Stat243: Problem Set 3, Due Friday Oct. 2, 10 am

September 22, 2020

This covers the first half of Unit 5.
It's due **as PDF submitted to Gradescope** and submitted via GitHub at 10 am on Oct. 2.
Comments:

1. The formatting requirements are the same as previous problem sets.
2. Please note my comments in the syllabus about when to ask for help and about working together. In particular, **please give the names of any other students that you worked with on the problem set and indicate in comments any ideas or code you borrowed from another student.**

Problems

1. The goal of Problem 1 is two-fold: first to give you practice with regular expressions and string processing and the second (more important) to have you thinking about writing well-structured, readable code. Regarding the latter, please focus your attention on writing short, modular functions that operate in a vectorized manner and also making use of *apply()*/*lapply()*/*sapply()* to apply functions to your data structures. Think carefully about how to structure your objects to store the debate information. You might have each candidate's response to a question be an element in a character vector or in a list.

   The website Commission on Presidential Debates has the text from recent debates between the candidates for President of the United States. (As a bit of background for those of you not familiar with the US political system, there are usually three debates between the Republican and Democratic candidates at which they are asked questions so that US voters can determine which candidate they would like to vote for.) Your task is to process the information and produce data on the debates. Note that while I present the problem below as subparts (a)-(f), your solution does not need to be divided into subparts in the same way, but you do need to make clear in your solution where and how you are doing what. Your solution should do all of the downloading and processing from within R so that your operations are self-contained and reproducible. For the purposes of this problem, please work on the **first** of the three debates from each of the years 2000, 2004, 2008, 2012, 2016. I'll call each individual response by a candidate to a question a "chunk". A chunk might just be a few words or might be multiple paragraphs. The result of all of this activity in parts (a)-(d) should be well-structured data object(s) containing the information about the debates and candidates.

   Given that in earlier problem sets, you already worked on downloading and processing HTML, I'm giving you the code (in the file *ps/ps3prob1.R*) to download the HTML and do some initial processing, so you can dive right into processing the actual debate text.

(a) Convert the text so that for each debate, the spoken words are split up into individual chunks of text spoken by each speaker (including the moderator). If there are two chunks in a row spoken by a candidate, it's best to combine them into a single chunk. Make sure that any formatting and non-spoken text (e.g., the tags for 'Laughter' and 'Applause') is stripped out. You should create some sort of metadata or attributes so that you can easily extract only the chunks for one candidate in later processing. You may need to do some looping as you manipulate the text to get the chunks, but try to do as much as possible in a vectorized way. Please print out or plot the number of chunks for the candidates.
Note: The 2008 McCain-Obama first debate has two copies of the same text. Just clean that up in whatever way you find easiest.

(b) Use regular expression processing to extract the sentences and individual words as character vectors, one element per sentence and one element per word.

(c) For each candidate, for each debate, count the number of words and characters and compute the average word length for each candidate. Store this information in an R data structure and make a plot of the word length for the candidates. Comment briefly on the results.

(d) For each candidate, count the following words or word stems and store in an R data structure: I, we, America{,n}, democra{cy,tic}, republic, Democrat{,ic}, Republican, free{,dom}, war, God [not including God bless], God Bless, {Jesus, Christ, Christian}. Make a plot or two and comment briefly on the results.

(e) Please include unit tests for the functions that do your regular expression processing (i.e., the functions that extract sentences, words, and interesting content words. For the sake of time, you can keep this to a small number of tests for each function.

(f) (Extra credit) We may give extra credit for particularly nice solutions.

Hint: Depending on how your process the text, you may end up with lists for which the name of a list element is very long. Syntax such as `names(myObj) <- NULL` may be helpful.

2. This problem asks you to design an object-oriented programming (OOP) approach to the debate text analysis of problem 2. You don't need to code anything up, but rather to decide what fields and methods you would create for a class that represents a debate, with additional class(es) as needed to represent the spoken chunks and metadata on the chunks for the candidates. The methods should include methods that take input text and create the fields in the classes. You can think of this in the context of R6 classes, or in the context of classes in an object-oriented language like Python or C++. To be clear, you do **not** have to write any of the code for the methods nor even formal code for the class structure; the idea is to design the formats of the classes. Basically if you were to redesign your functional programming code from problem 2 to use OOP, how would you design it? As your response, for each class, please provide a bulleted list of methods and bulleted list of fields and for each item briefly comment what the purpose is. Also note how each method uses other fields or methods.

3. This problem provides practice in hacking the features of a languge to do something that was not really intended by the developers of the language.
Suppose I want to be lazy and when I type "q" in the R or RStudio console, R or RStudio should quit without asking any further questions. Write a bit of R code to achieve this.
Hints: (a) What function call, with what arguments, would I need to run to get R to quit without any questions? (b) If you type "q", what sequence of steps are carried out by R? How can I get something different (namely the code in (a)) to be run?