Capstone project: report

Jacopo Malatesta

22/2/2020

Contents

Goals of the project

This small research aims at gaining insight into the Twitter content concerning the 2020 Democratic Party presidential primaries. The Twitter discourse around this topic has come under a lot of scrutiny in recent times due to the often aggressive behavior of both the candidates and their supporters.

The goal of this research is to analyze Twitter data concerning the four main Democratic candidates in order to highlight the underlying emotion and sentiment beneath their tweets and to shed light on the most frequent topics and issues. The candidates whose Tweets were analyzed are Bernie Sanders, Elizabeth Warren, Pete Buttigieg and Joe Biden.

Reasons behind my choice

The choice of this topic is partly driven by a long-standing interest in American politics and culture, but more importantly by a desire to deepen my knowledge of digital analysis methods and social media in general. After my master degree, I plan on working in the communication and marketing field and in order to be successful in this area acquiring some basic skills in the analysis of social media data is essential. Though I realize that in order to land a good marketing job many more skills need to be added to my talent stack, this small project might turn out to be a good stepping stone in my professional development.

Main features of the projects

The overall structure of this work was inspired by Céline Van den Rul's article "A Guide to Mining and Analysing Tweets with R", which also served as a source of inspiration for many of the lines of codes that were used.

This project consists of three separate steps: collecting Twitter data, analyzing the content and sentiment of its text and visualizing the results.

The data collection has been carried out by scraping Twitter data with an R studio package called "Rtweet"; the analysis and visualization parts instead required the use of basic text and sentimental analysis tools, such as the syuzhet and tidytext packages. Naturally, more common packages such as rio, dplyr and ggplot2 were used as well throughout the project.

Technical overview

In this part I'll go through each step of the project by commenting the scripts I wrote. However, due to the length of some of the chunks of codes, only some lines will be embedded and commented. Readers interested in having a deeper look at the codes can inspect the complete scripts in the scripts folder.

Packages used

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(rtweet)
library(rio)
library(tidytext)
library(quanteda)
```

```
## Package version: 1.5.2
```

```
## Parallel computing: 2 of 4 threads used.

## See https://quanteda.io for tutorials and examples.

##
## Attaching package: 'quanteda'

## The following object is masked from 'package:rio':
##
##     convert

## The following object is masked from 'package:utils':
##
##     View

library(ggplot2)
library(syuzhet)

##
## Attaching package: 'syuzhet'

## The following object is masked from 'package:rtweet':
##
##     get_tokens
```

Data collection

The candidates' tweets were retrieved with the get_timelines function from the Rtweet package, which allows users to scrape the most recent 3200 tweets from specific accounts.

The argument n allows me to set the number of tweets to scrape, in this case it was 1000 for each candidate.

```
overall_tweets <- get_timelines(c("JoeBiden", "PeteButtigieg", "ewarren", "BernieSanders"), n = 1000)

tweets <- import(here::here("output","tweets.csv"))
```

The data were then saved as comma-separated values files, as this is one of the most universal formats available for storing text and data.

Text pre-processing and text analysis

The text analysis was only conducted on organic tweets, that is tweets that are not retweets.

```
tweets_organic <- tweets[tweets$is_retweet==FALSE, ]
```

The first step consisted in removing unwanted text elements, such as http elements, hashtag mentions, "amp" elements, and punctuation. In order to achieve this, the gsub function was used. This function searches for a pattern and replaces it with a character string.

```
tweets_organic$text <- gsub("https\\S*", "", tweets_organic$text)

tweets_organic$text <- gsub("@\\S*", "", tweets_organic$text)

tweets_organic$text <- gsub("amp", "", tweets_organic$text)

tweets_organic$text <- gsub("[\r\n]", "", tweets_organic$text)

tweets_organic$text <- gsub("[[:punct:]]", "", tweets_organic$text)
```

The next step was using the tidytext package to remove stop words from the words in the text column.

Firstly, I split the text column into different words with the unnest_tokens function.

```
tweets_cleaned <- tweets_organic %>%
  select(text) %>%
  unnest_tokens(word, text)
```

Secondly, I removed the stop words with an anti_join. An anti_join gives us the observations in our original dataset (in this case, the words from bernie_cleaned) that do not exist in the other table (the stop words).

