Algorithmics for Data Mining: Deliverable 3

Democrat Vs. Republican Tweets

Oriol Borrell
FIB - UPC Student
Barcelona, Spain
oriol.borrell.roig@est.fib.upc.edu

April 13, 2020

0 Abstract

We will analyze what do politicians of the Republican and Democrat Parties (in USA) tweet about. We will first see if we can extract any conclusion about particularities of each party tweet. Afterwords we will train a model that will predict, given a tweet, if it belongs to the Republican or to the Democrat party.

1 Context

The tweets obtained are all tweet from 2019 of USA politician's. We have to take into account that Donald Trump (from the Republican party) is the president of USA since January 20, 2017. The Democrat's candidate was Hillary Clinton. The 2019 in USA was a Off-year election.

Republicans and Democrats are the two main and historically the largest political parties in the US. After every election, they hold the majority seats in the House of Representatives and the Senate as well as the highest number of Governors.

2 What do Democrat and Republican members tweet about?

Before analyze the tweets we applied the following preprocessing steps:

- Remove the newline characters
- Remove commonly used ampersand
- Remove ' from contractions such as I'm and don't
- Lowercase the string
- Remove https-links from the string
- Tokenize the string with the given pattern

Once we had the tokens, we computed the frequencies of each token in each party. We created the wordcloud shown in Figure 1.

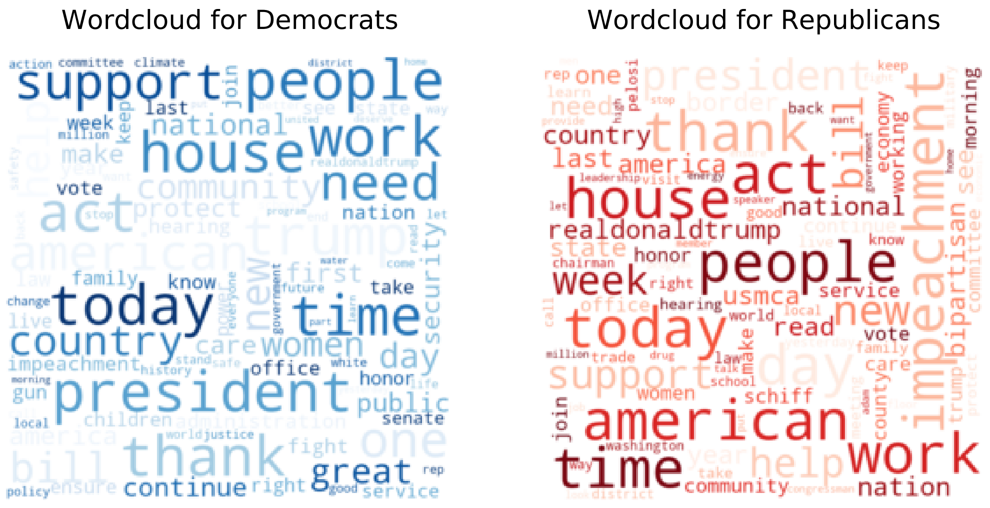

Figure 1: Wordcloud of the top used tokens of each party

Analyzing the results, we observe that the most used words of each party are fairly common words in the field of politicsWe cannot extract any particularity neither from the Democrats nor Republicans.

Doing a bit of research I founded a common concept in *Information Retival* called Term Frequency–Inverse Document Frequency, **TFIDF**. TFIIDF is a way to compute the importance a word is in a document. As shown in Equation 1 is computed using two statics, the *Term Frequency(tf)*, and the *Inverse document frequency(idf)*:

$$tf(t,d) = 0.5 + 0.5 \times \frac{f_{t,d}}{max\{f_{t',d} : t' \in d\}}$$
$$idf(t,D) = log\frac{N}{|d \in D : t \in d|} \tag{1}$$
$$tfidf(t,d,D) = tf(t,d) \times idf(t,D)$$

Where $f_{t,d}$ is the frequency of the token t in the document d, and N is the total number of documents in the collection.

Once we computed the TFIDF, we created another time the wordcloud of each party. The results are shown in Figure 2

