# Information Retrieval and Web Analysis on United States 2020 elections tweets

Ramon Vallés Puig
vallespuigramon@gmail.com
UPF (Information Retrieval & Web Analytics) 2020-2021

## Introduction

The 2020 U.S. presidential election was the 59th quadrennial presidential election, held on Tuesday, November 3, 2020. The Democratic ticket of former Vice President Joe Biden and California junior U.S. Senator Kamala Harris defeated Republican incumbent President Donald Trump and incumbent Vice President Mike Pence. The election saw the highest voter turnout by percentage since 1900, with each of the two major candidates receiving more than 74 million votes, surpassing Barack Obama's record of 69.5 million votes in 2008.

This has possibly been the most followed election of the last decades. Words such as "Biden", "Trump", or simply "US elections" were trending topics numerous times in social networks, including Twitter, the favorite tool of the then president Donald Trump.

In the following project, a compilation of the tweets circulating at that time has been compiled and analyzed using an information retrieval model and further analysis of the collected web data.

## Acknowledgments

I would like to thank Eduard Masip and David Gayete for their academic contribution to this project, and the professors of the Pompeu Fabra University in Barcelona, Ricardo Baeza, Ludovico Boratto and Mateo Manca for arming us with the necessary knowledge to carry out this project.

# Content

# 1. Dataset

To make possible the analysis of this topic, the following list of english keywords have been chosen with the aim of scraping related tweets. The tweets have been collected from the dates October 20-25. It has been believed those were the most used at the moment and represent from a partial point of view both, Republicans and Democrats.

- "Trump", "#Trump"
- "Biden", "#Biden"
- "Kamala Harris"
- "Mike Pence"
- "#UsElections2020"
- "GOP", "#GOP"
- "#GOPchairwoman"

## 1.1 About the dataset

The total number of tweets were automatically picked by the strapping algorithm and stored in a JSON file. In order to avoid the dataset to be corrupted with a single topic either from a TV show or a breaking news, the collection was carried out in different time intervals during 5 days, accumulating a total period of an hour of execution and 100k tweets of which 38,5K were unique.

- **Scrapping Time:** 1h 10min 48s ( 4248 seconds )
- **Total number of Tweets:** 100.000
- **Unique Tweets:** 38.592
- **Number of users:** 27.223

*The dataset file can be found here.*

## 1.2 Preprocessing

As a first step, we need to preprocess the documents to make them computable. This can be done by applying some operations such as:

- Removing stop-words.
- Removing punctuation.
- Stemming.
- and more …

Once we have “cleaned” the tweets we must consider what information is relevant to us, and how we want to keep it. For the purpose of this research, a dictionary has been used to collect the information from all the scraped tweets (in the JSON file). In particular we save, for each tweet, the most relevant details:

    - TweetID.
    - Original text.
    - Tokenized text.
    - Username.
    - Date of publication.
    - List of Hashtags.
    - Number of Likes.
    - Number of Retweets.
    - URLS in the tweet.
    - Type of tweet (Original/Retweet).

See that the tokens of each tweet are used to further analyze and rank the tweets. To obtain the tokens we have created a function “cleanTweet” (Fig. 1) , that receives the text as the input, and returns a list of words that are preprocessed. In order to do that, first it converts the text to lowercase and delete some characters (Notice that hashtags ‘**#**’ characters are NOT removed as it will be useful information to keep trace of the topic, neither are ‘**@**’ characters to detect the mentions which will be anonymized by a hash function). The implemented function also removes the stopwords and applies stemming to get the root of each word.

```
def cleanTweet(tweetText):
    stemming = PorterStemmer()
    stops = set(stopwords.words("english"))

    cleanText = tweetText.lower()   # Transform to lowercase
    cleanText = re.sub('[:\[\]&%$\"\'!./,;:?=¿^\-_*+)<>(¡]','',cleanText)
    cleanText = cleanText.split()   # Split the text into terms
    cleanText = [word for word in cleanText if word not in stops] # Drop
stopwords
    cleanText = [stemming.stem(word) for word in cleanText] # Perform stemming

    # Anonymize mentions
    for i in range(len(cleanText)):
        if cleanText[i][0] == '@':
            cleanText[i] = str(hash(cleanText[i]))

    return cleanText
```

