A Tidy Approach to Text Analysis in R

ART Forum 2019

Preamble

- The tidy approach is only *one* way to do text analysis in R.
- My goal is to get you doing text analysis as quickly as possible.
- I assume you have basic fluency in R and the tidyverse.
- Materials at github.com/marcdotson/tidy-text-analysis.
- Go to our shared RStudio Cloud project bit.ly/31qxlsJ.

Tokenizing and Visualization

Sentiment Analysis

Topic Modeling

Text as Unstructured Data

- Text is unstructured.
  - Authors can express themselves freely.
  - The same idea can be expressed in many ways.
- Text is increasingly available and important in marketing applications (e.g., social media, product reviews).

Working with Text

- The basic approach when analyzing text is called bag of words, where each word is considered separately (i.e., without syntax).
  - Each unique word is called a term.
  - Each realization of a term is a called a token.
  - A document is written by an author.
  - A collection of documents is called a corpus.
- By *counting* the tokens for each term, we produce word frequencies that we can visualize.
- We can use word frequencies to determine document sentiment and apply *unsupervised learning* techniques to find topics.

Tidy Text

- The tidyverse represents the state of the art for data wrangling and summarization in R and serves as the *foundation for a growing number of additional packages.*
- The tidytext package is an ideal example of this: it utilizes the tidyverse packages and adheres to the tidy data philosophy:
    1. Each observation (*token*) has its own row.
    2. Each variable has its own column.
    3. Each value has its own cell.

Text, Characters, and Strings

- Load the tidyverse and tidytext.

```
text <- c(
  "So long and thanks for all the fish,",
  "So sad that it should come to this,",
  "We tried to warn you all but oh dear!"
)
```

- What data type and structure is `text`?
- Turn `text` into a tibble (i.e., data frame) and number the lines.
- Is this tidy?

Tokenize

- Use `unnest_tokens()` to tokenize the text (i.e., split it into individual words or tokens).

  ```
  text_df %>%
    unnest_tokens(word, text)
  ```

- This is a *powerful* function.
  - Each token (i.e., word) has its own row.
  - The line number is preserved.
  - Tokens are converted to lowercase.
  - Whitespace and punctuation are stripped.
- Now that the data is tidy, all tidyverse tools are applicable!

Down the Rabbit Hole

- Install and load gutenbergr, a package that provides easy access to Project Gutenberg (the `gutenberg_id` is in a book's URL).
- Let's look at Lewis Carroll's *Alice's Adventures in Wonderland.*

```
tidy_carroll <- gutenberg_download(11) %>%
  unnest_tokens(word, text)
```

- Now that we have tidy data, how do we compute the word frequencies (in descending order)?
- Does this seem right to you?

Remove Stop Words

- Commonly used words aren't very informative and are referred to as stop words.

```
stop_words
```

- This is just a data frame, and we know how to join data frames!

```
tidy_carroll <- tidy_carroll %>%
  anti_join(stop_words)
```

- Why didn't we need to specify `by`?
- Produce the word frequencies (in descending order) again.

Visualize Word Frequencies

- Frequencies are counts, which says use bar plots.

```
tidy_carroll %>%
  count(word) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(x = word, y = n)) +
  geom_col()
```

- What can we do to make this plot readable?

Word Clouds

- With the size of words indicating frequency, a word cloud might also be a helpful visualization.
- Load the wordcloud package.

```
tidy_carroll %>%
  count(word) %>%
  with(wordcloud(word, n, min.freq = 10))
```

- Note that the location of a word in the cloud is random.

Exercise

Visit Project Gutenberg and select a book to analyze as follows.

1. In consecutive lines of code, download (the `gutenberg_id` is in a book's URL) the text, tokenize, and remove stop words.
2. Visualize the word frequencies using a column plot and a word cloud.

Tokenizing
and Visualization

Sentiment Analysis

Topic Modeling

Web Scraping

- Let's scrape data from the web; load the rvest package.

```
text <- read_html(
  "https://en.wikipedia.org/wiki/Provo,_Utah"
  ) %>%
  html_nodes("#content") %>%
  html_text() %>%
  str_split("\\\n\\\n\\\n") %>%
  unlist()
```

- The node and regular expression are also specific to this webpage.

