Data Science Capstone Project - Milestone Project

By John Maged

Introduction

This milestone report will be applying data science in the area of natural language processing (NLP).

Below, we will do the following:

- Data Loading,
- Data Cleaning,
- Exploratory Analysis, and
- Conclusion.

Let us start now by data extraction.

Data Loading

The data set consists of three files in US English.

- en_US.blogs.txt
- en_US.news.txt
- en_US.twitter.txt

Loading The Dataset

```
con <- file("en_US.blogs.txt", open="rb")
blogs <- readLines(con, encoding="UTF-8", skipNul=TRUE)
close(con)
rm(con)

con <- file("en_US.twitter.txt", open="rb")
twitter <- readLines(con, encoding="UTF-8", skipNul=TRUE)
close(con)
rm(con)

con <- file("en_US.news.txt", open="rb")
news <- readLines(con, encoding="UTF-8", skipNul=TRUE)
close(con)
rm(con)
```

Aggreagating A Data Sample

In order to enable faster data processing, a data sample from all three sources was generated.

```
sampleTwitter <- twitter[sample(1:length(twitter),5000)]
sampleNews <- news[sample(1:length(news),5000)]
sampleBlogs <- blogs[sample(1:length(blogs),5000)]
textSample <- c(sampleTwitter,sampleNews,sampleBlogs)

## Saving sample..
writeLines(textSample, "textSample.txt")

## Reading from the sample file..
theSampleCon <- file("textSample2.txt")
theSample <- readLines(theSampleCon)
close(theSampleCon)
```

Summary Statistics

```
# Checking the size and length of the files and calculate the word count
blogsFile <- file.info("en_US.blogs.txt")$size
newsFile <- file.info("en_US.news.txt")$size
twitterFile <- file.info("en_US.twitter.txt")$size
sampleFile <- file.info("textSample.txt")$size

# Line counts of different files
blogsLength <- length(blogs)
newsLength <- length(news)
twitterLength <- length(twitter)
sampleLength <- length(textSample)

# Word counts
blogsWords <- sum(sapply(gregexpr("\\S+", blogs), length))
newsWords <- sum(sapply(gregexpr("\\S+", news), length))
twitterWords <- sum(sapply(gregexpr("\\S+", twitter), length))
sampleWords <- sum(sapply(gregexpr("\\S+", textSample), length))
```

Building the data frame.

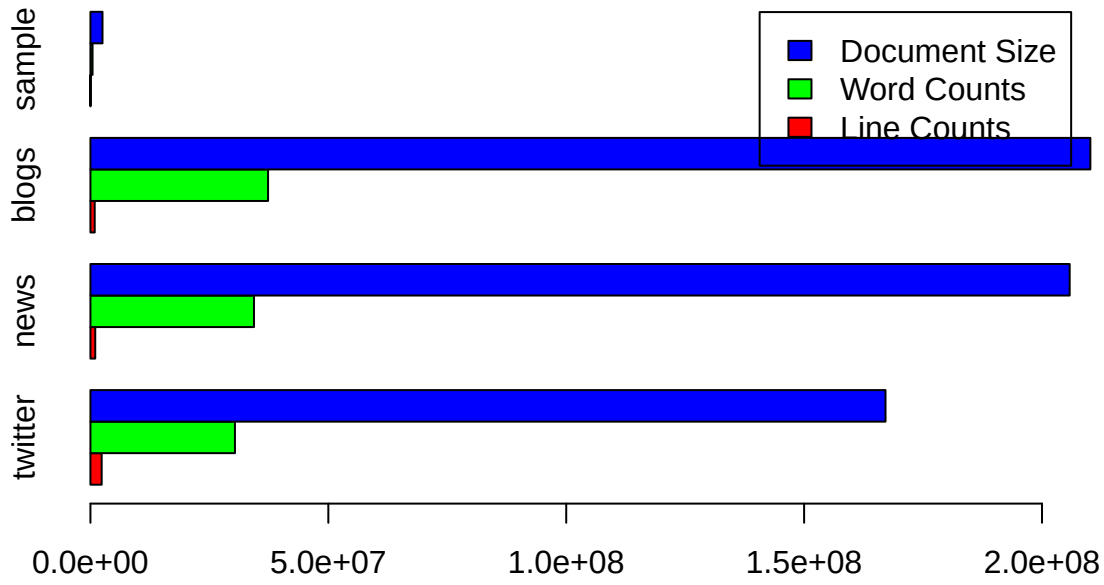
```
# Vectors of line count, word count, and document size for each file.
lineCounts <- c(twitterLength, newsLength, blogsLength, sampleLength)
names(lineCounts)<-c("twitter","news","blogs","sample")
wordCounts <- c(twitterWords, newsWords, blogsWords, sampleWords)
names(wordCounts)<-c("twitter","news","blogs","sample")
sizes<- c(twitterFile,newsFile,blogsFile,sampleFile)
names(sizes)<-c("twitter","news","blogs","sample")

# Building the data frame..
summary.df <- as.data.frame(cbind(lineCounts,wordCounts,sizes))
rownames(summary.df)<-c("twitter","news","blogs","sample")
colnames(summary.df)<-c("Line Counts", "Word Counts", "Document Size")
```

