JHU Data Science Capstone - Milestone Report

Kevin Carhart

December 9, 2018

Introduction

This is the Week Two milestone report for the Capstone project. The purpose of this report is to demonstrate to the reader that I have gotten started opening and reading the data, that I have done some exploratory analysis, and that my word-prediction application is underway.

Preparation

First of all, I cleared the memory space in R Studio. Some of these steps are going to tax my system resources so I want to start with a clean slate. I also set a seed from the system time.

```
rm(list=ls())
set.seed(as.numeric(as.POSIXct(Sys.time())))
```

Here are the libraries I will be using. I noticed in one of Len Greski's informative articles that he states emphatically, three package names. He's been around the Capstone possibly more than anyone else, so I'm going to follow this advice! The libraries are quanteda, data.table and sqldf. It happens that I have a lot of SQL in my past, so I am going to use sqldf a lot. They claim that it is fast, LG recommends it and I happen to know the syntax very well. So I'm glad to find out early that such a thing exists.

```
library(data.table)
library(quanteda)
library(sqldf)
library(dplyr)
library(tidyr)
library(ggplot2)
library(lexicon)
```

Open the data and find Descriptive Statistics

I needed to open the files as binary, I needed to use 'skipNul=TRUE' and I needed to pay attention to the locale. There are going to be some issues with character sets and diacritical marks. I am probably going to revisit this after the Milestone Report.
If you look in Code Section 1 in the appendix you will see the steps that I went through in order to build a basic table of descriptive statistics. In addition to word counts and line counts, I used a SQL "GROUP BY" to find the

frequency of words in the three files. I wanted to begin exploring the distributions and start thinking about whether I am going to be able to throw out some data from a "long tail" of words that only occur once. I do not yet know if this is a good idea but the simple table below enabled me to see that the mean of all three of these files consists of words that occur 10, 20 or 30 times. At the high end, it explodes with the basic English stopwords which occur hundreds of thousands of times.

```
##             [,1]      [,2]       [,3]      [,4]        [,5]           [,6]
## names_row   "Lines"   "Words"    "Uniques" "Freq. Min" "Freq. Mean"   "Freq. Max"
## twitter_row "2360148" "30374147" "1290289" "0"         "23.54"        "837023"
## blogs_row   "899288"  "37334650" "1103725" "0"         "33.83"        "1659151"
## news_row    "1010242" "2643986"  "197861"  "1"         "13.36"        "131810"
```

Benchmarking for a certain laptop

I went through a benchmarking exercise in order to get my bearings with how much data I will be able to process on my laptop. This would vary for every setup and is not reproducible since what is being tested is the delay, tailored just to a particular development environment. Speed will be an issue in Shiny, but I'll cross that bridge when I come to it. But I wanted to get calibrated and find out roughly how much I can do. (When you are playing an MP3 while rendering, and the playback of the song turns into vacuum-cleaner/drum'n'bass-style grinding, it's quite terrifying and is an early clue that your memory is approaching the abyss.)
Another reason for doing the benchmarking now, even though it might be used later, is that it is a way of prompting exploratory analysis through trying to accomplish something and seeing how it goes.
During the benchmarking exercise, I sampled different numbers of lines from the Twitter file and recorded the number of seconds' delay. I tried to sample 1 million lines and this was **too much**. Then I tried 100k through 600k, and found that the delay, in seconds, was 44-98-137-202-224-308, respectively. The delay of 5 minutes for 600k is acceptable. Therefore, if other factors will permit this, I am going to try to use a corpus size consisting of 600k lines, or 200k lines from each of the three files.
For the Milestone Report itself, I am going to use much less. In the interest of reproducibility, I will assign 600,000 to a variable and then apply a modifier. In the event that the reader wanted to reproduce this work, you could leave the 600,000 alone and just change the modifier value.

Sampling, generating a corpus

Now that I had a grand total, I created a corpus consisting of equally sized samples from the news, blogs and twitter files. You can refer to Code Section 2 for the steps I went through in R to make this corpus. Once the corpus was created, I saved this off to the disk. The goal is to spare the memory for resource-intensive jobs. After the corpus is saved, this is an opportunity to rerun rm(list=ls()), restart R Studio or reboot the computer if necessary and then be able to read the corpus file back in and continue.

