Capstone Project : Exploratory Analysis Milestone Report

Enock L. Dube

Saturday, March 28, 2015

Introduction

This report serves as a milestone report on the data science specialization capstone project offered by the Johns Hopkins Bloomberg School of Public Health. The main purpose of the project is a build a predictive text data product that could that makes it easier for people to type on their mobile devices. 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. A predictive software product could help reduce the number of keystrokes by predicting the next word such that the user can just selected the word in the sentence from a list instead of typing.

Data

In partners with Coursera, SwiftKey(http://swiftkey.com/ (http://swiftkey.com/)) has provided the data that was used in this project. According to their website, SwiftKey is a company that was founded 2008, and it's mission is to build technology that makes it easy for everyone to create and communicate on mobile. The data is text files compiled from news articles, twitter and blogs in four different languages, namely english, finish, german and russian. For the purposes of this project only the English text files will be used.The data can be downloaded from the following URL: http://d396qusza40orc.cloudfront.net/dsscapstone/dataset/Coursera-SwiftKey.zip (http://d396qusza40orc.cloudfront.net/dsscapstone/dataset/Coursera-SwiftKey.zip) .

The following R commands can be used to download the data from the provided URL. A listing of all the files is also shown below:

```
## Download and unzip file from provided URL
datafile <- "Coursera-SwiftKey.zip"
if(!file.exists(datafile)) {
   fileurl <- "http://d396qusza40orc.cloudfront.net/dsscapstone/dataset/Coursera-SwiftK
ey.zip"
  download.file(fileurl, destfile = datafile, method = "internal")
  unzip(datafile)
}
# list unzipped files
list.files("Coursera-SwiftKey", recursive = TRUE)
```

```
##  [1] "final/de_DE/de_DE.blogs.txt"   "final/de_DE/de_DE.news.txt"
##  [3] "final/de_DE/de_DE.twitter.txt" "final/en_US/en_US.blogs.txt"
##  [5] "final/en_US/en_US.news.txt"    "final/en_US/en_US.twitter.txt"
##  [7] "final/fi_FI/fi_FI.blogs.txt"   "final/fi_FI/fi_FI.news.txt"
##  [9] "final/fi_FI/fi_FI.twitter.txt" "final/ru_RU/ru_RU.blogs.txt"
## [11] "final/ru_RU/ru_RU.news.txt"    "final/ru_RU/ru_RU.twitter.txt"
```

The English text files to be used for the project are as listed below:

```
Eng_TextFiles <- list.files("Coursera-SwiftKey/final/en_US")
Eng_TextFiles
```

```
## [1] "en_US.blogs.txt"   "en_US.news.txt"    "en_US.twitter.txt"
```

Exploratory Analysis

Size of Text Files

In R the size (in btyes) of each of the English text file can be obtained using the file.info () commnands as shown below

```
fid <- paste("en_US/",Eng_TextFiles, sep="")
sizes <- paste("Size of ",fid, " = ", file.info(fid)$size, "bytes")
sizes
```

```
## [1] "Size of  en_US/en_US.blogs.txt  =  210160014 bytes"
## [2] "Size of  en_US/en_US.news.txt  =  205811889 bytes"
## [3] "Size of  en_US/en_US.twitter.txt  =  167105338 bytes"
```

Number of Lines in Each Text File

The number of lines in each of the three English files can be displayed using the following R commands.

```
 library (R.utils)
getLineCount <- function (filename, sourceDir) {

  filename <-  paste(sourceDir,"/", filename, sep="")
  return (countLines(filename))[1]
}
#number of lines
line_count <- sapply(Eng_TextFiles, "en_US", FUN = getLineCount)
line_count
```

```
##   en_US.blogs.txt    en_US.news.txt en_US.twitter.txt
##            899288           1010242           2360148
```

Data Sampling

As shown in previous section, the three English text files contain a large number of lines and therefore do not easiy fit into primary memory (RAM) for processing. For the purposes of this project, only a sample of the data will be used to build a predictive model. One way of getting samples of the data is to spit one large file into smaller files which can be easily read into memory. Shown below is an illustration of an R function that was written to split a big file into smaller files.

```
source("split_TextFiles.R")
split_Source_TextFile (sourceDir ="en_US", sourceFile="en_US.blogs.txt",
targetDir="en_US.blogsSampleData", numLinesPerFile = 1000, numberOfSampleFiles = 10)

list.files("en_US.blogsSampleData")
```

```
##  [1] "data_1.txt"  "data_10.txt" "data_2.txt"  "data_3.txt"  "data_4.txt"
##  [6] "data_5.txt"  "data_6.txt"  "data_7.txt"  "data_8.txt"  "data_9.txt"
```

In the above example, the function creates 10 sample files from the blogs text file. Each sample file contains 1000 lines of text. These smaller sample files are stored in a directory which can be referred to as the ** corpus ** directory.

The above example only shows a sample from the **en_US.blogs.txt** textfile. However, in order to build a more representative prediction model, samples will be taken from all three files.

