Text Analysis

INFO 640 | Pratt Institute | McSweeney

R Markdown

In this tutorial, we are going to cover some basic text analysis using the tm package for R. We will use both pre-made corpora and a collection of documents similar to what you might have as a corpus. This tutorial covers:

1. Cleaning corpora based on the type of analysis you want to do.
2. Supervised Sentiment Analysis
3. Topic Modeling with LDA

```
#to import the books from project gutenberg directly
install.packages("gutenbergr")
#to create and work with corpora
install.packages("tm")
#for LDA topic models
install.packages("topicmodels")
```

```
library(tidytext)
library(tm)
library(stringr)

library(topicmodels)
library(gutenbergr)

library(tidyverse)
library(dplyr)
library(data.table)
```

We will analyze both unstructured and structured data: books and plain text documents are unstructured whereas csv's of many documents are structured. You will see that the approaches are conceptually very similar, though typically these different formats arise in different domains.

Direct Text

Unstructured text is books and other plain text formats. Typically you would use approaches like this for understanding books, long form articles, and some speeches, doing genre classification, or "distant reading". Most Humanities applications of text analysis start with unstructured text data.

First let's read in a poem by Emily Dickenson. We'll use this test text to illustrate some of the possible functions in R. Generally, you won't read text in this way, but sometimes it can help to read a string directly so you can see the immediate changes.

```
text <- "Because I could not stop for         Death -
          He kindly stopped for me -
          The Carriage held but just Ourselves -
          and Immortality")
text
```

Try out some of the cleaning functions.

We'll make everything lowercase, remove any punctuation, and remove extraneous whitespace. Note that this may not actually be appropriate for poetry (or short form text) since punctuation and capitalization and even whitespace can have meaning. But let's just go with it for this example – say we're interested in word use rather than overall meaning.

```
tolower(text)

removePunctuation(text)

stripWhitespace(text)
text
```

Note that none of these transformations have been saved, to save the transformations, you will have to reassign them to a variable. We will do this in the next section.

More advanced cleaning (stemming, lemmatizing, etc.) can be done with qdap package. For more on qdap, install the package with install.packages("qdap") and then inspect it with ??qdap

Now we will move on to stopwords. Stopwords are words that are exceedingly common, such as determiners, prepositions, pronouns, auxiliaries, etc. "the, a, of, in, he, she, is, have". The tm package has some built in stopwords.

You can inspect different stopwords lists with the 2 letter abbreviation for a language. This 2 letter abbreviation is an ISO code.

```
stopwords("en")
```

```
removeWords(text, stopwords("en"))
```

Sometimes you will need to add words to your stopwords list. Maybe you are reviewing tweets about Uber, or newspaper articles about Trump, in each of these cases, you'd need to remove the target because it will be too prominent in your corpus (there are more sophisticated ways to do this, but we'll set that aside for today).

```
my_stops <- c(stopwords("en"), "death")
```

What do you notice? "He" and "Because" weren't removed - why? Hint: the stopwords list is all lowercase.

Exercise 1 (solutions at the end of this document):

1. Transform text to lowercase and save into a new variable (call it text_new)
2. Strip the whitespace and remove the punctuation.
3. Add "death" to the stopwords list
4. Remove the stopwords from text_new
5. Print text_new to the screen.

```
#your code
```

If we had wanted to add the word "stop" to our stopwords list, we would have had a problem - "stop" appears in 2 forms in our poem: "stop" and "stopped." When we are interested in meaning (rather than tense, genre, or inflection), we want to treat these as the same word. This is called stemming.

When you stem your corpus (the collection of document(2)), recompiling these words involves a bit of a trick.

We will use the stemCompletion function, and pass it our stemmed text and the original list of separated words. This ensures that every word in our stemmed corpus has a correlate in the lookup dictionary. As you will see, the completion function maps back to the same, most basic word form. After running the following commands, look closely at what happened to "stop" and "stopped"

First, spilt the new_text on the spaces, then turn it from a list to a vector and assign it to a new variable.

