Tidy Text mining with R

The Tidy Text Format

In previous lessons we discussed the three features of Tidy Data, as outlined in Hadley Wickham's paper:[1]

- Each variable is a column
- Each observation is a row
- Each type of observational unit is a table

Tidy text format can be defined as **a table with one-token-per-row.** A **token** is any meaningful unit of text, such as a word, that we are interested in using for analysis. **Tokenization** is the process of splitting text into tokens. This is in contrast to storing text in strings or in a document-term matrix (discussed later). Here, the token stored in a single row is most often a single word. The **tidytext**[2] package provides functionality to tokenize strings by words (or n-grams, or sentences) and convert to a one-term-per-row format. By keeping text in "tidy" tables, you can use the normal tools you're familiar with, including dplyr, tidyr, ggplot2, etc., for manipulation, analysis, and visualization. The tidytext package also includes functions to convert to and from other data structures for text processing, such as a *corpus*[3] or a *document-term matrix.*[4]

Sentiment Analysis

Let's start to do some high-level analysis of the text we have. Sentiment analysis[5], also called opinion mining, is the use of text mining to "systematically identify, extract, quantify, and study affective states and subjective information." It's a way to try to understand the emotional intent of words to infer whether a section of text is positive or negative, or perhaps characterized by some other more nuanced emotion like surprise or disgust.

If you make a simplifying assumption regarding the text you have as a combination of its individual words, you can treat the sentiment content of the whole text as the sum of the sentiment content of the individual words. It's a simplification, and it isn't the only way to approach sentiment analysis, but it's simple and easy to do with tidy principles.

To get started you'll need a *sentiment lexicon* that attempt to evaluate the opinion or emotion in text. The tidytext package contains several sentiment lexicons in the `sentiments` dataset. All three of these lexicons are based on single words in the English language, assigning scores for positive/negative sentiment, or assigning emotions like joy, anger, sadness, etc.

- **`nrc`** from Saif Mohammad and Peter Turney[6] categorizes words in a binary fashion ("yes"/"no") into categories of positive, negative, anger, anticipation, disgust, fear, joy, sadness, surprise, and trust.
- **`bing`** from Bing Liu and collaborators[7] categorizes words in a binary fashion into positive and negative categories.
- **`AFINN`** from Finn Arup Nielsen[8] assigns words with a score that runs between -5 and 5, with negative scores indicating negative sentiment and positive scores indicating positive sentiment.

The built-in `sentiments` dataset available when you load the tidytext package contains all of this information. You could filter it to a single lexicon with the dplyr `filter()` function, or use tidytext's `get_sentiments()` to get specific sentiment lexicons containing only the data used for that lexicon.

There are a few major caveats to be aware of.

[1] http://vita.had.co.nz/papers/tidy-data.html
[2] https://cran.r-project.org/web/packages/tidytext/index.html
[3] Corpus objects contain strings annotated with additional metadata.
[4] This is a (sparse) matrix describing a collection (corpus) of documents with one row for each document and one column for each term. The value in the matrix is typically word count or tf-idf for the document in that row for the term in that column.
[5] https://en.wikipedia.org/wiki/Sentiment_analysis
[6] http://saifmohammad.com/WebPages/NRC-Emotion-Lexicon.htm
[7] https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html
[8] http://www2.imm.dtu.dk/pubdb/views/publication_details.php?id=6010

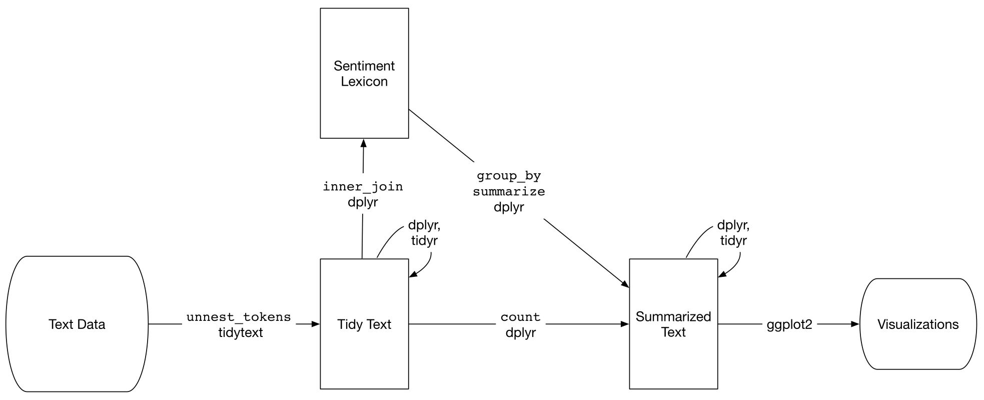

Figure 1: Workflow for text mining & sentiment analysis using tidy principles.

1. The sentiment lexicons we're using here were constructed either via crowdsourcing or by the work of the authors, and validated using crowdsourcing, movie/restaurant reviews, or Twitter data. It's unknown how useful it is to apply these lexicons to text from a completely different time and place (e.g., 200-year old fiction novels). Further, there are other domain-specific lexicons available, e.g., for finance data, that are better used in that context.
2. May words in the English language are fairly neutral, and aren't included in any sentiment lexicon.
3. Methods based on unigrams (single words) do not take into account qualifiers before a word, such as in "no good" or "not true". If you have sustained sections of sarcasm or negated text, this could be problematic.
4. The size of the chunk of text that we use to add up single-word sentiment scores matters. Sentiment across many paragraphs often has positive and negative sentiment averaging out to about zero, but sentence-sized or paragraph-sized text might be better.

