Milestone_Report

Jordan Woloschuk

1/27/2020

Data Science Capstone Project: Milestone Report

Project Overview

This this first interm report of the **Data Science Capstone Project** that involves the analysis of multiple, large text files to determine the the structure of the data and how common words are put together to develop a "predictive word model".

The goal of this report is just to display that we have worked with the data and that we are making progress on creating a prediction algorithm. This report will be published on R Pubs and will explain the exploratory analysis and next step goals.

Tasks to accomplish

1. **Exploratory analysis** - perform a thorough exploratory analysis of the data, understanding the distribution of words and relationship between the words in the corpora.
2. **Understand frequencies of words and word pairs** - build figures and tables to understand variation in the frequencies of words and word pairs in the data.

Setup

The following steps are necessary to begin analysis of the text files:

1. Set seed

We set the seed to ensure that the samples pulled from the data files can be reproduced.

```
# Set seed to 1
set.seed("1")
# set working directory
setwd("~/GitHub/Data_Science_Capstone/")
```

2.Load libraries

```
library(tm) # for text mining functions
library(RWeka) # for NGramtokenizer
library(SnowballC) # for Wordstem function
library(ggplot2)
library(dplyr)
library(stringi)
library(grid)
library(knitr)
```

Exploratory Analysis

The dataset useded for this project was downloaded from the following site: (https://d396qusza40orc.cloudfront.net/dsscapstone/dataset/Coursera-SwiftKey.zip)

After downloading the file, the zip file is uncompressed and the US English files are read.

1. Download the file, set file paths and file names

```
# Download and unzip file
if(!file.exists("dataset.zip")) {
        url <- "https://d396qusza40orc.cloudfront.net/dsscapstone/dataset/Coursera-SwiftKey.zip"
        download(url, dest="dataset.zip", mode="wb")
        zip_path <- "~/GitHub/Data_Science_Capstone/data/"
        unzip ("dataset.zip",exdir = zip_path)
}

# File names
Blogs_File_Name <- "en_US.blogs.txt"
News_File_Name <- "en_US.news.txt"
Twitter_File_Name <- "en_US.twitter.txt"

# File paths to data
Base_File_Path <- "~/GitHub/Data_Science_Capstone/data/"
en_US_Base_Path <- paste0( Base_File_Path, "final/en_US/" )
Blogs_File_Path <- paste0( en_US_Base_Path, Blogs_File_Name )
News_File_Path <- paste0( en_US_Base_Path, News_File_Name )
Twitter_File_Path <- paste0( en_US_Base_Path, Twitter_File_Name )


# Read the files
Blogs_data <- readLines( Blogs_File_Path, encoding = "UTF-8", skipNul = TRUE )
News_data <- readLines( News_File_Path, encoding = "UTF-8", skipNul = TRUE )
Twitter_data <- readLines( Twitter_File_Path, encoding = "UTF-8", skipNul = TRUE )
```

2. Data Summary

The following function is used to build a summary table detailing some basic information about the News, Twitter and Blog documents.

	File Name	Size (MB)	No. of Lines	Number of Characters	Number of Words	Longest Line Length
Twitter	en_US.twitter.txt	159.4	2,360,148	162,096,241	30,451,170	140
News	en_US.news.txt	196.3	77,259	15,639,408	2,651,432	5760
Blogs	en_US.blogs.txt	200.4	899,288	206,824,505	37,570,839	40833

From the data, it can be seen that there are a number of issues with the data that need to be fixed. There is profanity and non-english words/characters. These will all need to be removed from the dataset.

3. Creation of Corpus

We create a corpus To provide a uniform and easy way to access the text of the three documents. We will create and this corpus from the three text sources (Twitter, News, Blogs).

```
# Fucntion to load a single data file
Single_Corpus_Function <- function( filename ) {
  VCorpus( VectorSource( paste(readLines(file( filename )) ,collapse="\n") ),
           list(reader = readPlain) )
}
```

4. Sample data

Here we will pull a 5% sample from each of the text files. The reason we do this is to reduce the total datasize that we are working with and improve data processing.

