CHAPTER 5

Converting to and from Nontidy Formats

In the previous chapters, we've been analyzing text arranged in the tidy text format: a table with one token per document per row, such as is constructed by the function `unnest_tokens()`. This lets us use the popular suite of tidy tools such as dplyr, tidyr, and ggplot2 to explore and visualize text data. We've demonstrated that many informative text analyses can be performed using these tools.

However, most of the existing R tools for natural language processing, besides the tidytext package, aren't compatible with this format. The CRAN Task View for Natural Language Processing lists a large selection of packages that take other structures of input and provide nontidy outputs. These packages are very useful in text mining applications, and many existing text datasets are structured according to these formats.

Computer scientist Hal Abelson has observed that, "No matter how complex and polished the individual operations are, it is often the quality of the glue that most directly determines the power of the system" (Abelson 2008). In that spirit, this chapter will discuss the "glue" that connects the tidy text format with other important packages and data structures, allowing you to rely on both existing text mining packages and the suite of tidy tools to perform your analysis.

Figure 5-1 illustrates how an analysis might switch between tidy and nontidy data structures and tools. This chapter will focus on the process of tidying document-term matrices, as well as casting a tidy data frame into a sparse matrix. We'll also explore how to tidy Corpus objects, which combine raw text with document metadata, into text data frames, leading to a case study of ingesting and analyzing financial articles.

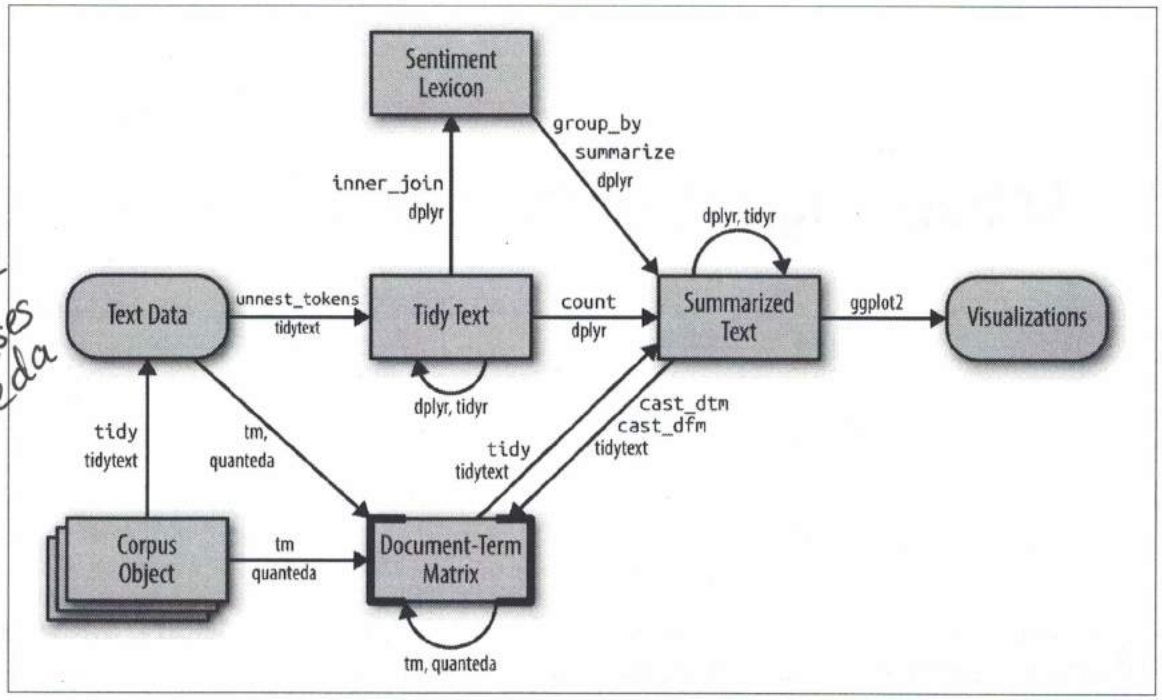


Figure 5-1. A flowchart of a typical text analysis that combines tidytext with other tools and data formats, particularly the tm or quanteda packages. This chapter shows how to convert back and forth between document-term matrices and tidy data frames, as well as convert from a Corpus object to a text data frame.

Tidying a Document-Term Matrix

One of the most common structures that text mining packages work with is the document-term matrix (or DTM). This is a matrix where:

- Each row represents one document (such as a book or article).
- Each column represents one term.
- Each value (typically) 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. We'll discuss several implementations of these matrices in this chapter.

DTM objects cannot be used directly with tidy tools, just as tidy data frames cannot be used as input for most text mining packages. Thus, the tidytext package provides two verbs that convert between the two formats:

- `tidy()` turns a document-term matrix into a tidy data frame. This verb comes from the broom package (Robinson 2017), which provides similar tidying functions for many statistical models and objects.
- `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()` (converting to a sparse matrix from the Matrix package), `cast_dtm()` (converting to a `DocumentTermMatrix` object from tm), and `cast_dfm()` (converting to a `dfm` object from quanteda).

As shown in Figure 5-1, a DTM is typically comparable to a tidy data frame after a `count` or a `group_by/summarize` that contains counts or another statistic for each combination of a term and document.

Tidying DocumentTermMatrix Objects

Perhaps the most widely used implementation of DTMs in R is the `DocumentTermMatrix` class in the tm package. Many available text mining datasets are provided in this format. For example, consider the collection of Associated Press newspaper articles included in the topicmodels package.

```
library(tm)

data("AssociatedPress", package = "topicmodels")
AssociatedPress
```

```
## <<DocumentTermMatrix (documents: 2246, terms: 10473)>>
## Non-/sparse entries: 302031/23220327
## Sparsity           : 99%
## Maximal term length: 18
## Weighting          : term frequency (tf)
```

We see that this dataset contains documents (each of them an AP article) and terms (distinct words). Notice that this DTM is 99% sparse (99% of document-word pairs are zero). We could access the terms in the document with the `Terms()` function.

```
terms <- Terms(AssociatedPress)
head(terms)
```

```
## [1] "aaron"      "abandon"    "abandoned"  "abandoning" "abbott"     "abboud"
```

If we wanted to analyze this data with tidy tools, we would first need to turn it into a data frame with one token per document per row. The broom package introduced the `tidy()` verb, which takes a nontidy object and turns it into a tidy data frame. The tidytext package implements this method for `DocumentTermMatrix` objects.

```
library(dplyr)
library(tidytext)

ap_td <- tidy(AssociatedPress)
ap_td
```

```
## # A tibble: 302,031 × 3
##    document       term count
##       <int>      <chr> <dbl>
## 1         1     adding     1
## 2         1      adult     2
## 3         1        ago     1
## 4         1    alcohol     1
## 5         1  allegedly     1
## 6         1      allen     1
## 7         1 apparently     2
## 8         1   appeared     1
## 9         1   arrested     1
## 10        1    assault     1
## # ... with 302,021 more rows
```

compare this with the non-tidy format. It's so much easier to see.

"document" column tells us this is from text/book #1. The next text/book would have "document" #"2."

Notice that we now have a tidy three-column `tbl_df`, with variables `document`, `term`, and `count`. This tidying operation is similar to the `melt()` function from the reshape2 package (Wickham 2007) for nonsparse matrices.

Notice that only the nonzero values are included in the tidied output: document 1 includes terms such as "adding" and "adult," but not "aaron" or "abandon." This means the tidied version has no rows where `count` is zero.

As we've seen in previous chapters, this form is convenient for analysis with the dplyr, tidytext, and ggplot2 packages. For example, you can perform sentiment analysis on these newspaper articles with the approach described in Chapter 2.

```
ap_sentiments <- ap_td %>%
  inner_join(get_sentiments("bing"), by = c(term = "word"))

ap_sentiments
```

```
## # A tibble: 30,094 × 4
##    document    term count sentiment
##       <int>   <chr> <dbl>     <chr>
## 1         1 assault     1  negative
## 2         1 complex     1  negative
## 3         1   death     1  negative
## 4         1    died     1  negative
## 5         1    good     2  positive
## 6         1 illness     1  negative
## 7         1  killed     2  negative
## 8         1    like     2  positive
## 9         1   liked     1  positive
```

```
## 10        1 miracle     1  positive
## # ... with 30,084 more rows
```

This would let us visualize which words from the AP articles most often contributed to positive or negative sentiment, seen in Figure 5-2. We can see that the most common positive words include "like," "work," "support," and "good," while the most negative words include "killed," "death," and "vice." (The inclusion of "vice" as a negative term is probably a mistake on the algorithm's part, since it likely usually refers to "vice president").

```
library(ggplot2)

ap_sentiments %>%
  count(sentiment, term, wt = count) %>%
  ungroup() %>%
  filter(n >= 200) %>%
  mutate(n = ifelse(sentiment == "negative", -n, n)) %>%
  mutate(term = reorder(term, n)) %>%
  ggplot(aes(term, n, fill = sentiment)) +
  geom_bar(stat = "identity") +
  ylab("Contribution to sentiment") +
  coord_flip()
```

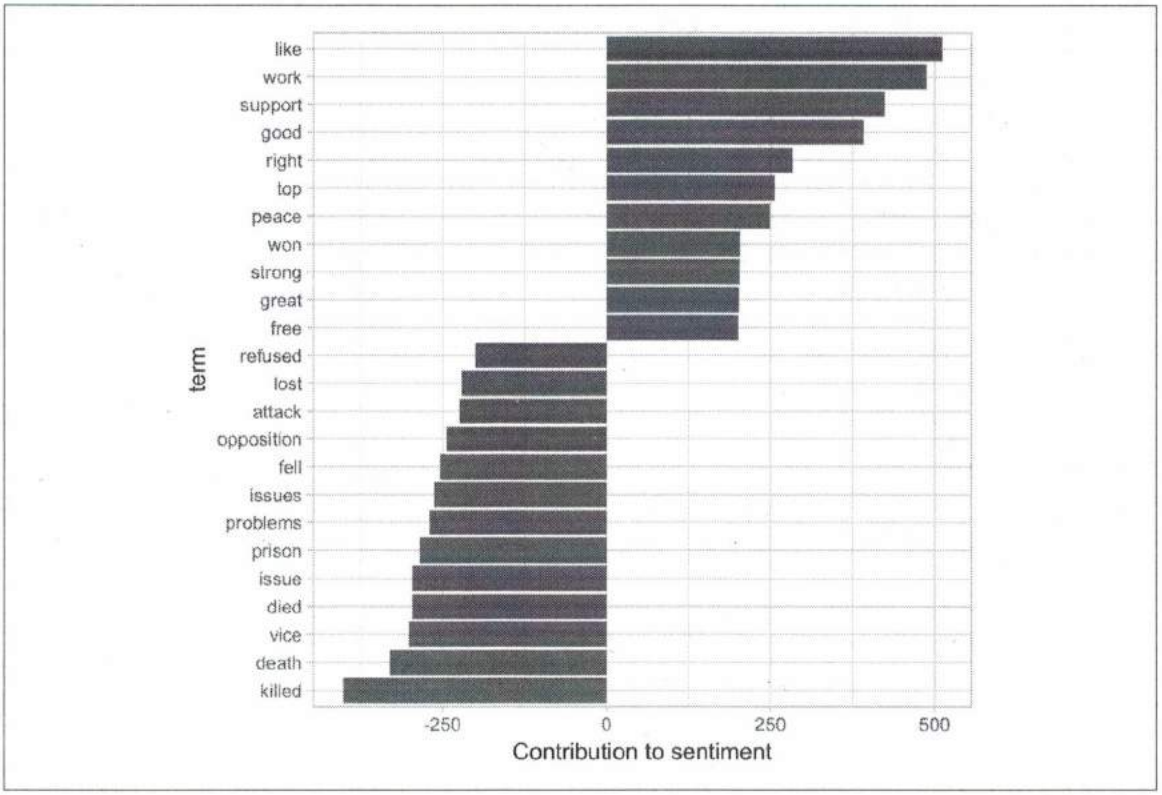


Figure 5-2. Words from AP articles with the greatest contribution to positive or negative sentiments, computed as the product of the word's AFINN sentiment score and its frequency.

Tidying dfm Objects

Other text mining packages provide alternative implementations of document-term matrices, such as the `dfm` (document-feature matrix) class from the quanteda package (Benoit and Nulty 2016). For example, the quanteda package comes with a corpus of presidential inauguration speeches, which can be converted to a `dfm` using the appropriate function.

```
library(methods)

data("data_corpus_inaugural", package = "quanteda")
inaug_dfm <- quanteda::dfm(data_corpus_inaugural, verbose = FALSE)

inaug_dfm
```

```
## Document-feature matrix of: 58 documents, 9,232 features (91.6% sparse).
```

The `tidy` method works on these document-feature matrices as well, turning them into a one-token-per-document-per-row table.

```
inaug_td <- tidy(inaug_dfm)
inaug_td
```

```
## # A tibble: 44,725 × 3
##            document   term count
##               <chr>  <chr> <dbl>
## 1  1789-Washington fellow     3
## 2  1793-Washington fellow     1
## 3       1797-Adams fellow     3
## 4   1801-Jefferson fellow     7
## 5   1805-Jefferson fellow     8
## 6     1809-Madison fellow     1
## 7     1813-Madison fellow     1
## 8      1817-Monroe fellow     6
## 9      1821-Monroe fellow    10
## 10      1825-Adams fellow     3
## # ... with 44,715 more rows
```

We may be interested in finding the words most specific to each of the inaugural speeches. This could be quantified by calculating the tf-idf of each term-speech pair using the `bind_tf_idf()` function, as described in Chapter 3.

```
inaug_tf_idf <- inaug_td %>%
  bind_tf_idf(term, document, count) %>%
  arrange(desc(tf_idf))

inaug_tf_idf
```

```
## # A tibble: 44,725 × 6
##            document        term count          tf      idf     tf_idf
##               <chr>       <chr> <dbl>       <dbl>    <dbl>      <dbl>
## 1  1793-Washington      arrive     1 0.006802721 4.060443 0.02762206
## 2  1793-Washington upbraidings     1 0.006802721 4.060443 0.02762206
```

```
## 3  1793-Washington    violated     1 0.006802721 3.367296 0.02290677
## 4  1793-Washington   willingly     1 0.006802721 3.367296 0.02290677
## 5  1793-Washington   incurring     1 0.006802721 3.367296 0.02290677
## 6  1793-Washington    previous     1 0.006802721 2.961831 0.02014851
## 7  1793-Washington   knowingly     1 0.006802721 2.961831 0.02014851
## 8  1793-Washington injunctions     1 0.006802721 2.961831 0.02014851
## 9  1793-Washington   witnesses     1 0.006802721 2.961831 0.02014851
## 10 1793-Washington     besides     1 0.006802721 2.674149 0.01819149
## # ... with 44,715 more rows
```

We could use this data to pick four notable inaugural addresses (from Presidents Lincoln, Roosevelt, Kennedy, and Obama), and visualize the words most specific to each speech, as shown in Figure 5-3.

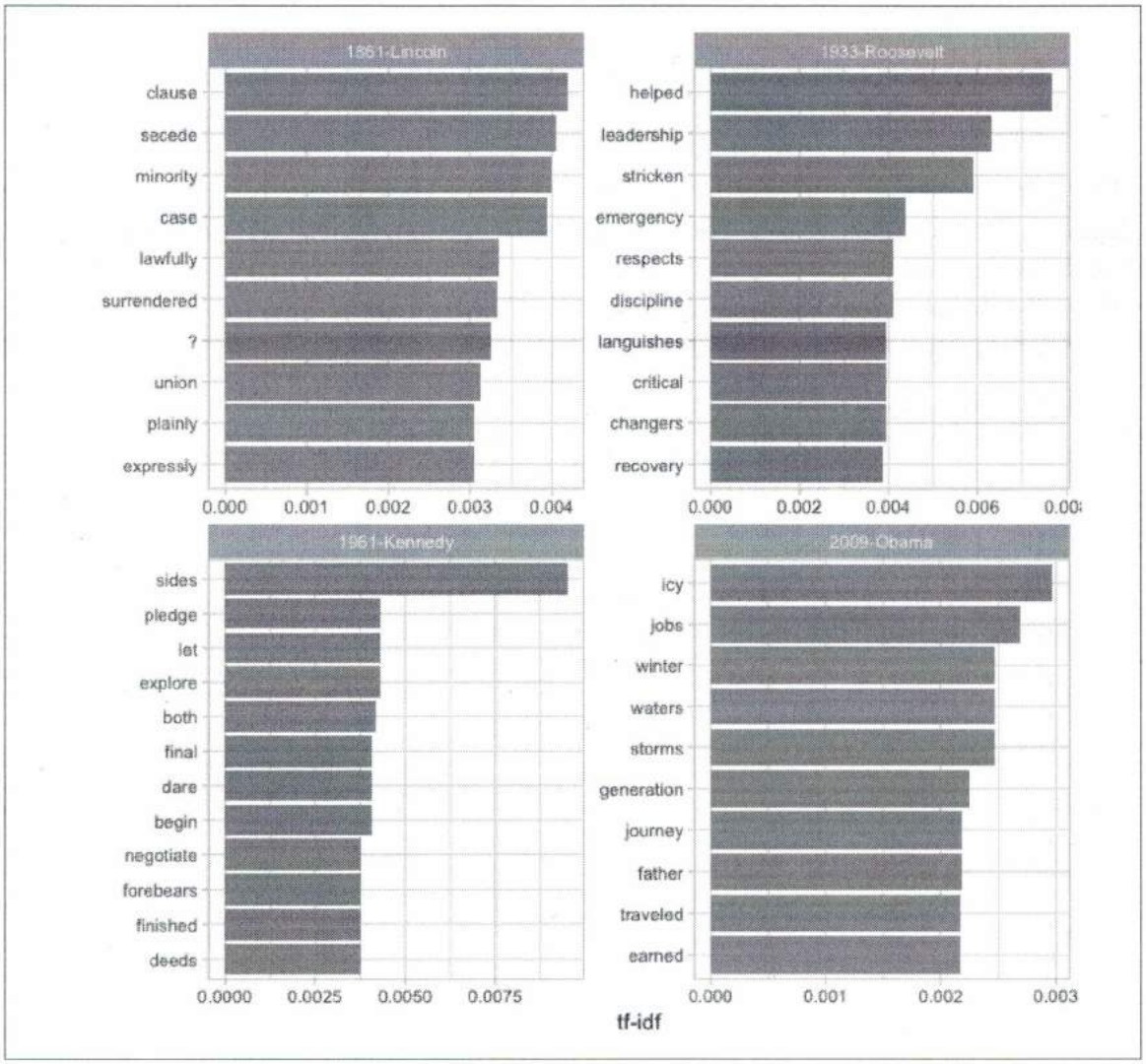


Figure 5-3. The terms with the highest tf-idf from each of four selected inaugural addresses. Note that quanteda's tokenizer includes the "?" punctuation mark as a term, though the texts we've tokenized ourselves with unnest_tokens do not.

