Analysis of tweets from the 2017 presidential campaign

Master 1 Medas – CNAM Nantes

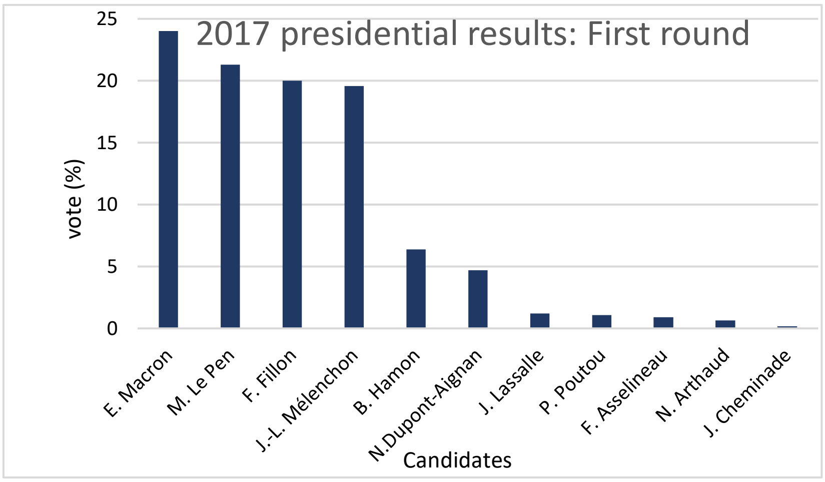
Christine Fouque
Lucas Boscherie
Cyntia Pérez

Introduction

Most of the data we encounter in the real world is unstructured datasets like tweets. Twitter has become an interesting indicator to know the reactions of its users on several issues like social, political, economic, environmental, etc. Furthermore, Twitter has become a central tool of communication for the political world, allowing direct communication with voters.

Our analyses have enabled us to bring out the main subjects of the French presidential campaign that took place in 2017. The performance of different techniques like normalization, tokenization, and lemmatization, stemming and clustering were crucial because these are the masterpieces of any automatic language processing tool.

It seemed interesting to us to parallel our analysis with the results of the 2017 presidential elections in order to see during our study what seems the most effective in terms of communication to convince voters. The election results of the first all are shown in the following graph.



Graph 1: 2017 presidential results: First round

I. Data Cleaning

The data are tweets and information from 42,923 tweets written by the eleven candidates for the French presidential election in 2017 from the start of the campaign until the election of Emmanuel Macron.

By looking at the data, we have noticed that some tweets start with "RT @SomeTwitto". These are the tweets that the candidates have retweeted. So, they have not been written by a candidate, they are a kind of quote. The following exercises are focused essentially on thematic, and stylistic. We have decided to remove those tweets from the dataset. This is now 31 910 tweets and only concerns tweets written by candidates.

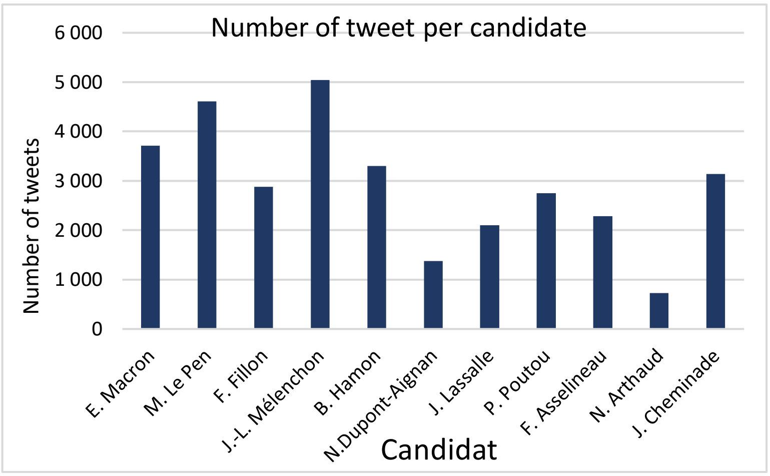
For this project, we've made the choice to work with Python, thanks to libraries NLTK, spaCy and scikit learn.

II. Exercise 2: Stylistic Analysis

Twitter is a great way to figure out how people feel about current events so tweets have become a powerful tool and an integral platform during French presidential campaign. In this study, we present the detailed description of stylistic variation amongst candidates. For each tweet we know the author's Twitter handle, tweet content and tweet tool.

In general, tweets from the same author tend to be similar and each candidate use certain patterns to communicate with their followers and try to convince them to vote for them.

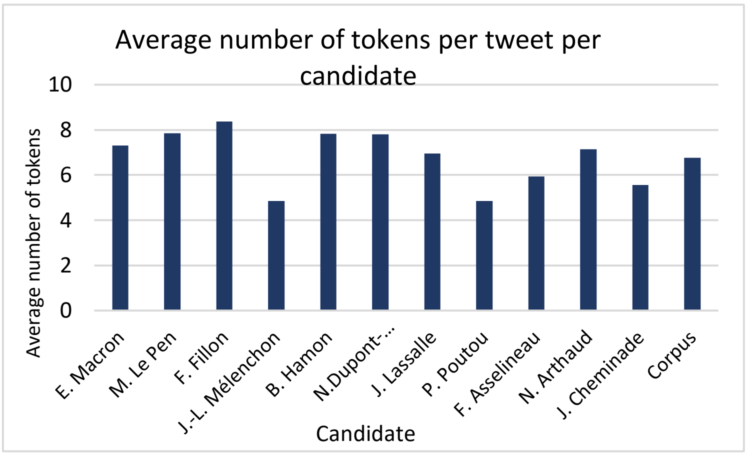
The goal of this study is therefore to discover the most important general patterns of stylistic variation on the candidates account. To carry out the stylistic analysis of a presidential candidate of 2017 through his tweets, we must carry out various pre-treatments. These are made in Python using two open-source libraries: spaCy and NLTK.



Graph 2 : Number of tweet per candidate

The easiest information to analyze about authors' writing styles is quantity. So we looked at the distribution of tweets among the 11 candidates. The graph represents the results obtained We can clearly see that the most prolific author presents on this network are Jean-Luc Mélenchon (5048 tweets) and then Marine Le Pen (4605). In contrast, Nathalie Arthaud is very unproductive because over the same period that publishes only 725 tweets, 5 times less.

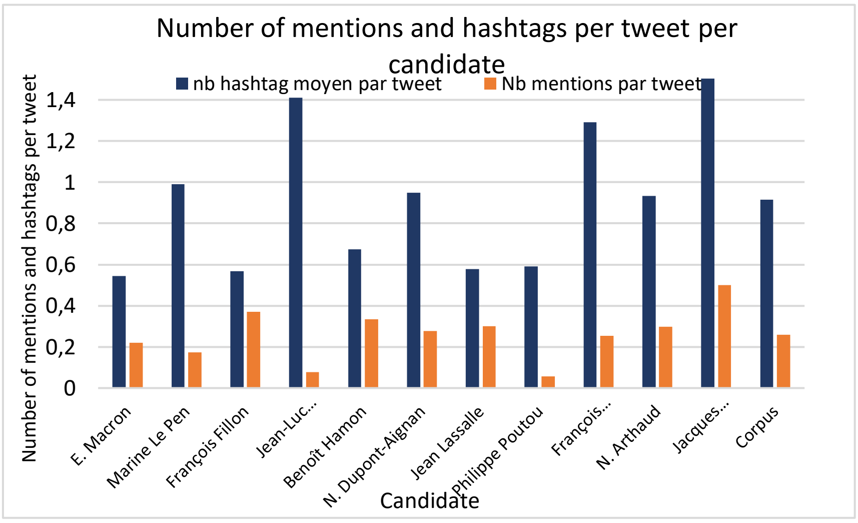
After this first observation, the tweets are pre-processed in order to obtain more evidence. The first step is tokenization. By using TweetTokenizer and spaCy's tokenizer we were able to convert the stream of words into tokens and then they can thus undergo other linguistic treatments. At the same time as this operation, to make it easier to match similar words, the case is homogenized and the whole tweet is lowered. Then the tokens are cleaned, or more exactly filtered. Indeed, to limit noise and recover only the most instructive elements, punctuation is deleted, spaces as well, as urls. To retain only the words which provide useful information, it is traditional to remove the stop words. This generally allows improves efficiency of text mining applications. After, we observed another stylistic indicator to compare the candidates: the number of tokens (informative) per tweet. For this study, we also removed the terms: "rt", "a", "do", "must", "if" and "that" from the list of stop words. We tested the stop word list of spacy and NLTK. This last one seemed more efficient for the project.



Graph 3 : Number of tweet per candidate

The graph represents the results obtained during this stylistic analysis. We can see that the average for all tweets in the corpus is 6.8 tokens per tweet. Candidates who finish higher in the lead tend to have more than the average number of tweets. They seem to use more significant words than the average. Jean-Luc Mélenchon, our most prolific author in number of tweets, ends up last in terms of number of tokens per tweet with Philippe Poutou. The average is strongly influenced by JLM and its 5,000 tweets. This one is a candidate with an atypical profile. Francois Fillon is the candidate with the highest average.

It seemed relevant to us to look for the @ and # signs which have real meaning on twitter. Indeed the @ is used to mention a person, and the Hashtag(#) to signify that it is a keyword, useful for searching on twitter by subject. So we have quantified the number of @ and #. This indicates the way in which the candidate chose to communicate with his followers and is therefore part of his author style. The results are grouped on the following histogram (graph 4).

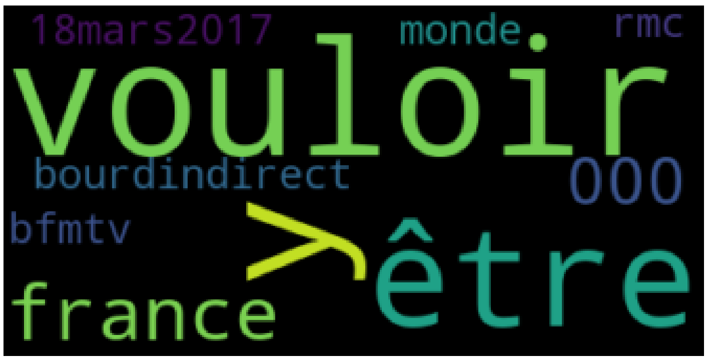


Graph 4 : Number of tweet per candidate

On average, candidates use 0.96 # and 0.26 @. We can observe some overlap among the candidates. The number of @ and # per tweet are quite similar to those of the average for N. Arthaud, Marine Le Pen and N.Dupont-Aignan.
Philippe Poutou uses these two indicators less than the average, unlike Jacques Cheminade. Francois Fillon, Benoit Hamon, Jean Lassalle and to a lesser extent, E. Macron seem to appreciate the use of mentions, but not so much the Hashtag. Again J-L. Mélenchon stands out with extreme use of the Hashtag and minimal for mention.

In order to increase the quality and the sense of the information it is interesting to use lemmatization. This consists in reducing the inflected forms of words to their basic form. The similarities are therefore more evident and it is easier to find the real meaning of the text. The search for the most frequent lemma by candidate seemed to us much more perceptible than those of tokens for which the risk of redundancy is high. To represent the most frequent lemmas by candidates we have chosen word clouds created via the wordcloud library for Python. You can see just below an example of a word cloud obtained on the whole corpus. The word clouds of each candidate are presented in the appendix, at the end of the report, with a summary table.

