American University - DATA 312

Instructions for Homework 2-A

Unit 2 – Text Mining
Part A - Introduction to text mining

Pre-requisite work:
R and RStudio are installed on your computer.

Make sure that the following R libraries are installed using install.packages("<library>",dependencies=TRUE):

- tidyverse
- **NEW!** tidytext
- **NEW!** gutenbergr

Work Process:
This assignment follows Chapter 1 of *Text mining with R*. You should read that as you work through this assignment, but this assignment is self-contained. The main objective of text mining is to look at the words in a document to figure out its main ideas. Text mining sounds pretty fancy, but ultimately it is rather dumb: you're just looking at how often certain words appear, or which words appear together frequently. This works surprisingly well, actually.

This assignment teaches how to load some text into R, clean it up a bit, and examine *word frequencies*. The frequency of a word is simply the number of times it occurs in a document, although sometimes we divide that count by the total number of words in the document. As you might expect, most authors use certain words more frequently, and the most frequent words tend to be those related to the main theme of the document. For fiction, the main characters' names are usually the most frequent words. For non-fiction, the most frequent words tend to be the subjects being discussed. But as you'll see, there are some wrinkles that need to be smoothed over…

1. Start by clearing your workspace and loading the tidyverse:

```
rm(list=ls())
gc()
library(tidyverse)
```

2. For the Text Mining Module, we also need to load the tidy text mining library:

```
library(tidytext)
```

3. Download the two sonnet text files (`sonnet27.txt` and `sonnet38.txt` both by Shakespeare) from Canvas, and place them in your R session's working directory. To keep this a fun exercise, refrain from reading these two poems until step 12!

4. You can read a text file into R as a variable like this:

```
sonnet27<-read_lines('sonnet27.txt')
```

Note that you may need to move the file to where you're running R, or alternatively, you may need to add some path info to help R find the file. On my computer, for instance, I have to do

```
sonnet27<-read_lines('teaching/stat312/textmining/sonnet27.txt')
```

since that's where the file lives! (Yours will be different!)

5. Have a look at how R has stored sonnet27. Try

```
View(sonnet27)
```

Notice that each line is a separate row. However, if you look in the Global Environment pane (upper right), it's listed as "Values", which means it's not a data frame (table). This makes it hard to work with because although sonnet27 has rows, it really doesn't have columns! Each row is a different number of characters long, and that's problematic.

To fix the sonnet27 so that we can process it, we need to break it up into words. We essentially want each row to be a separate word – so the table is really tall and thin – because then we can add additional columns later to help track properties of the words as they come. Also, the order of the words is (of course) important… and that'll be the order of the rows.

6. The first step is to turn sonnet27 into an actual R table, like so:

```
sonnet27_df <-tibble(text=sonnet27)
```

If you View(sonnet27_df), you'll notice that it's pretty similar to View(sonnet27)… the only thing that's really obviously different is that we've named the single column `text`.

Aside: Quotes in R are tricky… If you look on your keyboard, you probably have three kinds of quotes: ', ", and `. R thinks the first two kinds of quotes do the same thing – though you need to match them, so 'hi' is OK, but 'bye" is not OK. The second kind of quote (usually next to the number 1 on your keyboard, and sometimes called the *backtick*) is special in R. If you have a column name that's more than one word (it has spaces), you need to enclose the whole name in backticks. They crop up in other strange places, too. My advice is that if you need quotes somewhere, start with the usual ' or "… and if R complains, then try the `.

7. The reason why we've tibble-ified sonnet27 is that the next transformation, which puts each word on a separate row, is picky about its input. It only wants tibbles… whatever. So let's do that:

```
sonnet27_tidied <- unnest_tokens(sonnet27_df,word,text)
```

Have a look at the resulting data frame! You'll notice that there's one column, called `word`. If you wanted to call the column something else, you'd replace the `word` in the line above with whatever you wanted to call it. But most of the tidy text mining tools expect `word` as the column of words, so we'll use that.

