NLP Word Prediction EDA

Aki Taniguchi

31/01/2020

I. Synopsis

This paper is part of the Coursera Capstone Project which consists in developing an algorithm that predicts the next word of an input. Beforehand, exploratory data analysis is required in order to better understand the data we deal with. We will first look at how the 3 provided text files are composed with, and proceed to an N-gram analysis to see its structure.

II. Basic Data Exploration

Let us first load all necessary libraries and set up the workspace.

```
# Loading libraries
library(NLP)
library(tm)
library(RWeka)
library(dplyr)
library(ggplot2)
library(gridExtra)
library(LaF)
library(knitr)

# Setting up workspace
setwd("C:/Users/tngch/Documents/R/Learning/10. NLP - Capstone/en_US")
```

We will now create two functions which will help us understand the basic information of our text files. We will need the following information: the number of Lines and number of Words that compose the data.

```
# Create a function which gets the number of lines
# In order to not load the full data, we won't be using readText, but readBin instead,
# using a remote connexion
get_line_count <- function(file_name) {

    con <- file(file_name, open="rb")
    len_file <- 0
    while (length(chunk <- readBin(con, "raw", 65536)) > 0) {
        len_file <- len_file + sum(chunk == as.raw(10L))
    }
    close(con)

    rm(con, chunk)

    return(len_file)
}


# Create a function which gets the number of word
# Likewise, we use a connexion and read Line by Line
get_word_count <- function(file_name, len_file) {

    con <- file(file_name, "r")
    len_word <- 0
    for (i in seq(1:len_file)) {
        sentence = readLines(con, 1)
        sentence = strsplit(sentence, " ")
        sentence = unlist(sentence, use.names=F)
        len_word = len_word + length(sentence)
    }
    close(con)
    rm(sentence, con, i)
```

```
    return(len_word)
}
```

Now applying these two functions to our datasets, namely Twitter, Blog and News.

```
file_name <- "en_US.twitter.txt"
tweet_length <- get_line_count(file_name)
tweet_words <- get_word_count(file_name, tweet_length)

file_name <- "en_US.blogs.txt"
blogs_length <- get_line_count(file_name)
blogs_words <- get_word_count(file_name, blogs_length)

file_name <- "en_US.news.txt"
news_length <- get_line_count(file_name)
news_words <- get_word_count(file_name, news_length)

base_info <- data.frame("Source"=c("Twitter", "Blogs", "News"),
                        "Line Count"=c(tweet_length, blogs_length, news_length),
                        "Words count"=c(tweet_words, blogs_words, news_words))

kable(format(base_info, big.mark = ","), caption = "Base Information", align=rep("c", 3))
```

Table 1: Base Information

Source	Line.Count	Words.count
Twitter	2,360,148	30,373,543
Blogs	899,288	37,334,131
News	1,010,242	2,643,969

III. N-gram Exploration

Now that we have the basic information, we want to analyze further through an n-gram analysis. In order to do so however, we need to process the text using the "tm" package, which transforms the dataset into a Corpus. To that extent, we will create a function which transforms the text file into a Corpus, process it ("tm" package) and transforms it into a Frequency dataframe ("RWeka" package).

```
# Create a function which process the data (given we will apply it to the 3 documents)
text_processing <- function(text, n) {

    # Determine the "n" to apply to the RWeka N-gram tokenizer
    # "n" has been ceiled to 3 but the same logic can be extended for bigger "n" if needed be
    if (n == 1) {
        n_gram = function(x) {NGramTokenizer(x, Weka_control(min=1, max=1))}
    } else if (n == 2) {
        n_gram = function(x) {NGramTokenizer(x, Weka_control(min=2, max=2))}
    } else {
        n_gram = function(x) {NGramTokenizer(x, Weka_control(min=3, max=3))}
    }

    # Define a string replacing function to help tm_map
    kill_chars <- content_transformer(function (x , pattern) gsub(pattern, " ", x))

    df <- text %>%
    # Converting the text into a Corpus thanks to the tm package
        VectorSource() %>%
        VCorpus() %>%

    # Process the corpus with the "tm_map" function
    # Note that the basic functions are almost all used except for stemming
    # which shouldn't be used for this exercise
        tm_map(content_transformer(tolower)) %>%
        tm_map(removePunctuation) %>%
        tm_map(stripWhitespace) %>%
        tm_map(removeNumbers) %>%
        tm_map(removeWords, stopwords("english")) %>%

    # Remove all special characters (which can't be treated with TermDocumentMatrix)
        tm_map(kill_chars, "/") %>%
        tm_map(kill_chars, "@") %>%
        tm_map(kill_chars, "\\|") %>%
        tm_map(kill_chars, "â") %>%
        tm_map(kill_chars, "<") %>%
        tm_map(kill_chars, "~") %>%
        tm_map(kill_chars, "#") %>%
        tm_map(kill_chars, "ÿ") %>%
        tm_map(kill_chars, "ð") %>%
        tm_map(kill_chars, "®") %>%
        tm_map(kill_chars, "€") %>%
        tm_map(kill_chars, "") %>%
        tm_map(kill_chars, "¬") %>%
        tm_map(kill_chars, "•") %>%
        tm_map(kill_chars, "œ") %>%
        tm_map(kill_chars, """) %>%
```

```
        tm_map(kill_chars, "¦") %>%
        tm_map(kill_chars, "ž") %>%
        tm_map(kill_chars, "¤") %>%
        tm_map(kill_chars, "¶") %>%

    # Transform the output into a dataframe (through a Matrix), with the word as first column
        TermDocumentMatrix(control = list(tokenize = n_gram)) %>%
        as.data.frame.matrix() %>%
        mutate(word = row.names(.)) %>%

    # Sum all all column and keep sum only, to have a Word/Frequency table
        mutate(frequency = rowSums(select(., -"word"))) %>%
        arrange(desc(`1`)) %>%
        select(word, frequency) %>%
        arrange(desc(frequency))

    return(df)
}
```