Wordcloud of "Corpus"



Picture 1: cloud of the most frequent lemmas in the corpus

The most recurrent lemmas in the corpus logically find is the word: ''France''. We also find the media mainly used during the campaign (bourdindirect, rmc, bfmtv) and the action verbs: "vouloir " and "être". The numeric characters "000" probably correspond to all of the facts listed by the candidates.

When we observe the most frequent lemmas candidates, we see several trends as candidates both in the use of the tool twitter but also in the way of communicating with voters.

With the 6 candidates who finished at the head of the first round, we observe common lemmas: "France, français, vouloir" these are words traditionally used in politics that are found on the word cloud corresponding to the corpus.

Despite differents politicals currents and ideals, Marine Le Pen and Emmanuel Macron use the 8 same commons lemmas : « Français », « France », « vouloir », « européen » / « europe», « pays », « politique », « y », « pouvoir ». In Marine Le Pen' lemmas, we also discover ''Macron'', who corresponds to the tweets of the second round.

Candidates from smaller political parties tend to use twitter to often talk about themselves or their own party. Indeed, we can see for the last candidates of the first round:.

- P. Poutou :"poutou, jevotepoutou, npa"
- F. Asselineau "asselineau2017 ,upr, françois"
- N.Arthaud : "arthaud , nathalie, lutte "
- J.Cheminade: " @jcheminade, cheminade2017, cheminade jc2017 "

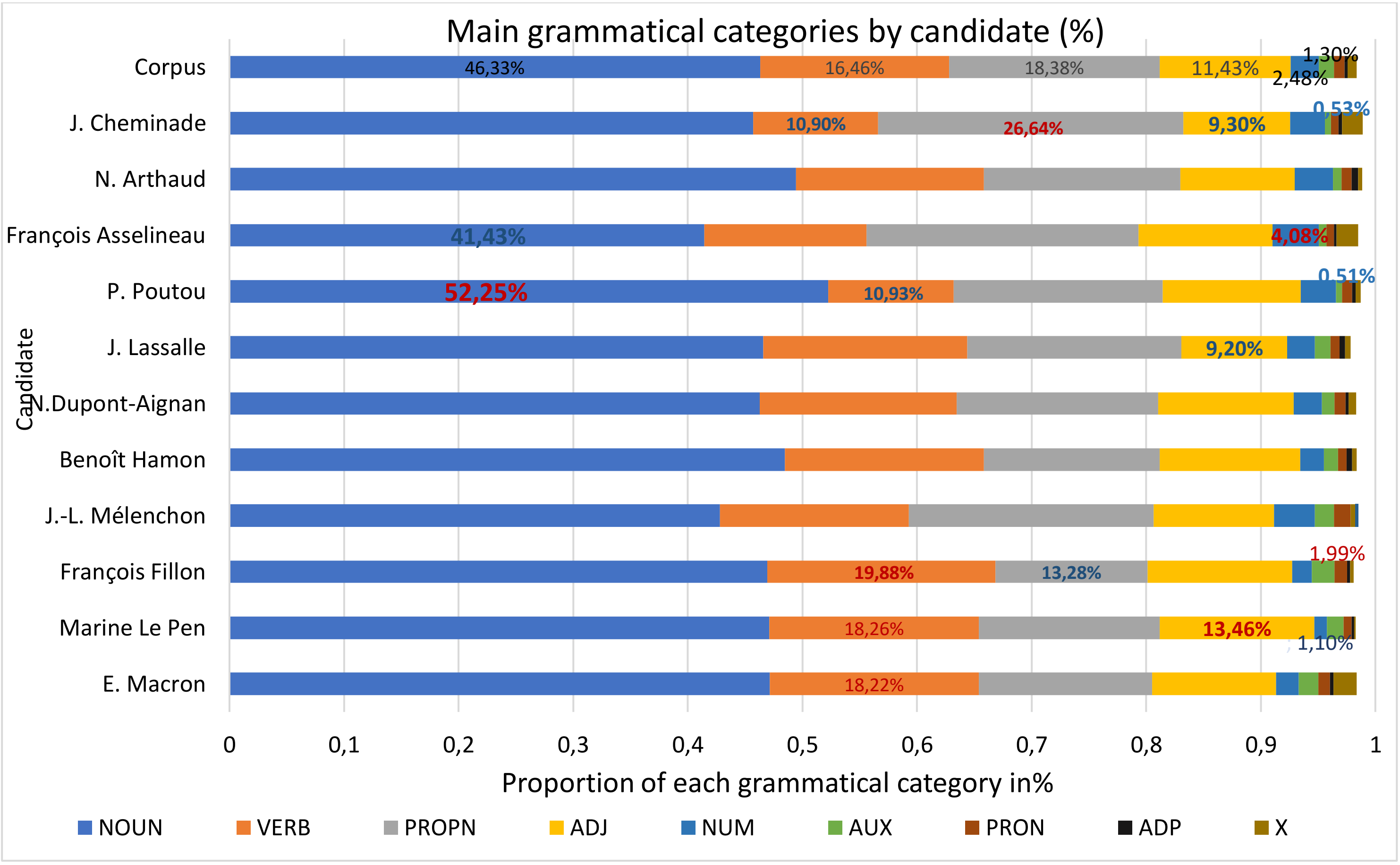
Some candidates stand out a little more from the others :

- All of the candidates use twitter to speak to their electorate and convince voters in general. Nathalie Artaud seems to do it more intensely. She uses fewer general terms than the other candidates but more words that target the traditional electorate of her party "workers struggle". We see the words : "travailleur », » lutte », « travail », « salarié » , « patronat », « classe » , « gouvernement ».
- Another candidate who stands out is J. Lassalle. In fact, he uses words that do not necessarily seem related to an electoral campaign: "marche", "ami", "bonjour", "lamarche" ,"ami",

”retrouver”, “rencontre”. He is a candidate who give the impression to play close to his constituents.

- Jean-Luc Mélenchon who appears to like to do differently from the other candidates uses twitter to talk about these interventions in the media, so we find :"bourdindirect, bfmtv, rmc". He looks to like to rely on big numbers to argue and convince because we find “000” in the top 10 of the most frequents words.

To observe the level of language of the authors or see if we can determine patters, we proceeded to a grammatical categorization of the tweets which is rather easy with spacy. Also to compare the candidates with each other, the corpus as a whole has also been categorized in order to serve as a reference. The figures in blue indicate the minimum values and in red the high values by category.



Graph 5 : Main grammatical categories by candidate

Interpreting this type of information is relatively complex. Always to compare the authors between them, the categorization is also done on the global corpus.

We first observe that all the candidates have the same first 4 categories, the order is not always quite the same. Almost 50% of nouns, then verbs, proper nouns and adjectives. The candidates arriving at the head use more verbs (and auxiliaries verbs) than their proper names, unlike those arriving at the end. Again we find Jean Luc Mélenchon who does differently from the others. This one is all the same in 4th position used more proper names than verbs.

When observing the digital category shows that Marine Le Pen is one that uses fewer digits. By cons, it uses more adjectives that all candidates. J. Cheminade and J. Lassalle use few adjectives, but proportionately more figures than the other candidates. The other categories are more difficult to analyse, it should be watching carefully as they contain.

For the analysis of named entities, we used the default spaCy package. This one gave us a lot of errors. After research it turned out that for French it is not ink developed enough. Another library called polyglot could have been used to carry out this type of analysis. Due to lack of time, this has not been finalized.

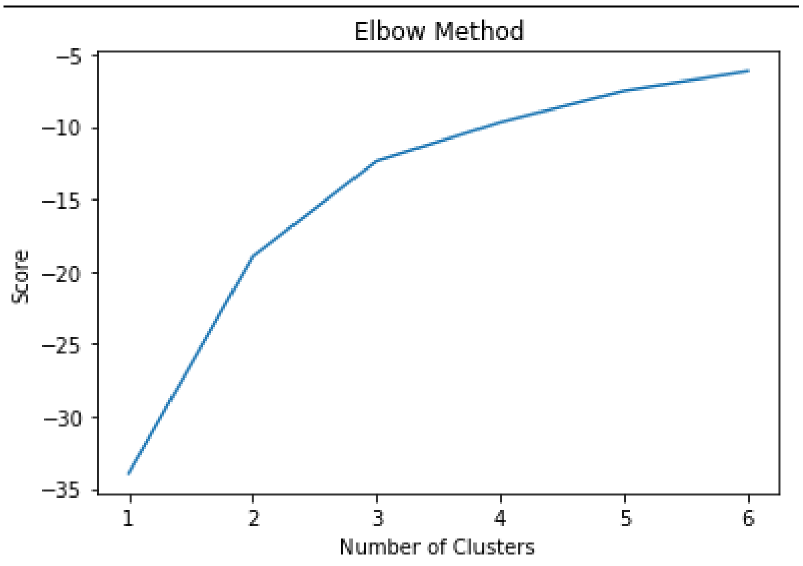
It lets us find the root word or the root of any given word. We can have multiple algorithms to do stemming but to work with our tweets in the French language, we have chosen FrenchStemmer. This is has its advantages because we can now count the frequency of each word occurring itself or in any of its derivation forms, any of its morphological variants.

III. Thematic research from tweets of each candidates

For this short analysis we will look at tweets of the eleven candidates, explore the dataset, and use kmeans a relatively simple machine learning algorithm, to extract topics from similar tweets. The goal of this task is to identify the main topics and top terms per cluster during the presidential campaign.

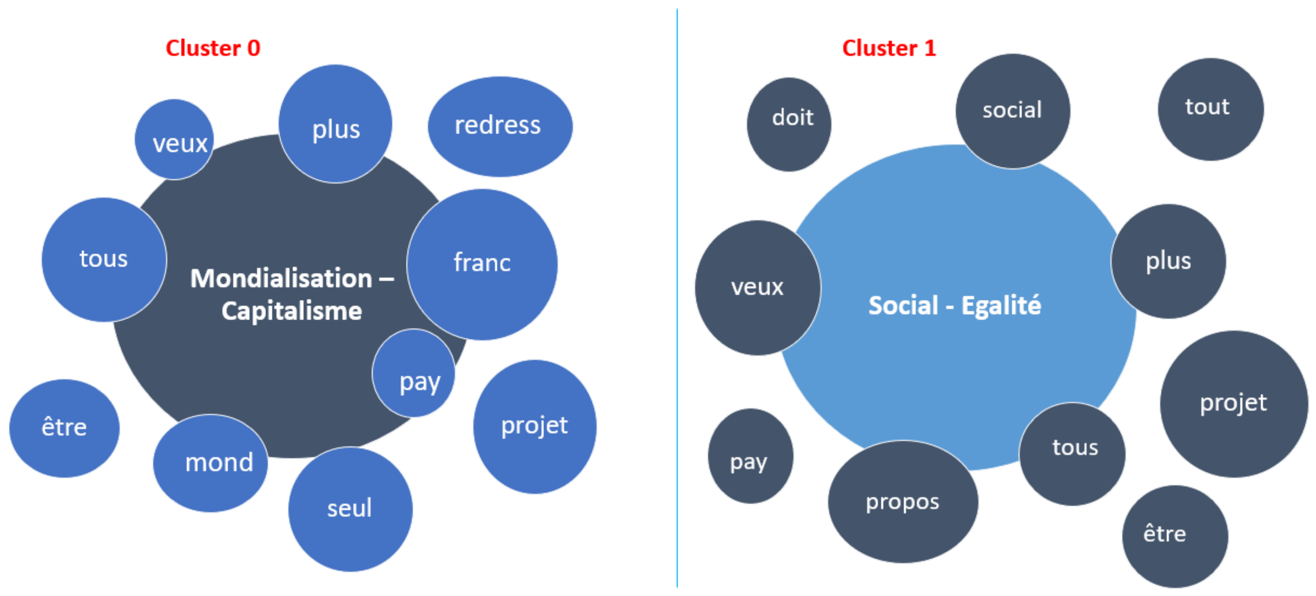
Clusters of all candidates: To begin, we have implemented a kmeans algorithm on all the tweets of all candidates. To use the data preparation made in the previous exercise, we took the list of stems about each tweet. With those lists, we have rewritten a string with only those stems. Then, we tfidf vectorized this document before applying the kmeans algorithm.

