American University - DATA 312

Instructions for Homework 2-B

Unit 2 – Text Mining
Part B - Sentiment analysis

Pre-requisite work:
Please complete Homework 2-A before starting this assignment.

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

- tidyverse
- tidytext
- gutenbergr
- **NEW!** textdata
- **NEW!** wordcloud
- **NEW!** reshape2 (should already be installed, but check first!)

Work Process:
For submission, include your .R file .

This assignment follows Chapter 2 of *Text mining with R:*

https://www.tidytextmining.com/sentiment.html

The main objectives are to render word clouds and to do some basic sentiment analysis on text. This assignment also introduces the concept of loops in R, and does a few different joins, anti-joins, and the like. The main idea of sentiment analysis is that certain words have emotional content: positive, negative, or otherwise. If you look over the set of words in a document and compare how frequently these different sentiments appear, you might be able to identify whether the document is tragedy or comedy -- say. As you might imagine, this isn't the end of the story, but it works better than you might expect!

1. First of all, we're going to start with a list of (most of) Shakespeare's works. Since these are on Project Gutenberg, we just need to know which Gutenberg IDs to download. I've done that for you in the `shakespeare_gutenberg.csv` file on Canvas. Download that now, and pull it into R:

```
shakespeare_corpus <- read_csv('shakespeare_gutenberg.csv')
```

This should give you a three column data frame listing each work with its Gutenberg ID. The columns tell you the Gutenberg ID, the Type of work (like "Comedy" or "Tragedy"), and the title.

2. If you recall from HW2-A, we needed to add some Early Modern English stop words... so let's do that.

```
eme_stop_words<-tibble(word=c('thee','thou','hath','thy','thine','ye','tis'))
```

3. We'd like to have all of Shakespeare's works in a single, big data frame for this assignment. That seems daunting, but it's really not too hard for a computer. You just need to have it download each row of the shakespeare_corpus table from Project Gutenberg. You could do that one-by-one. Indeed, let's start by downloading the first one:

```
book <- gutenberg_download(shakespeare_corpus$`Gutenberg ID`[1], mirror='http://gutenberg.readingroo.ms/')  # type it all in one line!
```

4. We can add more books to this list using the add_row function. Have a look at the book table, and remember the number of rows it has. Then try this:

```
book <- book %>%
   add_row(gutenberg_download(shakespeare_corpus$`Gutenberg ID`[2], mirror='http://gutenberg.readingroo.ms/'))
```

Compare the book table to what it was before.

5. You could keep doing this, but that would be boring. Computers don't get bored with repeated tasks! You need a loop! R has loops, and they look like this (which will load all the other books):

```
for(i in 3:nrow(shakespeare_corpus)){  #  We start with 3 since we just loaded the first two above!
 book<-book %>% add_row(gutenberg_download(shakespeare_corpus$`Gutenberg ID`[i], mirror='http://gutenberg.readingroo.ms/'))
}
```

This is a `for` loop…. the way that works is that it repeatedly sets the variable i to be 3, 4, 5, … up to the number of rows in the shakespeare_corpus table. The loop executes the commands between the curly brackets {} with the variable i changing once per value. This might take a minute or two, depending on how slow your network connection is…

6. Dig through the book table now! It's got everything in it!

7. Let's tidy the book table into something like what we used in HW2-A… Use unnest_tokens to tidy the data, and then anti_join to remove stop words:

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

8. It's handy to have an idea of what Type of work is involved, too. Right now, tidy_shakespeare has the Gutenberg ID in a column `gutenberg_id`. On the other hand, we have shakespeare_corpus that tells us the Type. So we need to join these two tables! One small wrinkle is that shakespeare_corpus doesn't have a `gutenberg_id` column… but it does have a `Gutenberg ID` column. R doesn't know that these are the same, but we do, and we can tell it!

```
tidy_shakespeare <- tidy_shakespeare %>%
   inner_join(shakespeare_corpus,by=c(`gutenberg_id`="Gutenberg ID"))
```

9. What are the most common words across all of Shakespeare's works? Dig back into HW2-A to figure out how to do this! (Hint: the answer may surprise you.)

