Milestone Report Submission

Rodrigo Bertollo de Alexandre

Thursday, September 18, 2014

As requirement for the first report submission, we are supposed to demonstrate what we have done so far. At first I've download the Capstone Dataset under the **Announcements** page within the *Task 0 - Understanding the problem*. After extraction, I started to work with the *en_US* folder containing the three files: - en_US.blogs.txt - en_US.news.txt - en_US.twitter.txt. Since these files are a large amount of text data, the best approach method is the **tm package**. It is important to highlight that inside the *DirSource* argument under the *Corpus* function, the encoding must be set to **"UTF-8"** to avoid encoding errors.

```
library("tm")
directory_source <- DirSource("F:/Program Files (x86)/Dropbox/Courses/Coursera Courses/Data Science Caps
                              encoding = "UTF-8")
us_files <- Corpus(directory_source,readerControl=list(reader=readPlain))
```

The file representing *blogs* data has about **nine hundred thousand lines**, the *news* has about **seventy seven thousand lines** and *twitter* files has almost **two million and four hundred thousand lines**. Before starting to work with these files, some formatting was done. At first, words and letters where transformed to lower case. Due to the presence of strange characters the best way to transform all words to lowercase was with the use of the **stringi package**. Moreover, later on this package also allowed to do further modifications to the text and *tokenization*.

```
library("stringi")
#convert to lower case
us_files <- tm_map(us_files, function(x) stri_trans_tolower(x[[1]]))
#transform some unknown characets to english_US
us_files <- tm_map(us_files, function(x) stri_trans_general(x, "en_US"))
```

Since we are working with the English language, all lowercase i and i' where transformed back to I and I' respectively. Unfortunately, the code that I had done for multiple substitutions was slower than doing them separately.

```
#transforming i in I again:
us_files <- tm_map(us_files, function(x) gsub("^i ", "I ", x))
us_files <- tm_map(us_files, function(x) gsub("^i'", "I'", x))
us_files <- tm_map(us_files, function(x) gsub(" i ", " I ", x))
us_files <- tm_map(us_files, function(x) gsub(" i'", " I'", x))
us_files <- tm_map(us_files, function(x) gsub(".i ", ". I ", x))
us_files <- tm_map(us_files, function(x) gsub(".i'", ". I'", x))
```

The function *removePunctuations* was not used due to the fact that this function would also remove . and ' from the text. Without the dots different phrases would be wrongly linked together and without the apostrophe some contractions words would lose its sense, example *"I'm"* to *"Im"*; *"it's"* to *"its"*. Therefore, all end phrasing punctuations where transformed to dots, and all other punctuations with exception of apostrophe where deleted.

```
#remove punctuation with exception of . and '
us_files <- tm_map(us_files, function(x) gsub("[!?,.]+", ".", x))
us_files <- tm_map(us_files, function(x) gsub('[])(;:#%$^*\\~{}[&+=@/"`|<>_]+', "", x))
```

Further on, other modifications like removing extra white spaces, numbers and separating every dot by a new line was also executed. The separation of new lines for every dot was important because during the use of n-grams different phrases wouldn't be linked as one.

```
#remove white spaces
us_files <- tm_map(us_files, stripWhitespace)
#remove numbers
us_files <- tm_map(us_files, removeNumbers)
#remove dot spaces for later split.
us_files <- tm_map(us_files, function(x) gsub(" \\.", ".", x))
us_files <- tm_map(us_files, function(x) gsub("\\. ", ".", x))
#splitting by dots
us_files <- tm_map(us_files, function(x) strsplit(x, "\\."))
```

Because bad languages use (*cursing and swearwords*) also make a critical an important part of the language, they were not removed of the database, but they will be removed during the prediction model.
For later use and backup purposes the file was constantly saved with the function *save(us_files, file="us_files.RData")*. The data was removed from the Corpus with the function **as.list** and converted to a list document file. A matrix was constituted using the argument *unlist* and *byrow = T*. All matrix lines that had less than three words were deleted from the matrix, and saved with the function *save(clean_data, file="clean_data.RData")*. The word count was performed with the **qdap package**.

```
library("qdap")
#convert to a list document
aslines <- as.list(us_files[1])
#convert to matrix with 1 column only
data_table <- matrix(unlist(aslines), ncol = 1, byrow = TRUE)
#remove lines that has less than 3 words
clean_data <- matrix(data_table[wc(data_table)>=3])
```

At the end, the cleaning and processing resulted in a matrix containing 7644814 lines. Tokenization was performed with the **stringi package**.

```
library("stringi")
token <- stri_extract_words(clean_data, locale = "en_US")
dic_data <- data.frame(table(matrix(unlist(token), ncol = 1, byrow = TRUE)))
```

The tokenization resulted in an absurd number of words (509478 words). This happened because some entries that were not words were also considered as words.

```
dic_data100 <- dic_data[1:100,]
alphorder <- dic_data100[order(dic_data100[,1], decreasing = T),]
```

```
head(alphorder)
```

```
##                                                                  Var1 Freq
## 100 aaaaaaaaaiwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwll    1
## 99                                                        aaaaaaaaages    1
## 98                                           aaaaaaaaaayyyyyyeeeeeeeee    1
## 97                                        aaaaaaaaaahhhhhhhhhhhhhhhhh    1
## 96                                                      aaaaaaaaaaargh    1
## 95                                              aaaaaaaaaaaammmmmmmmmm    1
```

Because of this, I've created English word list containing about 655000 english words gathered from innumerous databases containing nouns, verbs, names, names of cities and countries, swearwords and much more.

```
setwd("F:/Program Files (x86)/Dropbox/Courses/Coursera Courses/Data Science Capstone on Coursera/Data")
words <- read.csv("words.csv", sep=";")
length(words[,1])
```

```
## [1] 655110
```

I used this file to filter existing words from my *data_dic*. In this way I was able to create a "not word dictionary" for later use to exclude the lines containing these junk words from my ngrams.