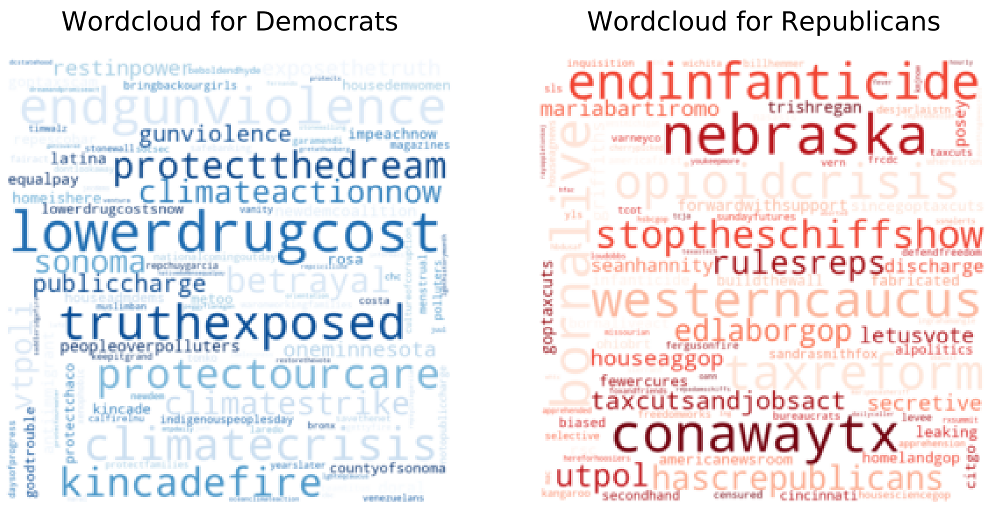

Figure 2: Wordcloud using TFIDF

The obtained wordclouds are very interesting. We see that the Democrats apparently have a larger focus on *climate* and *health care* where the Republicans focus on *anti-abortion* and *tax-cuts*. These topics are all more interesting and polarizing than the previous results, where we saw that both parties often referred to different political personalities, committees and other political jargon. We see a clear indicator, that the phrases for both parties are political slogans, such as *endgunviolence* and *bornalive*. Twitter, therefore, gives us a valuable insight into the key-issues and focal points of each party.

3 Convolutional Neural Networks (CNN)

A Convolutional Neural Network (CNN) is a specific type of artificial neural network that has some hidden layers called convolutional layers, where the transformation is made in this layer is a convolutional operation. CNNs is widely used to do image recognition, image classifications, objects detections, faces recognition etc., that's the reason I choosed this algorithm.

The model type that was used was Sequential. It allows you to build a model layer by layer. In our model we will use the following type of layers:

- **Conv2D:** This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs.
- **MaxPool2D:** Takes the maximum value of the kernel to reduce the number of parameters.
- **SpatialDropout2D:** We randomly set entire kernel to 0. So we will drop them, in order to prevent for overfiting.
- **Flatten:** It reduces the dimension of the data into 1 dimension.
- **Dropout:**It randomly turn off neurons in order to reduce overfitting.
- **Dense:** We take the dimension created in the flattern layer and convert it into the category to predict.

The following table shows the CNN i designed:

Layer(type)	**Units**	**Filter**	**Stride**
Conv2D	8	5	2
Conv2D	32	3	1
MaxPool2D	-	3	2
SpatialDropout2D	-	-	-
Conv2D	64	3	2
Conv2D	64	3	1
MaxPool2D	-	3	2
SpatialDropout2D	-	-	-
Conv2D	128	1	1
Conv2D	128	1	1
Conv2D	128	1	1
SpatialDropout2D	-	-	-
Flatten	-	-	-
Dropout	-	-	-
Dense	5	-	-

Table 1: Accuracy obtained respect k

When we built the model we tried allot of possibles parametritzations for the number of nodes in each Conv2D layer. Finaly, in the first set of layers we have 8 nodes, and 32, tn the second one two layers of 64 nodes, and in the third one three layers of 128 nodes. This are the number of nodes that made us reach the higher accuracy. Another parametrization we choosed was the kernel size, the size of the filter matrix for our convolution. We decided to create 3×3 size filter matrix, except in the first layer, that will be a 5×5 matrix, and in the last set of Conv2D layers.

We also had to define the number of *epochs* (number of times we train the model with the same training/testing data) we will perform. If we set a low number of epochs our model could not be enough trained.

If we set a big number of epochs, our model could be overfitted. In both cases the accuracy will decrease, so the number of epochs important.

So we first built the model with 50 epochs, and plotted the acuracy and lost values of each one. In Figures **??** and **??** we can clearly see that after epoch 15 we have a overfitted model, as is when in both figures the train and test lines are starting to take different directions. So we rebuilt the model with 15 epochs.

With all this parametritzations, once built the model we validated with the data saved for that purpose. We obtained an accuracy of **0.8373**. With this model we can be much more confident of our prediction in comparison to k-NN. We have to take into account that CNN is a neural network, and this works well with huge amount of data. So is probable that if we train the model with much more data, the accuracy will be higher.

4 Evaluation do the results

The results obtained with a KNN clasifier gave us an improvement comparing it with "flipping a coin" to predict the category. However, the accuracy obtained is not the one that we expected. Comparing the two models we created, for this project we would clearly choose the CNN one. The best parametrization we found for the CNN gave us a 0.8373 of accuracy, which we conclude that is an acceptable result for this project.

However, the important thing of this project was not only to obtain a high accuracy, but also to learn the different parts of a CNN and to detect overfitting. We also can affirm that we completed this goal of the project.

5 Implementation

In order to play a little with the model we built, just for fun, we drew som images using the paint application for windows operating systems. We used our model to predict the their category. The images created are the ones shown in Figure **??**:

The Table shows the obtained predictions. Each column represents a image (each letter refers to the letters assigned in Figure **??**). Each row represents the probability the model said of being for a certain category:

Category	**a**	**b**	**c**	**d**	**e**	**f**
Ambulance	0.00%	0.05%	0.12%	0.00%	100.00%	0.00%
Angel	0.00%	0.00%	0.00%	99.97%	0.00%	0.00%
Aircraft carrier	1.49%	99.85%	12.67%	0.00%	0.00%	0.00%
Airplane	98.51%	0.08%	84.13%	0.00%	0.00%	0.00%
Ant	0.00%	0.02%	3.07%	0.03%	0.00%	100.00%

Table 2: Accuracy obtained respect k