Tokenize, Tidy, and Visualize

- Turn `text` into a data frame, create a *section* variable, tokenize, and remove stop words.

```
tidy_text <- tibble(text) %>%
  mutate(section = row_number()) %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words)
```

- Now visualize the word frequencies.
- Counts suggest meaning, but what is the emotional content?

Sentiment Dictionaries

- Sentiment is a reference to the emotional content of words.
- Like bag of words, the basic approach to sentiment analysis is to use a sentiment dictionary (i.e., lexicon).

  ```
  get_sentiments("afinn")
  ```

- Look at the `bing` and `nrc` sentiment dictionaries.
- What are the ten sentiments in the `nrc` sentiment dictionary?

Sentiment Analysis

- A sentiment dictionary is just a data frame, and we know how to join data frames!

  ```
  sentiment_nrc <- tidy_text %>%
    inner_join(get_sentiments("nrc"))
  ```

- What sentiments are represented most frequently in our data?
- What words contribute to the "joy" sentiment in our data?
- Note that a sentiment dictionary is time and application-specific.

Changing Sentiment

- How does sentiment change over a lengthy document?

```
tidy_carroll <- gutenberg_download(11) %>%
  mutate(line = row_number()) %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words)
```

- Let's pipe into the visualization.

```
tidy_carroll %>%
  inner_join(get_sentiments("bing")) %>%
  count(index = line %/% 30, sentiment) %>%
  spread(sentiment, n, fill = 0) %>%
  mutate(sentiment = positive - negative) %>%
  ggplot(aes(x = index, y = sentiment)) +
  geom_col()
```

Exercise

Scrape a Wikipedia page or download from Project Gutenberg.

1. In consecutive lines of code, create sections of the text, tokenize, and remove stop words.
2. Append a sentiment dictionary. What sentiments are most frequently represented?
3. Visualize how sentiment changes across the document.

Tokenizing
and Visualization

Sentiment Analysis

Topic Modeling

Word Frequencies

- Let's compare two of Lewis Carroll's books: *Alice's Adventures in Wonderland* and *Through the Looking-Glass*.

```
tidy_carroll <- gutenberg_download(c(11, 12)) %>%
  unnest_tokens(word, text) %>%
  mutate(
    book = factor(
      gutenberg_id,
      labels = c(
        "Alice's Adventures in Wonderland",
        "Through the Looking-Glass"
      )
    )
  ) %>%
  count(book, word) %>%
  arrange(desc(n))
```

Term Frequency-Inverse Document Frequency

- The tf-idf statistic weights the word frequencies for each document by how uncommon the word is within the corpus.

```
tidy_carroll %>%
  bind_tf_idf(word, book, n)
```

- Overwrite `tidy_carroll` with this new data frame and arrange in descending order by tf-idf.
- What words are most unique (i.e., important) for each book?

Visualize tf-idf by Document

- Let's create a single visualization for our corpus.

```
tidy_carroll %>%
  group_by(book) %>%
  top_n(10, tf_idf) %>%
  ungroup() %>%
  mutate(word = reorder(word, tf_idf)) %>%
  ggplot(aes(word, tf_idf, fill = book)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~ book, scales = "free") +
  coord_flip()
```