Token Extraction and N-grams

The final next word prediction product will be build using a N-gram model based on the sampled text. It is envisaged that the model will contain unigram (1-gram), bigram (2-gram), trigram (3-gram) and 4-gram extrated from the text. R packages, such as **package tm** will be used and extract the N-gram tokens from the sample text file. Two functions were written to extract the tokens from the data files and store them in a tokens text file. Function **extractNgramsTokensFromFile (filename, corpusDir, appendFlag=TRUE)** extract n-gram tokens from a single text fil. The Second function, **extractNgramsTokensFromCorpus (SourceCorpusDir, appendStatus= TRUE)** calls/uses the first function to extract tokens from a set of files stored in a directory (**corpus**). The R code below shows an example of how the **extractNgramsTokensFromCorpus** can be used to extract tokens from the three(3) samples text files stored in the corpus directory ** ** as created in a previous section in this report. The code below also list the n-gram textv files that were created.

```
source("extractNgramsTokensFromFile.R")
extractNgramsTokensFromCorpus ("en_US.blogsSampleData")

#show list of n-gram text files created
list.files(pattern = "\\.txt$")
```

```
## [1] "biGramTokens.txt"   "quadGramTokens.txt" "triGramTokens.txt"
## [4] "uniGramTokens.txt"
```

Data cleaning

The extraction function used above were such that undesirable characters such and extra white space were removed from the text. Further cleaning on the outpuut n-gram text files is required to remove profane and other non-english words.

Building the Language Model

The N-gram language model can be built from the cleaned up n-gram text files that were created in the previous sections.

Frequency Tables

N-gram frequency table can be constructed from the text files. Shown below are sample table that were created based on the tokenized files; namely **uniGramTokens.txt**, **biGramTokens.txt**, and **triGramTokens.txt** . The dat comes from the 10

```
#source ("ConstructFrequencyTablesFromTextFiles.R" )
source ("buildFreqTablesFromTokenFiles.R")

#show sample freq tables
uniGramFreqTable
```

```
##           token count
##     1:      the 20422
##     2:      and 12012
##     3:       to 11640
##     4:        i 10073
##     5:        a  9943
##    ---
## 28780:   zurich     1
## 28781: zvelebil     1
## 28782:    zwick     1
## 28783:    zycam     1
## 28784:   zydeco     1
```

```
biGramFreqTable
```

```
##                token count word_i_1
##      1:       of the  2054       of
##      2:       in the  1680       in
##      3:       to the   983       to
##      4:         it s   909       it
##      5:       on the   802       on
##     ---
## 200667: zurich blake     1   zurich
## 200668: zvelebil the     1 zvelebil
## 200669:    zwick who     1    zwick
## 200670:    zycam and     1    zycam
## 200671:    zydeco is     1   zydeco
```

```
triGramFreqTable
```

```
##                   token count  word_i_1
##      1:         i don t   267     i don
##      2:     one of the   170    one of
##      3:       a lot of   153     a lot
##      4:       i didn t   131    i didn
##      5:        i can t   120     i can
##     ---
## 339990: zwick who care     1 zwick who
## 339991: zycam and some     1 zycam and
## 339992:    zydeco is a     1 zydeco is
## 339993:  zyl a brownie     1     zyl a
## 339994:    zyl a guide     1     zyl a
```