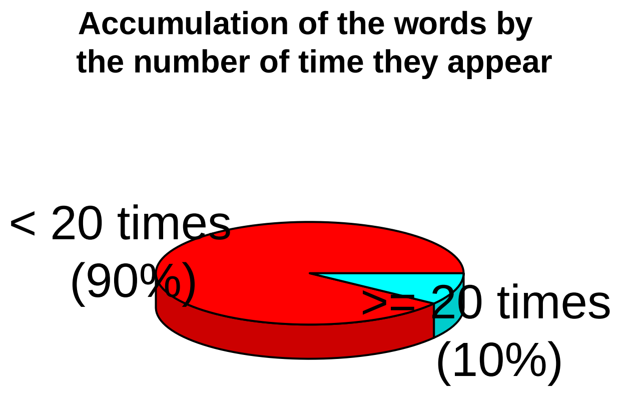
```
dic_data_nw <- dic_data
#couting how many times to work with the loop
y <- round(length(words[,1])/2500)+1
#loop for removing the real words from the TOKEN
for(n in 0:y){
    t = n*2500+1
    b = n + 1
    x = b*2500
    dic_data_nw <- dic_data_nw[!grepl(dic_data_nw[,1], pattern = paste("^",words[t:x,],"$", collapse="|
    print(c("Round",b))
    print(c("Last x value",x))
    print(c("remaining not words",length(dic_data_nw[,1])))
}
```

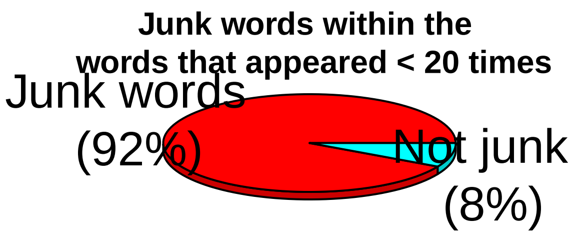
Interestingly, because of this procedure I was able to identify some writing mistakes that should be considered of correction before using the prediction models.

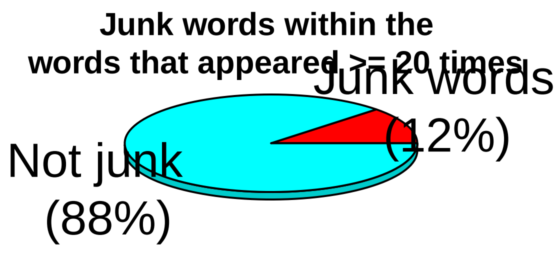
```
head(dic_data_nw,4)
```

```
##          Var1  Freq
## 258615   lol 46614
## 220335  it's 38814
## 210141    im 29989
## 121276 don't 29421
```

Furthermore, if among the 509478 token words, about 458278 appear at most 19 times, whereas 51200 had a frequency equal or higher than 20 times. After the removal of the existing words (as mentioned before), about 83% of the overall were consider junk, (92% of the less than 20 times and 12% of the >= 20 times). .pdf







This junk words will only be deleted after the creation of the n-grams because only the parts that are affected by it will be removed and not the whole phrase.
The n-grams will be performed by dividing the *clean_data* file in 8 different parts due to memory RAM usage. For this, the **ngram package** will be used. tryCatch function is used for avoiding errors due to phrases with less than the minimal words during the n-gram loop. The same code below will be used for all 8 files and for all different n-grams.

```
load("clean_data.RData")
library("ngram")
clean_data1 <- matrix(clean_data[1:1000000])
rm(clean_data)
ngram3_list1 <- apply(clean_data1, 1, function(x) tryCatch({ngram(x , n =3)}, error=function(e){}))
rm(clean_data1)
ngram3_1 <- rapply(ngram3_list1, function(x) as.matrix(get.ngrams(x)))
rm(ngram3_list1)
save(ngram3_1, file="ngram3_1.RData")
rm(ngram3_1)
```

After this, there will be further processing and data cleaning, and separation of the n-grams by alphabetical order in different files.

```
for(i in letters){
    for(n in 1:8){
        test <- get(paste("ngram3_", n, sep=""))
        #if the file does not exist create the file
        if(!exists(paste("With_",i,sep=""))){
            #if it starts with i and I
            if(i == "i"){
                assign(paste("With_",i,sep=""), test[grepl(test, pattern=paste("^[","iI","]", sep=""))]
            }
            #all other letters are lowercased
            else{
                assign(paste("With_",i,sep=""), test[grepl(test, pattern=paste("^[",i,"]", sep=""))])
            }
        }
        #if the file already exists join them toguether
        else if(exists(paste("With_",i,sep=""))){
            #if it starts with i and I
            if(i == "i"){
                assign(paste("With_",i,sep=""), c(get(paste("With_",i,sep="")), test[grepl(test, pattern
            }
            #all other letters are lowercased
            else{
                assign(paste("With_",i,sep=""), c(get(paste("With_",i,sep="")), test[grepl(test, pattern
            }
        }
    }
}
```