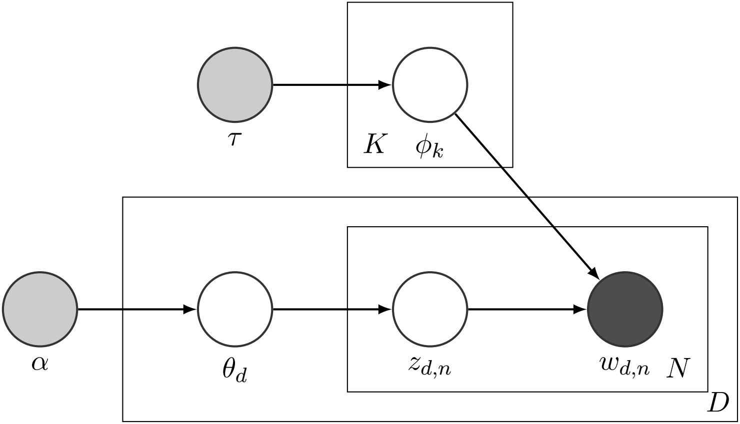
Beyond Word Frequencies

- Word frequencies (and tf-idf) suggest overall meaning and sentiment analysis describes emotional intent, but we would like to uncover the different topics being written about.
  - This is especially true as the size of a corpus increases.
- Topic models uncover groups of words (i.e., topics) via *unsupervised learning*.
- The most common topic model is called latent Dirichlet allocation or LDA.

Clustering vs. Topic Modeling

- Clustering
  - Clusters are uncovered based on *distance*, which is continuous.
  - Every object is assigned to a single cluster.
  - Clusters are summarized based on averages.
- Topic Modeling
  - Topics are uncovered based on *word frequency*, which is discrete.
  - Every document is a mixture (i.e., partial member) of every topic.
  - Topics are summarized based on word probabilities.

Latent Dirichlet Allocation



Create a Document Term Matrix

- Load the topicmodels package.
- Import `Roomba 650 Amazon Reviews.csv`, create a `review` id variable, tokenize, remove stop words, and select the `review` and `word` variables, and assign to `roomba_650`.
- The input for a topic model is not tidy data, it's a document term matrix. Let's "cast" our tidy data into a DTM.

```
dtm_text <- roomba_650 %>%
  count(review, word) %>%
  cast_dtm(review, word, n)
```

Run a Topic Model

- Running a topic model is straightforward with a DTM.

```
lda_out <- dtm_text %>%
  LDA(
    k = 2,
    method = "Gibbs",
    control = list(seed = 42)
  )
```

Topic Word Probabilities

- The most important output from a topic model are the topics themselves: the dictionary of words, sorted according to the probability the word is part of that topic.
- Let's "tidy" these probabilities (i.e., betas).

```
lda_topics <- lda_out %>%
  tidy(matrix = "beta")
```

Visualize, Name, and Choose K

- Now we can visualize our topics.

```
lda_topics %>%
  group_by(topic) %>%
  top_n(15, beta) %>%
  ungroup() %>%
  mutate(term = reorder(term, beta)) %>%
  ggplot(aes(term, beta, fill = as.factor(topic))) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~ topic, scales = "free") +
  coord_flip()
```

- How would you name these two topics?
- How can you be sure `k = 2` is enough topics?

Exercise

Using the `roomba_650` data, do the following.

1. Run a number of topic models with varying k.
2. Visualize and compare topics across solutions. Which solution is best?

Epilogue

- We've covered the basics of a tidy approach to text analysis, building our analysis on the tidyverse to investigate word frequencies, visualizations, sentiment analysis, and topic modeling.
- *Text Mining with R* tidytextmining.com
- DataCamp datacamp.com

Advanced Topics

- Structural Topic Models structuraltopicmodel.com
- word2vec/fastText github.com/facebookresearch/fastText

Bonus: HTML Nodes

- When scraping a webpage, you need to identify the HTML node you want to extract using a selector.
- An easy one to use is SelectorGadget.
- Use a selector to identify the node we want to extract from the Wikipedia entry for The Hitchhiker's Guide to the Galaxy.
- Load `tidyverse`, `tidytext`, and `rvest`.

```
read_html("<URL>") %>%
  html_nodes("<NODE>") %>%
  html_text()
```

Bonus: Regular Expressions

- *Regular expressions* (or regexps) are used to describe patterns in strings and are thus helpful in splitting a string into sections, lines, chapters, and documents. Some basics:
  - `"abc"` matches abc
  - `"."` matches any character
  - `"\\s"` matches whitespace
  - `"\\d"` matches any digit
  - `"(a|b)"` matches a or b
  - `"\\"` escapes special behavior (e.g., `"\\."` matches .)
- Practice writing regular expressions with `str_view_all()`.

```
test <- "Text in 452 is a textbook end to Winter 2018!"
str_view_all(test, "end")
```

- Write a regexp to match both uses of "text" in `test`.