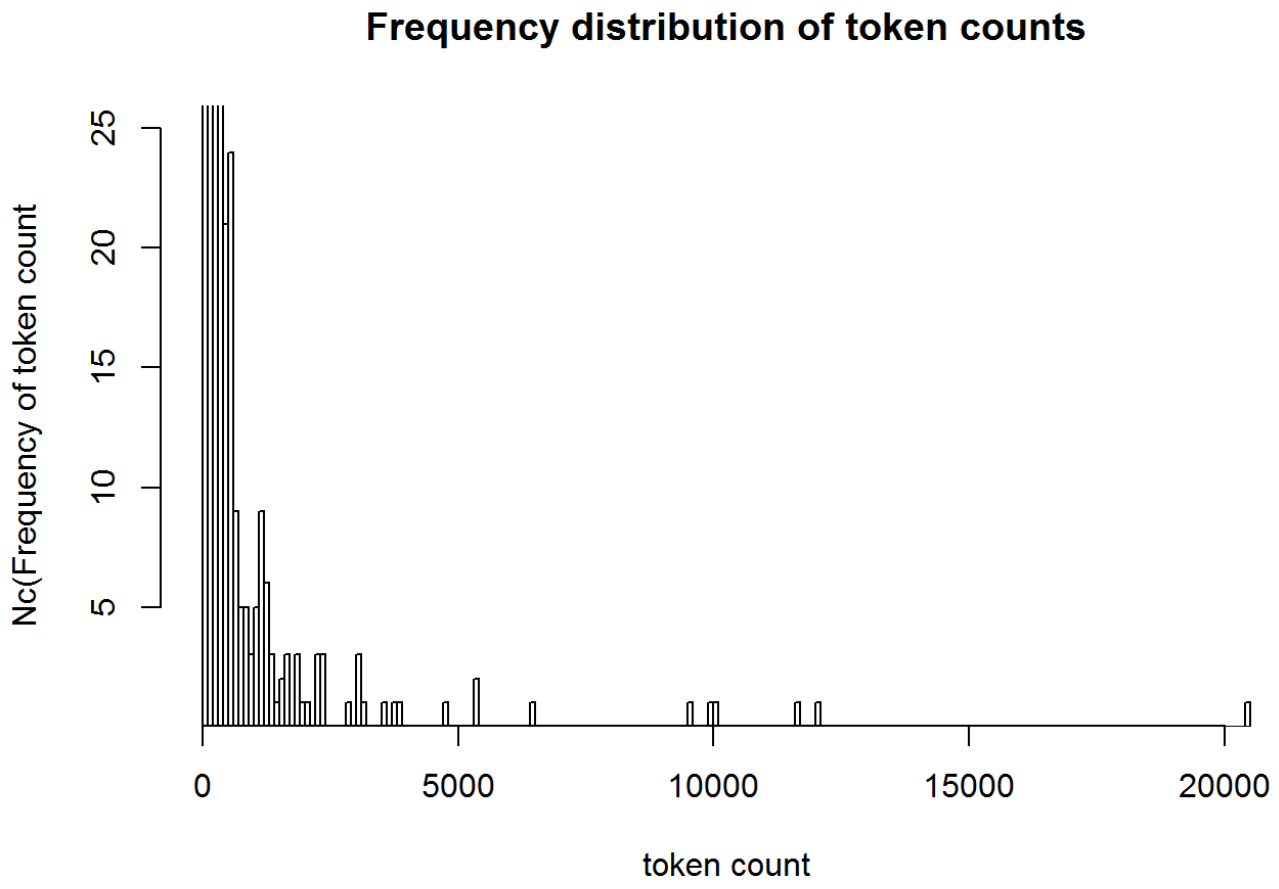
The printout above shows a huge different between largest token count and the smallest token count. Shown below is a summary of the unigram counts:

```
summaryTable <- summary(uniGramFreqTable$count)
summaryTable
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##     1.00     1.00     2.00    14.48     4.00 20420.00
```

The average token count in this sample was 14.48 The histogram below shows a distribution of the unigram tokens count.

```
hist(uniGramFreqTable$count, ylim=c(1,25), xlab = "token count", ylab = "Nc(Frequency o
f token count", breaks=150, main = "Frequency distribution of token counts")
```



Compute n-gram token probabilities

The n-gram token maximum likelihood probabilities can be computed from the frequency as shown in the R code below: The values shown are based on raw token counts. There is still a need to apply smooting in order to get better predictive probabilities. The good turing smooting technique will be used for this purpose. It is expected that the log in the final model the log values of these probabilities will be used instead of the raw scores.

```
source("ComputeProbabilityMatrixTables.R" )
unigramMatrix[order(-count),]
```

```
##           token count      MLEprob
##     1:      the 20422 4.899124e-02
##     2:      and 12012 2.881612e-02
##     3:       to 11640 2.792371e-02
##     4:        i 10073 2.416457e-02
##     5:        a  9943 2.385270e-02
##    ---
## 28780:   zurich     1 2.398944e-06
## 28781: zvelebil     1 2.398944e-06
## 28782:    zwick     1 2.398944e-06
## 28783:    zycam     1 2.398944e-06
## 28784:   zydeco     1 2.398944e-06
```

```
bigramMatrix[order(-count),]
```

```
##                token count word_i_1    MLEprob
##      1:       of the  2054       of 0.21534913
##      2:       in the  1680       in 0.25897950
##      3:       to the   983       to 0.08445017
##      4:         it s   909       it 0.16921072
##      5:       on the   802       on 0.26157860
##     ---
## 200667: zurich blake     1   zurich 1.00000000
## 200668: zvelebil the     1 zvelebil 1.00000000
## 200669:    zwick who     1    zwick 1.00000000
## 200670:    zycam and     1    zycam 1.00000000
## 200671:    zydeco is     1   zydeco 1.00000000
```

```
trigramMatrix[order(-count),]
```

```
##                   token count  word_i_1   MLEprob
##      1:        i don t   267     i don 1.0000000
##      2:     one of the   170    one of 0.4815864
##      3:       a lot of   153     a lot 0.6455696
##      4:       i didn t   131    i didn 1.0000000
##      5:        i can t   120     i can 0.4013378
##     ---
## 339990: zwick who care     1 zwick who 1.0000000
## 339991: zycam and some     1 zycam and 1.0000000
## 339992:    zydeco is a     1 zydeco is 1.0000000
## 339993:  zyl a brownie     1     zyl a 0.5000000
## 339994:    zyl a guide     1     zyl a 0.5000000
```

Way Forward

A reasonable amount of work has been done on this project, but the modelling part is not complete. The data needs more cleaning and removal of undesired words in order to get more accurate predictions. The probabilities need to be recalculated using smoothing to account for non-observed tokens. Finaly the a next word prediction shiny app will be built based on the model.