Data Science Capstone Milestone

R. Martinez

February 19, 2019

Introduction

This is Capstone project for the Data Science Specialiazation of the John Hopkins University. The main objective is to build a text model to predict the next words that a user will type on a text device such a mobile text message app.

This document shows the milestone report for week #2 where I will:
- Get the data file to train the text model.
- Perform data exploration to undestand the data.
- Profile the data file to identify basic characteriscti of the data.
- Provide a inital plan to build the prediction text model.

.

.

For detail about the code, here is the link to the GitHub repo: https://github.com/graulm/Data-Science-Capstone .

.

Library and Data

The data was provided by the professor and can be downloaded from this source: https://d396qusza40orc.cloudfront.net/dsscapstone/dataset/Coursera-SwiftKey.zip

```
library(tm);    library(SnowballC); library(wordcloud);   library(dplyr)
```

```
Loading required package: NLP

Loading required package: RColorBrewer


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(RWeka); library(ggplot2);   library(formattable); library(tokenizers)
```

```

Attaching package: 'ggplot2'

The following object is masked from 'package:NLP':

    annotate
```

```
set.seed(201902)
```

```
# Note that due the size of the data files (200MG, 196MG, and 159MG)
# I will use smaller size of the files selecting lines randomly (using rbinom)
#file_path  <- "C:/Users/raulm/Documents/John Hopkins University/#10 Data Science Capstone/Coursera-Swi
```

```r
file_path  <- "C:/Users/rlmartinez/Documents/John Hopkins University/#10 Data Science Capstone/Coursera
file_blogs <- "en_US.blogs.txt"
file_news  <- "en_US.news.txt"
file_twitter <- "en_US.twitter.txt"

# Load the "blogs" data file
incon <- file(paste(file_path, file_blogs ,sep=""),"r")
file <- readLines(incon, encoding="UTF-8", skipNul=TRUE)
data_blogs <- file[rbinom(length(file), 1, 0.01) == 1]
close(incon)
len_blogs <- length(file)
size_blogs <- object.size(file)
max_line_blogs <- max(nchar(file))
avg_line_blogs <- mean(nchar(file))
len_blogs_sample <- length(data_blogs)

# Load the "news" data file
incon <- file(paste(file_path, file_news ,sep=""),"rb")
file <- readLines(incon, encoding="UTF-8", skipNul=TRUE)
data_news <- file[rbinom(length(file), 1, 0.01) == 1]
close(incon)
len_news <- length(file)
size_news <- object.size(file)
max_line_news <- max(nchar(file))
avg_line_news <- mean(nchar(file))
len_news_sample <- length(data_news)

# Load the "twitter" data file
incon <- file(paste(file_path, file_twitter ,sep=""),"r")
file <- readLines(incon, encoding="UTF-8", skipNul=TRUE)
data_twitter <- file[rbinom(length(file), 1, 0.005) == 1]
close(incon)
len_twitter <- length(file)
size_twitter <- object.size(file)
max_line_twitter <- max(nchar(file))
avg_line_twitter <- mean(nchar(file))
len_twitter_sample <- length(data_twitter)

# Now remove the temp file to reselase memory
rm(file)
```

Once the data has been loaded, let's show basic metrics about the data files:

```r
# Basic metrics about the files
data_metrics <- data.frame(file_name = c("en_US.blogs.txt","en_US.news.txt","en_US.twitter.txt"),
                         size = c(format(size_blogs, units = "auto"),
                                  format(size_news, units = "auto"),
                                  format(size_twitter, units = "auto")),
                         lines = c(format(len_blogs, big.mark=","),
                                   format(len_news, big.mark=","),
                                   format(len_twitter, big.mark=",")),
                         Average_line_length = c(round(avg_line_blogs,0),
                                                 round(avg_line_news,0),
                                                 round(avg_line_twitter,0)),
                             max_line_length = c(format(max_line_blogs, big.mark=","),
                                                 format(max_line_news, big.mark=","),
                                                 format(max_line_twitter, big.mark=","))
                             )
# summary table
colnames(data_metrics) <- c('File Name', 'File Size', 'Number of Lines', 'Average Length of Lines', 'Ma
formattable(data_metrics)
```

