considered that there should be a **minimum number of documents** in which a term must appear in order for it to contribute context. Since, again, our lemmatized tweets are quite limited, we studied possible relatively low thresholds for the *no_below* argument of the **filter_extremes()** function. By studying the aforementioned frequencies, we determined this interval between **1 and 10** documents, and tested the filtering with these values. Since there was a more abrupt drop between 1 and 2 documents than between 2 and the rest, all of the lemmas with less than 2 documents have been removed.
After removing terms only present in 1 tweet, we are left with **5151 terms** in the dictionary.

Finally, we elaborate the **Bag-Of-Words representation** of the corpus, transforming a list of tokens into a list of tuples (*token id, token number of occurrences*) with the doc2bow() function. In addition, we prepare the environment with which we will work on the LDAs, starting with an initial test one to perform the specific stopword removal.

	ngram	score	size
0	castilla leon	1102.629735	40
1	isabel diaz	1121.244140	34
2	derechos humanos	1667.930641	33
3	plus ultra	1327.908992	32
4	extremadura amanecer	1485.648485	30
...	...	...	...
265	competencias digitales	13618.444444	3
266	vara medir	5002.693878	2
267	vargas llosa	14707.920000	2
268	esqui navacerrada	1885.630769	2
269	bla bla	4902.640000	2

Additionally, we tried an **N-Gram detection** to identify possible pairs of words that appeared together and check if, unlike the NER, this algorithm performed well. After several tests, we decided to leave the threshold at 1100, and the results were quite decent, finding **270 bi-grams**. Still, it seems too many to us, but by making the threshold smaller we obtained even more pairs, this time with much less sense. It is interesting to see that the top 5, except for "extremadura amanecer" are indeed pairs of words that have been largely used together in this time period.

Specific stopword removal

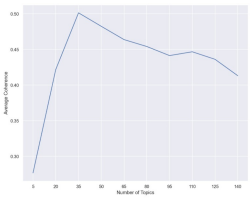
At this point, we have a B-o-W of fairly clean lemmas to which generic parsings have been applied. It is the moment to make several starting topic models, randomly initialized, in order to have a perspective of what terms we are dealing with. We then went on to make lists of words that, for this particular domain, were specific stopwords. Some of the words we identified and eliminated are the following, among others.

el	año	vivir	hacer	plan	sector	salir	gestion	resto	ir
yo	ley	gobierno	dato	politica	partido	favor	gracias	españa	señor

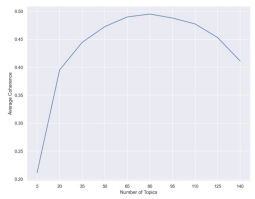
The word **Madrid**, although it is the most repeated term in the corpus, it only influences some topics when modeling. We have decided to leave it in the dictionary for this reason.

Coherence for number of topics

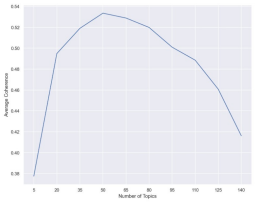
Next, we studied the coherence of the topics by applying the previous step several times.



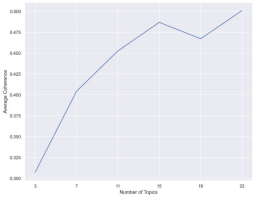
Just parties names as stopwords



Too many specific stopwords

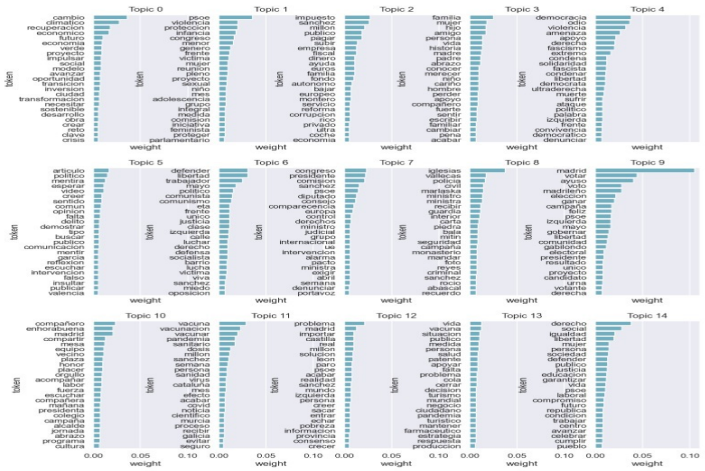


Best model for later classification task



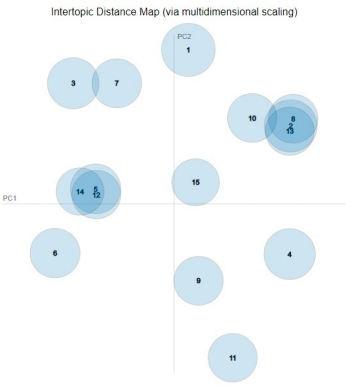
Best tradeoff model obtained (3-23 topics)