**Figure 1:** Implementation of "**cleanTweet**" function in python

## 1.3 Retweets:

To avoid the problem of retweets, since both the original tweet and the retweet would contain exactly the same terms and therefore would get the same tf-idf score when ranking the query results, it has been thought a simple filter that iterates over the dictionary where all the tweets are stored, and drops those retweets whose original tweet is contained in our dataset. These tweets can be removed from the dataset since the "retweet counter" can be recovered from the original

## 1.4 Word Cloud:

The first analysis of the dataset has been done by counting the most repeated terms and displaying them on the screen to visualize it as a Word Cloud (fig. 2). The final result is shown below (fig. 3). As we can see, there are two dominant terms (Biden & Trum) which are represented as bigger words to emphasize its importance. Notice that there are some words that are actually numbers, this is due to the anonymization of the mentions that were done in the cleaning phase, and make us think there are some important people that are involved in the topic ( in the case of *'-8603529214224664465'*, probably this hash corresponds to the user Donald Trump or Joe Biden, but it's just an hypothesis ).

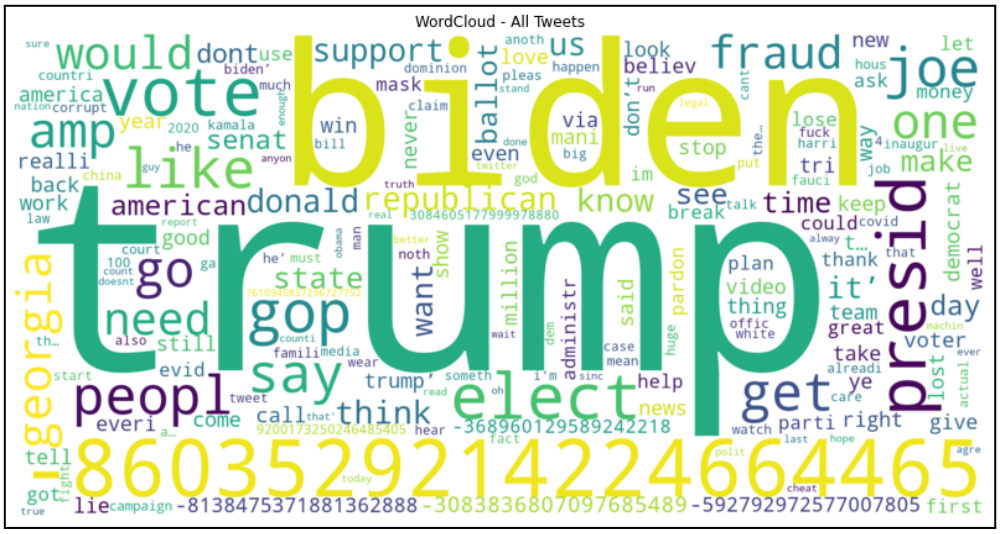
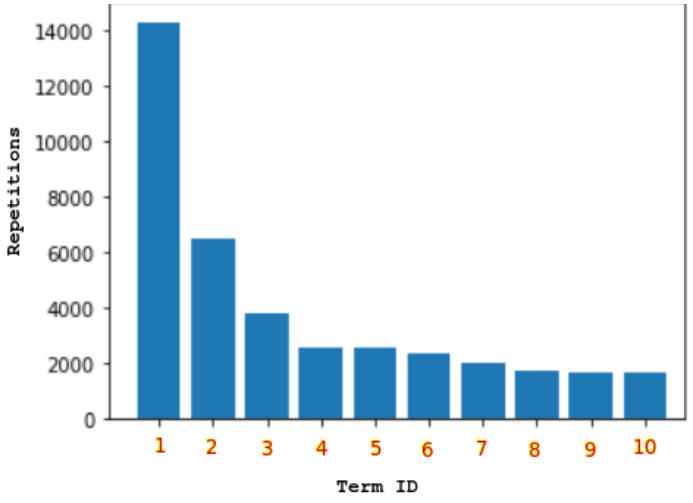



**Figure 2:** Word Cloud representation of the dataset terms.




1. 'trump'
2. 'biden'
3. '-8603529214224664465'
4. 'vote'
5. 'presid'
6. 'elect'
7. 'gop'
8. 'like'
9. 'joe'
10. 'say'

**Figure 3:** Column chart representation of the top 10 terms against the number of repetitions in the dataset.

# 2. Search Engine

We want to create a ***Search Engine*** that, given as input to a query, returns the most relevant tweets that contain all the words in the query and sorts them by their relevance with respect to the query!