Now that the function is created, we can proceed to analyzing word frequencies in each text of each source. Note that we used a random sampling based on the LaF package, where we keep only 0.3% of the original corpus. Let us see how many lines and words do we have after this sampling:

```
file_name <- "en_US.twitter.txt"
tweet_length_sample <- round(as.numeric(get_line_count(file_name)) * 0.003)
tweet_words_sample <- get_word_count(file_name, tweet_length_sample)

file_name <- "en_US.blogs.txt"
blogs_length_sample <- round(as.numeric(get_line_count(file_name)) * 0.003)
blogs_words_sample <- get_word_count(file_name, blogs_length_sample)

file_name <- "en_US.news.txt"
news_length_sample <- round(as.numeric(get_line_count(file_name)) * 0.003)
news_words_sample <- get_word_count(file_name, news_length_sample)

base_info <- data.frame("Source"=c("Twitter", "Blogs", "News"),
                        "Line Count"=c(tweet_length_sample, blogs_length_sample, news_length_sample),
                        "Words count"=c(tweet_words_sample, blogs_words_sample, news_words_sample))

kable(format(base_info, big.mark = ","), caption = "After sampling", align=rep("c", 3))
```

Table 2: After sampling

Source	Line.Count	Words.count
Twitter	7,080	90,523
Blogs	2,698	110,076
News	3,031	105,070

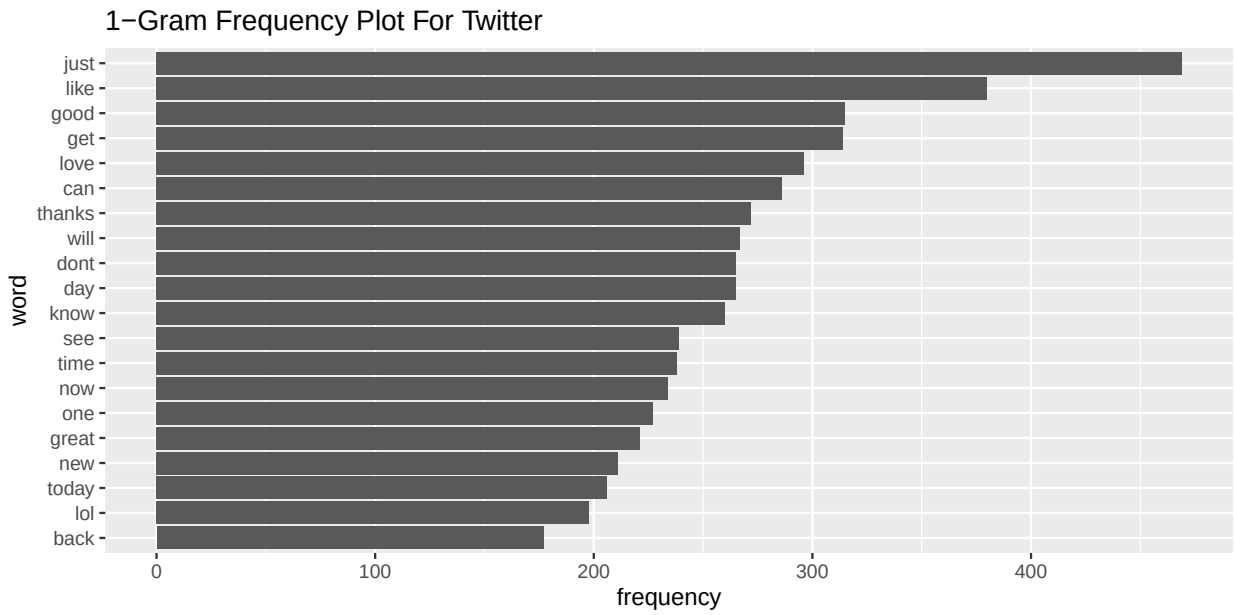
This is not huge, but the memory capacity of the working PC is very limited, while the "tm" package Corpus transformation requires a lot space. To deal with this issue, we will process to a random sampling, which should make the data more representative. Note that when training the prediction algorithm, we will process