Word and Document Frequencies

In text mining we're trying to get at "what is this text about?" We can start to get a sense of this by looking at the words that make up the text, and we can start to measure measure how important a word is by its **term frequency** (tf), how frequently a word occurs in a document. When we did this we saw some common words in the English language, so we took an approach to filter out our data first by a list of common stop words.

Another way is to look at a term's **inverse document frequency** (idf), which decreases the weight for commonly used words and increases the weight for words that are not used very much in a collection of documents. It's defined as:

$$idf(\text{term}) = \ln\left(\frac{n_{\text{documents}}}{n_{\text{documents containing term}}}\right)$$

If you multiply the two values together, you get the **tf-idf**[9], which is the frequency of a term adjusted for how rarely it is used. The tf-idf measures how important a word is to a document in a collection (or corpus) of documents, for example, to one novel in a collection of novels or to one website in a collection of websites.

We want to use tf-idf to find the important words for the content of each document by decreasing the weight for common words and increasing the weight for words that are not used very much in a corpus of documents (in this case, the group of Jane Austen's novels). Calculating tf-idf attempts to find the words that are important (i.e., common) in a text, but not *too* common.

You could do this all manually, but there's a nice function in the tidytext package called `bind_tf_idf` that does this for you. It takes a tidy text dataset as input with one row per word, per document. One column (`word` here) contains the terms/tokens, one column contains the documents (`book` in this case), and the last necessary column contains the counts, how many times each document contains each term (`n` in this example).

[9]https://en.wikipedia.org/wiki/Tf-idf

Project Gutenberg (https://www.gutenberg.org/) is a collection of freely available books in the public domain. The **gutenbergr**[10] package includes tools for finding and downloading books. We're going to download some classic science texts from Project Gutenberg and see what terms are important in these works, as measured by tf-idf. We'll use three classic physics texts, and a classic Darwin text. Let's use:

- *Discourse on Floating Bodies* by Galileo Galilei: http://www.gutenberg.org/ebooks/37729
- *Treatise on Light* by Christiaan Huygens: http://www.gutenberg.org/ebooks/14725
- *Experiments with Alternate Currents of High Potential and High Frequency* by Nikola Tesla: http://www.gutenberg.org/ebooks/13476
- *On the Origin of Species By Means of Natural Selection* by Charles Darwin: http://www.gutenberg.org/ebooks/5001

Topic Modeling

Topic modeling[11] is a method for unsupervised classification of such documents, similar to clustering on numeric data, which finds natural groups of items even when we're not sure what we're looking for. It's a way to find abstract "topics" that occur in a collection of documents, and it's frequently used to find hidden semantic structures in a text body. Topic models can help us understand large collections of unstructured text bodies. In addition to text mining tasks like what we'll do here, topic models have been used to detect useful structures in data such as genetic information, images, and networks, and have also been used in bioinformatics.[12]

Latent Dirichlet Allocation[13] is one of the most common algorithms used in topic modeling. LDA treats each document as a mixture of topics, and each topic as a mixture of words:

1. *Each document is a mixture of topics.* Each document contains words from several topics in particular proportions. For example, in a two-topic model we could say "Document 1 is 90% topic A and 10% topic B, while Document 2 is 30% topic A and 70% topic B."
2. *Every topic is a mixture of words.* Imagine a two-topic model of American news, with one topic for "politics" and one for "entertainment." Common words in the politics topic might be "President", "Congress", and "government", while the entertainment topic may be made up of words such as "movies", "television", and "actor". Words can be shared between topics; a word like "budget" might appear in both equally.

LDA attempts to estimate both of these at the same time: finding words associated with each topic, while simutaneously determining the mixture of topics that describes each document.

Document-term matrix: Before we can get started in topic modeling we need to take a look at another common format for storing text data that's not the tidy one-token-per-document-per-row format we've used so far (what we get from `unnest_tokens`). Another very common structure that's used by other text mining packages (such as **tm** or **quanteda**) is the **document-term matrix**[14] (DTM). This is a matrix where:

- Each row represents one document (such as a book or article).
- Each column represents one term.
- Each value contains the number of appearances of that term in that document.

Since most pairings of document and term do not occur (they have the value zero), DTMs are usually implemented as sparse matrices. These objects can be treated as though they were matrices (for example, accessing particular rows and columns), but are stored in a more efficient format. DTM objects can't be used directly with tidy tools, just as tidy data frames can't be used as input for other text mining packages. The tidytext package provides two verbs that convert between the two formats.

- `tidy()` turns a DTM into a tidy data frame. This verb comes from the broom package.
- `cast()` turns a tidy one-term-per-row data frame into a matrix. tidytext provides three variations of this verb, each converting to a different type of matrix:
  - `cast_sparse()`: converts to a sparse matrix from the Matrix package.
  - `cast_dtm()`: converts to a `DocumentTermMatrix` object from tm.
  - `cast_dfm()`: converts to a `dfm` object from quanteda.

[10] https://cran.r-project.org/web/packages/gutenbergr/vignettes/intro.html
[11] https://en.wikipedia.org/wiki/Topic_model
[12] Blei, David (April 2012). "Probabilistic Topic Models". *Communications of the ACM.* 55 (4): 77-84. doi:10.1145/2133806.2133826
[13] https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation
[14] https://en.wikipedia.org/wiki/Document-term_matrix