## 2.1 Inverted index and ranking score

Now that we have processed the dataset, it is time to design our search engine. For that we are going to create the inverted index of the terms retrieved from the tweets, we may also use a vocabulary data structure to make it clearer.

```
{
    Term_id_1: [document_1, document_2, document_4],
    Term_id_2: [document_1, document_3, document_5, document_6],
    etc...
}
```

## 2.2 Documents information

Since we are dealing with conjunctive queries (AND), each of the returned documents should contain all the words in the query.

The final output of a query will return (when present) the following information for each of the selected documents:

**Tweet | Username | Date | Hashtags | Likes | Retweets | Url**

(here the “Url” is meant as the tweet link)

## 2.3 Ranking Score

Given a query, we want to get the top-20 documents related to the query. In this project we provide 2 different ways of ranking

1. **TF-IDF** + cosine similarity (Classical scoring).
2. **Our personalized Score** + cosine similarity.
   Here it has been created a new score, which has been subject to an analysis (Pros & Cons). This personalized score involves ***the tweets information regarding the popularity over the social network*** (number of likes, number of tweets, number of comments, etc…).
3. **Word2Vec + cosine similarity.**

### 2.3.1 TF-IDF + Cosine Similarity

TF-IDF (short for **term frequency–inverse document frequency)**, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in searches of information retrieval, text mining, and user modeling. The tf–idf value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general.

**Term frequency:**

Number of times a term occurs within a document. (more occurrences -> more relevant, but not proportionally):

$$\mathrm{tf}(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

**Document frequency df:**

Document frequency tells us how much a given word is repeated, not in a tweet but in the whole dataset.

**IDF weight:** (inverse document frequency)

As we are interested in rare terms, which are usually more informative than frequent terms. We will fige a high weight on those terms that match the query but are rare to find in the dataset (thus the inverse of the document frequency).

- We use the logarithm to dampen the effect of idf → .
- IDF ranking is useful in sentence analysis. (Has no effect on a single word)
- e.g: {capricious person} capricious has higher weight than person.

**Cosine similarity:**

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space. It is defined to equal the cosine of the angle between them, which is also the same as the inner product of the same vectors normalized to both have length 1.

$$\text{sim}(\overline{X}, \overline{Y}) = \cos\theta$$

$$\text{sim}(\overline{X}, \overline{Y}) = \frac{\sum_{i=1}^{d} x_i \cdot y_i}{\sqrt{\sum_{i=1}^{d} x_i^2} \cdot \sqrt{\sum_{i=1}^{d} y_i^2}}$$

### 2.3.1 Our Personalized Score + cosine similarity

Our custom score, besides taking into account the frequency of terms in both, the tweet itself and the tweet list, has the added value of the popularity of the tweet. That is, to calculate how relevant the tweet is, a weighting is done between the tf-idf, the likes and the number of retweets it has. Therefore we implemented a function, which will map the number of likes to values between [0,1]. So, we designed an exponential function, that as the number of likes increases, the increase of its scoring is exponentially reduced. the function is the following:

$$f(likes) = 1 - exp(-\frac{likes}{5E4})$$

Similar to the likes score, we also apply an exponential score to the number of retweets.

