1 Introduction

If you work in analytics or data science, like we do, you are familiar with the fact that data is being generated all the time at ever faster rates. (You may even be a little weary of people pontificating about this fact.) Analysts are often trained to handle tabular or rectangular data that is mostly numeric, but much of the data proliferating today is unstructured and typically text-heavy. Many of us who work in analytic fields are not trained in even simple interpretation of natural language.

We developed a new R package, tidytext (Silge and Robinson 2016), because we were familiar with many methods for data wrangling and visualization, but couldn't easily apply these same methods to text. We found that using tidy data principles can make many text mining tasks easier, more effective, and consistent with tools already in wide use. By treating text as data frames of words, we can manipulate, summarize, and visualize the characteristics of text easily and integrate natural language processing into effective workflows we were already using.

The tools provided by the tidytext package are relatively simple; what is important is the possible applications. Thus, this book provides compelling examples of real text mining problems.

1.1 What is tidy text?

As described by Hadley Wickham (Wickham 2014), tidy data has a specific structure:

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

We thus define the tidy text format as being **a table with one-term-per-row.** This is in contrast to the ways text is often stored in current analyses, as raw strings or perhaps a specialized format like a document-term matrix. For tidy text mining, the **term** that is stored in each row can be, mostly commonly, a single word, or perhaps an n-gram, a sentence, or another unit of text of interest for a certain analysis. In the tidytext package, we provide functionality to tokenize by commonly used units of text like these and convert to a one-term-per-row format.

Tidy data sets allow manipulation with a standard set of "tidy" tools, including popular packages such as dplyr (Wickham and Francois 2016), tidyr (Wickham 2016), ggplot2 (Wickham 2009), and broom (Robinson et al. 2015). By keeping the input and output in tidy tables, users can transition fluidly between these tools. We've found these tidy tools extend naturally to many text analyses and explorations.

At the same time, the tidytext package doesn't expect a user to keep text data in a tidy form at all times during an analysis. The package includes functions to `tidy()` objects (see the broom package (Robinson et al. 2015)) from popular text mining R packages such as tm (Ingo Feinerer and Meyer 2008) and quanteda (Benoit and Nulty 2016). This allows, for example, a workflow with easy 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. The models can then be re-converted into a tidy form for interpretation and visualization with ggplot2.

1.2 About this book

This book is focused on practical software examples and data explorations. There are few equations, but a great deal of code. We especially focus on generating real insights from the literature, news, and social media that we analyze.

We don't assume any previous knowledge of text mining; professional linguists and text analysts will likely find our examples elementary, though we are confident they can build on the framework for their own analyses.

We do assume that the reader is at least slightly familiar with dplyr, ggplot2, and the `%>%` "pipe" operator in R, and is interested in applying these tools to text data. For users who don't have this background, we recommend books such as R for Data Science. We believe that with a basic background and interest in tidy data, even a user early in their R career can understand and apply our examples.

1.3 Outline

We start by introducing the tidy text format, and some of the ways dplyr, tidyr, and tidytext allow informative analyses of this structure.

- **Chapter 2** outlines the tidy text format and the `unnest_tokens()` function. It also introduces the gutenbergr and janeaustenr packages, which provide useful literary text datasets that we'll use throughout this book.
- **Chapter 3** shows how to perform sentiment analysis on a tidy text dataset, using the `sentiments` dataset from tidytext and `inner_join()` from dplyr.
- **Chapter 4** describes the tf-idf statistic (term frequency times inverse document frequency), a quantity used for identifying terms that are especially important to a particular document.
- **Chapter 5** introduces n-grams and how to analyze word networks in text using the widyr package.

Text won't be tidy at all stages of an analysis, and it is important to be able to convert back and forth from a tidy format.

- **Chapter 6** introduces methods for tidying document-term matrices and corpus objects from the tm and quanteda packages, as well as for casting tidy text datasets into those formats.
- **Chapter 7** explores the concept of topic modeling, and uses the `tidy()` method for interpreting and visualizing the output of the topicmodels package.

We conclude with several tidy text analyses that bring together multiple text mining approaches we've learned.

- **Chapter 8** demonstrates an application of a tidy text analysis by analyzing the authors' own Twitter archives. How do Dave's and Julia's tweeting habits compare?
- **Chapter 9** explores metadata from over 32,000 NASA datasets by looking at how keywords from the datasets are connected to title and description fields.
- **Chapter 10** analyzes a dataset of Usenet messages from a diverse set of newsgroups (focused on topics like politics, hockey, technology, atheism, and more) to understand patterns across the groups.

1.4 Topics this book does not cover

This book serves as an introduction to a framework along with a collection of examples, but it is far from a complete exploration of natural language processing. The CRAN Task View on Natural Language Processing provides details on more ways to use R for computational linguistics. There are several areas that you may want to explore in more detail according to your needs.

- **Clustering, classification, and prediction:** Machine learning on text is a vast topic that could easily fill its own volume. We introduce one method of unsupervised clustering (topic modeling) in Chapter 7 but many more machine learning algorithms can be used in dealing with text.
- **Word embedding:** Words can be mapped to vectors, and such vector representations of words can generate linguistic relationships about meanings of words or how words are used. Such representations of words are not tidy in the sense that we consider here, but can boost performance of machine learning algorithms.
- **More complex tokenization:** We hand tokenization off to the tokenizers package (Mullen 2016), which itself wraps a variety of tokenizers with a consistent interface, but many others exist for specific applications.
- **Languages other than English:** Some of our users have had success applying tidytext to their text mining needs for languages other than English but we are not covering those issues in this book.

We feel that the tidy data philosphy is a powerful tool to make handling data easier and more effective, and this is no less true when it comes to handling text. We also believe that the tools provided by tidy data principles and specifically tidy text mining are well suited to extensions beyond the examples we provide here.

1.5 Acknowledgements

We are so thankful for the contributions, help, and perspective of people who have moved us forward in this project. There are several people and organizations we would like to thank in particular. We would like to thank Gabriela de Queiroz for her contributions to the package while we were at the unconference where we began work on the tidytext package, Lincoln Mullen for his work on the tokenizers package, Kenneth Benoit for his work on the quanteda package, and Hadley Wickham for his work in framing tidy data principles and building tidy tools. We would also like to thank rOpenSci, which hosted us at the unconference where we began work, and the NASA Datanauts program, for the opportunities and support they have provided Julia during her time with them.

References

Silge, Julia, and David Robinson. 2016. “Tidytext: Text Mining and Analysis Using Tidy Data Principles in R.” *JOSS* 1 (3). The Open Journal. doi:10.21105/joss.00037.

Wickham, Hadley. 2014. “Tidy Data.” *Journal of Statistical Software* 59 (1): 1–23. doi:10.18637/jss.v059.i10.

Wickham, Hadley, and Romain Francois. 2016. *Dplyr: A Grammar of Data Manipulation*. https://CRAN.R-project.org/package=dplyr.

Wickham, Hadley. 2016. *Tidyr: Easily Tidy Data with `spread()` and `gather()` Functions*. https://CRAN.R-project.org/package=tidyr.

Wickham, Hadley. 2009. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. http://ggplot2.org.

Robinson, David, Matthieu Gomez, Boris Demeshev, Dieter Menne, Benjamin Nutter, Luke Johnston, Ben Bolker, Francois Briatte, and Hadley Wickham. 2015. *Broom: Convert Statistical Analysis Objects into Tidy Data Frames*. https://CRAN.R-project.org/package=broom.

Ingo Feinerer, Kurt Hornik, and David Meyer. 2008. “Text Mining Infrastructure in R.” *Journal of Statistical Software* 25 (5): 1–54. http://www.jstatsoft.org/v25/i05/.

Benoit, Kenneth, and Paul Nulty. 2016. *Quanteda: Quantitative Analysis of Textual Data*. https://CRAN.R-project.org/package=quanteda.

Mullen, Lincoln. 2016. *Tokenizers: A Consistent Interface to Tokenize Natural Language Text*. https://CRAN.R-project.org/package=tokenizers.

2 The tidy text format

We define the tidy text format as being a table with **one-term-per-row.** Structuring text data in this way means that it conforms to tidy data principles and can be manipulated with a set of consistent tools. This is worth contrasting with the ways text is often stored in text mining approaches.

- **Raw string**: Text can, of course, be stored as raw strings within R, and often text data is first read into memory in this form.
- **Corpus**: These types of objects typically annotate the raw string content with additional metadata and details.
- **Document-term matrix**: This is a sparse matrix describing a collection (i.e., a corpus) of documents with one row for each document and one column for each term. The value in the matrix is typically word count or tf-idf (see Chapter 4).

Let's hold off on exploring structures like a document-term matrix until Chapter 6, and get down to the basics of converting text to a tidy format.

2.1 The `unnest_tokens` function

Emily Dickinson wrote some lovely text in her time.

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

text
```

```
## [1] "Because I could not stop for Death -"   "He kindly stopped for me -"
## [3] "The Carriage held but just Ourselves -" "and Immortality"
```

This is a typical character vector that we might want to analyze. In order to turn it into a tidy text dataset, we first need to put it into a data frame.

```
library(dplyr)
text_df <- data_frame(line = 1:4, text = text)

text_df
```

```
## # A tibble: 4 × 2
##    line                                  text
##   <int>                                 <chr>
## 1     1   Because I could not stop for Death -
## 2     2             He kindly stopped for me -
## 3     3 The Carriage held but just Ourselves -
## 4     4                        and Immortality
```

Notice that this data frame isn't yet compatible with tidy tools. We can't filter out words or count which occur most frequently, since each row is made up of multiple combined words. We need to convert this so that it has **one-token-per-document-per-row**. A token, in this context, is a meaningful unit of text that we are interested in using for further analysis. Tokenization is the process of breaking up text into individual tokens, and it is most commonly done at the level of single words. Within our tidy text framework, we will both break the text into individual tokens *and* transform it to a tidy data structure.

To do this, we use tidytext's `unnest_tokens()` function.

```
library(tidytext)

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

```
## # A tibble: 20 × 2
##     line    word
##    <int>   <chr>
## 1      1 because
## 2      1       i
## 3      1   could
## 4      1     not
## 5      1    stop
## 6      1     for
## 7      1   death
## 8      2      he
## 9      2  kindly
## 10     2 stopped
## # ... with 10 more rows
```

We've now split each row so that there is one token (word) in each row of the new data frame; the default tokenization in `unnest_tokens()` is for single words, as shown here. Also notice:

- Other columns, such as the line number each word came from, are retained.
- Punctuation has been stripped.
- By default, `unnest_tokens()` converts the tokens to lowercase, which makes them easier to compare or combine with other datasets. (Use the `to_lower = FALSE` argument to turn off this behavior).

Having the text data in this format lets us manipulate, process, and visualize the text using the standard set of tidy tools, namely dplyr, tidyr, ggplot2, and broom.

2.2 Tidying the works of Jane Austen

Let's use the text of Jane Austen's 6 completed, published novels from the janeaustenr package (Silge 2016), and transform them into a tidy format. The janeaustenr package provides these texts in a one-row-per-line format. Let's start with that, annotate a `linenumber` quantity to keep track of lines in the original format, and use a regex to find where all the chapters are.

```
library(janeaustenr)
library(dplyr)
library(stringr)

original_books <- austen_books() %>%
  group_by(book) %>%
  mutate(linenumber = row_number(),
         chapter = cumsum(str_detect(text, regex("^chapter [\\divxlc]",
                                                 ignore_case = TRUE)))) %>%
  ungroup()

original_books
```

```
## # A tibble: 73,422 × 4
##                      text                book linenumber chapter
##                     <chr>              <fctr>      <int>   <int>
## 1   SENSE AND SENSIBILITY Sense & Sensibility          1       0
## 2                         Sense & Sensibility          2       0
## 3          by Jane Austen Sense & Sensibility          3       0
## 4                         Sense & Sensibility          4       0
## 5                  (1811) Sense & Sensibility          5       0
## 6                         Sense & Sensibility          6       0
## 7                         Sense & Sensibility          7       0
## 8                         Sense & Sensibility          8       0
## 9                         Sense & Sensibility          9       0
## 10              CHAPTER 1 Sense & Sensibility         10       1
## # ... with 73,412 more rows
```

To work with this as a tidy dataset, we need to restructure it in the **one-token-per-row** format. The `unnest_tokens()` function is a way to convert a data frame with a text column to be one-token-per-row.

```
library(tidytext)
tidy_books <- original_books %>%
  unnest_tokens(word, text)

tidy_books
```

```
## # A tibble: 725,054 × 4
##                   book linenumber chapter        word
##                 <fctr>      <int>   <int>       <chr>
## 1  Sense & Sensibility          1       0       sense
## 2  Sense & Sensibility          1       0         and
## 3  Sense & Sensibility          1       0 sensibility
## 4  Sense & Sensibility          3       0          by
## 5  Sense & Sensibility          3       0        jane
## 6  Sense & Sensibility          3       0      austen
## 7  Sense & Sensibility          5       0        1811
## 8  Sense & Sensibility         10       1     chapter
## 9  Sense & Sensibility         10       1           1
## 10 Sense & Sensibility         13       1         the
## # ... with 725,044 more rows
```

This function uses the tokenizers package (Mullen 2016) to separate each line of text in the original data frame into tokens. The default tokenizing is for words, but other options include characters, n-grams, sentences, lines, paragraphs, or separation around a regex pattern.

Now that the data is in one-word-per-row format, we can manipulate it with tidy tools like dplyr. We can remove stop words (kept in the tidytext dataset `stop_words`) with an `anti_join()`.

```
data(stop_words)

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

We can also use `count()` to find the most common words in all the books as a whole.

```
tidy_books %>%
  count(word, sort = TRUE)
```

```
## # A tibble: 13,914 × 2
##      word     n
##     <chr> <int>
## 1    miss  1855
## 2    time  1337
## 3   fanny   862
## 4    dear   822
## 5    lady   817
## 6     sir   806
## 7     day   797
## 8    emma   787
## 9  sister   727
## 10  house   699
## # ... with 13,904 more rows
```

For example, this allows us to visualize the commonly used words using ggplot2.

```
library(ggplot2)

tidy_books %>%
  count(word, sort = TRUE) %>%
  filter(n > 600) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n)) +
  geom_bar(stat = "identity") +
  xlab(NULL) +
  coord_flip()
```

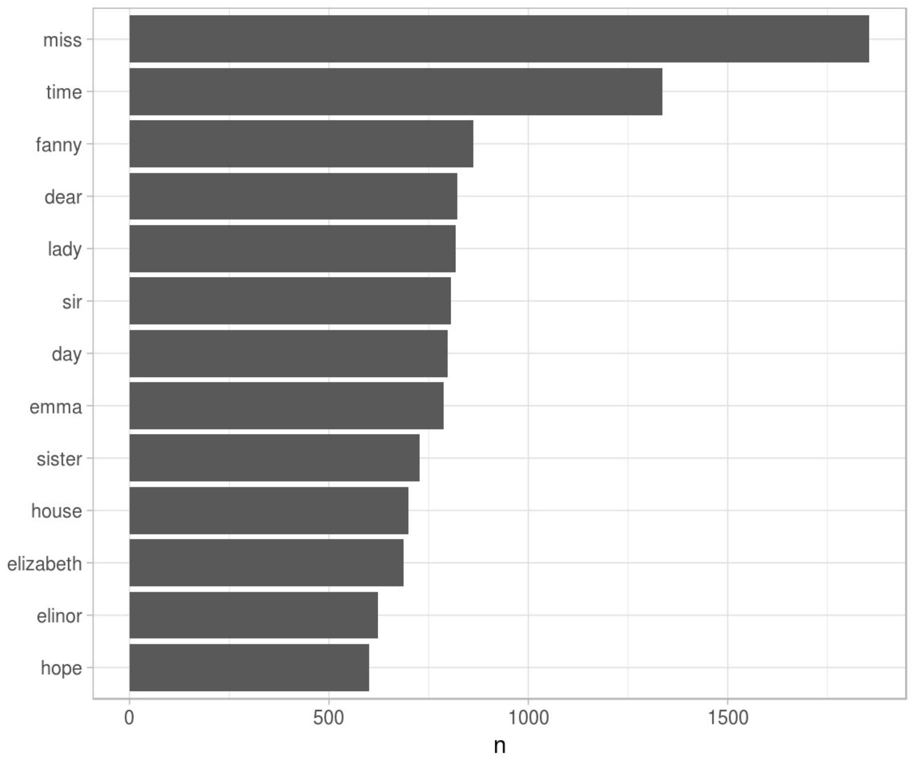


Figure 2.1: The most common words in Jane Austen's novels

We could pipe this straight into ggplot2 because of our consistent use of tidy tools.

2.3 The gutenbergr package

Now that we've used the janeaustenr package, let's introduce the gutenbergr package (Robinson 2016). The gutenbergr package provides access to the public domain works from the Project Gutenberg collection. The package includes tools both for downloading books (stripping out the unhelpful header/footer information), and a complete dataset of Project Gutenberg metadata that can be used to find works of interest. In this book, we will mostly use the function `gutenberg_download()` that downloads one or more works from Project Gutenberg by ID, but you can also use other functions to explore metadata, pair Gutenberg ID with title, author, language, etc., or gather information about authors. To learn more about gutenbergr, check out the package's tutorial at rOpenSci, where it is one of rOpenSci's packages for data access.

2.4 Word frequencies

A common task in text mining is to look at word frequencies, just like we have done above for Jane Austen's novels, and to compare frequencies across different texts. We can do this intuitively and smoothly using tidy data principles. We already have Jane Austen's works; let's get two more sets of texts to compare to. First, let's look at some science fiction and fantasy novels by H.G. Wells, who lived in the late 19th and early 20th centuries. Let's get *The Time Machine*, *The War of the Worlds*, *The Invisible Man*, and *The Island of Doctor Moreau*.

```
library(gutenbergr)

hgwells <- gutenberg_download(c(35, 36, 5230, 159))
```

```
tidy_hgwells <- hgwells %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words)
```

Just for kicks, what are the most common words in these novels of H.G. Wells?

```
tidy_hgwells %>%
  count(word, sort = TRUE)
```

```
## # A tibble: 11,769 × 2
##      word     n
##     <chr> <int>
## 1    time   454
## 2  people   302
## 3    door   260
## 4   heard   249
## 5   black   232
## 6   stood   229
## 7   white   222
## 8    hand   218
## 9    kemp   213
## 10   eyes   210
## # ... with 11,759 more rows
```

Now let's get some well-known works of the Brontë sisters, whose lives overlapped with Jane Austen's somewhat but who wrote in a rather different style. Let's get *Jane Eyre*, *Wuthering Heights*, *The Tenant of Wildfell Hall*, *Villette*, and *Agnes Grey*.

```
bronte <- gutenberg_download(c(1260, 768, 969, 9182, 766))
```

```
tidy_bronte <- bronte %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words)
```

What are the most common words in these novels of the Brontë sisters?

```
tidy_bronte %>%
  count(word, sort = TRUE)
```

```
## # A tibble: 25,714 × 2
##      word     n
##     <chr> <int>
## 1    time  1586
## 2    miss  1388
## 3    hand  1239
## 4     day  1136
## 5    eyes  1023
## 6   night  1011
## 7   house   960
## 8    head   957
## 9  looked   949
## 10   aunt   896
## # ... with 25,704 more rows
```

Interesting that “time”, “eyes”, and “hand” are in the top 10 for both H.G. Wells and the Brontë sisters.

Now, let’s calculate the frequency for each word for the works of Jane Austen, the Brontë sisters, and H.G. Wells.

```
tidy_both <- bind_rows(
  mutate(tidy_bronte, author = "Brontë Sisters"),
  mutate(tidy_hgwells, author = "H.G. Wells"))

austen_percent <- tidy_books %>%
  mutate(word = str_extract(word, "[a-z]+")) %>%
  count(word) %>%
  transmute(word, austen = n / sum(n))

frequency <- tidy_both %>%
  mutate(word = str_extract(word, "[a-z]+")) %>%
  count(author, word) %>%
  mutate(other = n / sum(n)) %>%
  left_join(austen_percent) %>%
  ungroup()
```

We use `str_extract()` here because the UTF-8 encoded texts from Project Gutenberg have some examples of words with underscores around them to indicate emphasis (like italics). The tokenizer treated these as words but we don't want to count "_any_" separately from "any". Now let's plot.

```r
library(scales)

ggplot(frequency, aes(x = other, y = austen, color = abs(austen - other))) +
  geom_abline(color = "gray40", lty = 2) +
  geom_jitter(alpha = 0.1, size = 2.5, width = 0.3, height = 0.3) +
  geom_text(aes(label = word), check_overlap = TRUE, vjust = 1.5) +
  scale_x_log10(labels = percent_format()) +
  scale_y_log10(labels = percent_format()) +
  scale_color_gradient(limits = c(0, 0.001), low = "darkslategray4", high = "gray75") +
  facet_wrap(~author, ncol = 2) +
  theme(legend.position="none") +
  labs(y = "Jane Austen", x = NULL)
```

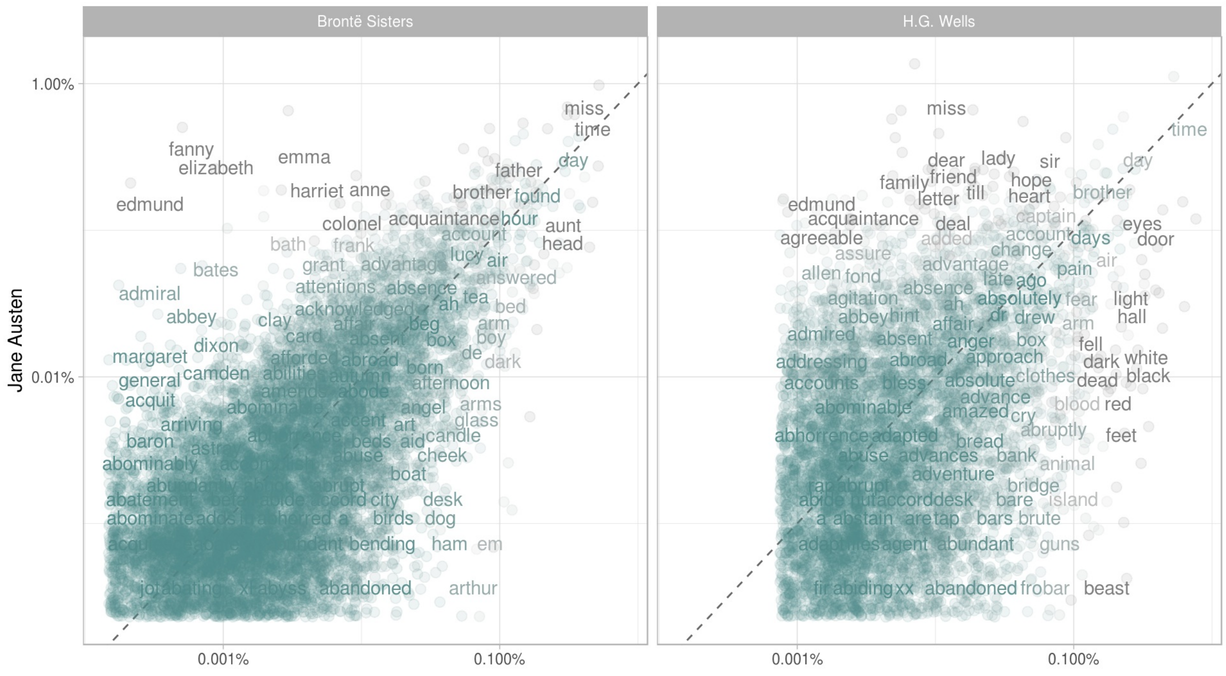


Figure 2.2: Comparing the word frequencies of Jane Austen, the Brontë sisters, and H.G. Wells

Words that are close to the line in these plots have similar frequencies in both sets of texts, for example, in both Austen and Brontë texts ("miss", "time", "day" at the upper frequency end) or in both Austen and Wells texts ("time", "day", "brother" at the high frequency end). Words that are far from the line are words that are found more in one set of texts than another. For example, in the Austen-Brontë panel, words like "elizabeth", "emma", and "edmund" (all proper nouns) are found in Austen's texts but not much in the Brontë texts, while words like "arthur", "dog", and "ham" are found in the Brontë texts but not the Austen texts. In comparing H.G. Wells with Jane Austen, Wells uses words like "beast", "island", "feet", and "black" that Austen does not, while Austen uses words like "family", "friend", "letter", and "dear" that Wells does not.

Overall, notice in Figure 2.2 that the words in the Austen-Brontë panel are closer to the zero-slope line than in the Austen-Wells panel and also extend to lower frequencies; Austen and the Brontë sisters use more similar words than Austen and H.G. Wells. Also, we notice that not all the words are found in all three sets of texts and there are fewer points in the plot for Austen and H.G. Wells.

Let's quantify how similar and different these sets of word frequencies are using a correlation test. How correlated are the word frequencies between Austen and the Brontë sisters, and between Austen and Wells?