As another example of a visualization possible with tidy data, we could extract the year from each document's name, and compute the total number of words within each year.

Note that we've used tidyr's `complete()` function to include zeroes (cases where a word doesn't appear in a document) in the table.

```
library(tidyr)

year_term_counts <- inaug_td %>%
  extract(document, "year", "(\\d+)", convert = TRUE) %>%
  complete(year, term, fill = list(count = 0)) %>%
  group_by(year) %>%
  mutate(year_total = sum(count))
```

This lets us pick several words and visualize how they changed in frequency over time, as shown in Figure 5-4. We can see that over time, American presidents became less likely to refer to the country as the "Union" and more likely to refer to "America." They also became less likely to talk about the "Constitution" and "foreign" countries, and more likely to mention "freedom" and "God."

```
year_term_counts %>%
  filter(term %in% c("god", "america", "foreign",
                     "union", "constitution", "freedom")) %>%
  ggplot(aes(year, count / year_total)) +
  geom_point() +
  geom_smooth() +
  facet_wrap(~ term, scales = "free_y") +
  scale_y_continuous(labels = scales::percent_format()) +
  ylab("% frequency of word in inaugural address")
```

With the Luthier stories, once we've identified which parallel passages are the same/diff using p. 31, we can also use this below method to evaluate the changes over time

ie we see both WHAT it is and HOW it is

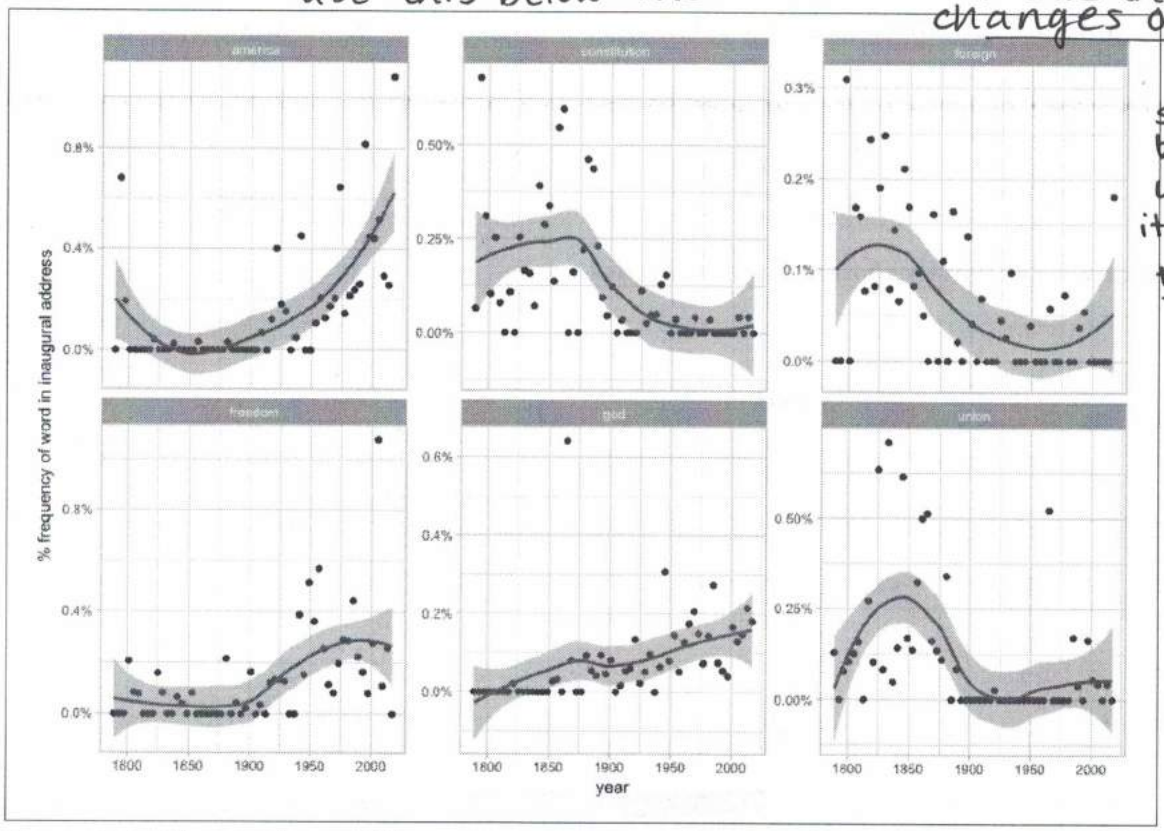


Figure 5-4. Changes in word frequency over time within Presidential inaugural addresses, for four selected terms (four selected presidents)

These examples show how you can use tidytext, and the related suite of tidy tools, to analyze sources even if their origin is not in a tidy format.

Casting Tidy Text Data into a Matrix

Just as some existing text mining packages provide document-term matrices as sample data or output, some algorithms expect such matrices as input. Therefore, tidytext provides `cast_` verbs for converting from a tidy form to these matrices.

For example, we could take the tidied AP dataset and cast it back into a document-term matrix using the `cast_dtm()` function.

```
ap_td %>%
  cast_dtm(document, term, count)
```

```
## <<DocumentTermMatrix (documents: 2246, terms: 10473)>>
## Non-/sparse entries: 302031/23220327
## Sparsity           : 99%
## Maximal term length: 18
## Weighting          : term frequency (tf)
```

```
dfm(term, document, count)
ment-feature matrix of: 10,473 documents, 2,246 features (98.7% sparse).
```

simply require a sparse matrix.

```
(Matrix)

into a Matrix object
_td %>%
sparse(document, term, count)

)

"dgCMatrix"
(,"package")
"Matrix"

n)
```

```
[1]  2246 10473
```

nd of conversion could easily be done from any of the tidy text structures sed so far in this book. For example, we could create a DTM of Jane Austen's n just a few lines of code.

```
rary(janeaustenr)

ten_dtm <- austen_books() %>%
nnest_tokens(word, text) %>%
ount(book, word) %>%
ast_dtm(book, word, n)

ten_dtm
```

```
<<DocumentTermMatrix (documents: 6, terms: 14520)>>
Non-/sparse entries: 40379/46741
Sparsity           : 54%
Maximal term length: 19
Weighting          : term frequency (tf)
```

sting process allows for reading, filtering, and processing to be done using nd other tidy tools, after which the data can be converted into a document-atrix for machine learning applications. In Chapter 6, we'll examine some es where a tidy text dataset has to be converted into a `DocumentTermMatrix` for ing.

Similarly, we could cast the table into a `dfm` object from quanteda's dfm with `cast_dfm()`.

```
ap_td %>%
  cast_dfm(term, document, count)
```

```
## Document-feature matrix of: 10,473 documents, 2,246 features (98.7% sparse).
```

Some tools simply require a sparse matrix.

```
library(Matrix)

# cast into a Matrix object
m <- ap_td %>%
  cast_sparse(document, term, count)

class(m)
```

```
## [1] "dgCMatrix"
## attr(,"package")
## [1] "Matrix"
```

```
dim(m)
```

```
## [1]  2246 10473
```

This kind of conversion could easily be done from any of the tidy text structures we've used so far in this book. For example, we could create a DTM of Jane Austen's books in just a few lines of code.

```
library(janeaustenr)

austen_dtm <- austen_books() %>%
  unnest_tokens(word, text) %>%
  count(book, word) %>%
  cast_dtm(book, word, n)

austen_dtm
```

```
## <<DocumentTermMatrix (documents: 6, terms: 14520)>>
## Non-/sparse entries: 40379/46741
## Sparsity           : 54%
## Maximal term length: 19
## Weighting          : term frequency (tf)
```

This casting process allows for reading, filtering, and processing to be done using dplyr and other tidy tools, after which the data can be converted into a document-term matrix for machine learning applications. In Chapter 6, we'll examine some examples where a tidy text dataset has to be converted into a `DocumentTermMatrix` for processing.

Tidying Corpus Objects with Metadata

Some data structures are designed to store document collections *before* tokenization, often called a "corpus." One common example is Corpus objects from the tm package. These store text alongside *metadata*, which may include an ID, date/time, title, or language for each document.

For example, the tm package comes with the `acq` corpus, containing 50 articles from the news service Reuters.

```
data("acq")
acq
```

```
## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 50
```

```
# first document
acq[[1]]
```

```
## <<PlainTextDocument>>
## Metadata:  15
## Content:  chars: 1287
```

A Corpus object is structured like a list, with each item containing both text and metadata (see the tm documentation for more on working with Corpus objects). This is a flexible storage method for documents, but doesn't lend itself to processing with tidy tools.

We can thus use the `tidy()` method to construct a table with one row per document, including the metadata (such as `id` and `datetimestamp`) as columns alongside the `text`.

```
acq_td <- tidy(acq)
acq_td
```

```
## # A tibble: 50 × 16
##                       author       datetimestamp description
##                        <chr>              <dttm>       <chr>
## 1                       <NA> 1987-02-26 10:18:06
## 2                       <NA> 1987-02-26 10:19:15
## 3                       <NA> 1987-02-26 10:49:56
## 4  By Cal Mankowski, Reuters 1987-02-26 10:51:17
## 5                       <NA> 1987-02-26 11:08:33
## 6                       <NA> 1987-02-26 11:32:37
## 7       By Patti Domm, Reuter 1987-02-26 11:43:13
## 8                       <NA> 1987-02-26 11:59:25
## 9                       <NA> 1987-02-26 12:01:28
## 10                      <NA> 1987-02-26 12:08:27
##                                            heading    id language
##                                              <chr> <chr>    <chr>
## 1   COMPUTER TERMINAL SYSTEMS <CPML> COMPLETES SALE    10       en
```

```
## 2    OHIO MATTRESS <OMT> MAY HAVE LOWER 1ST QTR NET    12       en
## 3     MCLEAN'S <MII> U.S. LINES SETS ASSET TRANSFER    44       en
## 4    CHEMLAWN <CHEM> RISES ON HOPES FOR HIGHER BIDS    45       en
## 5    <COFAB INC> BUYS GULFEX FOR UNDISCLOSED AMOUNT    68       en
## 6          INVESTMENT FIRMS CUT CYCLOPS <CYL> STAKE    96       en
## 7  AMERICAN EXPRESS <AXP> SEEN IN POSSIBLE SPINNOFF   110       en
## 8   HONG KONG FIRM UPS WRATHER<WCO> STAKE TO 11 PCT   125       en
## 9              LIEBERT CORP <LIEB> APPROVES MERGER    128       en
## 10     GULF APPLIED TECHNOLOGIES <GATS> SELLS UNITS   134       en
## # ... with 40 more rows, and 10 more variables: language <chr>, origin <chr>,
## #   topics <chr>, lewissplit <chr>, cgisplit <chr>, oldid <chr>,
## #   places <list>, people <lgl>, orgs <lgl>, exchanges <lgl>, text <chr>
```

This can then be used with `unnest_tokens()` to, for example, find the most common words across the 50 Reuters articles, or the ones most specific to each article.

```
acq_tokens <- acq_td %>%
  select(-places) %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words, by = "word")

# most common words
acq_tokens %>%
  count(word, sort = TRUE)
```

```
## # A tibble: 1,566 × 2
##        word     n
##       <chr> <int>
## 1      dlrs   100
## 2       pct    70
## 3       mln    65
## 4   company    63
## 5    shares    52
## 6    reuter    50
## 7     stock    46
## 8     offer    34
## 9     share    34
## 10 american    28
## # ... with 1,556 more rows
```

✓ exactly

```
# tf-idf
acq_tokens %>%
  count(id, word) %>%
  bind_tf_idf(word, id, n) %>%
  arrange(desc(tf_idf))
```

```
## Source: local data frame [2,853 x 6]
## Groups: id [50]
##
##       id     word     n         tf      idf    tf_idf
##    <chr>    <chr> <int>      <dbl>    <dbl>     <dbl>
## 1    186   groupe     2 0.13333333 3.912023 0.5216031
## 2    128  liebert     3 0.13043478 3.912023 0.5102639
## 3    474  esselte     5 0.10869565 3.912023 0.4252199
```

✓

```
## 4    371  burdett     6 0.10344828 3.912023 0.4046920
## 5    442 hazleton     4 0.10256410 3.912023 0.4012331
## 6    199  circuit     5 0.10204082 3.912023 0.3991860
## 7    162 suffield     2 0.10000000 3.912023 0.3912023
## 8    498     west     3 0.10000000 3.912023 0.3912023
## 9    441      rmj     8 0.12121212 3.218876 0.3901668
## 10   467  nursery     3 0.09677419 3.912023 0.3785829
## # ... with 2,843 more rows
```

Example: Mining Financial Articles

Corpus objects are a common output format for data-ingesting packages, which means the `tidy()` function gives us access to a wide variety of text data. One example is tm.plugin.webmining, which connects to online feeds to retrieve news articles based on a keyword. For instance, performing `WebCorpus(GoogleFinanceSource("NASDAQ:MSFT"))` allows us to retrieve the 20 most recent articles related to the Microsoft (MSFT) stock.

Here we'll retrieve recent articles relevant to nine major technology stocks: Microsoft, Apple, Google, Amazon, Facebook, Twitter, IBM, Yahoo, and Netflix.

These results were downloaded in January 2017, when this chapter was written, so you'll certainly find different results if you run it for yourself. Note that this code takes several minutes to run.

```
library(tm.plugin.webmining)
library(purrr)

company <- c("Microsoft", "Apple", "Google", "Amazon", "Facebook",
             "Twitter", "IBM", "Yahoo", "Netflix")
symbol <- c("MSFT", "AAPL", "GOOG", "AMZN", "FB", "TWTR", "IBM", "YHOO", "NFLX")

download_articles <- function(symbol) {
  WebCorpus(GoogleFinanceSource(paste0("NASDAQ:", symbol)))
}

stock_articles <- data_frame(company = company,
                             symbol = symbol) %>%
  mutate(corpus = map(symbol, download_articles))
```

This uses the `map()` function from the purrr package, which applies a function to each item in `symbol` to create a list, which we store in the `corpus` list column.

```
stock_articles
```

```
## # A tibble: 9 × 3
##     company symbol          corpus
##       <chr>  <chr>          <list>
```

```
## 1 Microsoft   MSFT <S3: WebCorpus>
## 2     Apple   AAPL <S3: WebCorpus>
## 3    Google   GOOG <S3: WebCorpus>
## 4    Amazon   AMZN <S3: WebCorpus>
## 5  Facebook     FB <S3: WebCorpus>
## 6   Twitter   TWTR <S3: WebCorpus>
## 7       IBM    IBM <S3: WebCorpus>
## 8     Yahoo   YHOO <S3: WebCorpus>
## 9   Netflix   NFLX <S3: WebCorpus>
```

Each of the items in the `corpus` list column is a `WebCorpus` object, which is a special case of a corpus like `acq`. We can thus turn each into a data frame using the `tidy()` function, unnest it with tidyr's `unnest()`, then tokenize the `text` column of the individual articles using `unnest_tokens()`.

```
stock_tokens <- stock_articles %>%
  unnest(map(corpus, tidy)) %>%
  unnest_tokens(word, text) %>%
  select(company, datetimestamp, word, id, heading)

stock_tokens
```

```
## # A tibble: 105,057 × 5
##      company       datetimestamp        word
##        <chr>              <dttm>       <chr>
## 1  Microsoft 2017-01-17 07:07:24   microsoft
## 2  Microsoft 2017-01-17 07:07:24 corporation
## 3  Microsoft 2017-01-17 07:07:24        data
## 4  Microsoft 2017-01-17 07:07:24     privacy
## 5  Microsoft 2017-01-17 07:07:24       could
## 6  Microsoft 2017-01-17 07:07:24        send
## 7  Microsoft 2017-01-17 07:07:24        msft
## 8  Microsoft 2017-01-17 07:07:24       stock
## 9  Microsoft 2017-01-17 07:07:24     soaring
## 10 Microsoft 2017-01-17 07:07:24          by
## # ... with 105,047 more rows, and 2 more variables: id <chr>, heading <chr>
```

Here we see some of each article's metadata alongside the words used. We could use tf-idf to determine which words were most specific to each stock symbol.

```
library(stringr)

stock_tf_idf <- stock_tokens %>%
  count(company, word) %>%
  filter(!str_detect(word, "\\d+")) %>%
  bind_tf_idf(word, company, n) %>%
  arrange(-tf_idf)
```

The top terms for each are visualized in Figure 5-5. As we'd expect, the company's name and symbol are typically included, but so are several of their product offerings and executives, as well as companies they are making deals with (such as Disney with Netflix).

