Lab #2: Data Preprocessing and Term Frequencies

Due in Sakai February 8, 2019 (by 10:00a)

Introduction

The purpose of this lab is twofold. First, you will practice turning a column vector of text documents into a corpus, preprocessing it, and turning it into a document-term matrix (DTM). Second, you will run some of your own basic term frequency analyses.

Getting Started

First, set your working directory. Open up a fresh RStudio session (if you open RStudio and a previous work session is loaded, be sure you save any R and/or RData files before you close them out). Then set your working directory:

```
setwd("working_directory_here")
```

Then open up a fresh R script. Save it using your first initial, full last name, and then "_lab2." So my script would be titled **MTaylor_lab2.R**.

You should always strive to keep your scripts tidy. At the top of your R script, type this (substituting in your first initial and last name):

```
##############################
##  MTaylor_lab2.R
##  Note: Code for lab assignment #2
##  Author: Marshall A. Taylor
##############################

###BEGIN###
```

You are ready to begin your code. Be sure to include all the code necessary for me to check your work. Document your code thoroughly (using "#").

Remember that saving your R work is a two-step process. You save your R script using Cmd+Enter (Mac) or Cntrl+Enter (Windows). You save your R data objects like this:

```
save.image("MTaylor_lab2.RData")
```

Lastly, prep a Word, Pages, or LaTeX document that has the same title structure: e.g., **MTaylor_lab2.docx**. This is where you put your write-ups and visualizations.

You should turn in three documents to Sakai: your **R script** that shows the code you used, **RData file** that provides the data your scraped, and **Word document** (or whatever text processor you choose to use) with your write-ups.

Assignment

Your task is to take your column vector of texts from lab #1, turn it into a corpus, preprocess it, make a DTM, and generate some basic term frequencies. Please complete each of the following sections.

Section #1: Prepare a DTM

First, complete the steps necessary to turn your texts into a DTM. **Note to Twitter data users: Take a look at the "hints" section before you begin the preprocessing steps. There are some unique challenges with your data that you'll need to address.**

1. First, in your Word document, remind me what your data are. Provide 3-5 sentences telling me why you chose to scrape what you scraped and what is interesting about it.
2. Convert your column vector of texts from your scraped data into a corpus.
3. Using the `inspect()` function, copy and paste the first 5 texts (tweets, article snippets, or Amazon reviews, depending on where your data from last week came from) into your Word document.
4. Remove punctuation from your corpus. Copy and paste the first 5 texts into the Word document again.
5. Remove capitalization from your corpus. Copy and paste the first 5 texts into the Word document again.
6. Remove numbers from your corpus. Copy and paste the first 5 texts into the Word document again.
7. Remove English stop words. Copy and paste the first 5 texts into the Word document again.
8. Stem the texts. Copy and paste the first 5 texts into the Word document again.
9. Remove excess whitespace. Copy and paste the first 5 texts into the Word document again.
10. Compare your unprocessed texts to your preprocessed texts. Why is it necessary to preprocess texts like this? Can you think of particular situations where you would *not* want to apply certain preprocessors to your texts?
11. The order in which an analyst applies preprocessors to their texts is very important. What would happen if, say, the analyst tried to stem the terms *before* removing punctuation?
12. Create your DTM. Take a look at a sample of the DTM using the `inspect()` function, and copy and paste the matrix sample to your Word document. What does this DTM tell us about the data?
13. **Extra Credit:** Using the `inspect()` and `removeSparseTerms()` functions, take a look at what your DTM would look like with the following four sparsity factors: .95, .9, .5, and .1 How many terms are retained in the DTM at each sparsity level? What is the sparsity level (in %) at each level? What does the sparsity level mean? How do the sparsity factors different from one another—that is, in what way is a .95 sparsity factor different from a .5 sparsity factor?

Section #2: Term Frequencies

Now you are ready to analyze some term frequencies.

1. What are the 10 most frequent terms in your DTM? Copy and paste them into the Word document.
2. What are the 10 least frequent terms in your DTM? Copy and paste them into the Word document.
3. What are the terms that occur at least 5 times? Copy and paste them into the Word document.
4. Create a bar graph that visualizes the frequency distribution of the top 10 most frequent terms. Save it as a PDF and put it in your Word document. Provide 3-5 sentences explaining what the bar graph is telling us.
5. Select three of the top 10 most frequent terms. What are the terms that correlate with each term at the .5 level? Copy and paste each of the three sets of "correlated" terms into your Word document. How do you interpret these correlations?
6. Are there any correlations that you find particularly interesting? Why?

7. **Extra Credit:** Create a new DTM where the cells are weighted by tf-idf scores instead of term frequencies. Now select four documents that span across the date range of your data. Create four bar graphs—one for each document—that show the top 5 or 10 terms with the highest tf-idf scores. How do tf-idf scores differ from term frequencies, and what do they tell us about these four documents?

Hints

General Hint

The R scripts for the preprocessing and term frequency slides (pres31.R and pres32.R) might be very helpful. Like, *really really helpful*. Especially for creating the tf-idf bar graphs.

Hint for Twitter Data Users

Note that if you are using Twitter data, you have a few extra preprocessing challenges. You probably want to remove some characters you don't want. After creating your corpus but *before* preprocessing (starting at step #4 in section #1), you will want to remove URLs, user names, and special characters. You can do that with the following (the `removeURL` bit of code is adapted from here, the `userNames` bit from here, and the `removeSpecial` from here):

```
removeURL <- function(x) gsub("http[^[:space:]]*", "", x)
corpus <- tm_map(corpus, removeURL)

userNames <- function(x) gsub("\\B@\\w+", "", x)
corpus <- tm_map(corpus, userNames)

removeSpecial <- function(x) gsub("[^\x20-\x7E]", "", x)
corpus <- tm_map(corpus, removeSpecial)

corpus <- tm_map(corpus, PlainTextDocument)
```

You are now ready to apply the preprocessors as we did in class. You have one final option, though, that you may want to consider since you're using Twitter data: retaining hashtags, but removing all other punctuation. Instead of using `removePunctuation` in the `tm_map` function, you can instead create your own preprocessor that removes all punctuation *except* the hashtag. You can do this with the following, where I create a preprocessor called `removeMostPunctuation` (code adapted from StackOverflow post here):

```
removeMostPunctuation <-
  function (x, preserve_intra_word_dashes = F)
  {
    rmpunct <- function(x) {
      x <- gsub("#", "\002", x)
      x <- gsub("[[:punct:]]+", "", x)
      gsub("\002", "#", x, fixed = T)
    }
    if (preserve_intra_word_dashes) {
      x <- gsub("(\\w)-(\\w)", "\\1\001\\2", x)
      x <- rmpunct(x)
      gsub("\001", "-", x, fixed = T)
    } else {
      rmpunct(x)
    }
  }
```

Now, when you go to remove punctuation within `tm_map()`, you would use `removeMostPunctuation` instead of `removePunctuation`. **If you choose to use `removeMostPunctuation` instead of `removePunctuation`**, you will need to run `corpus <- tm_map(corpus, PlainTextDocument)` one more time before moving on to the next preprocessor, otherwise you will get an error (essentially, you need to redefine your corpus documents as plain text documents after you use your own user-defined preprocessor function, which I did with `removeURL`, `userNames`, and `removeMostPunctuation`).

Of course, if you want to remove all punctuation including hashtags, then just use `removePuncutation` as normal.