Exploring Text Data

Shonda Kuiper

May 21, 2017

Packages Installed: stringi,tm, RWeka, wordcloud

Useful packages not required for this RMD: readtext, and [quanteda](https://cran.r-project.org/web/packages/quanteda/vignettes/quickstart.html).

#### Reading Text Data

The readLines() function from base R can be used to read in the data.

#read in blogs, twitter and news  
Blogs1 <- readLines("C:/Users/KUIPERS/Desktop/RStudio/datasciencecoursera/NLP/final/en\_US/en\_US.blogs.txt", encoding = "UTF-8", skipNul = TRUE, warn = FALSE)  
Twitter1 <- readLines("C:/Users/KUIPERS/Desktop/RStudio/datasciencecoursera/NLP/final/en\_US/en\_US.twitter.txt",encoding = "UTF-8", skipNul = TRUE, warn = FALSE)  
News1 <- readLines("C:/Users/KUIPERS/Desktop/RStudio/datasciencecoursera/NLP/final/en\_US/en\_US.news.txt", encoding = "UTF-8", skipNul = TRUE, warn = FALSE)

#### Creating and Cleaning a Corpus

As described in the [tm package vignette](https://cran.r-project.org/web/packages/tm/vignettes/tm.pdf) A **Corpus** is a collection of text documents (all the writings or works of a particular kind or on a particular subject).

* VCorpus(x, readerControl): The default implementation is the so-called VCorpus (short for Volatile Corpus). These are R objects held fully in memory. We denote this as volatile since once the R object is destroyed, the whole corpus is gone.
* PCorpus implements a Permanent Corpus semantics, i.e., the documents are physically stored outside of R (e.g., in a database), corresponding R objects are basically only pointers to external structures, and changes to the underlying corpus are reflected to all R objects associated with it. Compared to the volatile corpus the corpus encapsulated by a permanent corpus object is not destroyed if the corresponding R object is released.

Since this is a computationally intensive process, we will use only a random sample of .2 percent of the data. We set a seed to ensure the random sample of the data provides exactly the same data each time the code is run.

set.seed(123)  
# Randomly Sample 1% of the lines without replacement  
Blogs2 <- sample(Blogs1, size=length(Blogs1)\*.002, replace=FALSE)   
Twitter2 <- sample(Twitter1, size=length(Twitter1)\*.002, replace=FALSE)   
News2 <- sample(News1, size=length(News1)\*.002, replace=FALSE)   
  
data1 = c(Blogs2, Twitter2, News2)  
length1 = c(length(Blogs2),length(Twitter2), length(News2), length(data1))  
length1

## [1] 1798 4720 154 6672

The tm\_map() function applies each transformation (tolower, stripWhitespace, ect..) to all elements of the corpus.

#???I think this gets rid of unneeded spaces. Only corpus,token and other dfm objects are accepted. If the class of data1 is a character we cannot immediately use the dfm function???  
data2 <- sapply(data1, function(x) iconv(enc2utf8(x), sub = "byte"))  
data2 <- (data2[!is.na(data2)])  
  
# Use the tm package to convert to a Volatile Corpus which realizes semantics known from most R objects and then conduct some basic data cleaning  
Corpus1 <- VCorpus(VectorSource(data2))  
Corpus1 <- tm\_map(Corpus1, tolower) # Make all words lower case  
Corpus1 <- tm\_map(Corpus1, removePunctuation) # Remove all punctuation  
Corpus1 <- tm\_map(Corpus1, removeNumbers) # Remove all numbers  
Corpus1 <- tm\_map(Corpus1, stripWhitespace) # Remove all whitespace  
Corpus1 <- tm\_map(Corpus1, PlainTextDocument) # ???Remove all make plain text???  
profanity <- c("([Ff][Uu][Cc][Kk]",  
 "[Ss$][Hh][Ii][Tt]",  
 "[Aa@][Ss$][Ss$]",  
 "[Aa@][Ss$][Ss$][Hh][Oo][Ll][Ee]",  
 "[Cc][Uu][Nn][Tt]",  
 "[Dd][Aa][Mm][Nn]",  
 "[Nn][Ii][Gg][Gg][Ee][Rr])", sep="|")  
Corpus1 <- tm\_map(Corpus1, removeWords, profanity)  
  
#Other common transformations include removing common words (a, the, or):  
#tm\_map(abs, removeWords, stopwords("english")) # or  
#tm\_map(Corpus1, removeWords, c(stopwords("english"),"my","custom","words"))   
#stem words (using only the root words?) #tm\_map(Corpus1, stemDocument)  
#Corpus1 <- tm\_map(Corpus1, toSpace, "/|@|\\|") #????

A common approach in text mining is to create a **term-document matrix** from a corpus. In the tm package the classes TermDocumentMatrix and DocumentTermMatrix (depending on whether you want terms as rows and documents as columns, or vice versa) **employ sparse matrices for corpora??**. Term-document matrices tend to get very big already for normal sized data sets. Therefore we provide a method to remove sparse terms, i.e., terms occurring only in very few documents. Normally, this reduces the matrix dramatically without losing significant relations inherent to the matrix: > inspect(removeSparseTerms(dtm, 0.4)) This function call removes those terms which have at least a 40 percentage of sparse (i.e., terms occurring 0 times in a document) elements. {.2 would remove most words and .99 will essentially keep all words.}

Inspecting a term-document matrix displays a sample, whereas as.matrix() yields the full matrix in dense format (which can be very memory consuming for large matrices).