```
tweets_cleaned <- tweets_cleaned %>%
  anti_join(stop_words)

## Joining, by = "word"
```

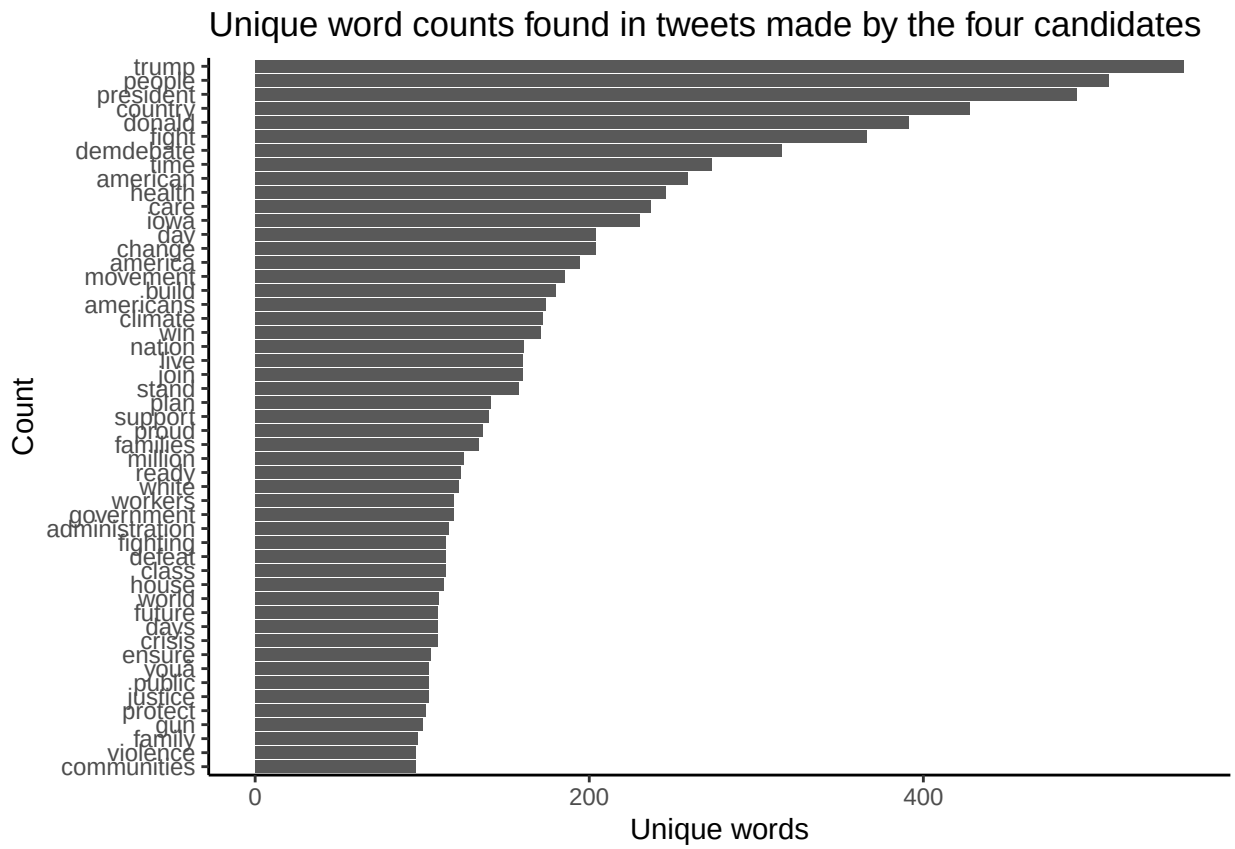
However, there are still some words left in there that need to be removed. In order to do so, I created a custom list (actually a vector) which I then turned into a dataframe.

```
custom_list <- c("aapi2020", "ve", "II", "Il", "ll", "lI", "â","iâ", "weâ", "canâ", "itâ", "caign", "hsl

custom_df <- tibble(joinColumn = custom_list)

tweets_cleaned <- tweets_cleaned %>%
  anti_join(custom_df,
            by = c("word" = "joinColumn"))
```

After gedding rid of the stop words, I briefly analyzed the text by counting the most frequent words in the candidates' tweets. I will delve deeper into these results in the final part of this report.

