Trum Tweets New

Mohamad Yassin

5/28/2020

Bag of Words

The method used for this analysis is the Bag of Words. It is a simple but powerful natural language processing model. The text is represented as the bag of its words, disregarding grammar and even word order but keeping multiplicity.

Trump Tweets Visual Analysis

The purpose of this analysis is exploratory. It aims to explore the difference in language used by president Trump before and after his presidency.

Two data sets were used for this purpose: 1)Trump tweets between 2010-2015 including 20,512 tweets 2)Trump tweets between 2016-May,21 2020 including 19,692 tweets. Given that the average tweet has 55 words, each dataset includes close to a million words.

The data was downloaded from the Trump Twitter Archive. Both data sets are available to download from the github repo.

The data was taken as is. Data integrity wasn't validated.

Let's start by reading the files

Make sure to have your files in a csv format and set stringAsFactor to FALSE

```
df_1 <- read.csv('Trump Tweets 2010-2015.csv', stringsAsFactors = F)
df_2 <- read.csv('Trump Tweets 2016-2020.csv', stringsAsFactors = F)
```

Data manipulation and cleaning

Create a character vector out of the dataframe

```
tweet1 <- df_1$text
tweet2 <- df_2$text
```

I always recommend testing you char vector bfor proceeding:

```
#test
head(tweet1)
str(tweet1)
class(tweet1)
```

Make a vector source

```
library(tm)
vecSource1 <- VectorSource(tweet1)
vecSource2 <- VectorSource(tweet2)
```

Make a volatile corpus

```
tweet_corp1 <- VCorpus(vecSource1)
tweet_corp2 <- VCorpus(vecSource2)
```

Clean the corpus:

The following method isn't the most efficient way to clean data. Writing a function is the way to go. I wrote a function which needed debugging so I added this alternative method for now. You can look at the function at the repo

first corpus

```
# Transform to lower case
clean_corpu1 <- tm_map(tweet_corp1, content_transformer(tolower))
# Remove punctuation
clean_corpu1 <- tm_map(clean_corpu1, removePunctuation)
# Remove @ ! / signs
clean_corpu1 <- tm_map(clean_corpu1, content_transformer(gsub), pattern = "@ | ! | /", replacement = ""
# Remove stopwords
clean_corpu1 <- tm_map(clean_corpu1, removeWords, words = c(stopwords("en"), "donaldtrump", "trump", "p
# Strip whitespace
clean_corpu1 <- tm_map(clean_corpu1, stripWhitespace)
```

second corpus

```
# Transform to lower case
clean_corpu2 <- tm_map(tweet_corp2, content_transformer(tolower))
# Remove punctuation
clean_corpu2 <- tm_map(clean_corpu2, removePunctuation)
# Remove @ ! / signs
clean_corpu2 <- tm_map(clean_corpu2, content_transformer(gsub), pattern = "@ | ! | /", replacement = ""
# Remove stopwords
clean_corpu2 <- tm_map(clean_corpu2, removeWords, words = c(stopwords("en"), "donaldtrump", "trump", "p
# Strip whitespace
clean_corpu2 <- tm_map(clean_corpu2, stripWhitespace)
```

Create a Term Document Matrix. Since the next step involves creating a matrix, matrices can occupy large memory space. In our case close to 5GB for each matrix. Since we are only going to use the top terms in our analysis, I removed sparse terms

```
tweet_tdm1 <- removeSparseTerms(TermDocumentMatrix(clean_corpu1),sparse=0.995)
tweet_tdm2 <- removeSparseTerms(TermDocumentMatrix(clean_corpu2),sparse=0.995)
```

We convert the TDMs into matrices.By running the dimensions, we can see that the sparsity in the first matrix is higher then the second

```
tweet_m1 <- as.matrix(tweet_tdm1)
tweet_m2 <- as.matrix(tweet_tdm2)
dim(tweet_m1)
```

```
## [1]   244 20512
```

```
dim(tweet_m2)
```

```
## [1]   446 19692
```