to this random sampling several time (e.g. bootstrap), so that we can cope with the memory limitation of reading files in one go.

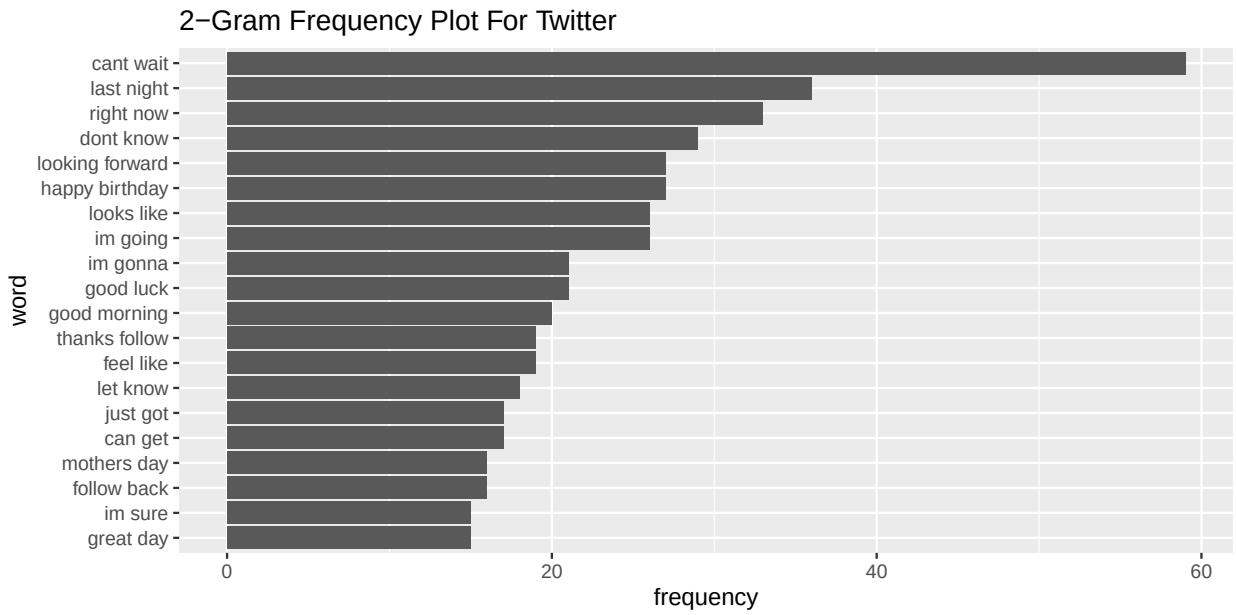
Once the random sampling done (using the "LaF" package "sample_lines" function), we will plot a frequency table for the first 20th most frequent words, up to a 4-Gram table.