#### Exploratory ngram Analysis

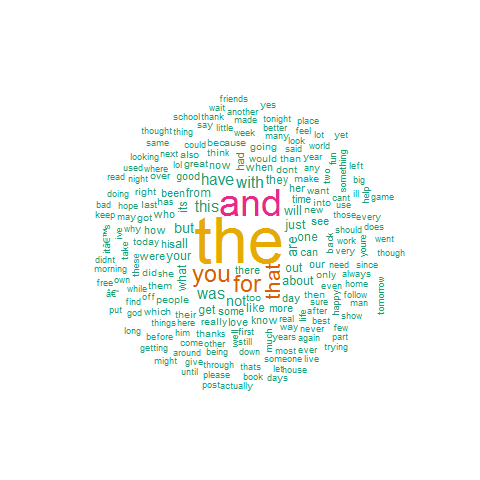
We are often interested in looking at common words or phrases withing the corpus.

We will use the RWeka package create 3 term-document matrices for unigrams, bigrams and rigrams. These are commonly referred to as **n-grams**, a contiguous sequence of n items from a given sequence of text or speech.

#UniTokens, BiTokens and TriTokens  
uniToken <- function(x) NGramTokenizer(x, Weka\_control(min = 1, max = 1))  
biToken <- function(x) NGramTokenizer(x, Weka\_control(min = 2, max = 2))  
triToken <- function(x) NGramTokenizer(x, Weka\_control(min = 3, max = 3))  
  
uniTm <- TermDocumentMatrix(Corpus1, control = list(tokenize = uniToken))  
UniTm <- removeSparseTerms(uniTm, 0.8)  
  
biTm <- TermDocumentMatrix(Corpus1, control = list(tokenize = biToken))  
biTm <- removeSparseTerms(biTm, 0.999) #???Is this needed???  
  
#triTm <- TermDocumentMatrix(Corpus1, control = list(tokenize = triToken))  
#triTm <- removeSparseTerms(triTm, 0.8)

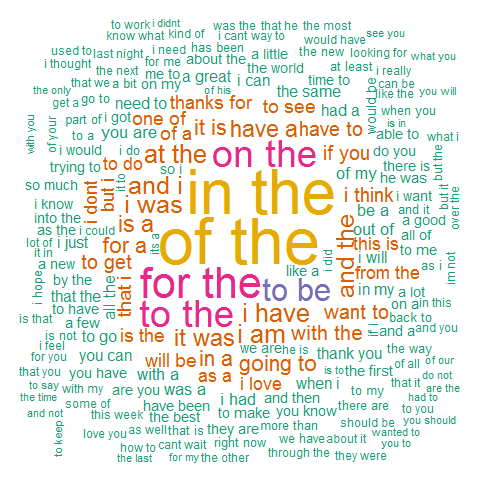
Creating a word cloud of the unigram data

Freq1 <- sort(rowSums(as.matrix(uniTm)), decreasing = TRUE)  
uniDF <- data.frame(word = names(Freq1), freq = Freq1)  
  
wordcloud(words = uniDF$word, freq = uniDF$freq, max.words=200,   
 random.order=FALSE, rot.per=0.1, use.r.layout=FALSE, ordered.colors=FALSE, colors=brewer.pal(6, "Dark2"))



Creating a word cloud of the bigram data

Freq2 <- sort(rowSums(as.matrix(biTm)), decreasing = TRUE)  
biDF <- data.frame(word = names(Freq2), freq = Freq2)  
  
wordcloud(words = biDF$word, freq = biDF$freq, max.words=200,   
 random.order=FALSE, rot.per=0.1, use.r.layout=FALSE,  
 ordered.colors=FALSE, colors=brewer.pal(6, "Dark2"))



The most frequent ngrams

# The most common words(unigrams)  
head(uniDF,20)

## word freq  
## the the 5830  
## and and 3172  
## you you 1676  
## for for 1538  
## that that 1397  
## with with 907  
## this this 856  
## was was 854  
## have have 806  
## are are 713  
## but but 670  
## not not 611  
## all all 560  
## your your 534  
## just just 528  
## about about 476  
## will will 472  
## from from 464  
## out out 460  
## like like 438

# The most common word pairs (bigrams)  
#Freq2 <- sort(rowSums(as.matrix(biTm)), decreasing = TRUE)  
#biDF <- data.frame(word = names(Freq2), freq = Freq2)  
head(biDF, 20)

## word freq  
## of the of the 505  
## in the in the 484  
## for the for the 317  
## to the to the 274  
## on the on the 272  
## to be to be 214  
## i am i am 164  
## at the at the 159  
## i have i have 158  
## i was i was 157  
## and the and the 151  
## and i and i 150  
## it was it was 147  
## have a have a 137  
## is a is a 134  
## for a for a 130  
## in a in a 127  
## going to going to 124  
## it is it is 122  
## with the with the 116