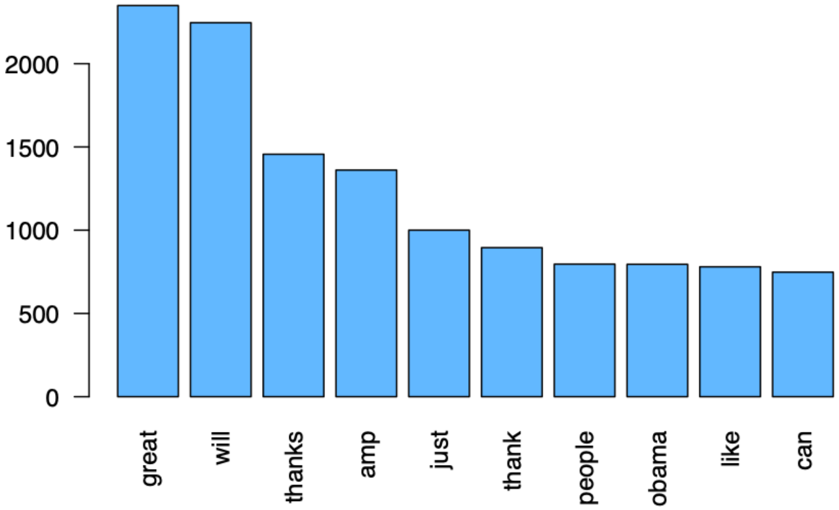
Now we add the row sums to get term frequencies, then we sort them in descending order

```
term_freq1 <- rowSums(tweet_m1)
term_freq2 <- rowSums(tweet_m2)
#sort
term_freq1 <- sort(term_freq1, decreasing = T)
term_freq2 <- sort(term_freq2, decreasing = T)
```

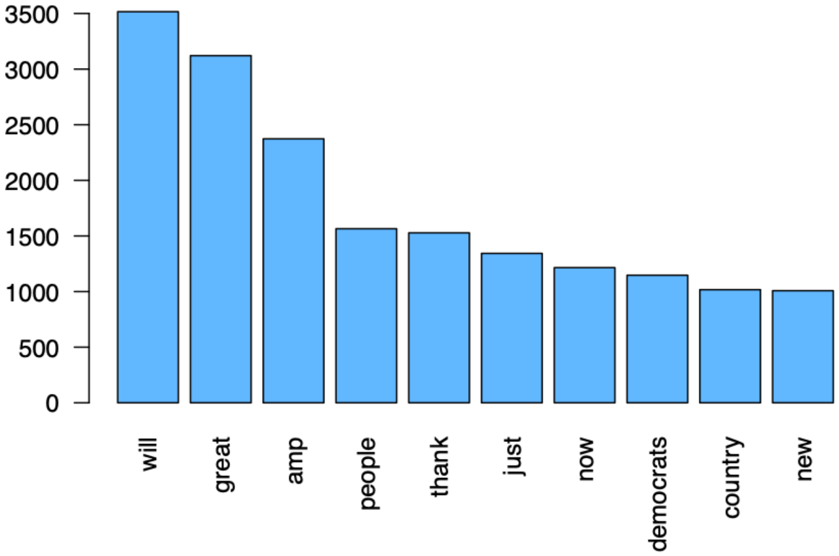
Bar Plot

Let's explore the top 10 terms in each document using a bar plot. Again we can see the higher sparsity in the first "before presidency" document by looking at the word counts. The language became more concentrated in the "after presidency" period.

```
barplot(term_freq1[1:10], col="steelblue1", las=2)
```



```
barplot(term_freq2[1:10], col="steelblue1", las=2)
```



Word Clouds

We need to convert the term frequencies into data frames befor we run our word cloud vizualizations

```r
wc_df1 <- data.frame(term = names(term_freq1), num = term_freq1)
wc_df2 <- data.frame(term = names(term_freq2), num = term_freq2)
```

Let's take a look at our word clouds. I find this particular visualiztion very effective at showing the content of the text. They show us what's on the surface, which is a good way to understand the big picture.

