Chapter 19, Part 2: More About Text as Data

- Beyond parsing and searching through text documents, there are a number of graphical displays and summarizations of large sets of text data that have been developed in recent years.
- We will study some of these tools on several example documents and collections of documents.
- We begin with an extended example from Section 19.2 on a large collection of research papers.

Extended Example: Data Science papers in the ArXiv

- The *arXiv* (pronounced "archive") is a electronic repository where scientists from various disciplines can upload preprints of their research papers before they undergo peer review by a journal.
- The R package `aRxiv` package provides an API for querying the files and metadata available at the arXiv.
- The function `arxiv_search` allows you to search the arXiv using search terms, years, etc.
- The data set `DataSciencePapers` in the `mdsr` package contains the results of such a search as of August 2020:
- It contains 15 variables measured on 1089 research papers.

The DataSciencePapers Data Set

- The key column `id` contains a unique identifier for every paper in the data set.
- The column `abstract` contains the text of the abstract of each paper.
- The variables `submitted` and `updated` give dates/times of initial submission and last update of the paper.
- These two variables are entered as character strings, so we can use `lubridate` to convert these to date-time objects to facilitate analysis (see example of summarizing submission years for Data Science papers).
- The `primary_category` variable gives the topic area of each paper, but some cleaning of this variable will make its values more useful.
- Note that this `DataSciencePapers` data set is a *corpus*, a collection of many text documents.

The `tidytext` Package and its Tools

- The `tidytext` package has some valuable tools for getting text data ready to analyze closely.
- A *token* in text analysis is the smallest individual unit of text that we care to analyze.
- The `unnest_tokens` function uses a *tokenizer* to break text lines into tokens, which are typically individual words.
- By default, the function converts all characters to lowercase as it does this, so that, for example, "The" is treated the same as "the".
- This allows us to calculate word frequencies for papers, or we could choose other definitions of tokens like sentences or "N-grams" (more on those later).

Stopwords

- We probably don't care much about very common, inconsequential words like "a", "the", "from", 'you", etc.
- These are called *stopwords* and we often want to remove them before doing an analysis on the individual words in a document.
- The `get_stopwords` function from the `tidytext` package makes use of the `stopwords` package to find the commonly agreed-upon stopwords that are in the document.
- We can easily remove these from the text object using the `anti_join` function and create a "clean" version of the text object before further analysis.
- See examples on the `DataSciencePapers` data.

Word Clouds

- A simple visualization of the frequency of words in a document is called a *word cloud*.
- This displays the most common words in the document, with the most frequent being printed in larger font.
- The `wordcloud` package and function will produce a word cloud, with various options for colors, size, number of words shown, etc.
- See word cloud for the `DataSciencePapers` data.

Sentiment Analysis

- Word clouds show prevalence of words, but they don't really summarize the `meanings` of the words in the document.
- *Sentiment analysis* is a simple way to summarize how *positive* or *negative* a written text is.
- It relies on a *lexicon*, which is a previously obtained collection of words.
- Each word in the lexicon has been assigned a *sentiment score* which is a numeric measure of how positive or negative a word's sentiment is, based on the judgment of language expert(s).
- There are several choices of such lexicons that have been created.
- The AFINN lexicon from 2011 gives each word in the lexicon an integer value, from -5 (most negative) to 5 (most positive) — see example words from lexicon.

Using the Sentiment Scores in a Text Analysis

- We can *join* the lexicon and our text object and thus assign sentiment scores to the words in our document (at least those words that can be found in the lexicon).
- If words in our document are not in the lexicon, a *left join* will allow them to appear in the joined data table with missing (`NA`) sentiment scores that will count as zeroes when we sum the sentiment scores in the text object.
- It's often most useful to calculate the *sentiment per word* of a document rather than the summed sentiment score, so that documents of varying lengths can be fairly compared.
- See the sentiment analyses of the abstracts in the `DataSciencePapers` data.

Bigrams and N-grams

- Beyond analyzing individual words, we can analyze longer sequences of words that appear in a document.
- An N-gram is a contiguous sequence of N words.
- So a 1-gram is an individual word, and a 2-gram (more commonly called a *bigram*) is a pair of consecutive words.
- By choosing a bigram or other N-gram as our token of choice in `unnest_tokens`, we can explore which sequences of words are most common in the document.
- Arguably, when dealing with longer N-grams, it may make sense not to remove stopwords, since these could serve as important connectors in longer phrases.
- See the analysis of bigrams in the `DataSciencePapers` data.

Document Term Matrices and Prevalence Measures

- A *document term matrix* summarizes the prevalence of terms (e.g., words) in a document using any of several measures.
- The *term frequency* or *tf* of a term in some document is formally a function $tf(t, d)$ that counts the number of times a term t appears in a document d, divided by the total number of words in the document.
- The *inverse document frequency* or *idf* measures how often a term t appears in a collection of documents D.
- The exact formula for the *idf* is given in Section 19.2.5.
- If there are 50 documents in the collection and the term appears in 10 of them, then $idf(t, D) = \log(50/10) = 1.61$.
- If there are 50 documents in the collection and the term appears in 30 of them, then $idf(t, D) = \log(50/30) = 0.51$.
- So terms that appear in lots of documents have a **lower** *idf*.

The *tf_idf* Measure

- The *term frequency - inverse document frequency* or *tf_idf* combines the *tf* and *idf* into one measure by multiplying them together.
- Formally, this measure is $tf\_idf(t, d, D)$, a function of the term t, the document d, and the collection of documents D.
- A term that has a high *tf_idf* score within a document appears in that document much more often than it appears in other documents.
- Therefore, terms with large *tf_idf* scores in a document are useful keywords for a document.
- Also, terms with large *tf_idf* scores are useful search terms that will bring up that document quickly in a search.
- The `bind_tf_idf` function will calculate these measures (see examples with the *ArXiv* collection).

Structure of the Document-Term Matrix

- In a document-term matrix (DTM), rows correspond to documents in the collection and columns correspond to terms.
- For example, in the `DataSciencePapers` collection, there are 1089 documents (abstracts) and 12317 terms (words), so the document-term matrix will have 1089 rows and 12317 columns.
- In the (i, j) element of the matrix, there is some measure of prevalence of the jth term in the ith document.
- Often this might be a simple count of how many times the jth term appears in the ith document.
- We could also choose that the elements in the matrix be any of the previously discussed measures (*tf*, *idf*, or *tf_idf*).

Working with the Document-Term Matrix

- The R function `cast_dtm` will calculate a DTM for a collection of documents.
- It will give a measure of sparsity (how many of the entries of the DTM are 0).
- If the sparsity is high (near 100%), then many terms do not appear in most documents in the collection.

Summary Statistics from the DTM

- We typically would never print out the whole DTM, since it's so massive.
- Instead, we can examine summarizations, like looking at the words with the largest prevalence values.
- We can also look at *word correlations*:
- For example, we can find words that have a high correlation with the word *causal* — these are the words that tend to appear in the same documents as *causal* does.
- See examples on the *ArXiv* collection.

Two More Extended Examples

- On the course website, there is code that performs text analysis of an article I wrote that was published in a journal in 2024.
- For this example, we could analyze the text of the article as a whole, or we can break the article into its sections and treat the different sections as separate "documents".
- Section 19.3 presents an example of scraping a website using the `rvest` package to import an HTML-formatted table into R and store it as an R data frame.
- The table has textual information containing titles of Beatles songs and other variables characterizing the songs.
- After some preprocessing, we can use our text data analysis tools to make some conclusions about the Beatles song titles (see examples).