The main drawback of the kmeans algorithm is that we need to specify in the input the number of clusters we want. Using the elbow method, we have trained 7 kmeans algorithm with a number of 1 to 7 clusters in each one. We then plotted the score to obtain the following for each algorithme :



Graph 6 : determination of the number of clusters by the elbow method

Seeing that we have a gap around the 2 clusters, we have decided to keep these “optimal” number of clusters. In the 2nd picture we looked at the top 10 stems in each group.



Picture 2

In both clusters we got the stems “veux”, “plus”, “projet”, we can deduce from this that all candidates want “something more” for their country.

We had to deduce a theme for each of these clusters, we could say that the first theme is more about globalization, capitalism and recovery (“mond”, “pay”, “redress”). While the second is about social and equality (“tout”, “tous”).

To analyze in more detail using the kmeans method, we decided to create two clusters for each candidate, the list of the main stems is indicated in the following table:

Nathalie Arthaud	Jacques Cheminade	Jean Lassalle	Jean-Luc Mélenchon	François Asselineau	Benoit Hamon	Marine Le Pen	Emmanuel Macron	Philippe Poutou	Nicolas Dupont-Aignant	François Fillon
travail	projet	facebook	franc	européen	franc	européen	travail	droit	franc	franc
lutt	économ	amis	droit	programm	gauch	peupl	europ	social	soutien	social
salari	modernis	photo	social	réunion	droit	national	econom	meeting	démocrat	lutt
ouvri	financi	Itinérair	nucléair	franc	travail	franc	franc	travail	débat	projet
manifest	youtub	march	européen	publiqu	européen	pouvoir	projet	solidar	réunion	redress
franc	facebook	soir	constitu	soir	inégal	mélenchon	publique	public	publi	national
travailleur	meeting	franc	indépend		femm	grand	renouvel	lutt	polit	pay

Table 1 : list of the main stems per candidate

Cluster Nathalie Arthaud: when we look at the top terms of this candidate, we can identify work, worker, and protest. Mrs. Arthaud represents the extreme left-wing, so her tweets are focused on the working class.

Cluster Jacques Cheminade: This politician was focused mainly on the following topics: economics, revamp, and social networks such as Facebook and YouTube. We can assume that communication through social networks is an important part of his communication strategy during the election campaign.

Cluster Jean Lassalle: The main themes identified in this cluster are meeting and Facebook. We suppose that this political figure has mainly focused on calling his followers to his party's events and has used Facebook as one of his most important advertising platforms.

Cluster Jean-Luc Mélenchon: The candidate of "La France Insoumise" (Unsubmissive France), an anti-capitalist anti-system anti-European movement of the far left has claimed several issues as social system, independence, Europe, and nuclear power.

Cluster Francois Asselineau: in this cluster we can identify two central themes Europe and meeting.

Cluster Benoit Hamon: this social candidate has focused on the broad lines of the socialist’s current, social system, law, women, and equality.

Cluster Marine Le Pen: this politician made it through to the second round of the presidential election in 2017 and her big themes are Europe, French, nationality, or nation.

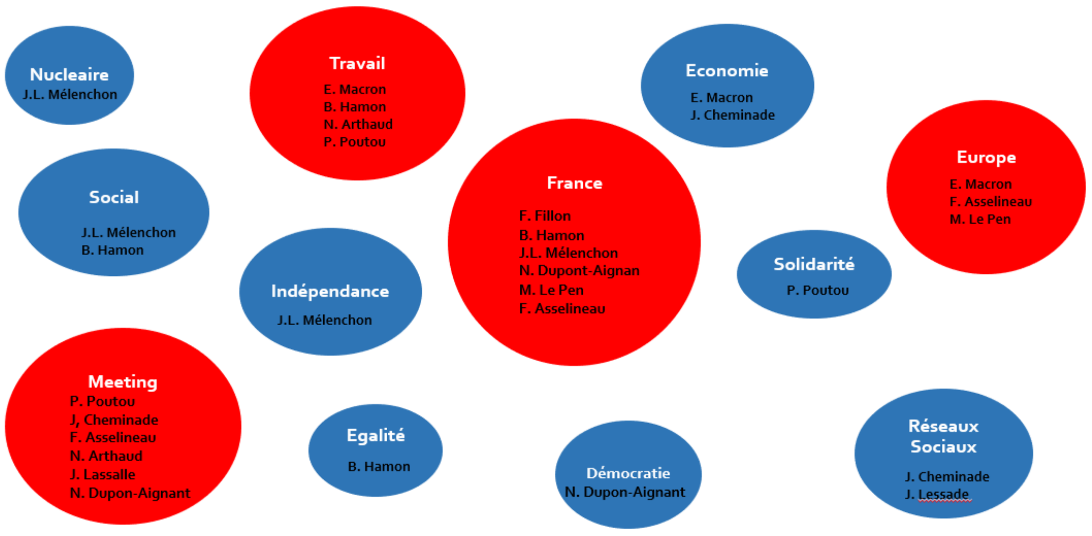
Cluster Emmanuel Macron: Macron's case stands out from the rest in so far as he is neither extreme left nor extreme right but – if the expression is not a contradiction in terms – extreme center. His cluster is one of the most eclectic with stems like work, Europe, France, public, economics and renewal.

Cluster Philippe Poutou: meeting, solidarity, and demonstration were his top preferences.

Cluster Nicolas Dupont-Aignant: Another small and more recent sovereigntist party, is Debout la France (Stand up France), led by this former Gaullist parliamentarian. Mr. Dupont-Aignant was mainly focused on the subjects: France, democracy, and debate.

Cluster François Fillon: when we look at the top terms of this Republican candidate, we can identify France, social and nationalism.

The cluster we illustrate below, allows us to identify the most repetitive issues that were addressed during the presidential campaign. We note that the candidates of different political currents are mainly concentrated on four themes: France, meeting, work, and Europe. The themes less present in the tweets sent were nuclear energy, independence, equality, democracy, and solidarity.



Picture 4

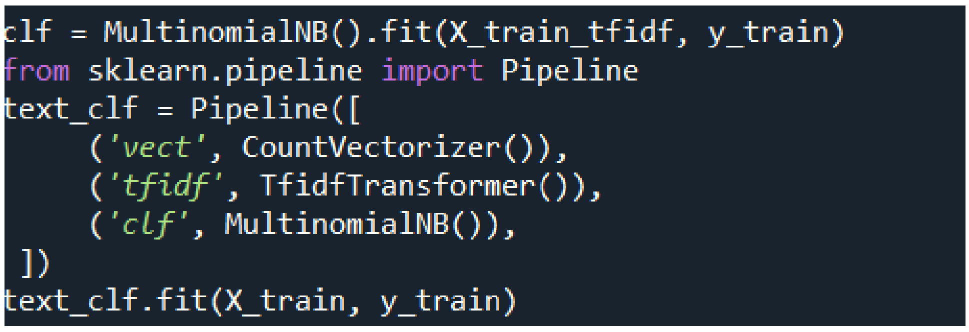
IV. Candidate's identification from tweets

The last part of this interesting analysis is to train and test a tweet classifier. The purpose is to decide, from a tweet, which candidate is the author.

To do so, we have trained several classifiers, using a train and test method. The test sample size is 33% of the total dataset.

1. The first one is the K-Neighbors Classifier. In this one, we have only included quantitative variables: retweets number and token number in the tweet. The target variable is the Author. We have chosen to input the algorithm with 10 neighbors. We obtain a low accuracy of 28%. But looking at the input we have entered, it is not so low, the retweet number is kind of correlated to the popularity of the candidate and the number of tokens is correlated to the author's style.

2. Then, we implemented a Naïve Bayes Multinomial Classification. This time, we used the original tweet. By using this, we had to tfidf transform the tweets, with CountVectorizer and then, TfidfTransformer (from sklearn library). Once this was done, we created a Pipeline (also from the sklearn library) to simplify the use of those transformations and applications to the classifier:



```
clf = MultinomialNB().fit(X_train_tfidf, y_train)
from sklearn.pipeline import Pipeline
text_clf = Pipeline([
     ('vect', CountVectorizer()),
     ('tfidf', TfidfTransformer()),
     ('clf', MultinomialNB()),
 ])
text_clf.fit(X_train, y_train)
```

Picture 5

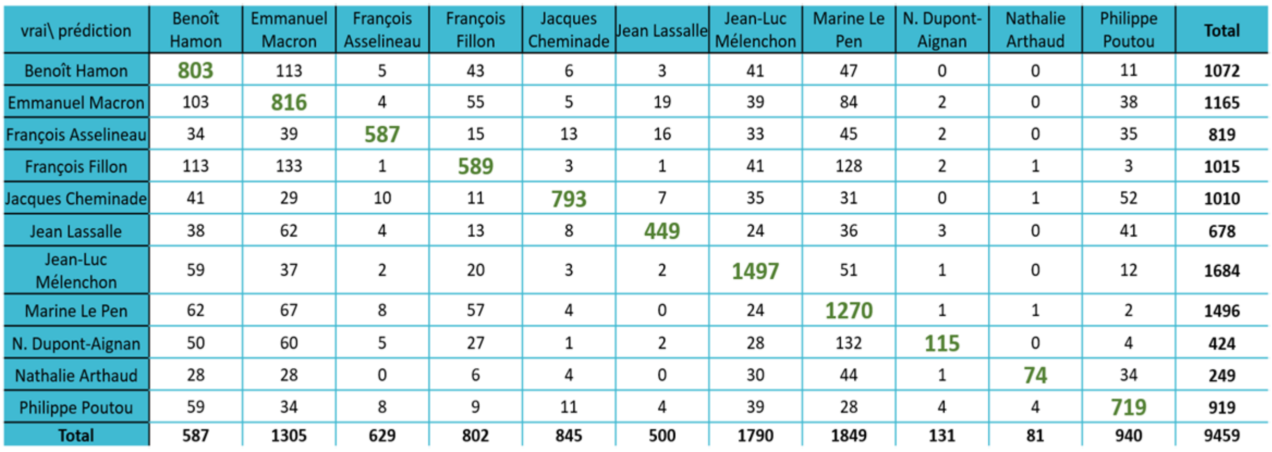
By applying this model to our test data and calculating the accuracy, we obtained an accuracy of 58 %.