We knew that we were dealing with a very specific corpus, very biased towards some topics and terms in particular and whose tweets are really short. According to the coherence model, we should use 50 topics. This will be the threshold for the model that will give the future classification input. However, we believe that this number is not really representative, since the topics that are dealt with on Twitter, being more diluted, are fewer. In order to be able to label them correctly, we looked for the tradeoff between coherence and maximum number of topics in a somewhat less heuristic way, and decided to leave the **number of topics at 15.**



Topic model analysis

After obtaining these results, we use the **pyLDAvis** library to conveniently visualize the topics and to label them. (*Note: represented colors are not related to any particular party).*



1	Society's values and rights
2	Madrid Elections (candidacies)
3	Climate change
4	Hatred and threats to the welfare state
5	Government management
6	Tax increase plan for the European Union
7	I Congress on Childhood and Adolescence
8	Madrid elections (candidates congratulations)
9	General politics
10	Social values related to families
11	Threats to left-wing politicians (bullet on card)
12	National Covid Crisis
13	Working class defense
14	Evolution of the vaccination plan
15	Garbage topic

As we were at the most important moment of the Madrid elections, it is normal that related topics cluster very tightly. We find terms used in political campaigns, congratulations to the winners and social claims (freedom, family, defense of the working class, etc...).

Other social issues (climate change, childhood, etc)

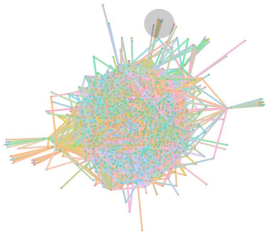
A very clear cluster is that of the government's management during these months, including the pandemic, vaccination, tax plan, etc.

More aggressive cluster related to allegations of hazards to society, terrorism, threats...

Topics that don't provide useful insights, including generic politics tweets and garbage topics.

With respect to the results obtained, our conclusion is that many topics that we see grouped in clusters in the image share a large number of terms, and therefore similarity, since the context with which we deal is very flexible, with very **small tweets**, and with certain **key language** that politicians use transversally on the platform to have more impact on their audience.

Graph representation



The conclusion inferred above becomes more evident when we try to represent the results in Gephi. To do so, we have exported 2 files, with the nodes and the weights, and tried to obtain an explanatory representation. Being small tweets, in which all politicians talk a little bit about everything, a clustered mass is observed, together with some outer clusters dealing with specific moments of discussion.

Task 2: Classification phase

In this section we will explain our methodology and approach for the implementation and evaluation of the performance of a set of classifiers. The goal is to predict the Parties label - one of the metadata available in our dataset - thus being able to classify a tweet into the political parties of its author.

Input data

We will be using the **TF-IDF representation** and the document **vectors provided by LDA** as input variables, comparing the performance of both. As we mentioned, we are not very satisfied with the performance of LDA, and this is reflected in the classification phase. To try to improve it, we have made multiple tests with LDAs with 15 and 50 topics (the reason is described in the previous page). The ones shown here are, again, the results with 15, since this is what gives the best performance. Additional plots and comments can be found in the Jupyter Notebooks **task2_classification_lda** and **task2_classification_tfidf**, where we separate our results for clarification purposes.

Visualizing the data

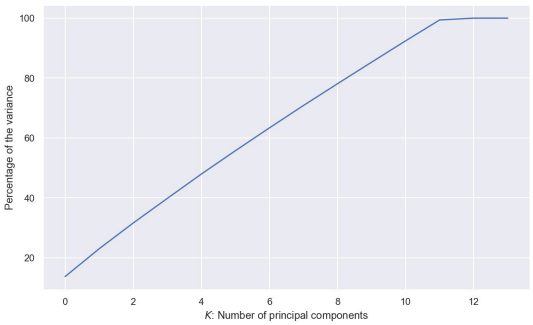
In order to have a better understanding of the data we are dealing with, which we know is not very revealing, we have decided to plot the first two components of the **T-distributed stochastic neighbor embedding** (t-SNE), which is a statistical method for visualizing high-dimensional data by giving each datapoint a location in a two or three-dimensional map.

We will only represent here the representation with this method, since other dimensionality reduction techniques such as PCA did not give enough suitable interpretable results. In the next section we explain it. In the first plot, we can see how the **different topics are presented in clusters**. In the second plot, we have represented the colors of the political parties. The conclusion we obtain from this representation is that **all political parties talk almost equally about all topics**, this being a consistent inference as we are studying tweets in an interval of 1 month in an area where relevant topics are broadly discussed from different political perspectives.