File Name

File Size

Number of Lines

Average Length of Lines

Maimun Length of a line

en_US.blogs.txt

255.4 Mb

899,288

230

40,833

en_US.news.txt

257.3 Mb

1,010,242

201

11,384

en_US.twitter.txt

319 Mb

2,360,148

69

140

Data Transformation

On this section I will prepare the data to make it ready for the prediction text model.
To do that:

- Ensure the data is all ASCII.
- The text will be all in lowecase.
- Digits, puntuation, stopwords, white spaces will be removed.
- And the text will be transfomred to its stem form.

```
# Make sure all characters are ASCII
data_blogs <- iconv(data_blogs,  to="ASCII", sub="")
data_news <- iconv(data_news,  to="ASCII", sub="")
data_twitter <- iconv(data_twitter,  to="ASCII", sub="")

# Create the corpus to use "tm" packages
my_corpus <- VCorpus(VectorSource(paste(data_blogs, data_news, data_twitter)))
```

```r
# Remove original doc to release memory
rm(data_blogs, data_news, data_twitter)

# Transformation using "tm" functions
my_corpus <- tm_map(my_corpus, removeNumbers)
my_corpus <- tm_map(my_corpus, tolower)
my_corpus <- tm_map(my_corpus, removePunctuation)
my_corpus <- tm_map(my_corpus, removeWords, stopwords("english"))
my_corpus <- tm_map(my_corpus, stripWhitespace)
my_corpus <- tm_map(my_corpus, stemDocument)
my_corpus <- tm_map(my_corpus, PlainTextDocument)

# Just to inspect the corpus
inspect(my_corpus[1])
```

```
<<VCorpus>>
Metadata:  corpus specific: 0, document level (indexed): 0
Content:  documents: 1

[[1]]
<<PlainTextDocument>>
Metadata:  7
Content:  chars: 186
```

```r
writeLines(as.character(my_corpus[1]))
```

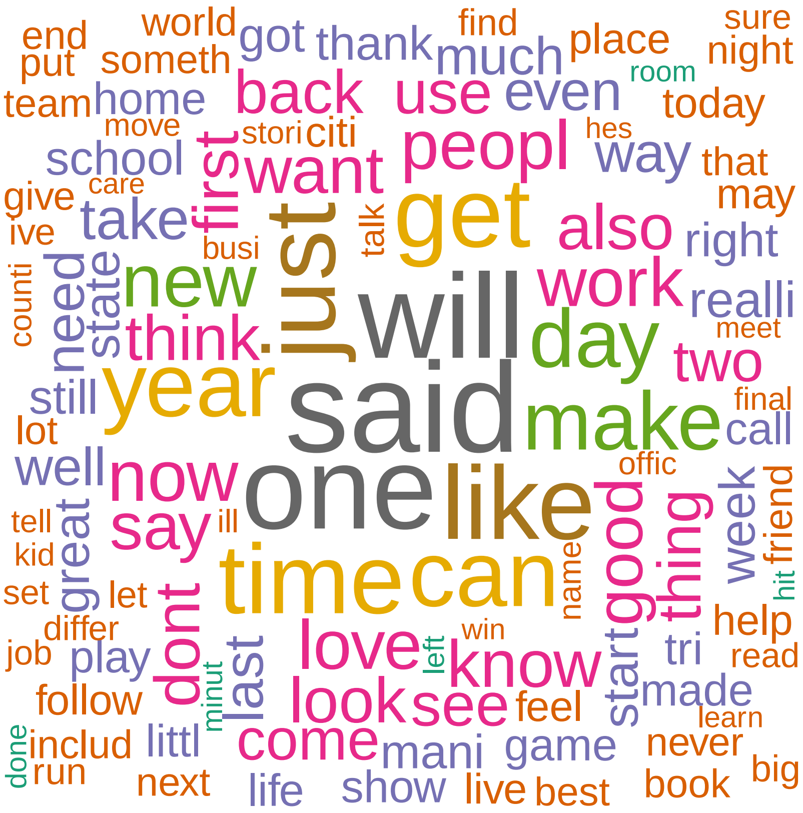
```
list(list(content = "scene shay apart saturday market park lot logjam driver circl ignor arrow paint pa
list()
list()
```

Data Exploration

On this section, I will show some data exploration made to the data that was all transfomed (previous section) into a **Corpus**.