Quanteda: cleaning data and generating n-grams

For most of the types of data cleaning, I did these by setting the arguments in the appropriate Quanteda calls, as you can see in Code Section 3. I am making a couple of concerted design decisions, at least for now. For stopwords, I have read on the boards that you do not necessarily need to remove them. Just because they are common and lack the "color" of concrete nouns and adjectives does not mean that the user does not want

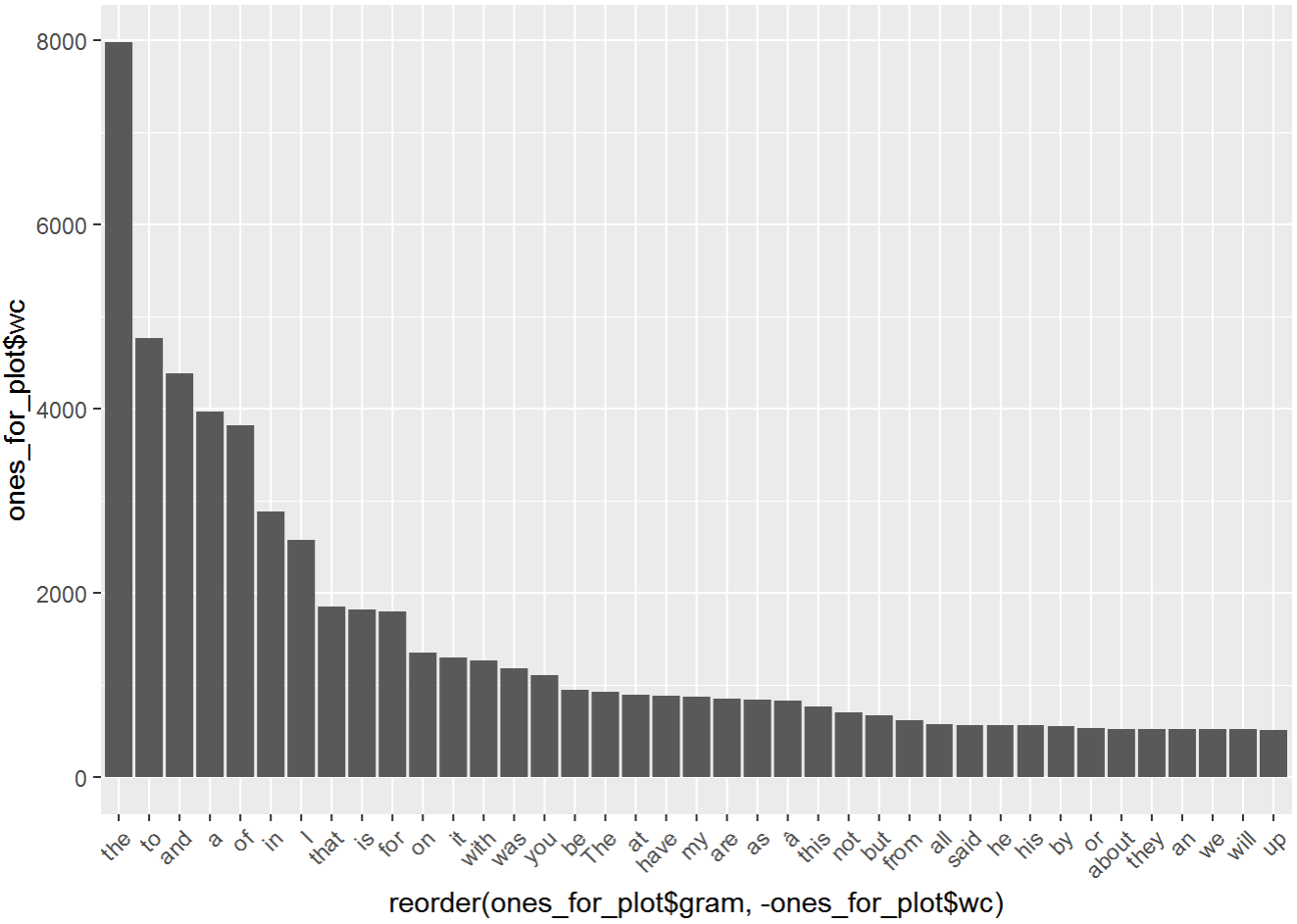
them. I am leaving stopwords in for now.
For the profanity removal, I used sqldf to do this, and I did it slightly later once the n-grams were already generated. I found that the standard behavior for quanteda's stopwords function is that you are using one of the prefab word lists for a language. It wasn't exactly right, so I switched to sqldf because I was taking too much time on one thing. Therefore, my steps in R were:
o Get a profanity list from the lexicon package and cast it as a data.table
o Once the 1-grams, 2-grams and 3-grams are ready, these will also be data.tables
o So now it is easy to write something like "Select from grams where word1 is not in (select word from profanity) and word2 is not in (select word from profanity) and word3 is not in (select word from profanity)"