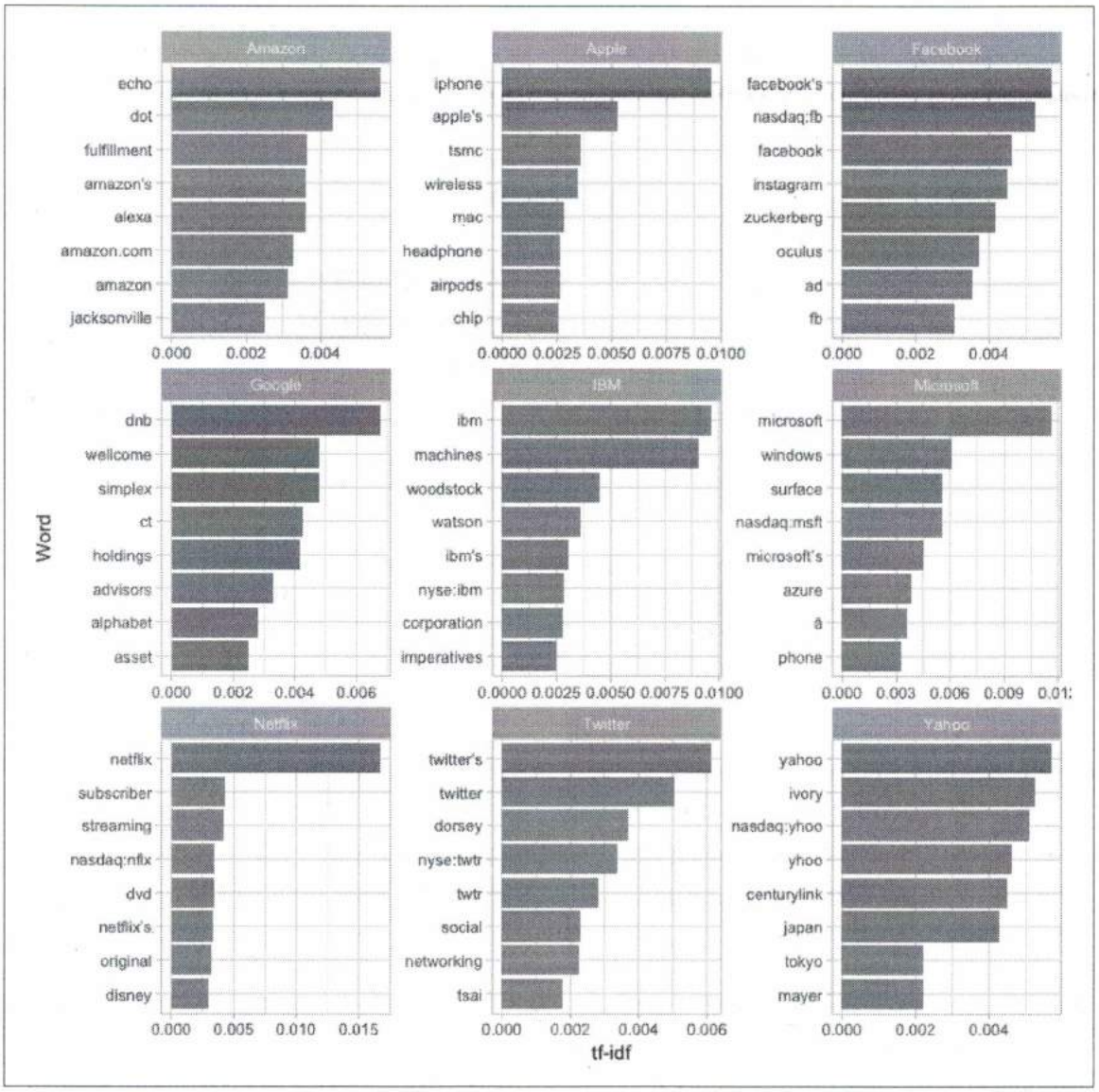


Figure 5-5. The eight words with the highest tf-idf in recent articles specific to each company

If we were interested in using recent news to analyze the market and make investment decisions, we'd likely want to use sentiment analysis to determine whether the news coverage was positive or negative. Before we run such an analysis, we should look at what words would contribute the most to positive and negative sentiments, as was shown in "Most Common Positive and Negative Words" on page 22. For example, we could examine this within the AFINN lexicon (Figure 5-6).

```
stock_tokens %>%
  anti_join(stop_words, by = "word") %>%
  count(word, id, sort = TRUE) %>%
  inner_join(get_sentiments("afinn"), by = "word") %>%
  group_by(word) %>%
```

```
summarize(contribution = sum(n * score)) %>%
top_n(12, abs(contribution)) %>%
mutate(word = reorder(word, contribution)) %>%
ggplot(aes(word, contribution)) +
geom_col() +
coord_flip() +
labs(y = "Frequency of word * AFINN score")
```

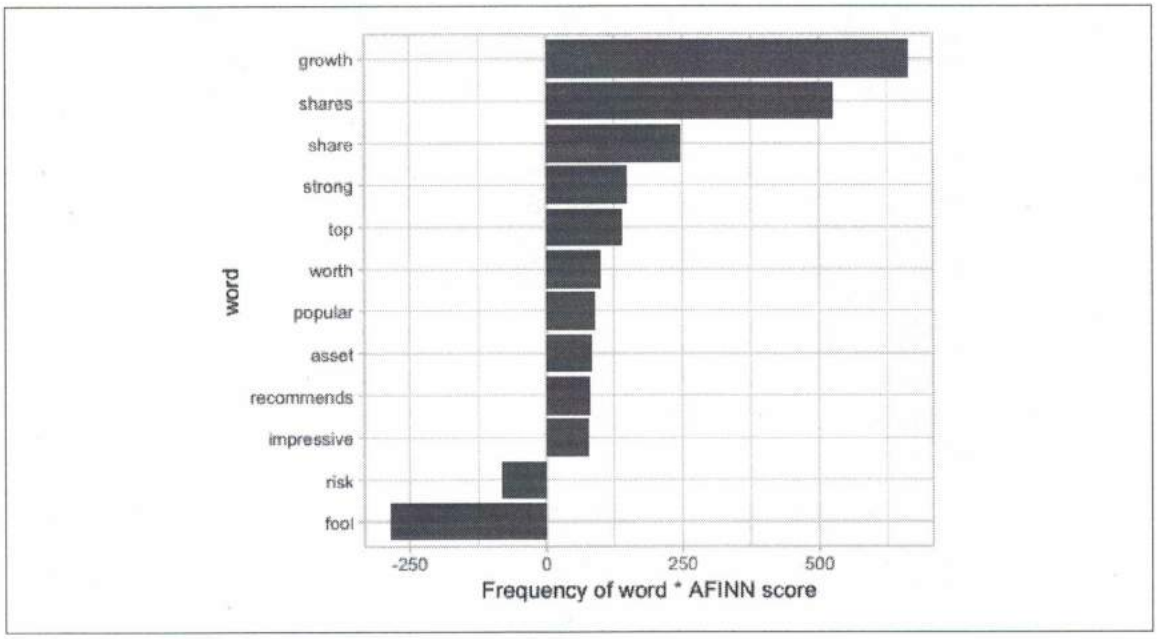


Figure 5-6. The words with the largest contribution to sentiment scores in recent financial articles, according to the AFINN dictionary. The "contribution" is the product of the word and the sentiment score.

In the context of these financial articles, there are a few big red flags here. The words "share" and "shares" are counted as positive verbs by the AFINN lexicon ("Alice will **share** her cake with Bob"), but they're actually neutral nouns ("The stock price is $12 per **share**") that could just as easily be in a positive sentence as a negative one. The word "fool" is even more deceptive: it refers to Motley Fool, a financial services company. In short, we can see that the AFINN sentiment lexicon is entirely unsuited to the context of financial data (as are the NRC and Bing lexicons).

Instead, we introduce another sentiment lexicon: the Loughran and McDonald dictionary of financial sentiment terms (Loughran and McDonald 2011). This dictionary was developed based on analyses of financial reports, and intentionally avoids words like "share" and "fool," as well as subtler terms like "liability" and "risk" that may not have a negative meaning in a financial context.

The Loughran data divides words into six sentiments: "positive," "negative," "litigious," "uncertain," "constraining," and "superfluous." We could start by examining

the most common words belonging to each sentiment within this text dataset (Figure 5-7).

```
stock_tokens %>%
  count(word) %>%
  inner_join(get_sentiments("loughran"), by = "word") %>%
  group_by(sentiment) %>%
  top_n(5, n) %>%
  ungroup() %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n)) +
  geom_col() +
  coord_flip() +
  facet_wrap(~ sentiment, scales = "free") +
  ylab("Frequency of this word in the recent financial articles")
```

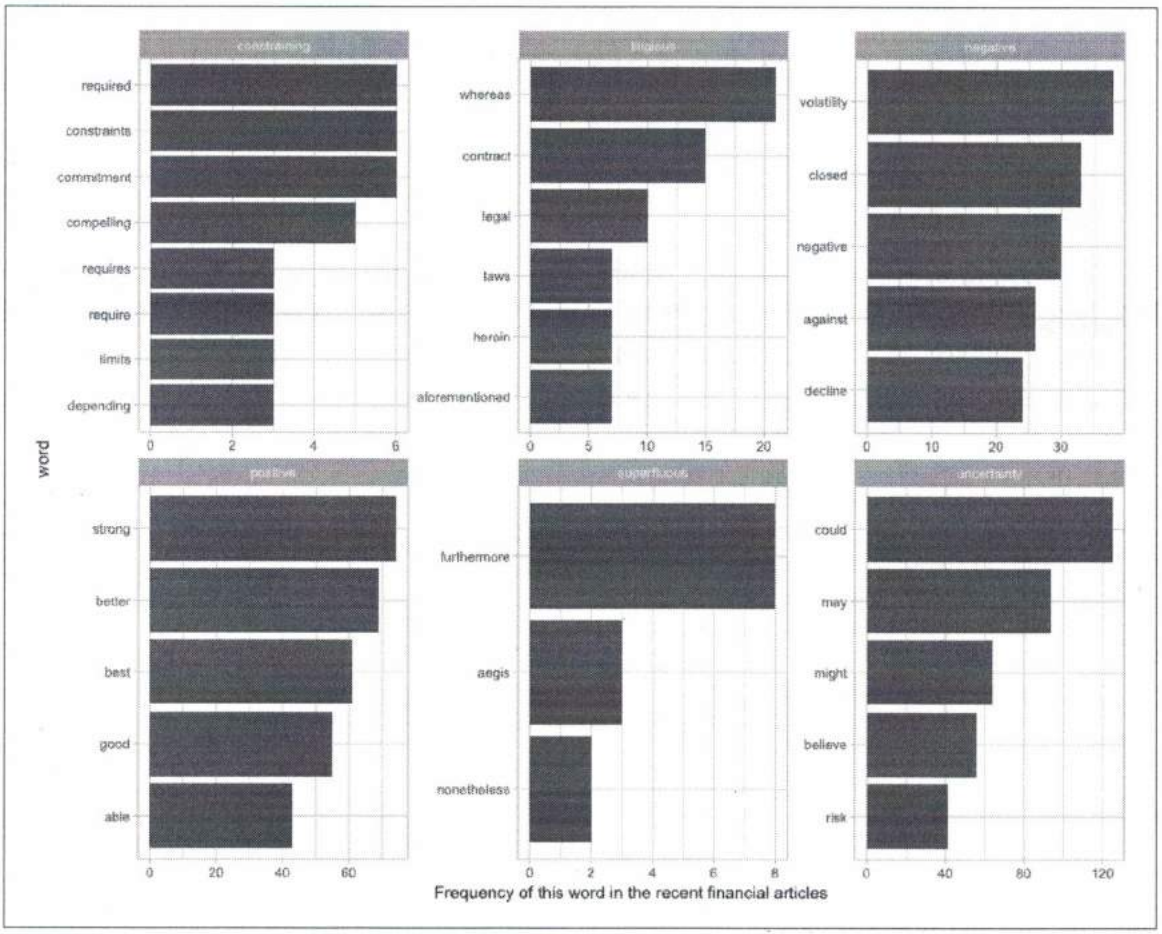


Figure 5-7. The most common words in the financial news articles associated with each of the six sentiments in the Loughran and McDonald lexicon

These assignments (Figure 5-7) of words to sentiments look more reasonable: common positive words include "strong" and "better," but not "shares" or "growth," while negative words include "volatility" but not "fool." The other sentiments look reasonable as well: the most common "uncertainty" terms include "could" and "may."

Now that we know we can trust the dictionary to approximate the articles' sentiments, we can use our typical methods for counting the number of uses of each sentiment-associated word in each corpus.

```
stock_sentiment_count <- stock_tokens %>%
  inner_join(get_sentiments("loughran"), by = "word") %>%
  count(sentiment, company) %>%
  spread(sentiment, n, fill = 0)

stock_sentiment_count
```

```
## # A tibble: 9 × 7
##      company constraining litigious negative positive superfluous uncertainty
## *      <chr>        <dbl>     <dbl>    <dbl>    <dbl>       <dbl>       <dbl>
## 1    Amazon            7         8       84      144           3          70
## 2     Apple            9        11      161      156           2         132
## 3  Facebook            4        32      128      150           4          81
## 4    Google            7         8       60      103           0          58
## 5       IBM            8        22      147      148           0         104
## 6 Microsoft            6        19       92      129           3         116
## 7   Netflix            4         7      111      162           0         106
## 8   Twitter            4        12      157       79           1          75
## 9     Yahoo            3        28      130       74           0          71
```

It might be interesting to examine which company has the most news with "litigious" or "uncertain" terms. But the simplest measure, much as it was for most analyses in Chapter 2, is to see whether the news is more positive or negative. As a general quantitative measure of sentiment, we'll use `(positive - negative) / (positive + negative)` (Figure 5-8).

```
stock_sentiment_count %>%
  mutate(score = (positive - negative) / (positive + negative)) %>%
  mutate(company = reorder(company, score)) %>%
  ggplot(aes(company, score, fill = score > 0)) +
  geom_col(show.legend = FALSE) +
  coord_flip() +
  labs(x = "Company",
       y = "Positivity score among 20 recent news articles")
```

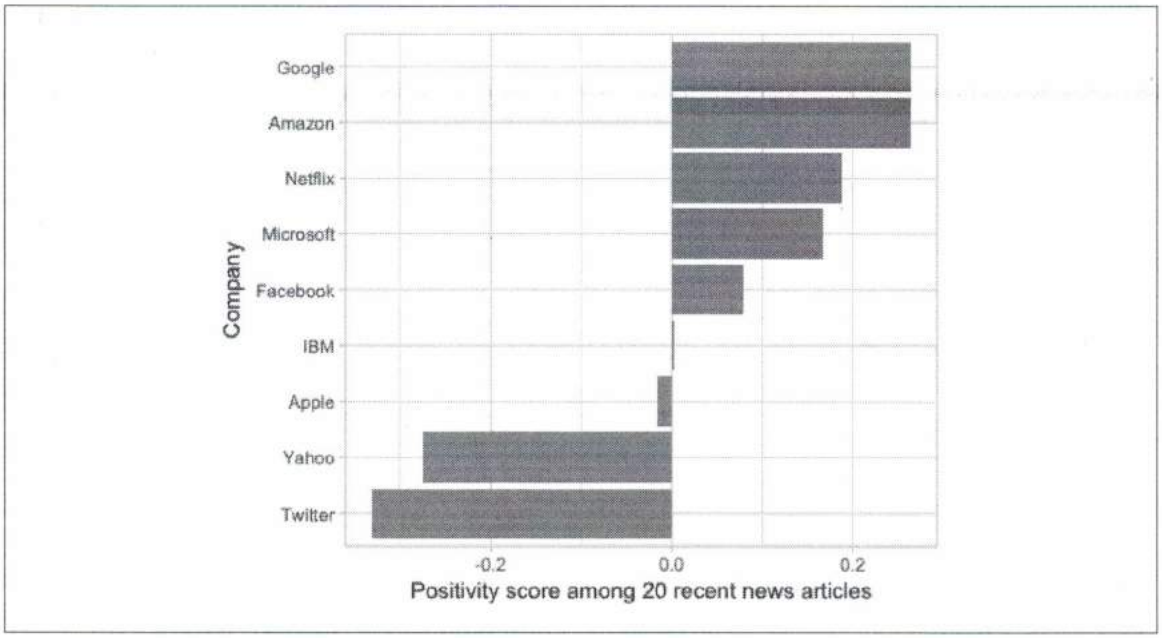


Figure 5-8. "Positivity" of the news coverage around each stock in January 2017, calculated as (positive - negative) / (positive + negative), based on uses of positive and negative words in 20 recent news articles about each company

Based on this analysis, we'd say that in January 2017, most of the coverage of Yahoo and Twitter was strongly negative, while coverage of Google and Amazon was the most positive. A glance at current financial headlines suggests that it's on the right track. If you were interested in further analysis, you could use one of R's many quantitative finance packages to compare these articles to recent stock prices and other metrics. eek, no thanks

Summary

Text analysis requires working with a variety of tools, many of which have inputs and outputs that aren't in a tidy form. This chapter showed how to convert between a tidy text data frame and sparse document-term matrices, as well as how to tidy a Corpus object containing document metadata. The next chapter will demonstrate another notable example of a package, topicmodels, that requires a document-term matrix as input, showing that these conversion tools are an essential part of text analysis.

CHAPTER 6
Topic Modeling

In text mining, we often have collections of documents, such as blog posts or news articles, that we'd like to divide into natural groups so that we can understand them separately. Topic modeling 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. or it can confirm what we thought we found with close reading

Latent Dirichlet allocation (LDA) is a particularly popular method for fitting a topic model. It treats each document as a mixture of topics, and each topic as a mixture of words. This allows documents to "overlap" each other in terms of content, rather than being separated into discrete groups, in a way that mirrors typical use of natural language.

what are the weaknesses of LDA model?

As Figure 6-1 shows, we can use tidy text principles to approach topic modeling with the same set of tidy tools we've used throughout this book. In this chapter, we'll learn to work with `LDA` objects from the topicmodels package, particularly tidying such models so that they can be manipulated with ggplot2 and dplyr. We'll also explore an example of clustering chapters from several books, where we can see that a topic model "learns" to tell the difference between the four books based on the text content.

eg if we have loads of Ents, we're probably looking at ROTK not FOTR.