Then stem the items in this new variable using stemDocument(). Print it to the screen to inspect what happened. Then make a new variable, completed_text by passing your vector of stemmed words (stem_text) and the "dictionary" for lookup (nvec). Print this to the screen to inspect what you get.

```
nvec <- unlist(strsplit(new_text, split=" "))
stem_text <- stemDocument(nvec)
```

```
print(stem_text)

completed_text <- stemCompletion(stem_text,nvec)
completed_text
```

Congratulations! You have a cleaned and stemmed dataset. We are going to move on to structured corpora that might be a collection of tweets, user comments, or other short formats. Afterwards, we will move on to copora you would typically find in books, magazines, or other long-form documents.

Structured Text

We're going to work with a collection of peace treaties, we want to understand the major themes across the treaties.

If your dataset is in csv form, first load your dataset with the read.csv() function and then inspect its contents. By default, R will read the text column as a factor rather than character. You want to override this behavior, so set "stringsAsFactors" to False. It can be helpful to specify the encoding to be sure that emojis and other special characters are preserved. I know that my dataset is in utf-8.

```
peace_res <- read.csv("Desktop/INFO640-Labs/Datasets/pax_20_02_2018_1_CSV.csv",encoding = "u

str(peace_res)
glimpse(peace_res)
```

This is a rectangular (flat) dataset, we are going to reshape it into a corpus (dictionary) form.

First transform the text column into a corpus. This is a 2 step process, it can be nested, but there is always the intermediary step.

We need to preserve the document ID so we can join the work we do on the corpus and topic modeling to the other data in the dataframe. For this, we need to use the DataframeSource function (if we didn't care about the docID, we would use VSource, as shown in the comment).

To make this work, the text variable must be called "text" and the document ID variable must be called "doc_id".

First, change the names. Then specify the DataframeSource (the source for our corpus), and then transform it into a corpus with the VCorpus() function.

```
names(peace_res)[names(peace_res)=="AgtId"] <- "doc_id"
peace_res$doc_id <- as.character(peace_res$doc_id)
names(peace_res)[names(peace_res)=="OthAgr"] <- "text"
colnames(peace_res)
```

```
peace_source <- DataframeSource(peace_res)
peace_corpus <- VCorpus(peace_source)
```

Now our text column has been transformed into a special type of format: a corpus. Now we can use functions in the tm package to analyze it. The tm package has special functions and we would not be able to use with a rectangular dataset. Let's inspect this new data format.

If you just call print on the corpus itself, you will get some metadata about it. To call a specific record by its location, use indexing. The first number in the index is the record. Calling it will just return the metadata about that record. You can specify which entry of the record with the second value. The first index is with 2 sets of brackets because you are calling the whole record. The second index is with one set of brackets because it is a level beneath the record: the entry. Each record is a unit with multiple entries. Just like each record in a dictionary is a unit with multiple datatypes in it: the part-of-speech, the pronunciation, the definition, etc.

```
print(peace_corpus)

print(peace_corpus[[10]])
print(peace_corpus[[10]][1])
```

See what happens when you call the second entry of that record. Experiment with calling different records.

Now we will clean up our corpus in the same way we cleaned the text above. We will first make a new corpus to perform all of our transformations (that way if we make a mistake, we don't have to go back as far).

However, we cannot interact with the text directly - we must map the function onto the corpus. We will use the tm_map() function to map functions (yes, it's recursive). tm_map() takes 2 arguments: a corpus and a function. If the function requires and argument, it should be listed AFTER the function. Some functions that exist in base R (they are not part of the tm package) need to be wrapped in a content transformer function to apply to the corpus, see below.

First we will remove the numbers from our corpus.

```
peace_cleaned <- tm_map(peace_corpus, removeNumbers)
peace_cleaned[[10]][1]
```

Assignment (hints directly below)

Use the tm_map function to finish cleaning the peace corpus:

1. Remove punctuation
2. Strip excess whitespace

3. Make everything lowercase

4. Add “peace” and “agreement” to your stopwords list (you do not need tm_map() for this)

5. Remove stopwords

6. Inspect the output by looking at the text of the 5th entry.

We are not going to stem this document because it involves some advanced transformations and it is not essential to this project now.

