Milestone Report

John Klapp

December 18, 2019

Week 2 Milestone Report

This milestone report described both what was done through the Week 1 and 2 work, and planning done towards the rest of the project.

Week 1 Task 0: Understanding the problem

In short, the objective in this capstone project is to create a predictive text algorithm. The objectives are very open-ended, as is targeted harware. Since I'm using largerly my work laptop to do this, and it is severely limited by both low RAM and an aggressive anti-virus software,

Week 2 Task 1: Getting and Cleaining the Data

In the interest of efficiency, and the philosophy of using the best tool for the job, the zip file was downloaded with wget, and unzipped using the OS native tool. From there, the three english files were evaluated using the code block below.

```
scanTwitter = function() {
  con <- file("en_US.twitter.txt", "r")
  numLines = as.integer(0)
  maxLength = as.integer(0)
  hasHate = as.integer(0)
  hasLove = as.integer(0)
  hasPhrase = as.integer(0)

  while ( TRUE ) {
    line = readLines(con, n = 1)
    if ( length(line) == 0 ) {
      break
    }
    maxLength = max(maxLength,nchar(line))
    numLines = numLines + 1
    hasHate = hasHate + grepl("hate", line)
    hasLove = hasLove + grepl("love", line)
    hasPhrase = hasPhrase + grepl("A computer once beat me at chess, but it was no match
for me at kickboxing",line)

    if(grepl("biostats",line)) {print(line)}
    #if (numLines > 1000) {break} # This just to read the first thousand to test
  }
  close(con)
}
```

This code gave the following results:

Week 2 Task 2: Exploratory Data Analysis

This task was interleaved with task 1; the simple analysis above was acomplished before much data cleaning. Unsurprisingly, Twitter has the shortest lines due to the charachter limit. That the longest string of text was a blog post is also unsurprising; most blogs have neither an editor nor the requirement to fit in print. Both Twitter and the blogs had a love/hate ratio above 4, but the news is much less inclined to use the word “love” with a love/hate ratio of 3.4.

	Twitter	News	Blogs
maxLength	213	5760	40835
NumLines	2360148	77259	899288
hasHate	22138	322	11098
hasLove	90956	1105	49167