3. The next model is a simple Naïve Bayes Multinomial Classification. This time, with countVectorizer, we created a bag of words. But we did not tfidf transform it. And we trained the Multinomial NB classifier. This time, we obtained an accuracy of 62 %.

4. Another model is the SVM (support vector margin). We used the svc function, still from scikit learn. It's the best model with an accuracy of 73 %.

5. Lastly, we trained a neural network: A Multi-Layer Perceptron classifier. We used a lbfgs solver and 5 hidden layers. But it was the worst accuracy with only 16 %. We increased the number of hidden layers to 10 and we obtained an accuracy of 51 %. By increasing this number to 20, we observe over-learning, with an accuracy of 16 %.

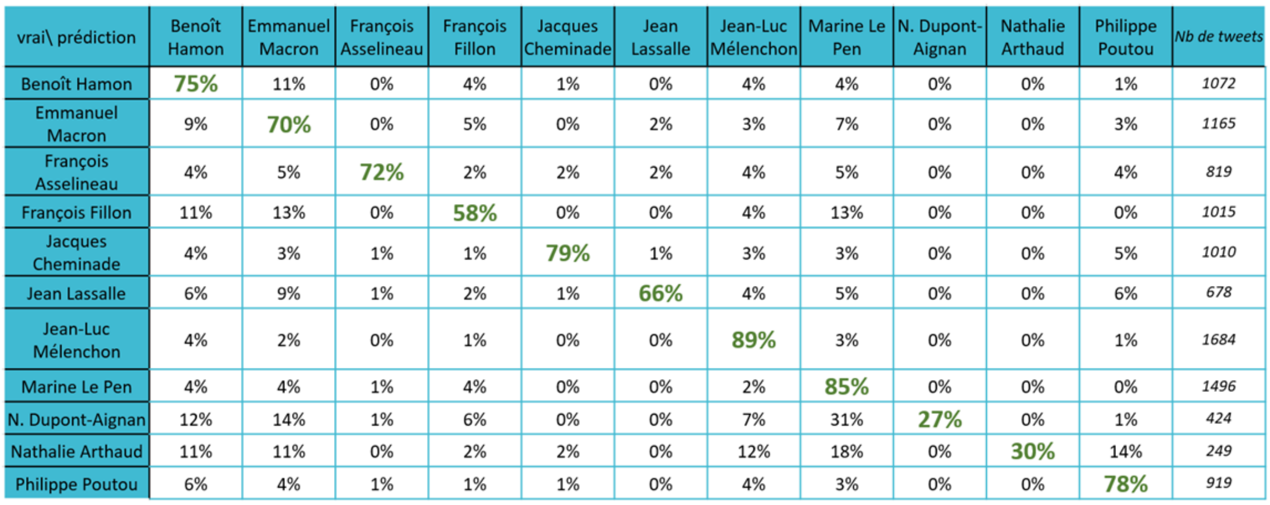
So, the SVM model has the best accuracy. Let us look at its confusion matrix.



vrai\ prédiction	Benoît Hamon	Emmanuel Macron	François Asselineau	François Fillon	Jacques Cheminade	Jean Lassalle	Jean-Luc Mélenchon	Marine Le Pen	N. Dupont-Aignan	Nathalie Arthaud	Philippe Poutou	Total
Benoît Hamon	803	113	5	43	6	3	41	47	0	0	11	1072
Emmanuel Macron	103	816	4	55	5	19	39	84	2	0	38	1165
François Asselineau	34	39	587	15	13	16	33	45	2	0	35	819
François Fillon	113	133	1	589	3	1	41	128	2	1	3	1015
Jacques Cheminade	41	29	10	11	793	7	35	31	0	1	52	1010
Jean Lassalle	38	62	4	13	8	449	24	36	3	0	41	678
Jean-Luc Mélenchon	59	37	2	20	3	2	1497	51	1	0	12	1684
Marine Le Pen	62	67	8	57	4	0	24	1270	1	1	2	1496
N. Dupont-Aignan	50	60	5	27	1	2	28	132	115	0	4	424
Nathalie Arthaud	28	28	0	6	4	0	30	44	1	74	34	249
Philippe Poutou	59	34	8	9	11	4	39	28	4	4	719	919
Total	587	1305	629	802	845	500	1790	1849	131	81	940	9459

Picture 6

And in line percentage:



vrai\ prédiction	Benoît Hamon	Emmanuel Macron	François Asselineau	François Fillon	Jacques Cheminade	Jean Lassalle	Jean-Luc Mélenchon	Marine Le Pen	N. Dupont-Aignan	Nathalie Arthaud	Philippe Poutou	*Nb de tweets*
Benoît Hamon	**75%**	11%	0%	4%	1%	0%	4%	4%	0%	0%	1%	*1072*
Emmanuel Macron	9%	**70%**	0%	5%	0%	2%	3%	7%	0%	0%	3%	*1165*
François Asselineau	4%	5%	**72%**	2%	2%	2%	4%	5%	0%	0%	4%	*819*
François Fillon	11%	13%	0%	**58%**	0%	0%	4%	13%	0%	0%	0%	*1015*
Jacques Cheminade	4%	3%	1%	1%	**79%**	1%	3%	3%	0%	0%	5%	*1010*
Jean Lassalle	6%	9%	1%	2%	1%	**66%**	4%	5%	0%	0%	6%	*678*
Jean-Luc Mélenchon	4%	2%	0%	1%	0%	0%	**89%**	3%	0%	0%	1%	*1684*
Marine Le Pen	4%	4%	1%	4%	0%	0%	2%	**85%**	0%	0%	0%	*1496*
N. Dupont-Aignan	12%	14%	1%	6%	0%	0%	7%	31%	**27%**	0%	1%	*424*
Nathalie Arthaud	11%	11%	0%	2%	2%	0%	12%	18%	0%	**30%**	14%	*249*
Philippe Poutou	6%	4%	1%	1%	1%	0%	4%	3%	0%	0%	**78%**	*919*

Picture 7

In green, we have the quantity of tweets well predicted. So, we predicted with a rate greater than 50 % 9 candidates above 11. Nathalie Arthaud and N. Dupont-Aignan's tweets are less well predicted but they are the two candidates who tweet the less.

It is interesting to note that candidates with a similar politic can be predicted in the same class. For Example: N. Dupont-Aignan's tweets are often predicted to be Marine Le Pen’s. So by analyzing a supervised learning model, we can identify candidates with similar programs. But also, those who have less similarities like Jacques Cheminade’s.

Jean-Luc Mélenchon who is the one who tweets the most is more predictable. Maybe we should have split our dataset with 2/3 in train of each candidates. By doing so, we might have had different results with better predictions in small classes.

We were able to learn similarities thanks to the model’s errors, for example between Marine Le Pen and N. Dupont-Aignan.

Conclusion

The stylistic analysis that we carried out is based on relatively simple processing to set up: tokenization, lemmatization, normalization and elimination of unnecessary words. These indicators give us information useful on the way in which candidates use twitter to communicate and convince voters.

Using machine learning techniques, we were able to extract a few insights from a large dataset of tweets associated with the 2017 French presidential campaign. In this case study, each political figure has represented a political current and they have defended it through their “speeches” (neutral, positive, negative) by sending tweets to their followers.

Our analysis not only provides a meaningful and holistic description of data mining techniques but also offers evidence that there was a communication strategy underlying the use of the Twitter platform by all the candidates.

We can conclude that it is necessary to master data mining to be able to know, describe and analyze (linguistically and statistically) the political messages sent on this microblogging platform.