Note that unnest_tokens did a few other things as well. Can you figure out what these are? (Hint: compare the output with Step 6.) If you want more information on it, type

```
?unnest_tokens
```

and read the documentation!

8. OK, let's count the words! You can do this in several ways depending on what you want to see…

```
count(sonnet27_tidied,word)
```

or try

```
count(sonnet27_tidied,word,sort=TRUE)
```

What is the difference between the output of these two?

Also, it's helpful to use the pipe %>% notation in the future with more complex pipelines… it's not necessary here, but it's a good place to practice. Instead of the above lines, you can try

```
sonnet27_tidied %>% count(word,sort=TRUE)
```

It should give you the same output, because what %>% does is it drops whatever is on the left (sonnet27_tidied in this case) into the first input of whatever is on the right (count(…., word, sort=TRUE)).

This probably seems like a silly trick right now (and you'd be right), but pretty soon you'll see the wisdom of this!

9. Look closely at the word frequencies you just produced. Can you explain why some words are more common? Some of them just aren't very informative, since they're words like "my", "a", and such. Linguists call these "stop words", and we'd like to get rid of them for most of our analysis. Now, an important question is whether you want to get rid of them, or **get rid of them**. Probably not the latter, I'm thinking, because there might be some situations where they're useful. Hold that thought.

One of the things loaded when you brought in the tidytext library was a list of modern English stop words in the table stop_words. Have a look at them! (Hint: View or just say stop_words at the prompt.)

10. Coming back to the task of removing stop words, what we have is a table sonnet27_tidied, in which there is one column called `word`, and many rows, one for each word. We want to remove (temporarily) each row that has a word that's in the stop_words table. If you were writing this in Java or Python, you'd probably use a loop to do that… but R makes this process a snap with something called an *anti-join*. Specifically, an anti-join takes two tables and removes all the rows in the first table according to a matching rule that's built from the second table. This rule matches up two columns – usually called *keys* -- one from each table. (As you might expect, there's also a *join* as well that puts two tables together. We'll use that in HW2-B.)

Aside: if you read the documentation for anti_join, you'll probably find that it's a bit mysterious. That's because anti_join is a generic function, and can do lots of other matching rules, and various other kinds of tricks too! It's very useful!

OK, enough theory. Let's get rid of those stop words!

```
anti_join(sonnet27_tidied, stop_words)
```

does the job… kind of! Why not? Well, it doesn't change sonnet27_tidied, and it just dumped the output to your screen. You could save the output to a new table, but it's often handy not to do this… Here's where the pipe is handy. If you just want word frequencies without the stop words, you want to string the anti_join with count. OK, like this:

```
count(anti_join(sonnet27_tidied,stop_words),word,sort=TRUE)
```

Now if you like parentheses, this is just fine, and you can keep nesting away. But most people start to think of stringing one step after another, and so find the following a bit clearer:

```
sonnet27_tidied %>% anti_join(stop_words) %>% count(word,sort=TRUE)
```

I think I agree that's nicer!

11. Especially… when it comes to plotting, it's handy to use the pipe! Let's turn our textual word frequency list into something more graphical. In HW1-* you learned a little about ggplot2, so let's put it to work:

```
sonnet27_tidied %>%
  anti_join(stop_words) %>%
  count(word)  %>%  # Count makes a new column titled `n` with counts of each word
  mutate(word=reorder(word,n)) %>% # Sorts the list of words using the new column `n`
  filter(n>1) %>%
  ggplot(aes(n,word)) + # Note that this uses the `n` column made by count
  geom_col()
```

Figure out what each step does! (Hint: you can remove individual steps off the end and then investigate the result.)

12. Rerun the whole process (steps 4-11) with the other sonnet, `sonnet38.txt`. My suggestion is that you name the new tables sonnet38, sonnet38_tidied, etc. Notice that the piping idea means that you usually can simply copy the code from before, just changing the input, which is the first line! That's super convenient.