Sample process based on the length of the dataframes.

```
Sample_Percent <- 0.05

# Function samples the text data based on the desired sample percent.
Sample_Data_Function <- function( df, sp = Sample_Percent ) { sample(df, length(df) * sp )  }

# Call the function for each text source and combine data
Combined_Sample_Data <- c( Sample_Data_Function( Blogs_data ),
                           Sample_Data_Function( News_data ),
                           Sample_Data_Function( Twitter_data ) )

# Save the combined data subset as a txt file
Combined_Sample_Data_File_Name <- paste0( Base_File_Path, "en_US.combined_subset.txt" )

# Check to see if the combined data exists, if so, it removes the file name
if (file.exists(Combined_Sample_Data_File_Name)) file.remove(Combined_Sample_Data_File_Name)
```

```
## [1] TRUE
```

```
Combined_Sample_Data_File <- file(Combined_Sample_Data_File_Name)

writeLines(Combined_Sample_Data, Combined_Sample_Data_File)

close(Combined_Sample_Data_File)

# Create a corpus of the combined three data corpuses
Corpus_Sample_Data <- Single_Corpus_Function( Combined_Sample_Data_File_Name )
```

Processing the full set of text from the three files takes a long time. By taking a sample of data files allows us to understand the most-used words, and start the anaylsis of word frequencies. We will also look at n-gram distribution.

The much smaller **Corpus_Sample_Data** file will be used for further analyis.
It contains 5 percent of each text data file.

Data Preprocessing - Text Mining

As a part of this analysis we have used the R package **tm** for "text mining". This package contains a number of helpful functions for processing text.

1. Load the swear word list

The swear words list was obtain from get a list of profanity this Github page: https://raw.githubusercontent.com/shutterstock/List-of-Dirty-Naughty-Obscene-and-Otherwise-Bad-Words/master/en This list contains more than 700 words and acronyms.

```
Swear_Words_File_Path <- "~/GitHub/Data_Science_Capstone/data/"

if (!file.exists('swearwords.txt')) {
  download.file('https://raw.githubusercontent.com/shutterstock/List-of-Dirty-Naughty-Obscene-and-Otherw
                dest = paste0(Swear_Words_File_Path, 'swearwords.txt'),
                method = 'curl', quiet = T)
  }
Swear_Word_List <- readLines(paste0(Swear_Words_File_Path,'swearwords.txt'))
```

2. Build Corpus Processing Function

The following function processes the Corpus and cleans it up. This function will:

- Convert text to lowercase, which helps create uniformity
- Remove punctuation, white spaces and numbers
- Remove swear words and other profanity
- Remove non-english words/characters
- Remove stopwords
- Stems the data (reducing inflection so argued/arguing becomes argu)

```
Corpus_Process_Function <- function( Corpus_Data ) {

  # Basic helper functions - for later use
  Remove_Stopwords_Function <- function(x) removeWords( x, stopwords("english") )
  Remove_Swear_Words_Function <- function(x) removeWords( x, Swear_Word_List )
  Remove_NonEnglish_Function <- function(x) iconv(x, "UTF-8", "ASCII", sub="")


  # Convert to lowercase
  Corpus_Data <- tm_map( Corpus_Data, tolower )

  # punctuation, white spaces and numbers
  Corpus_Data <- tm_map( Corpus_Data, removePunctuation )
  Corpus_Data <- tm_map( Corpus_Data, stripWhitespace )
  Corpus_Data <- tm_map( Corpus_Data, removeNumbers )

  # Remove non-english words, swear words and common "stopwords"
  Corpus_Data <- tm_map( Corpus_Data, function(x) iconv(x, "UTF-8", "ASCII", sub="") )
  Corpus_Data <- tm_map( Corpus_Data, removeWords, Swear_Word_List )
  Corpus_Data <- tm_map( Corpus_Data, removeWords, stopwords("english") )
```

```
  # Stem the data - unsure if this step should be kept
  Corpus_Data <- tm_map( Corpus_Data, stemDocument )

  # Convert data to plain text
  Corpus_Data <- tm_map( Corpus_Data, PlainTextDocument )


  # Return cleaned corpus data
  Corpus_Data
}
```

3. Process Corpus data

```
# Call the Corpus data processng function with the sampled data
Corpus_Processed_Data <- Corpus_Process_Function( Corpus_Sample_Data )
```

4. Tokenization of Processed Corpus

We will now tokenize the corpus data in order to produce the **n-grams**. A n-grams is a sequence of "n" items pulled from our Corpus.