```
cor.test(data = frequency[frequency$author == "Brontë Sisters",],
         ~ other + austen)
```

```
##
##  Pearson's product-moment correlation
##
## data:  other and austen
## t = 119.43, df = 10765, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.7466616 0.7629140
## sample estimates:
##       cor
## 0.7549037
```

```
cor.test(data = frequency[frequency$author == "H.G. Wells",],
         ~ other + austen)
```

```
##
##  Pearson's product-moment correlation
##
## data:  other and austen
## t = 35.91, df = 6027, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.3988024 0.4403950
## sample estimates:
##       cor
## 0.4198191
```

The relationship between the word frequencies is different between these sets of texts, as it appears in the plots.

References

Silge, Julia. 2016. *Janeaustenr: Jane Austen’s Complete Novels*. https://CRAN.R-project.org/package=janeaustenr.

Mullen, Lincoln. 2016. *Tokenizers: A Consistent Interface to Tokenize Natural Language Text*. https://CRAN.R-project.org/package=tokenizers.

Robinson, David. 2016. *Gutenbergr: Download and Process Public Domain Works from Project Gutenberg*. https://cran.rstudio.com/package=gutenbergr.

3 Sentiment analysis with tidy data

In the previous chapter, we explored in depth what we mean by the tidy text format and showed how this format can be used to approach questions about word frequency. This allowed us to analyze which words are used most frequently in documents and to compare documents, but now let’s investigate a different topic. Let’s address the topic of opinion mining or sentiment analysis. When human readers approach a text, we use our understanding of the emotional intent of words to infer whether a section of text is positive or negative, or perhaps characterized by some other more nuanced emotion like surprise or disgust. We can use the tools of text mining to approach the emotional content of text programmatically.

One way to analyze the sentiment of a text is to consider the text as a combination of its individual words and the sentiment content of the whole text as the sum of the sentiment content of the individual words. This isn’t the only way to approach sentiment analysis, but it is an often-used approach, *and* an approach that naturally takes advantage of the tidy tool ecosystem.

3.1 The `sentiments` dataset

As discussed above, there are a variety of methods and dictionaries that exist for evaluating the opinion or emotion in text. The tidytext package contains several sentiment lexicons in the `sentiments` dataset.

```
library(tidytext)

sentiments
```

```
## # A tibble: 23,165 × 4
##            word sentiment lexicon score
##           <chr>     <chr>   <chr> <int>
## 1        abacus     trust     nrc    NA
## 2       abandon      fear     nrc    NA
## 3       abandon  negative     nrc    NA
## 4       abandon   sadness     nrc    NA
## 5     abandoned     anger     nrc    NA
## 6     abandoned      fear     nrc    NA
## 7     abandoned  negative     nrc    NA
## 8     abandoned   sadness     nrc    NA
## 9   abandonment     anger     nrc    NA
## 10 abandonment      fear     nrc    NA
## # ... with 23,155 more rows
```

The three general-purpose lexicons are

- `AFINN` from Finn Årup Nielsen,
- `bing` from Bing Liu and collaborators, and
- `nrc` from Saif Mohammad and Peter Turney.

All three of these lexicons are based on unigrams, i.e., single words. These lexicons contain many English words and the words are assigned scores for positive/negative sentiment, and also possibly emotions like joy, anger, sadness, and so forth. The `nrc` lexicon categorizes words in a binary fashion ("yes"/"no") into categories of positive, negative, anger, anticipation, disgust, fear, joy, sadness, surprise, and trust. The `bing` lexicon categorizes words in a binary fashion into positive and negative categories. The `AFINN` lexicon assigns words with a score that runs between -5 and 5, with negative scores indicating negative sentiment and positive scores indicating positive sentiment. All of this information is tabulated in the `sentiments` dataset, and tidytext provides a function `get_sentiments()` to get specific sentiment lexicons without the columns that are not used in that lexicon.

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

```
## # A tibble: 2,476 × 2
##           word score
##          <chr> <int>
## 1      abandon    -2
## 2    abandoned    -2
## 3     abandons    -2
## 4     abducted    -2
## 5    abduction    -2
## 6   abductions    -2
## 7        abhor    -3
## 8     abhorred    -3
## 9    abhorrent    -3
## 10      abhors    -3
## # ... with 2,466 more rows
```

```
get_sentiments("bing")
```

```
## # A tibble: 6,788 × 2
##            word sentiment
##           <chr>     <chr>
## 1       2-faced  negative
## 2       2-faces  negative
## 3            a+  positive
## 4      abnormal  negative
## 5       abolish  negative
## 6    abominable  negative
## 7    abominably  negative
## 8     abominate  negative
## 9   abomination  negative
## 10        abort  negative
## # ... with 6,778 more rows
```

```
get_sentiments("nrc")
```

```
## # A tibble: 13,901 × 2
##           word sentiment
##          <chr>     <chr>
## 1       abacus     trust
## 2      abandon      fear
## 3      abandon  negative
## 4      abandon   sadness
## 5    abandoned     anger
## 6    abandoned      fear
## 7    abandoned  negative
## 8    abandoned   sadness
## 9  abandonment     anger
## 10 abandonment      fear
## # ... with 13,891 more rows
```

How were these sentiment lexicons put together and validated? They were constructed via either crowdsourcing (using, for example, Amazon Mechanical Turk) or by the labor of one of the authors, and were validated using some combination of crowdsourcing again, restaurant or movie reviews, or Twitter data. Given this information, we may hesitate to apply these sentiment lexicons to styles of text dramatically different from what they were validated on, such as narrative fiction from 200 years ago. While it is true that using these sentiment lexicons with, for example, Jane Austen's novels may give us less accurate results than with tweets sent by a contemporary writer, we still can measure the sentiment content for words that are shared across the lexicon and the text.

There are also some domain-specific sentiment lexicons available, constructed to be used with text from a specific content area. Chapter 6 explores an analysis using a sentiment lexicon specifically for finance.

Dictionary-based methods like the ones we are discussing find the total sentiment of a piece of text by adding up the individual sentiment scores for each word in the text. Not every English word is in the lexicons because many English words are pretty neutral. It is important to keep in mind that these methods do not take into account qualifiers before a word, such as in "no good" or "not true"; a lexicon-based method like this is based on unigrams only. For many kinds of text (like the narrative examples below), there are not sustained sections of sarcasm or negated text, so this is not an important effect. Also, we can use a tidy text approach to begin to understand what kinds of negation words are important in a given text; see Chapter 10 for an extended example of such an analysis.

One last caveat is that the size of the chunk of text that we use to add up unigram sentiment scores can have an effect on an analysis. A text the size of many paragraphs can often have positive and negative sentiment averaged out to about zero, while sentence-sized or paragraph-sized text often works better.

3.2 Sentiment analysis with inner join

With data in a tidy format, sentiment analysis can be done as an inner join. This is another of the great successes of viewing text mining as a tidy data analysis task; much as removing stop words is an antijoin operation, performing sentiment analysis is an inner join operation.

Let's look at the words with a joy score from the NRC lexicon. What are the most common joy words in *Emma*? First, we need to take the text of the novels and convert the text to the tidy format using `unnest_tokens()` .

```
library(janeaustenr)
library(dplyr)
library(stringr)

tidy_books <- austen_books() %>%
  group_by(book) %>%
  mutate(linenumber = row_number(),
         chapter = cumsum(str_detect(text, regex("^chapter [\\divxlc]",
                                                 ignore_case = TRUE)))) %>%
  ungroup() %>%
  unnest_tokens(word, text)
```

Now that the text is in a tidy format with one term per row, we are ready to do the sentiment analysis. First, let's use the NRC lexicon and `filter()` for the joy words. Next, let's `filter()` the data frame with the text from the books for the words from *Emma* and then use `inner_join()` to perform the sentiment analysis. What are the most common joy words in *Emma*? Let's use `count()` from dplyr.

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

tidy_books %>%
  filter(book == "Emma") %>%
  inner_join(nrcjoy) %>%
  count(word, sort = TRUE)
```

```
## # A tibble: 303 × 2
##       word     n
##      <chr> <int>
## 1     good   359
## 2    young   192
## 3   friend   166
## 4     hope   143
## 5    happy   125
## 6     love   117
## 7     deal    92
## 8    found    92
## 9  present    89
## 10    kind    82
## # ... with 293 more rows
```

We see many positive, happy words about hope, friendship, and love here.

Or instead we could examine how sentiment changes during each novel. We can do this with just a handful of lines that are mostly dplyr functions. First, we find a sentiment score for each word using the Bing lexicon and `inner_join()` .

Next, we count up how many positive and negative words there are in defined sections of each book. We define an `index` here to keep track of where we are in the narrative; this index (using integer division) counts up sections of 100 lines of text. The `%/%` operator does integer division (`x %/% y` is equivalent to `floor(x/y)`) so the index keeps track of which 100-line section of text we are counting up negative and positive sentiment in. Small sections of text may not have enough words in them to get a good estimate of sentiment while really large sections can wash out narrative structure. For these books, using 100 lines works well, but this can vary depending on individual texts, how long the lines were to start with, etc.

We then use `spread()` so that we have negative and positive sentiment in separate columns, and lastly calculate a net sentiment (positive - negative).

```r
library(tidyr)

janeaustensentiment <- tidy_books %>%
  inner_join(get_sentiments("bing")) %>%
  count(book, index = linenumber %/% 100, sentiment) %>%
  spread(sentiment, n, fill = 0) %>%
  mutate(sentiment = positive - negative)
```

Now we can plot these sentiment scores across the plot trajectory of each novel. Notice that we are plotting against the `index` on the x-axis that keeps track of narrative time in sections of text.

```r
library(ggplot2)

ggplot(janeaustensentiment, aes(index, sentiment, fill = book)) +
  geom_bar(alpha = 0.8, stat = "identity", show.legend = FALSE) +
  facet_wrap(~book, ncol = 2, scales = "free_x")
```

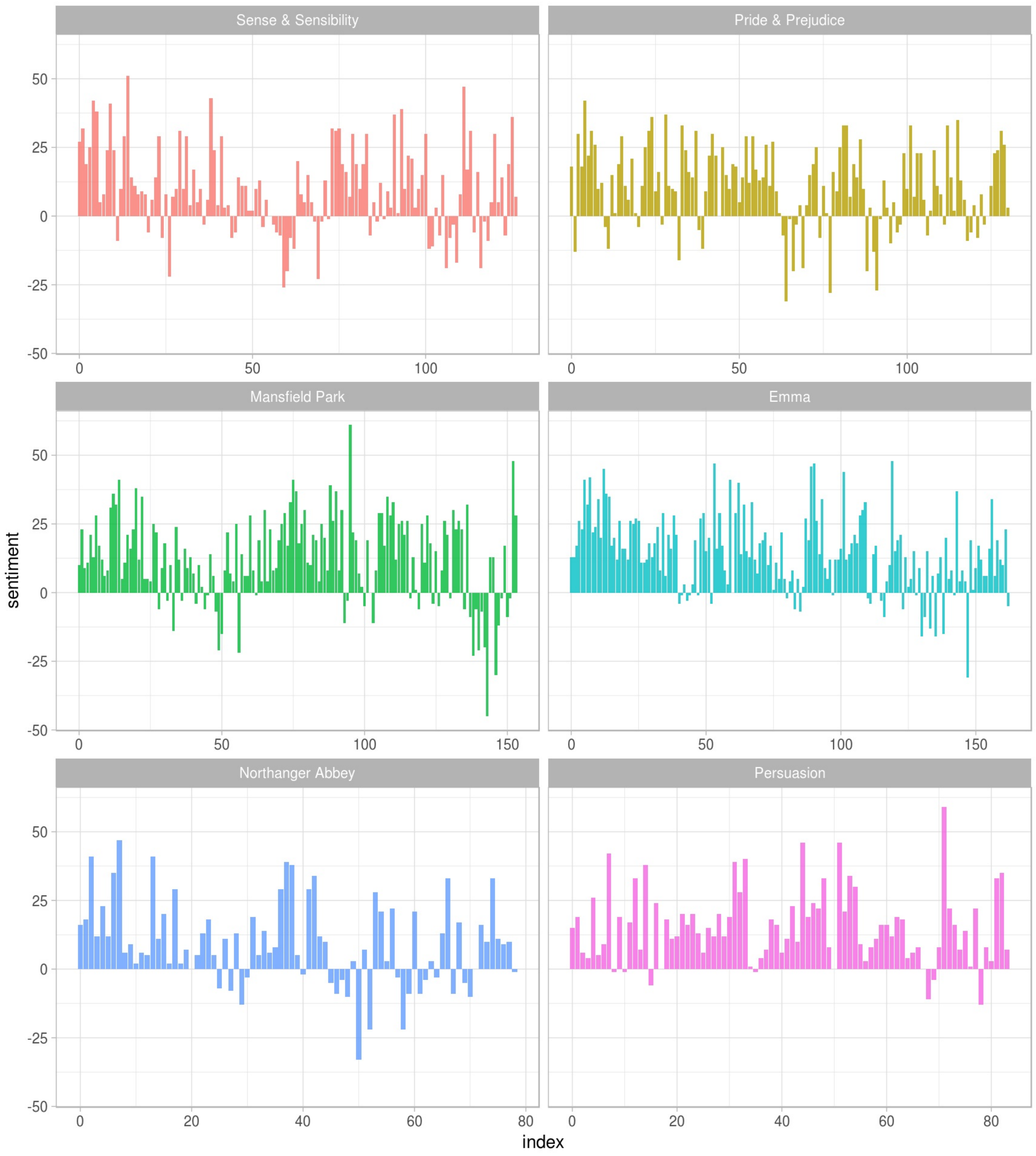


Figure 3.1: Sentiment through the narratives of Jane Austen's novels

We can see in Figure 3.1 how the plot of each novel changes toward more positive or negative sentiment over the trajectory of the story.

3.3 Comparing the three sentiment dictionaries

With several options for sentiment lexicons, you might want some more information on which one is appropriate for your purposes. Let's use all three sentiment lexicons and examine how the sentiment changes across the narrative arc of *Pride and Prejudice*. First, let's use `filter()` to choose only the words from the one novel we are interested in.

```
pride_prejudice <- tidy_books %>%
  filter(book == "Pride & Prejudice")

pride_prejudice
```

```
## # A tibble: 122,204 × 4
##                  book linenumber chapter      word
##                <fctr>      <int>   <int>     <chr>
## 1  Pride & Prejudice          1       0     pride
## 2  Pride & Prejudice          1       0       and
## 3  Pride & Prejudice          1       0 prejudice
## 4  Pride & Prejudice          3       0        by
## 5  Pride & Prejudice          3       0      jane
## 6  Pride & Prejudice          3       0    austen
## 7  Pride & Prejudice          7       1   chapter
## 8  Pride & Prejudice          7       1         1
## 9  Pride & Prejudice         10       1        it
## 10 Pride & Prejudice         10       1        is
## # ... with 122,194 more rows
```

Now, we can use `inner_join()` to calculate the sentiment in different ways. Remember from above that the AFINN lexicon measures sentiment with a numeric score between -5 and 5, while the other two lexicons categorize words in a binary fashion, either positive or negative. To find a sentiment score in chunks of text throughout the novel, we will need to use a different pattern for the AFINN lexicon than for the other two.

Let’s again use integer division (`%/%`) to define larger sections of text that span multiple lines, and we can use the same pattern with `count()` , `spread()` , and `mutate()` to find the net sentiment in each of these sections of text.

```
afinn <- pride_prejudice %>%
  inner_join(get_sentiments("afinn")) %>%
  group_by(index = linenumber %/% 80) %>%
  summarise(sentiment = sum(score)) %>%
  mutate(method = "AFINN")

bing_and_nrc <- bind_rows(pride_prejudice %>%
                            inner_join(get_sentiments("bing")) %>%
                            mutate(method = "Bing et al."),
                          pride_prejudice %>%
                            inner_join(get_sentiments("nrc") %>%
                                         filter(sentiment %in% c("positive",
                                                                 "negative"))) %>%
                            mutate(method = "NRC")) %>%
  count(method, index = linenumber %/% 80, sentiment) %>%
  spread(sentiment, n, fill = 0) %>%
  mutate(sentiment = positive - negative)
```

We now have an estimate of the net sentiment (positive - negative) in each chunk of the novel text for each sentiment lexicon. Let’s bind them together and plot them.

```
bind_rows(afinn,
          bing_and_nrc) %>%
  ggplot(aes(index, sentiment, fill = method)) +
  geom_bar(alpha = 0.8, stat = "identity", show.legend = FALSE) +
  facet_wrap(~method, ncol = 1, scales = "free_y")
```

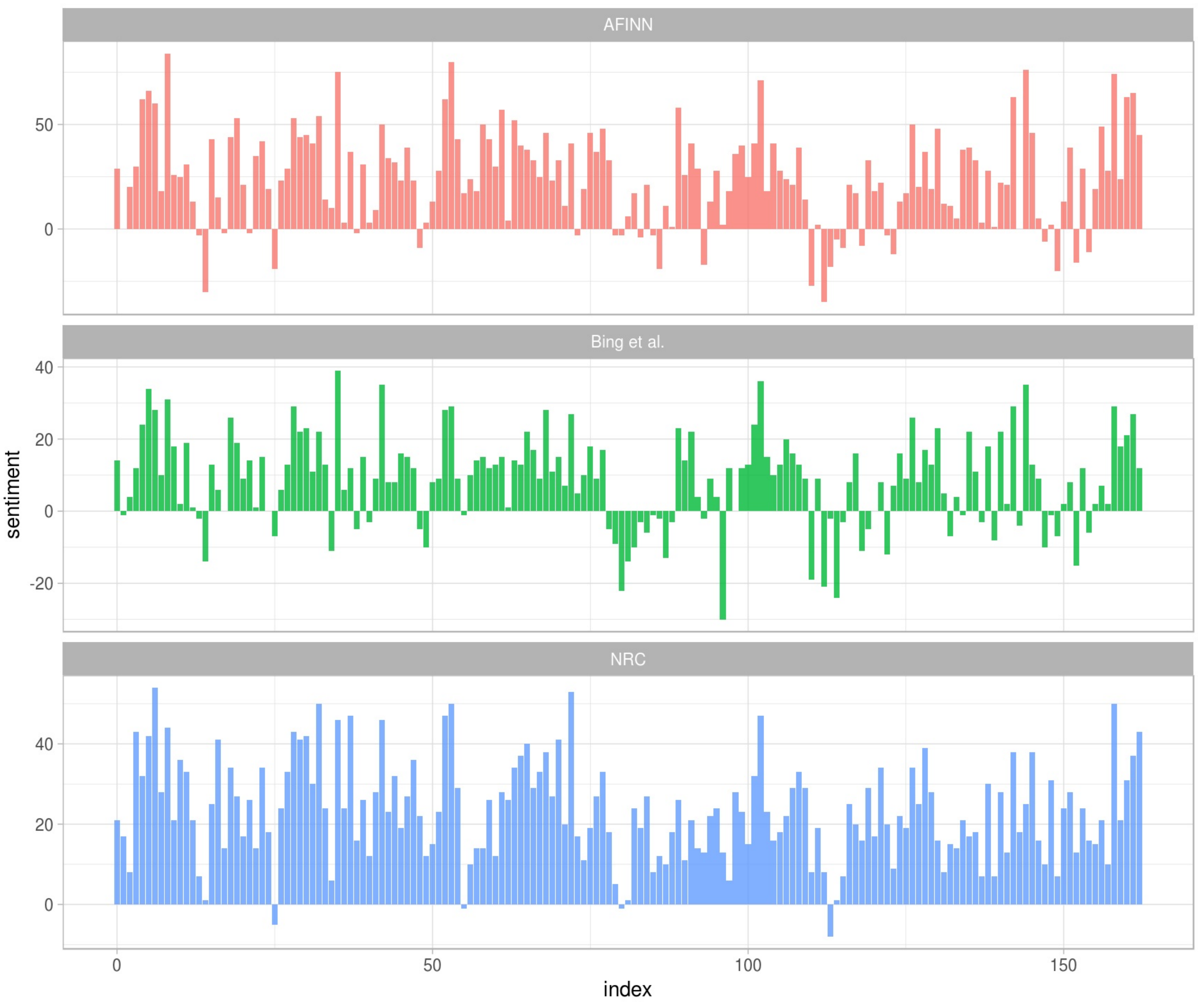


Figure 3.2: Comparing three sentiment lexicons using *Pride and Prejudice*

The three different lexicons for calculating sentiment give results that are different in an absolute sense but have similar relative trajectories through the novel. We see similar dips and peaks in sentiment at about the same places in the novel, but the absolute values are significantly different. The AFINN lexicon gives the largest absolute values, with high positive values. The lexicon from Bing et al. has lower absolute values and seems to label larger blocks of contiguous positive or negative text. The NRC results are shifted higher relative to the other two, labeling the text more positively, but detects similar relative changes in the text. We find similar differences between the methods when looking at other novels; the NRC sentiment is high, the AFINN sentiment has more variance, the Bing et al. sentiment appears to find longer stretches of similar text, but all three agree roughly on the overall trends in the sentiment through a narrative arc.

Why is, for example, the result for the NRC lexicon biased so high in sentiment compared to the Bing et al. result? Let’s look briefly at how many positive and negative words are in these lexicons.

```
get_sentiments("nrc") %>%
     filter(sentiment %in% c("positive",
                             "negative")) %>%
  count(sentiment)
```

```
## # A tibble: 2 × 2
##   sentiment     n
##       <chr> <int>
## 1  negative  3324
## 2  positive  2312
```

```
get_sentiments("bing") %>%
  count(sentiment)
```

```
## # A tibble: 2 × 2
##   sentiment     n
##       <chr> <int>
## 1  negative  4782
## 2  positive  2006
```

Both lexicons have more negative than positive words, but the ratio of negative to positive words is higher in the Bing lexicon than the NRC lexicon. This will contribute to the effect we see in the plot above, as will any systematic difference in word matches, e.g. if the negative words in the NRC lexicon do not match the words that Jane Austen uses very well. Whatever the source of these differences, we see similar relative trajectories across the narrative arc, with similar changes in slope, but marked differences in absolute sentiment from lexicon to lexicon. This is all important context to keep in mind when choosing a sentiment lexicon for analysis.