Dimensionality reduction

We have considered applying dimensionality reduction in order to observe trends, jumps, clusters and outliers. We wanted to study how our features behaved and whether any were dispensable or did not provide information.

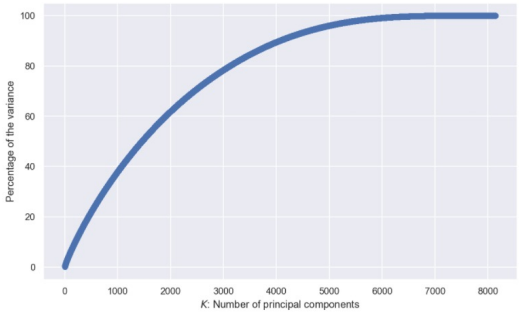
PCA Using LDA vector as input



It made us realise that it obviously didn't work well. This is due to the **nature of the LDA**, that creates the topics in such a way that the information contained in them is as uncorrelated and redundant as possible and that is why the **PCA does not perform well**. In fact, all the Principal Components seem to explain the same amount of variability except the last 2 (this is because Likes and Retweets have been explained among the other PCA components). Moreover, the weights of the Likes and Retweets of the last two PCA components **are shown as 0** (1e-18).

PCA Using TF-IDF vector as input

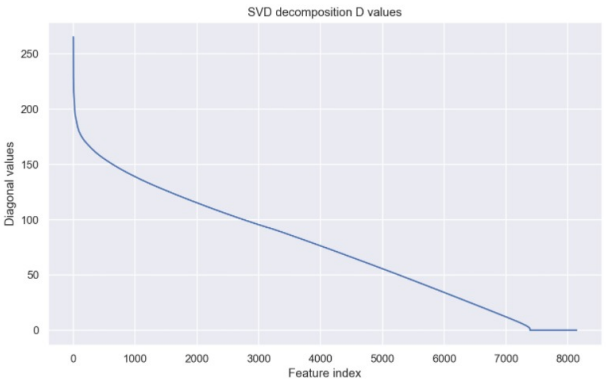
As it can clearly be seen from the graph, PCA was not very useful either with TFIDF. This is due to the fact that our **TFIDF matrix is excessively sparse**. This happens because of the brief nature of the tweets: there are very few words in each tweet, and they do not appear repeatedly too much in other tweets. Thus, we are left with a matrix with few non-zero values.



Therefore, we conclude that PCA is not a suitable technique to try to reduce the dimensionality of our data, either in LDA vector representation or in TFIDF vector representation, for the two reasons mentioned above. In order to deepen our approach, we will try to perform an Latent Semantic Indexing for the same purpose.

LSI using TF-IDF vector as input

Latent Semantic Indexing is a technique in natural language processing for analysing relationships between a set of documents and the terms they contain. When we apply it to our dataset and plot the **diagonal matrix, S,** values, we can see that the first 150-ish values are the largest ones, with a clear difference with respect to the rest. Knowing this, we will keep the **first 140 features from the D matrix** and continue our classification phase (for TFIDF) with our dimensionally reduced data in this way.

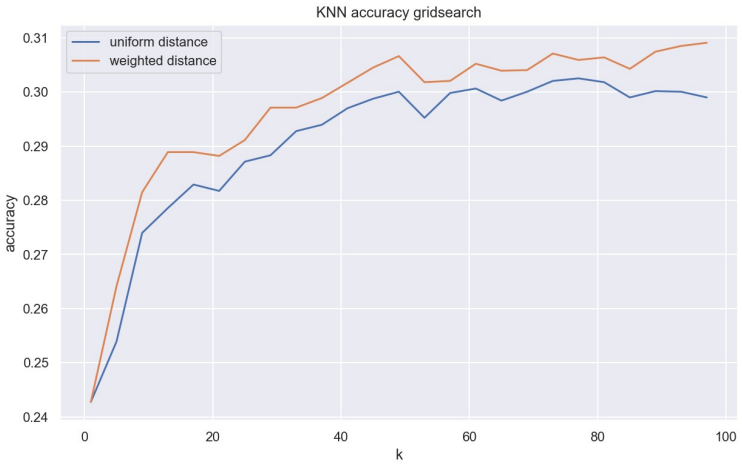


Classification using LDA vector representation

To perform the training of the models and the subsequent prediction, we considered it appropriate to slightly modify our labels. Instead of having absolutely all political parties, which is our target to predict, we have decided to label the tweets as belonging to PSOE, PP, VOX, Unidas Podemos, Ciudadanos or an aggregate group of Other Parties..