Annex 1 : Python's code

```python
# -*- coding: utf-8 -*-
"""
Created on Mon Jun  8 14:08:20 2020

@author: lboscherie
Pour le dernier exo : https://scikit-
learn.org/stable/tutorial/text_analytics/working_with_text_data.html
"""
import datetime
t = datetime.datetime.now()
from nltk.tokenize import TweetTokenizer
import pandas as pd
from collections import Counter
from nltk.corpus import stopwords
from nltk.stem.snowball import FrenchStemmer
import statistics as st
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
import numpy as np
from sklearn import svm
from  sklearn.neural_network import MLPClassifier
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from collections import Counter
from wordcloud import WordCloud
#import spacy

"""Exo 1 : Import des données.

df = pd.read_excel('tweet2.xlsx')
df = df[~df['Tweet'].astype(str).str.startswith('RT @')]

"""
Exercice 2 : Analyse stylistique des candidats
L'analyse stylistique consiste à étudier le style d'un auteur, à l'aide de
différents indicateurs.
Dans le cadre des tweets, les indicateurs intéressants peuvent être le
nombre de mots différents utilisés,
les mots ou les lemmes les plus utilisés (par exemple, les 10 les plus
fréquents), les catégories grammaticales
les plus utilisées ou les pourcentages d'utilisation de chaque catégorie
grammaticale… La fouille de motifs
séquentielle peut également être utilisée pour calculer les motifs les plus
fréquents de chaque candidat,
au niveau des mots, des lemmes et/ou des catégories grammaticales.
Vous pourrez comparer les valeurs de ces indicateurs entre les candidats
ainsi que par rapport aux valeurs
globalement calculées sur l'ensemble du corpus de tweets. """
#indicateur que l'on peut calculer : nb de mot moyen
#Lemmes les plus utilisés
#Categories grammaticales
#NER
#%age utilisation de chaque catégorie grammaticale
#Motifs le plus fréquents
```

```
################# spaCy ######################
# Initialization of a statistical model in French (nlp)
    ## small model
#nlp_sm = spacy.load(\"fr_core_web_sm\")
    ## or medium model
nlp=spacy.load('fr_core_news_md')
# Conversion to string of the tweet (from dataframe)
df= df.astype({"Tweet": str})

#filter is_retweet
for i in range(0, len(df[['Tweet']])) :
    df= df[~df['Tweet'].astype(str).str.startswith('RT @')]
print (df.head(20),df.columns)

# number of tweet per candidate: :
nb_tweet={}
for i in range(0, len(df[['Auteur']])) :
    auteur = df.iloc[i]['Auteur']
    if auteur not in nb_tweet :
        nb_tweet[auteur] = 0
    nb_tweet[auteur]=nb_tweet[auteur]+1
#print (nb_tweet)

#customization of the stopword list for spaCy:
my_stopwords = {'rt' , 'a' , 'faire', 'faut','si', 'ça',}
for stopword in my_stopwords:
    lexeme = nlp.vocab[stopword]
    # ajout des mots de my_stopwords dans la liste de stop-words de spaCy:
    lexeme.is_stop = True

# Nber of token per tweet
Nb_token = list()
#POS : part of speech
tokenf_pos = []
#lemmas from filtered tokens per tweet
tokenf_lemma = []
#list of filtered tokens per tweet
tokenf_token = []
#list of filtered tokens from corpus
tokens_corpus =[]
## list of NERs
#ent_text = []
## list of NER's label
#ent_label = []
# Number of hashtags per tweet
nb_hashtags = []
#Number of hashtags in Corpus
nb_hashtag_corpus=0
# Number of mentions per tweet
nb_arobases =[]
# Number of mentions  in Corpus
nb_mention_corpus=0
 # Nombre de lien par tweet
 #nb_urls=[]
#part of speech 's label
pos_corpus=[]

#   parse tweets
for i in range(0, len(df[['Tweet']])) :

#   doc is an "nlp Object" used to create documents with linguistic
annotations
    doc = nlp(df.iloc[i]['Tweet'].lower())  #normalization in lowercase
    # urls
    #urls = [u for u in doc if str(u).startswith("https")]
    #nb_url=len(urls)
    # nb_urls.append(nb_url)

    #mentions : @ in tweets and corpus
    arobases =[a for a in doc if str(a).startswith("@")]
    nb_arobase = len(arobases)
    nb_arobases.append(nb_arobase)
    nb_mention_corpus += nb_arobase

    #hashtags : # in tweets and corpus
    hashtags = [h for h in doc if str(h).startswith("#")]
    nb_hashtag=len(hashtags)
    nb_hashtags.append(nb_hashtag)
    nb_hashtag_corpus += nb_hashtag

    # tokens are filtred :
    token_filtred = list(filter(lambda token: token.is_stop == False # stop
words
                                        and token.like_url== False #  URLs
                                        and token.is_punct == False #
punctuation
#                                         (or token._.is_hashtag == True)
                                        and token.is_space == False, doc ))
# and space
    Nb_token.append(len(token_filtred))
    tokens_corpus = tokens_corpus + token_filtred

    # local variables
    Nb_cat_pos= {}
    tokenf_l=[]
    tokenf_t=[]

    for token in token_filtred :
        #verif existence  or init of the category
        if token.pos_ not in Nb_cat_pos :
            Nb_cat_pos[token.pos_]=0
        # increment the category
        Nb_cat_pos[token.pos_]= Nb_cat_pos[token.pos_]+1
        tokenf_l.append(token.lemma_)
        tokenf_t.append(token.text)
    tokenf_token.append(tokenf_t)
    tokenf_lemma.append(tokenf_l)
    tokenf_pos.append(Nb_cat_pos)
    pos_corpus= pos_corpus + tokenf_pos

    # Adding new columns in df:
df = df.assign(
                TweetTokenize = tokenf_token ,
                Nombre_Token = Nb_token,
                TweetLemmatize = tokenf_lemma,
                Etiquette_grammaticale = tokenf_pos,
                Nombre_hashtag= nb_hashtags,
                #Nombre_url= nb_urls,
                Nombre_mention=nb_arobases
                )

     #### corpus ####
#list of lemmas from corpus
lemmes_corpus=[]
for i in range(0, len(df[['Tweet']])) :
    doc = nlp(df.iloc[i]['Tweet'].lower())
    lemmes= [t.lemma_ for t in doc if t.is_stop == False # stop words
                                    and t.like_url== False #  URLs
                                    and t.is_punct == False # Punctuation
                                    and t.is_space == False ]
    lemmes_corpus += lemmes

CounterLemmeCorpus= Counter(lemmes_corpus)
Most_Common_Lemme_Corpus= CounterLemmeCorpus.most_common(10)
dict_Most_Common_Lemme_Corpus = dict (Most_Common_Lemme_Corpus)
print(Most_Common_Lemme_Corpus)

#Number Averages : mentions, Hashtags, tokens
nb_token_corpus=len(tokens_corpus)
nb_tweet_corpus= len(df[['Tweet']])
nb_token_moy_corpus=nb_token_corpus/nb_tweet_corpus
print ("nb_token_moy_corpus :", nb_token_moy_corpus)
nb_mention_moy_corpus = nb_mention_corpus/nb_tweet_corpus
print ("nb_mention_moy_corpus :",nb_mention_moy_corpus)
nb_hashtag_moy_corpus= nb_hashtag_corpus/nb_tweet_corpus
print ("nb_hashtag_moy_corpus :" ,nb_hashtag_moy_corpus)

#POS from corpus
dfpos_corpus=pd.DataFrame(pos_corpus)
#print (dfpos_corpus)


     #### Most common word and lemmas per candidate #####
list of candidate
ListeCandidat = [
                 "Nathalie Arthaud" ,
                 "Jacques Cheminade" ,
                "Jean Lassalle",
                 "Jean-Luc Mélenchon" ,
                 "François Asselineau" ,
                  "Benoît Hamon" ,
                 'Marine Le Pen' ,
                 'Emmanuel Macron',
                 'Philippe Poutou',
                 'N. Dupont-Aignan',
                 'François Fillon'
                 ]

#list most common word lemma and POS
Most_Common_Words = list()
Most_Common_Lemme = list()
Most_Common_Pos=list()

# Mean list of numbers of hashtags, mentions and words
Nb_mot_Moyen = list()
Nb_hashtag_Moyen =list()
# Nb_url_Moyen=list()
Nb_arobase_Moyen=list()

#parse Listecandidat
for candidat in ListeCandidat :
    # dfCandidat, local dataframe
    dfCandidat = df[df['Auteur'] == candidat]

    #Most commen token and lemmas
    Tokens = dfCandidat['TweetTokenize'].tolist()
    Lemmes = dfCandidat['TweetLemmatize'].tolist()
    Nb_Tokens = dfCandidat['Nombre_Token'].tolist()
    tokenf_pos = dfCandidat['Etiquette_grammaticale'].tolist() #liste de
dict
```

```
    # hashtags
    nb_hashtags = dfCandidat['Nombre_hashtag'].tolist()
    # urls
    #nb_urls = dfCandidat['Nombre_url'].tolist()
    # mentions
    nb_arobases= dfCandidat['Nombre_mention'].tolist()

    # POS
    counterPos=Counter()
    for pos in tokenf_pos :
        counterPos  += Counter(pos)
    Most_Common_Pos = Most_Common_Pos + [counterPos.most_common(10)]

    #tokens
    counterToken = Counter()
    for token in Tokens :
         counterToken += Counter(token)
    Most_Common_Words = Most_Common_Words + [counterToken.most_common(10)]
    #lemmas
    counterLemme = Counter()
    for lemme in Lemmes :
        counterLemme += Counter(lemme)
    Most_Common_Lemme = Most_Common_Lemme + [counterLemme.most_common(10)]
    # Means
    Nb_mot_Moyen += [st.mean(Nb_Tokens)]
    Nb_hashtag_Moyen += [st.mean(nb_hashtags)]
#   Nb_url_Moyen += [st.mean(nb_urls)]
    Nb_arobase_Moyen += [st.mean(nb_arobases)]

data = {
    'Candidat':  ListeCandidat,
    'Mots les plus communs': Most_Common_Words,
    'Lemmes les plus communs' : Most_Common_Lemme,
    'Nb de tokens moyen' : Nb_mot_Moyen,
    'categories gramaticales les plus communes' : Most_Common_Pos,
    'Nb de hashtags moyen' : Nb_hashtag_Moyen,
    'Nb de mentions moyen':Nb_arobase_Moyen
        }

data_Candidat = pd.DataFrame (data, columns = [
                                                'Candidat',
                                                'Mots les plus communs',
                                                'Lemmes les plus communs',
                                                'Nb de tokens moyen',
                                                'categories gramaticales
les plus communes',
                                                'Nb de hashtags moyen',
                                                'Nb de mentions moyen'
                                                                     ])

     #####Generating a square wordcloud #####

def generate_wordcloud(freq, key):
# Generate a word cloud image
    wordcloud = WordCloud().generate_from_frequencies(freq)
#"import matplotlib.pyplot as plt
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis("off")
    plt.title('Wordcloud of "{}"'.format(key))
    plt.savefig('wordcloud/{}.png'.format(key))


for i in range(len(data_Candidat)):
    candidat = data_Candidat.iloc[i]['Candidat']
    lemmes = data_Candidat.iloc[i]['Lemmes les plus communs']
    diLemmes = dict(lemmes)
    print(diLemmes)
    generate_wordcloud(diLemmes, candidat)

generate_wordcloud(dict_Most_Common_Lemme_Corpus,"Corpus")

            ################# NLTK #######################

#Creation d'un objet tokenizer
tokenizer = TweetTokenizer()
stemmer = FrenchStemmer()
df = df.astype({"Tweet": str}) #Conversion en string de la colonne tweet
#Creation d'une liste permettant de stocker la tokenization
# Les parties du code en commentaires correspondant à l'utilisation de
spacy pour le reconnaissance des entités nommées
L= list()
Nb_token = list()
Lemmatize = list()
#REN = list()
#nlp = spacy.load('fr')
#Ajout de stopwords
"""On pourrait peut-être ajouté aussi dans les stopwords les noms des
candidats """
_stopwords = stopwords.words('french') + ['rt' , 'a' , 'faire',
'faut','si', 'ça']
for i in range(0, len(df[['Tweet']])) : #On tokenize tous les tweets
    # On garde les tokens qui sont alphanumériques
    tokens = [w for w in tokenizer.tokenize((df.iloc[i]['Tweet'].lower()))
        if w.isalpha()]
    #On enlève les stopwords
    no_stops = [t for t in tokens
                if t not in _stopwords]

    lemme = [stemmer.stem(a) for a in no_stops ]
    L = L + [no_stops]
    Lemmatize = Lemmatize + [lemme]
    Nb_token = Nb_token + [len(no_stops)]
    #nlp.entity
    #doc = nlp(df.iloc[i]['Tweet'].lower())
    #REN += [doc]
    print(i)

#Ajout des tokens dans une nouvelle colonne du dataset
df = df.assign(TweetTokenize = L, Nombre_Token = Nb_token,TweetLemmatize =
Lemmatize)

#Mots les plus fréquents par candidat, Lemmes les plus fréquents
ListeCandidat = ["Nathalie Arthaud" , 'Jacques Cheminade' , 'Jean
Lassalle','Jean-Luc Mélenchon'
                 ,'François Asselineau' , 'Benoît Hamon' , 'Marine Le Pen'
, 'Emmanuel Macron','Philippe Poutou'
                 ,'N. Dupont-Aignan','François Fillon']

Most_Common_Words = list()
Most_Common_Lemme = list()
Nb_mot_Moyen = list()
for i in ListeCandidat :

    dfCandidat = df[df['Auteur'] == i]
    #Most commen token
    Tokens = dfCandidat['TweetTokenize'].tolist()
    Lemmes = dfCandidat['TweetLemmatize'].tolist()
    Nb_Tokens = dfCandidat['Nombre_Token'].tolist()
    counter = Counter()
    for j in Tokens :
        counter += Counter(j)
    counterLemme = Counter()
    for j in Lemmes :
        counterLemme += Counter(j)
    Most_Common_Words = Most_Common_Words + [counter.most_common(10)]
    Most_Common_Lemme = Most_Common_Lemme + [counterLemme.most_common(10)]
    Nb_mot_Moyen += [st.mean(Nb_Tokens)]

data = {'Candidat':  ListeCandidat,
    'Mots les plus communs': Most_Common_Words,
    'Lemmes les plus communs' : Most_Common_Lemme,
    'Nb de tokens moyen' : Nb_mot_Moyen
    }

data_Candidat = pd.DataFrame (data, columns = ['Candidat','Mots les plus
communs',
                                                'Lemmes les plus
communs','Nb de tokens moyen'])


###EXO 3 : Clustering

# https://towardsdatascience.com/k-means-clustering-8e1e64c1561c
""" On utilise les lemmes créés à l'exo précédent afin d'avoir des tweets
déjà nettoyés.
On peut peut-être aussi essayer d'utiliser les tokens.
Edit : Le plus intéressant semble être avec les lemmes
Le code est fait sur l'ensemble du corpus. Ensuite, on pourra faire la même
chose, par candidat, pour connaître les thématiques abordées par chaque
candidat
"""
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from sklearn.preprocessing import normalize
#from sklearn.metrics import adjusted_rand_score

#On utilise la variable corpus, qu'on va clusterisé

corpus = df['Tweet'].tolist() # -> Tweet originels
#Utilisation des lemmes créés
corpus_Lemme = list()
for i in Lemmes :
    TweetLemme = ''
    for j in i :
        TweetLemme += ' ' + j
    corpus_Lemme += [TweetLemme]
corpus_Tokens = list()
for i in Tokens :
    TweetToken = ''
    for j in i :
        TweetToken += ' ' + j
    corpus_Tokens += [TweetToken]
""" choisir quoi utiliser pour clusteriser avec la variable document :
    corpus : twwets brut ; corpus_Lemme : tweet lemmatisé
    corpus_Tokens : Tweet tokenisé     """
documents = corpus_Lemme
    #First step is to do the tfidf transfromation
vectorizer = TfidfVectorizer(stop_words=_stopwords)
X = vectorizer.fit_transform(documents)
tf_idf_norm = normalize(X)
tf_idf_array = tf_idf_norm.toarray()
```

```
number_clusters = range(1, 7)

#Do an ACP  to reduce the dimensionality of our feature matrix so we can
plot it in two dimensions
#ACP sur deux composantes
sklearn_pca = PCA(n_components = 2)
    #Appliqué au tfidf
Y_sklearn = sklearn_pca.fit_transform(tf_idf_array)

# nombre de classe optimal

kmeans = [KMeans(n_clusters=i, max_iter = 600) for i in number_clusters]
kmeans
score = [kmeans[i].fit(Y_sklearn).score(Y_sklearn) for i in
range(len(kmeans))]
score
plt.plot(number_clusters, score)
plt.xlabel('Number of Clusters')
plt.ylabel('Score')
plt.title('Elbow Method')
plt.show()
# 3classes
#Puis on fait un kmeans
true_k = 2
model = KMeans(n_clusters=true_k, init='k-means++', max_iter=100, n_init=1)
model.fit(tf_idf_norm)

# Affichage des termes les plus fréquent par clusters
print("Top terms per cluster:")
order_centroids = model.cluster_centers_.argsort()[:, ::-1]
terms = vectorizer.get_feature_names()
for i in range(true_k):
    print("Cluster %d:" % i),
    for ind in order_centroids[i, :10]:
        print(' %s' % terms[ind]),
    print

print("\n")
print("Prediction")

# Application d'une prédiction à un tweet
Y = vectorizer.transform(["A Tweet"])
prediction = model.predict(Y)
print(prediction)

# Clustering par candidat
############################################################################

ListeCandidat = ["Nathalie Arthaud" , 'Jacques Cheminade' , 'Jean
Lassalle','Jean-Luc Mélenchon'
                 ,'François Asselineau' , 'Benoît Hamon' , 'Marine Le Pen'
, 'Emmanuel Macron','Philippe Poutou'
                 ,'N. Dupont-Aignan','François Fillon']
dfCandidat = df[df['Auteur'] == "François Fillon"]
corpus = dfCandidat['Tweet'].tolist() # -> Tweet originels
#Utilisation des lemmes créés
corpus_Lemme = list()
for i in dfCandidat['TweetLemmatize'] :
    TweetLemme = ''
    for j in i :
        TweetLemme += ' ' + j
    corpus_Lemme += [TweetLemme]
corpus_Tokens = list()
```

```
for i in Tokens :
    TweetToken = ''
    for j in i :
        TweetToken += ' ' + j
    corpus_Tokens += [TweetToken]
""" choisir quoi utiliser pour clusteriser avec la variable document :
    corpus : twwets brut ; corpus_Lemme : tweet lemmatisé
    corpus_Tokens : Tweet tokenisé     """
documents = corpus_Lemme
    #First step is to do the tfidf transfromation
vectorizer = TfidfVectorizer(stop_words=_stopwords)
X = vectorizer.fit_transform(documents)
tf_idf_norm = normalize(X)
tf_idf_array = tf_idf_norm.toarray()

number_clusters = range(1, 7)

#Do an ACP  to reduce the dimensionality of our feature matrix so we can
plot it in two dimensions
#ACP sur deux composantes
sklearn_pca = PCA(n_components = 2)
    #Appliqué au tfidf
Y_sklearn = sklearn_pca.fit_transform(tf_idf_array)

# nombre de classe optimal

kmeans = [KMeans(n_clusters=i, max_iter = 600) for i in number_clusters]
kmeans
score = [kmeans[i].fit(Y_sklearn).score(Y_sklearn) for i in
range(len(kmeans))]
score
plt.plot(number_clusters, score)
plt.xlabel('Number of Clusters')
plt.ylabel('Score')
plt.title('Elbow Method')
plt.show()
# 3classes
#Puis on fait un kmeans
true_k = 3
model = KMeans(n_clusters=true_k, init='k-means++', max_iter=100, n_init=1)
model.fit(tf_idf_norm)

# Affichage des termes les plus fréquent par clusters
print("Top terms per cluster:")
order_centroids = model.cluster_centers_.argsort()[:, ::-1]
terms = vectorizer.get_feature_names()
for i in range(true_k):
    print("Cluster %d:" % i),
    for ind in order_centroids[i, :10]:
        print(' %s' % terms[ind]),
    print

print("\n")
print("Prediction")

# Application d'une prédiction à un tweet
Y = vectorizer.transform(["A Tweet"])
prediction = model.predict(Y)
print(prediction)
```

```
"""EXO 4 : Vous utiliserez des méthodes supervisées pour construire un
classifieur permettant
d'attribuer un nouveau tweet au candidat qui est le plus susceptible de
l'avoir écrit.
Pour cela, vous découperez le corpus global des tweets par candidat en un
corpus de
développement, un corpus de validation et un corpus de test (vous pourrez
également utiliser de la
validation croisée). Vous appliquerez des méthodes de classification
supervisée et vous donnerez les
résultats obtenus sur le corpus de test. Vous pourrez réutiliser les
indicateurs identifiés à l'exercice 2,
pur choisir les caractéristiques les plus pertinentes à utiliser dans vos
classifieurs."""

########## k plus proches voisins  : Avec seulement les variables quanti
####################################################
"""Nettoyage df"""

df['Retweet'] = df['Retweet'].replace('#', 0)
X = df[[#'Tweet','Outil',
        'Retweet',#'TweetTokenize',
        'Nombre_Token'#,'TweetLemmatize'"""
        ]]
y = df['Auteur']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
random_state=42)




#x = df.loc[:,"Outil"]
y = X_train.loc[:,"Retweet"]
#z = df.loc[:,"TweetTokenize"]
az = X_train.loc[:,"Nombre_Token"]
#bz = df.loc[:,"TweetLemmatize"]


k = 100
model = KNeighborsClassifier(n_neighbors=k)
model.fit(X_train,y_train)
#prediction= model.predict(["données à predire"])
#Accuracy of the model
model.score(X_test,y_test)
y_scores = model.predict_proba(X_test)
#fpr, tpr, threshold = roc_curve(y_test, y_scores[:, 1])


### Préparation pour les prochains modéles
X = df['Tweet'#,'Outil',
        #'Retweet','TweetTokenize',
        #'Nombre_Token','TweetLemmatize'"""
        ]
y = df['Auteur']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
random_state=42)


## K neighbors classifier with tweet
k=5
```