Let's start showing a **words cloud** with the top 200 most frequent words:

```r
wordcloud(my_corpus, max.words = 200, random.order = FALSE, colors=brewer.pal(8,"Dark2"))
```



Now let's build the Ngrams tokenization to determine the most frequent uni-grams, bi-brams and three-grams:

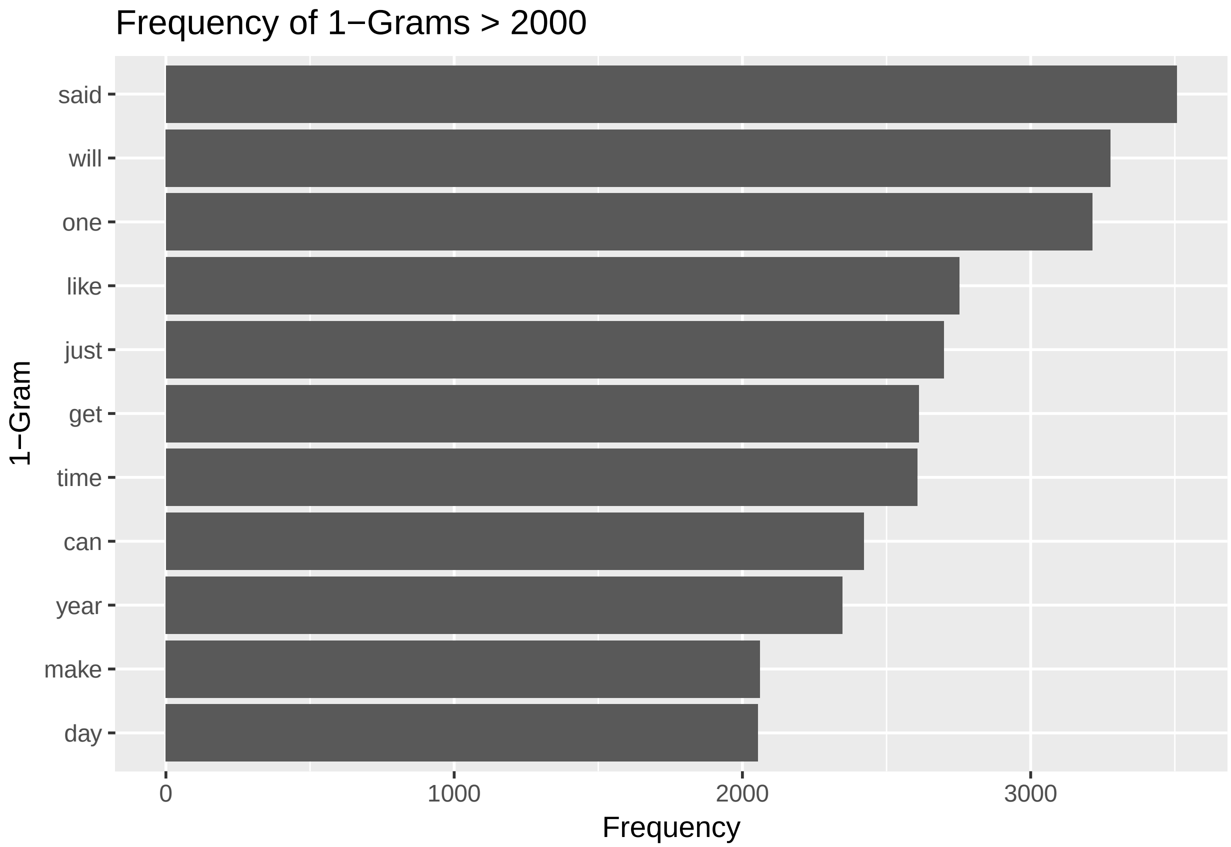
Unigrams Tokenization:

```
one_gramTokenizer <- function(x) NGramTokenizer(x=x, control=Weka_control(min = 1, max = 1))
one_dtm <- DocumentTermMatrix(my_corpus, control = list(tokenize = one_gramTokenizer))
# Because the matrix is too much sparse, let's find something that is not so much
one_dtm_sparse <- removeSparseTerms(one_dtm, sparse=0.99)

one_dtm_freq <- sort(colSums(as.matrix(one_dtm_sparse)),decreasing = TRUE)
one_dtm_freq_df <- data.frame(word = names(one_dtm_freq), frequency = one_dtm_freq)
head(one_dtm_freq_df, 10)
```

```
     word frequency
said said      3507
will will      3277
one   one      3213
like like      2752
just just      2699
get   get      2612
time time      2606
can   can      2421
year year      2347
make make      2061
```

```
one_dtm_plot <- subset(one_dtm_freq_df, frequency > 2000)
ggplot(one_dtm_plot, aes(x=reorder(word, frequency), y=frequency)) +
       geom_bar(stat = "identity") +  coord_flip() +
       theme(legend.title=element_blank()) +
       xlab("1-Gram") + ylab("Frequency") +
       labs(title = "Frequency of 1-Grams > 2000")
```