K-Nearest Neighbors

The KNN algorithm tries to find the K points closest to a given point in order to infer its value. This algorithm belongs to the set of supervised machine learning techniques. We start our approach with **k = 3 neighbors**, to test the performance, both with the test set and the validation one. **Initial Validation Accuracy = 0.258**, **Initial Test Accuracy = 0.237**



As we have commented before, as all politicians talk about the same topics, the results of the classification are quite mediocre. This, we believe, would not happen if the tweets were 2 or 3 times higher in content, so the vector representation of the topics would be clearer. Next, we perform a more exhaustive analysis through a GridSearch to find the hyperparameters that maximize our accuracy, as far as possible.

The optimal number of neighbors k=97, with an accuracy of 0.30910141617428816

SVM (RBF kernel)

Following the approach used for the previous KNN, we performed a GridSearch to check if there is any combination of hyperparameters that allows us to obtain better results using the Support Vector Machine algorithm. Trying with the values **c** = [**0.1,1,10,100,1000**] and **gamma** = [**0.001,0.01,0.1,1,10**], we obtain these results **{'C': 10, 'gamma': 0.01}, Accuracy = 0.31**

```
Confussion Matrix
[[  0  14   0  63   0   5]
 [  0 107   0 202   0  63]
 [  0 136   0 189   0  71]
 [  0  87   0 141   0  42]
 [  0  23   0 139   0  31]
 [  0 100   0 365   0  46]]
```

Decision Tree

We considered doing so since we thought that, by giving us the classification rules, it would perform well. In the initial test, we observed that this model was completely useless, it did not explain anything since it took only Retweets and Likes as a reference. After leaving them aside and testing again, we observed how it classifies poorly, barely distinguishing between VOX and PSOE. Definitely not useful.

Classification using TF-IDF vector representation

In light of the results obtained in the previous phase, we decided to focus our efforts on the design and implementation of a KNN using the TF-IDF vector as input.

This time, after performing an exhaustive Grid Search, we arrived at the optimal result of neighbors with k=15.

Best neighbor parameter value	Best accuracy obtained
15	0.43

Conclusions about classification

After all, we conclude that due to the nature of our dataset, a classification task of predicting to which party a tweet belongs is not an easy task and probably not the most enlightening question to make.

We believe the low accuracy with these models does not have to do with the classification algorithms but instead with the data we provide them. Tweets are very short documents with very few lemmas, so the chance of a word appearing in many documents is very reduced.

We have come to the conclusion that by increasing the number of documents scrapped, we could have been able to achieve better results, but the needed amount of tweets to achieve this would take too much time to scrape, lo que suponía un coste computacional que no podíamos asumir.

Another idea for improvement would be to somehow expand the content of these tweets. It has occurred to us now, in view of the results, that an interesting approach could have been to scrape twitter lists associated with different topics, in which the tweets of politicians or accounts associated to those topics would be collected.

We could try all the feature selection and extraction we wanted with LDA and TFIDF, but if our documents come from texts that are too short, these vector representations will not be very useful when it comes to classification information depending on them.

User's Code Manual

Our code is expected to be run from the jupyter notebook files in order to visualize the analysis we performed on the scraped data and the results. Nevertheless, we will explain here how the code is structured and what should be taken into account when running it.

Folders

The main code is organized in 4 folders:

1. The *main* directory, which contains the notebooks, the required libraries to install and the main.py code.
2. The *data* directory, which is a subdirectory of the main one, is where we store all our dataframes, such as the scrapped and parsed dataset (data.csv) or the lda dataframe (final_tm_df.xlsx)
3. In the *models* directory, another subdirectory of the main one, we store all the saved models from the notebooks, such as LDA models and Bag of Words.
4. The last subdirectory is *nlp,* which contains all the scraper code and the NLP pipeline, which takes the raw scraped tweets and parses them.

Requirements

Before attempting to run any code, one should have installed all the required libraries stated in the "*requirements.txt*" with the specified version.

Running the code

Our NLP and analysis is organized in notebooks, as stated before. We are going to state now what should be taken into account when running them. In order to visualize the results, one must run the respective notebook:

Task 1 notebook

As the raw data only needs to be parsed once, there is no need to re-create the csv file that we use for our NLP, so those lines are commented in order to avoid misruns by the user. The "Creating the csv" cell should only be uncommented and rerun when the raw data changes (new scraping) and it needs to be reparsed. It also needs to be taken into account that the user should set its own *mallet_path.*

Task 2 - LDA and TFIDF notebooks

These notebooks are purely for the analysis of the data obtained from task 1. The only thing to be taken into account is the time that it takes to run them, as we perform a few algorithms that require some computational time.