For all of the rest of the cleanups, I used quanteda's tokenizer. Please refer to Code Section 3 for the R code that I executed to perform the removals and generate n-grams. Note that I followed the same philosophy of getting the large, cumbersome data out of memory and on to disk as soon as possible. This is immediately followed by another rm(list=ls()) and a chance to reboot if necessary.

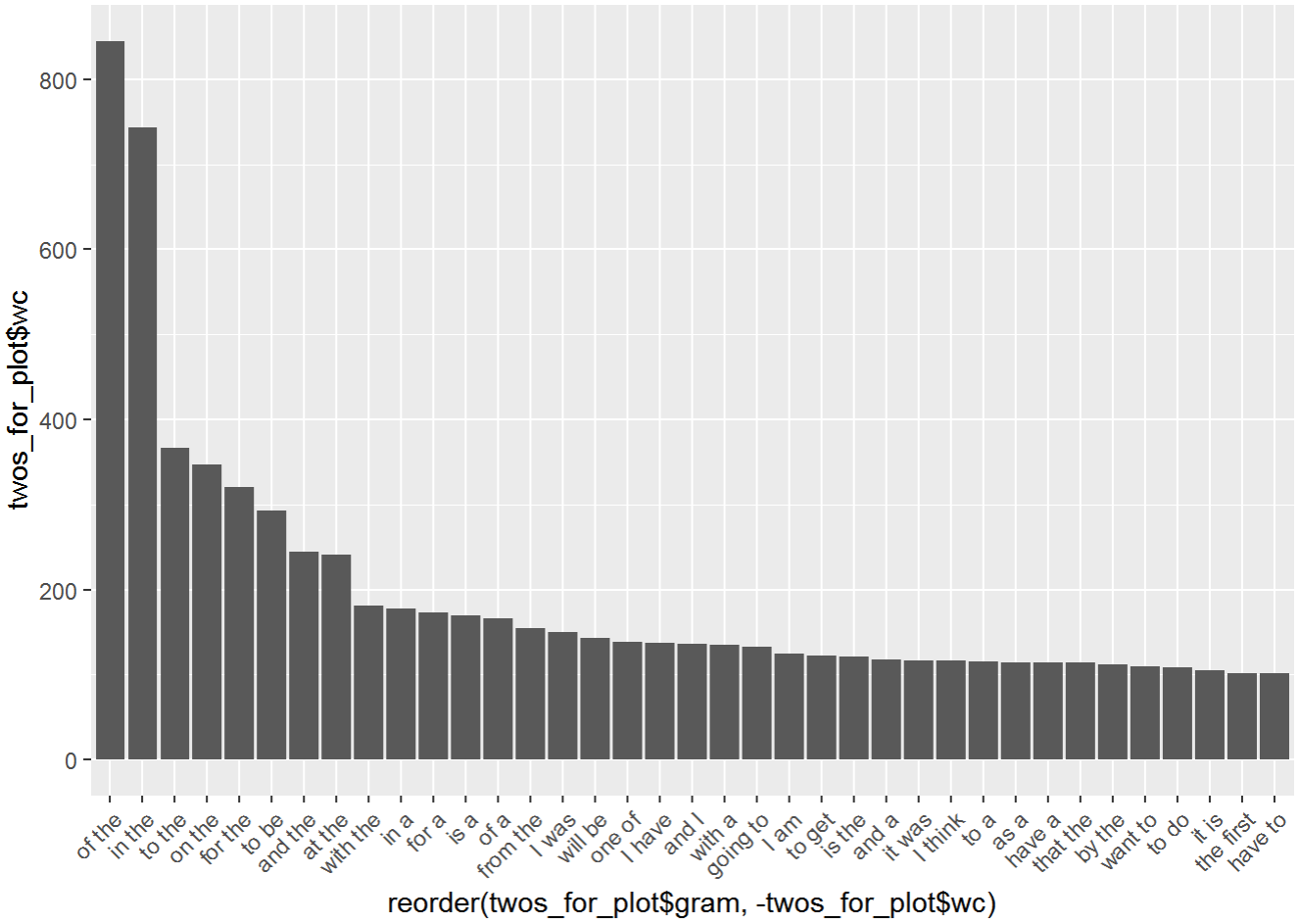
Create n-gram plots

I used ggplot and heavy sqldf, and created plots of the most common n-grams from the three tables. These plots come from a much smaller subset than I hope to use later.
(For this 4th piece of R code related to plotting, I have not called it out to the Appendix because there is not that much, and it is interspersed with the plots themselves. So this code is below.)