```
tweets_cleaned %>%
  count(word, sort = TRUE) %>%
  top_n(50) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(x = word, y = n)) +
  geom_col () +
  xlab(NULL) +
  coord_flip () +
  theme_classic() +
  labs (x = "Count",
        y = "Unique words",
        title = "Unique word counts found in tweets made by the four candidates")
```

```
## Selecting by n
```



Sentiment analysis

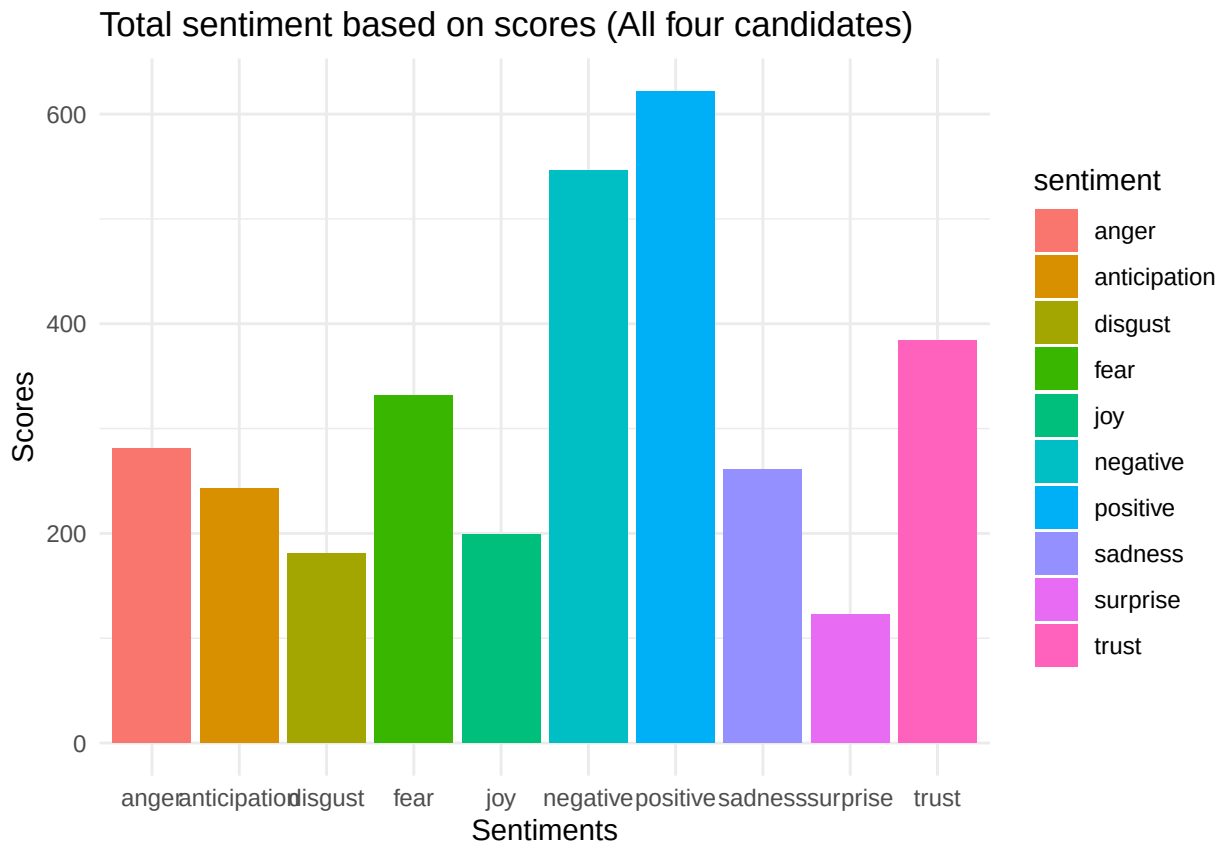
In order to conduct some basic sentiment analysis, I converted the sets of words from UTF-8 to ASCII. Afterwards, I used the get_nrc_sentiment function from the syuzhet package, which calls the NRC sentiment dictionary to calculate the presence of eight different emotions and their corresponding valence in a text file.

```
tweets_cleaned <- iconv(tweets_cleaned, from="UTF-8", to="ASCII", sub="")

tweets_sentiment <- get_nrc_sentiment((tweets_cleaned))
sentimentscores_tweets <- data.frame(colSums(tweets_sentiment[,]))

names(sentimentscores_tweets) <- "Score"

sentimentscores_tweets <- cbind("sentiment"=rownames(sentimentscores_tweets),sentimentscores_tweets)

rownames(sentimentscores_tweets) <- NULL
```

Finally, I visualized the scores for each emotion with the ggplot2 package.