Words are the secret to creation. What I found interesting in the language of president Trump was the use of the word "will". Which reflects his strong will that enabled him to run against the current and win. He used positive words such as "great" and "thanks" consistently. "Obama" showed up during the first period. The second period new words emerged such as "fake", "hillary" and "democrats". A trend of negative words was introduced in the second period.

```r
library(wordcloud2)
wordcloud2(wc_df1, size = 1,
           color = c("tomato","gold","black"), backgroundColor = "white")
```

```r
wordcloud2(wc_df2, size = 1,
           color = c("tomato","gold","black"), backgroundColor = "white")
```

We need to run some data manipulation and cleaning before we can make use of commonality and comparison clouds

Let's starting by making one character vector of length 2 that includes both of our documents

```r
all_tweets1 <- paste(df_1$text, collapse = "")
all_tweets2 <- paste(df_2$text, collapse = "")
all_tweets <- c(all_tweets1,all_tweets2)
```

Create VC and VCorpus

```r
all_vc <- VectorSource(all_tweets)
all_corpus <- VCorpus(all_vc)
```

Clean the corpus

```r
# Transform to lower case
all_clean <- tm_map(all_corpus, content_transformer(tolower))
# Remove punctuation
all_clean <- tm_map(all_clean, removePunctuation)
# Remove @ ! / signs
all_clean <- tm_map(all_clean, content_transformer(gsub), pattern = "@ | ! | /", replacement = "")
# Remove stopwords
all_clean <- tm_map(all_clean, removeWords, words = c(stopwords("en"), "donaldtrump", "trump", "presiden
# Strip whitespace
all_clean <- tm_map(all_clean, stripWhitespace)
```

Create TDM then convert to matrix

```r
all_tdm <- TermDocumentMatrix(all_clean)
all_m <- as.matrix(all_tdm)
```

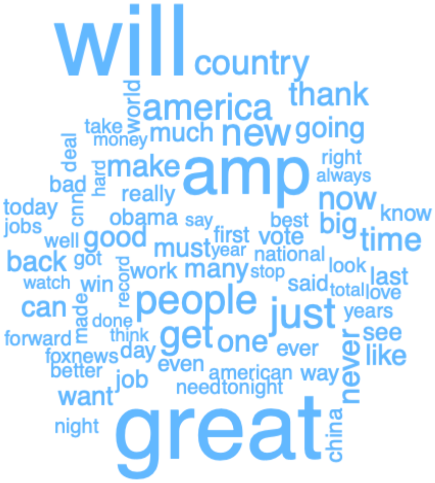
Commonality cloud depicts the common words included within the two periods.

Unfortunately R wasn't able to plot the commonality cloud correctly. Please let me know if you have a fix to the plot canvas limit ***

```r
library(wordcloud)
```

```
## Loading required package: RColorBrewer
```

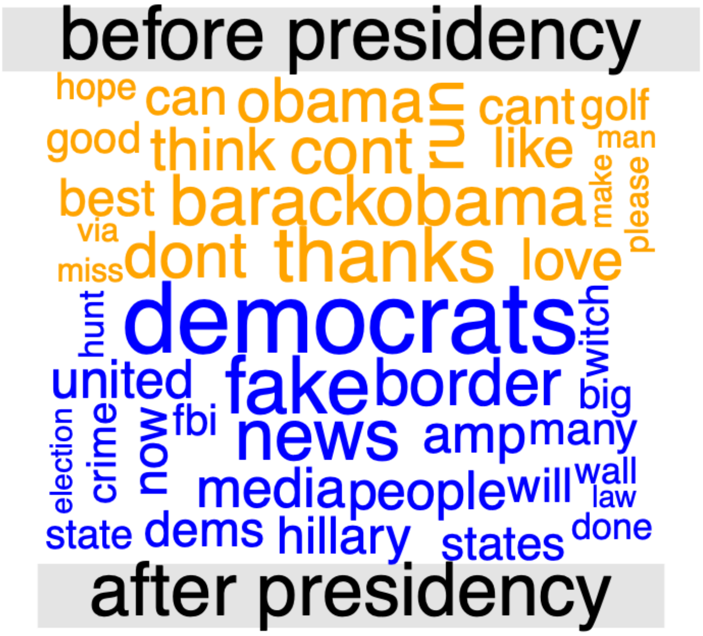
```r
commonality.cloud(all_m, colors = "steelblue1", max.words = 75)
```



Comparison cloud shows the distinct words within each period

Unfortunately R wasn't able to plot the comparison cloud correctly. Please let me know if you have a fix to the plot canvas limit ***

```r
colnames(all_m) <- c("before presidency", "after presidency")
comparison.cloud(all_m,
                 colors = c("orange", "blue"),
                 max.words = 100)
```



