Predictive Text Analysis - Milestone Report

Vedant Mane

July 19, 2020

Overview

Around the world, people are spending an increasing amount of time on their mobile devices for email, social networking, banking and a whole range of other activities. But typing on mobile devices can be a serious pain. SwiftKey builds a smart keyboard that makes it easier for people to type on their mobile devices. One cornerstone of their smart keyboard is predictive text models.

In this project, we will use our knowledge of Data Science in order to clean and analyze data, building and sampling predictive text models and finally use basics of Natural Language Processing for predicting text output from users in order for them to type on their mobile devices with ease.

Packages

Let us load the packages we will need for cleaning, transforming & exploring our dataset.

```r
#install.packages("tm")
#install.packages("RWeka")
#install.packages("wordcloud")
#install.packages("ggplot2")
#install.packages("cowplot")
require(tm)
require(RWeka)
require(wordcloud)
require(ggplot2)
require(cowplot)
require(knitr)
require(rmarkdown)
```

We know that we our process will require lot of computation time and so we will perform parallel computation of our tasks forming a cluster of cores.

```r
#install.packages("doParallel")
require(doParallel)
# Start Parallel Processing
ncores <- makeCluster(detectCores() - 1)
registerDoParallel(cores = ncores)
getDoParWorkers()
```

```
## [1] 3
```

Dataset

This is the training data to get started that will be the basis for the capstone. You must download the data from the link below and not from external websites to start.

https://d396qusza40orc.cloudfront.net/dsscapstone/dataset/Coursera-SwiftKey.zip

Downloading Data

```
if(!file.exists("./data/data.zip")) {
    zip <- "https://d396qusza40orc.cloudfront.net/dsscapstone/dataset/Coursera-SwiftKey.zip"
    dir.create("./data")
    download.file(zip, destfile = "./data/data.zip")
    unzip("./data/data.zip")
}
```

Reading Data files

```
#Reading Files
readBlogs <- readLines("./final/en_US/en_US.blogs.txt",
                       skipNul = TRUE, encoding = "UTF-8", warn = FALSE)
readNews <- readLines("./final/en_US/en_US.news.txt",
                      skipNul = TRUE, encoding = "UTF-8", warn = FALSE)
readTwitter <- readLines("./final/en_US/en_US.twitter.txt",
                         skipNul = TRUE, encoding = "UTF-8", warn = FALSE)
```

Data Summary

Let us perform some basic exploration tasks on our dataset to identify the size, type and other characteristics of our dataset.

```
tab <- data.frame()
#File Size
sizeBlogs <- file.size("./final/en_US/en_US.blogs.txt") / (1024 * 1024)
sizeNews <- file.size("./final/en_US/en_US.blogs.txt") / (1024 * 1024)
sizeTwitter <- file.size("./final/en_US/en_US.blogs.txt") / (1024 * 1024)
sizeBlogs <- paste(round(sizeBlogs, 0), "MB")
sizeNews <- paste(round(sizeNews, 0), "MB")
sizeTwitter <- paste(round(sizeTwitter, 0), "MB")
##Append Data Frame
tab <- rbind(tab, c(sizeBlogs, sizeNews, sizeTwitter))
names(tab) <- c("Blogs", "News", "Twitter")
row.names(tab)[1] <- c("Size")
#File - Number of lines
linesBlogs <- length(readBlogs)
linesNews <- length(readNews)
linesTwitter <- length(readTwitter)
##Append Data Frame
tab <- rbind(tab, c(linesBlogs, linesNews, linesTwitter))
```

```r
row.names(tab)[2] <- c("Number of lines")
#File - Number of characters
charsBlogs <- sapply(readBlogs, nchar)
charsNews <- sapply(readNews, nchar)
charsTwitter <- sapply(readTwitter, nchar)
##Append Data Frame
tab <- rbind(tab, c(sum(charsBlogs), sum(charsNews), sum(charsTwitter)))
row.names(tab)[3] <- c("Number of characters")
#File - Max Number of Character in a Single Line
maxBlogs <- max(charsBlogs)
maxNews <- max(charsNews)
maxTwitter <- max(charsTwitter)
##Append Data Frame
tab <- rbind(tab, c(maxBlogs, maxNews, maxTwitter))
row.names(tab)[4] <- c("Max Characters")
tab
```

```
##                          Blogs     News   Twitter
## Size                    200 MB   200 MB    200 MB
## Number of lines         899288    77259   2360148
## Number of characters 206824505 15639408 162096241
## Max Characters           40833     5760       140
```