3.4 Most common positive and negative words

One advantage of having the data frame with both sentiment and word is that we can analyze word counts that contribute to each sentiment.

```
bing_word_counts <- tidy_books %>%
  inner_join(get_sentiments("bing")) %>%
  count(word, sentiment, sort = TRUE) %>%
  ungroup()

bing_word_counts
```

```
## # A tibble: 2,585 × 3
##         word sentiment     n
##        <chr>     <chr> <int>
## 1       miss  negative  1855
## 2       well  positive  1523
## 3       good  positive  1380
## 4      great  positive   981
## 5       like  positive   725
## 6     better  positive   639
## 7     enough  positive   613
## 8      happy  positive   534
## 9       love  positive   495
## 10 pleasure  positive   462
## # ... with 2,575 more rows
```

This can be shown visually, and we can pipe straight into ggplot2, if we like, because of the way we are consistently using tools built for handling tidy data frames.

```
bing_word_counts %>%
  group_by(sentiment) %>%
  top_n(10) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n, fill = sentiment)) +
  geom_bar(alpha = 0.8, stat = "identity", show.legend = FALSE) +
  facet_wrap(~sentiment, scales = "free_y") +
  labs(y = "Contribution to sentiment",
       x = NULL) +
  coord_flip()
```

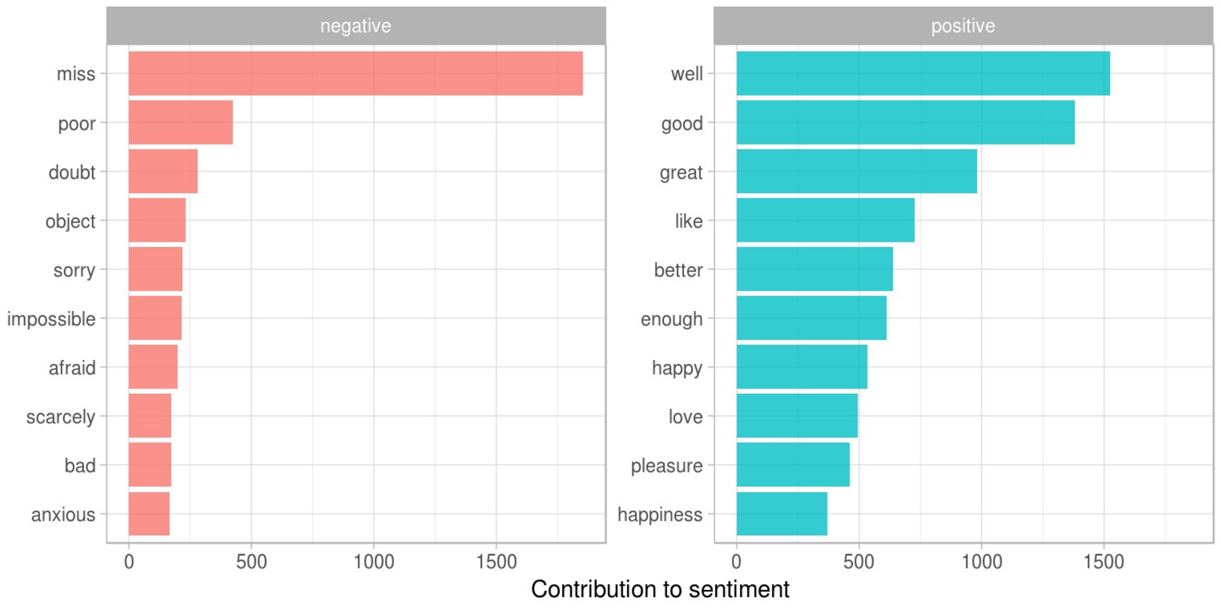


Figure 3.3: Words that contribute to positive and negative sentiment in Jane Austen's novels

This lets us spot an anomaly in the sentiment analysis; the word "miss" is coded as negative but it is used as a title for young, unmarried women in Jane Austen's works. If it were appropriate for our purposes, we could easily add “miss” to a custom stop-words list using `bind_rows()`.

3.5 Wordclouds

We’ve seen that this tidy text mining approach works well with ggplot2, but having our data in a tidy format is useful for other plots as well.

For example, consider the wordcloud package, which uses base R graphics. Let’s look at the most common words in Jane Austen’s works as a whole again.

```r
library(wordcloud)

tidy_books %>%
  anti_join(stop_words) %>%
  count(word) %>%
  with(wordcloud(word, n, max.words = 100))
```

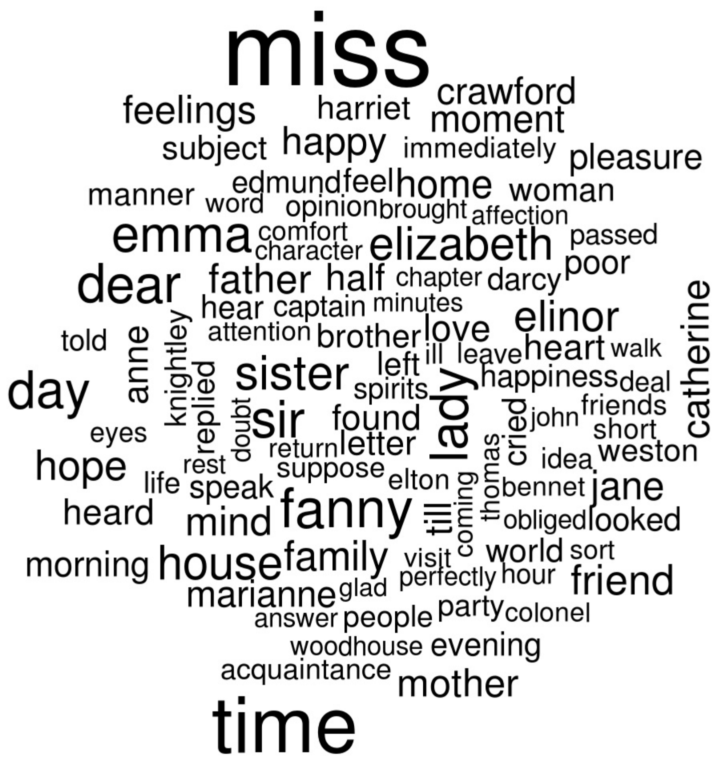


Figure 3.4: The most common words in Jane Austen’s novels

In other functions, such as `comparison.cloud()`, you may need to turn the data frame into a matrix with reshape2’s `acast()`. Let’s do the sentiment analysis to tag positive and negative words using an inner join, then find the most common positive and negative words. Until the step where we need to send the data to `comparison.cloud()`, this can all be done with joins, piping, and dplyr because our data is in tidy format.

```
library(reshape2)

tidy_books %>%
  inner_join(get_sentiments("bing")) %>%
  count(word, sentiment, sort = TRUE) %>%
  acast(word ~ sentiment, value.var = "n", fill = 0) %>%
  comparison.cloud(colors = c("#F8766D", "#00BFC4"),
                   max.words = 100)
```

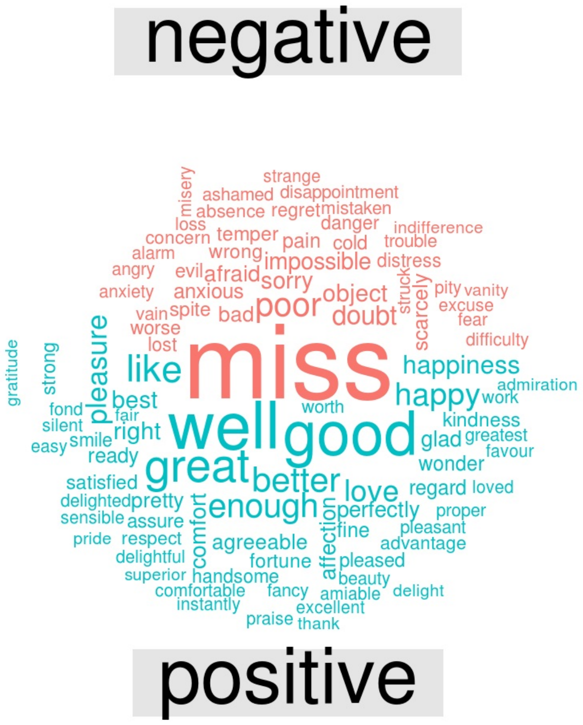


Figure 3.5: Most common positive and negative words in Jane Austen's novels

The size of a word's text in Figure 3.5 is in proportion to its frequency within its sentiment. We can use this visualization to see the most important positive and negative words, but the sizes of the words are not comparable across sentiments.

3.6 Looking at units beyond just words

Lots of useful work can be done by tokenizing at the word level, but sometimes it is useful or necessary to look at different units of text. For example, some sentiment analysis algorithms look beyond only unigrams (i.e. single words) to try to understand the sentiment of a sentence as a whole. These algorithms try to understand that

> I am not having a good day.

is a sad sentence, not a happy one, because of negation. The Stanford CoreNLP tools and the sentimentr R package (currently available on Github but not CRAN) are examples of such sentiment analysis algorithms. For these, we may want to tokenize text into sentences.

```
PandP_sentences <- data_frame(text = prideprejudice) %>%
  unnest_tokens(sentence, text, token = "sentences")
```

Let’s look at just one.

```
PandP_sentences$sentence[2]
```

```
## [1] "however little known the feelings or views of such a man may be on his first entering 
```

The sentence tokenizing does seem to have a bit of trouble with UTF-8 encoded text, especially with sections of dialogue; it does much better with punctuation in ASCII. One possibility, if this is important, is to try using `iconv()`, with something like `iconv(text, to = 'latin1')` in a mutate statement before unnesting.

Another option in `unnest_tokens()` is to split into tokens using a regex pattern. We could use this, for example, to split the text of Jane Austen’s novels into a data frame by chapter.

```
austen_chapters <- austen_books() %>%
  group_by(book) %>%
  unnest_tokens(chapter, text, token = "regex",
                pattern = "Chapter|CHAPTER [\\dIVXLC]") %>%
  ungroup()

austen_chapters %>%
  group_by(book) %>%
  summarise(chapters = n())
```

```
## # A tibble: 6 × 2
##                  book chapters
##                <fctr>    <int>
## 1 Sense & Sensibility       51
## 2   Pride & Prejudice       62
## 3      Mansfield Park       49
## 4                Emma       56
## 5    Northanger Abbey       32
## 6          Persuasion       25
```

We have recovered the correct number of chapters in each novel (plus an “extra” row for each novel title). In this data frame, each row corresponds to one chapter.

Near the beginning of this vignette, we used a similar regex to find where all the chapters were in Austen’s novels for a tidy data frame organized by one-word-per-row. We can use tidy text analysis to ask questions such as what are the most negative chapters in each of Jane Austen’s novels? First, let’s get the list of negative words from the Bing lexicon. Second, let’s make a data frame of how many words are in each chapter so we can normalize for the length of chapters. Then, let’s find the number of negative words in each chapter and divide by the total words in each chapter. Which chapter has the highest proportion of negative words?

```
bingnegative <- get_sentiments("bing") %>%
  filter(sentiment == "negative")

wordcounts <- tidy_books %>%
  group_by(book, chapter) %>%
  summarize(words = n())

tidy_books %>%
  semi_join(bingnegative) %>%
  group_by(book, chapter) %>%
  summarize(negativewords = n()) %>%
  left_join(wordcounts, by = c("book", "chapter")) %>%
  mutate(ratio = negativewords/words) %>%
  filter(chapter != 0) %>%
  top_n(1) %>%
  ungroup
```

```
## # A tibble: 6 × 5
##                  book chapter negativewords words      ratio
##                <fctr>   <int>         <int> <int>      <dbl>
## 1 Sense & Sensibility      43           161  3405 0.04728341
## 2   Pride & Prejudice      34           111  2104 0.05275665
## 3      Mansfield Park      46           173  3685 0.04694708
## 4                Emma      15           151  3340 0.04520958
## 5    Northanger Abbey      21           149  2982 0.04996647
## 6          Persuasion       4            62  1807 0.03431101
```

These are the chapters with the most sad words in each book, normalized for number of words in the chapter. What is happening in these chapters? In Chapter 43 of *Sense and Sensibility* Marianne is seriously ill, near death, and in Chapter 34 of *Pride and Prejudice* Mr. Darcy proposes for the first time (so badly!). Chapter 46 of *Mansfield Park* is almost the end, when everyone learns of Henry's scandalous adultery, Chapter 15 of *Emma* is when horrifying Mr. Elton proposes, and in Chapter 21 of *Northanger Abbey* Catherine is deep in her Gothic faux fantasy of murder, etc. Chapter 4 of *Persuasion* is when the reader gets the full flashback of Anne refusing Captain Wentworth and how sad she was and what a terrible mistake she realized it to be.

4 Analyzing word and document frequency: tf-idf

A central question in text mining and natural language processing is how to quantify what a document is about. Can we do this by looking at the words that make up the document? One measure of how important a word may be is its *term frequency* (tf), how frequently a word occurs in a document; we have examined how to measure word frequency using tidy data principles in Chapter 2. There are words in a document, however, that occur many times but may not be important; in English, these are probably words like “the”, “is”, “of”, and so forth. We might take the approach of adding words like these to a list of stop words and removing them before analysis, but it is possible that some of these words might be more important in some documents than others. A list of stop words is not a very sophisticated approach to adjusting term frequency for commonly used words.

Another approach is to look at a term’s *inverse document frequency* (idf), which decreases the weight for commonly used words and increases the weight for words that are not used very much in a collection of documents. This can be combined with term frequency to calculate a term’s *tf-idf* (the two quantities multiplied together), the frequency of a term adjusted for how rarely it is used. It is intended to measure how important a word is to a document in a collection (or corpus) of documents. It is a rule-of-thumb or heuristic quantity; while it has proved useful in text mining, search engines, etc., its theoretical foundations are considered less than firm by information theory experts. The inverse document frequency for any given term is defined as

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

We can use tidy data principles, as described in Chapter 2, to approach tf-idf analysis and use consistent, effective tools to quantify how important various terms are in a document that is part of a collection.

4.1 Term frequency in Jane Austen’s novels

Let’s start by looking at the published novels of Jane Austen and examine first term frequency, then tf-idf. We can start just by using dplyr verbs such as `group_by()` and `join()`. What are the most commonly used words in Jane Austen’s novels? (Let’s also calculate the total words in each novel here, for later use.)

```r
library(dplyr)
library(janeaustenr)
library(tidytext)

book_words <- austen_books() %>%
  unnest_tokens(word, text) %>%
  count(book, word, sort = TRUE) %>%
  ungroup()

total_words <- book_words %>%
  group_by(book) %>%
  summarize(total = sum(n))

book_words <- left_join(book_words, total_words)

book_words
```

```
## # A tibble: 40,379 × 4
##                  book  word     n  total
##                <fctr> <chr> <int>  <int>
## 1      Mansfield Park   the  6206 160460
## 2      Mansfield Park    to  5475 160460
## 3      Mansfield Park   and  5438 160460
## 4                Emma    to  5239 160996
## 5                Emma   the  5201 160996
## 6                Emma   and  4896 160996
## 7      Mansfield Park    of  4778 160460
## 8   Pride & Prejudice   the  4331 122204
## 9                Emma    of  4291 160996
## 10 Pride & Prejudice    to  4162 122204
## # ... with 40,369 more rows
```

There is one row in this `book_words` data frame for each word-book combination; `n` is the number of times that word is used in that book and `total` is the total words in that book. The usual suspects are here with the highest `n`, “the”, “and”, “to”, and so forth. Let’s look at the distribution of `n/total` for each novel, the number of times a word appears in a novel divided by the total number of terms (words) in that novel. This is exactly what term frequency is.

```r
library(ggplot2)

ggplot(book_words, aes(n/total, fill = book)) +
  geom_histogram(alpha = 0.8, show.legend = FALSE) +
  xlim(NA, 0.0009) +
  facet_wrap(~book, ncol = 2, scales = "free_y")
```

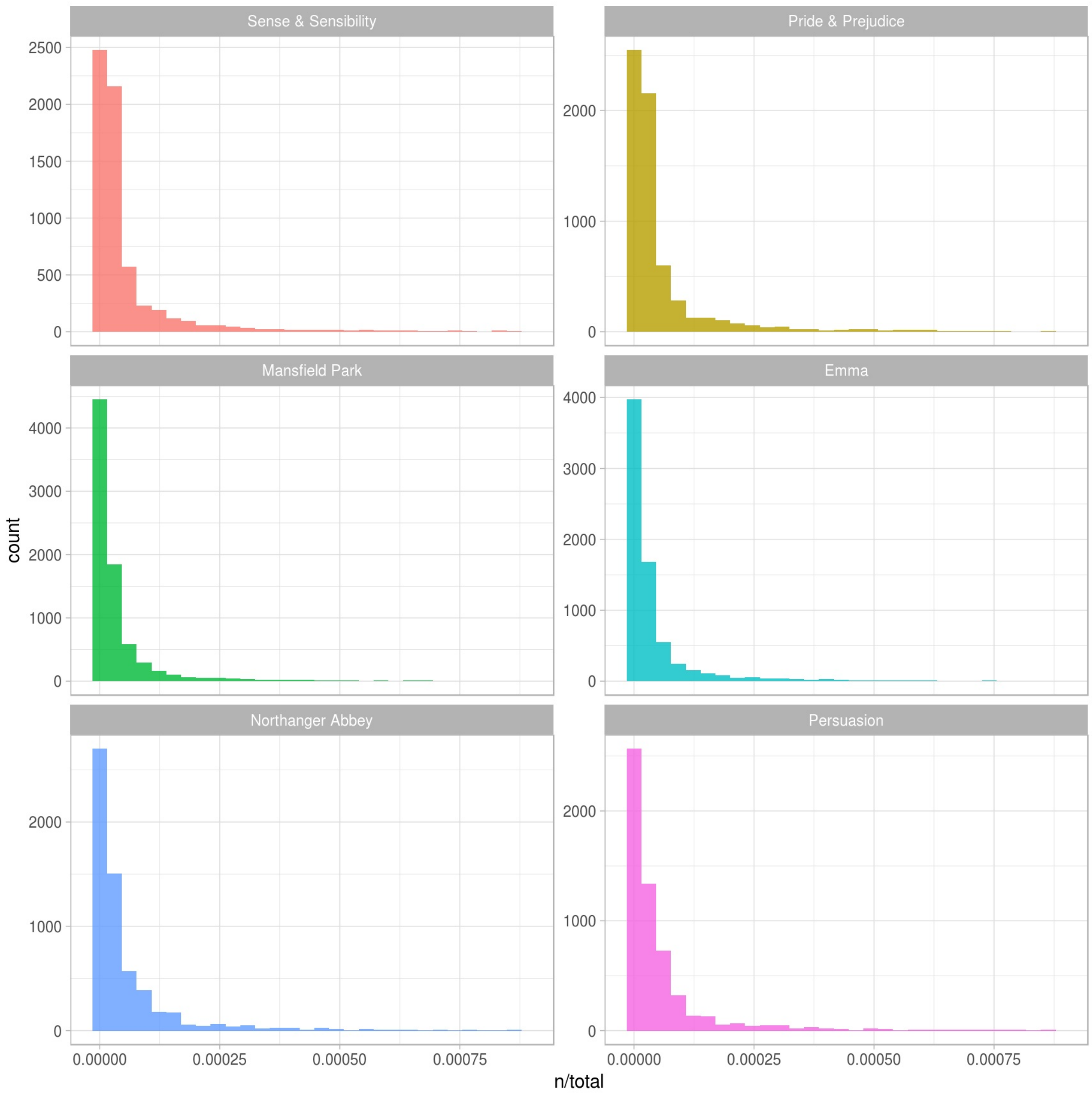


Figure 4.1: Term Frequency Distribution in Jane Austen's Novels

There are very long tails to the right for these novels (those extremely common words!) that we have not shown in these plots. These plots exhibit similar distributions for all the novels, with many words that occur rarely and fewer words that occur frequently.

4.2 Zipf's law

Distributions like those shown in Figure 4.1 are typical in language. In fact, those types of long-tailed distributions are so common in any given corpus of natural language (like a book, or a lot of text from a website, or spoken words) that the relationship between the frequency that a word is used and its rank has been the subject of study; a classic version of this relationship is called Zipf's law, after George Zipf, a 20th century American linguist. Zipf's law states that the frequency of a word is inversely proportional to its rank. Since we have the data frame we used to plot term frequency, we can examine Zipf's law for Jane Austen's novels with just a few lines of dplyr functions.

```
freq_by_rank <- book_words %>%
  group_by(book) %>%
  mutate(rank = row_number(),
         `term frequency` = n/total)

freq_by_rank
```

```
## Source: local data frame [40,379 x 6]
## Groups: book [6]
##
##                  book  word     n  total  rank `term frequency`
##                <fctr> <chr> <int>  <int> <int>            <dbl>
## 1     Mansfield Park   the  6206 160460     1       0.03867631
## 2     Mansfield Park    to  5475 160460     2       0.03412065
## 3     Mansfield Park   and  5438 160460     3       0.03389007
## 4               Emma    to  5239 160996     1       0.03254118
## 5               Emma   the  5201 160996     2       0.03230515
## 6               Emma   and  4896 160996     3       0.03041069
## 7     Mansfield Park    of  4778 160460     4       0.02977689
## 8  Pride & Prejudice   the  4331 122204     1       0.03544074
## 9               Emma    of  4291 160996     4       0.02665284
## 10 Pride & Prejudice    to  4162 122204     2       0.03405780
## # ... with 40,369 more rows
```

The `rank` column here tells us the rank of each word within the frequency table; the table was already ordered by `n` so we could use `row_number()` to find the rank. Then, we can calculate the term frequency in the same way we did before. Zipf's law is often visualized by plotting rank on the x-axis and term frequency on the y-axis, on logarithmic scales. Plotting this way, an inversely proportional relationship will have a constant, negative slope.

```
freq_by_rank %>%
  ggplot(aes(rank, `term frequency`, color = book)) +
  geom_line(size = 1.2, alpha = 0.8) +
  scale_x_log10() +
  scale_y_log10()
```

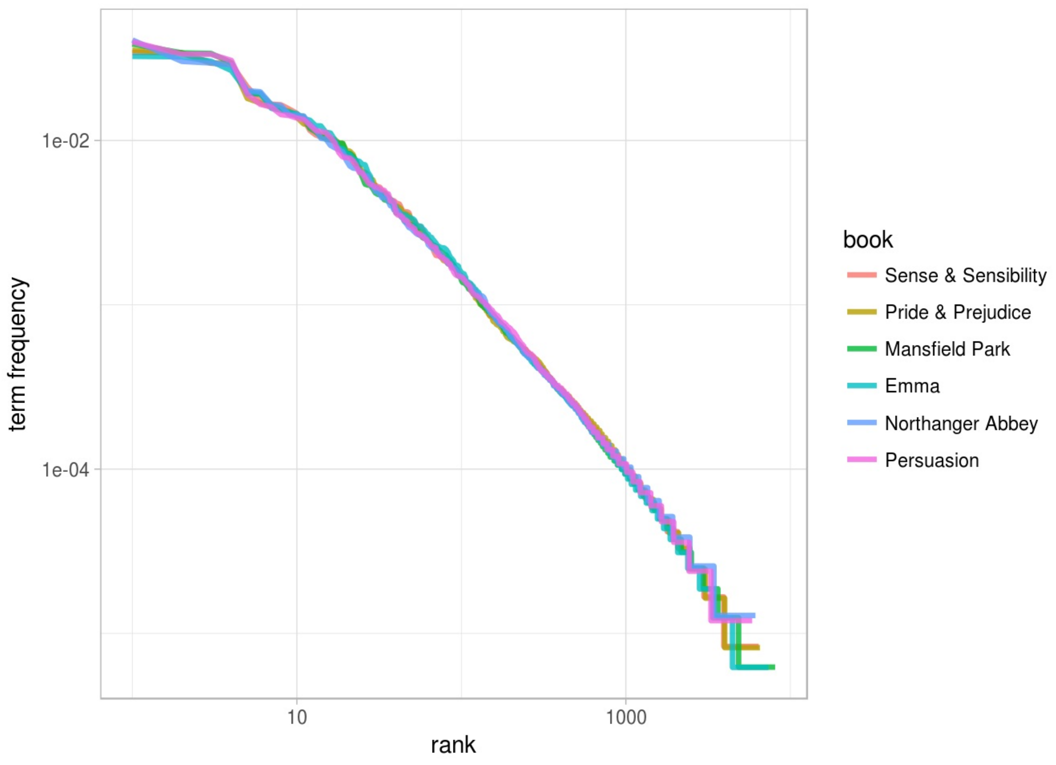


Figure 4.2: Zipf's law for Jane Austen's novels

Notice that this plot is in log-log coordinates. We see that all six of Jane Austen's novels are similar to each other, and that the relationship between rank and frequency does have negative slope. It is not quite constant, though; perhaps we could view this as a broken power law with, say, three sections. Let's see what the exponent of the power law is for the middle section of the rank range.

```
rank_subset <- freq_by_rank %>%
  filter(rank < 500,
         rank > 10)

lm(log10(`term frequency`) ~ log10(rank), data = rank_subset)
```

```
##
## Call:
## lm(formula = log10(`term frequency`) ~ log10(rank), data = rank_subset)
##
## Coefficients:
## (Intercept)  log10(rank)
##     -0.6225      -1.1125
```

Classic versions of Zipf's law have

\[\text{frequency} \propto \frac{1}{\text{rank}}\] and we have in fact gotten a slope close to -1 here. Let's plot this fitted power law with the data to see how it looks.

```
freq_by_rank %>%
  ggplot(aes(rank, `term frequency`, color = book)) +
  geom_abline(intercept = -0.62, slope = -1.1, color = "gray50", linetype = 2) +
  geom_line(size = 1.2, alpha = 0.8) +
  scale_x_log10() +
  scale_y_log10()
```

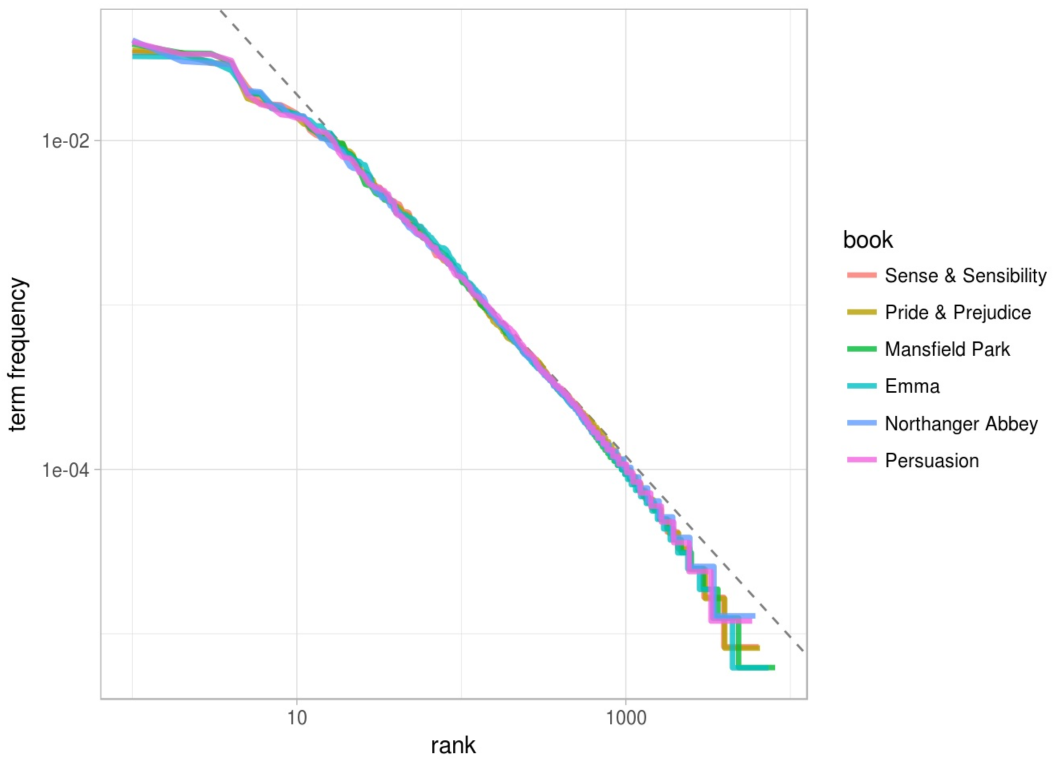


Figure 4.3: Fitting an exponent for Zipf’s law with Jane Austen’s novels

We have found a result close to the classic version of Zipf’s law for the corpus of Jane Austen’s novels. The deviations we see here at high rank are not uncommon for many kinds of language; a corpus of language often contains fewer rare words than predicted by a single power law. The deviations at low rank are more unusual. Jane Austen uses a lower percentage of the most common words than many collections of language. This kind of analysis could be extended to compare authors, or to compare any other collections of text; it can be implemented simply using tidy data principles.