The following table provides an overview of the imported data. You will find the number of lines in each file including the three files plus the sample file. In addition, you will find the number of words and size of each file.

```
summary.df
```

```
##         Line Counts Word Counts Document Size
## twitter     2360148    30373583     167105338
## news        1010242    34372530     205811889
## blogs        899288    37334131     210160014
## sample        15000      441568       2534423
```

Here are a plotting of the above summary:



Building a clean corpus

Using the tm package, corpus is cleaned - for example, text data is converted into lower case, further punction, numbers and URLs are getting removed. Next to that stop words are erased from the text sample. At the end we are getting a clean text corpus which enables an easy subsequent processing.

```
theSampleCon <- file("textSample.txt")
textSample <- readLines(theSampleCon)
close(theSampleCon)

## Build the corpus, and specify the source to be character vectors
cleanSample <- Corpus(VectorSource(textSample))
rm(textSample)
```

```
## Make it work with the new tm package
cleanSample <- tm_map(cleanSample,
                      content_transformer(function(x)
                              iconv(x, to="UTF-8", sub="byte")))

## Convert to lower case
cleanSample <- tm_map(cleanSample, content_transformer(tolower), lazy = TRUE)

## remove punction, numbers, URLs, stop, and stemming
cleanSample <- tm_map(cleanSample, content_transformer(removePunctuation))
cleanSample <- tm_map(cleanSample, content_transformer(removeNumbers))
removeURL <- function(x) gsub("http[[:alnum:]]*", "", x)
cleanSample <- tm_map(cleanSample, content_transformer(removeURL))
cleanSample <- tm_map(cleanSample, stripWhitespace)
cleanSample <- tm_map(cleanSample, removeWords, stopwords("english"))
cleanSample <- tm_map(cleanSample, stemDocument)
cleanSample <- tm_map(cleanSample, stripWhitespace)
```

Exploratory analysis

The N-Gram Tokenization

In Natural Language Processing (NLP), an n-gram is a contiguous sequence of n items from a given sequence of text or speech.

The following function is used to extract 1-grams, 2-grams and 2-grams from the cleaned text corpus.

```
## Budilding the n-grams
finalCorpusDF <-data.frame(text=unlist(sapply(cleanSample,`[`, "content")),
                           stringsAsFactors = FALSE)


## Building the tokenization function for the n-grams
ngramTokenizer <- function(theCorpus, ngramCount) {
        ngramFunction <- NGramTokenizer(theCorpus,
                                Weka_control(min = ngramCount, max = ngramCount,
                                delimiters = " \\r\\n\\t.,;:\"()?!"))
        ngramFunction <- data.frame(table(ngramFunction))
        ngramFunction <- ngramFunction[order(ngramFunction$Freq,
                                             decreasing = TRUE),][1:10,]
        colnames(ngramFunction) <- c("String","Count")
        ngramFunction
}
```

By the usage of the tokenizer function for the n-grams a distribution of the following top 10 words and word combinations can be inspected. Unigrams are single words, while bigrams are two word combinations and trigrams are three word combinations.