Cleaning & Transforming Data

From exploring the dataset, we see that we will need to clean our dataset as our data is not in english language format we will need for further training and prediction of our models.

```r
#Removing text not in English Language
readBlogs_en <- sapply(readBlogs,
                       function(word)
                           iconv(word, from = "latin1", to = "ASCII", sub = ""))
readNews_en <- sapply(readNews,
                       function(word)
                           iconv(word, from = "latin1", to = "ASCII", sub = ""))
readTwitter_en <- sapply(readTwitter,
                       function(word)
                           iconv(word, from = "latin1", to = "ASCII", sub = ""))
#Counting number of characters in this new data
chars_en <- sapply(list(readBlogs_en, readNews_en, readTwitter_en),nchar)
newChars_en <- sapply(chars_en,sum)
#Percentage of removed characters
rmChars <- ave(1 - as.integer(newChars_en) / as.integer(tab[3,]))[1]
```

Sampling Dataset

Now, we know that a sample of a population will infer predictions to the population and as our dataset is so large and we don't have computation power and memory required for analysis, we will sample 5% of english language data from each of the sources i.e. blogs, news and twitter.

```r
#Sampling the huge dataset
set.seed(123)
blogs <- sample(readBlogs_en, length(readBlogs_en) * 0.05)
set.seed(456)
news <- sample(readNews_en, length(readNews_en) * 0.05)
set.seed(789)
twitter <- sample(readTwitter_en, length(readTwitter_en) * 0.05)
save(blogs, news, twitter, file = "dataset.RData")
#Removing unwanted variables
rm(readBlogs, readBlogs_en, readNews, readNews_en, readTwitter, readTwitter_en)
```

Generating Corpus

We will remove impurities, punctuations, numbers, excess whitespaces, remove most commonly used prepositions and convert into a plain text document.

```r
#Loading dataset
load("dataset.RData")
#Buinding the Corpora
corpus <- VCorpus(VectorSource(c(blogs, news, twitter)),
                  readerControl = list(reader = readPlain, language = "en"))
#Analysing spaces and replacements
spacing <- content_transformer(function(charVec, paTTern) gsub(pattern = paTTern,
                                                              replacement = " ", x = charVec))
corpus <- tm_map(corpus, FUN = spacing, "-")
corpus <- tm_map(corpus, FUN = spacing, "_")
#Performing Transformations
corpus <- tm_map(corpus, content_transformer(tolower))
corpus <- tm_map(corpus, removePunctuation)
corpus <- tm_map(corpus, removeNumbers)
corpus <- tm_map(corpus, stripWhitespace)
corpus <- tm_map(corpus, PlainTextDocument)
corpus <- tm_map(corpus, removeWords, stopwords("english"))
corpus <- tm_map(corpus, stemDocument)
```

Building N-Grams

Now, let us see using n-grams technique which are the words that appear most frequently in our dataset (Unigrams). We will also see which are the words that appear most commonly is pairs and triplets.