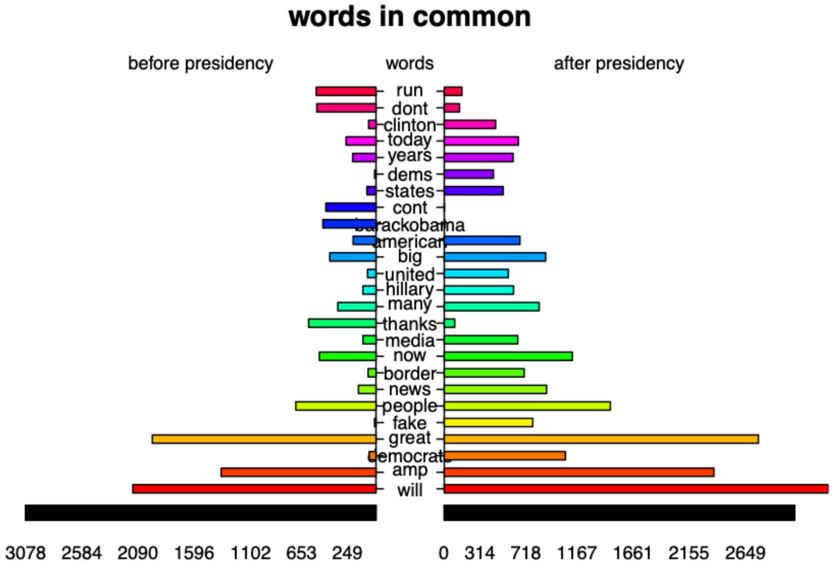
Pyramid Plot

Pyramid plot provides a visual comparison between word counts across the two periods. First, we need to do some manipulation to get our top 25 comparison words

```
common_words <- subset(all_m,
                       all_m[,1]>0 & all_m[,2]>0)
difference <- abs(common_words[,1] - common_words[,2])
common_words <- cbind(common_words, difference)
common_words <- common_words[order(common_words[,3],
                                   decreasing = T),]
top25_df <- data.frame(x = common_words[1:25,1],
                       y= common_words[1:25,2],
                       labels = rownames(common_words[1:25,]))
```

Create pyramid plot

```
library(plotrix)
pyramid.plot(top25_df$x, top25_df$y,
             labels = top25_df$labels,
             main = "words in common",
             gap = 300, laxlab = NULL,
             raxlab = NULL,
             space = 0.5,
             unit = NULL,
             labelcex=0.75,
             top.labels = c("before presidency" ,"words","after presidency"))
```



```
## 3378 3378
```

```
## [1] 5.1 4.1 4.1 2.1
```

Cluster analysis

Hierarchical clustering is a powerful mthod for algorithmic analysis. It can also provide visual guidance. The hierarchical cluster object presents a set of dissimilar words

Let's start the data manipulation by removing sparce terms from our TDM, which will help us create a dendogram

```
rm_sparce_tdm1 <- removeSparseTerms(tweet_tdm1, sparse = 0.975)
rm_sparce_tdm2 <- removeSparseTerms(tweet_tdm2, sparse = 0.975)
```

Create a matrix

```
sparce_m1 <- as.matrix(rm_sparce_tdm1)
sparce_m2 <- as.matrix(rm_sparce_tdm2)
```