Note that, in the denominator of both functions, a constant value is established, independent of each function. This value was chosen empirically according to the criterion that a tweet usually has more likes than retweets, and that after a certain number, the score should converge and be considered as a popular tweet.

$$f(retweets) = 1 - exp(-\frac{retweets}{2.5E4})$$

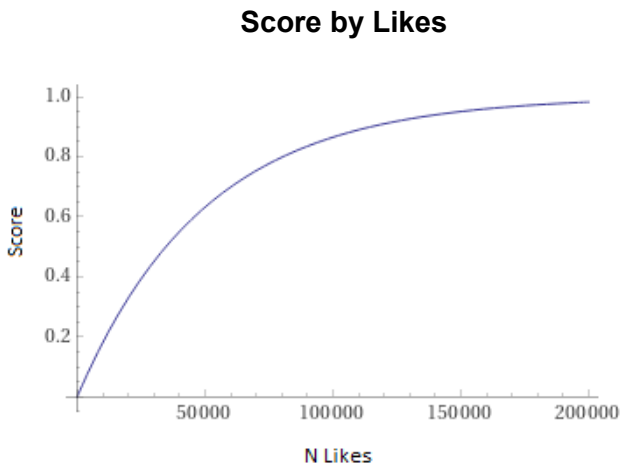



**Figure 4:** Exponential Score function against number of likes

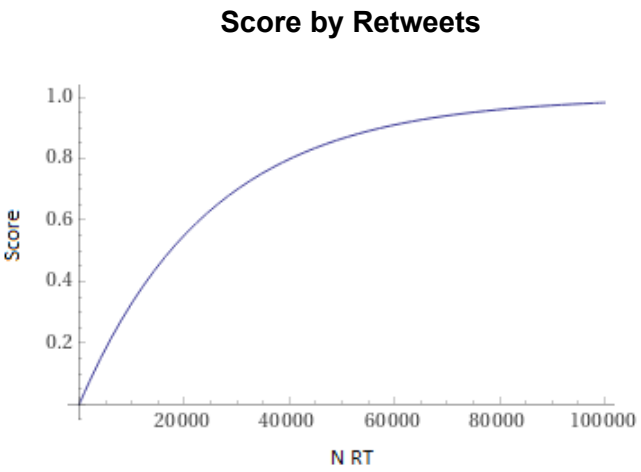



**Figure 5:** Exponential Score function against the number of retweets.

## The final score is given by the ponderation:

$$F.\,S. \;=\; 0.6(Score\ by\ TF\ -\ IDF) \;+\; 0.2(Score\ by\ Likes) \;+\; 0.2(Score\ by\ RT)$$

# 3. Analysis

To analyse the different developed search engines, we will test them with the following queries. These queries have been selected based on the popularity of the terms (keywords ranked by term frequencies or by tf-idf, etc…)

1. *GOP wins elections?*
2. *The symbol of the GOP is an elephant.*
3. *Trump says elections have been corrupted*
4. *Biden takes the action*
5. *Fraud republican election*
6. *Kamala Harris will be the first woman Vice President of the US.*
7. *Biden will fight climate change*
8. *Can I see elections on netflix?*
9. *Thanks realdonaldtrump*
10. *Trump's attempt to steal the election unravels as coronavirus cases surge*

Once the project has been executed, we will find in the output directory a set of files containing the top-20 list of documents (TSV files) for each of the 10 queries, using **tf-idf** + cosine similarity, our **personalized model**, and the **word2vec** + cosine similarity methods with their corresponding scores.

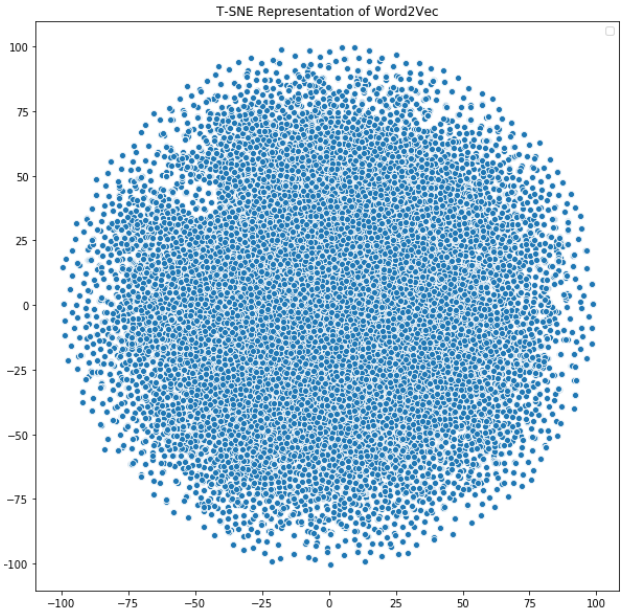
Other representations that could be used instead Word2Vec such as Sentence2vec and Doc2Vec.