III.A. Twitter

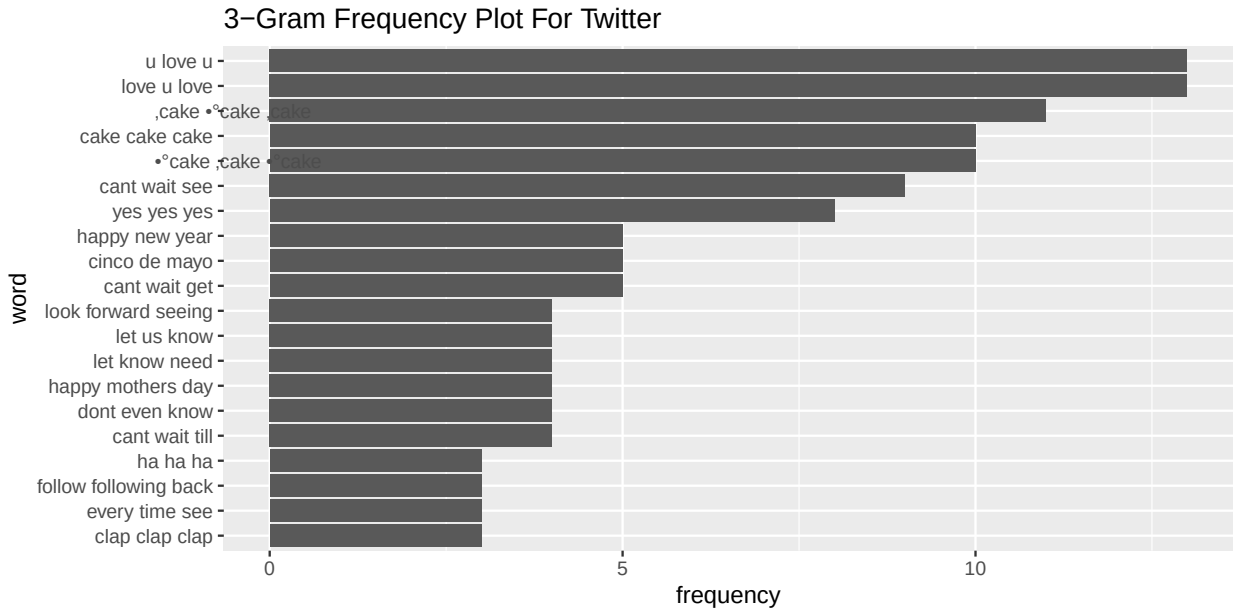
```
file_name <- "en_US.twitter.txt"
tweet_info <- text_processing(sample_lines(file_name, n=tweet_length*0.003, tweet_length), 1)
plotDf <- tweet_info[1:20,]
plotDf$word <- reorder(plotDf$word, plotDf$frequency)
g1 = ggplot(plotDf, aes(x = word, y = frequency)) + geom_bar(stat = "identity") + coord_flip() +
    ggtitle("1-Gram Frequency Plot For Twitter")

tweet_info <- text_processing(sample_lines(file_name, n=tweet_length*0.003, tweet_length), 2)
plotDf <- tweet_info[1:20,]
plotDf$word <- reorder(plotDf$word, plotDf$frequency)
g2 = ggplot(plotDf, aes(x = word, y = frequency)) + geom_bar(stat = "identity") + coord_flip() +
    ggtitle("2-Gram Frequency Plot For Twitter")

tweet_info <- text_processing(sample_lines(file_name, n=tweet_length*0.003, tweet_length), 3)
plotDf <- tweet_info[1:20,]
plotDf$word <- reorder(plotDf$word, plotDf$frequency)
g3 = ggplot(plotDf, aes(x = word, y = frequency)) + geom_bar(stat = "identity") + coord_flip() +
    ggtitle("3-Gram Frequency Plot For Twitter")

g1
```




```
g2
```