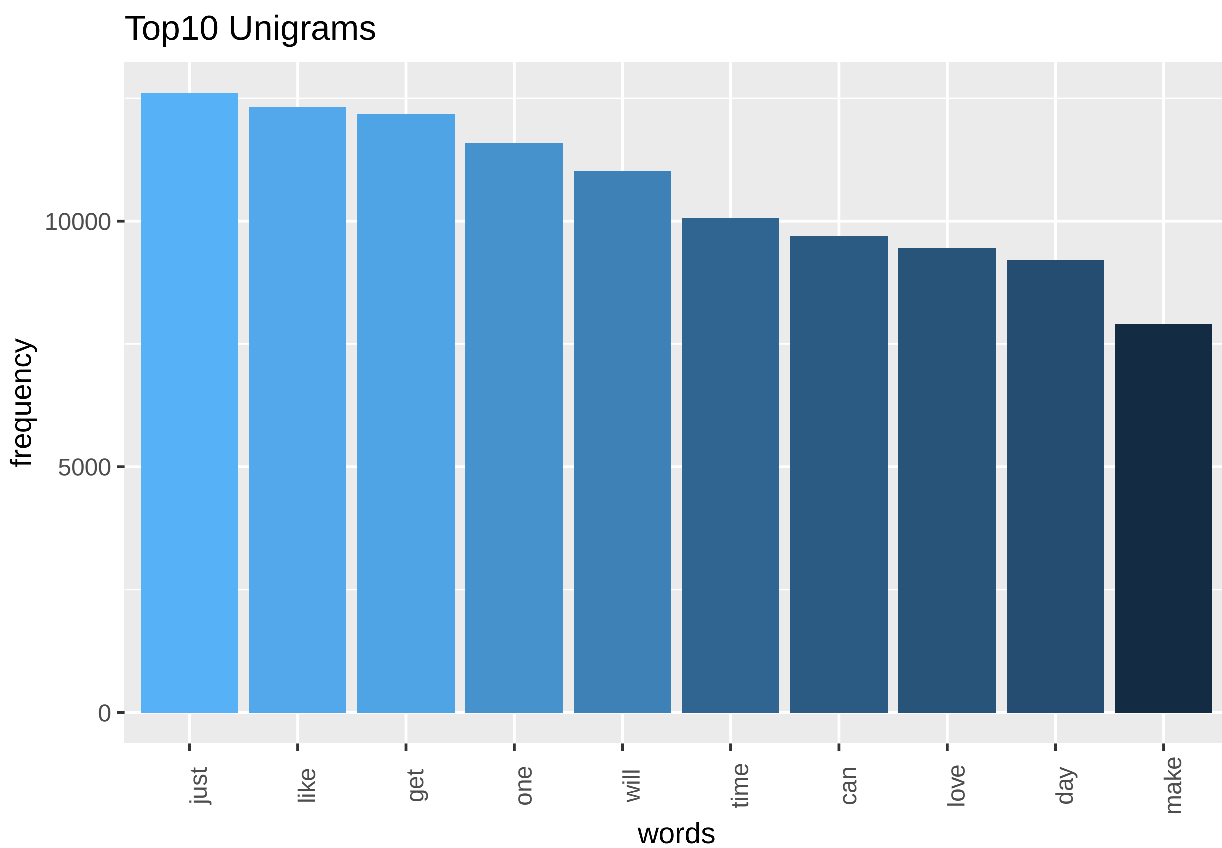
```r
# Tokenizing for n-grams
tokenUniGram <- function(x) {
    NGramTokenizer(x, control = Weka_control(min = 1, max = 1))
}
tokenBiGram <- function(x) {
    NGramTokenizer(x, control = Weka_control(min = 2, max = 2))
}
tokenTriGram <- function(x) {
    NGramTokenizer(x, control = Weka_control(min = 3, max = 3))
}
# Constructing the Document Term Matrix
```

```r
dtmU <- DocumentTermMatrix(corpus, control = list(tokenize = tokenUniGram))
dtmB <- DocumentTermMatrix(corpus, control = list(tokenize = tokenBiGram))
dtmT <- DocumentTermMatrix(corpus, control = list(tokenize = tokenTriGram))
# Finding terms with particular threshold
unigram <- findFreqTerms(dtmU, lowfreq = 200)
bigram <- findFreqTerms(dtmB, lowfreq = 50)
trigram <- findFreqTerms(dtmT, lowfreq = 50)
# Calculating Frequency Terms
freqUni <- colSums(as.matrix(dtmU[,unigram]))
freqBi <- colSums(as.matrix(dtmB[,bigram]))
freqTri <- colSums(as.matrix(dtmT[,trigram]))
# Constructing data frames of n-grams
freqU <- data.frame(word = names(freqUni), frequency = freqUni, row.names = NULL)
freqB <- data.frame(word = names(freqBi), frequency = freqBi, row.names = NULL)
freqT <- data.frame(word = names(freqTri), frequency = freqTri, row.names = NULL)
#Filtering the Top 10 n-grams
dfU <- freqU[order(-freqU$frequency),][1:10,]
dfB <- freqB[order(-freqB$frequency),][1:10,]
dfT <- freqT[order(-freqT$frequency),][1:10,]
```