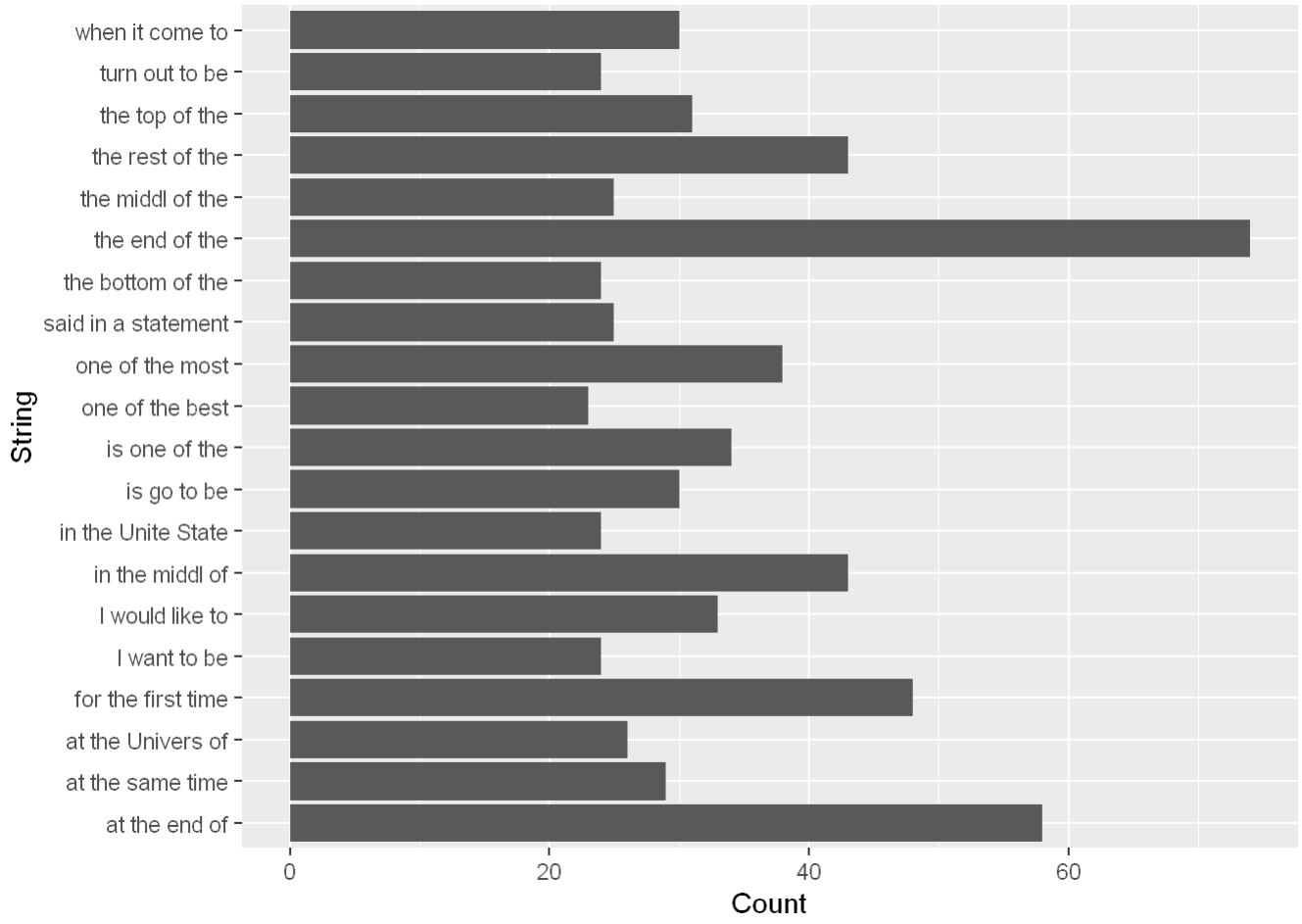
Week 2 Task 3: Modeling

The major objective of this task was to evaluate and explore ngram, the sequences of words in these collections of text. Many examples of the *tm* library include removing words like *the* and *and* as they don’t contribute largely to natural language processing. I chose to include them, as they are used in speech.

A challenge is that the computer I’m using can’t reasonably manipulate the entire dataset at once, so any word cleaning required (in R) copying the entire dataset, which in turn required a few gigabytes of swapping (pushing memory to/from the hard drive) and it died. So, for this, I chose to grab only 10% of the data to do the cleaning and extracting the ngrams (phrases) and count those.

```
ngramTokenizer <- function(theCorpus, ngramCount) {
  ngramFunction <- NGramTokenizer(theCorpus,
                                  Weka_control(min = ngramCount, max = ngramCount,
                                               delimiters = " \\r\\n\\t.,;:\"()?!"))
  ngramFunction <- data.frame(table(ngramFunction))
  ngramFunction <- ngramFunction[order(ngramFunction$Freq,
                                       decreasing = TRUE),]
  colnames(ngramFunction) <- c("String","Count")
  ngramFunction
}
```

There are some deliberate decisions in here. One mistake I made was with the recursive algorithm not using the results of the previous level, but that wasn’t, strangely, significantly expensive. With sampling 10% of the corupus, I got each of the 4 levels to run in about 5 minutes with minimal swapping. For an example of the results, the table below shows the 20 most common four word strings (the sorting is the order of first appearance). From this start, it appears that the decision to keep words like “the” and “of” in the corpus is validated, as those words appear very often in the language.



Way Forward

My planned way forward is to develop a shiny app that takes user input from a text box, and looks for the most likely next word. Below are the big pieces I'm thinking about:

- First pass of the algorithm waits for a space bar to guess at next word
- When a 3 word string doesn't strongly predict the following word, using only the last 2 words to predict the next word
- Once those are working, then extend the algorithm to pick the most likely next word as the next word is being typed (match the letters as they're typed)

I appreciate your thoughts and feedback.