Bigrams Tokenization:

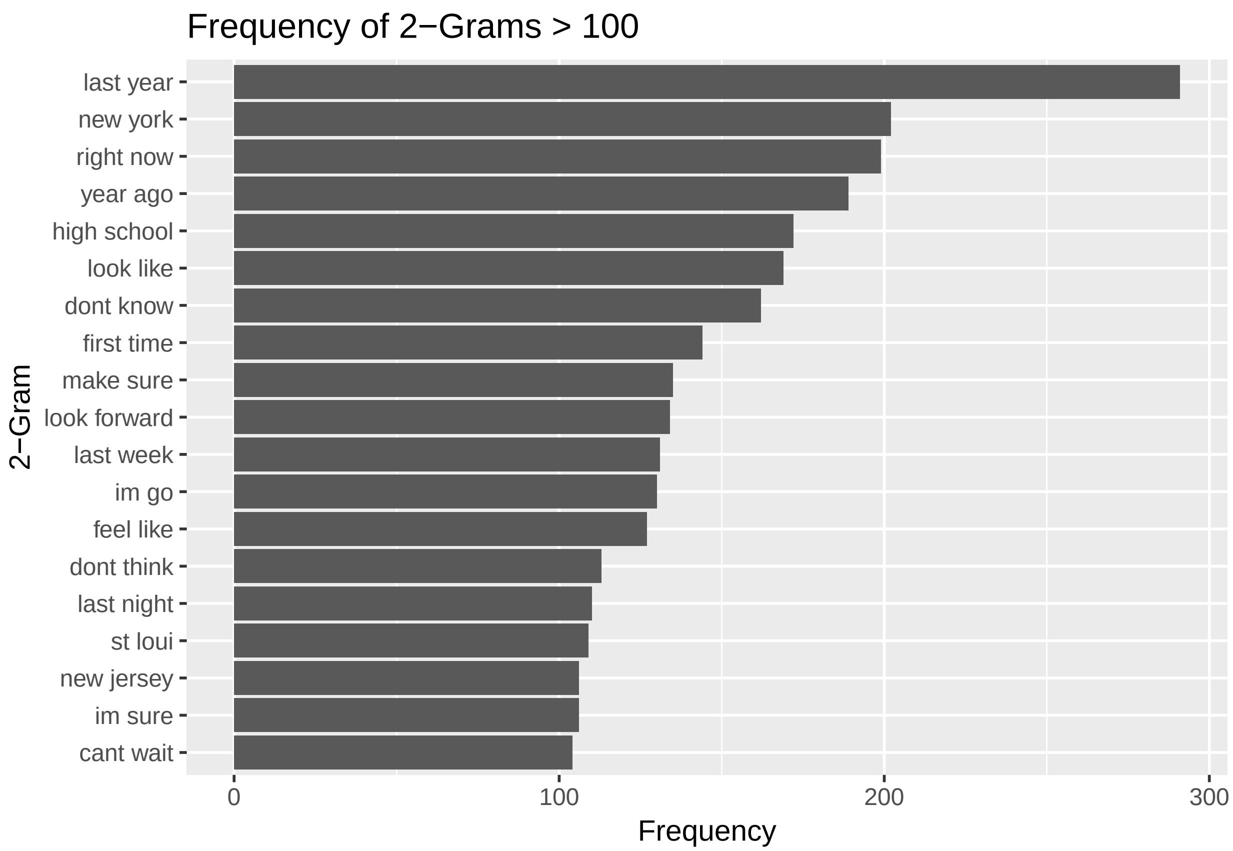
```
two_gramTokenizer <- function(x) NGramTokenizer(x=x, control=Weka_control(min = 2, max = 2))
two_dtm <- DocumentTermMatrix(my_corpus, control = list(tokenize = two_gramTokenizer))
# Because the matrix is too much sparse, let's find something that is not so much
two_dtm_sparse <- removeSparseTerms(two_dtm, sparse=0.9997)

two_dtm_freq <- sort(colSums(as.matrix(two_dtm_sparse)),decreasing = TRUE)
two_dtm_freq_df <- data.frame(word = names(two_dtm_freq), frequency = two_dtm_freq)
head(two_dtm_freq_df, 10)
```

```
                   word frequency
last year     last year       291
new york       new york       202
right now     right now       199
year ago       year ago       189
high school high school       172
look like     look like       169
dont know     dont know       162
first time   first time       144
make sure     make sure       135
```

```
look forward look forward       134
```

```
two_dtm_plot <- subset(two_dtm_freq_df, frequency > 100)
ggplot(two_dtm_plot, aes(x=reorder(word, frequency), y=frequency)) +
        geom_bar(stat = "identity") +  coord_flip() +
        theme(legend.title=element_blank()) +
        xlab("2-Gram") + ylab("Frequency") +
        labs(title = "Frequency of 2-Grams > 100")
```