Create a distribution

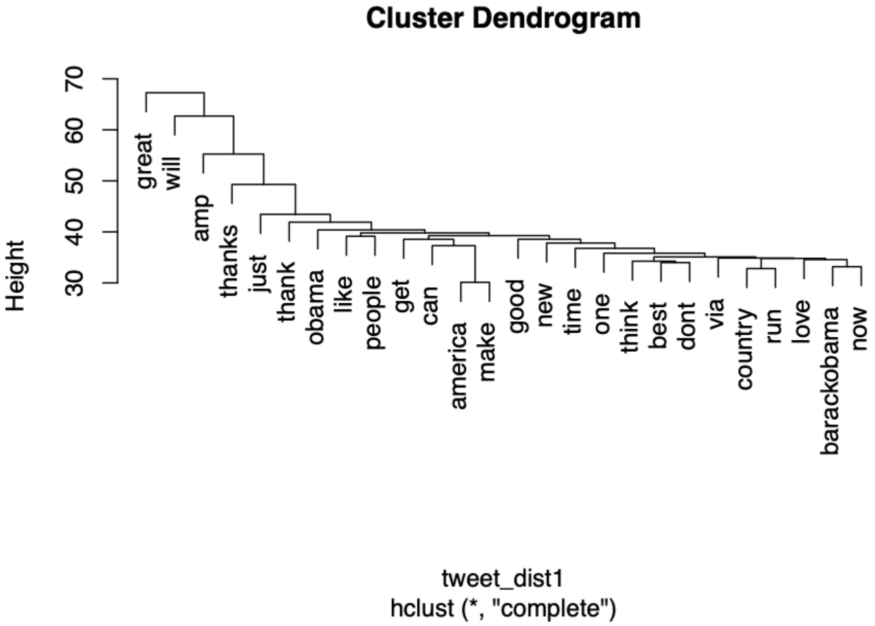
```
tweet_dist1 <- dist(sparce_m1)
tweet_dist2 <- dist(sparce_m2)
```

Create a hierarchical cluster

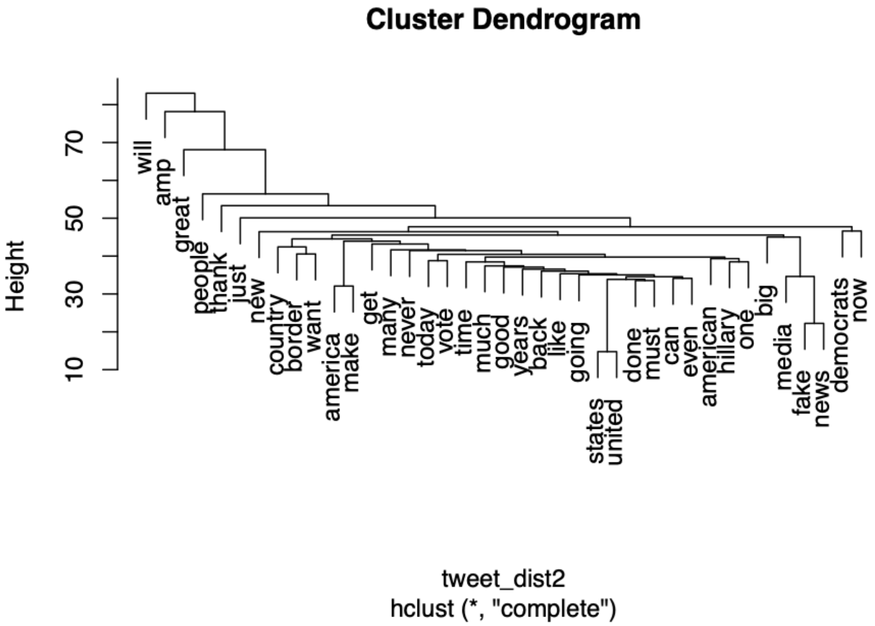
```
hc1 <- hclust(tweet_dist1)
hc2 <- hclust(tweet_dist2)
```

Let's plot our clusters

```
plot(hc1)
```



```
plot(hc2)
```



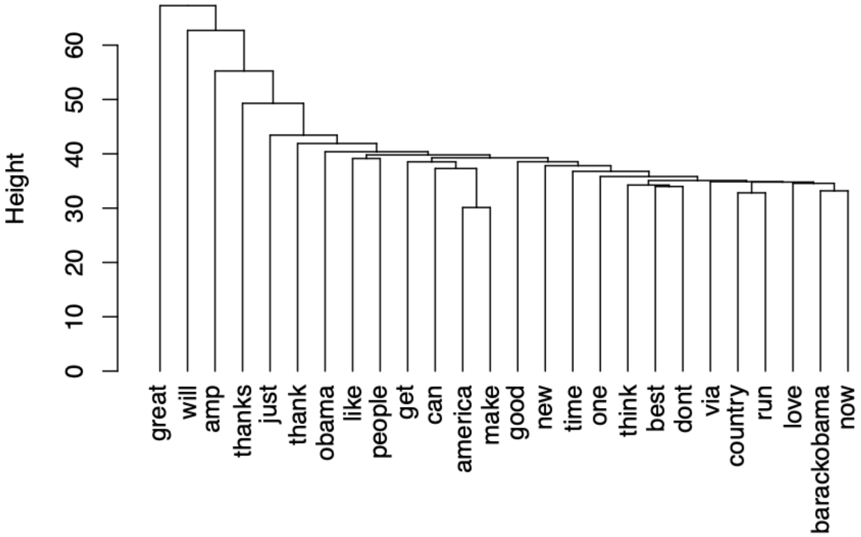
Create a dendogram.