4.3 The `bind_tf_idf` function

The idea of tf-idf is to find the important words for the content of each document by decreasing the weight for commonly used words and increasing the weight for words that are not used very much in a collection or corpus of documents, in this case, the group of Jane Austen’s novels as a whole. Calculating tf-idf attempts to find the words that are important (i.e., common) in a text, but not *too* common. Let’s do that now.

```
book_words <- book_words %>%
  bind_tf_idf(word, book, n)
book_words
```

```
## # A tibble: 40,379 × 7
##                 book  word     n  total         tf   idf tf_idf
##               <fctr> <chr> <int>  <int>      <dbl> <dbl>  <dbl>
## 1     Mansfield Park   the  6206 160460 0.03867631     0      0
## 2     Mansfield Park    to  5475 160460 0.03412065     0      0
## 3     Mansfield Park   and  5438 160460 0.03389007     0      0
## 4               Emma    to  5239 160996 0.03254118     0      0
## 5               Emma   the  5201 160996 0.03230515     0      0
## 6               Emma   and  4896 160996 0.03041069     0      0
## 7     Mansfield Park    of  4778 160460 0.02977689     0      0
## 8  Pride & Prejudice   the  4331 122204 0.03544074     0      0
## 9               Emma    of  4291 160996 0.02665284     0      0
## 10 Pride & Prejudice    to  4162 122204 0.03405780     0      0
## # ... with 40,369 more rows
```

Notice that idf and thus tf-idf are zero for these extremely common words. These are all words that appear in all six of Jane Austen’s novels, so the idf term (which will then be the natural log of 1) is zero. The inverse document frequency (and thus tf-idf) is very low (near zero) for words that occur in many of the documents in a collection; this is how this approach decreases the weight for common words. The inverse document frequency will be a higher number for words that occur in fewer of the documents in the collection.

Let’s look at terms with high tf-idf in Jane Austen’s works.

```
book_words %>%
  select(-total) %>%
  arrange(desc(tf_idf))
```

```
## # A tibble: 40,379 × 6
##                   book      word     n          tf      idf      tf_idf
##                 <fctr>     <chr> <int>       <dbl>    <dbl>       <dbl>
## 1  Sense & Sensibility    elinor   623 0.005193528 1.791759 0.009305552
## 2  Sense & Sensibility  marianne   492 0.004101470 1.791759 0.007348847
## 3       Mansfield Park  crawford   493 0.003072417 1.791759 0.005505032
## 4    Pride & Prejudice     darcy   373 0.003052273 1.791759 0.005468939
## 5           Persuasion    elliot   254 0.003036207 1.791759 0.005440153
## 6                 Emma      emma   786 0.004882109 1.098612 0.005363545
## 7     Northanger Abbey    tilney   196 0.002519928 1.791759 0.004515105
## 8                 Emma    weston   389 0.002416209 1.791759 0.004329266
## 9    Pride & Prejudice    bennet   294 0.002405813 1.791759 0.004310639
## 10          Persuasion wentworth   191 0.002283132 1.791759 0.004090824
## # ... with 40,369 more rows
```

Here we see all proper nouns, names that are in fact important in these novels. None of them occur in all of novels, and they are important, characteristic words for each text within the corpus of Jane Austen’s novels. Some of the values for idf are the same for different terms because there are 6 documents in this corpus and we are seeing the numerical value for \(\ln(6/1)\), \(\ln(6/2)\), etc. Let’s look at a visualization for these high tf-idf words.

```
plot_austen <- book_words %>%
  arrange(desc(tf_idf)) %>%
  mutate(word = factor(word, levels = rev(unique(word))))

ggplot(plot_austen[1:20,], aes(word, tf_idf, fill = book)) +
  geom_bar(alpha = 0.8, stat = "identity") +
  labs(x = NULL, y = "tf-idf") +
  coord_flip()
```

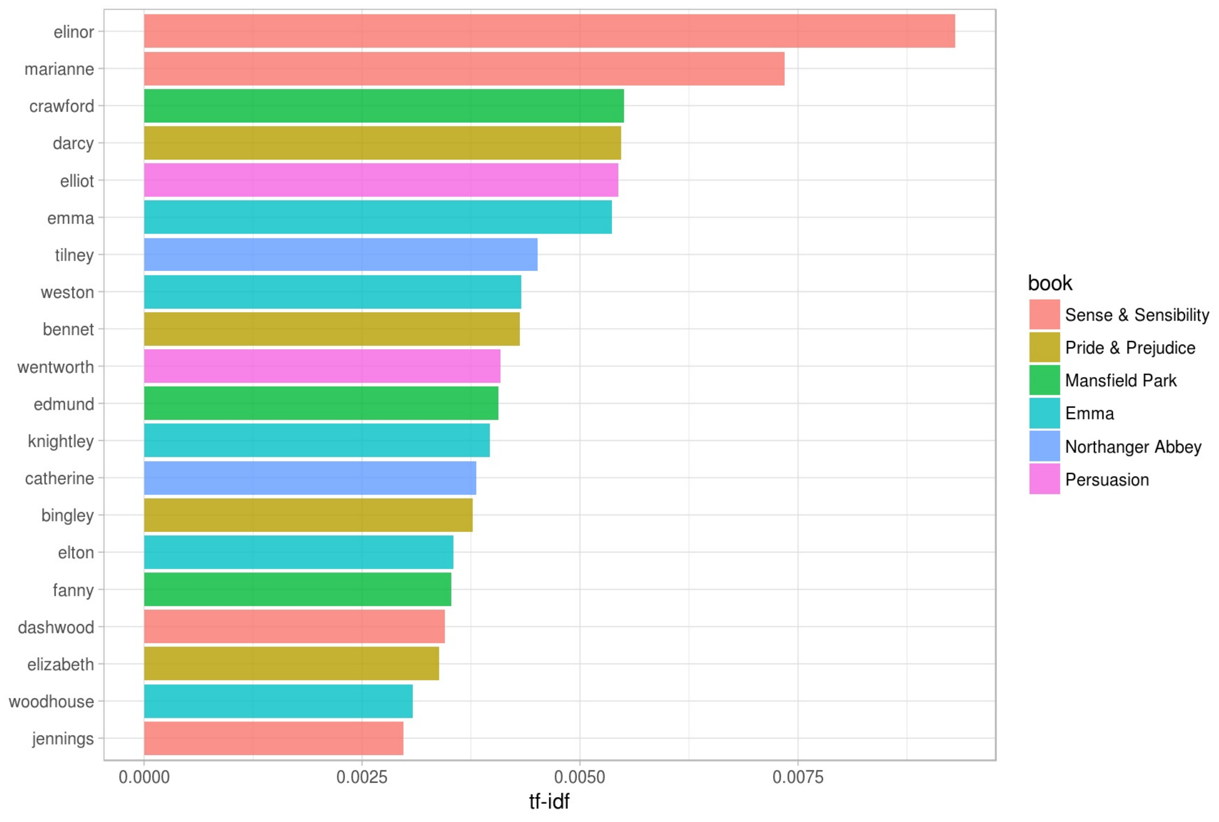


Figure 4.4: Highest tf-idf words in Jane Austen's Novels

Let's look at the novels individually.

```
plot_austen <- plot_austen %>%
  group_by(book) %>%
  top_n(15) %>%
  ungroup

ggplot(plot_austen, aes(word, tf_idf, fill = book)) +
  geom_bar(alpha = 0.8, stat = "identity", show.legend = FALSE) +
  labs(x = NULL, y = "tf-idf") +
  facet_wrap(~book, ncol = 2, scales = "free") +
  coord_flip()
```

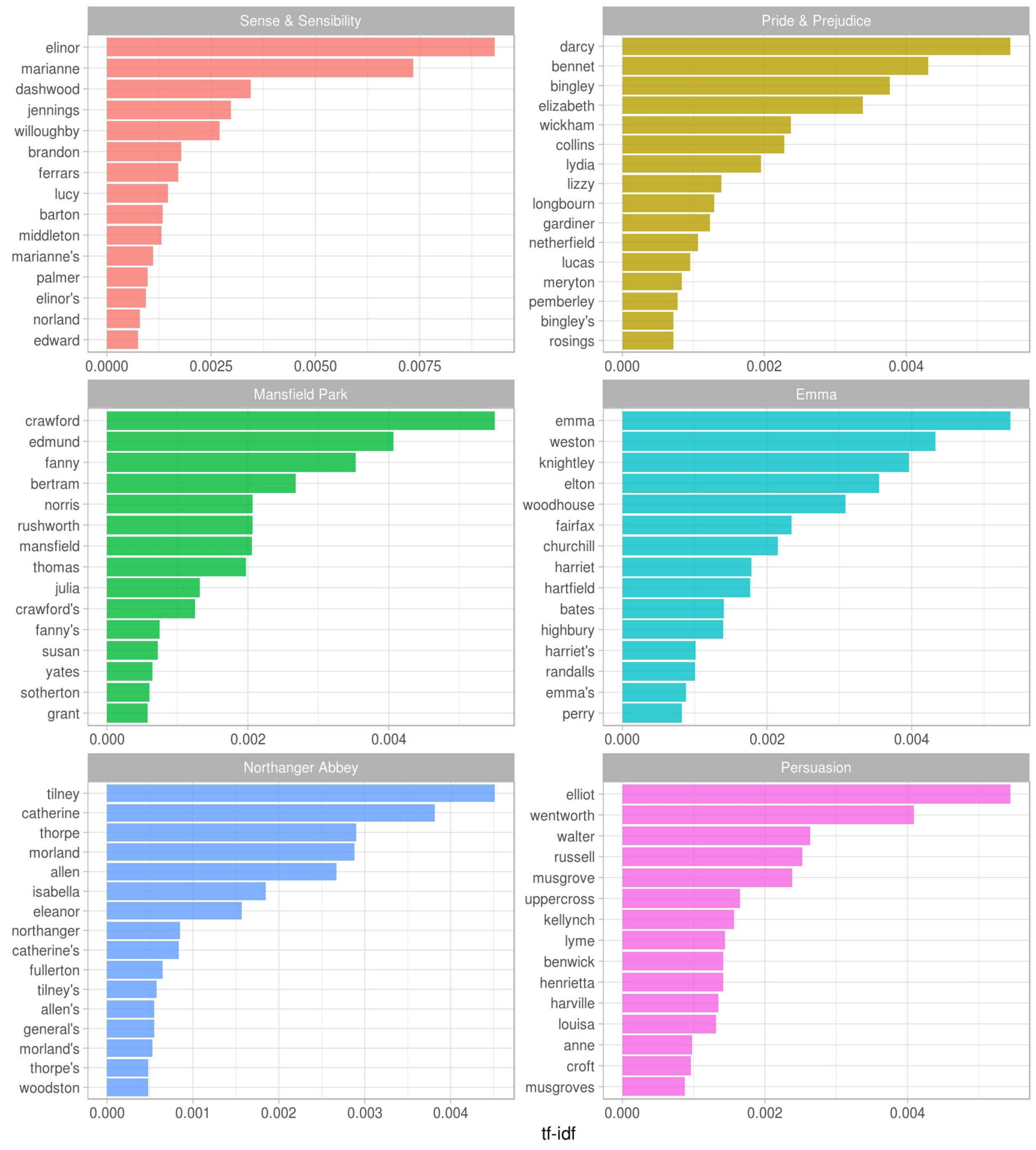


Figure 4.5: Highest tf-idf words in each of Jane Austen's Novels

Still all proper nouns! These words are, as measured by tf-idf, the most important to each novel and most readers would likely agree.

4.4 A corpus of physics texts

Let's work with another corpus of documents, to see what terms are important in a different set of works. In fact, let's leave the world of fiction and narrative entirely. Let's download some classic physics texts from Project Gutenberg and see what terms are important in these works, as measured by tf-idf. Let's download *Discourse on Floating Bodies* by Galileo Galilei, *Treatise on Light* by Christiaan Huygens, *Experiments with Alternate Currents of High Potential and High Frequency* by Nikola Tesla, and *Relativity: The Special and General Theory* by Albert Einstein.

This is a pretty diverse bunch. They may all be physics classics, but they were written across a 300-year timespan, and some of them were first written in other languages and then translated to English. Perfectly homogeneous these are not, but that doesn't stop this from being an interesting exercise!

```
library(gutenbergr)
physics <- gutenberg_download(c(37729, 14725, 13476, 5001),
                              meta_fields = "author")
```

Now that we have the texts, let's use `unnest_tokens()` and `count()` to find out how many times each word was used in each text.

```
physics_words <- physics %>%
  unnest_tokens(word, text) %>%
  count(author, word, sort = TRUE) %>%
  ungroup()

physics_words
```

```
## # A tibble: 12,592 × 3
##                 author  word     n
##                  <chr> <chr> <int>
## 1     Galilei, Galileo   the  3760
## 2        Tesla, Nikola   the  3604
## 3  Huygens, Christiaan   the  3553
## 4     Einstein, Albert   the  2994
## 5     Galilei, Galileo    of  2049
## 6     Einstein, Albert    of  2030
## 7        Tesla, Nikola    of  1737
## 8  Huygens, Christiaan    of  1708
## 9  Huygens, Christiaan    to  1207
## 10       Tesla, Nikola     a  1176
## # ... with 12,582 more rows
```

Here we see just the raw counts; we need to remember that these documents are all different lengths. Let's go ahead and calculate tf-idf.

```r
physics_words <- physics_words %>%
  bind_tf_idf(word, author, n)

plot_physics <- physics_words %>%
  arrange(desc(tf_idf)) %>%
  mutate(word = factor(word, levels = rev(unique(word)))) %>%
  mutate(author = factor(author, levels = c("Galilei, Galileo",
                                            "Huygens, Christiaan",
                                            "Tesla, Nikola",
                                            "Einstein, Albert")))

ggplot(plot_physics[1:20,], aes(word, tf_idf, fill = author)) +
  geom_bar(alpha = 0.8, stat = "identity") +
  labs(x = NULL, y = "tf-idf") +
  coord_flip()
```

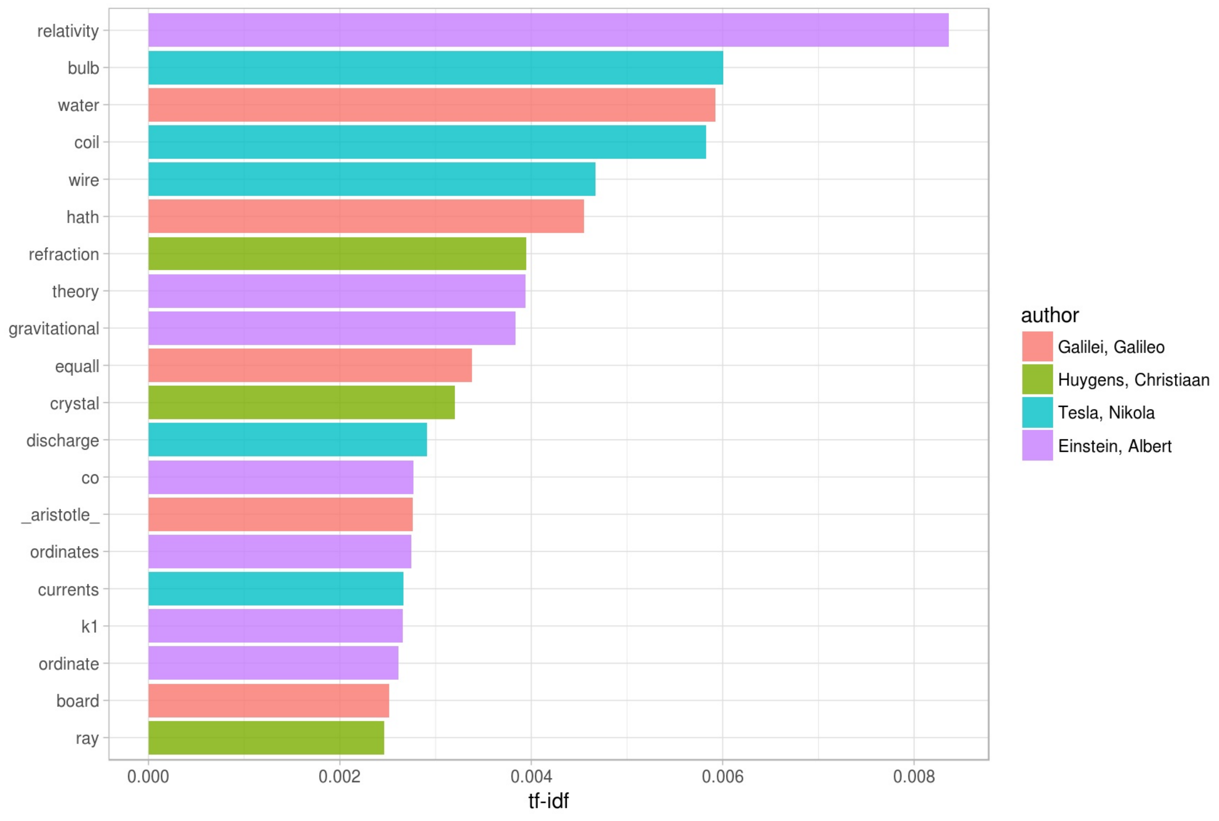


Figure 4.6: Highest tf-idf words in classic physics corpus

Nice! Let's look at each text individually.

```
plot_physics <- plot_physics %>%
  group_by(author) %>%
  top_n(15, tf_idf) %>%
  mutate(word = reorder(word, tf_idf))

ggplot(plot_physics, aes(word, tf_idf, fill = author)) +
  geom_bar(alpha = 0.8, stat = "identity", show.legend = FALSE) +
  labs(x = NULL, y = "tf-idf") +
  facet_wrap(~author, ncol = 2, scales = "free") +
  coord_flip()
```

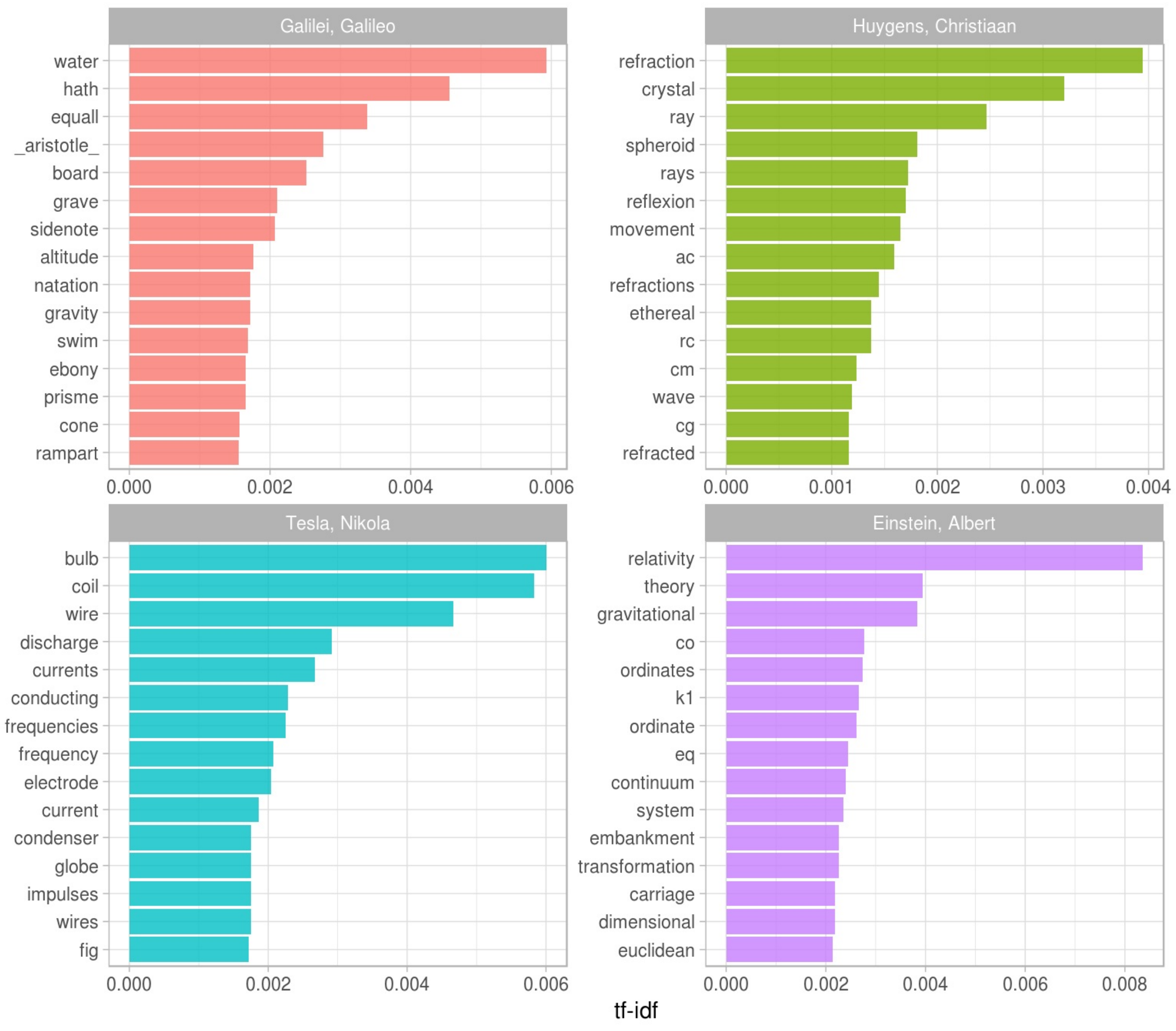


Figure 4.7: Highest tf-idf words in each physics texts

Very interesting indeed. One thing we see here is “gif” in the Einstein text?!

```
grep("eq\\.", physics$text, value = TRUE)[1:10]
```

```
##  [1] "                         eq. 1: file eq01.gif"
##  [2] "                         eq. 2: file eq02.gif"
##  [3] "                         eq. 3: file eq03.gif"
##  [4] "                         eq. 4: file eq04.gif"
##  [5] "                       eq. 05a: file eq05a.gif"
##  [6] "                       eq. 05b: file eq05b.gif"
##  [7] "the distance between the points being eq. 06 ."
##  [8] "direction of its length with a velocity v is eq. 06 of a metre."
##  [9] "velocity v=c we should have eq. 06a ,"
## [10] "the rod as judged from K1 would have been eq. 06 ;"
```

Some cleaning up of the text is in order. “K1” is the name of a coordinate system for Einstein:

```
grep("K1", physics$text, value = TRUE)[1]
```

```
## [1] "to a second co-ordinate system K1 provided that the latter is"
```

Maybe it makes sense to keep this one. Also notice that in this line we have “co-ordinate”, which explains why there are separate “co” and “ordinate” items in the high tf-idf words for the Einstein text; the `unnest_tokens()` function separates around punctuation.