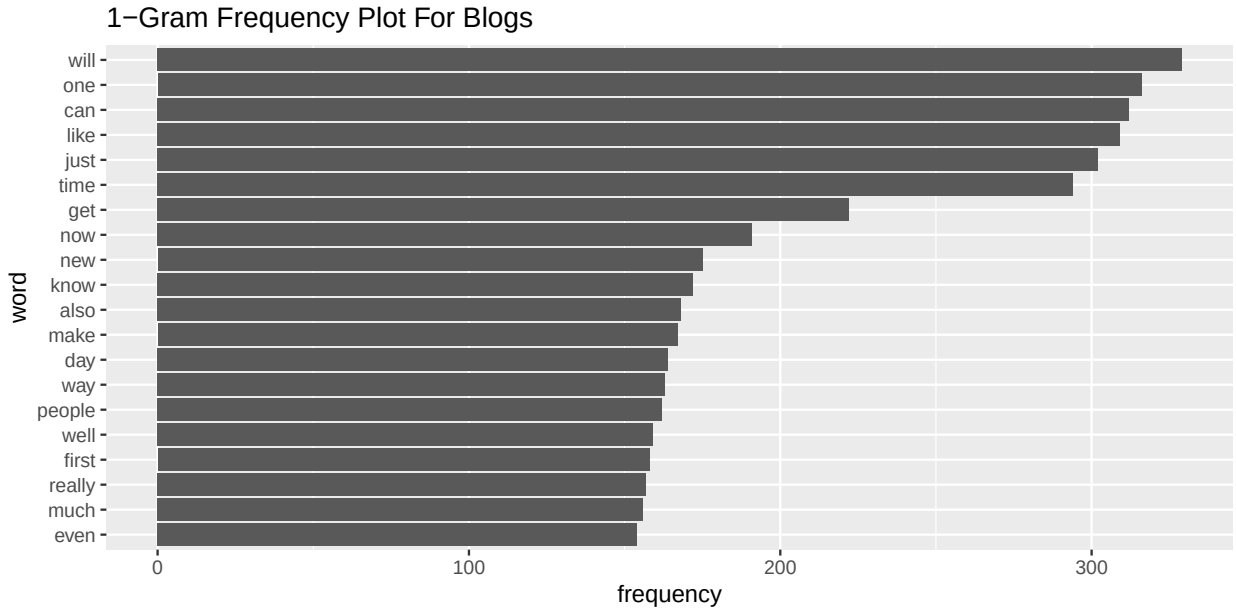

```
g3
```



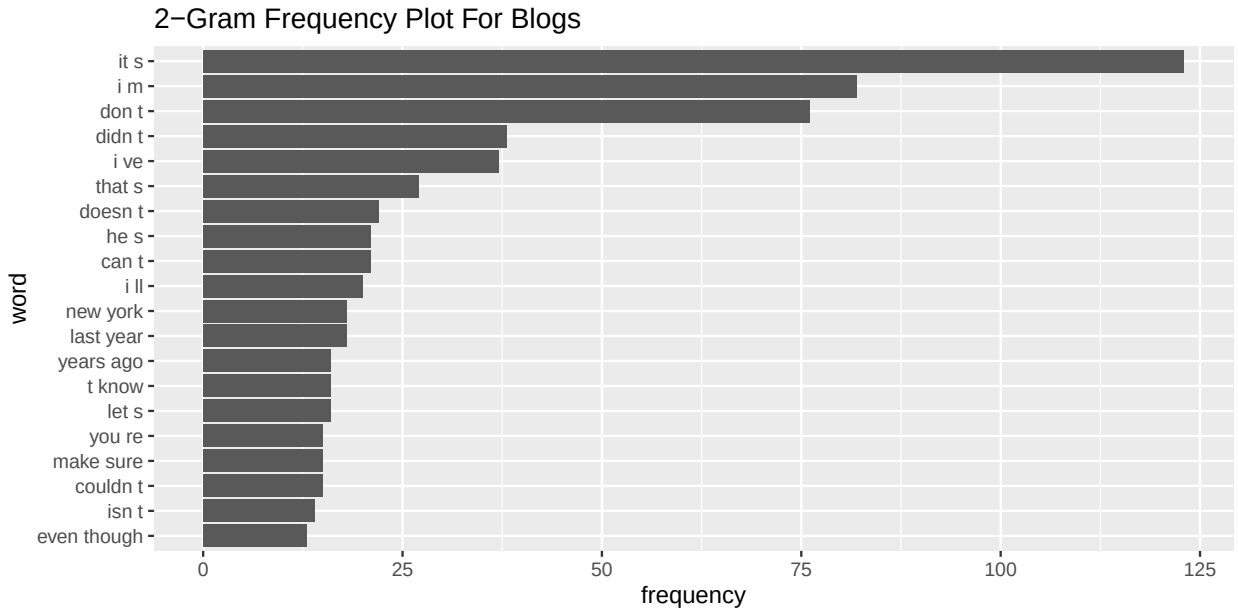
III.B. Blogs

```
file_name <- "en_US.blogs.txt"
blogs_info <- text_processing(sample_lines(file_name, n=blogs_length*0.003, blogs_length), 1)
plotDf <- blogs_info[1:20,]
plotDf$word <- reorder(plotDf$word, plotDf$frequency)
g1 = ggplot(plotDf, aes(x = word, y = frequency)) + geom_bar(stat = "identity") + coord_flip() +
    ggtitle("1-Gram Frequency Plot For Blogs")
```