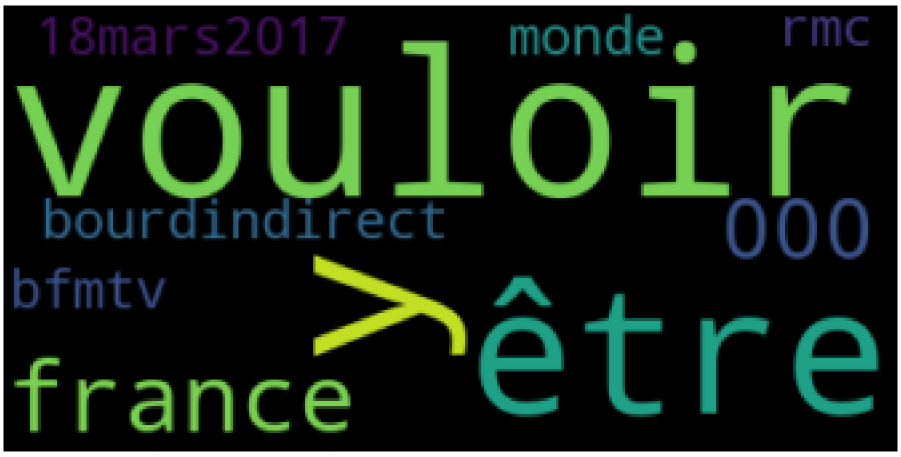
```
k_nn = Pipeline([
     ('vect', CountVectorizer()),
     ('tfidf', TfidfTransformer()),
     ('clf', KNeighborsClassifier(n_neighbors=k)),
 ])
#model = KNeighborsClassifier(n_neighbors=k)
k_nn.fit(X_train,y_train)
#prediction= model.predict(["données à predire"])
#Accuracy of the model
# Accuracy
predicted = k_nn.predict(X_test)
np.mean(predicted == y_test)

y_predict = k_nn.predict(X_test)
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_predict)


#### Multinomial NB
##########################################################

 # https://scikit-
learn.org/stable/tutorial/text_analytics/working_with_text_data.html

count_vect = CountVectorizer()


vectorizer = TfidfVectorizer(stop_words=_stopwords)
X = vectorizer.fit_transform(X_train)


X_train_counts = count_vect.fit_transform(X_train)
X_train_counts.shape
X_train_counts = X_train_counts.toarray()
count_vect.vocabulary_.get(u'algorithm')

tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
X_train_tfidf.shape



clf = MultinomialNB().fit(X_train_tfidf, y_train)
text_clf = Pipeline([
     ('vect', CountVectorizer()),
     ('tfidf', TfidfTransformer()),
     ('clf', MultinomialNB()),
 ])
text_clf.fit(X_train, y_train)
# Accuracy
predicted = text_clf.predict(X_test)
np.mean(predicted == y_test)



#Multinomial NB with CountVectorizer
#######################################

count = CountVectorizer()
bag_of_words = count.fit_transform(df['Tweet'#,'Outil',
        #'Retweet','TweetTokenize',
        #'Nombre_Token','TweetLemmatize'"""
        ])
```

```
X = bag_of_words.toarray()
y = df['Auteur']
#y = df['Auteur'].to_list()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.50,
random_state=42)


clf = MultinomialNB()
model = clf.fit(X_train, y_train)
#Accuracy
model.score(X_test,y_test)
y_scores = model.predict_proba(X_test)

###### SVM
###################################################################


count_vect = CountVectorizer()
X = df['Tweet'#,'Outil',
        #'Retweet','TweetTokenize',
        #'Nombre_Token','TweetLemmatize'"""
        ]
y = df['Auteur']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
random_state=42)

vectorizer = TfidfVectorizer(stop_words=_stopwords)
X = vectorizer.fit_transform(X_train)


X_train_counts = count_vect.fit_transform(X_train)
X_train_counts.shape
X_train_counts = X_train_counts.toarray()
count_vect.vocabulary_.get(u'algorithm')

tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
X_train_tfidf.shape
clf = svm.SVC()
clf.fit(X_train_tfidf, y_train)

text_clf2 = Pipeline([
     ('vect', CountVectorizer()),
     ('tfidf', TfidfTransformer()),
     ('clf', svm.SVC()),
 ])
text_clf2.fit(X_train, y_train)
# Accuracy
predicted = text_clf2.predict(X_test)
np.mean(predicted == y_test)

y_predict = text_clf2.predict(X_test)
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_predict)


#MSVM with CountVectorizer #######################################

count = CountVectorizer()
bag_of_words = count.fit_transform(df['Tweet'#,'Outil',
        #'Retweet','TweetTokenize',
        #'Nombre_Token','TweetLemmatize'"""
        ])
```