In any case, compare the two word frequency plots, one for each sonnet. What differences can you see?

Oh, and go enjoy the sonnets now :-)

13. Sonnets are fun, but the computer can handle much more! You can read entire books into R and play with them… If you don't know about Project Gutenberg, you should go and have a look at their website now!

14. R seems to have libraries for everything… and as you might expect, there is a library for Project Gutenberg:

```
library(gutenbergr)
```

15. Downloading a book is pretty easy, and the following might just work for you (it didn't work for me because my computer couldn't find the default Gutenberg website)

```
book<-gutenberg_download(1511)
```

If that doesn't work for some reason, you can try another "mirror", which is a website that copies all of Project Gutenberg to give faster access… For instance, you might have to use

```
book<-gutenberg_download(1511,mirror='http://gutenberg.readingroo.ms/')
```

but that ought to work.

16. Now you have a book in a data frame, and you can basically just do what we did with the sonnet:

```
temp_book <- book %>% unnest_tokens(word,text) %>% anti_join(stop_words)
```

so we now have a data frame with just the non-stop words, ready to go.

17. Have a look at our temp_book… What book is it? You should be a little perplexed, because it doesn't look like the text of a book, at least until you remember that books have title pages, headers, and the like. Oops. The title page isn't too helpful!

It's time to do a little data cleaning. Data cleaning is a bit of an art, and it's best to learn it by doing. I personally find that removing the header info in tidy_book is troublesome, because the missing stop words make it hard for me to figure out what's what. Also, the tidying process got rid of all the formatting info that makes it easy to spot the end of the title page and front matter. So… let's go back to the beginning.

Have a look at raw book data:

```
View(book)
```

You should see what actually looks like a digital transcription of a book, complete with title page. And you should be able to identify the book, of course!

Your task: find the row number at the end of the front matter. I think it's around row 47…

18. You can get rid of rows in a dataframe pretty easily by using the slice command:

```
book %>% slice(47:n())
```

This should produce a dataframe with rows 47 through the end of the book (the n() is shorthand for "however many rows there are"… note that this is different from the `n` column that is made by count()).

Tidy this up and remove stop words; store the result in a new variable tidy_book. We'll use that going forward.

19. Have a look at word frequencies:

```
tidy_book %>% count(word,sort=TRUE)
```

but notice that this is a bit weird, because there are some stop words still present!

20. In particular, our stop_words table only has modern English not **early** modern English stop words, so "thee" and "thy" aren't on the list. Fine. Let's make a new table of these kind of words! You can make your own table with the tibble command:

```
tibble(word=c('foo','bar','baaz'))
```

which will make a new table with three rows, and one column. The column is named `word` like in Step 10. Of course, you should pick the actual new stop words you want to remove, and then anti-join that to your tidy_book, perhaps calling this tidier_book.

21. If you've done step 20 correctly, when you run

```
tidier_book %>% count(word,sort=TRUE)
```

your top few words will be the names of the main characters in this book! Neat! Why is that? (Hint: look back at the raw data.)

22. For our final act, let's compare word frequencies between two books. Gutenberg book number 1511 (which we've just been working with) is Shakespeare's *King John*. Let's store our work thus far

```
tidy_john <- tidier_book
```

23. Going to a totally different genre, let's explore

```
book<-gutenberg_download(35937,mirror='http://gutenberg.readingroo.ms/')
```

which is E. Walter Maunder, *Are the Planets Inhabited*? Do a word frequency analysis on this. You should not get character names! Are there **any** similarities?

24. Let's pick something a little closer to Shakespeare so that we can get some comparison.

```
book<-gutenberg_download(1998,mirror='http://gutenberg.readingroo.ms/')
```

Don't peek at the book… just run a word frequency analysis again, and try to figure out who wrote this book from that! (However, **do** use your early Modern English stop words… Don't worry about the frontmatter; it doesn't bias the word frequencies too much.) The book is famous, if only for its colorful character names… Hint: Richard Strass wrote a piece of famous music to accompany it.