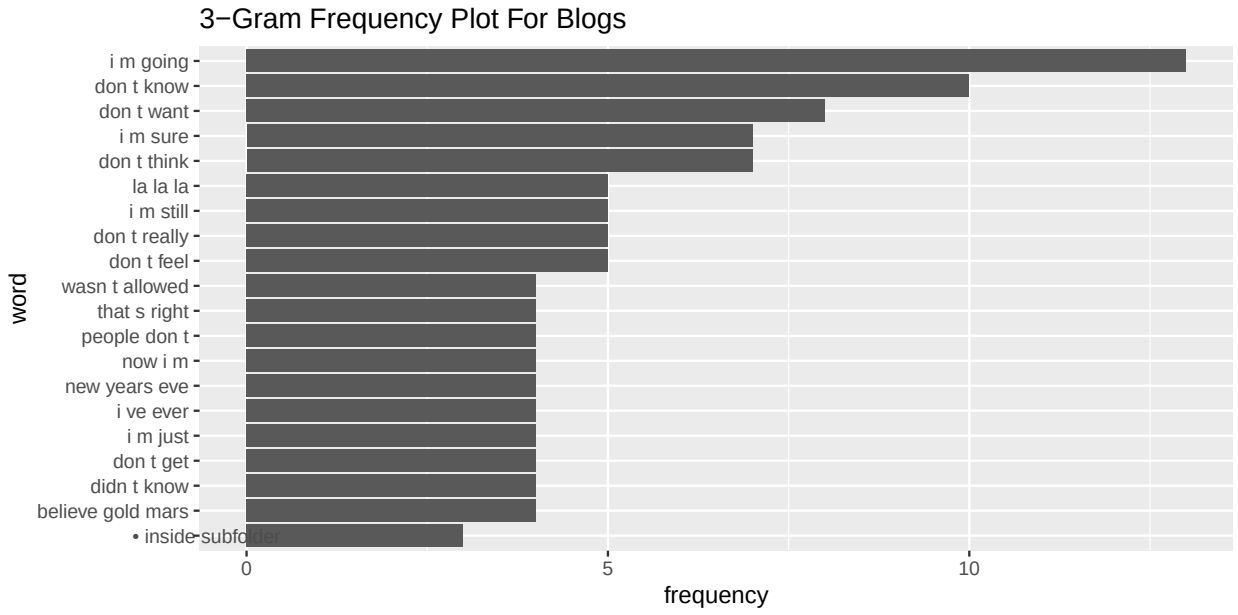
```
blogs_info <- text_processing(sample_lines(file_name, n=blogs_length*0.003, blogs_length), 2)
plotDf <- blogs_info[1:20,]
plotDf$word <- reorder(plotDf$word, plotDf$frequency)
g2 = ggplot(plotDf, aes(x = word, y = frequency)) + geom_bar(stat = "identity") + coord_flip() +
    ggtitle("2-Gram Frequency Plot For Blogs")

blogs_info <- text_processing(sample_lines(file_name, n=blogs_length*0.003, blogs_length), 3)
plotDf <- blogs_info[1:20,]
plotDf$word <- reorder(plotDf$word, plotDf$frequency)
g3 = ggplot(plotDf, aes(x = word, y = frequency)) + geom_bar(stat = "identity") + coord_flip() +
    ggtitle("3-Gram Frequency Plot For Blogs")

g1
```




```
g2
```