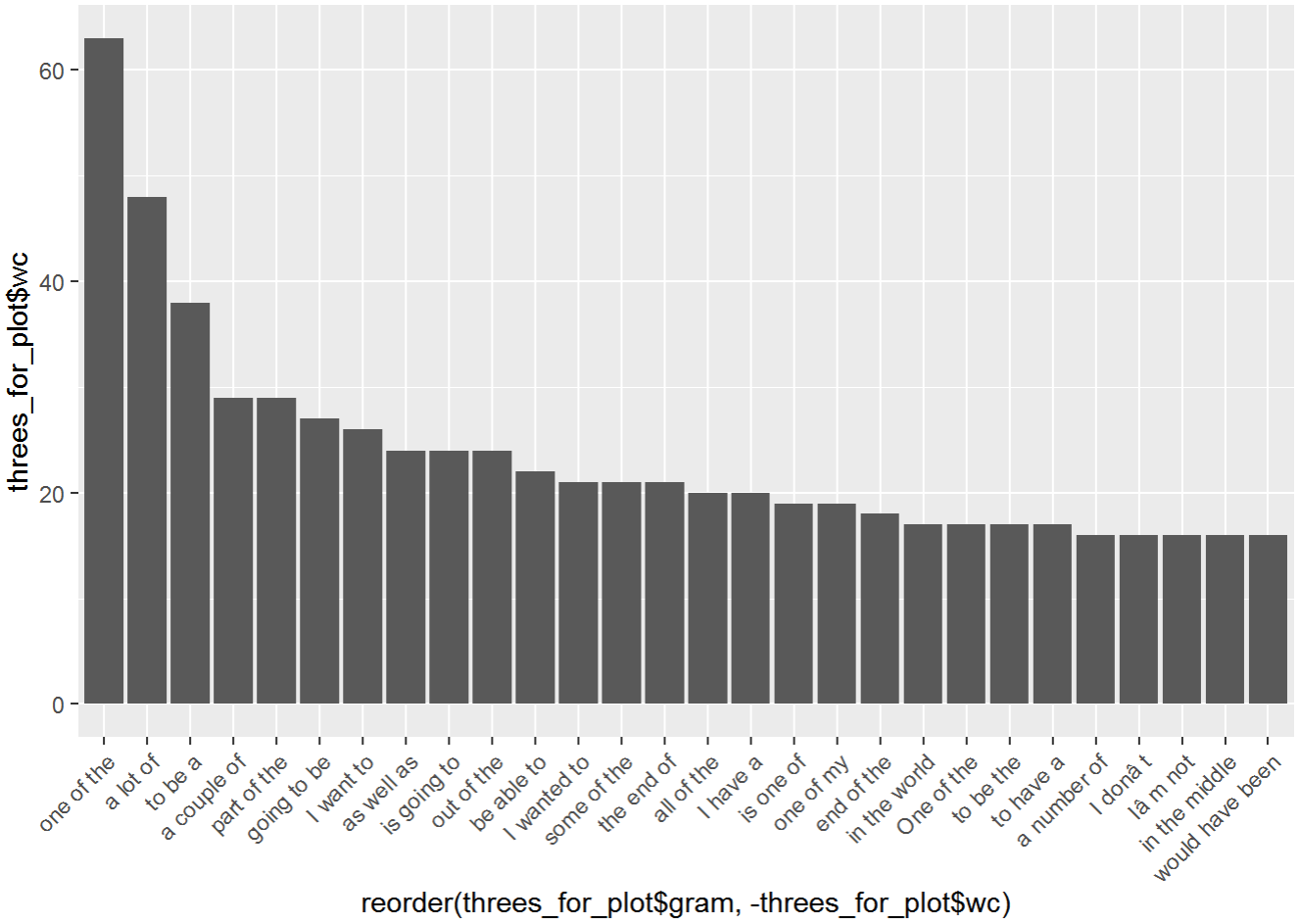
```
rm(list=ls())
ones <- as.data.table(read.table("C:/C2/jhu2018/capstone/ones.dt"))
twos <- as.data.table(read.table("C:/C2/jhu2018/capstone/twos.dt"))
threes <- as.data.table(read.table("C:/C2/jhu2018/capstone/threes.dt"))


onegram_frequency <- sqldf("Select word1 as gram,count(word1) as wc from ones group b
y gram order by wc desc")
twogram_frequency <- sqldf("Select word1||' '||word2 as gram,count(word1||word2) as w
c from twos group by (word1||word2) order by wc desc")
threegram_frequency <- sqldf("Select word1||' '||word2||' '||word3 as gram,count(word
1||word2||word3) as wc from threes group by (word1||word2||word3) order by wc desc")

ones_for_plot <- subset(onegram_frequency,onegram_frequency$wc > 500)
p <- ggplot(ones_for_plot, aes(x = reorder(ones_for_plot$gram,-ones_for_plot$wc), y =
ones_for_plot$wc)) + geom_bar(stat = "identity") + theme(axis.text.x=element_text(ang
le=45, hjust=1))
p
```




```
twos_for_plot <- subset(twogram_frequency,twogram_frequency$wc > 100)
p <- ggplot(twos_for_plot, aes(x = reorder(twos_for_plot$gram,-twos_for_plot$wc), y =
twos_for_plot$wc)) + geom_bar(stat = "identity") + theme(axis.text.x=element_text(ang
le=45, hjust=1))
p
```




```
threes_for_plot <- subset(threegram_frequency,threegram_frequency$wc > 15)
p <- ggplot(threes_for_plot, aes(x = reorder(threes_for_plot$gram,-threes_for_plot$wc
), y = threes_for_plot$wc)) + geom_bar(stat = "identity") + theme(axis.text.x=element
_text(angle=45, hjust=1))
p
```



Future Directions for the Algorithm and Application

I will continue to read the boards and may make changes in direction. I would like to gather your feedback also. As of now, I plan on using sqldf to implement the Backoff method. While I still need to study the theory more, it is going to be a big help on the mechanics that I can represent the necessary selections as a SQL query. For instance:

What is the result count from the 3-grams where word1 = 'pizza' and word2 = 'is'? If the result count is zero, back off to the 2-grams table and pose the same query, maybe just using word1 to find word2.

Also, unless there is an efficiency problem, I plan on using GROUP BY, COUNT and ORDER BY in order to get things sorted in descending order by frequency. This will mean that the most common grams are sitting right there in the first few rows of the query result!