OK, now that you've done that, save off the tidied book (ie. after slicing off the header, running unnest_tokens, and anti-joining the stop words) as

```
tidy_other_book
```

25. Let's end with a side-by-side comparison of the word frequencies in these two books. This is a bit of a multi-step process!

We start with the two data frames: tidy_john and tidy_other_book. Each has one column, and many rows. Our goal is a table with three columns: the word in the first column, and then a frequency for that word in each of the two books. If you know any Python or Java, you'll probably be thinking of loops. This being R, we can do it **all at once** without a loop! (R does have loops but they're slower than doing it at once.)

The first step is simply to add a new column to keep track of which book is which. Recall from HW1-A that mutate does this:

```
tidy_john2<-mutate(tidy_john,book="John")

tidy_other2<-mutate(tidy_other_book,book="Other")
```

26. Now let's stack these two guys together:

```
stacked<-bind_rows(tidy_john2,tidy_other2)
```

27. And summarize the word frequencies…

```
counts<-count(stacked,book,word)
```

but the syntax is a little different from above because we want to retain the new book column we made in step 25, so we can tell which word frequency goes with which book!

28. Word frequencies aren't so helpful because one book quite a bit longer than the other. Which one? So lets normalize by the number of words, so that all frequencies are between 0 and 1. This is a little tricky, but R helps us here by providing some useful short hands if you give it a little help. There is a nifty command called group_by that does what the name suggests. Try it:

```
group_by(counts,book)
```

If you look at that using View, it probably looks pretty similar to counts. But, the output of group_by contains some metadata that's not visible with View so that R knows that you think the entries in book (namely "John" and "Other") are groups. You can see this if you use the head() command in the console instead of View. (One thing that's tricky about tidyverse is that many commands have "side effects" that can be hard to see unless you dig a bit.)

In particular, if you say sum(n), this sum only extends over a single group. (If you're confused about where `n` came from, it's the default column name produced by count in Step 27.) That makes it possible to do something like

```
frequencies<-counts %>% group_by(book) %>% mutate(proportion=n/sum(n))
```

which gives you a new column `proportion` that is just what we wanted!

29. Finally, the last magic trick is to transform frequencies (columns: book, word, n, proportion) into what we wanted (columns: word, John, Other). This is called a pivot:

```
final_table <- pivot_wider(frequencies,word,names_from=book,values_from=proportion)
```

30. We did steps 25-29 one by one, saving each intermediate step. You probably don't need them all, and you can save on memory in your computer by piping them together with %>%. You should be able to figure out how to do that! (If you need help, just ask.)

31. OK, let's plot it:

```
final_table %>%
  ggplot(aes(x=John,y=Other)) +
```

geom_text(aes(label=word))

You should get a nice scatter plot! But when I did this, I didn't because everything just piled on top. How do you fix that? Change the formula in Step 29 to proportion=-log10(n/sum(n)) instead of what we did there. This gives you logarithmic scaling!

Redo your plot and look closely at where the words lie. You should be able to explain quite a few of them!

Assignment

For your submission to Canvas, include your .R file.

1. Select two books from Project Gutenberg to analyze. Warning: gutenbergr doesn't work if the book lacks a **plain text** version for the book you want! They can be of any genre. Include the Gutenberg IDs for these two books at the top of your R file in a comment.

2. Determine where the header of each book ends. Note the row numbers in a comment in your R file.

3. Do you need any unusual stop words? Make a claim one way or another, and explain why in a comment. Make a table of additional stop words if you need them.

4. Produce a histogram of the top 10 most frequent words and their frequencies in both books, with stop words and header removed.

5. Make a scatter plot like we do in Step 31 for your two books. Call out five (5) different words on the plot, and explain why they appear where they do in the scatter plot.