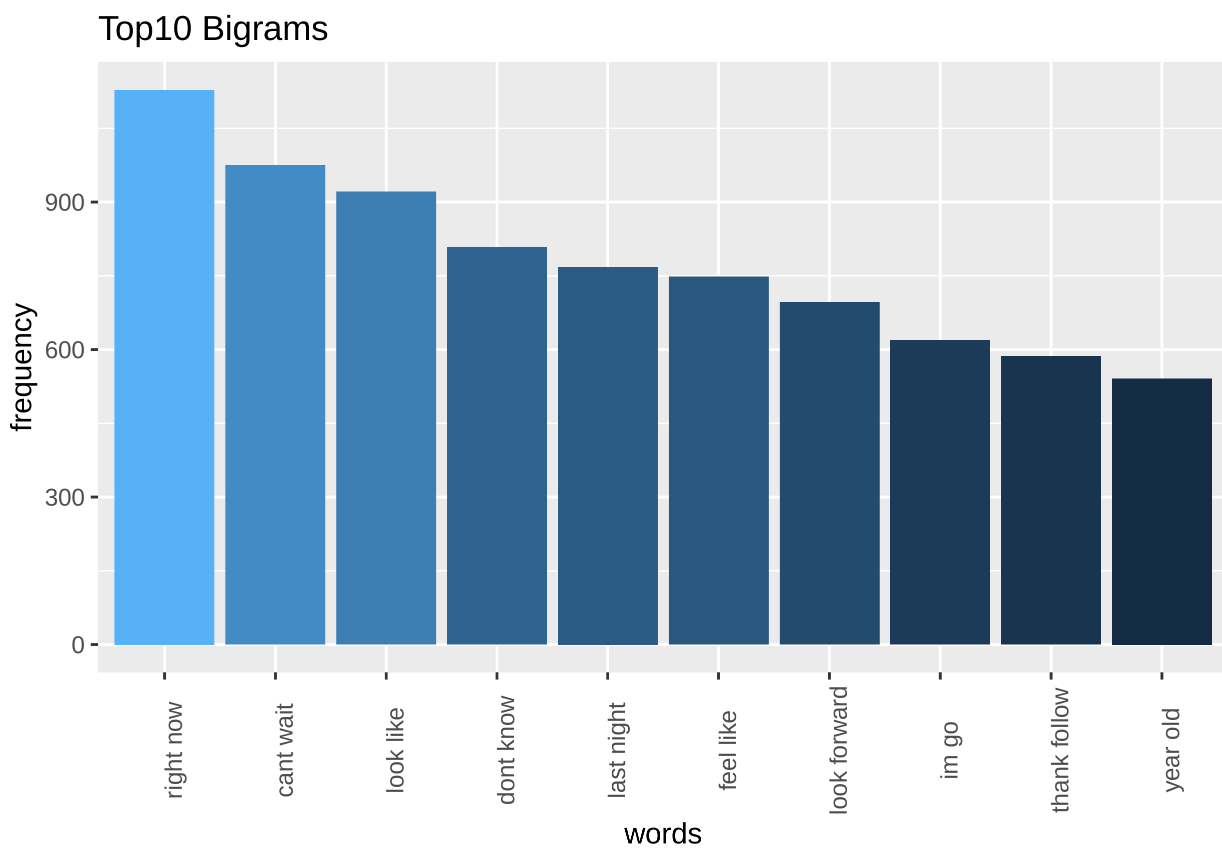
Plotting - EDA

Let us plot the top 10 unigrams, bigrams & trigrams and create a wordcloud of the 100 most frequent words in our dataset.

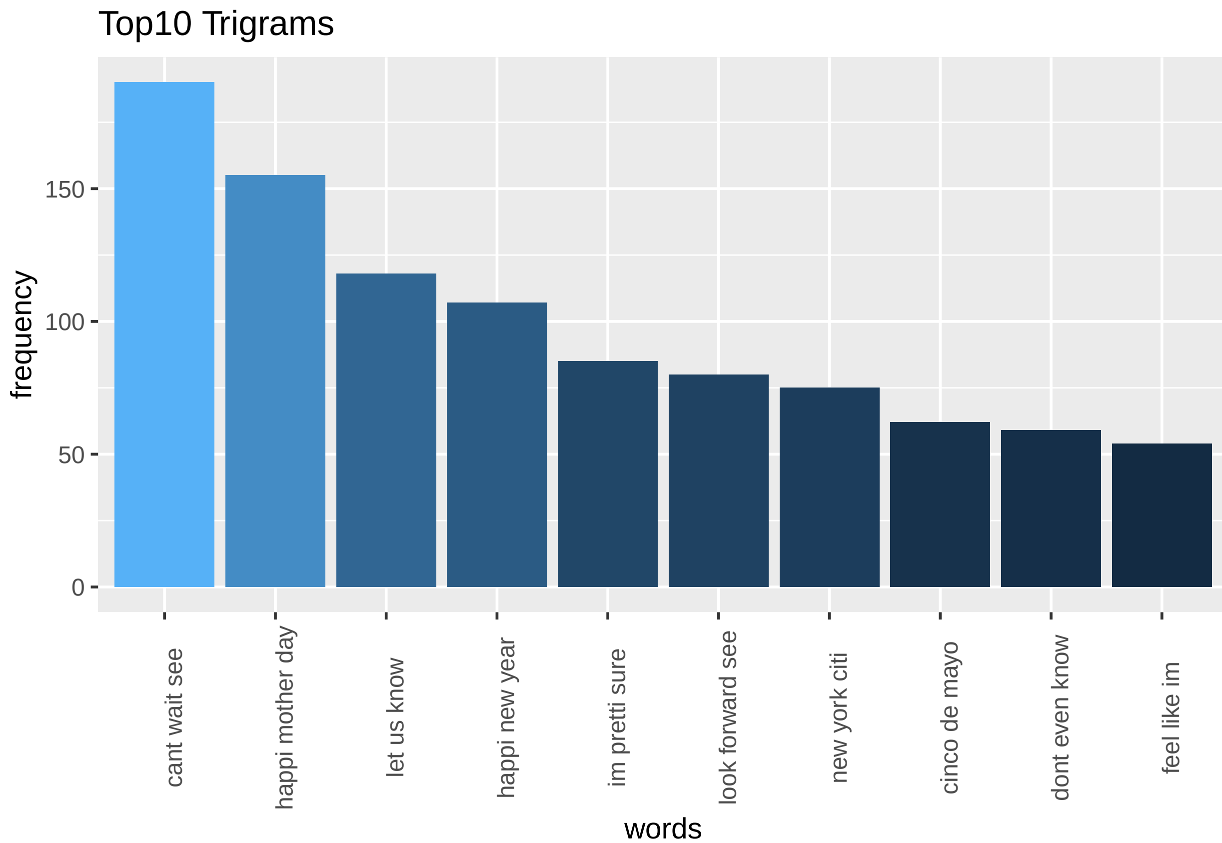
```r
# Top 10 Unigrams
plotU <- ggplot(data=dfU, aes(x=word, y=frequency,fill=frequency)) +
    geom_bar(stat="identity")+guides(fill=FALSE)+
    theme(axis.text.x=element_text(angle=90))+
    scale_x_discrete(limits=dfU$word)+
    labs(title="Top10 Unigrams")+xlab("words")
plotU
```