- For n = 1, this is called a unigram
- For n = 2, this is called a bigram
- For n = 3, this is called a trigram

5. Term Matrix Processing

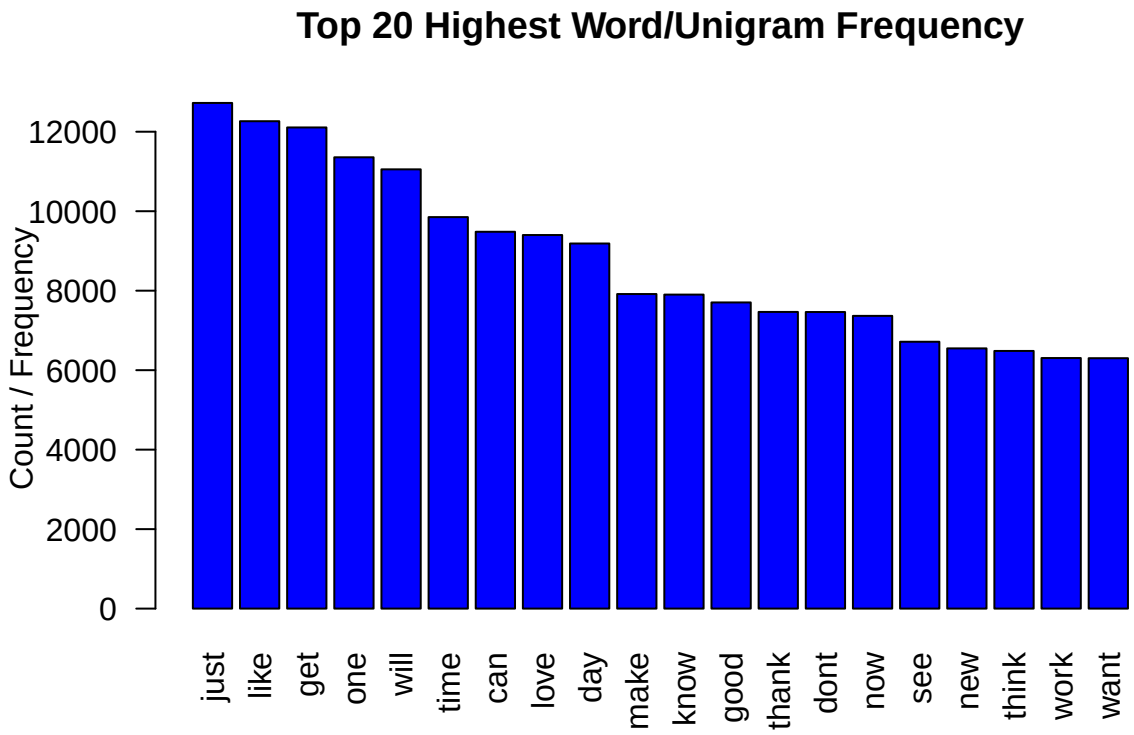
The following produces a matrix that shows the frequency of terms that occurs in Corpus of the Twitter, News and Blog files.

```
# Call the TM TermDocumentMatrix function to construct a term-document matrix
Corpus_Data_TDM  <- TermDocumentMatrix( Corpus_Processed_Data )

# Convert the TDM into a matrixx format
Corpus_Data_TDMx <- as.matrix( Corpus_Data_TDM )

# Calculate the frequency for the terms in the Corpus
Corpus_Frequency <- rowSums(Corpus_Data_TDMx)

#Sort the frequency list from highest to lowest - show only the top 100
Corpus_Frequency <- sort(Corpus_Frequency, decreasing = TRUE)[1:100]

head(Corpus_Frequency)
```

```
##  just  like   get   one  will  time
## 12724 12264 12107 11356 11053  9851
```

6. Top Frequent Single Terms (Unigrams) from the Corpus Dataset:

```
# Plot of the top 20 unigrams
barplot(head(Corpus_Frequency,20),main="Top 20 Highest Word/Unigram Frequency",
        ylab = "Count / Frequency", col = "blue", las = 2)
```



7. N-Gram Processing

Using the cleaned corpus data, we first develop a number of fuctions to determine the number of n-grams.

```
# Frequency function - determines the frequency
Frequency_Function <- function(TDM){
  freq <- sort(rowSums(as.matrix(TDM)), decreasing=TRUE)
  Frequency_Function <- data.frame(word=names(freq), freq=freq)
  return(Frequency_Function)
}

# Determines bigrarms = 2 words combos
Bigram_Tokenizer_Function <-
  function(x)
    unlist(lapply(ngrams(words(x), 2), paste, collapse = " "), use.names = FALSE)

# Determines trgrarms = 3 words combos
Trigram_Tokenizer_Function <-
```

```
  function(x)
    unlist(lapply(ngrams(words(x), 3), paste, collapse = " "), use.names = FALSE)
```

Using these functions, we determine the number of bigrams and trigrams contained in the corpus data.