10. Drawing a word cloud is a nice way to show words and their frequencies. It's easy to do once you load the wordcloud library:

```
library(wordcloud)
```

The wordcloud function takes two require inputs: the words and their counts. This is pretty easy if you have everything packed together

```
word_counts <- tidy_shakespeare %>% count(word)
```

This table has two columns: word and n (the counts). You can then say

```
wordcloud(word_counts$word,word_counts$n, max.words=100)
```

Change max.words to be however many words you want to plot. Now, if you're feeling especially tidyverse-y, then there's a slicker way to do this all at once:

```
tidy_shakespeare %>%
  count(word) %>%
  with(wordcloud(word,n,max.words=100))
```

Have a look at the documentation for with() to see what it's up to. It's a neat tool, but a little subtle! The upshot is that with() takes an input (that's the output of our count on the previous line) and then allows you to just refer to columns of that input data... so you can say word instead of whatever_table$word. It's a bit of a shortcut, but makes for easier reading. Maybe.

11. If you're interested in just one play, for instance, you can get it by filtering:

```
tidy_shakespeare %>%
  filter(`gutenberg_id`==1118) %>%  # "Much Ado about Nothing"
  count(word) %>%
  with(wordcloud(word,n,max.words=100))
```

12. Let's pivot to sentiment analysis... The way sentiment analysis works is that there's a list of words, each tagged with a positive or negative "sentiment" (supposed to mean something like emotional content). There are standard lists for this... and the tidytext library has several. For instance

```
bing_sentiments <- get_sentiments("bing")
```

Have a look! (Note: this may require you to accept a license agreement.)

13. We can just inner_join these sentiments into our tidy_shakespeare -- adding in a column for sentiment for each word:

```
tidy_shakespeare %>%
  inner_join(bing_sentiments) %>% # Now we have a `sentiment` column...
```

```
  group_by(Type) %>%
  count(sentiment)
```

With the group_by, this should tell you how many positive or negative sentiment words show up in each type of work.

14. This seems a little hard to visualize, so maybe do a bar plot? Try

```
tidy_shakespeare %>%
  inner_join(bing_sentiments) %>%
  group_by(Type) %>%
  count(sentiment) %>%
  ggplot(aes(x=Type,y=n,fill=sentiment)) +
  geom_col(position='dodge')
```

You might want to change the plotting options at the end. See if you can make a nicer looking plot than I did!

15. This seems like a contingency table situation... STAT-202 maybe? Let's make one!

```
ct <- tidy_shakespeare %>%
  inner_join(bing_sentiments) %>%
  group_by(Type) %>%
  count(sentiment) %>%
  pivot_wider(names_from=c(sentiment), values_from=n)
```

Have a look at ct and compare it with the plot in Step 15 and counts in Step 14 to make sure you understand how the pivot_wider works. It's pretty simple, but worth digging into carefully.

16. Right. Two-way tables mean chi-squared! R can do this easily. Since the chi-squared test is a base R function, there's a small misalignment between tidyverse what the chi-square requires. For a small example (like what we're doing here) you can manually type in the contingency table. Here's how to do this:

```
M<-as.table(rbind(c(10,20),c(30,40)))
dimnames(M)<-list(type=c('Tragedy','Comedy'),sentiment=c('positive','negative'))
```

(You'll have to alter the values in the first line and possibly the labels in the second line to reflect your actual data.)

Once you've done that, then you can say

```
chSq<-chisq.test(M)
```

to run the chi-square test. If you then say

```
chSq
```

You'll get the p-value. (I'd claim that the very small p-value means that there's something statistically significant happening...) If you want a recap of what you sent as input, say

```
chSq$observed
```

just to make sure everything is correct. On the other hand, if you want to see what was expected (based on the marginals) you can say

```
chSq$expected
```

Pretty neat!

