PredictiveTextEngine

Carlos Mercado

December 4, 2017

The Plan

Using over 4 million text entries, combined from twitter, blog posts, and news articles, we will create a predictive text engine in a Shiny App.

The steps will be as followed:

- Download
- Describe the relevant functions we will use
- Clean the data, using said functions
- Trim the data using said functions AND the input given to us by the app user
- Finally, present the predictions in order.

NOTE: In the Final Shiny APP due to ShinyApps.io memory limits The app's version of thebigcorpus is derived from the following "code" set.seed(4) x <- sample(thebigcorpus,100000) dput(x,file = "thebigcorpus")

Let's Begin.

The Download

This may take between 1-3 minutes depending on your system.

We download, unzip, extract the ENGLISH language files, and combine the twitter, blog, and news posting into one corpus. I am going to diverge from others here and NOT format it into a data.table and NOT do any sampling or arbitrary weighting.

Using the data.table approach is useful for the prediction engine, but much slower for the indexing of which entries are relevant to us.

Sampling is great, but I want to be able to return predictions even for rarer English phrases. And of course, the "Unreasonable Effectiveness of Data" applies.

I feel that weighting blog posts or news articles higher than tweets has a strong argument, as twitter's 140 character limit incentivizes users to use novel and shorthand english (idk, lol, nbd, idc, wyd… the list goes on). I am going to rely on the lazy algorithm approach to ignore those unless prevalent. Weighting will be done using a Katz Backoff approach, described later.

```
zipURL <- "https://d396qusza40orc.cloudfront.net/dsscapstone/dataset/Coursera-SwiftKey.zip"
download.file(zipURL, destfile = "./swiftzip.zip") #548 MB
unzip("./swiftzip.zip", list = TRUE) #get the file names don't extract
```

```
##                              Name    Length                Date
## 1                           final/         0 2014-07-22 10:10:00
## 2                  final/de_DE/         0 2014-07-22 10:10:00
## 3  final/de_DE/de_DE.twitter.txt  75578341 2014-07-22 10:11:00
## 4    final/de_DE/de_DE.blogs.txt  85459666 2014-07-22 10:11:00
## 5     final/de_DE/de_DE.news.txt  95591959 2014-07-22 10:11:00
## 6                  final/ru_RU/         0 2014-07-22 10:10:00
## 7    final/ru_RU/ru_RU.blogs.txt 116855835 2014-07-22 10:12:00
## 8     final/ru_RU/ru_RU.news.txt 118996424 2014-07-22 10:12:00
## 9  final/ru_RU/ru_RU.twitter.txt 105182346 2014-07-22 10:12:00
## 10                 final/en_US/         0 2014-07-22 10:10:00
## 11 final/en_US/en_US.twitter.txt 167105338 2014-07-22 10:12:00
## 12   final/en_US/en_US.news.txt 205811889 2014-07-22 10:13:00
## 13   final/en_US/en_US.blogs.txt 210160014 2014-07-22 10:13:00
## 14                 final/fi_FI/         0 2014-07-22 10:10:00
## 15    final/fi_FI/fi_FI.news.txt  94234350 2014-07-22 10:11:00
## 16   final/fi_FI/fi_FI.blogs.txt 108503595 2014-07-22 10:12:00
## 17 final/fi_FI/fi_FI.twitter.txt  25331142 2014-07-22 10:10:00
```

```
filepth <- getwd() # get the working directory
unzip("./swiftzip.zip", exdir = filepth) #unzip the files into the working directory
#inside the directory is now a new directory called "final"

#NOTE if data are already downloaded, start reupload from this line -----

#inside "final" is 4 folders, 1 for each language

setdirectory <- "./final/en_US" #specify the directory with the ENGLISH US files
setwd(setdirectory)
 #twitter, news, and blog txt files


twittertxt <- readLines("en_US.twitter.txt", skipNul = TRUE)  #twitter data
blogtxt <- readLines("en_US.blogs.txt", skipNul = TRUE)       #blog data

newscon <- file("en_US.news.txt",'rb')                        #news data
newstxt <- readLines(newscon)
close(newscon)

#News was downloaded in a binary read due to special chars, nulls, and SUBs in the data

rm(newscon) #clean environment

thebigcorpus <- c(twittertxt,blogtxt,newstxt) #combine it all
rm(twittertxt,blogtxt,newstxt) #remove duplicates
rm(filepth,setdirectory,zipURL) #clean environment
```

Relevant Functions

Our example phrase to help us think about the process: “I want to go to the beach because I like swimming.”