“AB”, “RC”, and so forth are names of rays, circles, angles, and so forth for Huygens.

```
grep("AK", physics$text, value = TRUE)[1]
```

```
## [1] "Now let us assume that the ray has come from A to C along AK, KC; the"
```

Let’s remove some of these less meaningful words to make a better, more meaningful plot. Notice that we make a custom list of stop words and use `anti_join()` to remove them; this is a flexible approach that can be used in many situations. We will need to go back a few steps since we are removing words from the tidy data frame.

```r
mystopwords <- data_frame(word = c("eq", "co", "rc", "ac", "ak", "bn",
                                   "fig", "file", "cg", "cb", "cm"))
physics_words <- anti_join(physics_words, mystopwords, by = "word")
plot_physics <- physics_words %>%
  arrange(desc(tf_idf)) %>%
  mutate(word = factor(word, levels = rev(unique(word)))) %>%
  group_by(author) %>%
  top_n(15, tf_idf) %>%
  ungroup %>%
  mutate(author = factor(author, levels = c("Galilei, Galileo",
                                            "Huygens, Christiaan",
                                            "Tesla, Nikola",
                                            "Einstein, Albert")))

ggplot(plot_physics, aes(word, tf_idf, fill = author)) +
  geom_bar(alpha = 0.8, stat = "identity", show.legend = FALSE) +
  labs(x = NULL, y = "tf-idf") +
  facet_wrap(~author, ncol = 2, scales = "free") +
  coord_flip()
```

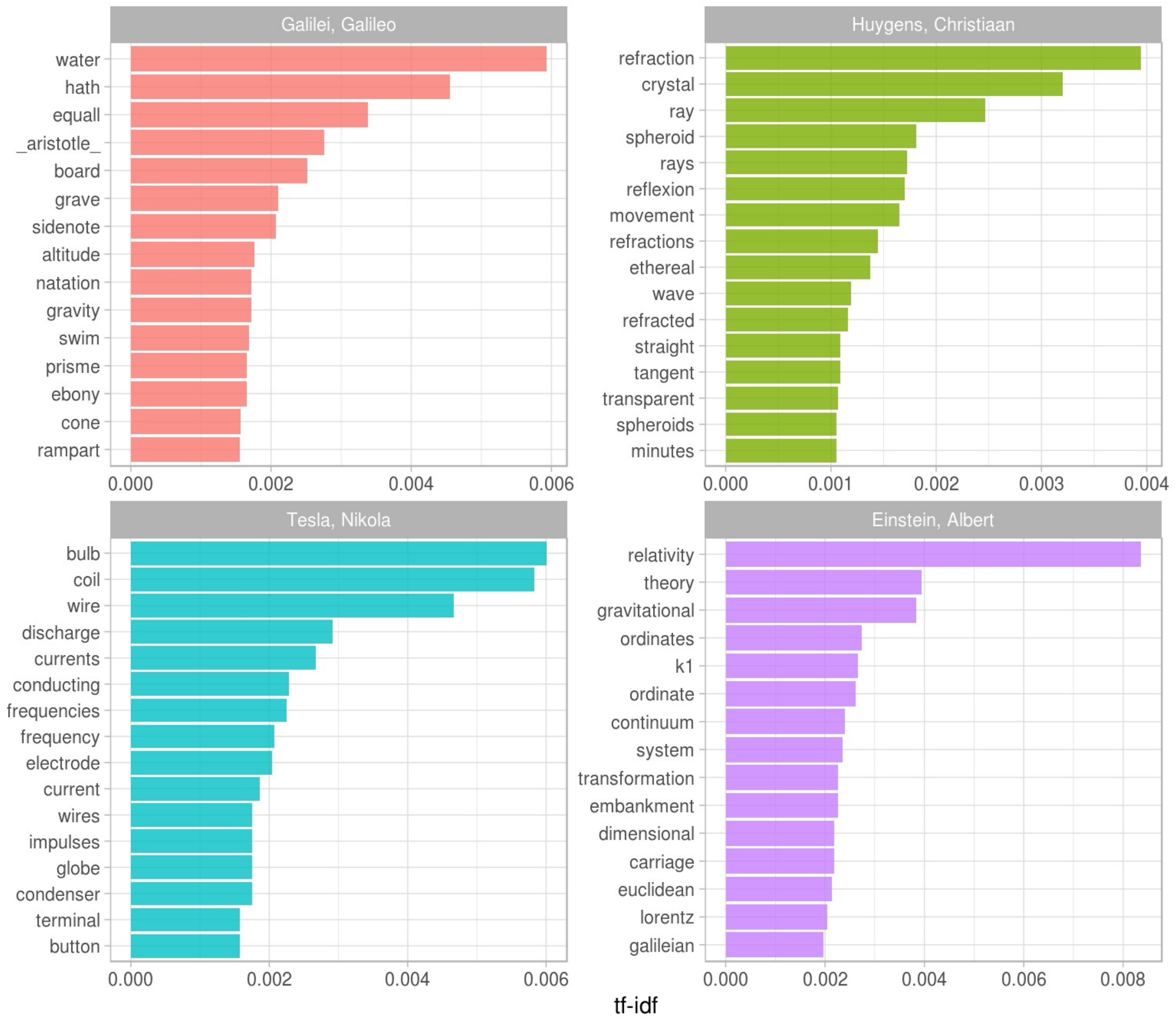


Figure 4.8: Highest tf-idf words in classic physics texts

We don’t hear enough about ramparts or things being ethereal in physics today.

5 Relationships between words

So far we've considered words as individual units, and considered their relationships to sentiments or to documents. However, many interesting text analyses are based on the relationships between words, whether examining which words tend to follow others immediately, or that tend to co-occur within the same documents.

In this chapter, we'll explore some of the methods tidytext offers for calculating and visualizing relationships between words in your text dataset. This includes the `token = "ngrams"` argument, which tokenizes by pairs of adjacent words rather than by individual ones. We'll also introduce two new packages: ggraph, which extends ggplot2 to construct network plots, and widyr, which calculates pairwise correlations and distances within a tidy data frame. Together these expand our toolbox for exploring text within the tidy data framework.

5.1 Tokenizing by n-gram

We've been using the `unnest_tokens` function to tokenize by word, or sometimes by sentence or paragraph, which is useful for the kinds of sentiment and frequency analyses we've been doing so far. But we can also use the function to tokenize into consecutive sequences of words, called **n-grams**. By seeing how often word X is followed by word Y, we can then build a model of the relationships between them.

We do this by adding the `token = "ngrams"` option to `unnest_tokens()`, and setting `n` to the number of words we wish to capture in each n-gram. When we set `n` to 2, we are examining pairs of two consecutive words, often called "bigrams":

```r
library(dplyr)
library(tidytext)
library(janeaustenr)

austen_bigrams <- austen_books() %>%
  unnest_tokens(bigram, text, token = "ngrams", n = 2)

austen_bigrams
```

```
## # A tibble: 725,048 × 2
##                   book          bigram
##                 <fctr>           <chr>
## 1  Sense & Sensibility       sense and
## 2  Sense & Sensibility and sensibility
## 3  Sense & Sensibility  sensibility by
## 4  Sense & Sensibility         by jane
## 5  Sense & Sensibility     jane austen
## 6  Sense & Sensibility     austen 1811
## 7  Sense & Sensibility    1811 chapter
## 8  Sense & Sensibility       chapter 1
## 9  Sense & Sensibility           1 the
## 10 Sense & Sensibility      the family
## # ... with 725,038 more rows
```

This data structure is still a variation of the tidy text format. It is structured as one-row-per-token (with extra metadata, such as `book`, still preserved), but each token now represents a bigram. Notice that these bigrams overlap: “sense and” is one token, while “and sensibility” is another.

5.1.1 Counting and filtering n-grams

Our usual tidy tools apply equally well to n-gram analysis. We can examine the most common bigrams using dplyr’s `count()`:

```
austen_bigrams %>%
  count(bigram, sort = TRUE)
```

```
## # A tibble: 211,237 × 2
##      bigram     n
##       <chr> <int>
## 1    of the  3017
## 2     to be  2787
## 3    in the  2368
## 4    it was  1781
## 5      i am  1545
## 6   she had  1472
## 7    of her  1445
## 8    to the  1387
## 9   she was  1377
## 10 had been  1299
## # ... with 211,227 more rows
```

As one might expect, a lot of the most common bigrams are pairs of common (uninteresting) words, such as `of the` and `to be`: what we call “stop-words” (see Chapter). This is a useful time to use tidyr’s `separate()`, which splits a column into multiple based on a delimiter. This lets us separate it into two columns, “word1” and “word2”, at which we can remove cases where either is a stop-word.

```
library(tidyr)

bigrams_separated <- austen_bigrams %>%
  separate(bigram, c("word1", "word2"), sep = " ")

bigrams_filtered <- bigrams_separated %>%
  filter(!word1 %in% stop_words$word) %>%
  filter(!word2 %in% stop_words$word)

# New bigram counts:
bigram_counts <- bigrams_filtered %>%
  count(word1, word2, sort = TRUE)

bigram_counts
```

```
## Source: local data frame [33,421 x 3]
## Groups: word1 [6,711]
##
##      word1     word2     n
##      <chr>     <chr> <int>
## 1      sir    thomas   287
## 2     miss  crawford   215
## 3  captain wentworth   170
## 4     miss woodhouse   162
## 5    frank churchill   132
## 6     lady   russell   118
## 7     lady   bertram   114
## 8      sir    walter   113
## 9     miss   fairfax   109
## 10 colonel   brandon   108
## # ... with 33,411 more rows
```

We can see that names (whether first and last or with a salutation) are the most common pairs in Jane Austen books.

In other analyses, we may want to work with the recombined words. tidyr's `unite()` function is the inverse of `separate()`, and lets us recombine the columns into one. Thus, "separate/filter/count/unite" let us find the most common bigrams not containing stop-words.

```
bigrams_united <- bigrams_filtered %>%
  unite(bigram, word1, word2, sep = " ")

bigrams_united
```

```
## # A tibble: 44,784 × 2
##                   book                   bigram
## *               <fctr>                    <chr>
## 1  Sense & Sensibility              jane austen
## 2  Sense & Sensibility              austen 1811
## 3  Sense & Sensibility             1811 chapter
## 4  Sense & Sensibility                chapter 1
## 5  Sense & Sensibility             norland park
## 6  Sense & Sensibility surrounding acquaintance
## 7  Sense & Sensibility               late owner
## 8  Sense & Sensibility             advanced age
## 9  Sense & Sensibility       constant companion
## 10 Sense & Sensibility             happened ten
## # ... with 44,774 more rows
```

In other analyses you may be interested in the most common trigrams, which are consecutive sequences of 3 words. We can find this by setting `n = 3`:

```
austen_books() %>%
  unnest_tokens(trigram, text, token = "ngrams", n = 3) %>%
  separate(trigram, c("word1", "word2", "word3"), sep = " ") %>%
  filter(!word1 %in% stop_words$word,
         !word2 %in% stop_words$word,
         !word3 %in% stop_words$word) %>%
  count(word1, word2, word3, sort = TRUE)
```

```
## Source: local data frame [8,757 x 4]
## Groups: word1, word2 [7,462]
##
##         word1     word2     word3     n
##         <chr>     <chr>     <chr> <int>
## 1        dear      miss woodhouse    23
## 2        miss        de    bourgh    18
## 3        lady catherine        de    14
## 4   catherine        de    bourgh    13
## 5        poor      miss    taylor    11
## 6         sir    walter    elliot    11
## 7         ten  thousand    pounds    11
## 8        dear       sir    thomas    10
## 9      twenty  thousand    pounds     8
## 10    replied      miss  crawford     7
## # ... with 8,747 more rows
```

5.1.2 Analyzing bigrams

This one-row-per-bigram format is helpful for exploratory analyses of the text. As a simple example, we might be interested in the most common “streets” mentioned in each book:

```
bigrams_filtered %>%
  filter(word2 == "street") %>%
  count(book, word1, sort = TRUE)
```

```
## Source: local data frame [34 x 3]
## Groups: book [6]
##
##                   book       word1     n
##                 <fctr>       <chr> <int>
## 1  Sense & Sensibility    berkeley    16
## 2  Sense & Sensibility      harley    16
## 3     Northanger Abbey    pulteney    14
## 4     Northanger Abbey      milsom    11
## 5       Mansfield Park     wimpole    10
## 6    Pride & Prejudice gracechurch     9
## 7  Sense & Sensibility     conduit     6
## 8  Sense & Sensibility        bond     5
## 9           Persuasion      milsom     5
## 10          Persuasion      rivers     4
## # ... with 24 more rows
```

A bigram can also be treated as a term in a document in the same way that we treated individual words. For example, we can look at the TF-IDF (Chapter 4) of bigrams across Austen novels. These TF-IDF values can be visualized within each book, just as we did for words (Figure).

```
bigram_tf_idf <- bigrams_united %>%
  count(book, bigram) %>%
  bind_tf_idf(bigram, book, n) %>%
  arrange(desc(tf_idf))

bigram_tf_idf
```

```
## Source: local data frame [36,217 x 6]
## Groups: book [6]
##
##                    book            bigram     n         tf      idf     tf_idf
##                  <fctr>             <chr> <int>      <dbl>    <dbl>      <dbl>
## 1            Persuasion captain wentworth   170 0.02985599 1.791759 0.05349475
## 2        Mansfield Park        sir thomas   287 0.02873160 1.791759 0.05148012
## 3        Mansfield Park     miss crawford   215 0.02152368 1.791759 0.03856525
## 4            Persuasion      lady russell   118 0.02072357 1.791759 0.03713165
## 5            Persuasion        sir walter   113 0.01984545 1.791759 0.03555828
## 6                  Emma    miss woodhouse   162 0.01700966 1.791759 0.03047722
## 7      Northanger Abbey       miss tilney    82 0.01594400 1.791759 0.02856782
## 8   Sense & Sensibility   colonel brandon   108 0.01502086 1.791759 0.02691377
## 9                  Emma   frank churchill   132 0.01385972 1.791759 0.02483329
## 10    Pride & Prejudice    lady catherine   100 0.01380453 1.791759 0.02473439
## # ... with 36,207 more rows
```

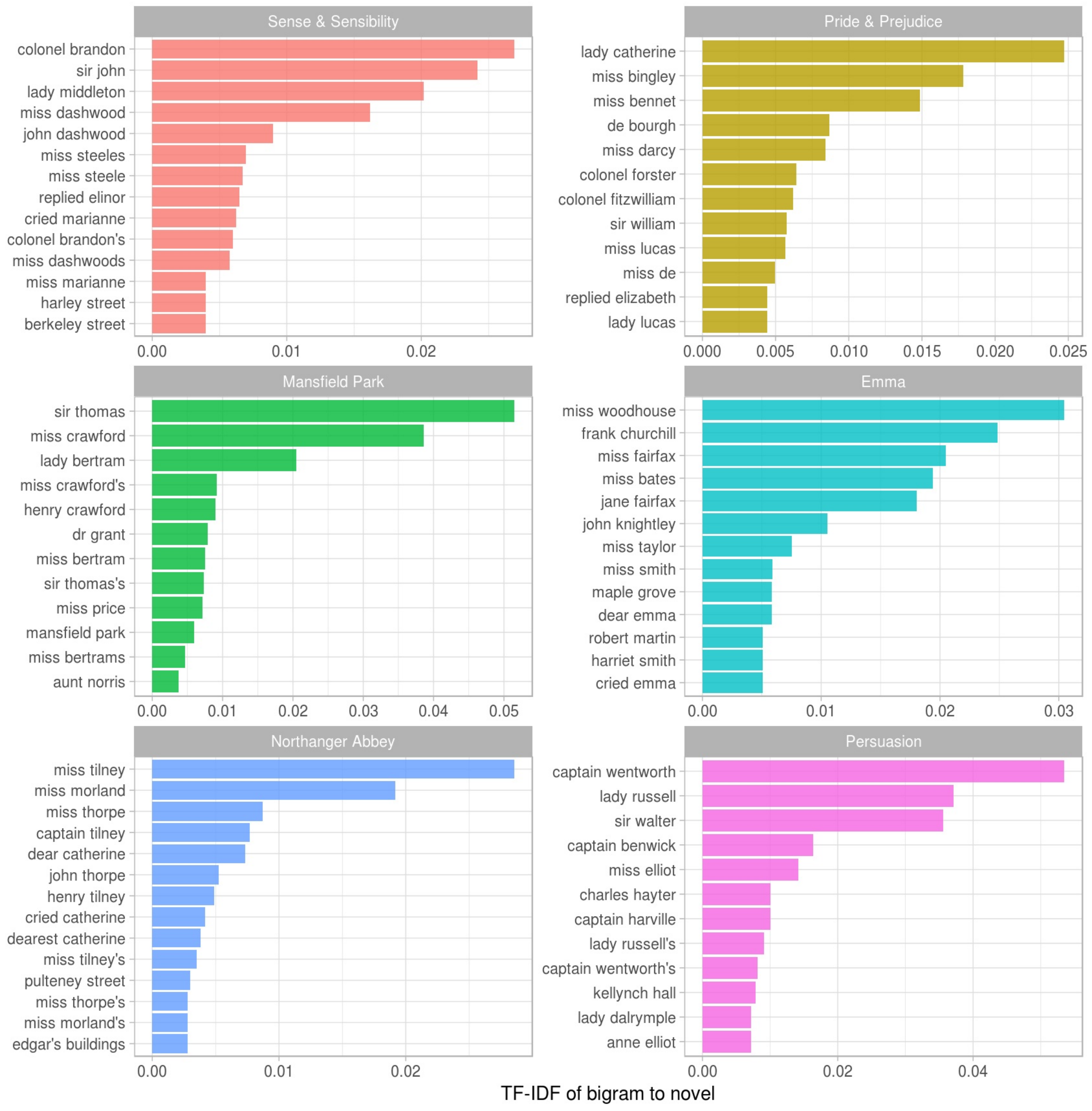


Figure 5.1: The 12 bigrams with the highest TF-IDF from each Jane Austen novel.

Much as we discovered in Chapter , the units that distinguish each Austen book are almost exclusively names. We also notice some pairings of a common verb and a name, such as "replied elizabeth" in Pride & Prejudice, or "cried emma" in Emma.

There are advantages and disadvantages to examining the TF-IDF of bigrams rather than individual words. Pairs of consecutive words might capture structure that isn't present when one is just counting single words, and may provide context that makes tokens more understandable (for example, "pulteney street", in Northanger Abbey, is more informative than "pulteney"). However, the per-bigram counts are also *sparser*: a typical two-word pair is rarer than either of its component words. Thus, bigrams can be especially useful when you have a very large text dataset.

5.1.3 Using bigrams to provide context in sentiment analysis

Our sentiment analysis approch in Chapter simply counted the appearance of positive or negative words, according to a reference lexicon. One of the problems with this approach is that a word's context can matter nearly as much as its presence. For example, the words "happy" and "like" will be counted as positive, even in a sentence like "I'm not **happy** and I don't **like** it!"

Now that we have the data organized into bigrams, it's easy to tell how often words are preceded by a word like "not":

```
bigrams_separated %>%
  filter(word1 == "not") %>%
  count(word1, word2, sort = TRUE)
```

```
## Source: local data frame [1,246 x 3]
## Groups: word1 [1]
##
##    word1 word2     n
##    <chr> <chr> <int>
## 1    not    be   610
## 2    not    to   355
## 3    not  have   327
## 4    not  know   252
## 5    not     a   189
## 6    not think   176
## 7    not  been   160
## 8    not   the   147
## 9    not    at   129
## 10   not    in   118
## # ... with 1,236 more rows
```

By performing sentiment analysis on the bigram data, we can examine how often sentiment-associated words are preceded by "not" or other negating words. We could use this to ignore or even reverse their contribution to the sentiment score.

Let's use the AFINN lexicon for sentiment analysis, which you may recall gives a numeric sentiment score for each word, with positive or negative numbers indicating the direction of the sentiment.

```
AFINN <- get_sentiments("afinn")

AFINN
```

```
## # A tibble: 2,476 × 2
##          word score
##         <chr> <int>
## 1     abandon    -2
## 2   abandoned    -2
## 3    abandons    -2
## 4    abducted    -2
## 5   abduction    -2
## 6  abductions    -2
## 7       abhor    -3
## 8    abhorred    -3
## 9   abhorrent    -3
## 10     abhors    -3
## # ... with 2,466 more rows
```

We can then examine the most frequent words that were preceded by “not” and were associated with a sentiment.

```
not_words <- bigrams_separated %>%
  filter(word1 == "not") %>%
  inner_join(AFINN, by = c(word2 = "word")) %>%
  count(word2, score, sort = TRUE) %>%
  ungroup()

not_words
```

```
## # A tibble: 245 × 3
##      word2 score     n
##      <chr> <int> <int>
## 1     like     2    99
## 2     help     2    82
## 3     want     1    45
## 4     wish     1    39
## 5    allow     1    36
## 6     care     2    23
## 7    sorry    -1    21
## 8    leave    -1    18
## 9  pretend    -1    18
## 10   worth     2    17
## # ... with 235 more rows
```

For example, the most common sentiment-associated word to follow “not” was “like”, which would normally have a (positive) score of 2.