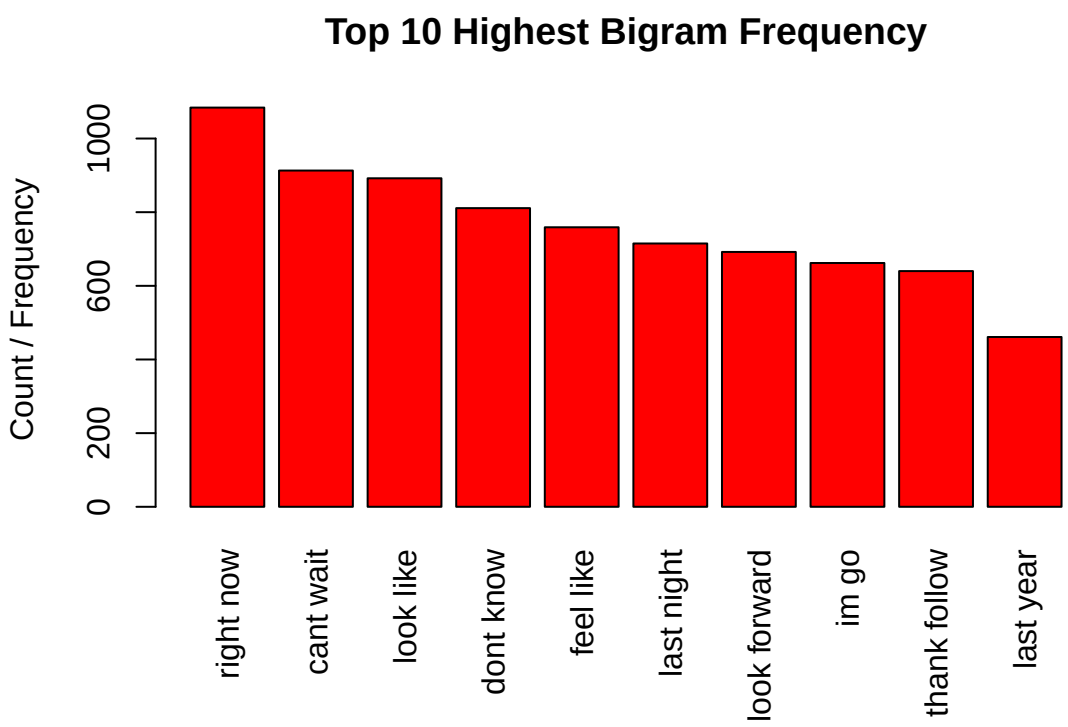
```
# Call Trigram and Frequency functions
bigram <- removeSparseTerms(
  TermDocumentMatrix(Corpus_Processed_Data, control = list(tokenize = Bigram_Tokenizer_Function)), 0.999

bigram_frequency <- Frequency_Function(bigram)


# Call Trigram and Frequency functions
trigram <- removeSparseTerms(
  TermDocumentMatrix(Corpus_Processed_Data, control = list(tokenize = Trigram_Tokenizer_Function)), 0.99

trigram_frequency <- Frequency_Function(trigram)
```

Here are the top 10 bigrams from the Corpus data and a plot:

```
kable(bigram_frequency[1:10,], format = "markdown")
```

	word	freq
right now	right now	1084
cant wait	cant wait	913
look like	look like	892
dont know	dont know	811
feel like	feel like	759
last night	last night	715
look forward	look forward	692
im go	im go	662
thank follow	thank follow	640
last year	last year	461

```
par(mar=c(8,4,4,4))

barplot(height = bigram_frequency$freq[1:10], names.arg = bigram_frequency$word[1:10], main="Top 10 High
        ylab = "Count / Frequency", col = "red", las = 3,)
```