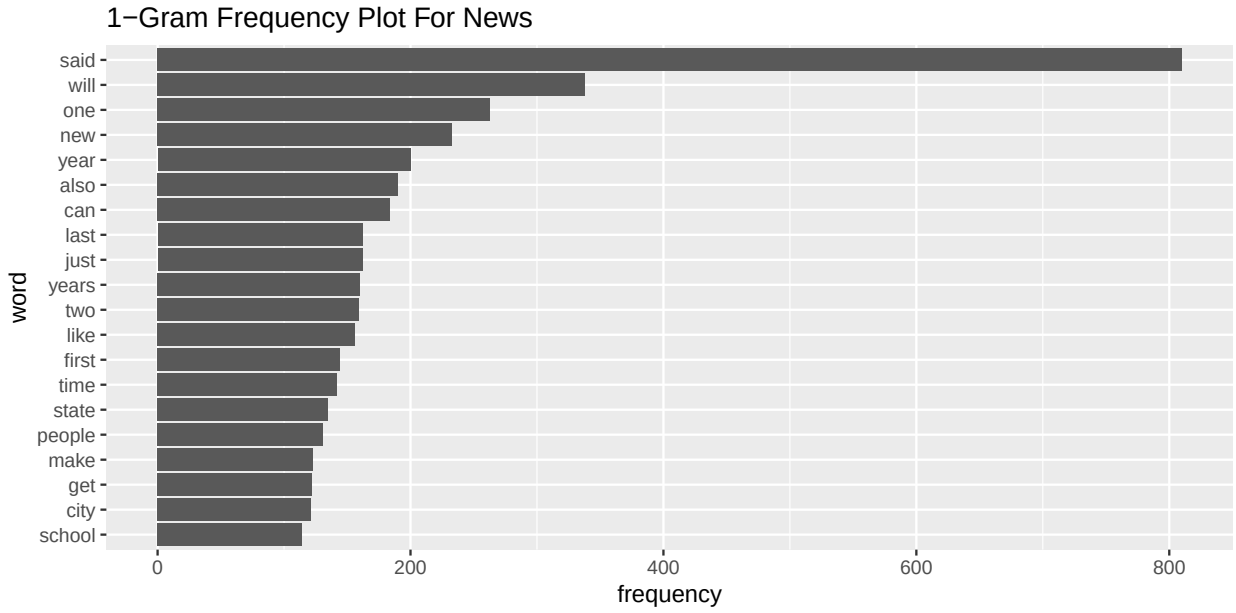

```
g3
```



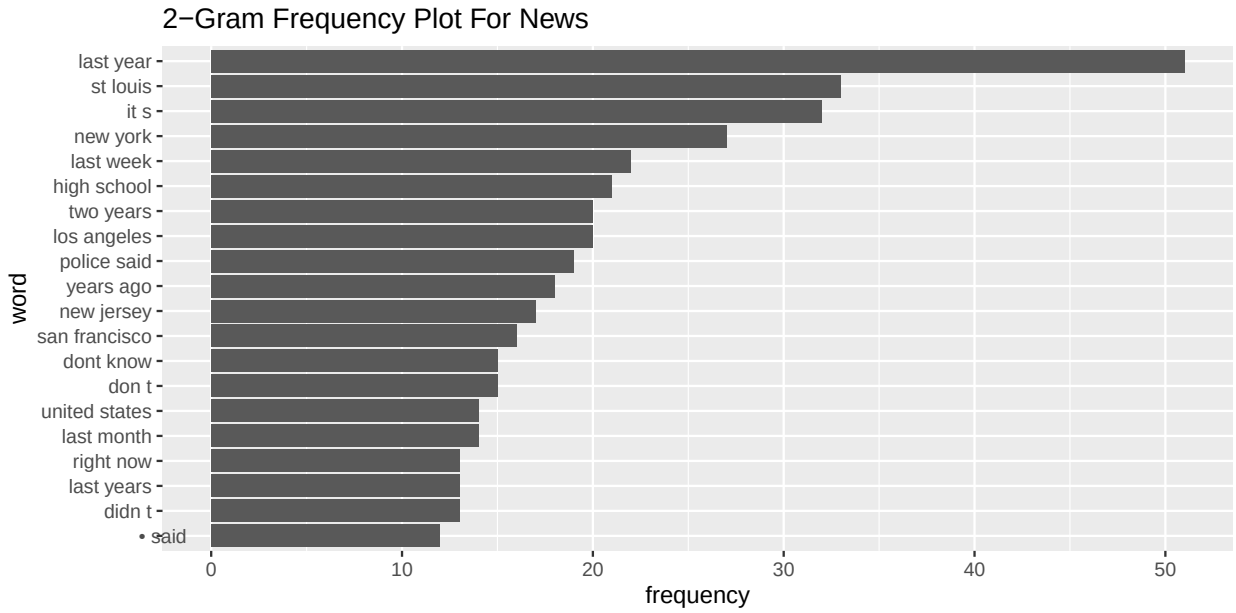
III.C. News

```
file_name <- "en_US.news.txt"
news_info <- text_processing(sample_lines(file_name, n=news_length*0.003, news_length), 1)
plotDf <- news_info[1:20,]
plotDf$word <- reorder(plotDf$word, plotDf$frequency)
g1 = ggplot(plotDf, aes(x = word, y = frequency)) + geom_bar(stat = "identity") + coord_flip() +
    ggtitle("1-Gram Frequency Plot For News")
```