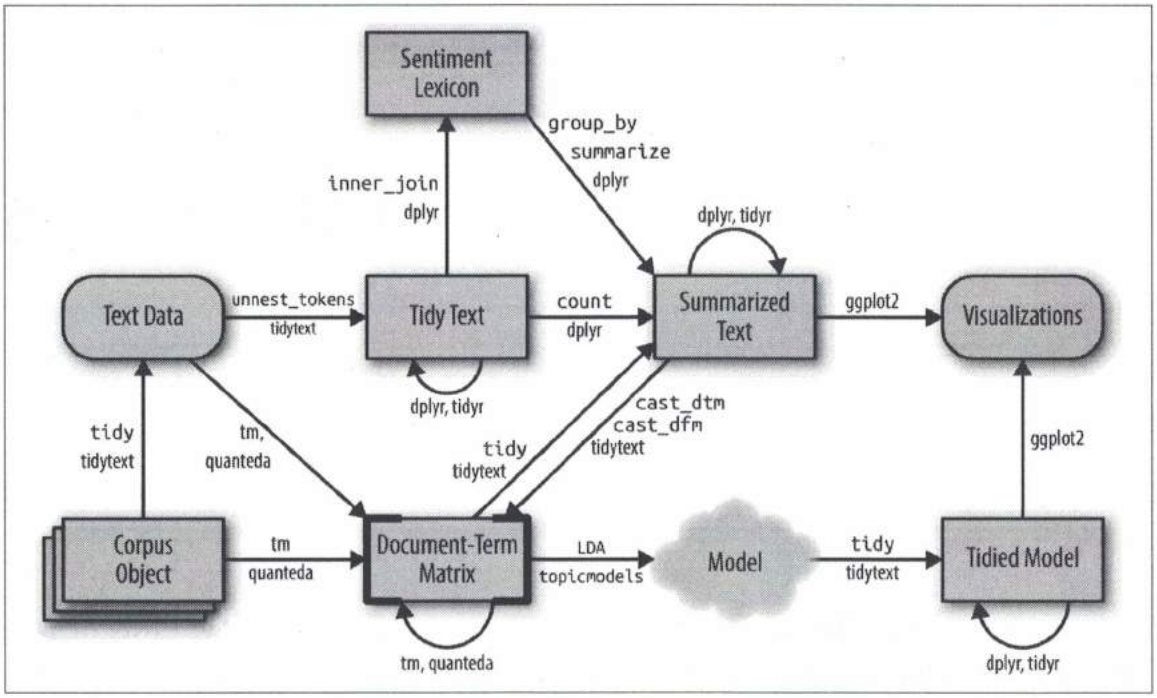


Figure 6-1. A flowchart of a text analysis that incorporates topic modeling. The topicmodels package takes a document-term matrix as input and produces a model that can be tidied by tidytext, such that it can be manipulated and visualized with dplyr and ggplot2.

Latent Dirichlet Allocation

Latent Dirichlet allocation is one of the most common algorithms for topic modeling. Without diving into the math behind the model, we can understand it as being guided by two principles: document < topics < words

Every document is a mixture of topics

We imagine that each document may contain 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."

Every topic is a mixture of words

For example, we could imagine a two-topic model of American news, with one topic for "politics" and one for "entertainment." The most 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." Importantly, words can be shared between topics; a word like "budget" might appear in both equally.

LDA is a mathematical method for estimating both of these at the same time: finding the mixture of words that is associated with each topic, while also determining the mixture of topics that describes each document. There are a number of existing implementations of this algorithm, and we'll explore one of them in depth.

In Chapter 5 we briefly introduced the `AssociatedPress` dataset, provided by the topicmodels package, as an example of a `DocumentTermMatrix`. This is a collection of 2,246 news articles from an American news agency, mostly published around 1988.

```
library(topicmodels)

data("AssociatedPress")
AssociatedPress
```

```
## <<DocumentTermMatrix (documents: 2246, terms: 10473)>>
## Non-/sparse entries: 302031/23220327
## Sparsity           : 99%
## Maximal term length: 18
## Weighting          : term frequency (tf)
```

We can use the `LDA()` function from the topicmodels package, setting `k = 2`, to create a two-topic LDA model.

Almost any topic model in practice will use a larger k, but we will soon see that this analysis approach extends to a larger number of topics.

This function returns an object containing the full details of the model fit, such as how words are associated with topics and how topics are associated with documents.

```
# set a seed so that the output of the model is predictable
ap_lda <- LDA(AssociatedPress, k = 2, control = list(seed = 1234))
ap_lda
```

```
## A LDA_VEM topic model with 2 topics.
```

Fitting the model was the "easy part": the rest of the analysis will involve exploring and interpreting the model using tidying functions from the tidytext package.

Word-Topic Probabilities

In Chapter 5 we introduced the `tidy()` method, originally from the broom package (Robinson 2017), for tidying model objects. The tidytext package provides this method for extracting the per-topic-per-word probabilities, called β ("beta"), from the model.

```
library(tidytext)

ap_topics <- tidy(ap_lda, matrix = "beta")
ap_topics
```

```
## # A tibble: 20,946 × 3
##    topic       term         beta
##    <int>      <chr>        <dbl>
## 1      1      aaron 1.686917e-12
## 2      2      aaron 3.895941e-05
## 3      1    abandon 2.654910e-05
## 4      2    abandon 3.990786e-05
## 5      1  abandoned 1.390663e-04
## 6      2  abandoned 5.876946e-05
## 7      1 abandoning 2.454843e-33
## 8      2 abandoning 2.337565e-05
## 9      1     abbott 2.130484e-06
## 10     2     abbott 2.968045e-05
## # ... with 20,936 more rows
```

Notice that this has turned the model into a one-topic-per-term-per-row format. For each combination, the model computes the probability of that term being generated from that topic. For example, the term "aaron" has a 1.686917×10^{-12} probability of being generated from topic 1, but a 3.8959408×10^{-5} probability of being generated from topic 2.

We could use dplyr's top_n() to find the 10 terms that are most common within each topic. As a tidy data frame, this lends itself well to a ggplot2 visualization (Figure 6-2).

```
library(ggplot2)
library(dplyr)

ap_top_terms <- ap_topics %>%
  group_by(topic) %>%
  top_n(10, beta) %>%
  ungroup() %>%
  arrange(topic, -beta)

ap_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()
```

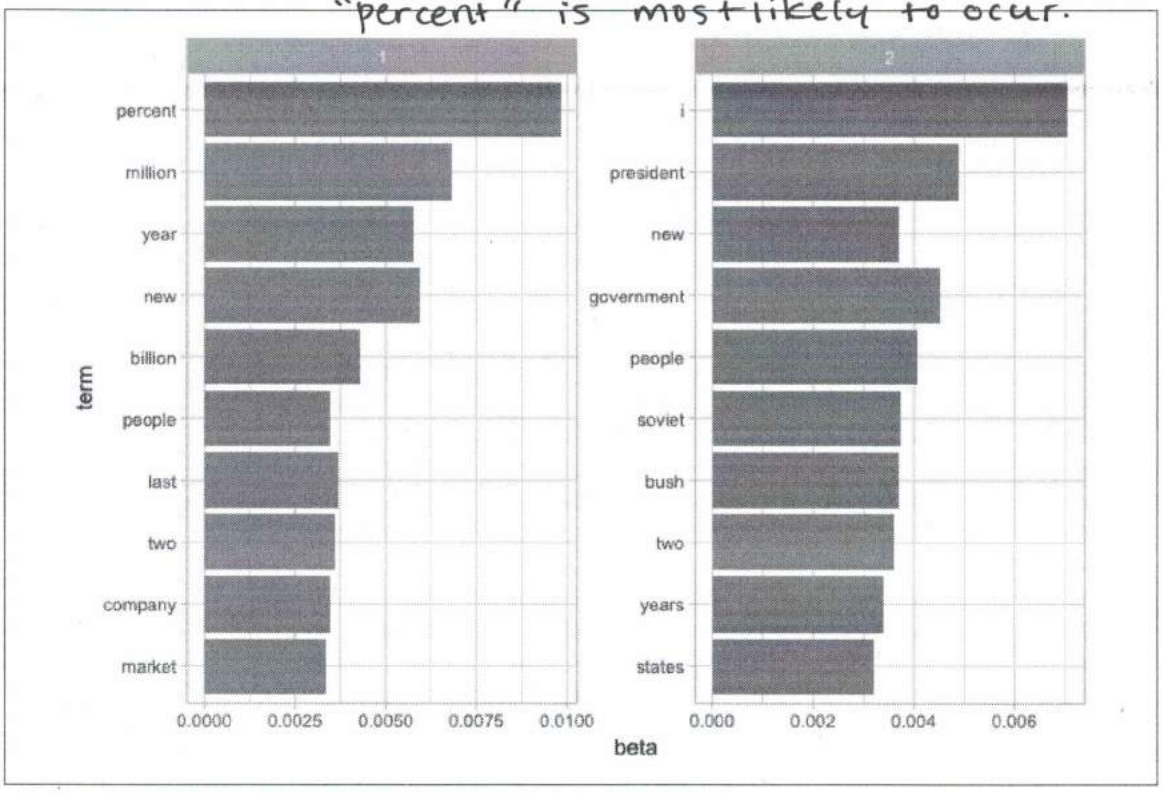


Figure 6-2. The terms that are most common within each topic

This visualization lets us understand the two topics that were extracted from the articles. The most common words in topic 1 include "percent," "million," "billion," and "company," which suggests it may represent business or financial news. Those most common in topic 2 include "president," "government," and "soviet," suggeting that this topic represents political news. One important observation about the words in each topic is that some words, such as "new" and "people," are common within both topics. This is an advantage of topic modeling as opposed to "hard clustering" methods: topics used in natural language could have some overlap in terms of words. As an alternative, we could consider the terms that had the *greatest difference* in β between topic 1 and topic 2. This can be estimated based on the log ratio of the two: $\log_2\left(\frac{\beta_2}{\beta_1}\right)$.

A log ratio is useful because it makes the difference symmetrical: β_2 being twice as large leads to a log ratio of 1, while β_1 being twice as large results in –1.

To constrain it to a set of especially relevant words, we can filter for relatively common words, such as those that have a β greater than 1/1000 in at least one topic.

```
library(tidyr)

beta_spread <- ap_topics %>%
  mutate(topic = paste0("topic", topic)) %>%
  spread(topic, beta) %>%
  filter(topic1 > .001 | topic2 > .001) %>%
  mutate(log_ratio = log2(topic2 / topic1))

beta_spread
```

```
## # A tibble: 198 × 4
##              term       topic1       topic2   log_ratio
##             <chr>        <dbl>        <dbl>       <dbl>
## 1  administration 4.309502e-04 1.382244e-03   1.6814189
## 2             ago 1.065216e-03 8.421279e-04  -0.3390353
## 3       agreement 6.714984e-04 1.039024e-03   0.6297728
## 4             aid 4.759043e-05 1.045958e-03   4.4580091
## 5             air 2.136933e-03 2.966593e-04  -2.8486628
## 6        american 2.030497e-03 1.683884e-03  -0.2700405
## 7        analysts 1.087581e-03 5.779708e-07 -10.8778386
## 8            area 1.371397e-03 2.310280e-04  -2.5695069
## 9            army 2.622192e-04 1.048089e-03   1.9989152
## 10          asked 1.885803e-04 1.559209e-03   3.0475641
## # ... with 188 more rows
```

The words with the greatest differences between the two topics are visualized in Figure 6-3.

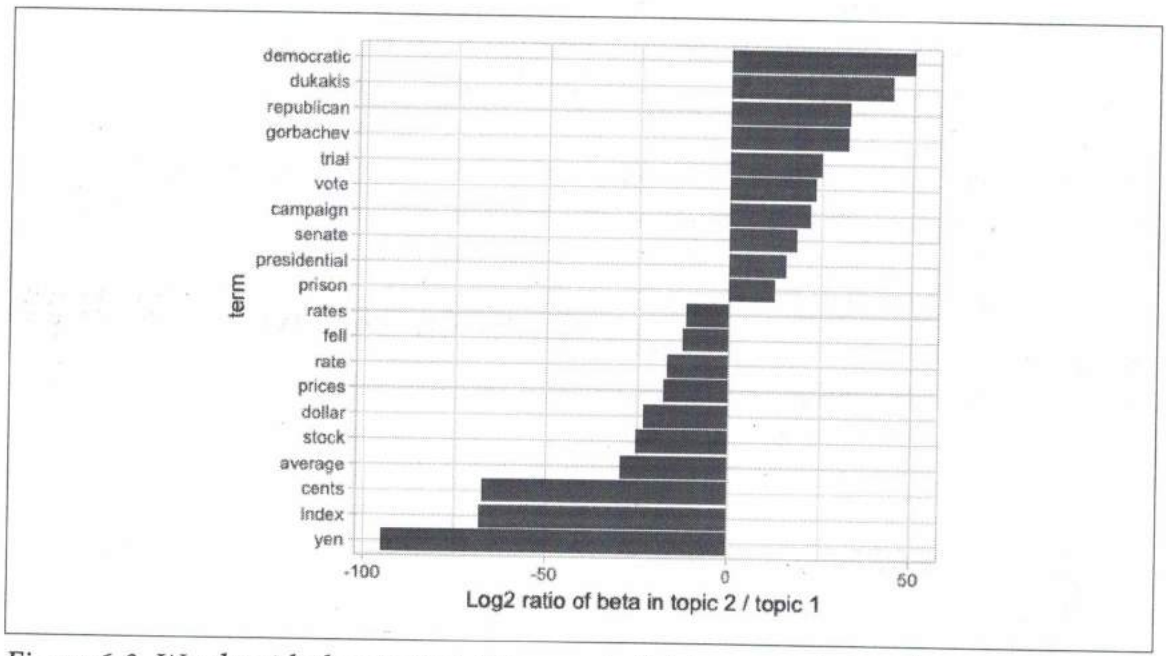


Figure 6-3. Words with the greatest difference in β between topic 2 and topic 1

We can see that the words more common in topic 2 include political parties such as "democratic" and "republican," as well as politician's names such as "dukakis" and

"gorbachev." Topic 1 is more characterized by currencies like "yen" and "dollar," as well as financial terms such as "index," "prices," and "rates." This helps confirm that the two topics the algorithm identified are political and financial news.

Document-Topic Probabilities

Besides estimating each topic as a mixture of words, LDA also models each document as a mixture of topics. We can examine the per-document-per-topic probabilities, called γ ("gamma"), with the `matrix = "gamma"` argument to `tidy()`.

```
ap_documents <- tidy(ap_lda, matrix = "gamma")
ap_documents
```

```
## # A tibble: 4,492 × 3
##    document topic        gamma
##       <int> <int>        <dbl>
## 1         1     1 0.2480616686
## 2         2     1 0.3615485445
## 3         3     1 0.5265844180
## 4         4     1 0.3566530023
## 5         5     1 0.1812766762
## 6         6     1 0.0005883388
## 7         7     1 0.7734215655
## 8         8     1 0.0044516994
## 9         9     1 0.9669915139
## 10       10     1 0.1468904793
## # ... with 4,482 more rows
```

Each of these values is an estimated proportion of words from that document that are generated from that topic. For example, the model estimates that only about 24.8% of the words in document 1 are generated from topic 1.

or basically, what's the p that a randomly selected word in doc = is of topic y?

We can see that many of these documents are drawn from a mix of the two topics, but that document 6 is drawn almost entirely from topic 2, having a γ from topic 1 close to zero. To check this answer, we could `tidy()` the document-term matrix (see "Tidying a Document-Term Matrix" on page 70) and check what the most common words in that document are.

```
tidy(AssociatedPress) %>%
  filter(document == 6) %>%
  arrange(desc(count))
```

```
## # A tibble: 287 × 3
##    document           term count
##       <int>          <chr> <dbl>
## 1         6        noriega    16
## 2         6         panama    12
## 3         6        jackson     6
## 4         6         powell     6
## 5         6 administration     5
## 6         6       economic     5
```

```
## 7         6         general     5
## 8         6               i     5
## 9         6      panamanian     5
## 10        6        american     4
## # ... with 277 more rows
```

Based on the most common words, this appears to be an article about the relationship between the American government and Panamanian dictator Manuel Noriega, which means the algorithm was right to place it in topic 2 (as political/national news).

Example: The Great Library Heist

When examining a statistical method, it can be useful to try it on a very simple case where you know the "right answer." For example, we could collect a set of documents that definitely relate to four separate topics, then perform topic modeling to see whether the algorithm can correctly distinguish the four groups. This lets us double-check that the method is useful, and gain a sense of how and when it can go wrong. We'll try this with some data from classic literature.

Suppose a vandal has broken into your study and torn apart four of your books:

- *Great Expectations* by Charles Dickens
- *The War of the Worlds* by H.G. Wells
- *Twenty Thousand Leagues Under the Sea* by Jules Verne
- *Pride and Prejudice* by Jane Austen

This vandal has torn the books into individual chapters, and left them in one large pile. How can we restore these disorganized chapters to their original books? This is a challenging problem since the individual chapters are *unlabeled*: we don't know what words might distinguish them into groups. We'll thus use topic modeling to discover how chapters cluster into distinct topics, each of them (presumably) representing one of the books.

We'll retrieve the text of these four books using the gutenbergr package introduced in Chapter 3.

```
titles <- c("Twenty Thousand Leagues under the Sea", "The War of the Worlds",
            "Pride and Prejudice", "Great Expectations")

library(gutenbergr)

books <- gutenberg_works(title %in% titles) %>%
  gutenberg_download(meta_fields = "title")
```

As preprocessing, we divide these into chapters, use tidytext's `unnest_tokens()` to separate them into words, then remove `stop_words`. We're treating every chapter as a separate "document," each with a name like `Great Expectations_1` or `Pride and`

`Prejudice_11`. (In other applications, each document might be one newspaper article, or one blog post).

```
library(stringr)

# divide into documents, each representing one chapter
reg <- regex("^chapter ", ignore_case = TRUE)
by_chapter <- books %>%
  group_by(title) %>%
  mutate(chapter = cumsum(str_detect(text, reg))) %>%
  ungroup() %>%
  filter(chapter > 0) %>%
  unite(document, title, chapter)

# split into words
by_chapter_word <- by_chapter %>%
  unnest_tokens(word, text)

# find document-word counts
word_counts <- by_chapter_word %>%
  anti_join(stop_words) %>%
  count(document, word, sort = TRUE) %>%
  ungroup()

word_counts
```

```
## # A tibble: 104,721 × 3
##                    document    word     n
##                       <chr>   <chr> <int>
## 1     Great Expectations_57     joe    88
## 2      Great Expectations_7     joe    70
## 3     Great Expectations_17   biddy    63
## 4     Great Expectations_27     joe    58
## 5     Great Expectations_38 estella    58
## 6      Great Expectations_2     joe    56
## 7     Great Expectations_23  pocket    53
## 8     Great Expectations_15     joe    50
## 9     Great Expectations_18     joe    50
## 10 The War of the Worlds_16 brother    50
## # ... with 104,711 more rows
```

LDA on Chapters

Right now our data frame `word_counts` is in a tidy form, with one term per document per row, but the topicmodels package requires a `DocumentTermMatrix`. As described in "Casting Tidy Text Data into a Matrix" on page 77, we can cast a one-token-per-row table into a `DocumentTermMatrix` with tidytext's `cast_dtm()`.

```
chapters_dtm <- word_counts %>%
  cast_dtm(document, word, n)

chapters_dtm
```

```
## <<DocumentTermMatrix (documents: 193, terms: 18215)>>
## Non-/sparse entries: 104721/3410774
## Sparsity           : 97%
## Maximal term length: 19
## Weighting          : term frequency (tf)
```

2 we picked 4 since we have 4 books

We can then use the `LDA()` function to create a four-topic model. In this case we know we're looking for four topics because there are four books; in other problems we may need to try a few different values of k.

```
chapters_lda <- LDA(chapters_dtm, k = 4, control = list(seed = 1234))
chapters_lda
```

```
## A LDA_VEM topic model with 4 topics.
```

Much as we did on the Associated Press data, we can examine per-topic-per-word probabilities.

```
chapter_topics <- tidy(chapters_lda, matrix = "beta")
chapter_topics
```

```
## # A tibble: 72,860 × 3
##    topic    term         beta
##    <int>   <chr>        <dbl>
## 1      1     joe 5.830326e-17
## 2      2     joe 3.194447e-57
## 3      3     joe 4.162676e-24
## 4      4     joe 1.445030e-02
## 5      1   biddy 7.846976e-27
## 6      2   biddy 4.672244e-69
## 7      3   biddy 2.259711e-46
## 8      4   biddy 4.767972e-03
## 9      1 estella 3.827272e-06
## 10     2 estella 5.316964e-65
## # ... with 72,850 more rows
```

Notice that this has turned the model into a one-topic-per-term-per-row format. For each combination, the model computes the probability of that term being generated from that topic. For example, the term "joe" has an almost zero probability of being generated from topics 1, 2, or 3, but it makes up 1.45% of topic 4.