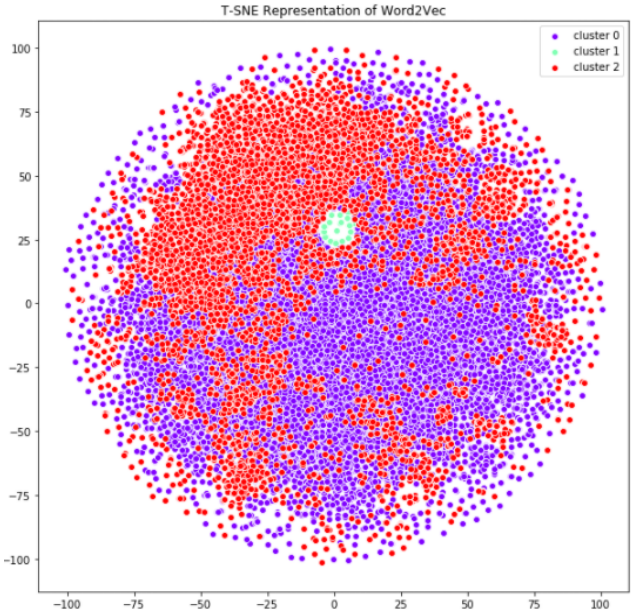
The first representation (Sentence2vec) would be better if the user is querying a long and specific phrase rather than by keywords, so that the search engine could capture the semantic relationships between the words.

A second representation could be Doc2Vec. This way we could code the whole document (tweet), with the same idea of SentenceToVec or rather Word2Vec. But we think that this might not give adequate results, or at least not as efficient as the previous representations, since tweets are limited to 280 characters. As Twitter has this condition, the documents to be searched are small enough to be considered only sentences (or a composition of very few sentences)

## 3.1 Vector representation

Now that everything seems fine, we can proceed to make a deeper analysis of the search engine performance. For that we will choose a vector representation, tf-idf or word2vec, and represent the tweets in a two dimensional scatter plot through the **t-sne** algorithm. To do so, we may need first to represent the word as a vector, and then the tweet, i.e., resulted as the average value over the words involved.




At a first glance, it seems impossible to distinguish any subgroup within the scatter plot as it is a huge dataset. Nevertheless we can take advantage of clustering algorithms, for example with k-means and we can now observe a big cluster (red), and 2 small clusters. These clusters may appear a bit sparse, but this is due to the flatten redimension of the vector, which can only be plotted and “Human”-interpreted in a 2D plane.

At this point you may wander, what are the most relevant keywords in the tweets that are part of each cluster? Well, if we gather some information from each cluster we will find the following information:

**Cluster 0** groups a variety of tweets/ideologies and we can observe that The KeyWord Biden appears 4011 times while Trump appears 6850. In all the dataset we can observe that trump appears more times than biden, but we can consider that cluster 0 is most neutral on average.

C**luster 1**, instead, contains tweets that have a lot of mentions, not about a specific topic but a set of tweets that mentions 5 or more other accounts.

In **Cluster 2** we found that Trump appears almost 6957 times while Biden only 2143. We can say that in this cluster is more relevant the keyword Trump. This may indicate that cluster 2, can be labeled as a Republican cluster.

## 3.2 Diversification

After making several analyses of our dataset, it seems that the content is mainly divided into two political party categories, Republicans and Democrats. (A third noticeable category would be a neutral position).
As we want our search engine to be as neutral as possible, we will diversify the query results by applying a function that exponentially reduces the score of a result depending on the number of outputs in the same category. The diversified function is as follows:

$$Diversified\ Score\ =\ Score \cdot e^{-(\ Position\ of\ the\ result\ within\ the\ category\ )}$$

After re-ranking with this score function, we can notice some difference with respect the Non-neutral search engine.