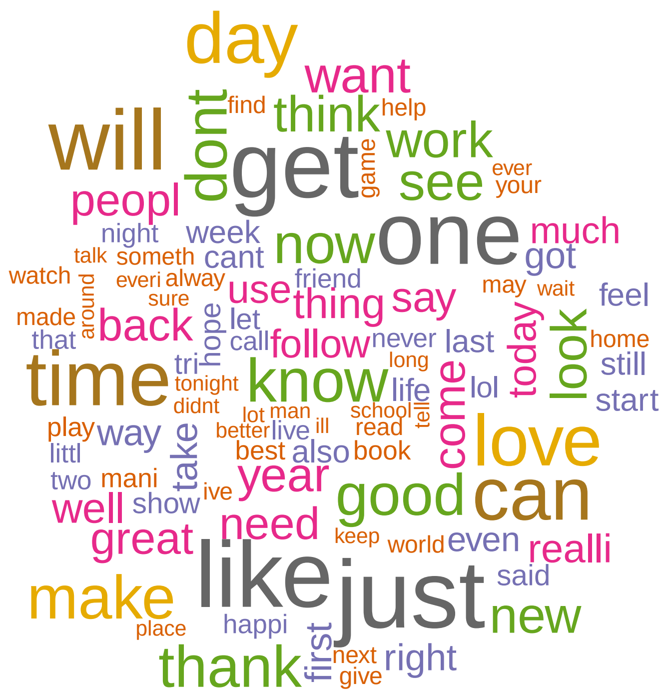

```
# Top 10 Bigrams
plotB <- ggplot(data=dfB, aes(x=word, y=frequency,fill=frequency)) +
    geom_bar(stat="identity")+guides(fill=FALSE)+
    theme(axis.text.x=element_text(angle=90))+
    scale_x_discrete(limits=dfB$word)+
    labs(title="Top10 Bigrams")+xlab("words")
plotB
```




```r
# Top 10 Trigrams
plotT <- ggplot(data=dfT, aes(x=word, y=frequency,fill=frequency)) +
    geom_bar(stat="identity")+guides(fill=FALSE)+
    theme(axis.text.x=element_text(angle=90))+
    scale_x_discrete(limits=dfT$word)+
    labs(title="Top10 Trigrams")+xlab("words")
plotT
```




```
#plot_grid(plotU, plotB, plotT, nrow = 1, ncol = 3)
#Creating Word Cloud
set.seed(5)
wordcloud(names(freqUni), freqUni, max.words = 100,
          scale = c(3, 0.1), colors = brewer.pal(8,"Dark2"))
```



References

The code repository is available at https://github.com/vedantmane/SwiftKey-PredictiveTextAnalyis

```
# Stop Parallel Processing
stopCluster(ncores)
```

Conclusion

This report is about the cleaning, transformations and exploration of our dataset. The n-grams built will be useful for prediction of the text. We will also build a model to handle the unseen n-grams. We will further build a prediction model and integrate it into our shiny app to be deployed on the shiny server.