If time permits, I have an idea I would like to implement since the rubric includes a question like "Did this person do something extra or make an effort to do something special?" I wrote about this on the boards a few days ago. When we ask ourselves the question "Does the user want the best and most efficient answer, or do they want something that seems imaginative or fun?" I think it's up in the air and not determined. Therefore, I would like to introduce a little bit of randomness or the possibility of a word suggestion that comes out of left field. Maybe I would let the user toggle whether they want this or not. I have gotten the impression from Greski and others that quality is really not very good, even from Swiftkey. Therefore, I think there is room for this kind of thing - nobody knows what high quality really entails, because the response is going to differ from person to person. This would be the discretionary bit that I will try to do if I can. I would like your remarks on this idea.

Thank you for reading this report.

Appendix

Code Section 1 - Descriptive Statistics

```
f <- file("c:/c2/JHU2018/Capstone/en_US.twitter.txt", open="rb")
nlines <- 0L
while (length(chunk <- readBin(f, "raw", 65536)) > 0) {
   nlines <- nlines + sum(chunk == as.raw(10L))
}
twitter_linecount <- nlines
close(f)
f <- file("c:/c2/JHU2018/Capstone/en_US.blogs.txt", open="rb")
nlines <- 0L
while (length(chunk <- readBin(f, "raw", 65536)) > 0) {
   nlines <- nlines + sum(chunk == as.raw(10L))
}
blogs_linecount <- nlines
close(f)
f <- file("c:/c2/JHU2018/Capstone/en_US.news.txt", open="rb")
nlines <- 0L
while (length(chunk <- readBin(f, "raw", 65536)) > 0) {
   nlines <- nlines + sum(chunk == as.raw(10L))
}
news_linecount <- nlines
close(f)
twitter_linecount <- round(twitter_linecount,0)
blogs_linecount <- round(blogs_linecount,0)
news_linecount <- round(news_linecount,0)
```

```
twitter_words <- scan("c:/c2/JHU2018/Capstone/en_US.twitter.txt", quote=NULL, what="x
",skipNul=TRUE)
blogs_words <- scan("c:/c2/JHU2018/Capstone/en_US.blogs.txt", quote=NULL, what="x",sk
ipNul=TRUE)
news_words <- scan("c:/c2/JHU2018/Capstone/en_US.news.txt", quote=NULL, what="x",skip
Nul=TRUE)
twitter_wordcount <- length(twitter_words)
blogs_wordcount <- length(blogs_words)
news_wordcount <- length(news_words)
twitter_words_dt <- as.data.table(twitter_words)
blogs_words_dt <- as.data.table(blogs_words)
news_words_dt <- as.data.table(news_words)
twitter_wordfrequency <- sqldf("Select twitter_words,count(twitter_words) as wc from
twitter_words_dt group by twitter_words order by wc desc")
blogs_wordfrequency <- sqldf("Select blogs_words,count(blogs_words) as wc from blogs_
words_dt group by blogs_words order by wc desc")
news_wordfrequency <- sqldf("Select news_words,count(news_words) as wc from news_word
s_dt group by news_words order by wc desc")

twitter_uniques <- dim(twitter_wordfrequency)[1]
blogs_uniques <- dim(blogs_wordfrequency)[1]
news_uniques <- dim(news_wordfrequency)[1]
twitter_uniques_min <- min(twitter_wordfrequency$wc)
blogs_uniques_min <- min(blogs_wordfrequency$wc)
news_uniques_min <- min(news_wordfrequency$wc)
twitter_uniques_mean <- mean(twitter_wordfrequency$wc)
blogs_uniques_mean <- mean(blogs_wordfrequency$wc)
news_uniques_mean <- mean(news_wordfrequency$wc)
twitter_uniques_max <- max(twitter_wordfrequency$wc)
blogs_uniques_max <- max(blogs_wordfrequency$wc)
news_uniques_max <- max(news_wordfrequency$wc)
twitter_row <- round(c(twitter_linecount,twitter_wordcount,twitter_uniques,twitter_un
iques_min,twitter_uniques_mean,twitter_uniques_max),2)
blogs_row <- round(c(blogs_linecount,blogs_wordcount,blogs_uniques,blogs_uniques_min,
blogs_uniques_mean,blogs_uniques_max),2)
news_row <- round(c(news_linecount,news_wordcount,news_uniques,news_uniques_min,news_
uniques_mean,news_uniques_max),2)
names_row <- c('Lines','Words','Uniques','Freq. Min','Freq. Mean', 'Freq. Max')
rbind(names_row,twitter_row,blogs_row,news_row)
```