With the previous classification, 9 of the first 10 results fell on the category of cluster 0. After applying the diversity score the results are appearing in a more diverse way. With the same query we can find 6 results of category 0 and 4 of category 2, these seconds now with a higher position in the ranking.

*The results are shown in the output directory as TSV files.*

Notice that we have used the method coverage: number of different clusters returned by the output. We can see in the screenshots on the other-outputs folder, that the TOP 3 results are the same with or without diversification, but after the third result the scores vary in both versions of scores. Exists some results that are not retrieved for the diversity score and others that are not retrieved for the non-diversity score.

# 4. Link Analysis (extra)

To make a deeper analysis of the dataset, as an extra chapter we are going to take the scrapped tweets and generate a retweet graph (fig. 6). The directed final graph G = (V,E) will be constituted by all the users retweeting at least once, and a generic edge (u,v) which means that users u retweeted at least once a tweet posted by the user v. In this part, we're going to test some contact recommendation algorithms to predict next retweets.

## 4.1 ADAMIC ADAR

Adamic Adar Index is a value that decreases exponentially according to the degree of the neighboring node they have in common, or the sum in case they have more than one. Therefore the value of the Adamic Adar Index will be much higher if they have a common neighbor node that has a very low degree. In this case, it would correspond to two users that make RT to an unpopular user (a user with few retweets).

$$A(x,y) = \sum_{u \in N(x) \cap N(y)} \frac{1}{\log |N(u)|}$$

After testing Adamic-Adar we think that this should be the most accurate algorithm, since it takes into account the most important factors. For instance, it gives more relevance to those tweets that have neighboring users in common, even if the user they link to is not very popular. In fact, when this happens it gives even more importance. This is very close to reality, since some users who follow or retweet another not very famous user, make this user more strange (relevant).

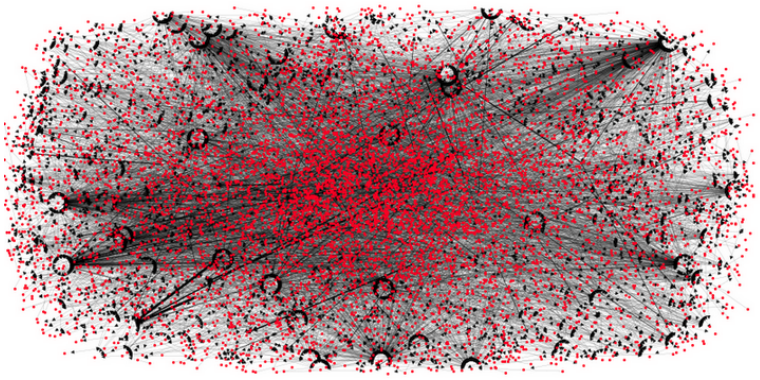


**Figure 6:** Screenshot of the directed graph that represents the users that retweets other users, we fine tune the graph with a weight of each edge, that represents the number of times that one user retweets another user to make a great representation.

# 5. CONCLUSIONS

After much effort and dedication, a search engine has been developed that covers different types of algorithms. This search engine can be considered neutral in terms of politics which allows, through a statistical study on the content of the corpus, to consult the different tweets that can be found, and retrieve those that may be of greater interest without influencing the user's political beliefs.

The greatest disadvantage or handicap that may have been encountered is the limitation due to lack of resources. Both in the Dataset, as well as at the computational level.

A lot of computing power and memory are needed to execute certain functions, and to get the expected results we stuck a lot of time for this limitation. For that reason it is suggested (unless having a good machine) to run the engine on a distributed system, or make use of Google Collab, which offers to users free resources on the cloud that might accelerate the process of execution.

In spite of all these problems, we can ensure that our search engine has acceptable recall, and the actions performed to fine-tune the score works well. We think that if we can compute the results or analyze it with a lot of more data, we can extract better conclusions such as the meaning of the clusters, finetune the number of clusters and better diversity within the search engine.