STA 141C - HW 2 - Text Processing

Version 1.3

- See Canvas for due dates and grading rubric.
- Turn in a neatly typed report that answers the questions in clear English, using complete sentences. The written part should be 2-4 pages. Use a separate page for the results in question 3.
- Attach your code in an appendix. This code must run, and support your answers.

It will be easier to do this assignment in R. If you understand how R's `parLapply`, `parLapplyLB` functions work, and can find equivalent versions in another language, then you are welcome to use another language.

Overview

In this assignment we will transform raw text into a term document matrix. We can use this transformed data for statistical analysis, say clustering the agencies based on what they actually spend their money on.

We'll learn about:

- text processing
- parallelism using multiple processors

Data

The data is the same as for homework 1, and is available at http://anson.ucdavis.edu/~clarkf/sta141c/ see `awards.zip`. Look up the meanings of the data and fields at https://www.usaspending.gov/#/.

Details

The `description` column contains a text description of what the award was for. Let's use these words to describe each agency.

One idea is to weight each word using the amount of the award. To use an example from the bikes data, one award is for $5295 for:

```
TANDEM BICYCLE FOR EXERCISE AND PLEASURE.
```

There are 6 words, so we can assign each word a weight of 5295/6. This gives equal weight to the common words `FOR` and `AND`. We don't want to do this, because these common words appear everywhere, so they aren't interesting. Dropping the common words and punctuation gives us:

```
TANDEM BICYCLE EXERCISE PLEASURE
```

There are now 4 words, so we can assign each a weight of 5295/4.

Another description is for $3319 and contains:

```
BICYCLES FOR MENTAL HEALTH
```

We would like the singular word `bicycle` and the plural word `bicycles` to map to the same word, because they are the same concept. This procedure is called 'stemming'. We'll use some convenient off the shelf packages for it: the R packages tm and SnowBallC.

These software packages map both singular and plural words to the stem `bicycl`:

```
> tm::stemDocument(c("bicycle", "bicycles"))
[1] "bicycl" "bicycl"
```

After we stem and remove the stop words, the string becomes a three word string with a weight of 3319.

```
BICYCL MENTAL HEALTH
```

If our data only consists of these two rows, then we should produce the following table:

```
word    | weight
------- | --------
bicycl  | 5294 / 4 + 3319 / 3
mental  | 3319 / 3
health  | 3319 / 3
tandem  | 5294 / 4
exercis | 5294 / 4
pleasur | 5294 / 4
```

Hints

Here are some suggestions for text preprocessing steps, in no particular order. You may also use any other reasonable string transformations.

- removing punctuation
- stemming words
- changing to upper / lower case
- unicode / ASCII conversions
- removing 'stop words' that appear frequently and add little meaning

Questions

1. Write a function to compute the top `n = 25` most heavily weighted words for each agency. Describe your approach to text preprocessing. What steps did you take in what order? Why?

2. Use this function to process the 91 agencies that have file sizes between 1 MB and 50 MB. (Recall 1 MB = 2^20 = 1048576 bytes) Examine your result and use it to improve the function you wrote above. What did you have to change as you looked at the result?
3. Show your results in a table for the following agencies:
   - National Science Foundation (655)
   - Federal Bureau of Investigation (262)
   - U.S. Customs and Border Protection (778)
   - Forest Service (110)
4. For the agencies listed above, are the results consistent with the name of the agency? For example, do the words associated with the National Science Foundation seem related to science? Did you notice any strange results in these or in any other agencies?
5. Make your program parallel on your local machine with `parallel::parLapply` and with `parallel::parLapplyLB`. Use 2 or more processes on your local machine and time both results. What do these functions do? Are they faster than the serial version of `lapply`? Which is fastest?