Class “dendrogram” provides general functions for handling tree-like structures. There are multiple ways to visualize dendograms and hierarchical clusters

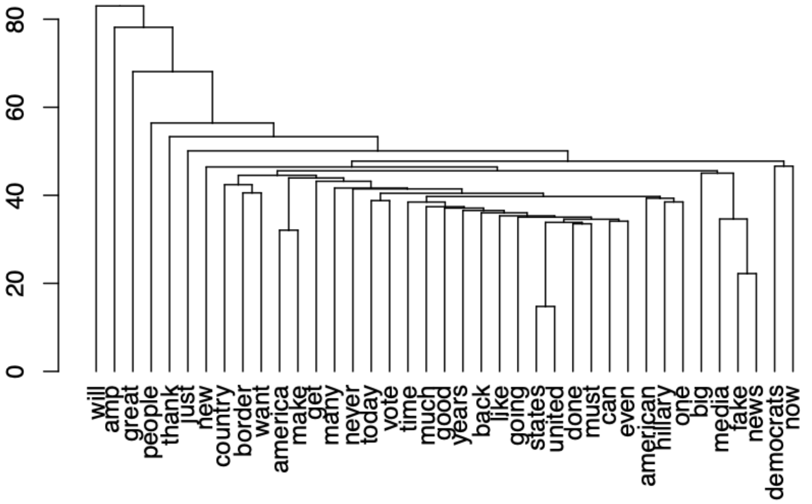
```r
hcd1 <- as.dendrogram(hc1)
hcd2 <- as.dendrogram(hc2)
```

Plot the dendograms

```r
plot(hcd1, ylab = "Height")
```


Height
0
10
20
30
40
50
60
great
will
amp
thanks
just
thank
obama
like
people
get
can
america
make
good
new
time
one
think
best
dont
via
country
run
love
barackobama
now

```r
plot(hcd2)
```


0
20
40
60
80
will
amp
great
people
thank
just
new
country
border
want
america
make
get
many
never
today
vote
time
much
good
years
back
like
going
states
united
done
must
can
even
american
hillary
one
big
media
fake
news
democrats
now

N-grams

Now let's move on to n-gram analysis. n-grams computes the (contiguous sub-sequences of length n) of a given sequence. In our case, we will create a bi-gram; n-gram of length = 2. This will allow us to analyze our text by looking at sub-sequence of two words which can offer further understanding.

First step is to create a tokenizer function that we can use in our TDMs

```
bigramTokens <-function(x){
  unlist(lapply(NLP::ngrams(words(x), 2), paste, collapse = " "),
         use.names = FALSE)
}
```

Create TDM

```
bigram_tdm1 <- TermDocumentMatrix(clean_corpu1,                                 control=list(tokenize=b

bigram_tdm2 <- TermDocumentMatrix(clean_corpu2,                               control=list(tokenize=big
```

Before creating matricies, we will remove sparce terms because above TDM are huge and creating matrices out of them can result in each matrix occupying close to 10GB of memory soace. Which is too high for most average desktops and laptops.

```
bigram_sparce1 <- removeSparseTerms(bigram_tdm1, sparse = 0.999)
bigram_sparce2 <- removeSparseTerms(bigram_tdm2, sparse = 0.999)
```