Once you have a cleaned corpus, you can start to do some analysis on it.

```
#Ex
peace_cleaned <- tm_map(peace_corpus, removeNumbers)

#1 Make lowercase
# Hint: What do you want to do this operation on?
___ <- tm_map(___, content_transformer(___))

#2  Add "peace" and "agreement" to the stopwords list
___ = c(stopwords("en"), ___, ___)

#3 Remove your stopwords from your corpus
___ = tm_map(___, ___, ___)

#4 Remove Punctuation
___ <- tm_map(___, ___)

#5 Strip Whitespace
___ <- tm_map(___, ___)

#6 Inspect an entry

___[[___]][___]
```

Congratulations! You cleaned your first corpus. Now let’s do some analysis!

Topic Modeling

Topic modeling is a way to describe what topics are associated with a document and what words are associated with which topics. It is not perfect and requires significant human involvement to interpret the topics, but it does a pretty good job of

1. Clustering texts together
2. Getting a general sense of a document or documents

3. Identifying themes across documents

First we need to make a document-term matrix from our cleaned dataset. A document-term matrix is a table where every word is a column and every document is a row. If the word appears in the document once, put a 1 in the proper cell. If it appears twice, a 2, etc.

```
peace_dtm <- DocumentTermMatrix(peace_cleaned)
peace_dtm
```

Then make sure that you don't have any empty rows. Do this by finding the unique indexes and selecting only the unique indexes. If you had had an empty row in your csv, it will cause problems for your analysis. Finally, tidy the result so we can use the tidyverse features to analyze it.

```
unique_indexes <- unique(peace_dtm$i)

peace_dtm <- peace_dtm[unique_indexes,]
peace_dtm


peace_dtm_tidy <- tidy(peace_dtm)
peace_dtm_tidy

#possibly remove this section
cleaned_peace_res <- peace_dtm_tidy %>%
group_by(document) %>%
mutate(terms = toString(rep(term, count))) %>%
select(document, terms) %>%
unique()

head(cleaned_peace_res)
```

Then run an LDA analysis from the topic modeling package. We want 6 topics, you can choose more or less. We'll put 5 terms into each topic. Again, it can be a bit of trial and error here. The seed is an arbitrary number, but essential for shuffling the terms. It is common to pick 1234.

```
k <- 6
peace_lda <- LDA(peace_dtm, k = k, control = list(seed=1234))
peace_lda
peace_lda_words <- terms(peace_lda,5)
```

That gives us the top 5 terms for each of 6 topics. We would like to save this to a csv to review later. We'll use the paste function to paste together the name for the file we're writing.

```
peace_lda_topics <-as.matrix(peace_lda_words)
head(peace_lda_topics)
write.csv(peace_lda_topics,file=paste("Desktop/INFO640-Labs/peace_LDA_",k,".csv"))
```

It might be easier to visualize that than to read a table. We'll need to tidy, group and then make a bar chart.

```
peace_lda_tidy <- tidy(peace_lda)
peace_lda_tidy
```

That table gets us part of the way there, we now have a statistic for each word of how much it is associated with a topic. We can order the words from most prominent to least for each topic.

```
top_terms <- peace_lda_tidy %>%
  group_by(topic) %>%
  top_n(5, beta) %>%
  ungroup() %>%
  arrange(topic, -beta)

top_terms
```

Part way there. We'll reorder our terms based on the term and its prevalence and use ggplot to display it as a bar chart where the topic is the color and the size of the bar is the beta (or prevalence of the term). We'll facet it by topic so each topic is its own subplot.

```
top_terms %>%
  mutate(term = reorder(term, beta)) %>%
  ggplot(aes(term, beta, fill=factor(topic))) +
  geom_col(show.legend=FALSE) +
  facet_wrap(~ topic, scales = "free")+
  coord_flip()
```

The best way to improve your topic models is to refine your stopwords heavily. Removing stopwords will remove noise from the topics and make them more interpretable.

Exercise (optional)

Are there any more stopwords you would want to add? What words? Add these words to your list and rerun your code until you are satisfied with the output.