It's worth asking which words contributed the most in the "wrong" direction. To compute that, we can multiply their score by the number of times they appear (so that a word with a score of +3 occurring 10 times has as much impact as a word with a sentiment score of +1 occurring 30 times). We visualize the result with a bar plot (Figure).

```
not_words %>%
  mutate(contribution = n * score) %>%
  arrange(desc(abs(contribution))) %>%
  head(20) %>%
  mutate(word2 = reorder(word2, contribution)) %>%
  ggplot(aes(word2, n * score, fill = n * score > 0)) +
  geom_bar(stat = "identity", show.legend = FALSE) +
  xlab("Words preceded by \"not\"") +
  ylab("Sentiment score * number of occurrences") +
  coord_flip()
```

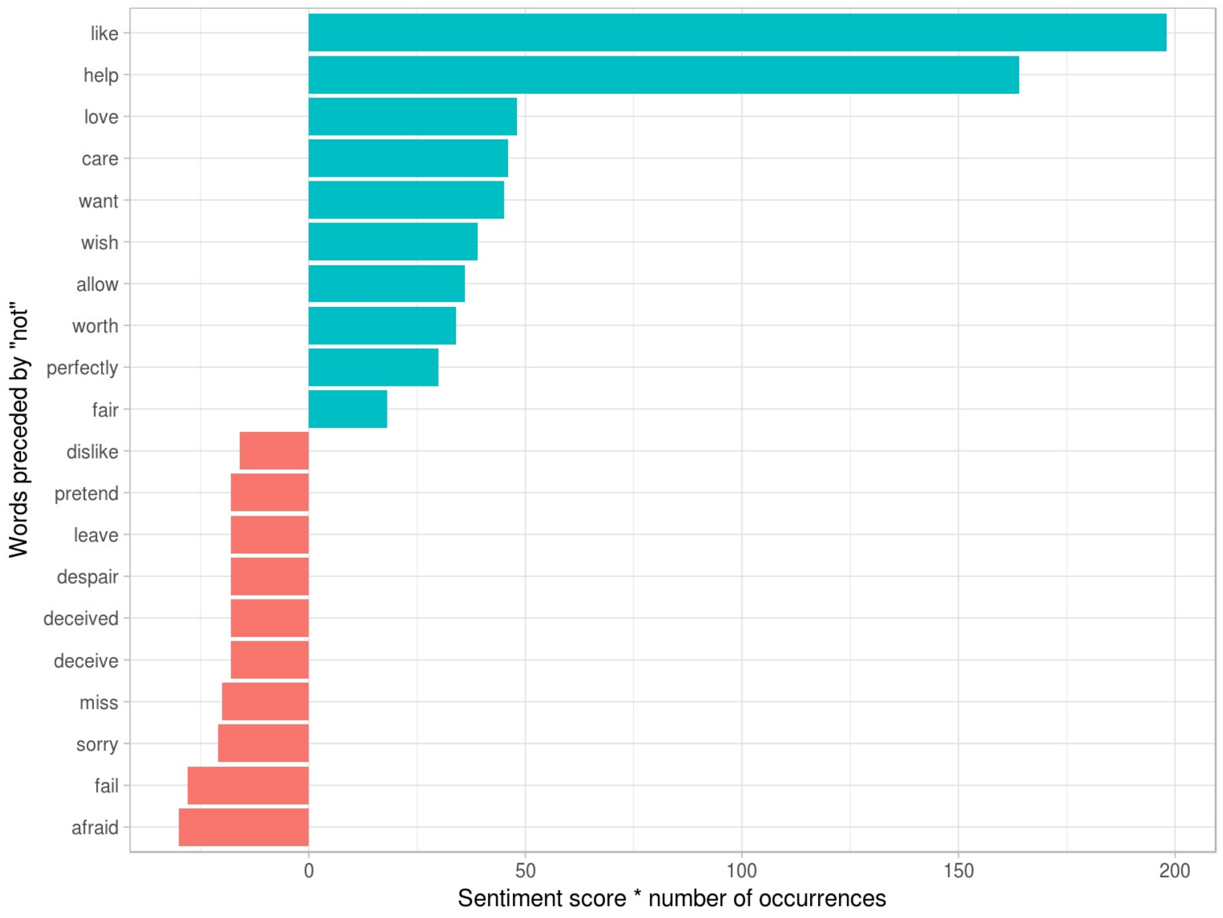


Figure 5.2: The 20 words followed by 'not' that had the greatest contribution to sentiment scores, in either a positive or negative direction.

The bigrams "not like" and "not help" were overwhelmingly the largest causes of misidentification, making the text seem much more positive than it is. But we can see phrases like "not afraid" and "not fail" sometimes suggest text is more negative than it is.

"Not" isn't the only term that provides some context for the following word. We could pick four common words (or more) that negate the subsequent term, and use the same joining and counting approach to examine all of them at once.

```r
negation_words <- c("not", "no", "never", "without")

negated_words <- bigrams_separated %>%
  filter(word1 %in% negation_words) %>%
  inner_join(AFINN, by = c(word2 = "word")) %>%
  count(word1, word2, score, sort = TRUE) %>%
  ungroup()
```

We could then visualize what the most common words to follow each particular negation are (Figure). While "not like" and "not help" are still the two most common examples, we can also see pairings such as "no great" and "never loved." These are just a few examples of how finding consecutive words can give context to text mining methods.

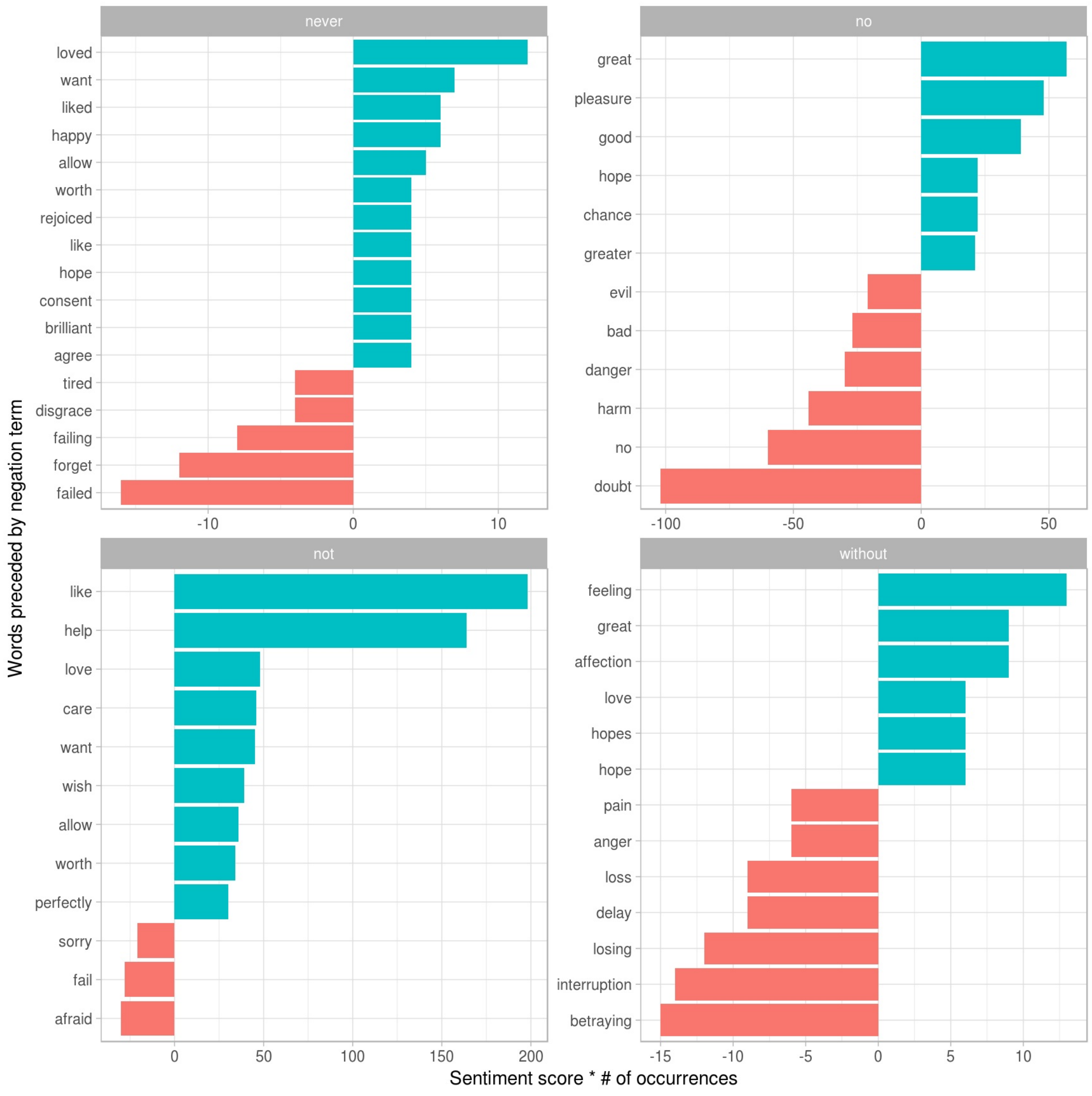


Figure 5.3: The most common positive or negative words to follow negations such as 'never', 'no', 'not', and 'without'

5.1.4 Visualizing a network of bigrams with igraph

We may be interested in visualizing all of the relationships among words simultaneously, rather than just the top few at a time. As one common visualization, we can arrange the words into a network, or “graph.” Here we’ll be referring to a “graph” not in the sense of a visualization, but as a combination of connected nodes. A graph can be constructed from a tidy object since it has three variables:

- **from**: the node an edge is coming from
- **to**: the node an edge is going towards
- **weight**: A numeric value associated with each edge

The igraph package has many powerful functions for manipulating and analyzing networks. One way to create an igraph object from tidy data is the `graph_from_data_frame()` function, which takes a data frame that starts with “from”, “to”, and “node”, columns, in that order.

```
library(igraph)

# original counts
bigram_counts
```

```
## Source: local data frame [33,421 x 3]
## Groups: word1 [6,711]
##
##      word1     word2     n
##      <chr>     <chr> <int>
## 1      sir    thomas   287
## 2     miss  crawford   215
## 3  captain wentworth   170
## 4     miss woodhouse   162
## 5    frank churchill   132
## 6     lady   russell   118
## 7     lady   bertram   114
## 8      sir    walter   113
## 9     miss   fairfax   109
## 10 colonel   brandon   108
## # ... with 33,411 more rows
```

```
# filter for only relatively common combinations
bigram_graph <- bigram_counts %>%
  filter(n > 20) %>%
  graph_from_data_frame()

bigram_graph
```

```
## IGRAPH DN-- 91 77 --
## + attr: name (v/c), n (e/n)
## + edges (vertex names):
##  [1] sir      ->thomas     miss     ->crawford   captain ->wentworth  miss     ->woodhouse
##  [5] frank    ->churchill  lady     ->russell    lady    ->bertram    sir      ->walter
##  [9] miss     ->fairfax    colonel ->brandon    miss    ->bates      lady     ->catherine
## [13] sir      ->john       jane     ->fairfax    miss    ->tilney     lady     ->middleton
## [17] miss     ->bingley    thousand->pounds     miss    ->dashwood   miss     ->bennet
## [21] john     ->knightley  miss     ->morland    captain ->benwick    dear     ->miss
## [25] miss     ->smith      miss     ->crawford's henry   ->crawford   miss     ->elliot
## [29] dr       ->grant      miss     ->bertram    sir     ->thomas's   ten      ->minutes
## + ... omitted several edges
```

igraph has plotting functions built in, but they're not what the package is designed to do, so many other packages have developed visualization methods for graph objects. We recommend the ggraph package, because it implements these visualizations in terms of the grammar of graphics, which we are already familiar with from ggplot2.

We can convert an igraph object into a ggraph with the `ggraph` function, after which we add layers to it, much like layers are added in ggplot2. For example, for a basic graph we need to add three layers: nodes, edges, and text

```r
library(ggraph)
set.seed(2017)

ggraph(bigram_graph, layout = "fr") +
  geom_edge_link() +
  geom_node_point() +
  geom_node_text(aes(label = name), vjust = 1, hjust = 1)
```

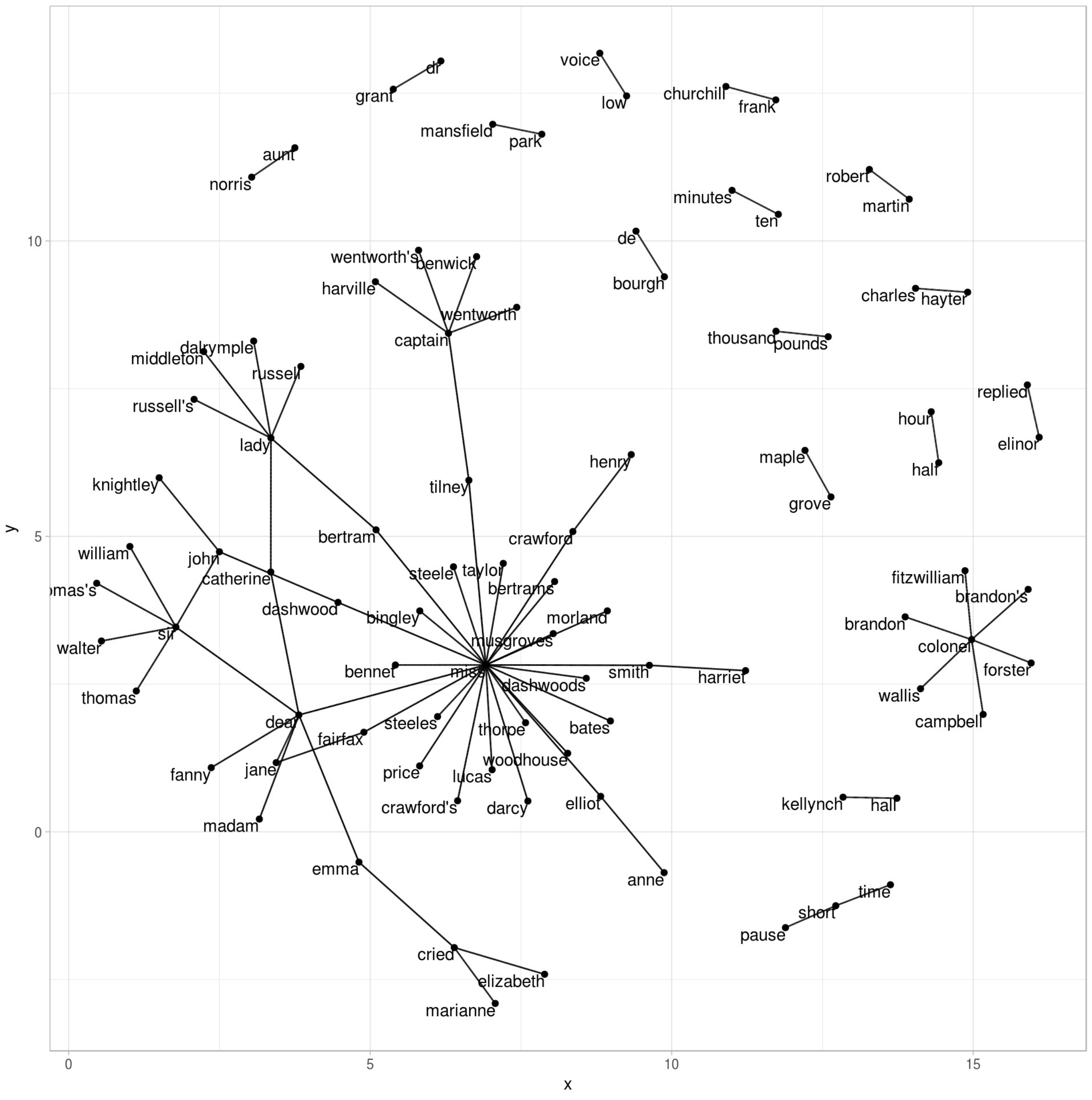


Figure 5.4: Common bigrams in Pride and Prejudice, showing those that occurred more than 20 times and where neither word was a stop-word.

In Figure , we can visualize some details of the text structure. For example, we see that salutations such as “miss”, “lady”, “sir”, “and”colonel" form common centers of nodes, which are often followed by names. We also see pairs or triplets along the outside that form common short phrases (“half hour”, “thousand pounds”, or “short time/pause”).

We conclude with a few polishing operations to make a better looking graph (Figure):

- We add the `edge_alpha` aesthetic to the link layer to make links transparent based on how common or rare the bigram is
- We add directionality with an arrow, constructed using `grid::arrow()`
- We tinker with the options to the node layer to make the nodes more attractive (larger, blue points)
- We add a theme that’s useful for plotting networks, `theme_void()`

```r
set.seed(2016)

a <- grid::arrow(type = "closed", length = unit(.15, "inches"))

ggraph(bigram_graph, layout = "fr") +
  geom_edge_link(aes(edge_alpha = n), show.legend = FALSE, arrow = a) +
  geom_node_point(color = "lightblue", size = 5) +
  geom_node_text(aes(label = name), vjust = 1, hjust = 1) +
  theme_void()
```

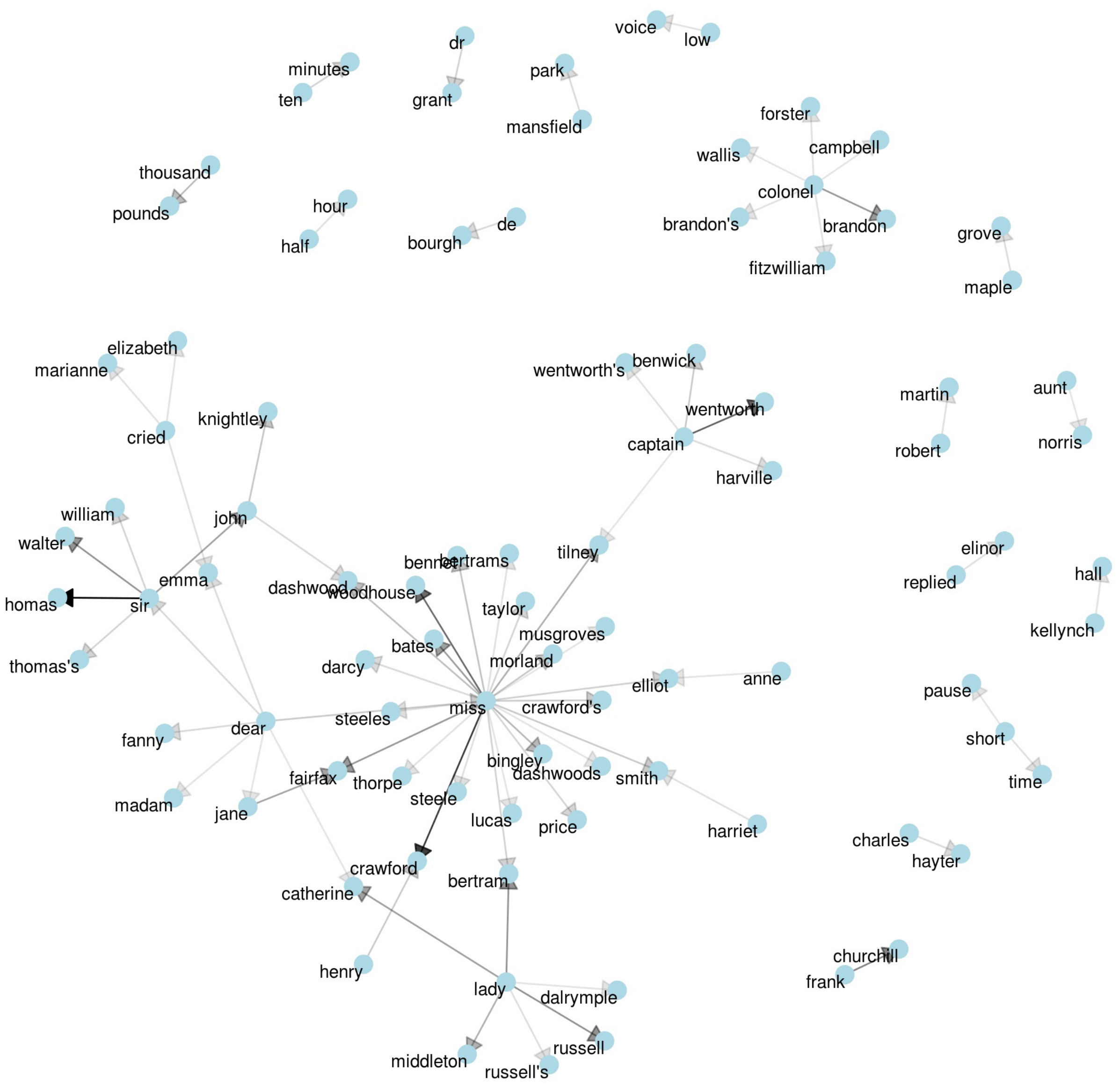


Figure 5.5: Common bigrams in Pride and Prejudice, with some polishing.

It may take a some experimentation with ggraph to get your networks into a presentable format like this, but the network structure is useful and flexible way to visualize relational tidy data.

Note that this is a visualization of a **Markov chain**, a common model in text processing. In a Markov chain, each choice of word depends only on the previous word. In this case, a random generator following this model might spit out “dear”, then “sir”, then “william/walter/thomas/thomas’s”, by following each word to the most common words that follow it. To make the visualization interpretable, we chose to show only the most common word to word connections, but one could imagine an enormous graph representing all connections that occur in the text.

5.1.5 Visualizing bigrams in other texts

We went to a good amount of work in cleaning and visualizing bigrams on a text dataset, so let's collect it into a function so that we can do it on other text datasets easily.

```r
count_bigrams <- function(dataset) {
  dataset %>%
    unnest_tokens(bigram, text, token = "ngrams", n = 2) %>%
    separate(bigram, c("word1", "word2"), sep = " ") %>%
    filter(!word1 %in% stop_words$word,
           !word2 %in% stop_words$word) %>%
    count(word1, word2, sort = TRUE)
}

visualize_bigrams <- function(bigrams) {
  set.seed(2016)
  a <- grid::arrow(type = "closed", length = unit(.15, "inches"))

  bigrams %>%
    graph_from_data_frame() %>%
    ggraph(layout = "fr") +
    geom_edge_link(aes(edge_alpha = n), show.legend = FALSE, arrow = a) +
    geom_node_point(color = "lightblue", size = 5) +
    geom_node_text(aes(label = name), vjust = 1, hjust = 1) +
    theme_void()
}
```

At this point, we could visualize bigrams in other works, such as the King James Version of the Bible:

```r
# The King James version is book 10 on Project Gutenberg:
library(gutenbergr)
kjv <- gutenberg_download(10)
```

```r
library(stringr)

kjv_bigrams <- kjv %>%
  count_bigrams()

# filter out rare combinations, as well as digits
kjv_bigrams %>%
  filter(n > 40,
         !str_detect(word1, "\\d"),
         !str_detect(word2, "\\d")) %>%
  visualize_bigrams()
```

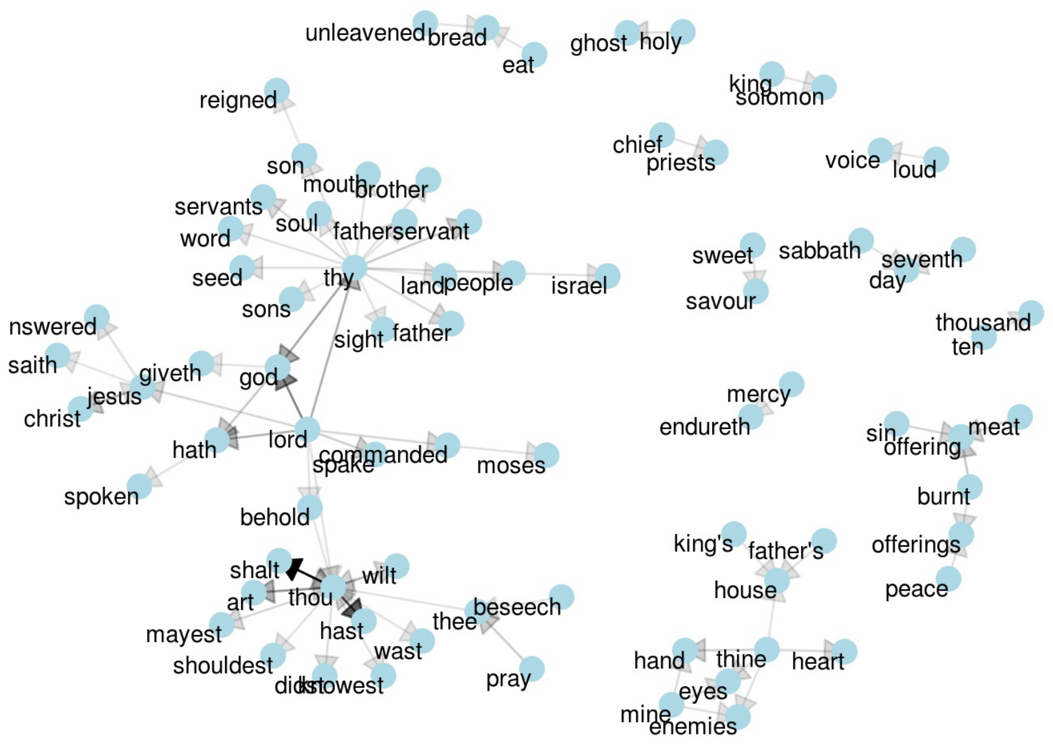


Figure 5.6: Directed graph of common bigrams in the King James Bible, showing those that occurred more than 40 times.