We could use dplyr's `top_n()` to find the top five terms within each topic.

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

```
top_terms
```

```
## # A tibble: 20 × 3
##    topic      term        beta
##    <int>     <chr>       <dbl>
## 1      1 elizabeth 0.014107538
## 2      1     darcy 0.008814258
## 3      1      miss 0.008706741
## 4      1    bennet 0.006947431
## 5      1      jane 0.006497512
## 6      2   captain 0.015507696
## 7      2  nautilus 0.013050048
## 8      2       sea 0.008850073
## 9      2      nemo 0.008708397
## 10     2       ned 0.008030799
## 11     3    people 0.006797400
## 12     3  martians 0.006512569
## 13     3      time 0.005347115
## 14     3     black 0.005278302
## 15     3     night 0.004483143
## 16     4       joe 0.014450300
## 17     4      time 0.006847574
## 18     4       pip 0.006817363
## 19     4    looked 0.006365257
## 20     4      miss 0.006228387
```

interesting... "conseil" shows up in mine

This tidy output lends itself well to a ggplot2 visualization (Figure 6-4).

```
library(ggplot2)

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()
```

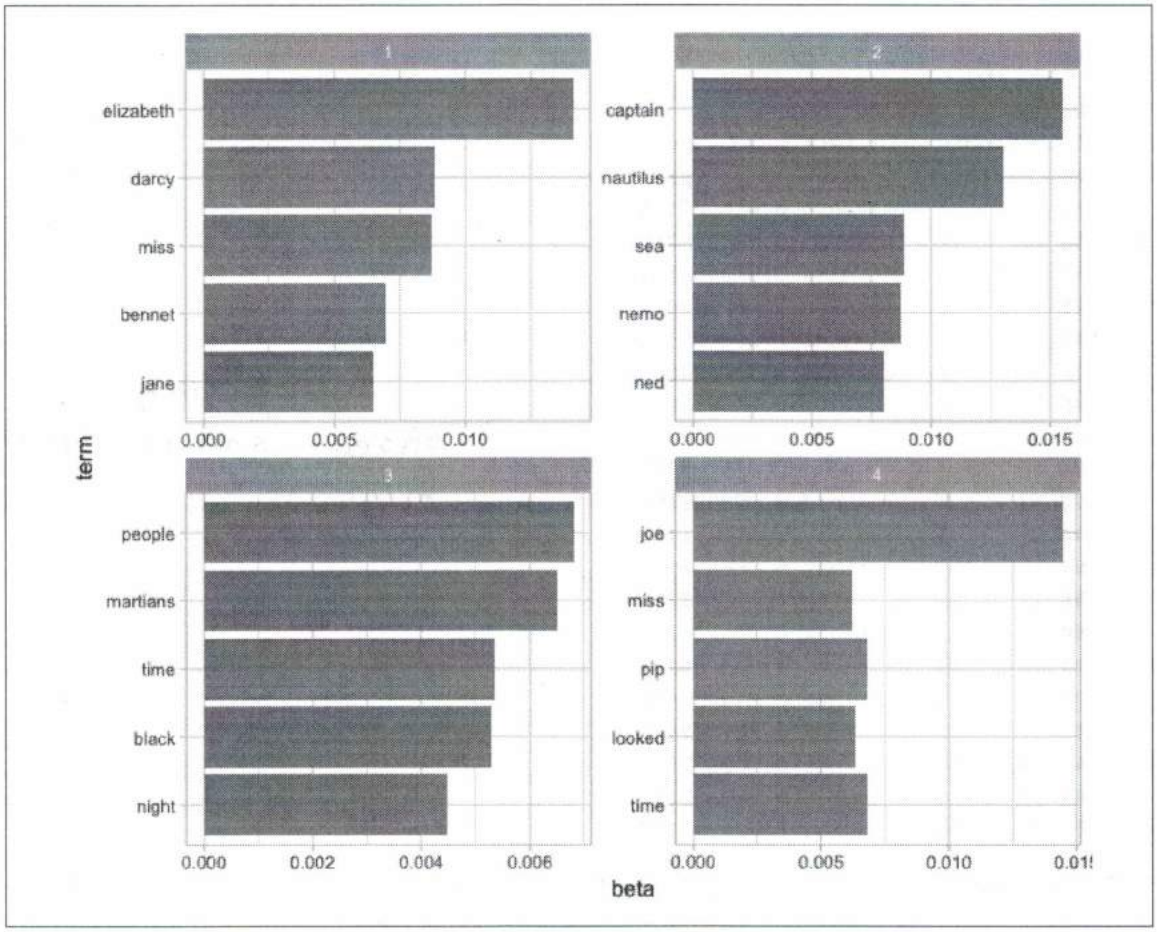


Figure 6-4. The terms that are most common within each topic

These topics are pretty clearly associated with the four books! There's no question that the topic of "captain," "nautilus," "sea," and "nemo" belongs to *Twenty Thousand Leagues Under the Sea*; and that "jane," "darcy," and "elizabeth" belongs to *Pride and Prejudice*. We see "pip" and "joe" from *Great Expectations*, and "martians," "black," and "night" from *The War of the Worlds*. We also notice that, in line with LDA being a "fuzzy clustering" method, there can be words in common between multiple topics, such as "miss" in topics 1 and 4, and "time" in topics 3 and 4.

Per-Document Classification

Each document in this analysis represented a single chapter. Thus, we may want to know which topics are associated with each document. Can we put the chapters back together in the correct books? We can find this by examining the per-document-per-topic probabilities, γ ("gamma").

```
chapters_gamma <- tidy(chapters_lda, matrix = "gamma")
chapters_gamma
```

```
## # A tibble: 772 × 3
##                    document topic        gamma
##                       <chr> <int>        <dbl>
## 1     Great Expectations_57     1 1.351886e-05
## 2      Great Expectations_7     1 1.470726e-05
## 3     Great Expectations_17     1 2.117127e-05
## 4     Great Expectations_27     1 1.919746e-05
## 5     Great Expectations_38     1 3.544403e-01
## 6      Great Expectations_2     1 1.723723e-05
## 7     Great Expectations_23     1 5.507241e-01
## 8     Great Expectations_15     1 1.682503e-02
## 9     Great Expectations_18     1 1.272044e-05
## 10 The War of the Worlds_16     1 1.084337e-05
## # ... with 762 more rows
```

Each of these values is an estimated proportion of words from that document that are generated from that topic. For example, the model estimates that each word in the `Great Expectations_57` document has only a 0.00135% probability of coming from topic 1 (*Pride and Prejudice*).

Now that we have these topic probabilities, we can see how well our unsupervised learning did at distinguishing the four books. We'd expect that chapters within a book would be found to be mostly (or entirely) generated from the corresponding topic.

First we re-separate the document name into title and chapter, after which we can visualize the per-document-per-topic probability for each (Figure 6-5).

```
chapters_gamma <- chapters_gamma %>%
  separate(document, c("title", "chapter"), sep = "_", convert = TRUE)

chapters_gamma
```

```
## # A tibble: 772 × 4
##                    title chapter topic        gamma
## *                  <chr>   <int> <int>        <dbl>
## 1     Great Expectations      57     1 1.351886e-05
## 2     Great Expectations       7     1 1.470726e-05
## 3     Great Expectations      17     1 2.117127e-05
## 4     Great Expectations      27     1 1.919746e-05
## 5     Great Expectations      38     1 3.544403e-01
## 6     Great Expectations       2     1 1.723723e-05
## 7     Great Expectations      23     1 5.507241e-01
## 8     Great Expectations      15     1 1.682503e-02
## 9     Great Expectations      18     1 1.272044e-05
## 10 The War of the Worlds      16     1 1.084337e-05
## # ... with 762 more rows
```

```
# reorder titles in order of topic 1, topic 2, etc. before plotting
chapters_gamma %>%
  mutate(title = reorder(title, gamma * topic)) %>%
  ggplot(aes(factor(topic), gamma)) +
  geom_boxplot() +
  facet_wrap(~ title)
```

This is such a great technique. I wonder where it's been used IRL, eg to ID if texts fragments belonged to Shakespeare or Marlowe?

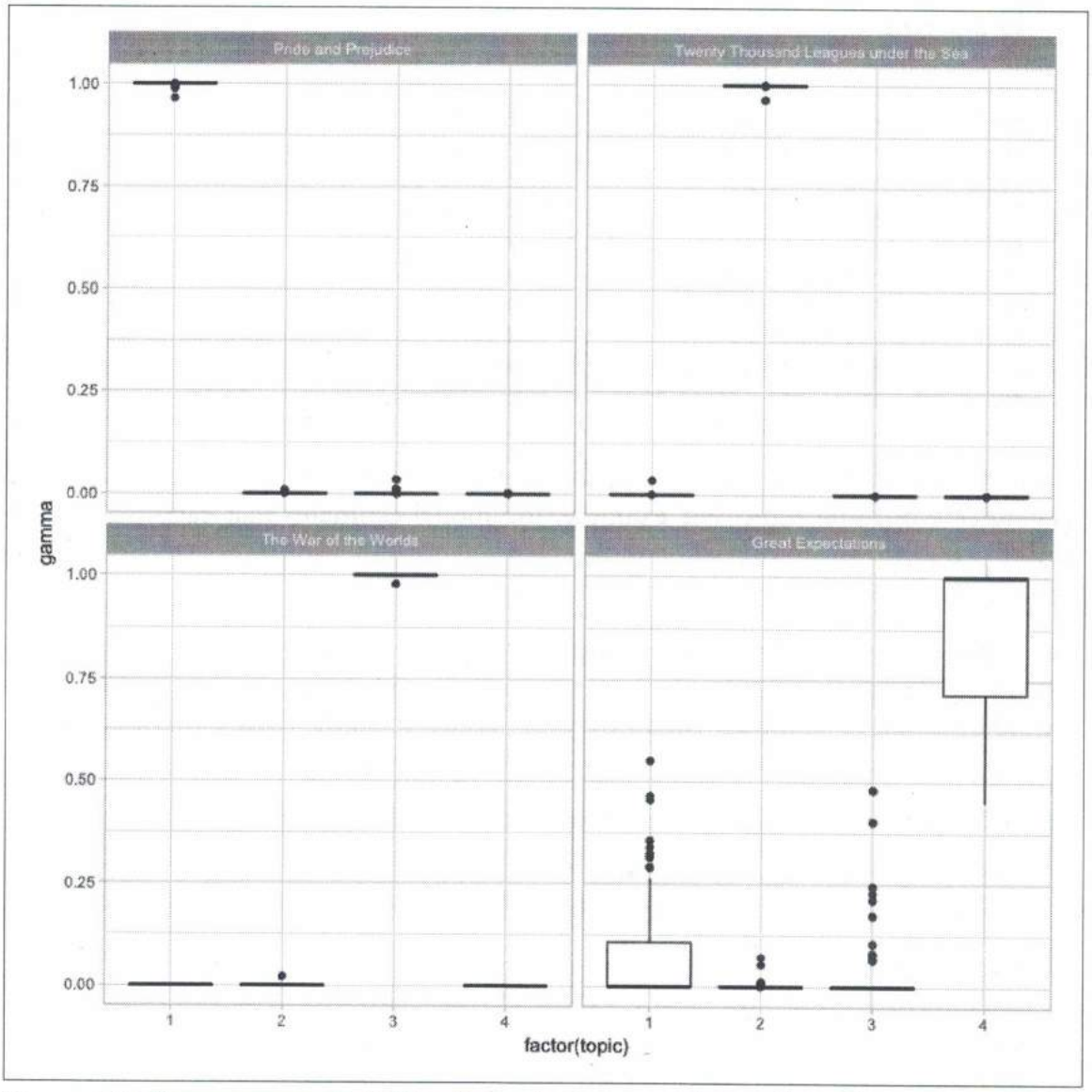


Figure 6-5. The gamma probabilities for each chapter within each book

what can we do to increase accuracy?

We notice that almost all of the chapters from *Pride and Prejudice*, *The War of the Worlds*, and *Twenty Thousand Leagues Under the Sea* were uniquely identified as a single topic each.

It does look like some chapters from *Great Expectations* (which should be topic 4) were somewhat associated with other topics. Are there any cases where the topic most associated with a chapter belonged to another book? First we'd find the topic that was most associated with each chapter using `top_n()`, which is effectively the "classification" of that chapter.

```
chapter_classifications <- chapters_gamma %>%
  group_by(title, chapter) %>%
  top_n(1, gamma) %>%
```

```
  ungroup()

chapter_classifications
```

```
## # A tibble: 193 × 4
##                  title chapter topic     gamma
##                  <chr>   <int> <int>     <dbl>
## 1   Great Expectations      23     1 0.5507241
## 2  Pride and Prejudice      43     1 0.9999610
## 3  Pride and Prejudice      18     1 0.9999654
## 4  Pride and Prejudice      45     1 0.9999038
## 5  Pride and Prejudice      16     1 0.9999466
## 6  Pride and Prejudice      29     1 0.9999300
## 7  Pride and Prejudice      10     1 0.9999203
## 8  Pride and Prejudice       8     1 0.9999134
## 9  Pride and Prejudice      56     1 0.9999337
## 10 Pride and Prejudice      47     1 0.9999506
## # ... with 183 more rows
```

We can then compare each to the "consensus" topic for each book (the most common topic among its chapters), and see which were most often misidentified.

```
book_topics <- chapter_classifications %>%
  count(title, topic) %>%
  group_by(title) %>%
  top_n(1, n) %>%
  ungroup() %>%
  transmute(consensus = title, topic)

chapter_classifications %>%
  inner_join(book_topics, by = "topic") %>%
  filter(title != consensus)
```

```
## # A tibble: 2 × 5
##                title chapter topic     gamma              consensus
##                <chr>   <int> <int>     <dbl>                  <chr>
## 1 Great Expectations      23     1 0.5507241   Pride and Prejudice
## 2 Great Expectations      54     3 0.4803234 The War of the Worlds
```

We see that only two chapters from *Great Expectations* were misclassified, as LDA described one as coming from the *Pride and Prejudice* topic (topic 1) and one from *The War of the Worlds* (topic 3). That's not bad for unsupervised clustering!

By-Word Assignments: augment

One step of the LDA algorithm is assigning each word in each document to a topic. The more words in a document are assigned to that topic, generally, the more weight (`gamma`) will go on that document-topic classification.