1. removestopwords() / remove.vector.stopwords()

This function removes common english words such as: I, a, the, of, did, not… The full list of words is available in the tm package in the stopwords(kind="en") function.

This function, applied to our example phrase, would output: "want go beach like swimming" This is great for connecting "go"" to the word "beach"" or "beach"" to the word "swimming"" in as few words as possible, it also cuts the data in size while losing minimal meaning.

NOTE: This function works great for single strings, but for large characters (1000+ elements) it's too slow. We need to create a vectorized version - i.e. remove.vector.stopwords()

2. countwords() - counts the number of words in a phrase
3. removenwords() - removes a desired number (n) of words from a phrase, starting at the first one.
4. getlastnwords() - retrieves a desired number (n) of words from a phrase, starting from the last one This function is used to get our prediction word. For example, "want go beach like _____" hypothetically, could return "sun", "swim", "swimming", "drink", "tan"… we're going to be trusting the quantity of data to best predict words.
5. makeInternational() - replaces foreign letters such as "ê" with "e". Can handle large elements, does NOT need to be looped/applied.
6. thebigclean() - cleans the input by forcing lowercase, changing foreign characters, removing punctuation, numbers, stopwords, and extra spaces. Each of these arguments is optional, but recommended. -NOTE: It is critical that the order of the functions be as written above: lowercase, then international, then punctuation, then numbers, then stopwords, then extra spaces last.
7. katzbackoff.grep() - The powerhouse function. It takes an input, then indexes the corpus to find which entries contain the input. It then repeats the process by removing words to increase flexibility (default is everything but last two words, this can be changed). Entries that contain more of the words are weighted by duplication. The duplicates can be erased if desired, but this is mathematically equivalent to only searching a single word (the last word) and returning the most common next word.

```
library(tm)
```

```
## Loading required package: NLP
```

```
removestopwords <- function(inputtext, language = "en", lowercase = TRUE){

        #This version is good for single strings, like inputtext and makes lowercase
        #It allows for punctuation, I won't be using it, but it may be useful to you.

        stops <- stopwords(kind = language) #from the tm package
        inputtext <- strsplit(inputtext, split = " ") #list of length 1

        if(lowercase == TRUE) #the stopwords are lowercase, so be careful
        inputtext <- tolower(inputtext[[1]]) #character class and lowercase
        else inputtext <- inputtext[[1]]
        inputtext <- subset(inputtext,
                            !(inputtext %in% stops)) #subset, NOT stopwords

        return(paste0(inputtext, collapse = " ")) #back in string form
}

remove.vector.stopwords <- function(inputcorp) {

        library(tm)
        #This is the fast version for removing stopwords from large character classes
        #make sure to remove punctuation, numbers, symbols, foreign letters, FIRST

        badwords <- stopwords(kind="en") #get the 174 stopwords
        badwords <- gsub("\'","",badwords) #remove apostrophes
        #again, punctuation should already be removed

        inputcorp <- removeWords(inputcorp,badwords)
        return(inputcorp)
        #Remove extra whitespace LAST: see thebigclean()
}

countwords <- function(inputtext){
        inputtext <- strsplit(inputtext, split = " ") #list
        inputtext <- inputtext[[1]] #character class
        baselength <- length(inputtext)
        return(baselength)
}

removenwords <- function(inputtext, num.words.to.remove = 1){
        inputtext <- strsplit(inputtext, split = " ") #list
        inputtext <- inputtext[[1]] #character class
        baselength <- length(inputtext)
        if(num.words.to.remove >= baselength)
                return("you erased everything!")

        inputtext <- inputtext[(num.words.to.remove+1):baselength] #starts at n+1
        return(paste0(inputtext, collapse = " ")) #back in string form
}

getlastnwords <- function(inputtext, num.words.to.get = 1){
        inputtext <- strsplit(inputtext, split = " ") #list
        inputtext <- inputtext[[1]] #character class
```

```
        baselength <- length(inputtext)
        if(num.words.to.get > baselength)
                return("stop, you're getting everything!")
        inputtext <- inputtext[(baselength + 1 - num.words.to.get):baselength]
        return(paste0(inputtext, collapse = " "))
}

makeInternational <- function(inputtext){
        inputtext <- iconv(inputtext, to="ASCII//TRANSLIT")
        #makes foreign letters into standard letters, some symbols become punctuation
        #NOTE this does NOT need to be looped over large character classes, just input all of it
        return(inputtext)
        }

thebigclean <- function(inputtext, removestops = TRUE,lowercase = TRUE,
                        punctuation = TRUE, numbers = TRUE, international =TRUE){
        library(tm)
        if(lowercase == TRUE)
        inputtext <- tolower(inputtext) #works on large characters

        if(international == TRUE)
        inputtext <- makeInternational(inputtext) #works on large characters

        if(punctuation == TRUE)
        inputtext <- gsub("[[:punct:]]", "", inputtext) #removes punctuation
        #works on large characters

          if(numbers == TRUE)
        inputtext <- gsub("[0-9]","",inputtext) #removes numbers
        #works on large characters

        if(removestops == TRUE)
        inputtext <- remove.vector.stopwords(inputtext)
        #removes stopwords (punctuation adjusted) using tm package, faster than a loop

        inputtext <- gsub("\\s+"," ",inputtext) #removes extra spaces from above, DO THIS LAST!
        #works on large characters

        return(inputtext)
        }

katzbackoff.grep <- function(inputtext,thecorpus,rmvstops = TRUE,minwords = 2,duplicates = TRUE)
{

        workingtext <- thebigclean(inputtext, removestops = rmvstops) #cleans the input text
         thecorpusindex = NULL #creates an empty index vector

        #Note for speed, we want to immediately trim thecorpus by the minimum words.
         #i.e There is no point in searching the entire corpus for "go beach like" if
         #our minimum is that every entry has "beach like"
         #this backward search allows for faster weighting of "go beach like", "want go beach li
ke"... etc

         minimum.criteria <- getlastnwords(workingtext, num.words.to.get = minwords)
```

```
         thecorpus <- grep(minimum.criteria, thecorpus, value = TRUE) #only relevant corpus is s
earched for weighting #TRUE is to get the element NOT the index

        repeat{  #repeat until the minimum words allowed are reached
        newindex <- grep(workingtext,thecorpus)   #search for the text in the corpus
        thecorpusindex <- c(thecorpusindex, newindex) #add it to the index
        workingtext <- removenwords(workingtext, 1) #take off the first word and repeat

        if(countwords(workingtext) < minwords){  #finish after minwords has been run.
                break
                }
        if(workingtext == "error, try more input?"){ #if your input text has too few non-stopwor
ds
                break
        }
        }

        if(duplicates == FALSE){ #it is better to allow duplicates for weighting
        thecorpusindex <- unique(thecorpusindex) #equivalent to searching last non-stop word
        }

        return(thecorpusindex) #returns an index of relevant entries, duplicated as default
        }
```