```
ggplot(data=sentimentscores_tweets,aes(x=sentiment,y=Score))+
  geom_bar(aes(fill=sentiment),stat = "identity")+
  theme(legend.position="none")+
```

```
xlab("Sentiments")+ylab("Scores")+
ggtitle("Total sentiment based on scores (All four candidates)")+
theme_minimal()
```



Problems and shortcomings

This section of the report will go through the main challanges and struggles I have encountered in my work, as well its failures and shortcomings.

Lack of a permanent dataset

As explained in the previous section, the Twitter data have been collected with the `get_timelines` function from the R tweet dataset. This function retrieves the most recent tweets shared by any specified Twitter account. This feature, albeit useful, lead to the first setback I suffered in this work: since it always retrieves the most recent tweets, this function gets different tweets depending on when it is run.

As a result, in this project I wasn't able to produce a fixed and unchanging dataset. So, for example, if one year from now other users were to run my lines of code to scrape the candidates' tweets, they would not get the same tweets that I am working on today. Therefore the reproducibility of my work and analysis is compromised.

As far as I know, the Rtweet package package doesn't allow users to use the get_timelines function with a specific time interval (the "from" and "until" arguments are only available for the search_tweets function).

Removing stop words

Removing stop words turned out to be more challenging than expected. This was mostly due to the fact that the `anti_join(stop_words)` function did not remove all the stop words I intended to leave out. However, after a couple of quick Google searches I found out a way to add some extra stop words, by inserting them manually into a dataframe and then removing from the set of words via an anti_join, as I explained in the previous section.

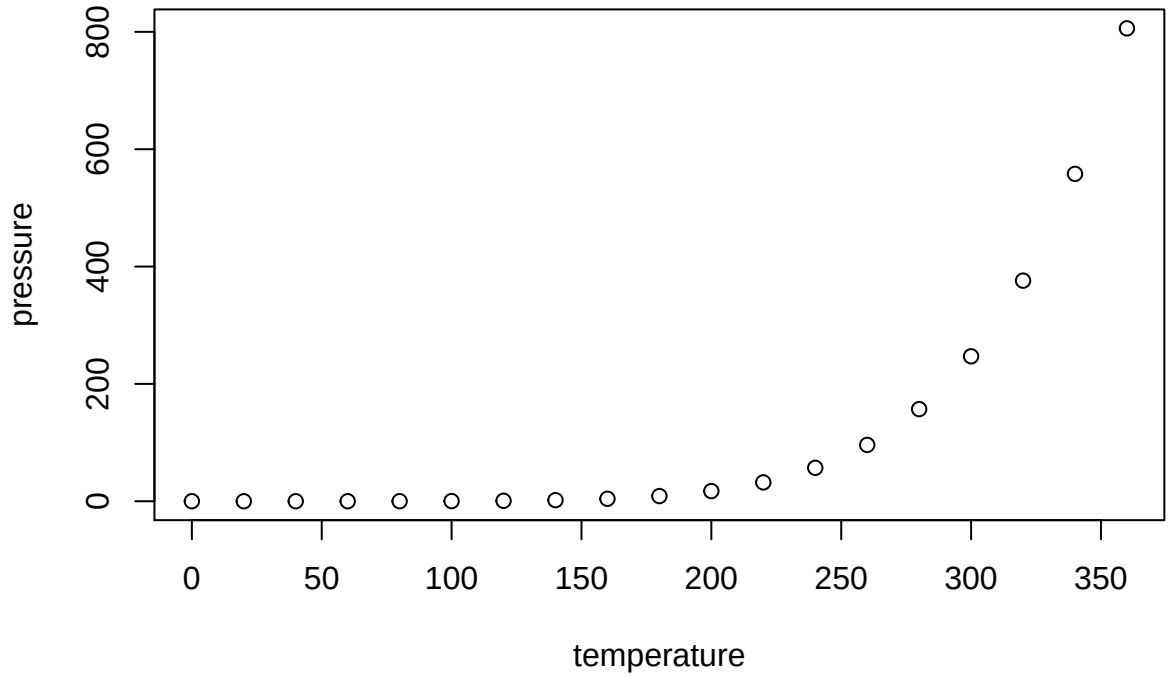
Commenting the results

Things this project taught me

This project turned out to be useful in more ways than one. There are many things I learned by working on it that I will treasure in the future:

- **Learning to use rtweet**: in recent years I developed an interest in Twitter, so I had a natural curiosity in learning to use this package. Rtweet is just as useful as it is easy to use (although, as I mentioned in previous sections, I had some issues with the `get_timelines` function). This project allowed to delve deeper into its characteristics and its functions, as well as its limitations (most notably the fact that users cannot scrape more than 18 thousands tweets every fifteen minutes).
- **Learning the basics of text analysis**: before this course and this project, I only had a vague idea as to what text and sentiment analysis were. Although I'm still far from being an expert of course, now I have a better grasp of these topics. Most importantly, I had the chance to learn these techniques hands-on, instead of just studying them on textbooks. However, I remain skeptical about sentiment analysis and the very arbitrary way in which it assigns emotions and sentiments to words, often regardless of their context.

You can also embed plots, for example:



Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.