We may want to take the original document-word pairs and find which words in each document were assigned to which topic. This is the job of the `augment()` function, which also originated in the broom package as a way of tidying model output. While

`tidy()` retrieves the statistical components of the model, `augment()` uses a model to add information to each observation in the original data.

```
assignments <- augment(chapters_lda, data = chapters_dtm)
assignments
```

```
## # A tibble: 104,721 × 4
##                  document  term count .topic
##                     <chr> <chr> <dbl>  <dbl>
## 1  Great Expectations_57   joe    88      4
## 2   Great Expectations_7   joe    70      4
## 3  Great Expectations_17   joe     5      4
## 4  Great Expectations_27   joe    58      4
## 5   Great Expectations_2   joe    56      4
## 6  Great Expectations_23   joe     1      4
## 7  Great Expectations_15   joe    50      4
## 8  Great Expectations_18   joe    50      4
## 9   Great Expectations_9   joe    44      4
## 10 Great Expectations_13   joe    40      4
## # ... with 104,711 more rows
```

This returns a tidy data frame of book-term counts, but adds an extra column, `.topic`, with the topic each term was assigned to within each document. (Extra columns added by `augment` always start with `.` to prevent overwriting existing columns.) We can combine this `assignments` table with the consensus book titles to find which words were incorrectly classified.

```
assignments <- assignments %>%
  separate(document, c("title", "chapter"), sep = "_", convert = TRUE) %>%
  inner_join(book_topics, by = c(".topic" = "topic"))

assignments
```

```
## # A tibble: 104,721 × 6
##                  title chapter  term count .topic          consensus
##                  <chr>   <int> <chr> <dbl>  <dbl>              <chr>
## 1  Great Expectations      57   joe    88      4 Great Expectations
## 2  Great Expectations       7   joe    70      4 Great Expectations
## 3  Great Expectations      17   joe     5      4 Great Expectations
## 4  Great Expectations      27   joe    58      4 Great Expectations
## 5  Great Expectations       2   joe    56      4 Great Expectations
## 6  Great Expectations      23   joe     1      4 Great Expectations
## 7  Great Expectations      15   joe    50      4 Great Expectations
## 8  Great Expectations      18   joe    50      4 Great Expectations
## 9  Great Expectations       9   joe    44      4 Great Expectations
## 10 Great Expectations      13   joe    40      4 Great Expectations
## # ... with 104,711 more rows
```

This combination of the true book (`title`) and the book assigned to it (`consensus`) is useful for further exploration. We can, for example, visualize a *confusion matrix*, showing how often words from one book were assigned to another, using dplyr's `count()` and ggplot2's `geom_tile` (Figure 6-6).

```
assignments %>%
  count(title, consensus, wt = count) %>%
  group_by(title) %>%
  mutate(percent = n / sum(n)) %>%
  ggplot(aes(consensus, title, fill = percent)) +
  geom_tile() +
  scale_fill_gradient2(high = "red", label = percent_format()) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1),
        panel.grid = element_blank()) +
  labs(x = "Book words were assigned to",
       y = "Book words came from",
       fill = "% of assignments")
```

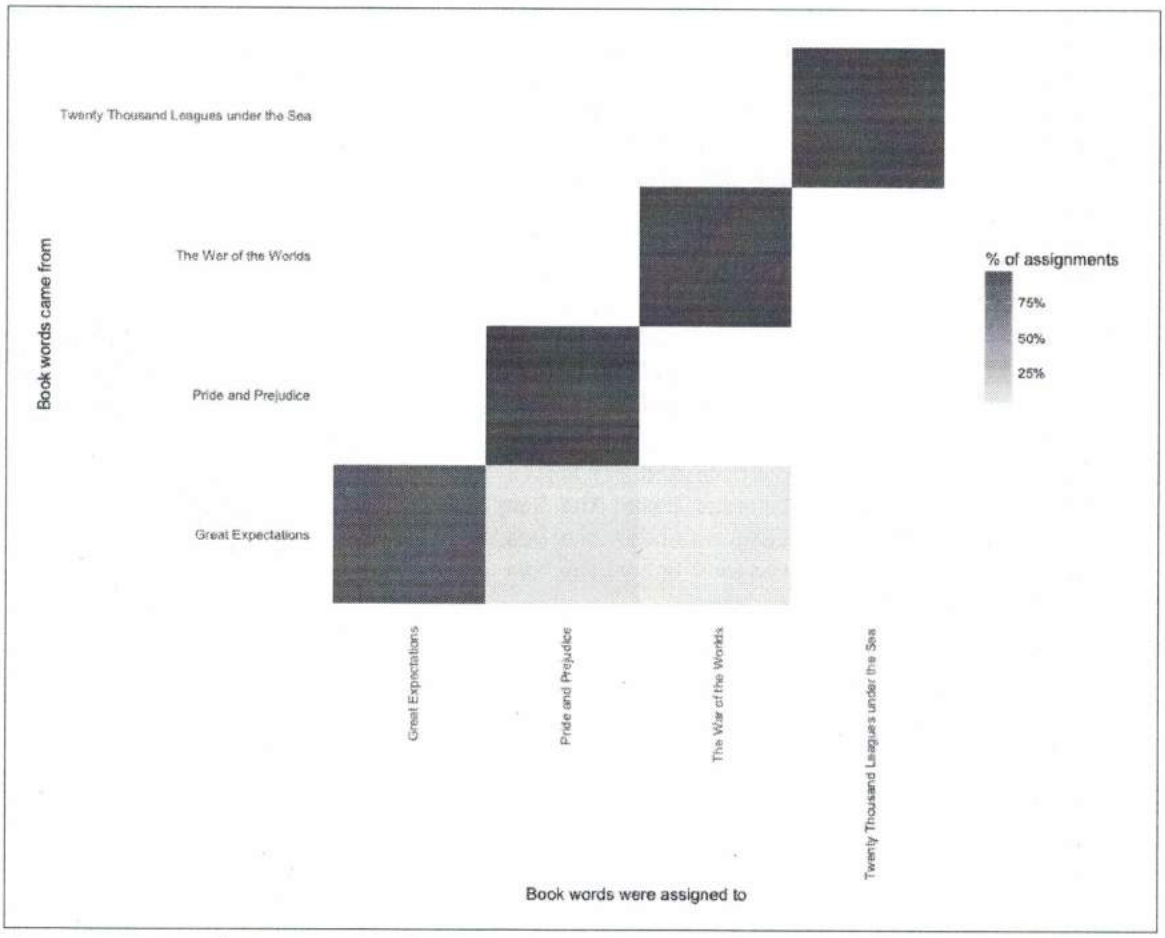


Figure 6-6. Confusion matrix showing where LDA assigned the words from each book. Each row of this table represents the true book each word came from, and each column represents what book it was assigned to.

We notice that almost all the words for *Pride and Prejudice*, *Twenty Thousand Leagues Under the Sea*, and *The War of the Worlds* were correctly assigned, while *Great Expectations* had a fair number of misassigned words (which, as we saw above, led to two chapters getting misclassified).

What were the most commonly mistaken words?

```
wrong_words <- assignments %>%
  filter(title != consensus)

wrong_words
```

```
## # A tibble: 4,535 × 6
##                                    title chapter     term count .topic
##                                    <chr>   <int>    <chr> <dbl>  <dbl>
## 1                     Great Expectations      38  brother     2      1
## 2                     Great Expectations      22  brother     4      1
## 3                     Great Expectations      23     miss     2      1
## 4                     Great Expectations      22     miss    23      1
## 5  Twenty Thousand Leagues under the Sea       8     miss     1      1
## 6                     Great Expectations      31     miss     1      1
## 7                     Great Expectations       5 sergeant    37      1
## 8                     Great Expectations      46  captain     1      2
## 9                     Great Expectations      32  captain     1      2
## 10                The War of the Worlds      17  captain     5      2
##                                 consensus
##                                     <chr>
## 1                   Pride and Prejudice
## 2                   Pride and Prejudice
## 3                   Pride and Prejudice
## 4                   Pride and Prejudice
## 5                   Pride and Prejudice
## 6                   Pride and Prejudice
## 7                   Pride and Prejudice
## 8  Twenty Thousand Leagues under the Sea
## 9  Twenty Thousand Leagues under the Sea
## 10 Twenty Thousand Leagues under the Sea
## # ... with 4,525 more rows
```

```
wrong_words %>%
  count(title, consensus, term, wt = count) %>%
  ungroup() %>%
  arrange(desc(n))
```

```
## # A tibble: 3,500 × 4
##                 title             consensus     term     n
##                 <chr>                 <chr>    <chr> <dbl>
## 1  Great Expectations   Pride and Prejudice     love    44
## 2  Great Expectations   Pride and Prejudice sergeant    37
## 3  Great Expectations   Pride and Prejudice     lady    32
## 4  Great Expectations   Pride and Prejudice     miss    26
## 5  Great Expectations The War of the Worlds     boat    25
## 6  Great Expectations   Pride and Prejudice   father    19
## 7  Great Expectations The War of the Worlds    water    19
## 8  Great Expectations   Pride and Prejudice     baby    18
## 9  Great Expectations   Pride and Prejudice  flopson    18
## 10 Great Expectations   Pride and Prejudice   family    16
## # ... with 3,490 more rows
```

We can see that a number of words were often assigned to the *Pride and Prejudice* or *War of the Worlds* cluster even when they appeared in *Great Expectations*. For some of these words, such as "love" and "lady," that's because they're more common in *Pride and Prejudice* (we could confirm that by examining the counts).

On the other hand, there are a few wrongly classified words that never appeared in the novel they were misassigned to. For example, we can confirm "flopson" appears only in *Great Expectations*, even though it's assigned to the *Pride and Prejudice* cluster.

```
word_counts %>%
  filter(word == "flopson")
```

```
## # A tibble: 3 × 3
##                document    word     n
##                   <chr>   <chr> <int>
## 1 Great Expectations_22 flopson    10
## 2 Great Expectations_23 flopson     7
## 3 Great Expectations_33 flopson     1
```

The LDA algorithm is stochastic, and it can accidentally land on a topic that spans multiple books.

Alternative LDA Implementations

The `LDA()` function in the topicmodels package is only one implementation of the latent Dirichlet allocation algorithm. For example, the mallet package (Mimno 2013) implements a wrapper around the MALLET Java package for text classification tools, and the tidytext package provides tidiers for this model output as well.

The mallet package takes a somewhat different approach to the input format. For instance, it takes nontokenized documents and performs the tokenization itself, and requires a separate file of stop words. This means we have to collapse the text into one string for each document before performing LDA.

```
library(mallet)

# create a vector with one string per chapter
collapsed <- by_chapter_word %>%
  anti_join(stop_words, by = "word") %>%
  mutate(word = str_replace(word, "'", "")) %>%
  group_by(document) %>%
  summarize(text = paste(word, collapse = " "))

# create an empty file of "stop words"
file.create(empty_file <- tempfile())
docs <- mallet.import(collapsed$document, collapsed$text, empty_file)

mallet_model <- MalletLDA(num.topics = 4)
mallet_model$loadDocuments(docs)
mallet_model$train(100)
```

Once the model is created, however, we can use the `tidy()` and `augment()` functions described in the rest of the chapter in an almost identical way. This includes extracting the probabilities of words within each topic or topics within each document.

```
# word-topic pairs
tidy(mallet_model)

# document-topic pairs
tidy(mallet_model, matrix = "gamma")

# column needs to be named "term" for "augment"
term_counts <- rename(word_counts, term = word)
augment(mallet_model, term_counts)
```

We could use ggplot2 to explore and visualize the model in the same way we did the LDA output.

Summary

This chapter introduced topic modeling for finding clusters of words that characterize a set of documents, and showed how the `tidy()` verb lets us explore and understand these models using dplyr and ggplot2. This is one of the advantages of the tidy approach to model exploration: the challenges of different output formats are handled by the tidying functions, and we can explore model results using a standard set of tools. In particular, we saw that topic modeling is able to separate and distinguish chapters from four separate books, and explored the limitations of the model by finding words and chapters that it assigned incorrectly.

CHAPTER 7

Case Study: Comparing Twitter Archives

One type of text that gets plenty of attention is text shared online via Twitter. In fact, several of the sentiment lexicons used in this book (and commonly used in general) were designed for use with and validated on tweets. Both authors of this book are on Twitter and are fairly regular users of it, so in this case study, let's compare the entire Twitter archives of Julia and David.

Getting the Data and Distribution of Tweets

An individual can download his or her own Twitter archive by following directions available on Twitter's website. We each downloaded ours and will now open them up. Let's use the lubridate package to convert the string timestamps to date-time objects and initially take a look at our tweeting patterns overall (Figure 7-1).

```
library(lubridate)
library(ggplot2)
library(dplyr)
library(readr)

tweets_julia <- read_csv("data/tweets_julia.csv")
tweets_dave <- read_csv("data/tweets_dave.csv")
tweets <- bind_rows(tweets_julia %>%
                      mutate(person = "Julia"),
                    tweets_dave %>%
                      mutate(person = "David")) %>%
  mutate(timestamp = ymd_hms(timestamp))

ggplot(tweets, aes(x = timestamp, fill = person)) +
  geom_histogram(position = "identity", bins = 20, show.legend = FALSE) +
  facet_wrap(~person, ncol = 1)
```

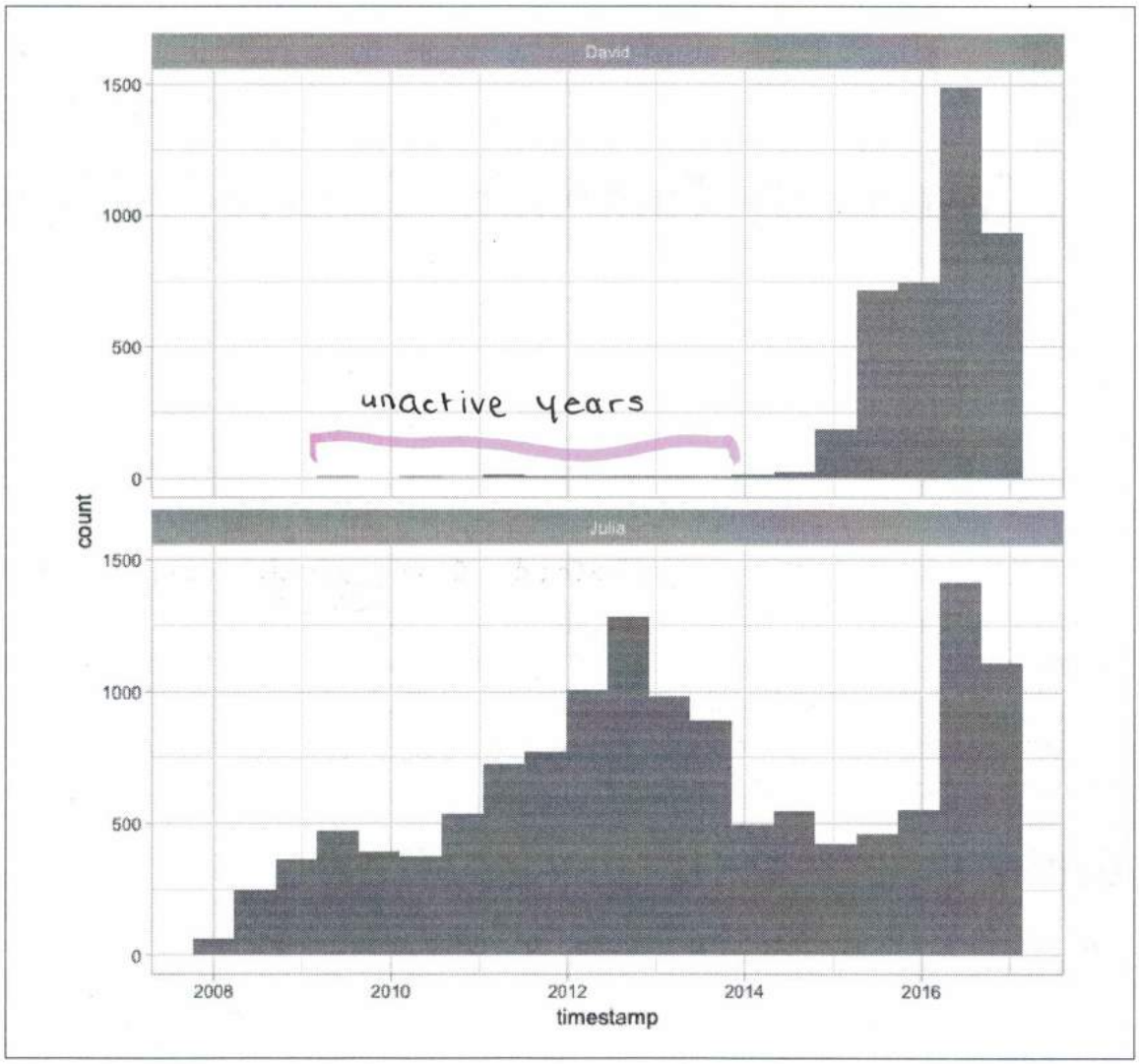


Figure 7-1. All tweets from our accounts

David and Julia tweet at about the same rate currently and joined Twitter about a year apart from each other, but there were about five years where David was not active on Twitter and Julia was. In total, Julia has about four times as many tweets as David.

Word Frequencies

Let's use `unnest_tokens()` to make a tidy data frame of all the words in our tweets, and remove the common English stop words. There are certain conventions in how people use text on Twitter, so we will do a bit more work with our text here than, for example, we did with the narrative text from Project Gutenberg.

First, we will remove tweets from this dataset that are retweets so that we only have tweets that we wrote ourselves. Next, the `mutate()` line removes links and cleans out some characters that we don't want, like ampersands and such.

In the call to `unnest_tokens()`, we unnest using a regex pattern, instead of just looking for single unigrams (words). This regex pattern is very useful for dealing with Twitter text; it retains hashtags and mentions of usernames with the `@` symbol.

Because we have kept these types of symbols in the text, we can't use a simple `anti_join()` to remove stop words. Instead, we can take the approach shown in the `filter()` line that uses `str_detect()` from the stringr package.

```
library(tidytext)
library(stringr)

replace_reg1 <- "https://t.co/[A-Za-z\\d]+|"
replace_reg2 <- "http://[A-Za-z\\d]+|&amp;|&lt;|&gt;|RT|https"
replace_reg <- paste0(replace_reg1, replace_reg2)
unnest_reg <- "([^A-Za-z_\\d#@']|'(?![A-Za-z_\\d#@]))"
tidy_tweets <- tweets %>%
  filter(!str_detect(text, "^RT")) %>%
  mutate(text = str_replace_all(text, replace_reg, "")) %>%
  unnest_tokens(word, text, token = "regex", pattern = unnest_reg) %>%
  filter(!word %in% stop_words$word,
         str_detect(word, "[a-z]"))
```

Now we can calculate word frequencies for each person. First, we group by person and count how many times each person used each word. Then we use `left_join()` to add a column of the total number of words used by each person. (This is higher for Julia than David since she has more tweets than David.) Finally, we calculate a frequency for each person and word.

```
frequency <- tidy_tweets %>%
  group_by(person) %>%
  count(word, sort = TRUE) %>%
  left_join(tidy_tweets %>%
              group_by(person) %>%
              summarise(total = n())) %>%
  mutate(freq = n/total)

frequency
```

```
## Source: local data frame [20,736 x 5]
## Groups: person [2]
##
##    person           word     n total        freq
##     <chr>          <chr> <int> <int>       <dbl>
## 1   Julia           time   584 74572 0.007831358
## 2   Julia    @selkie1970   570 74572 0.007643620
## 3   Julia       @skedman   531 74572 0.007120635
## 4   Julia            day   467 74572 0.006262404
## 5   Julia           baby   408 74572 0.005471222
## 6   David @hadleywickham   315 20161 0.015624225
## 7   Julia           love   304 74572 0.004076597
```

```
## 8   Julia   @haleynburke   299 74572 0.004009548
## 9   Julia          house   289 74572 0.003875449
## 10  Julia        morning   278 74572 0.003727941
## # ... with 20,726 more rows
```

This is a nice and tidy data frame, but we would actually like to plot those frequencies on the x- and y-axes of a plot, so we will need to use `spread()` from tidyr to make a differently shaped data frame.

```
library(tidyr)

frequency <- frequency %>%
  select(person, word, freq) %>%
  spread(person, freq) %>%
  arrange(Julia, David)

frequency
```

```
## # A tibble: 17,640 × 3
##                word        David        Julia
##               <chr>        <dbl>        <dbl>
## 1                 's 4.960071e-05 1.340986e-05
## 2  @accidental__art 4.960071e-05 1.340986e-05
## 3       @alice_data 4.960071e-05 1.340986e-05
## 4        @alistaire 4.960071e-05 1.340986e-05
## 5       @corynissen 4.960071e-05 1.340986e-05
## 6     @jennybryan's 4.960071e-05 1.340986e-05
## 7           @jsvine 4.960071e-05 1.340986e-05
## 8     @lizasperling 4.960071e-05 1.340986e-05
## 9       @ognyanova 4.960071e-05 1.340986e-05
## 10      @rbloggers 4.960071e-05 1.340986e-05
## # ... with 17,630 more rows
```