Figure thus lays out a common "blueprint" of language within the Bible, particularly focused around "thy" and "thou" (which could probably be considered stopwords!) You can use the gutenbergr package and these `count_bigrams` / `visualize_bigrams` functions to visualize bigrams in other classic books you're interested in.

5.2 Counting and correlating pairs of words with the widyr package

Tokenizing by n-gram is a useful way to explore pairs of adjacent words. However, we may also be interested in words that tend to co-occur within particular documents or particular chapters, even if they don't occur next to each other.

Tidy data is a useful structure for comparing between variables or grouping by rows, but it can be challenging to compare between rows: for example, to count the number of times that two words appear within the same document, or to see how correlated they are. Most operations for finding pairwise counts or correlations need to turn the data into a wide matrix first.

```
knitr::include_graphics("images/widyr.jpg")
```

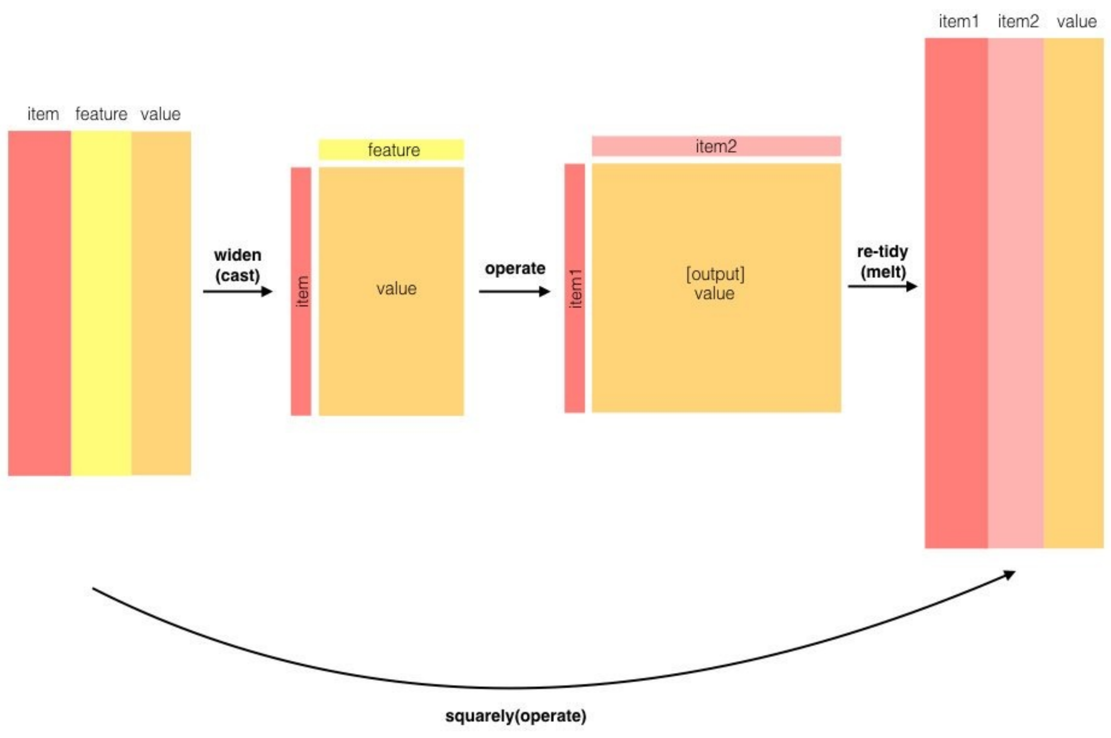


Figure 5.7: The philosophy behind the widyr package, which can perform operations such as counting and correlating on pairs of values in a tidy dataset. The widyr package first 'casts' a tidy dataset into a wide matrix, performs an operation such as a correlation on it, then re-tidies the result.

We'll examine some of the ways tidy text can be turned into a wide matrix in Chapter 6, but in this case it isn't necessary. The widyr package makes operations such as computing counts and correlations easy, by simplifying the pattern of "widen data, perform an operation, then re-tidy data" (Figure). We'll focus on a set of functions that make pairwise comparisons between groups of observations (for example, between documents, or sections of text).

5.2.1 Counting and correlating among sections

Consider the book "Pride and Prejudice" divided into 10-line sections, as we did for sentiment analysis in Chapter 3. We may be interested in what words tend to appear within the same section.

```
austen_section_words <- austen_books() %>%
  filter(book == "Pride & Prejudice") %>%
  mutate(section = row_number() %/% 10) %>%
  filter(section > 0) %>%
  unnest_tokens(word, text) %>%
  filter(!word %in% stop_words$word)

austen_section_words
```

```
## # A tibble: 37,240 × 3
##                  book section        word
##                <fctr>   <dbl>       <chr>
## 1  Pride & Prejudice       1        truth
## 2  Pride & Prejudice       1  universally
## 3  Pride & Prejudice       1 acknowledged
## 4  Pride & Prejudice       1       single
## 5  Pride & Prejudice       1   possession
## 6  Pride & Prejudice       1      fortune
## 7  Pride & Prejudice       1         wife
## 8  Pride & Prejudice       1     feelings
## 9  Pride & Prejudice       1        views
## 10 Pride & Prejudice       1     entering
## # ... with 37,230 more rows
```

One useful function from widyr is the `pairwise_count()` function. The prefix `pairwise_` means it will result in one row for each pair of words in the `word` variable. This lets us count common pairs of words co-appearing within the same section:

```
library(widyr)

# count words co-occuring within sections
word_pairs <- austen_section_words %>%
  pairwise_count(word, section, sort = TRUE)

word_pairs
```

```
## # A tibble: 796,008 × 3
##         item1     item2     n
##         <chr>     <chr> <dbl>
## 1      darcy elizabeth   144
## 2  elizabeth     darcy   144
## 3       miss elizabeth   110
## 4  elizabeth      miss   110
## 5  elizabeth      jane   106
## 6       jane elizabeth   106
## 7       miss     darcy    92
## 8      darcy      miss    92
## 9  elizabeth   bingley    91
## 10   bingley elizabeth    91
## # ... with 795,998 more rows
```

Notice that while the input had one row for each pair of a document and a word, the output has one row for each pair of words. This is also a tidy format, but of a very different structure that we can use to answer new questions.

For example, we can see that the most common pair of words in a section is “Elizabeth” and “Darcy” (the two main characters). We can easily find the words that most often occur with Darcy:

```
word_pairs %>%
  filter(item1 == "darcy")
```

```
## # A tibble: 2,930 × 3
##    item1     item2     n
##    <chr>     <chr> <dbl>
## 1  darcy elizabeth   144
## 2  darcy      miss    92
## 3  darcy   bingley    86
## 4  darcy      jane    46
## 5  darcy    bennet    45
## 6  darcy    sister    45
## 7  darcy      time    41
## 8  darcy      lady    38
## 9  darcy    friend    37
## 10 darcy   wickham    37
## # ... with 2,920 more rows
```

5.2.2 Pairwise correlation

Pairs like “Elizabeth” and “Darcy” are the most common co-occurring words, but that’s not particularly meaningful since *they’re also the most common individual words*. We may instead want to examine **correlation** among words, which indicates how often they appear together relative to how often they appear separately.

In particular, here we'll focus on the phi coefficient, a common measure for binary correlation. The focus of the phi coefficient is how much more likely it is that either **both** word X and Y appear, or **neither** do, than that one appears without the other.

Consider the following table:

	Has word Y	No word Y	Total
Has word X	n_{11}	n_{10}	$n_{1\cdot}$
No word X	n_{01}	n_{00}	$n_{0\cdot}$
Total	$n_{\cdot 1}$	$n_{\cdot 0}$	n

For example, that n_{11} represents the number of documents where both word X and word Y appear, n_{00} the number where neither appears, and n_{10} and n_{01} the cases where one appears without the other. In terms of this table, the phi coefficient is:

$$\phi=\frac{n_{11}n_{00}-n_{10}n_{01}}{\sqrt{n_{1\cdot}n_{0\cdot}n_{\cdot0}n_{\cdot1}}}$$

(The phi coefficient is equivalent to the Pearson correlation, which you may have heard of elsewhere, when it is applied to binary data).

The `pairwise_cor()` function in widyr lets us find the correlation correlation between words based on how often they appear in the same section. Its syntax is similar to `pairwise_count()`.

```r
library(widyr)

# We need to filter for at least relatively common words first
word_cors <- austen_section_words %>%
  group_by(word) %>%
  filter(n() >= 20) %>%
  pairwise_cor(word, section, sort = TRUE)

word_cors
```

```
## # A tibble: 154,842 × 3
##        item1     item2 correlation
##        <chr>     <chr>       <dbl>
## 1     bourgh        de   0.9508501
## 2         de    bourgh   0.9508501
## 3     pounds  thousand   0.7005808
## 4   thousand    pounds   0.7005808
## 5    william       sir   0.6644719
## 6        sir   william   0.6644719
## 7  catherine      lady   0.6633048
## 8       lady catherine   0.6633048
## 9    forster   colonel   0.6220950
## 10   colonel   forster   0.6220950
## # ... with 154,832 more rows
```

This output format is helpful for exploration. For example, we could find the words most correlated with a word like “pounds” using a `filter` operation.

```
word_cors %>%
  filter(item1 == "pounds")
```

```
## # A tibble: 393 × 3
##      item1     item2 correlation
##      <chr>     <chr>       <dbl>
## 1  pounds  thousand  0.70058081
## 2  pounds       ten  0.23057580
## 3  pounds   fortune  0.16386264
## 4  pounds   settled  0.14946049
## 5  pounds wickham's  0.14152401
## 6  pounds  children  0.12900011
## 7  pounds  mother's  0.11905928
## 8  pounds  believed  0.09321518
## 9  pounds    estate  0.08896876
## 10 pounds     ready  0.08597038
## # ... with 383 more rows
```

This would let us examine the most-correlated words with any selection of words. For example, we could pick four words and visualize the words most associated with them (Figure).

```
word_cors %>%
  filter(item1 %in% c("elizabeth", "pounds", "married", "pride")) %>%
  group_by(item1) %>%
  top_n(6) %>%
  mutate(item2 = reorder(item2, correlation)) %>%
  ggplot(aes(item2, correlation)) +
  geom_bar(stat = "identity") +
  facet_wrap(~ item1, scales = "free") +
  coord_flip()
```

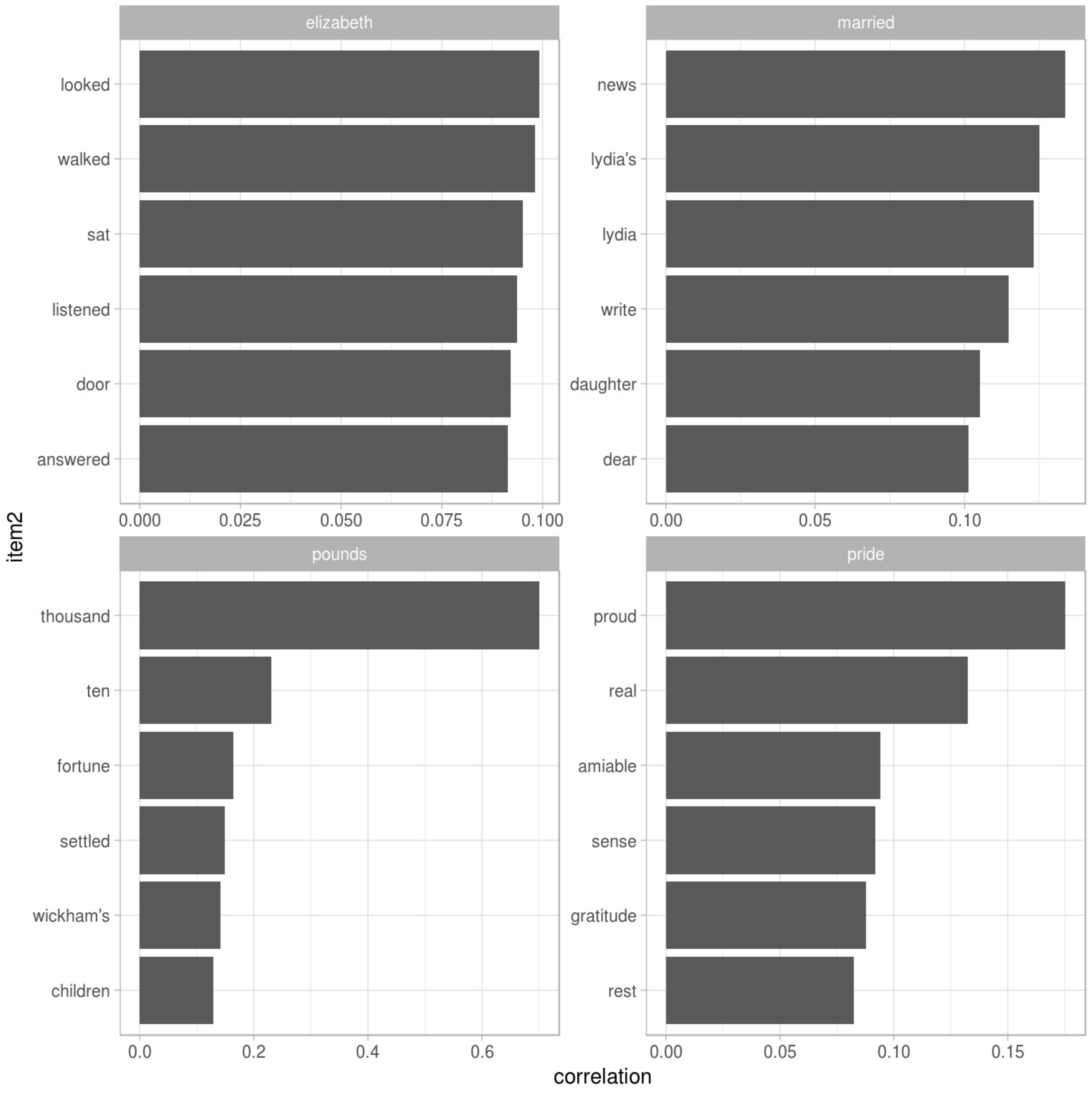


Figure 5.8: Words from Pride and Prejudice that were most correlated with 'elizabeth', 'pounds', 'married', and 'pride'.

Just as we used ggraph to visualize bigrams, we can use it to visualize the correlations and clusters of words that were found by the widyr package (Figure).

```r
set.seed(2016)

word_cors %>%
  filter(correlation > .15) %>%
  graph_from_data_frame() %>%
  ggraph(layout = "fr") +
  geom_edge_link(aes(edge_alpha = correlation), show.legend = FALSE) +
  geom_node_point(color = "lightblue", size = 5) +
  geom_node_text(aes(label = name), repel = TRUE) +
  theme_void()
```

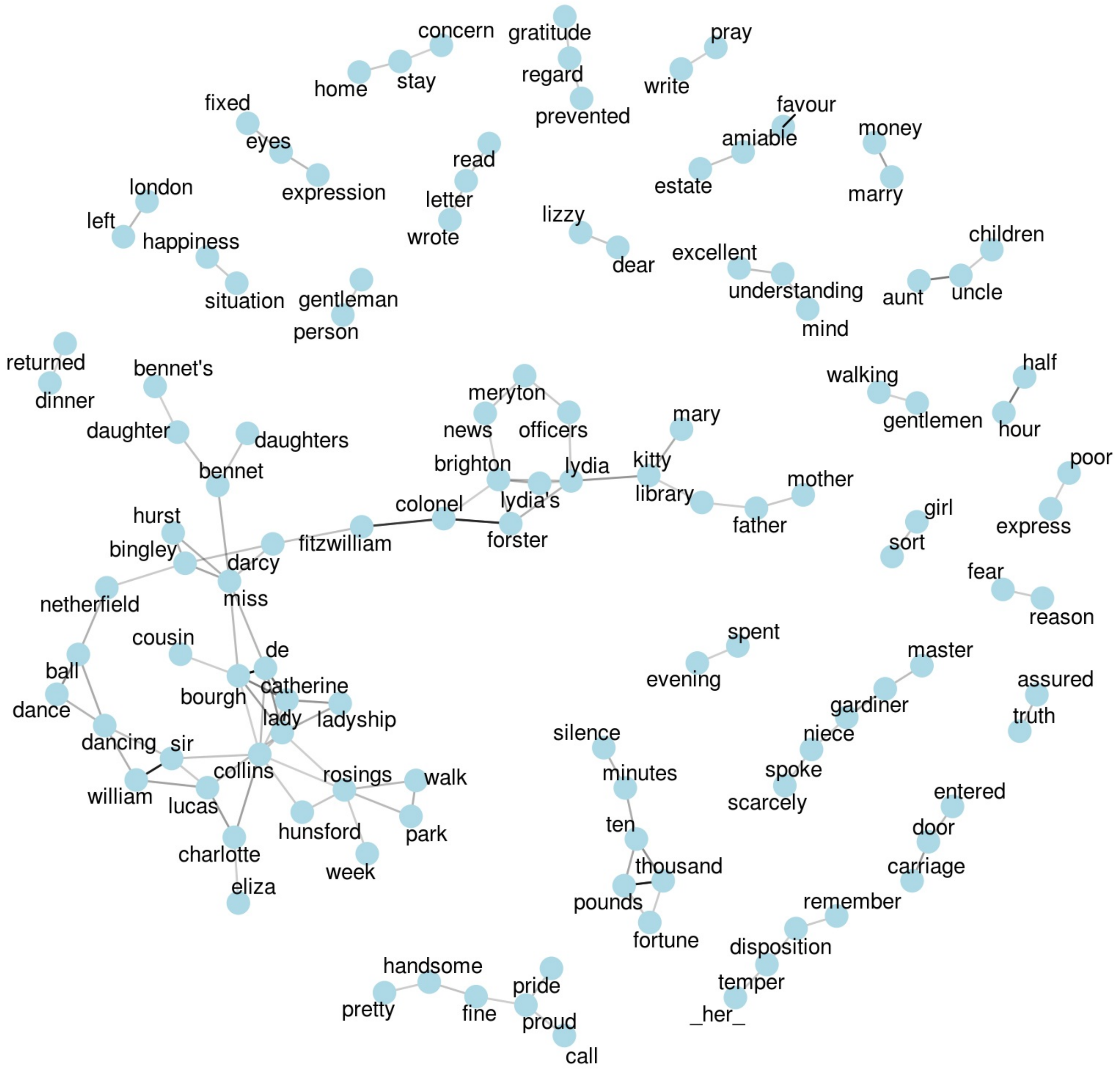


Figure 5.9: Pairs of words in Pride and Prejudice that show at least a .15 correlation of appearing within the same 10-line section.

Note that unlike the bigram analysis, the relationships here are symmetrical, rather than directional (there are no arrows). We can also see that while pairings of names and titles that dominated bigram pairings are common, such as “colonel/fitzwilliam”, we can also see pairings of words that appear close to each other, such as “walk” and “park”, or “dance” and “ball”.

These network visualizations are a flexible tool for exploring relationships, and will play an important role in the case studies in later chapters.

6 Tidying and casting document-term matrices and corpus objects

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 `unnest_tokens` function. 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 non-tidy outputs. These package 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." 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. In particular, we'll examine the process of tidying document-term matrices, as well as casting a tidy data frame into a sparse matrix.

6.1 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, and
- each value (typically) contains the number of appearances of that term in that document.

DTMs are usually implemented as sparse matrices, meaning the vast majority of values are 0. 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.

DTMs are not tidy and cannot be used directly with tidy tools, just as tidy data frames cannot be given to text mining packages. Thus, the tidytext package provides two verbs that convert between the two types of formats.

- `tidy` turns a document-term matrix into a tidy data frame. This verb comes from the broom package (Robinson et al. 2015), which provides tidiers 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).

6.1.1 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 corpus 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 non-tidy object and turns it into a tidy data frame. The tidytext package implements that 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
```

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 non-sparse matrices. Notice that only the non-zero values are included: document 1 includes terms such as "adding" and "adult", but not "aaron" or "abandon", and thus 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 .

```
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 . 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(alpha = 0.8, stat = "identity") +
  ylab("Contribution to sentiment") +
  coord_flip()
```

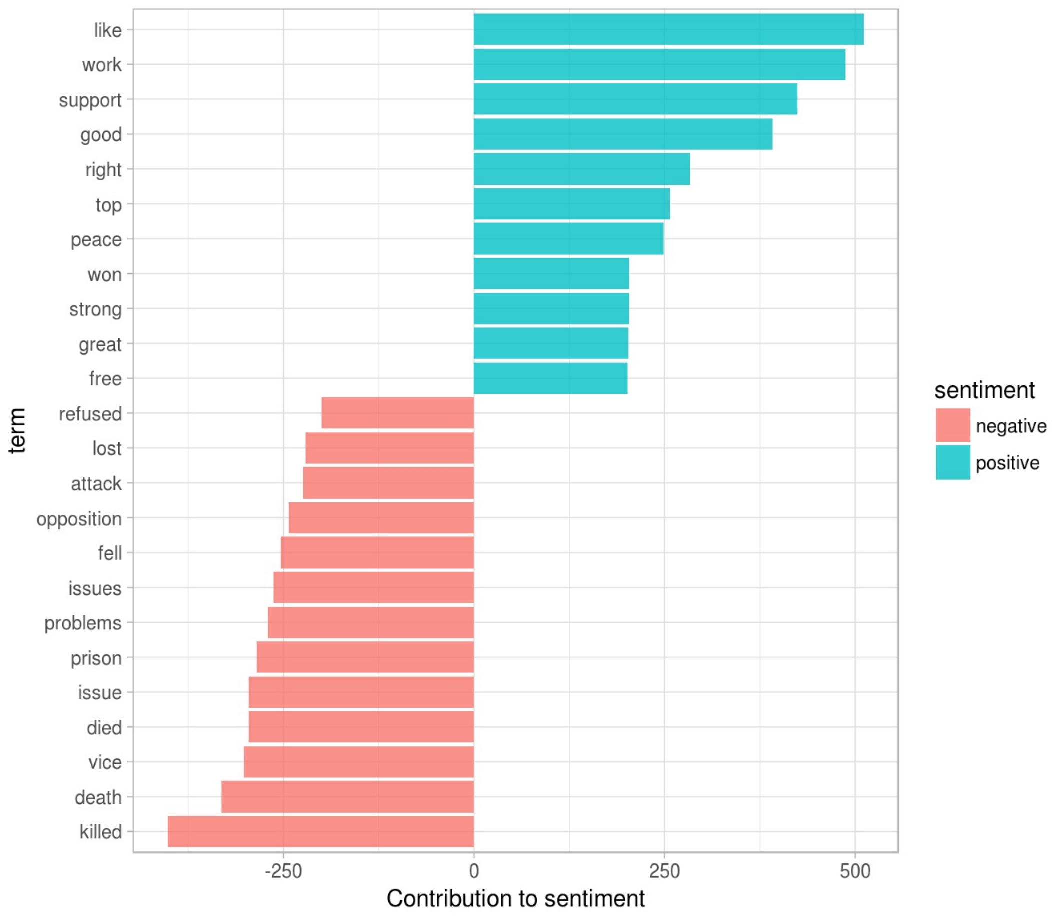


Figure 6.1: Words from AP articles with the greatest contribution to positive or negative sentiments, computed as the product of the word’s AFINN sentiment score and its frequency.

6.1.2 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.

```r
library(methods)

data("inaugCorpus", package = "quanteda")
inaug_dfm <- quanteda::dfm(inaugCorpus)
```

```r
inaug_dfm
```

```
## Document-feature matrix of: 57 documents, 9,174 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:

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

```
## # A tibble: 44,178 × 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 44,168 more rows
```

We may be interested in finding the words most specific to each 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 .

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

inaug_tf_idf
```

```
## # A tibble: 44,178 × 6
##           document        term count          tf      idf     tf_idf
##              <chr>       <chr> <dbl>       <dbl>    <dbl>      <dbl>
## 1  1793-Washington      arrive     1 0.006802721 4.043051 0.02750375
## 2  1793-Washington upbraidings     1 0.006802721 4.043051 0.02750375
## 3  1793-Washington    violated     1 0.006802721 3.349904 0.02278846
## 4  1793-Washington   willingly     1 0.006802721 3.349904 0.02278846
## 5  1793-Washington   incurring     1 0.006802721 3.349904 0.02278846
## 6  1793-Washington    previous     1 0.006802721 2.944439 0.02003020
## 7  1793-Washington   knowingly     1 0.006802721 2.944439 0.02003020
## 8  1793-Washington injunctions     1 0.006802721 2.944439 0.02003020
## 9  1793-Washington   witnesses     1 0.006802721 2.944439 0.02003020
## 10 1793-Washington     besides     1 0.006802721 2.656757 0.01807318
## # ... with 44,168 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 .