Now, you can do this all automatically using the output (the table ct) from your previous analysis in Step 15 -- no manual table entry -- but there is one small irritation. The base R chi-squared test doesn't like the fact that our contingency table ct has the values of the Type variable in the first column. The column_to_rownames() fixes that; we don't want to keep that change, but we can do it temporarily as needed. OK. let's run the test:

```
chisq.test(column_to_rownames(ct,var="Type"))
```

You should compare this to what you got with the chSq above to make sure that everything is indeed correct.

Finally, you might want to see whether there's a specific Type that impacts Sentiment more strongly than expected. While there are special statistical tests called *post hoc* tests -- and it's important to be wary of the p-values these tests produce -- you can get an intuitive idea of what's happening using the *standard residuals* computed as part of the chi squared test. To access them, investigate

```
chSq$stdres
```

This produces a matrix of z- or t-scores, one for each of the entries in your two-way table. Very large (or very negative) values are "unexpected" and therefore are worthy of further investigation!

17. Sentiments need not be simply positive or negative. Some words are stronger than others! Here's a lexicon that gives you some indication of that strength:

```
afinn_sentiments<-get_sentiments("afinn")
```

(Note: this may require you to accept a license agreement.) Notice that there are far fewer words in this lexicon than the previous one...

18. Again, let's join the sentiments in:

```
tidy_shakespeare_sentimental <- tidy_shakespeare %>%
  inner_join(afinn_sentiments) # Now we have a `value` column...
```

19. Since these new sentiments are numerical, we can ask for mean values:

```
tidy_shakespeare_sentimental %>%
```

```
  group_by(Type) %>%
  summarize(value=mean(value))
```

Notice that Shakespeare is not a happy author, on average!

20. It might be nice to see a graphic like the bar chart we made in Step 14. But since we have numerical values, we really need to capture the distribution of values.. I haven't found a plot I really like of this, but here's one option

```
tidy_shakespeare_sentimental %>% ggplot(aes(x=Type,y=value)) + geom_count()
```

Try some of the other possible geom_()'s... perhaps the violin plot looks better to you?

21. Finally, let's use these numerical sentiments to help place words in a word cloud. This is a little complicated because wordcloud() isn't a tidyverse function. However, there is a comparison.cloud function that does the job, once you format its data correctly. The thing it wants is a model that compares word versus sentiment, which can be produced using the special acast() function in the reshape2 library. So...

```
library(reshape2)
```

Next, we need to prep the data for acast(). This is basically what we've done previously:

```
data_for_acast<-tidy_shakespeare %>%
  inner_join(get_sentiments("bing")) %>%
  count(word, sentiment, sort = TRUE)
```

And then here's the call to acast to build the model we need

```
acasted <- data_for_acast %>% acast(word ~ sentiment, value.var = "n", fill = 0)
```

The ~ operator in R usually can be read "versus", so the new data is word versus sentiment. The value.var tells you that the entries in the new data come from the `n` column. (Note the weird quotes. Sorry. R is weird sometimes.) And the fill means that if there's a missing word and/or sentiment to just put in zero.

Now we're ready to plot:

```
acasted %>% comparison.cloud(colors = c("gray60", "gray20"),max.words = 100)
```

Mess with colors or word count if you like! (The default options that I've picked above might not look particularly nice...)

Assignment

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

1. Select two plays: one comedy and one tragedy. They need not be Shakespeare, but they should be available on Project Gutenberg. (Note: you will need to add your own Type column to your data, like what comes from the shakespeare_gutenberg.csv file.)

2. As in HW2-A, determine where the headers end, and remove (slice) the header off. We didn't do that in the instructions above because it would have been annoying to do that to **all** the books! Since you're just looking at two, it's easy enough to do.

3. Plot word clouds for both plays. What are the key words? Why are they important in the play?

4. Can you tell which is play is the comedy using sentiment analysis? Try it, and then explain why or why not? Hint: the chi-square analysis is really effective at answering the second question conclusively.

5. Use either of the sentiment lexicons described above ("bing" or "afinn") and show the distribution of positive/negative words in both works.