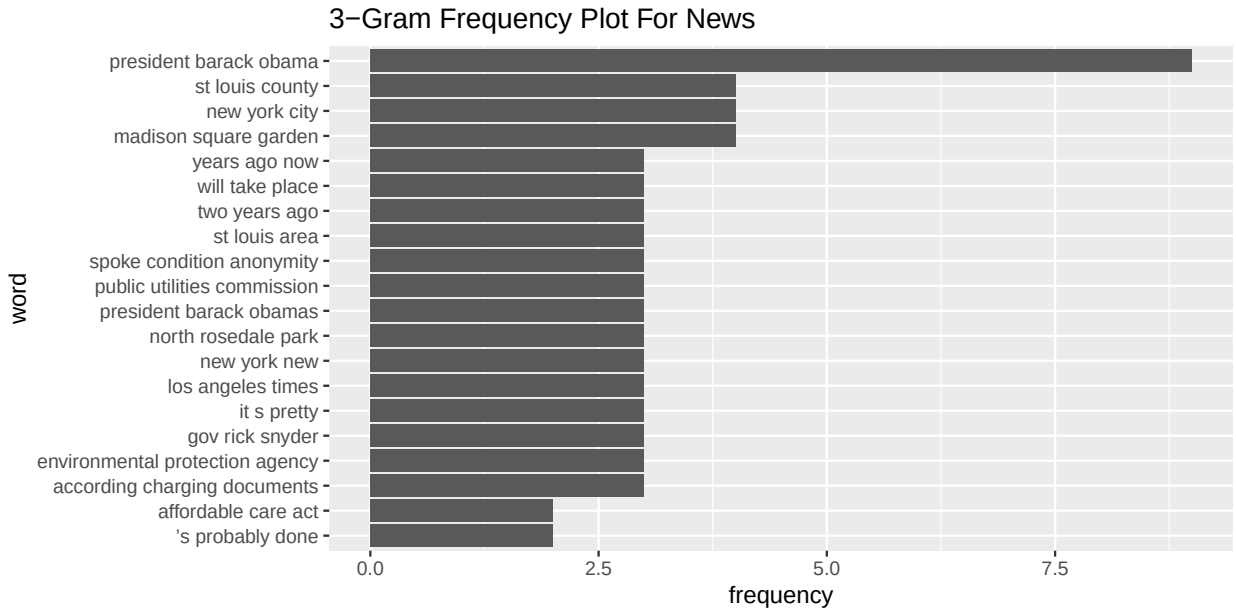
```
news_info <- text_processing(sample_lines(file_name, n=news_length*0.003, news_length), 2)
plotDf <- news_info[1:20,]
plotDf$word <- reorder(plotDf$word, plotDf$frequency)
g2 = ggplot(plotDf, aes(x = word, y = frequency)) + geom_bar(stat = "identity") + coord_flip() +
    ggtitle("2-Gram Frequency Plot For News")

news_info <- text_processing(sample_lines(file_name, n=news_length*0.003, news_length), 3)
plotDf <- news_info[1:20,]
plotDf$word <- reorder(plotDf$word, plotDf$frequency)
g3 = ggplot(plotDf, aes(x = word, y = frequency)) + geom_bar(stat = "identity") + coord_flip() +
    ggtitle("3-Gram Frequency Plot For News")

g1
```




```
g2
```


2−Gram Frequency Plot For News
last year
st louis
it s
new york
last week
high school
two years
los angeles
police said
years ago
new jersey
san francisco
dont know
don t
united states
last month
right now
last years
didn t
• said
word
0
10
20
30
40
50
frequency

```
g3
```


3−Gram Frequency Plot For News
president barack obama
st louis county
new york city
madison square garden
years ago now
will take place
two years ago
st louis area
spoke condition anonymity
public utilities commission
president barack obamas
north rosedale park
new york new
los angeles times
it s pretty
gov rick snyder
environmental protection agency
according charging documents
affordable care act
's probably done
word
0.0
2.5
5.0
7.5
frequency

IV. Conclusion

In this paper, we have seen that we are based on big sized datasets, with more than 2 million lines in the most extreme case. We used random sampling of 0.3% of the whole dataset due to memory limitations, but we will use the bootstrapping methodology when predicting the next word.
Now that we have understood our base datasets, the next step will be to build the word-prediction model before deploying it on a Shiny server.