Top Unigrams

```
##       String Count
## 20424   said  1456
```

```
## 26236   will  1396
## 16922    one  1361
## 13788   like  1209
## 23910   time  1127
## 9654     get  1103
## 12680   just  1066
## 9848      go  1047
## 26703   year  1041
## 3819     can   994
```

Top Bigrams

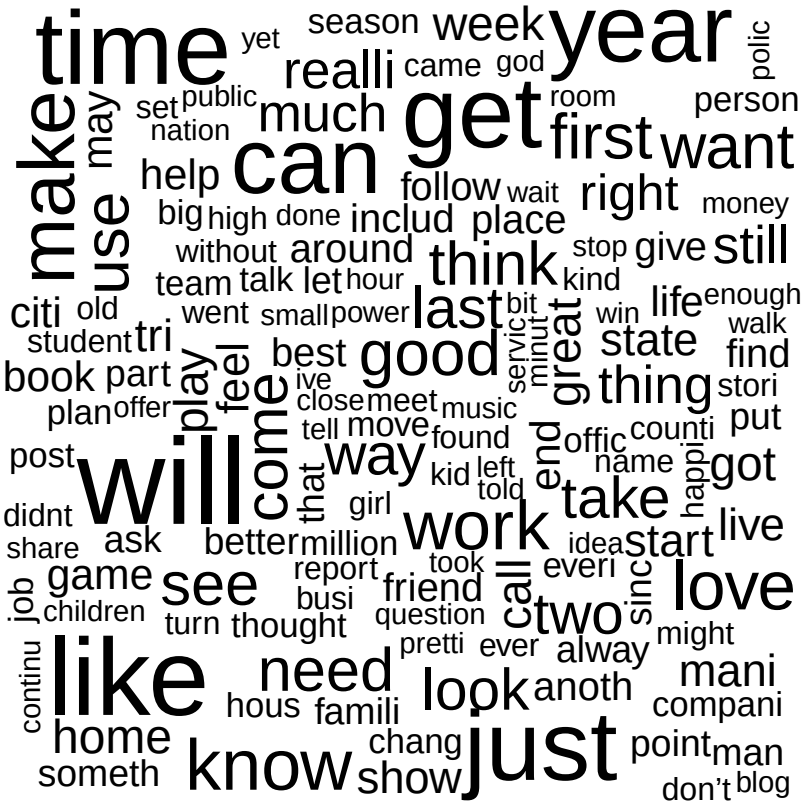
```
##             String Count
## 99991    last year   107
## 153807   right now    93
## 123060    new york    91
## 208083    year ago    77
## 83826  high school    70
## 106968   look like    70
## 64273    feel like    64
## 99980    last week    63
## 51798    dont know    58
## 66717   first time    56
```

Top Trigrams

```
##                      String Count
## 31520         cant wait see    14
## 141280        new york citi    14
## 92006      happi mother day    11
## 116979          let us know     9
## 162820  presid barack obama     9
## 24943  bootleg edit bootleg     7
## 61436     edit bootleg edit     7
## 177190          rock n roll     7
## 213027    thump thump thump     7
## 238035         world war ii     7
```

Creating a wordcloud

A word cloud usually provides a first overview of the word frequencies. The word cloud displays the data of the aggregated sample file.



Conclusion

- A data sample file was created as the raw data size is big and takes a lot of time in processing. This was very useful when working on this sample on the next steps - data cleaning and exploratory analysis.I will work on how sampling is done in a better way that will not impact the final results.
- In Data Cleaning, I converted all text into lower case and removed any punctuations, numbers, URLs, stop words, and did stemming.
- In exploratory analysis, I did some analysis for N-grams as you saw earlier in this report. Top unigrams, bigrams, and trigrams in the sample data file was represented.

The Following Steps

The next steps in the project will be:

- Enhancing the sampling process on the raw data or find a way enhancing the performance when applying different processing tasks on the raw data itself.
- Building the prediction model that will predict the next word a user wants to write as the SwiftKey applications.