Create matrix

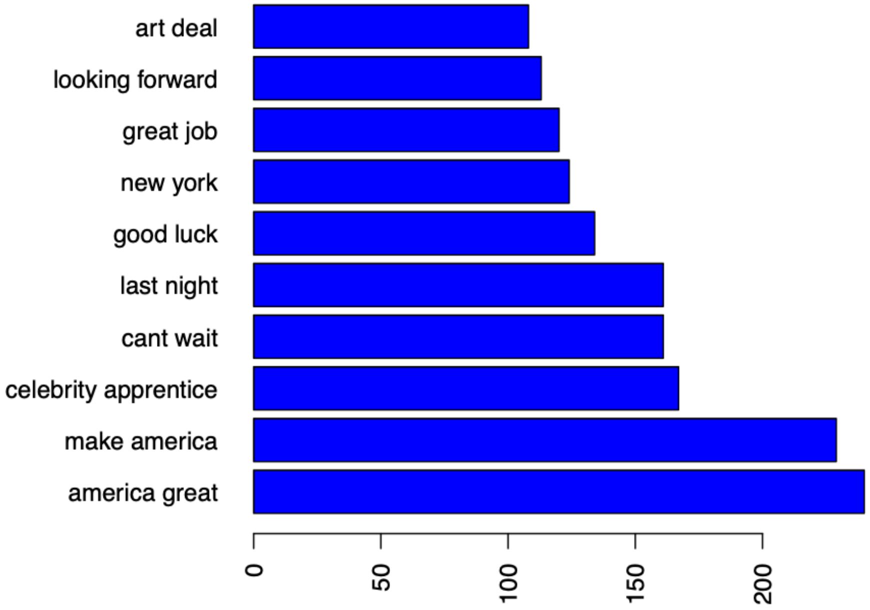
```
bigram_m1 <- as.matrix(bigram_sparce1)
bigram_m2 <- as.matrix(bigram_sparce2)
```

Create terms frequency then sort

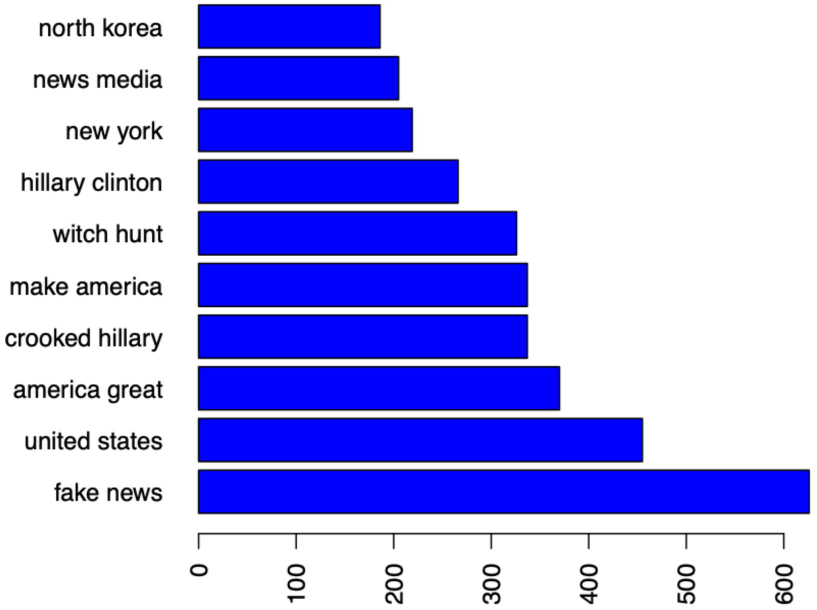
```
bigram_freq1 <- rowSums(bigram_m1)
bigram_freq2 <- rowSums(bigram_m2)

sort_freq1 <- sort(bigram_freq1, decreasing = T)
sort_freq2 <- sort(bigram_freq2, decreasing = T)
```

Bar plot. We can see the difference between word analysis and bi-gram analysis here. Some interesting terms appear!

```
par(mar=c(3, 9.5, 1, 2))
barplot(sort_freq1[1:10], col = "blue", las=2, horiz = T)
```



```r
barplot(sort_freq2[1:10], col = "blue", las=2, horiz = T)
```



Bi-gram wordclouds

First let's convert the term frequencies into dataframes

```
word_freq1 <- data.frame(term=names(sort_freq1), num=sort_freq1)
word_freq2 <- data.frame(term=names(sort_freq2), num=sort_freq2)
```

Make wordcloud. Bi-gram word clouds give more detailed understanding of the text

```
wordcloud2(word_freq1, size = 1,
           color = c("tomato","gold","black"), backgroundColor = "white")
```

```
wordcloud2(word_freq2, size = 1,
           color = c("tomato","gold","steelblue"), backgroundColor = "white")
```