Data Cleaning and Caveats

I want the data to:

- Be in all lowercase
- have no foreign letters: âíüÂáÁêãçàúü (some ASCII conversion makes punctuations)
- Have no punctuation: ~!@#$%^&* (mailto:~!@#$%%5E&*)(){}_+:“<>?,./;’[]-=
- Have no numbers
- Have no stopwords
- Have extra spaces removed

A 100,000 element sample of thebigcorpus took 1.4 seconds to makeInternational. A 100,000 element sample of thebigcorpus took 3.4 seconds to do thebigclean (stopwords not removed) A 100,000 element sample of thebigcorpus took 5 seconds to remove stopwords A 100,000 element sample of the bigcorpus took 10 seconds to do the entire thebigclean() process ordered appropriately.

For stability purposes, we’ll do each process on thebigcorpus individually.

```
thebigcorpus <- tolower(thebigcorpus) #makes lowercase; 37 seconds
thebigcorpus <- makeInternational(thebigcorpus) #changes foreign characters; 60 seconds
thebigcorpus <- gsub("[[:punct:]]", "",thebigcorpus) #erases punctuation; 42 seconds
thebigcorpus <- gsub("[0-9]","",thebigcorpus) #erases numbers; 21 seconds
thebigcorpus <- remove.vector.stopwords(thebigcorpus) #removes stopwords; 209 seconds
thebigcorpus <- gsub("\\s+"," ",thebigcorpus) #replaces all series (1+) of spaces with a single
 space; 63 seconds
```

For the Shiny app, the output of this cleaning process (which will be smaller in memory) will be loaded with the app for speed purposes.

Prediction Model

With a weighted index of relevant entries (katzbackoff.grep), we filter the corpus using the relevant entries (weighted as duplicates) and get all the n-grams that begin with the last n words of our input text.

For example, “I want to go to the beach because I like +predictedword”, becomes “want go beach like +predictword”, the bigrams are “like + predictedword”, the trigrams are “beach like +predictedword”.