A more portable technique

That was a lot of steps. It would be much better if we could encapsulate all of that into a function and then just call that function to perform the LDA and visualize the result. This is far beyond what we have done in this course so

far, so if it doesn't make complete sense, don't worry. This is not an essential, though it will make your life much easier.

```
get_LDA_topics_terms_by_topic <- function(input_corpus, plot = TRUE, number_of_topics = 6, n
{
  my_dtm <- DocumentTermMatrix(input_corpus)

  unique_indexes <- unique(my_dtm$i)
  my_dtm <- my_dtm[unique_indexes,]

  my_lda <- LDA(my_dtm, k = number_of_topics, control = list(seed=1234))
  my_topics <- tidy(my_lda, matrix="beta")

  my_lda_words <- terms(my_lda, number_of_words)
  my_lda_topics <- as.matrix(my_lda_words)
  write.csv(my_lda_topics,file=paste(path,k,".csv"))

  my_top_terms <- my_topics %>%
    group_by(topic) %>%
    top_n(number_of_words, beta) %>%
    ungroup() %>%
    arrange(topic, -beta)

  if(plot==TRUE){
  my_top_terms %>%
    mutate(term = reorder(term, beta)) %>%
    ggplot(aes(term, beta, fill=factor(topic))) +
    geom_col(show.legend=FALSE) +
    facet_wrap(~ topic, scales = "free")+
    coord_flip()
  }else{
  return(my_top_terms)
  }
}
```

```
get_LDA_topics_terms_by_topic(peace_cleaned)
```

With a function, you can change any of the parameters you set above. We've set up our function to pass the corpus, number of topics, and the number of words

```
get_LDA_topics_terms_by_topic(peace_cleaned, number_of_topics = 4, number_of_words = 4)
```

Now we can use that function on ANY corpus we have created. *Keep this function handy - you will use it shortly*

Maybe instead of seeing the top topics across the entire corpus, we might want to see what topics are associated with which documents. The LDA Vector calls this calculation "gamma".

```
peace_lda_document_topics <- tidy(peace_lda, matrix="gamma")
peace_lda_document_topics
```

```
write.csv(peace_lda_document_topics,file=paste("Desktop/INFO640-Labs/peace_LDA_document_topi
```

```
head(peace_lda_document_topics)
```

Finally, it would be ideal if we could join these back to perform other analyses (i.e., which topic was most likely to result in a resolution, what topics trended over time, etc.). We can do sql style merges & joins with our dataframe. First let's inspect our dataframe and then "untidy" it. The spread() function is the opposite of tidy() and will turn your tidy dataframe into a regular, more usable dataframe.

```
head(peace_lda_document_topics)
dim(peace_lda_document_topics)

peace_lda_document <- spread(peace_lda_document_topics, topic, gamma)
dim(peace_lda_document)
head(peace_lda_document)
```

Now let's identify the max topic for each document. Assign a new column name. Select only the 2nd through 7th column, and apply a function that takes the dataframe and reads it by rows (axis=1), and asks "which one is the max"

```
peace_lda_document$max_topic <- colnames(peace_lda_document[2:7])[apply(peace_lda_document,1
```

```
head(peace_lda_document)
```

Now let's say we wanted to visualize the relationship between topics and other variables such as date, signing status, etc. To do this we need to merge the original peace_res dataframe to the topic weights using a specified key for each dataframe. First we will specify what we want the key to be and then merge the data frames. This uses the data.table library. Remember when we made our corpus, we made the "doc_id" column. It's time to put it to use!

```
dt1 <- data.table(peace_lda_document, key = "document")
dt2 <- data.table(peace_res, key = "doc_id")

peace_merged <- dt1[dt2]
dim(peace_merged)
colnames(peace_merged)
```

We probably don't want to deal with all of those variables, so let's select just the first ones. Refer back to the docs to understand these better.

```
peace_analyze <- select(peace_merged, c(Con,    Contp,  Reg, Dat,   Status, Lgt,    Agtp,

head(peace_analyze)
```

Congratulations! You can now compare different variables as they relate to the topics. In the interest of time, we are going to skip the actual visualization here. You can refer back to the Data Visualization Lab for more.

Unstructured Text

Let's work with something structurally simpler, but linguistically more complicated, like a book we can download from Project Gutenberg (for more on Project Gutenberg, visit www.gutenberg.org). Let's investigate Don Quixote https://www.gutenberg.org/ebooks/996. We already loaded the gutenbergr package above. This allows us to directly access texts from the project as long as we know the number. DonQuixote is 996.

```
dq <- gutenberg_download(996)
```

```
dq
```

This is a book, which has less structure than the csv, but is more complex than the text we worked with before.