Code Section 2 - Corpus

```
corpus_size <- 600000
milestone_modifier <- (1/100)
```

```
milestone_corpus_size <- corpus_size * milestone_modifier
milestone_twitter_sample_size = round(milestone_corpus_size/3,0)
milestone_blogs_sample_size = round(milestone_corpus_size/3,0)
milestone_news_sample_size = round(milestone_corpus_size/3,0)

handle_to_news <- file("c:/c2/JHU2018/Capstone/en_US.news.txt", open="rb")
news <- readLines(handle_to_news,skipNul=TRUE)
handle_to_blogs <- file("c:/c2/JHU2018/Capstone/en_US.blogs.txt", open="rb")
blogs <- readLines(handle_to_blogs,skipNul=TRUE)
handle_to_twitter <- file("c:/c2/JHU2018/Capstone/en_US.twitter.txt", open="rb")
twitter <- readLines(handle_to_twitter,skipNul=TRUE)
close(handle_to_twitter)
close(handle_to_blogs)
close(handle_to_news)
sampletwitter <- sample(twitter,milestone_twitter_sample_size)
sampleblogs <- sample(blogs,milestone_blogs_sample_size)
samplenews <- sample(news,milestone_news_sample_size)
corpus <- c(samplenews,sampleblogs,sampletwitter)
writeLines(corpus,"c:/c2/JHU2018/capstone/corpus.txt")
```

Code Section 3 - Cleanup and n-grams

```
separatorstring_unescaped <- "{OBSCUREDELIMITER}"
separatorstring_escaped <- "\\{OBSCUREDELIMITER\\}"
ones <- tokens(corpus,what="word", remove_numbers=TRUE,remove_punct=TRUE,remove_symbo
ls=TRUE,remove_separators=TRUE,remove_twitter=TRUE,remove_hyphens=TRUE,remove_url=TRU
E, ngrams=1L, concatenator=separatorstring_unescaped)
ones <- as.character(ones)
ones <- as.data.table(ones)
twos <- tokens(corpus,what="word", remove_numbers=TRUE,remove_punct=TRUE,remove_symbo
ls=TRUE,remove_separators=TRUE,remove_twitter=TRUE,remove_hyphens=TRUE,remove_url=TRU
E, ngrams=2L, concatenator=separatorstring_unescaped)
twos <- as.character(twos)
twos <- as.data.table(twos)
threes <- tokens(corpus,what="word", remove_numbers=TRUE,remove_punct=TRUE,remove_sym
bols=TRUE,remove_separators=TRUE,remove_twitter=TRUE,remove_hyphens=TRUE,remove_url=T
RUE, ngrams=3L, concatenator=separatorstring_unescaped)
threes <- as.character(threes)
threes <- as.data.table(threes)
ones_dt <- as.data.table(ones) %>% separate(ones,c('word1'),sep=separatorstring_escap
ed)
twos_dt <- as.data.table(twos) %>% separate(twos,c('word1','word2'),sep=separatorstri
ng_escaped)
threes_dt <- as.data.table(threes) %>% separate(threes,c('word1','word2','word3'),sep
=separatorstring_escaped)
pza <- as.data.table(profanity_zac_anger)
ones_dt <- sqldf("Select word1 from ones_dt where word1 not in (select * from pza)")
twos_dt <- sqldf("Select word1,word2 from twos_dt where word1 not in (select * from p
za) and word2 not in (select * from pza)")
threes_dt <- sqldf("Select word1,word2,word3 from threes_dt where word1 not in (selec
t * from pza) and word2 not in (select * from pza) and word3 not in (select * from pz
a)")
write.table(ones_dt,"C:/c2/jhu2018/capstone/ones.dt")
write.table(twos_dt,"C:/c2/jhu2018/capstone/twos.dt")
write.table(threes_dt,"C:/c2/jhu2018/capstone/threes.dt")
```