Now this is ready for us to plot. Let's use `geom_jitter()` so that we don't see the discreteness at the low end of frequency as much, and `check_overlap = TRUE` so the text labels don't all print out on top of each other (only some will print; see Figure 7-2).

```
library(scales)

ggplot(frequency, aes(Julia, David)) +
  geom_jitter(alpha = 0.1, size = 2.5, width = 0.25, height = 0.25) +
  geom_text(aes(label = word), check_overlap = TRUE, vjust = 1.5) +
  scale_x_log10(labels = percent_format()) +
  scale_y_log10(labels = percent_format()) +
  geom_abline(color = "red")
```

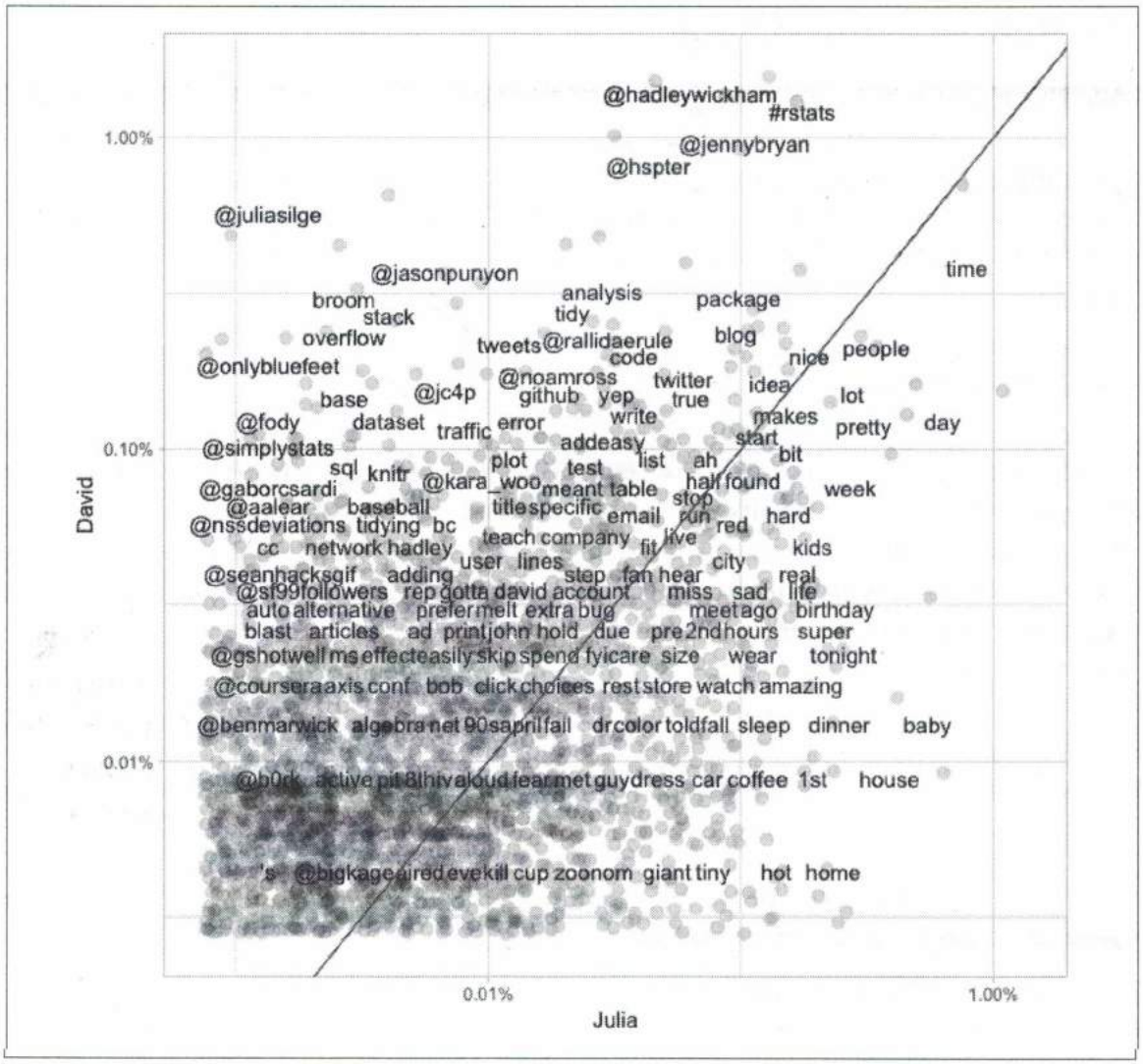


Figure 7-2. Comparing the frequency of words used by Julia and David

Do this test on Béren & Luthiens multiple texts too.

Words near the line in Figure 7-2 are used with about equal frequencies by David and Julia, while words far away from the line are used much more by one person compared to the other. Words, hashtags, and usernames that appear in this plot are ones that we have both used at least once in tweets.

This may not even need to be pointed out, but David and Julia have used their Twitter accounts rather differently over the course of the past several years. David has used his Twitter account almost exclusively for professional purposes since he became more active, while Julia used it for entirely personal purposes until late 2015 and still uses it more personally than David. We see these differences immediately in this plot exploring word frequencies, and they will continue to be obvious in the rest of this chapter.

Comparing Word Usage

We just made a plot comparing raw word frequencies over our whole Twitter histories; now let's find which words are more or less likely to come from each person's account using the log odds ratio. First, let's restrict the analysis moving forward to tweets from David and Julia sent during 2016. David was consistently active on Twitter for all of 2016, and this was about when Julia transitioned into data science as a career.

```
tidy_tweets <- tidy_tweets %>%
  filter(timestamp >= as.Date("2016-01-01"),
         timestamp < as.Date("2017-01-01"))
```

Next, let's use `str_detect()` to remove Twitter usernames from the `word` column, because otherwise, the results here are dominated only by people who Julia or David know and the other does not. After removing these, we count how many times each person uses each word and keep only the words used more than 10 times. After a `spread()` operation, we can calculate the log odds ratio for each word, using:

$$\text{log odds ratio} = \ln\left(\frac{\left[\frac{n+1}{\text{total}+1}\right]_{\text{David}}}{\left[\frac{n+1}{\text{total}+1}\right]_{\text{Julia}}}\right)$$

where n is the number of times the word in question is used by each person, and the total indicates the total words for each person.

```
word_ratios <- tidy_tweets %>%
  filter(!str_detect(word, "^@")) %>%
  count(word, person) %>%
  filter(sum(n) >= 10) %>%
  ungroup() %>%
  spread(person, n, fill = 0) %>%
  mutate_if(is.numeric, funs((. + 1) / sum(. + 1))) %>%
  mutate(logratio = log(David / Julia)) %>%
  arrange(desc(logratio))
```

What are some words that have been about equally likely to come from David's or Julia's account during 2016?

```
word_ratios %>%
  arrange(abs(logratio))
```

```
## # A tibble: 377 × 4
##          word       David       Julia    logratio
##         <chr>       <dbl>       <dbl>       <dbl>
## 1          map 0.002321655 0.002314815 0.002950476
## 2        email 0.002110595 0.002083333 0.013000812
## 3         file 0.002110595 0.002083333 0.013000812
## 4        names 0.003799071 0.003703704 0.025423332
```

```
## 5     account 0.001688476 0.001620370 0.041171689
## 6         api 0.001688476 0.001620370 0.041171689
## 7    function 0.003376952 0.003240741 0.041171689
## 8  population 0.001688476 0.001620370 0.041171689
## 9         sad 0.001688476 0.001620370 0.041171689
## 10      words 0.003376952 0.003240741 0.041171689
## # ... with 367 more rows
```

We are about equally likely to tweet about maps, email, APIs, and functions.

Which words are most likely to be from Julia's account or from David's account? Let's just take the top 15 most distinctive words for each account and plot them in Figure 7-3.

```
word_ratios %>%
  group_by(logratio < 0) %>%
  top_n(15, abs(logratio)) %>%
  ungroup() %>%
  mutate(word = reorder(word, logratio)) %>%
  ggplot(aes(word, logratio, fill = logratio < 0)) +
  geom_col(show.legend = FALSE) +
  coord_flip() +
  ylab("log odds ratio (David/Julia)") +
  scale_fill_discrete(name = "", labels = c("David", "Julia"))
```

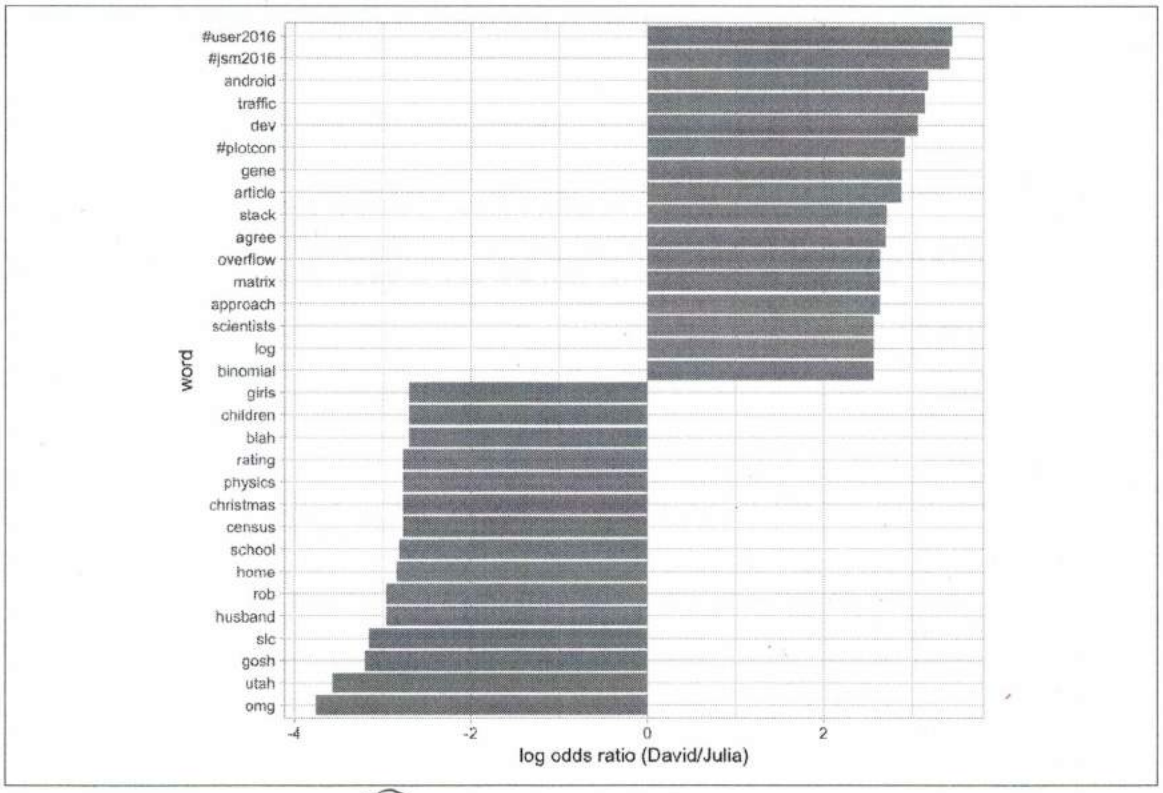


Figure 7-3. Comparing the odds ratios of words from our accounts

So David has tweeted about specific conferences he has gone to, genes, Stack Overflow, and matrices; while Julia tweeted about Utah, physics, Census data, Christmas, and her family.

Changes in Word Use

The section above looked at overall word use, but now let's ask a different question. Which words' frequencies have changed the fastest in our Twitter feeds? Or to state this another way, which words have we tweeted about at a higher or lower rate as time has passed? To do this, we will define a new time variable in the data frame that defines which unit of time each tweet was posted in. We can use `floor_date()` from lubridate to do this, with a unit of our choosing; using 1 month seems to work well for this year of tweets from both of us.

After we have the time bins defined, we count how many times each of us used each word in each time bin. After that, we add columns to the data frame for the total number of words used in each time bin by each person and the total number of times each word was used by each person. We can then `filter()` to only keep words used at least some minimum number of times (30, in this case).

```
words_by_time <- tidy_tweets %>%
  filter(!str_detect(word, "^@")) %>%
  mutate(time_floor = floor_date(timestamp, unit = "1 month")) %>%
  count(time_floor, person, word) %>%
  ungroup() %>%
  group_by(person, time_floor) %>%
  mutate(time_total = sum(n)) %>%
  group_by(word) %>%
  mutate(word_total = sum(n)) %>%
  ungroup() %>%
  rename(count = n) %>%
  filter(word_total > 30)

words_by_time
```

```
## # A tibble: 970 × 6
##    time_floor person    word count time_total word_total
##         <dttm>  <chr>   <chr> <int>      <int>      <int>
## 1  2016-01-01  David #rstats     2        307        324
## 2  2016-01-01  David     bad     1        307         33
## 3  2016-01-01  David     bit     2        307         45
## 4  2016-01-01  David    blog     1        307         60
## 5  2016-01-01  David   broom     2        307         41
## 6  2016-01-01  David    call     2        307         31
## 7  2016-01-01  David   check     1        307         42
## 8  2016-01-01  David    code     3        307         49
## 9  2016-01-01  David    data     2        307        276
## 10 2016-01-01  David     day     2        307         65
## # ... with 960 more rows
```

Each row in this data frame corresponds to one person using one word in a given time bin. The `count` column tells us how many times that person used that word in that time bin, the `time_total` column tells us how many words that person used during that time bin, and the `word_total` column tells us how many times that person used that word over the whole year. This is the data set we can use for modeling.

We can use `nest()` from tidyr to make a data frame with a list column that contains little miniature data frames for each word. Let's do that now and take a look at the resulting structure.

```
nested_data <- words_by_time %>%
  nest(-word, -person)

nested_data
```

```
## # A tibble: 112 × 3
##    person    word              data
##     <chr>   <chr>            <list>
## 1   David #rstats <tibble [12 × 4]>
## 2   David     bad  <tibble [9 × 4]>
## 3   David     bit <tibble [10 × 4]>
## 4   David    blog <tibble [12 × 4]>
## 5   David   broom <tibble [10 × 4]>
## 6   David    call  <tibble [9 × 4]>
## 7   David   check <tibble [12 × 4]>
## 8   David    code <tibble [10 × 4]>
## 9   David    data <tibble [12 × 4]>
## 10  David     day  <tibble [8 × 4]>
## # ... with 102 more rows
```

This data frame has one row for each person-word combination; the `data` column is a list column that contains data frames, one for each combination of person and word. Let's use `map()` from the purrr library to apply our modeling procedure to each of those little data frames inside our big data frame. This is count data, so let's use `glm()` with `family = "binomial"` for modeling.

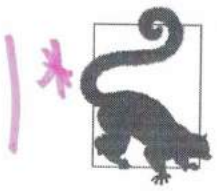
```
library(purrr)

nested_models <- nested_data %>%
  mutate(models = map(data, ~ glm(cbind(count, time_total) ~ time_floor, .,
                                  family = "binomial")))

nested_models
```

```
## # A tibble: 112 × 4
##    person    word              data    models
##     <chr>   <chr>            <list>    <list>
## 1   David #rstats <tibble [12 × 4]> <S3: glm>
## 2   David     bad  <tibble [9 × 4]> <S3: glm>
## 3   David     bit <tibble [10 × 4]> <S3: glm>
## 4   David    blog <tibble [12 × 4]> <S3: glm>
```

```
## 5   David   broom <tibble [10 × 4]> <S3: glm>
## 6   David    call  <tibble [9 × 4]> <S3: glm>
## 7   David   check <tibble [12 × 4]> <S3: glm>
## 8   David    code <tibble [10 × 4]> <S3: glm>
## 9   David    data <tibble [12 × 4]> <S3: glm>
## 10  David     day  <tibble [8 × 4]> <S3: glm>
## # ... with 102 more rows
```



We can think about this modeling procedure answering questions like, "Was a given word mentioned in a given time bin? Yes or no? How does the count of word mentions depend on time?"