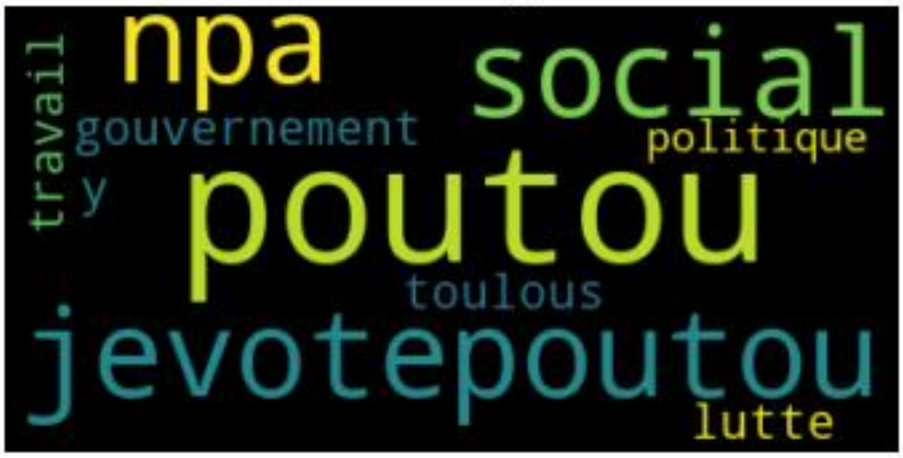
```
X = bag_of_words.toarray()
y = df['Auteur']
#y = df['Auteur'].to_list()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.50,
random_state=42)

clf = svm.SVC()
model = clf.fit(X_train, y_train)
#Accuracy
model.score(X_test,y_test)
y_scores = model.predict_proba(X_test)




###### Neural Network #####################################
clf_nn = Pipeline([
     ('vect', CountVectorizer()),
     ('tfidf', TfidfTransformer()),
     ('clf', MLPClassifier(solver='adam', alpha=1e-
5,hidden_layer_sizes=(10, 2), random_state=1)),
 ])
clf_nn.fit(X_train, y_train)
# Accuracy
predicted = clf_nn.predict(X_test)
np.mean(predicted == y_test)


#Temps d'excution du script
print (datetime.datetime.now()-t)
```

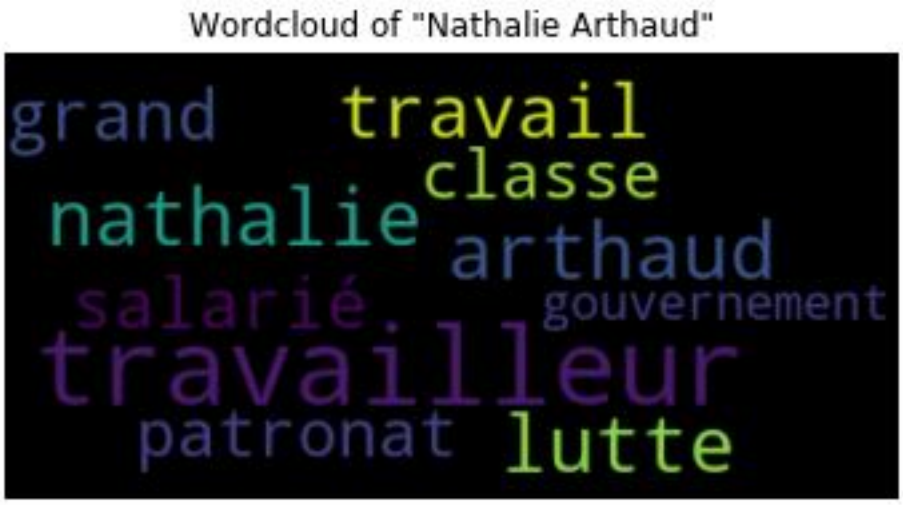
Annex 2: the 10 most commons lemmas - word cloud

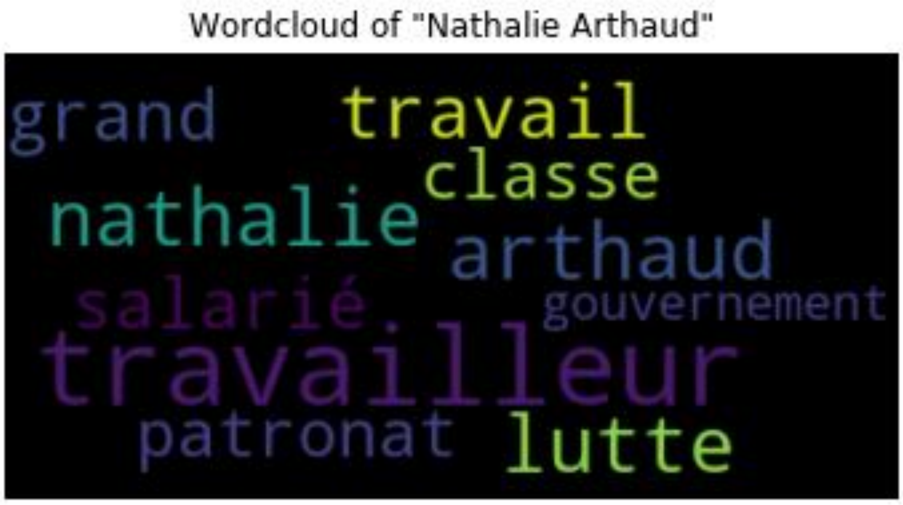
Wordcloud of "Corpus"



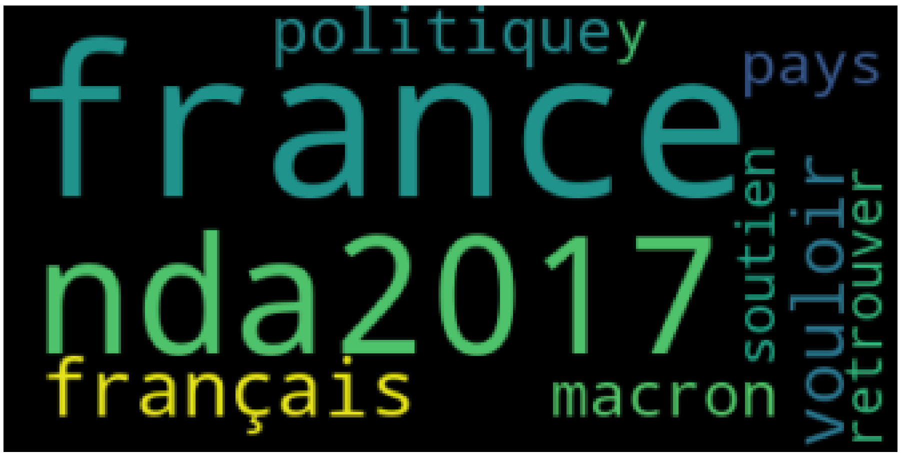
Wordcloud of "Philippe Poutou"



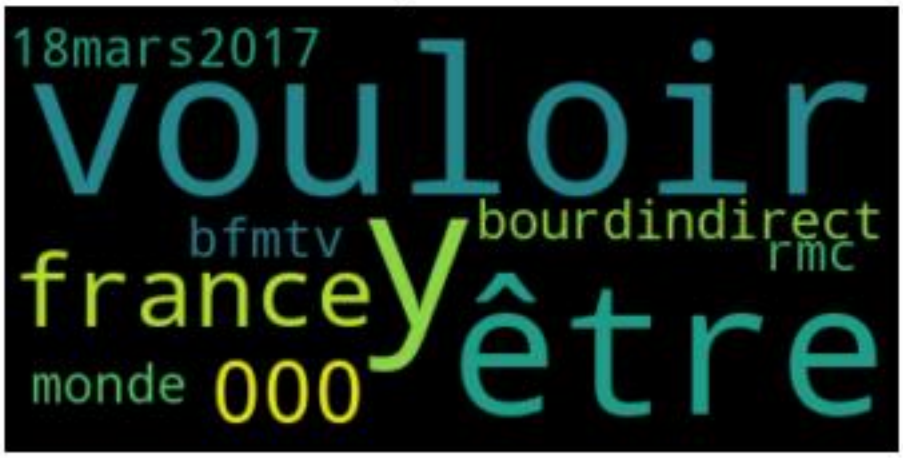

Wordcloud of "Nathalie Arthaud"
grand
travail
classe
nathalie
arthaud
salarié
gouvernement
travailleur
patronat
lutte


Wordcloud of "Nathalie Arthaud"
grand
travail
classe
nathalie
arthaud
salarié
gouvernement
travailleur
patronat
lutte

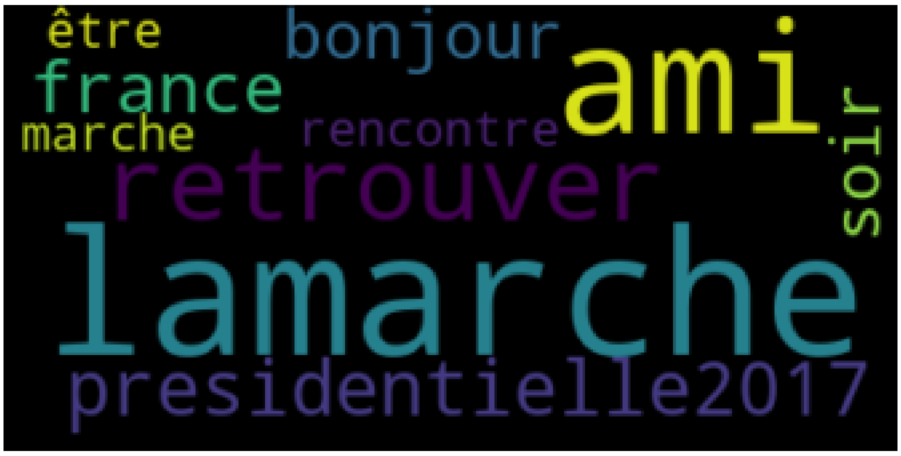
Wordcloud of "N. Dupont-Aignan"