Note that the default in the katzbackoff.grep is 2 words, which would be trigram searches. “beach like …” is the minimum that every relevant entry contains.

We then extract all the predictions (+predictword) and compute their proportion/probability and return it to the app user.

```
#NOTE This model runs all previous functions with their default values
#Strongly recommend to leave the minimum number of words at 2 or more to increase speed.

library(tidytext) #for n-grams
library(dplyr) #masks filter for subsetting
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
textpred <- function(inputtext, thecorpus = thebigcorpus, minimum.n = 2, number.of.predictions =
 3){
        if(countwords(inputtext) > 8)
                print("Please be patient with longer inputs")

        workingtext <- thebigclean(inputtext) #cleans everything as the corpus was cleaned
        theindex <- katzbackoff.grep(workingtext,thecorpus, minwords = minimum.n) #duplicates de
fault
        #"I want to go to the beach because I like" takes 6.04 seconds to repeat search the enti
re corpus
        #NOTE This returns an index based on a corpus meeting the minimum criteria (see the func
tion)

        if(length(theindex) == 0){
                return("Input too short or too rare")
        }
         #This code is taken from the katzbackoff.grep code
         # "theindex"" above matches the corpus "initialcorpus" created here:
         minimum.criteria <- getlastnwords(workingtext, num.words.to.get = minimum.n)
         initialcorpus <- grep(minimum.criteria, thecorpus, value = TRUE)
         #initial corpus is the elements of the original corpus that meet the minimum criteria

        therelevantcorpus <- initialcorpus[theindex] #
        #NOTE: The duplication is key-  although we search using the minimum number of words
        # Entries with more keywords are weighted by being duplicated
        # every entry meets the minium word criteria, per the katz backoff method

         therelevantcorpus <- data.frame(therelevantcorpus,
                                         stringsAsFactors = FALSE) #for tokening
        #makes a data frame of only the relevant entries of the corpus

        ngram.table <- unnest_tokens(therelevantcorpus, ngram,
                                     therelevantcorpus,
                                     token = "ngrams",
                                     n = minimum.n + 1)
        #uses the minimum number of words to make n-grams from the relevant corpus
        # minimum words from the input + a prediction word

        keywords <- getlastnwords(workingtext, minimum.n) #gets the ending of the input
        keywords <- paste0("^",keywords, collapse = "") #makes it searchable in grepl
        ngram.table <- filter(ngram.table, grepl(keywords,ngram)) #finds it
        #NOTE grepl returns TRUE/FALSE as opposed to grep which returns indexes or values of tru
e

        #to predict a new word, we filter by ngrams that begin with the minimum.n of the working
text, counting backwards based on the minimum number of words used to search with.

        #NOTE: if the last word in your search has alternative forms
        #such as {like, likely, liked}, they're included - A way around this is to STEM words
        #a process not used here

        #now we simply split the grams into the correct number of columns and return a table of
 the final column
```

```
        #e.g. our predicted word

       predtable <- sort(table(sapply(ngram.table$ngram,getlastnwords, USE.NAMES = FALSE)), decr
easing = TRUE)
        #cuts the names for aesthetic, only shows the predicted word

        predtable <- round(predtable / sum(predtable),2) #gets proportion

        if(length(predtable) < number.of.predictions){
                number.of.predictions <- length(predtable)}
        #prevents NAs from being shown if the requested number of predictions is too high

        return(predtable[1:number.of.predictions])

        }
```

Test Cases

A few test cases I thought be interesting:

“I can’t wait to meet someone _____”

“I don’t know what I want to eat today, maybe I’ll eat _____”

“After I left the airport I took a _____”

“I’ve been feeling sick all week, maybe I should go to the _____”

The default will be to return the top 3 results for each test, using a minimum of the last two (non-stop) words.

```
test1 <- "I can't wait to meet someone"
test2 <- "I don't know what I want to eat today, maybe I'll eat"
test3 <- "After I left the airport I took a"
test4 <- "I've been feeling sick all week, maybe I should go to the"

textpred(test1)
```

```
##
##  new else like
## 0.09 0.04 0.03
```

```
textpred(test2)
```

```
## [1] "Please be patient with longer inputs"
```

```
##
## lettuce  makeup    much
##    0.06    0.06    0.06
```

```
textpred(test3)
```

```
##
## diverted     full    house
##     0.17     0.17     0.17
```

```
textpred(test4)
```

```
## [1] "Please be patient with longer inputs"
```

```
##
## thing  back  idea
##  0.04  0.03  0.02
```