Threegrams Tokenization:

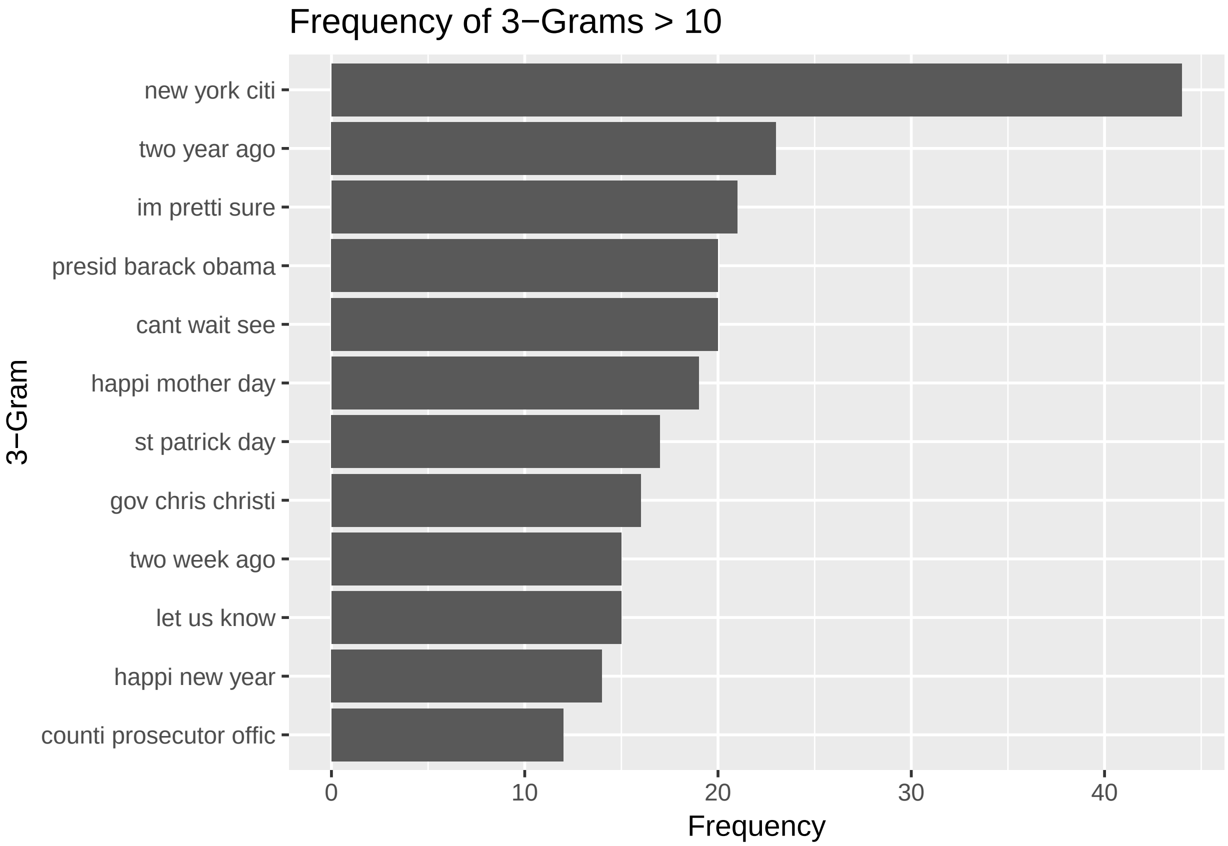
```
three_gramTokenizer <- function(x) NGramTokenizer(x=x, control=Weka_control(min = 3, max = 3))
three_dtm <- DocumentTermMatrix(my_corpus, control = list(tokenize = three_gramTokenizer))
# Because the matrix is too much sparse, let's find something that is not so much
three_dtm_sparse <- removeSparseTerms(three_dtm, sparse=0.9998)

three_dtm_freq <- sort(colSums(as.matrix(three_dtm_sparse)),decreasing = TRUE)
three_dtm_freq_df <- data.frame(word = names(three_dtm_freq), frequency = three_dtm_freq)
head(three_dtm_freq_df, 10)
```

```
                                   word frequency
new york citi                new york citi        44
two year ago                  two year ago        23
im pretti sure              im pretti sure        21
cant wait see                cant wait see        20
presid barack obama presid barack obama        20
happi mother day          happi mother day        19
```

```
st patrick day           st patrick day        17
gov chris christi      gov chris christi       16
let us know                  let us know       15
two week ago                two week ago       15
```

```
three_dtm_plot <- subset(three_dtm_freq_df, frequency > 10)
ggplot(three_dtm_plot, aes(x=reorder(word, frequency), y=frequency)) +
        geom_bar(stat = "identity") +  coord_flip() +
        theme(legend.title=element_blank()) +
        xlab("3-Gram") + ylab("Frequency") +
        labs(title = "Frequency of 3-Grams > 10")
```