Now notice that we have a new column for the modeling results; it is another list column and contains `glm` objects. The next step is to use `map()` and `tidy()` from the broom package to pull out the slopes for each of these models and find the important ones. We are comparing many slopes here, and some of them are not statistically significant, so let's apply an adjustment to the p-values for multiple comparisons.

```
library(broom)

slopes <- nested_models %>%
  unnest(map(models, tidy)) %>%
  filter(term == "time_floor") %>%
  mutate(adjusted.p.value = p.adjust(p.value))
```

Now let's find the most important slopes. Which words have changed in frequency at a moderately significant level in our tweets?

```
top_slopes <- slopes %>%
  filter(adjusted.p.value < 0.1) %>%
  select(-statistic, -p.value)

top_slopes
```

```
## # A tibble: 6 × 8
##   person      word       term      estimate    std.error adjusted.p.value
##    <chr>     <chr>      <chr>         <dbl>        <dbl>            <dbl>
## 1  David   ggplot2 time_floor -8.262540e-08 1.969448e-08     2.996837e-03
## 2  Julia   #rstats time_floor -4.496395e-08 1.119780e-08     6.467858e-03
## 3  Julia      post time_floor -4.818545e-08 1.454440e-08     9.784245e-02
## 4  Julia      read time_floor -9.327168e-08 2.542485e-08     2.634712e-02
## 5  David     stack time_floor  8.041202e-08 2.193375e-08     2.634841e-02
## 6  David #user2016 time_floor -8.175896e-07 1.550152e-07     1.479603e-05
```

To visualize our results, we can plot the use of these words for both David and Julia over this year of tweets (Figure 7-4).

```
words_by_time %>%
  inner_join(top_slopes, by = c("word", "person")) %>%
  filter(person == "David") %>%
```

```
ggplot(aes(time_floor, count/time_total, color = word, lty = word)) +
geom_line(size = 1.3) +
labs(x = NULL, y = "Word frequency")
```

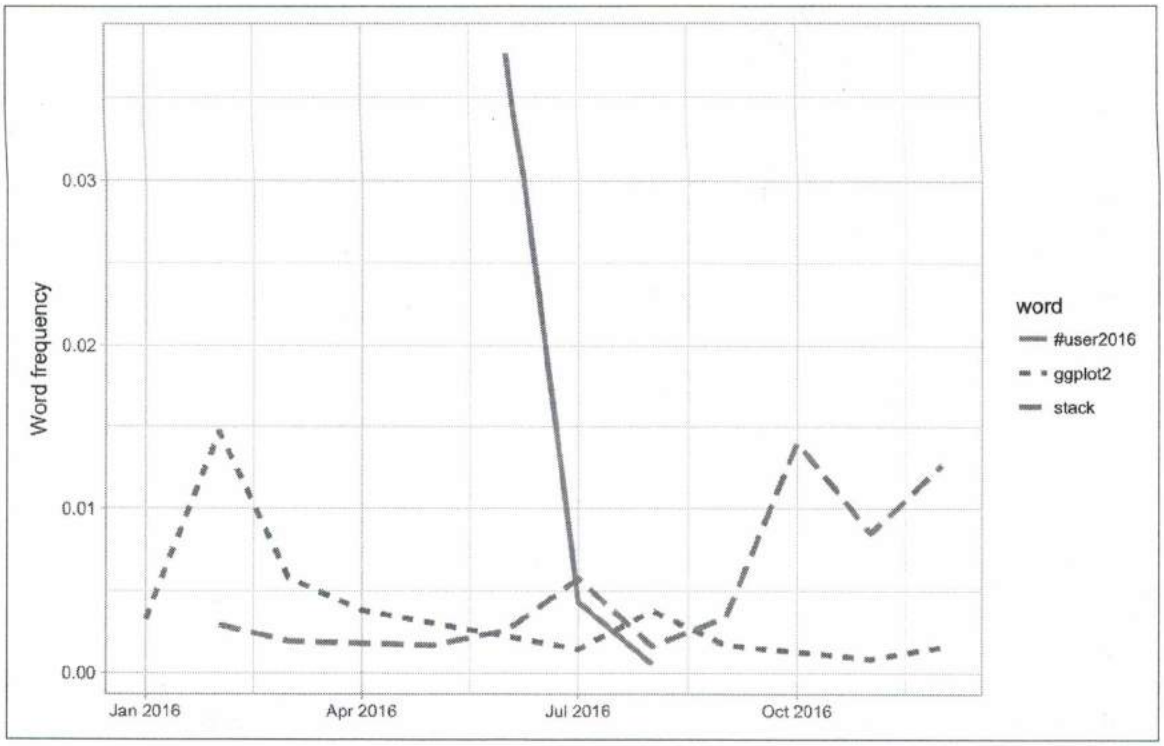


Figure 7-4. Trending words in David's tweets

We see in Figure 7-4 that David tweeted a lot about the UseR conference while he was there and then quickly stopped. He has tweeted more about Stack Overflow toward the end of the year and less about ggplot2 as the year has progressed.

Now let's plot words that have changed frequency in Julia's tweets in Figure 7-5.

```
words_by_time %>%
  inner_join(top_slopes, by = c("word", "person")) %>%
  filter(person == "Julia") %>%
  ggplot(aes(time_floor, count/time_total, color = word, lty = word)) +
  geom_line(size = 1.3) +
  labs(x = NULL, y = "Word frequency")
```

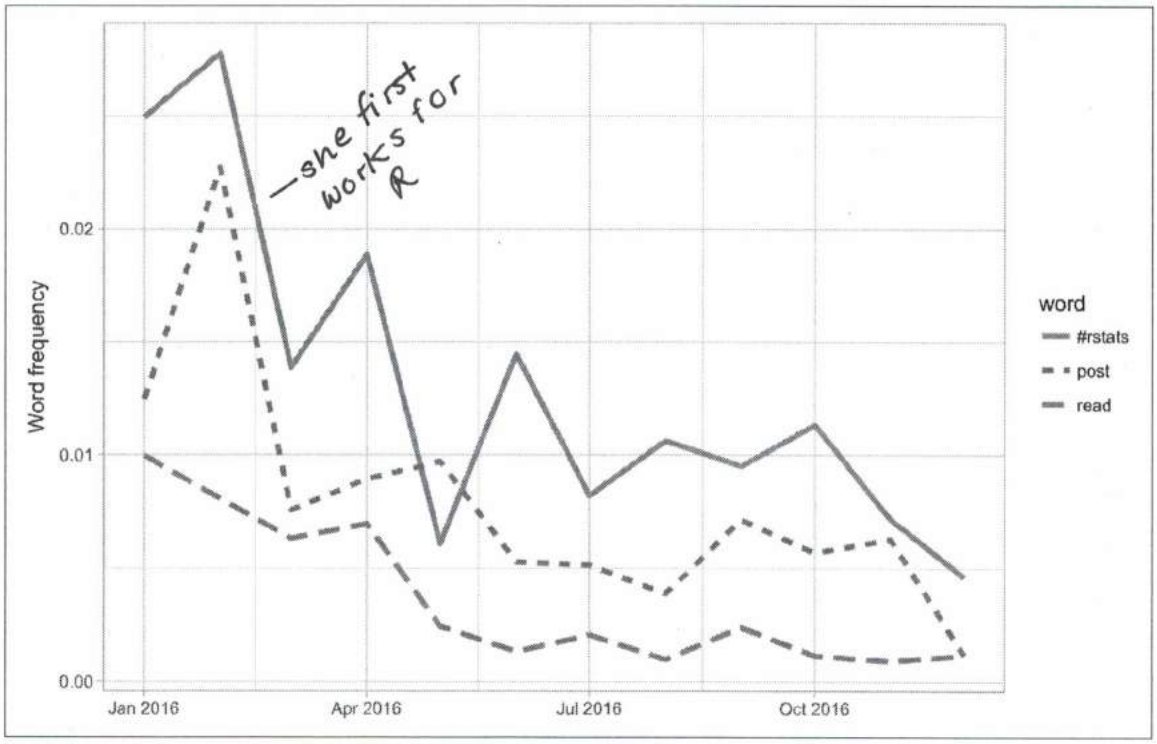


Figure 7-5. Trending words in Julia's tweets

All the significant slopes for Julia are negative. This means she has not tweeted at a higher rate using any specific words, but instead used a variety of different words; her tweets earlier in the year contained the words shown in this plot at higher proportions. Words she uses when publicizing a new blog post, like the #rstats hashtag and "post," have gone down in frequency, and she has tweeted less about reading.

Favorites and Retweets

Another important characteristic of tweets is how many times they are favorited or retweeted. Let's explore which words are more likely to be retweeted or favorited for Julia's and David's tweets. When a user downloads his or her own Twitter archive, favorites and retweets are not included, so we constructed another dataset of the authors' tweets that includes this information. We accessed our own tweets via the Twitter API and downloaded about 3,200 tweets for each person. In both cases, that is about the last 18 months worth of Twitter activity. This corresponds to a period of increasing activity and increasing numbers of followers for both of us.

```
tweets_julia <- read_csv("data/juliasilge_tweets.csv")
tweets_dave <- read_csv("data/drob_tweets.csv")
tweets <- bind_rows(tweets_julia %>%
                      mutate(person = "Julia"),
                    tweets_dave %>%
                      mutate(person = "David")) %>%
  mutate(created_at = ymd_hms(created_at))
```

Now that we have this second, smaller set of only recent tweets, let's use unnest_tokens() to transform these tweets to a tidy data set. Let's remove all retweets and replies from this data set so we only look at regular tweets that David and Julia have posted directly.

```
tidy_tweet %>%
  select(-source)
  filter(!str_detect(text, "^(RT|@)")) %>%
  mutate(text = str_replace_all(text, replace_reg, "")) %>%
  unnest_tokens(word, text, token = "regex", pattern = unnest_reg) %>%
  anti_join(stop_words)

tidy_tweets
```

```
## # A tibble: 11,078 × 7
##              id          created_at retweets favorites person       word
##           <dbl>              <dttm>    <int>     <int>  <chr>      <chr>
## 1  8.044026e+17 2016-12-01 19:11:43        1        15  David      worry
## 2  8.043967e+17 2016-12-01 18:48:07        4         6  David         j's
## 3  8.043611e+17 2016-12-01 16:26:39        8        12  David  bangalore
## 4  8.043611e+17 2016-12-01 16:26:39        8        12  David     london
## 5  8.043611e+17 2016-12-01 16:26:39        8        12  David developers
## 6  8.041571e+17 2016-12-01 02:56:10        0        11  Julia management
## 7  8.041571e+17 2016-12-01 02:56:10        0        11  Julia      julie
## 8  8.040582e+17 2016-11-30 20:23:14       30        41  David         sf
## 9  8.040324e+17 2016-11-30 18:40:27        0        17  Julia     zipped
## 10 8.040324e+17 2016-11-30 18:40:27        0        17  Julia         gb
## # ... with 11,068 more rows
```

To start with, let's look at the number of times each of our tweets was retweeted. Let's find the total number of retweets for each person.

```
totals <- tidy_tweets %>%
  group_by(person, id) %>%
  summarise(rts = sum(retweets)) %>%
  group_by(person) %>%
  summarise(total_rts = sum(rts))

totals
```

```
## # A tibble: 2 × 2
##   person total_rts
##    <chr>     <int>
## 1  David    110171
## 2  Julia     12701
```

Now let's find the median number of retweets for each word and person. We probably want to count each tweet/word combination only once, so we will use group_by() and summarise() twice, one right after the other. The first summarise() statement counts how many times each word was retweeted, for each tweet and person. In the second summarise() statement, we can find the median retweets for each person and word, count the number of times each word was used by each person, and keep that

in uses. Next, we can join this to the data frame of retweet totals. Let's `filter()` to only keep words mentioned at least five times.

```
word_by_rts <- tidy_tweets %>%
  group_by(id, word, person) %>%
  summarise(rts = first(retweets)) %>%
  group_by(person, word) %>%
  summarise(retweets = median(rts), uses = n()) %>%
  left_join(totals) %>%
  filter(retweets != 0) %>%
  ungroup()

word_by_rts %>%
  filter(uses >= 5) %>%
  arrange(desc(retweets))
```

```
## # A tibble: 178 × 5
##    person          word retweets  uses total_rts
##     <chr>         <chr>    <dbl> <int>     <int>
## 1   David     animation     85.0     5    110171
## 2   David      download     52.0     5    110171
## 3   David         start     51.0     7    110171
## 4   Julia      tidytext     50.0     7     12701
## 5   David     gganimate     45.0     8    110171
## 6   David   introducing     45.0     6    110171
## 7   David understanding     37.0     6    110171
## 8   David             0     35.0     7    110171
## 9   David         error     34.5     8    110171
## 10  David      bayesian     34.0     7    110171
## # ... with 168 more rows
```

At the top of this sorted data frame, we see tweets from Julia and David about packages that they work on, like gutenbergr, gganimate, and tidytext. Let's plot the words that have the highest median retweets for each of our accounts (Figure 7-6).

```
word_by_rts %>%
  filter(uses >= 5) %>%
  group_by(person) %>%
  top_n(10, retweets) %>%
  arrange(retweets) %>%
  ungroup() %>%
  mutate(word = factor(word, unique(word))) %>%
  ungroup() %>%
  ggplot(aes(word, retweets, fill = person)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~ person, scales = "free", ncol = 2) +
  coord_flip() +
  labs(x = NULL,
       y = "Median # of retweets for tweets containing each word")
```

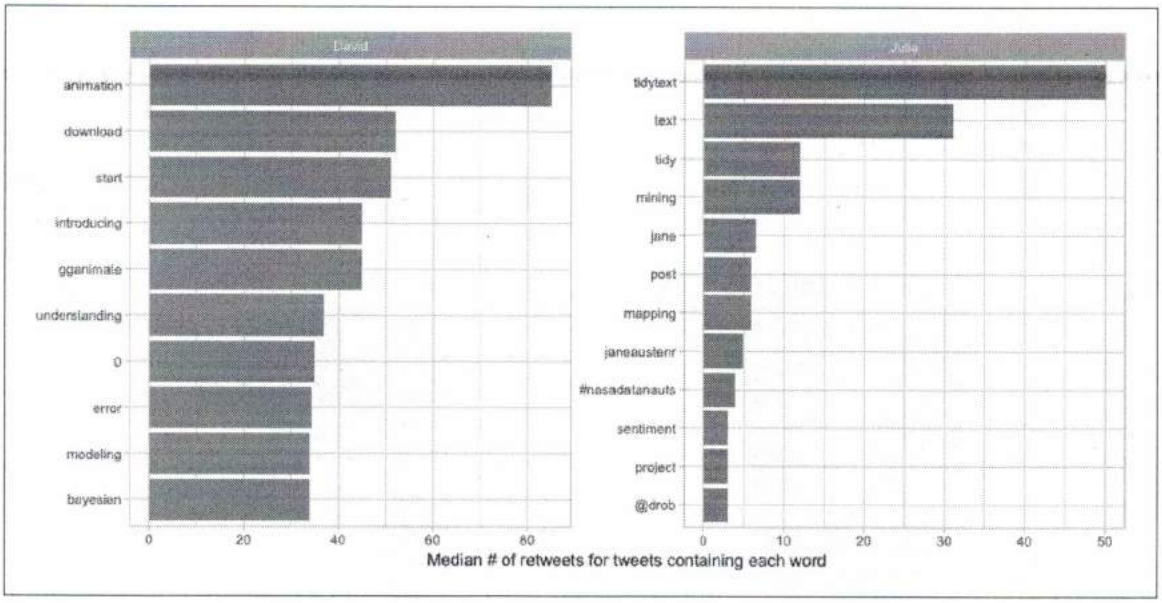


Figure 7-6. Words with highest median retweets

We see lots of words about R packages, including tidytext, a package about which you are reading right now! The "0" for David comes from tweets where he mentions version numbers of packages, like "broom 0.4.0" or similar.

We can follow a similar procedure to see which words led to more favorites. Are they different than the words that lead to more retweets?

```
totals <- tidy_tweets %>%
  group_by(person, id) %>%
  summarise(favs = sum(favorites)) %>%
  group_by(person) %>%
  summarise(total_favs = sum(favs))

word_by_favs <- tidy_tweets %>%
  group_by(id, word, person) %>%
  summarise(favs = first(favorites)) %>%
  group_by(person, word) %>%
  summarise(favorites = median(favs), uses = n()) %>%
  left_join(totals) %>%
  filter(favorites != 0) %>%
  ungroup()
```

We have built the data frames we need. Now let's make our visualization in Figure 7-7.

```
word_by_favs %>%
  filter(uses >= 5) %>%
  group_by(person) %>%
  top_n(10, favorites) %>%
  arrange(favorites) %>%
  ungroup() %>%
  mutate(word = factor(word, unique(word))) %>%
```

```
ungroup() %>%
ggplot(aes(word, favorites, fill = person)) +
geom_col(show.legend = FALSE) +
facet_wrap(~ person, scales = "free", ncol = 2) +
coord_flip() +
labs(x = NULL,
     y = "Median # of favorites for tweets containing each word")
```

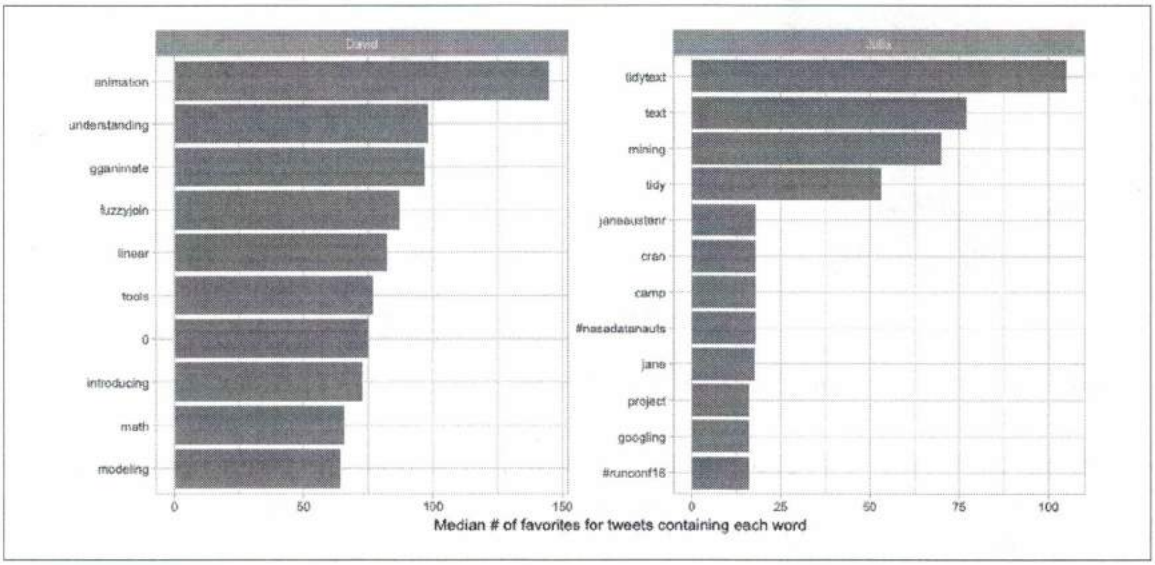


Figure 7-7. Words with highest median favorites

We see some minor differences between Figures 7-6 and 7-7, especially near the bottom of the top 10 list, but these are largely the same words as for retweets. In general, the same words that lead to retweets lead to favorites. A prominent word for Julia in both plots is the hashtag for the NASA Datanauts program that she has participated in; read on to Chapter 8 to learn more about NASA data and what we can learn from text analysis of NASA datasets.

Summary

This chapter was our first case study, a beginning-to-end analysis that demonstrated how to bring together the concepts and code we have been exploring in a cohesive way to understand a text data set. Comparing word frequencies allowed us to see which words we tweeted more and less frequently, and the log odds ratio showed which words are more likely to be tweeted from each of our accounts. We can use `nest()` and `map()` with the `glm()` function to find which words we have tweeted at higher and lower rates as time has passed. Finally, we can find which words in our tweets led to higher numbers of retweets and favorites. All of these are examples of approaches to measure how we use words in similar and different ways, and how the characteristics of our tweets are changing or compare with each other. These are flexible approaches to text mining that can be applied to other types of text as well.