Create the corpus similar to how we did before. We will not rely on any of the metadata that originally came with this book, so we can use a more direct method of source.

```
dq_source <- VectorSource(dq)
dq_corpus <- VCorpus(dq_source)
```

Exercise (optional)

Following the steps above,

1. Clean and prepare the text of Don Quixote for topic modeling.
2. Call your topic modeling function to view the output.
3. Find 4, 5, 6, and 7 topics (any number of words, but keep it consistent). Be sure to change the name of the file that is outputted.
4. What do you notice? Did you have to revisit any of the terms?

We will move on now to sentiment analysis.

Sentiment Analysis

A very common technique in text analysis is sentiment analysis. We'd like to understand the sentiment of different songs – is David Bowie overall positive or negative? We'll use the song lyrics dataset to investigate.

The tidytext library comes pre-packaged with a 3 sentiment dictionaries: one with intensity rankings, another with emotions, and a third with polarity.

```
sentiments

get_sentiments("afinn")
get_sentiments("nrc")
get_sentiments("bing")
```

These obviously have different purposes. We'll try out a couple different options. First, though, let's read in our dataset.

```
lyrics_raw <- read.csv("Desktop/INFO640-Labs/Datasets/songdata.csv",encoding = "utf-8",  hea
summary(lyrics_raw)
```

We'll tidy the dataframe rather than treat it as a corpus since we don't want to do any cleaning of song lyrics necessarily.

```
tidy_lyrics <- lyrics_raw %>%
  ungroup() %>%
  unnest_tokens(word, text)

summary(tidy_lyrics)
head(tidy_lyrics)
```

Let's find all the words that David bowie sang about that are deemed joyful. That is, what does "joy" look like for David Bowie? First filter the nrc sentiment dictionary for joy. Then filter the tyidy lyrics for the artist, David Bowie. OK, so we've narrowed down the words.

Perform an inner join on the "word" column for both the sentiment and the tidy lyrics. Then count the number of times each word appears & sort the list. We have to specify that we want the dplyr package for our count function since "count" is a function in many packages, and now that we've loaded so many, common functions like "count" must be specified. Sometimes you won't know this until after you get an error.

```
nrc_sent <- get_sentiments("nrc") %>%
filter(sentiment == "joy")

tidy_lyrics_bowie <- tidy_lyrics %>%
  filter(artist == "David Bowie")

tidy_lyrics_bowie %>%
```

```
  inner_join(nrc_sent) %>%
  dplyr::count(word, sort = TRUE)

bowie_sentiment <- tidy_lyrics_bowie %>%
  inner_join(get_sentiments("bing")) %>%
  dplyr::count(song, sentiment) %>%
  spread(sentiment, n, fill = 0) %>%
  mutate(sentiment = positive - negative)

head(bowie_sentiment)
```

Let's plot Bowie's songs as a function of their positive and negative sentiment to see if there are any clusters or interesting patterns.

```
ggplot(bowie_sentiment, aes(negative, positive, color = song)) +
  geom_jitter(show.legend = FALSE)
```

Finally, let's get an overall score of how positive or negative David Bowie was throughout his career.

```
bowie_career_sentiment <- mean(bowie_sentiment$sentiment)
bowie_career_sentiment
```

Exercise

Pick 2 more artists to compare. Who is overall more positive in their career? To find a list of all the artists to choose from, get a list of all the unique values with the unique function.

```
unique(lyrics_raw$artist)
```

Essential Answers

```
peace_cleaned <- tm_map(peace_corpus, removeNumbers)
peace_cleaned <- tm_map(peace_cleaned, content_transformer(tolower))
mystops = c(stopwords("en"), "peace", "agreement", "shall", "government", "page", "parties")
peace_cleaned = tm_map(peace_cleaned, removeWords, mystops)
peace_cleaned <- tm_map(peace_cleaned, removePunctuation)
peace_cleaned <- tm_map(peace_cleaned, stripWhitespace)
```