Calcualte of the size of the data-file to be used in the text predition model

This section calculate the size fo the object that hold the ngrams that will be used for the model. Unfortunately there are RAM constraints due the small side of the computer where I'm buiding this project. And based on the guideline of the project, we should assume we will have some RAM constraints on the devide where the model will be running on. So, for those reasons, I'm showing here the size of the ngrams object with some percentages of the corpus:

```
# Basic metrics for the bigrams dataframe
bigrams_metrics <- data.frame(file_name = c("Bigrams 50%","Bigrams 75%","Bigrams 90%","Bigrams 100%"),
  size = c(format(object.size(two_dtm_freq_df[1:round(nrow(two_dtm_freq_df)*0.5),]), units="auto"),
           format(object.size(two_dtm_freq_df[1:round(nrow(two_dtm_freq_df)*0.75),]),units="auto"),
           format(object.size(two_dtm_freq_df[1:round(nrow(two_dtm_freq_df)*0.90),]),units="auto"),
           format(object.size(two_dtm_freq_df[1:round(nrow(two_dtm_freq_df)*1.0),]), units="auto")
          )
```

```
                                )

# summary table
colnames(bigrams_metrics) <- c('Bigrams Data', 'File Size')
formattable(bigrams_metrics)
```

Bigrams Data

File Size

Bigrams 50%

1.3 Mb

Bigrams 75%

1.5 Mb

Bigrams 90%

1.7 Mb

Bigrams 100%

1.8 Mb

Strategy to build the Text prediction Model

Now that I have clear understanding of the data, I have trasnformed it into a clean Corpus, generated the ngrams, and calculated the pontial size of the object to be used in the model, follows is the first draft of the strategy on how the text predtion model will work:

1- Build the Bigrams with a Corpus Tranformed (removed whitespace, stemming, removed stopwords, etc.).
2- For quick access build a dictionary with the Bigrams.
3- Based on the size contraints, use at least 90% coverage of the Bigrams object.
4- When the user enters a word, try to find the word in the dictionary.
5- If the word exists, suggest the word that follows based on the Bigrams dataframe.
6- If the word does not exist, suggest a word that exists in the Bigrams dataframe selected randomly.

.
.
.
.