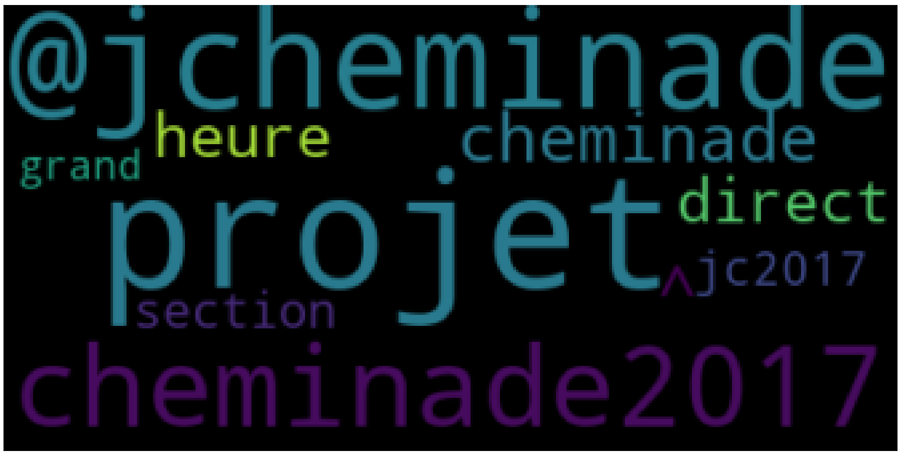
Wordcloud of "Jean-Luc Mélenchon"



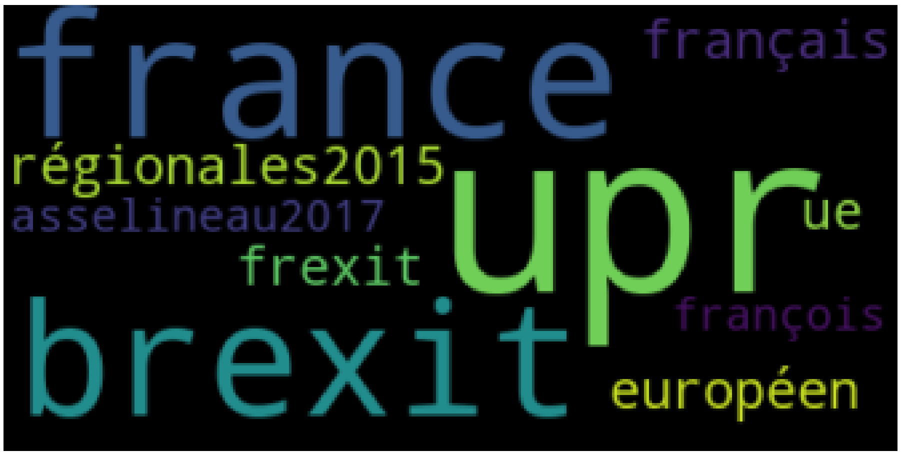
Wordcloud of "Jean Lassalle"



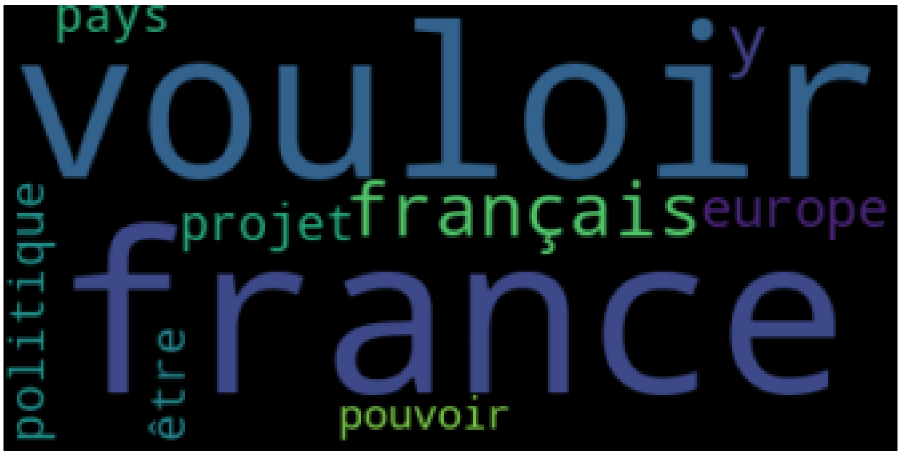
Wordcloud of "Jacques Cheminade"



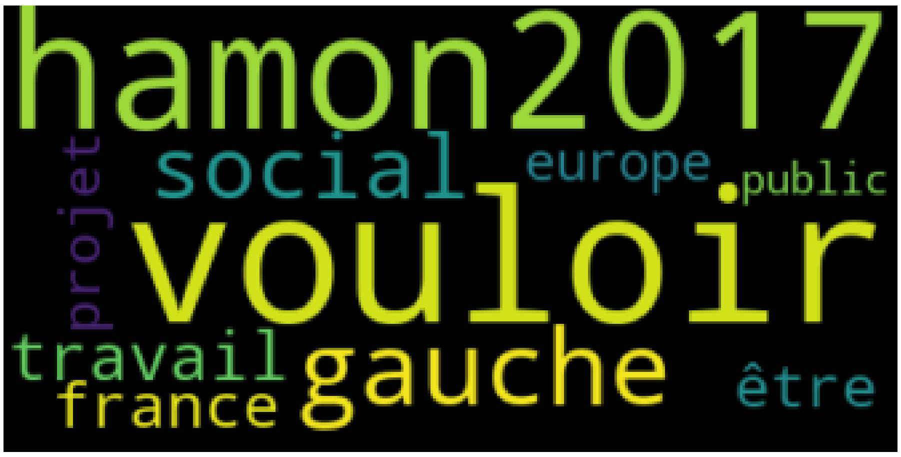
Wordcloud of "François Asselineau"



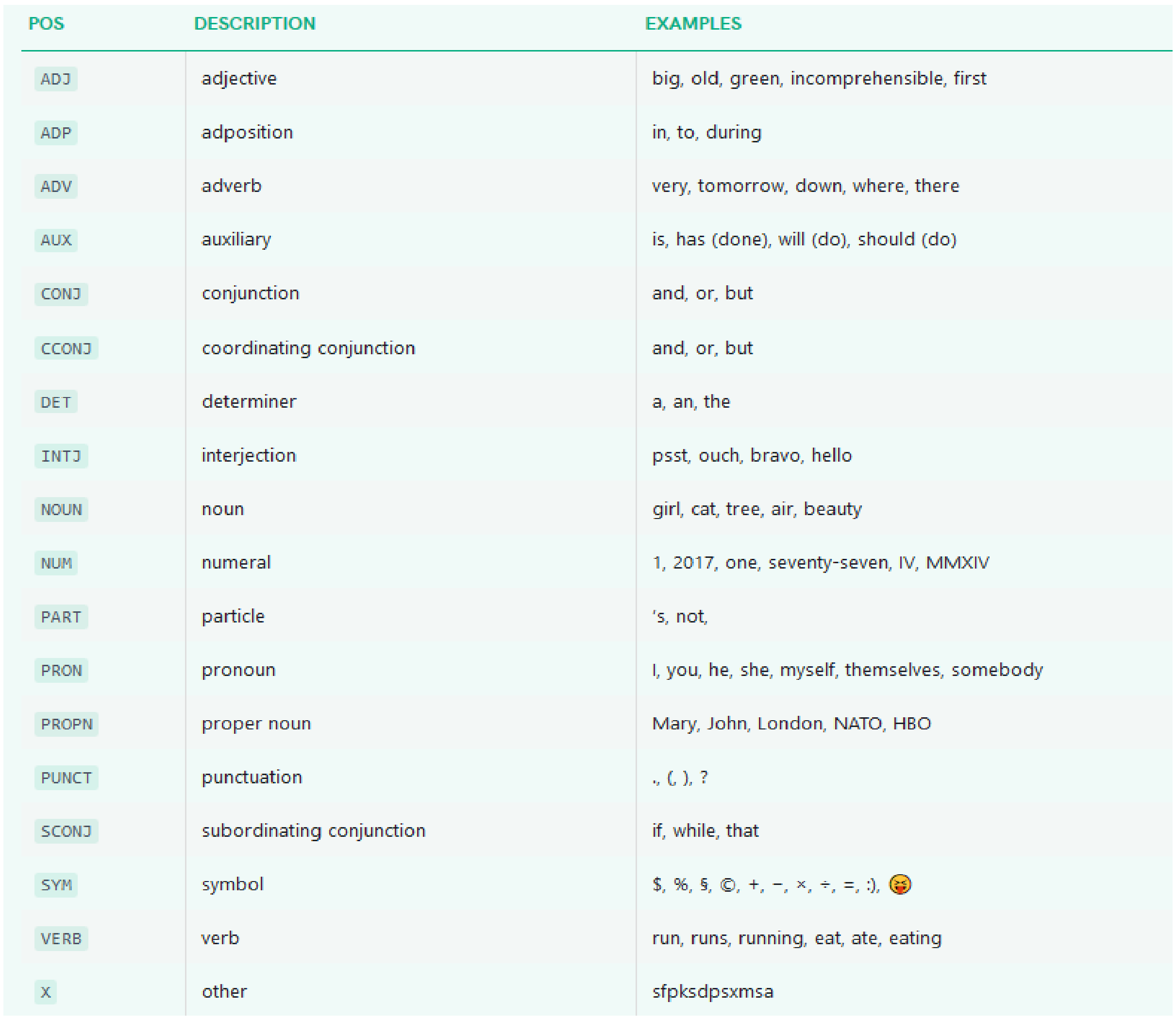
Wordcloud of "Emmanuel Macron"



Wordcloud of "Benoît Hamon"



Annexe 3 : spaCy's POS label



POS	DESCRIPTION	EXAMPLES
ADJ	adjective	big, old, green, incomprehensible, first
ADP	adposition	in, to, during
ADV	adverb	very, tomorrow, down, where, there
AUX	auxiliary	is, has (done), will (do), should (do)
CONJ	conjunction	and, or, but
CCONJ	coordinating conjunction	and, or, but
DET	determiner	a, an, the
INTJ	interjection	psst, ouch, bravo, hello
NOUN	noun	girl, cat, tree, air, beauty
NUM	numeral	1, 2017, one, seventy-seven, IV, MMXIV
PART	particle	's, not,
PRON	pronoun	I, you, he, she, myself, themselves, somebody
PROPN	proper noun	Mary, John, London, NATO, HBO
PUNCT	punctuation	., (,), ?
SCONJ	subordinating conjunction	if, while, that
SYM	symbol	$, %, §, ©, +, −, ×, ÷, =, :), 😝
VERB	verb	run, runs, running, eat, ate, eating
X	other	sfpksdpsxmsa