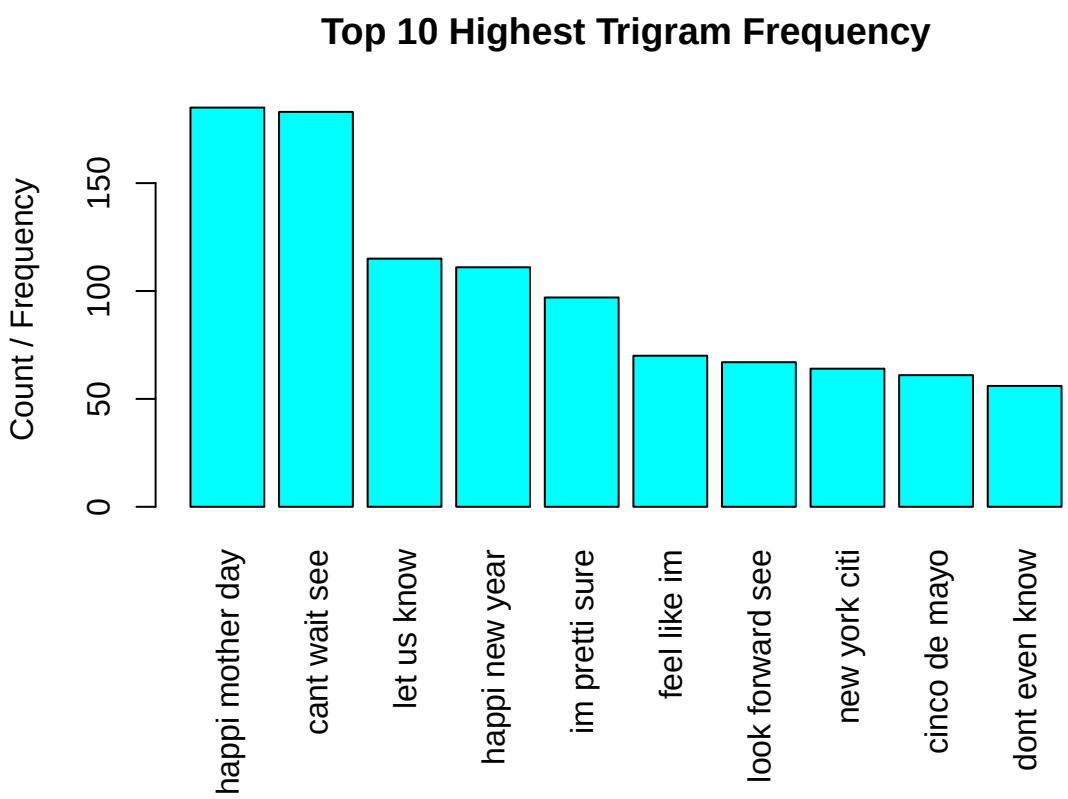


Here are the top 10 trigrams from the Corpus data and a plot:

```
kable(trigram_frequency[1:10,], format = "markdown")
```

	word	freq
happi mother day	happi mother day	185
cant wait see	cant wait see	183
let us know	let us know	115
happi new year	happi new year	111
im pretti sure	im pretti sure	97
feel like im	feel like im	70
look forward see	look forward see	67
new york citi	new york citi	64
cinco de mayo	cinco de mayo	61
dont even know	dont even know	56

```
par(mar=c(8,4,4,4))

barplot(height = trigram_frequency$freq[1:10], names.arg = trigram_frequency$word[1:10], main="Top 10 H
        ylab = "Count / Frequency", col = "cyan", las = 3,)
```



Conclusion

When examining the full set of data from the three sources, we find it is an extremely long and large data file that would be too combersome to examine. Therefore we reduced the corpus to a smaller sample; 5% of the total data, in order to improve the speed of analysis and performance.

From this smaller data sample, we are still able to determine the frequency of most used words and n-grams.

- The most common word (unigram) in the Corpus was: just
- The most common bigram in the Corpus was: right now
- The most common bigram in the Corpus was: happi mother day

Next Steps

During the next part of the project we will continue to examine the Bigrams and Trigrams.This is key to predict the next word and develop the prediction algorithm.

Immediate next steps include:

- Review of our approach and adjust sample size and corpus cleaning to improve prediction.
- Build the predictive model with the identified tokens
- Developed the predictive model and build a shiny app