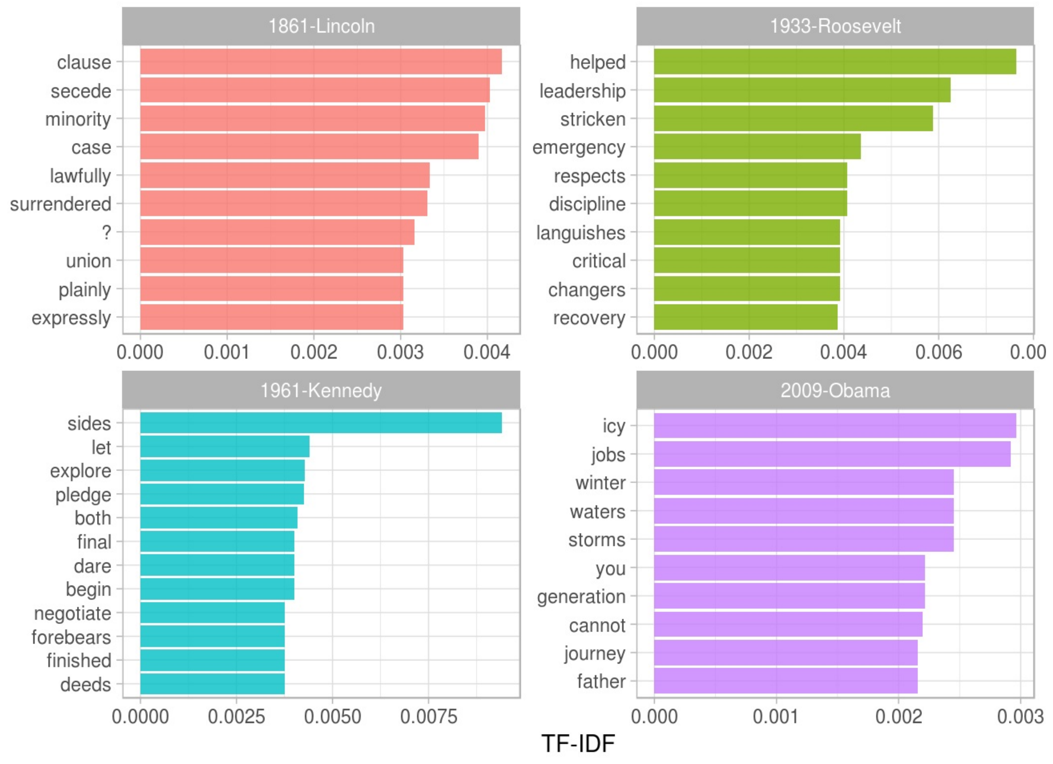


Figure 6.2: The terms with the highest TF-IDF from each of four selected inaugural addresses.

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.

```
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 . 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")
```

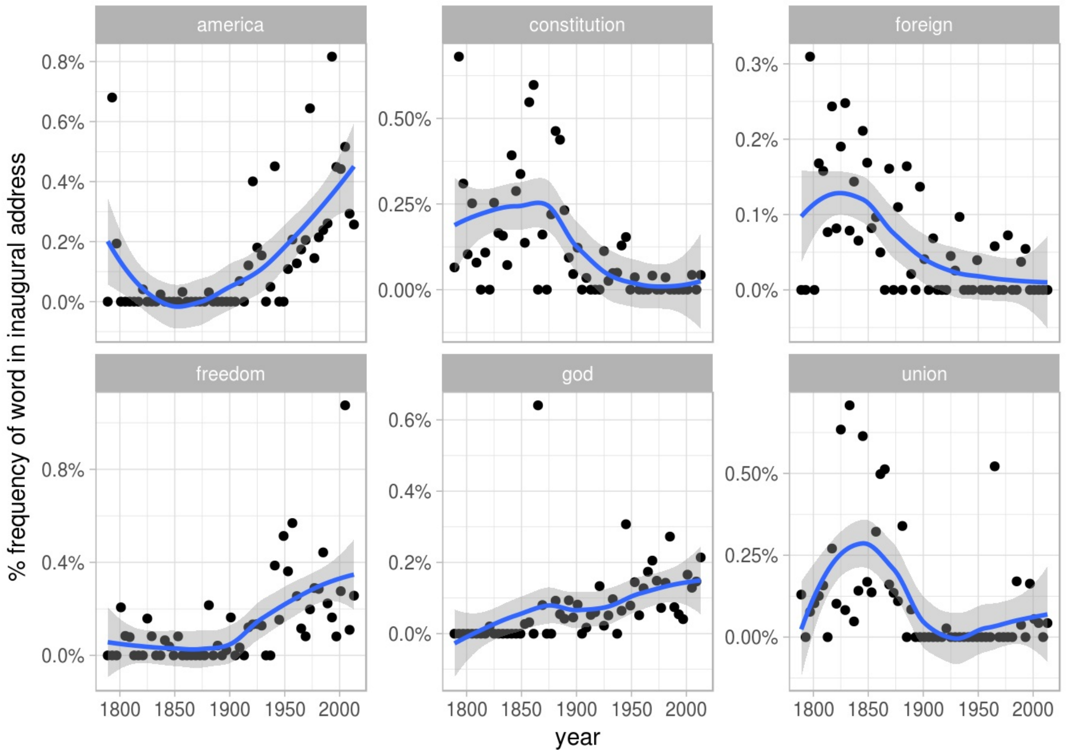


Figure 6.3: Changes in word frequency over time within Presidential inaugural addresses, for four selected terms.

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

6.2 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.

```r
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)
```

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

```r
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:

```r
library(Matrix)

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

class(m)
```

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

```r
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 , we'll examine some examples where a tidy-text dataset has to be converted into a DocumentTermMatrix for processing.

6.3 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 documents). 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 15:18:06
## 2                        <NA> 1987-02-26 15:19:15
## 3                        <NA> 1987-02-26 15:49:56
## 4  By Cal Mankowski, Reuters 1987-02-26 15:51:17
## 5                        <NA> 1987-02-26 16:08:33
## 6                        <NA> 1987-02-26 16:32:37
## 7      By Patti Domm, Reuter 1987-02-26 16:43:13
## 8                        <NA> 1987-02-26 16:59:25
## 9                        <NA> 1987-02-26 17:01:28
## 10                       <NA> 1987-02-26 17:08:27
##                                              heading    id language            origin topics
##                                                <chr> <chr>    <chr>             <chr>  <chr>
## 1   COMPUTER TERMINAL SYSTEMS <CPML> COMPLETES SALE    10       en Reuters-21578 XML    YES
## 2    OHIO MATTRESS <OMT> MAY HAVE LOWER 1ST QTR NET    12       en Reuters-21578 XML    YES
## 3     MCLEAN'S <MII> U.S. LINES SETS ASSET TRANSFER    44       en Reuters-21578 XML    YES
## 4    CHEMLAWN <CHEM> RISES ON HOPES FOR HIGHER BIDS    45       en Reuters-21578 XML    YES
## 5    <COFAB INC> BUYS GULFEX FOR UNDISCLOSED AMOUNT    68       en Reuters-21578 XML    YES
## 6          INVESTMENT FIRMS CUT CYCLOPS <CYL> STAKE    96       en Reuters-21578 XML    YES
## 7  AMERICAN EXPRESS <AXP> SEEN IN POSSIBLE SPINNOFF   110       en Reuters-21578 XML    YES
## 8   HONG KONG FIRM UPS WRATHER<WCO> STAKE TO 11 PCT   125       en Reuters-21578 XML    YES
## 9               LIEBERT CORP <LIEB> APPROVES MERGER   128       en Reuters-21578 XML    YES
## 10    GULF APPLIED TECHNOLOGIES <GATS> SELLS UNITS   134       en Reuters-21578 XML    YES
## # ... with 40 more rows, and 8 more variables: 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.

```r
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
```

```r
# 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
```

6.3.1 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, but you'll certainly find different results if you ran it for yourself).

```
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))
```

```
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                                             id
##        <chr>              <dttm>       <chr>                                          <chr>
## 1  Microsoft 2017-01-17 12:07:24   microsoft tag:finance.google.com,cluster:52779347599411
## 2  Microsoft 2017-01-17 12:07:24 corporation tag:finance.google.com,cluster:52779347599411
## 3  Microsoft 2017-01-17 12:07:24        data tag:finance.google.com,cluster:52779347599411
## 4  Microsoft 2017-01-17 12:07:24     privacy tag:finance.google.com,cluster:52779347599411
## 5  Microsoft 2017-01-17 12:07:24       could tag:finance.google.com,cluster:52779347599411
## 6  Microsoft 2017-01-17 12:07:24        send tag:finance.google.com,cluster:52779347599411
## 7  Microsoft 2017-01-17 12:07:24        msft tag:finance.google.com,cluster:52779347599411
## 8  Microsoft 2017-01-17 12:07:24       stock tag:finance.google.com,cluster:52779347599411
## 9  Microsoft 2017-01-17 12:07:24     soaring tag:finance.google.com,cluster:52779347599411
## 10 Microsoft 2017-01-17 12:07:24          by tag:finance.google.com,cluster:52779347599411
## # ... with 105,047 more rows, and 1 more variables: heading <chr>
```

Here we see the 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.

```r
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 . As we'd expect the company's name and is 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).

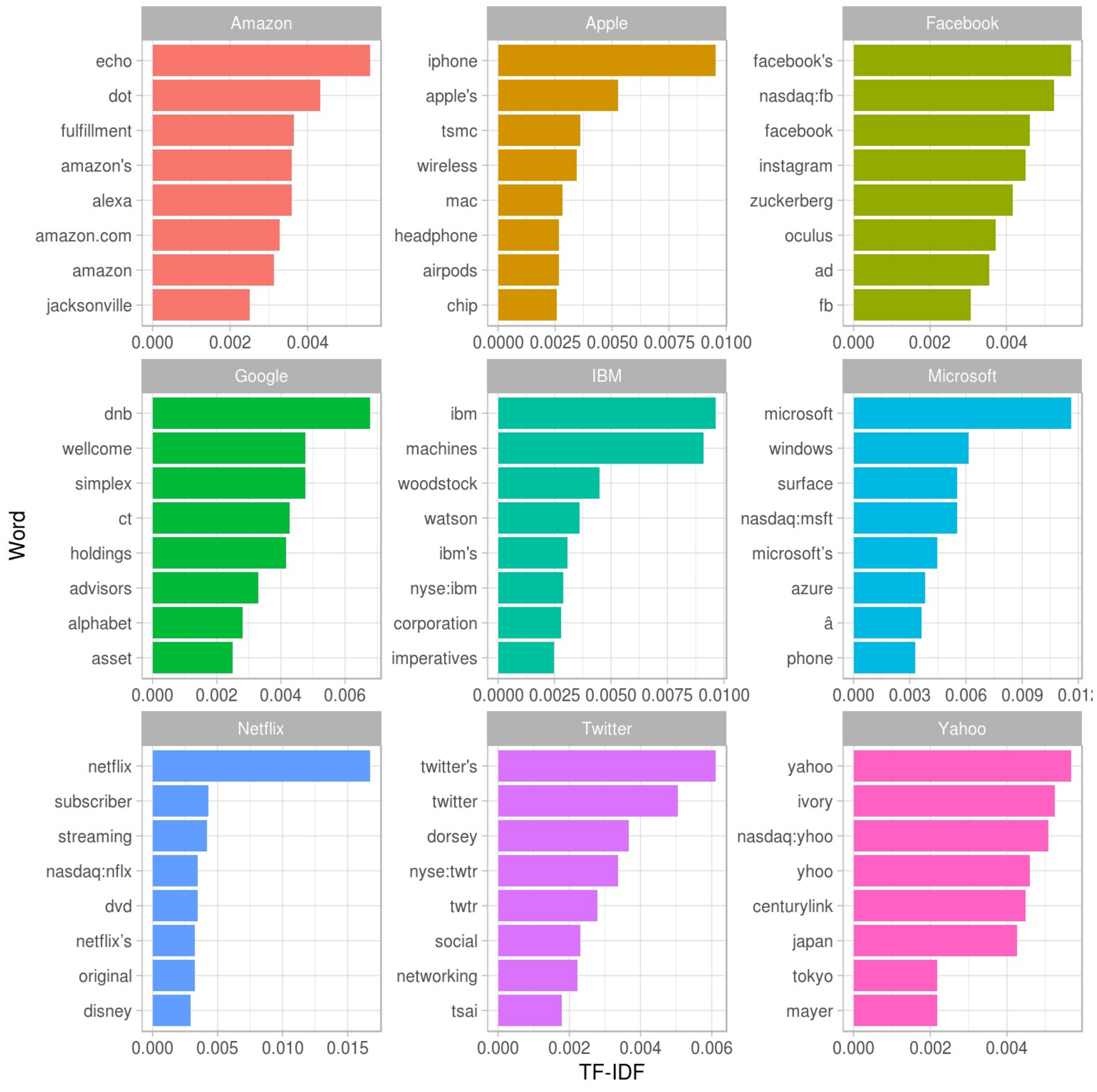


Figure 6.4: The 8 words with the highest TF-IDF in recent articles specific to each company.

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 Chapter . For example, we could examine this within the AFINN lexicon (Figure).

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

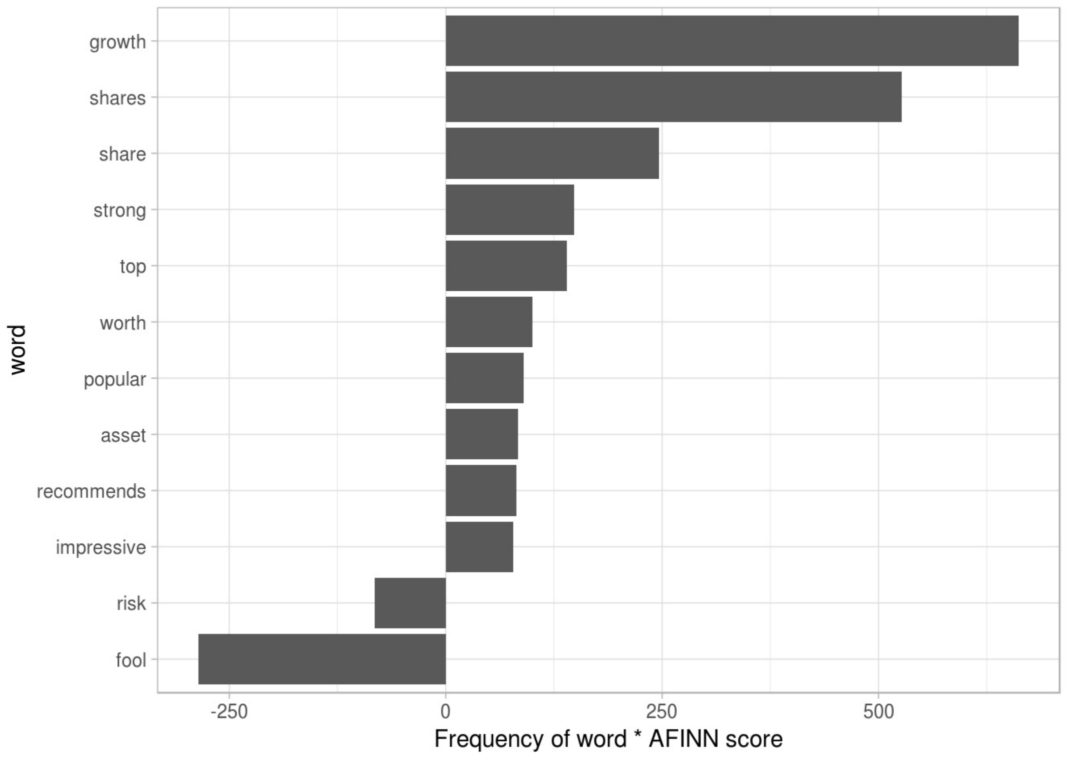


Figure 6.5: The words with the largest contribution to sentiment scores in recent financial articles, according to the AFINN dictionary. The 'contribution' is the product of the word and the sentiment score.

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 was $X 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).

Instead, we introduce another sentiment lexicon: the Loughran and McDonald dictionary of financial sentiment terms (**???**). 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.

```
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")
```

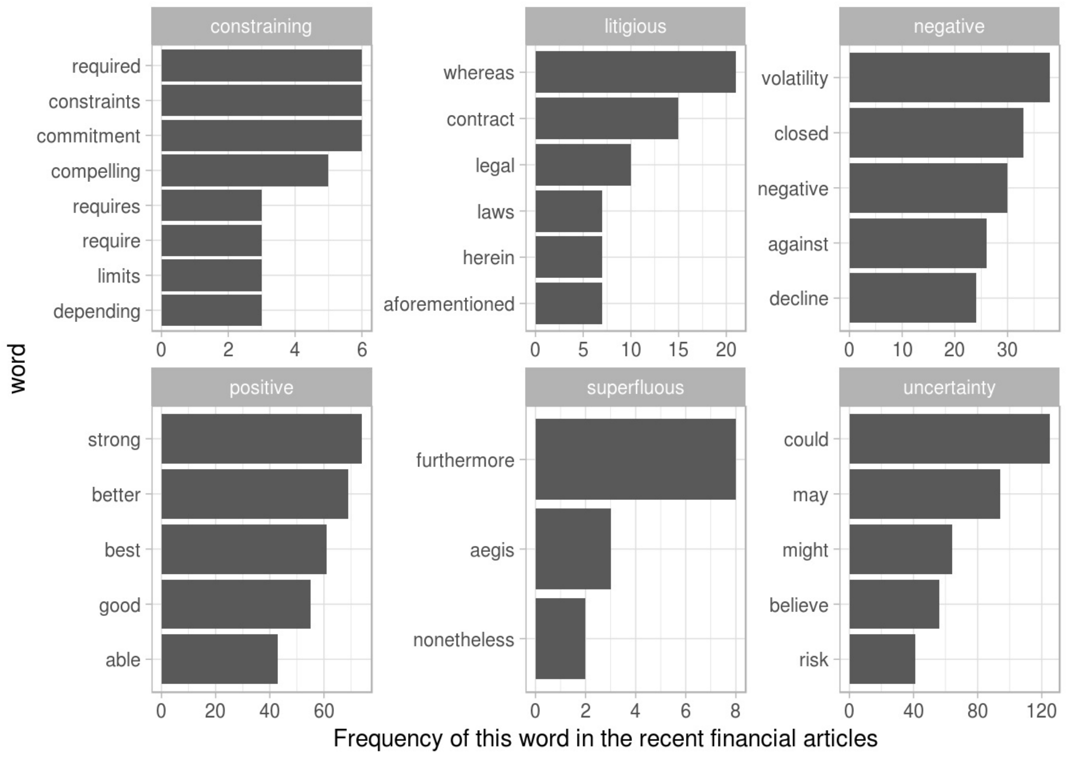


Figure 6.6: The most common words in the financial news articles associated with each of the six sentiments in the Loughran and McDonald lexicon

These assignments (Figure) 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 analysis in Chapter , 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).

```
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")
```

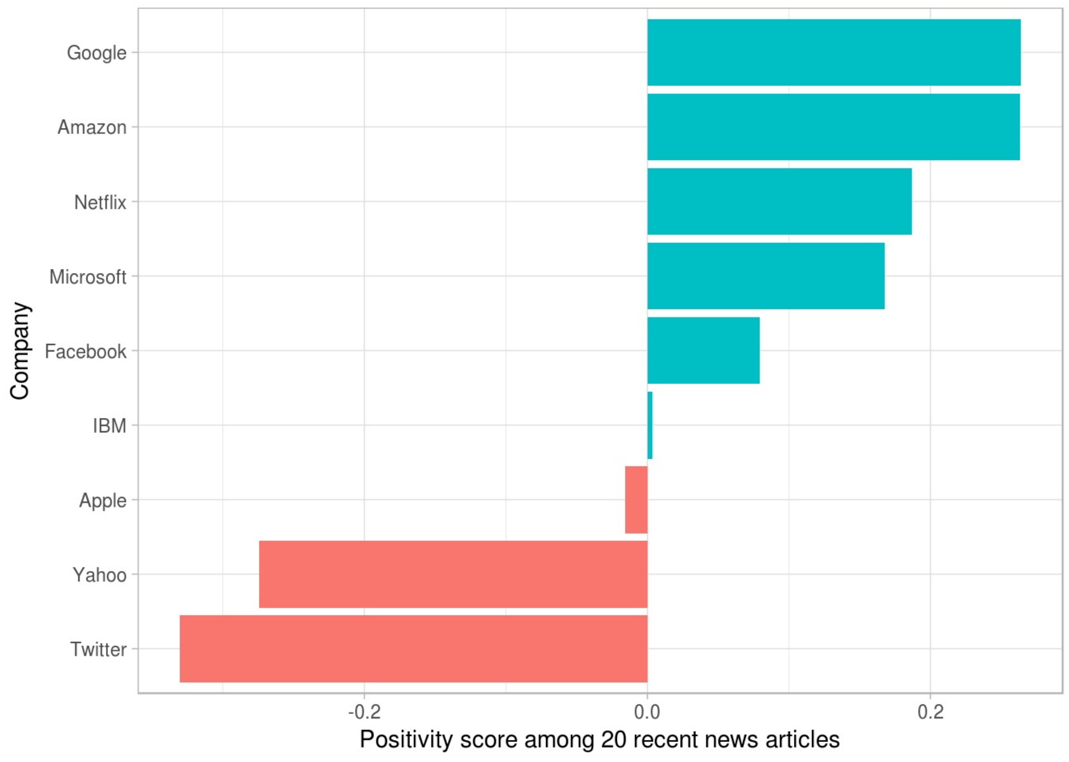


Figure 6.7: ‘Positivity’ of the news coverage around each stock in January 2017, calculated as (positive - negative) / (positive + negative), based on uses of positive and negative words in 20 recent news articles

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 suggest 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.

References

Robinson, David, Matthieu Gomez, Boris Demeshev, Dieter Menne, Benjamin Nutter, Luke Johnston, Ben Bolker, Francois Briatte, and Hadley Wickham. 2015. *Broom: Convert Statistical Analysis Objects into Tidy Data Frames*. https://CRAN.R-project.org/package=broom.

Wickham, Hadley. 2007. “Reshaping Data with the reshape Package.” *Journal of Statistical Software* 21 (12): 1–20. http://www.jstatsoft.org/v21/i12/.

Benoit, Kenneth, and Paul Nulty. 2016. *Quanteda: Quantitative Analysis of Textual Data*. https://CRAN.R-project.org/package=quanteda.

7 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 numerical data, which finds natural groups of items even when we're not sure what we're looking for.

Latent Dirichlet allocation 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.

We can use tidy text principles to approach topic modeling using consistent and effective tools. In this chapter, we'll learn to tidy LDA objects from the topicmodels package to examine model results using 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.

7.1 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.

- **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.
- **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."

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 word. It has a number of existing implementations, and we'll explore one of them.

In Chapter we briefly introduced the `AssociatedPress` dataset provided by the topicmodels package, as an example of a DocumentTermMatrix. This is a collection of 2246 news articles from an American news agency, mostly published around 1988.

```
library(topicmodels)

data("AssociatedPress")
AssociatedPress
```

Processing math: 100%

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

We can then the `LDA()` function from the topicmodels package to create a two-topic Latent Dirichlet allocation model. This function returns an object containing the full details of the 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 remainder of the analysis will involve exploring and interpreting the model using tidying functions.

7.1.1 Word-topic probabilities

In Chapter we introduced the `tidy()` method, originally from the broom package, 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_lda_td <- tidy(ap_lda)
ap_lda_td
```

```
## # 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

We could use dplyr's `top_n()` to find the top 10 terms within each topic.As a tidy data frame, this lends itself well to a ggplot2 visualization (Figure).

```
library(ggplot2)

ap_top_terms <- ap_lda_td %>%
  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_bar(alpha = 0.8, stat = "identity", show.legend = FALSE) +
  facet_wrap(~ topic, scales = "free") +
  coord_flip()
```

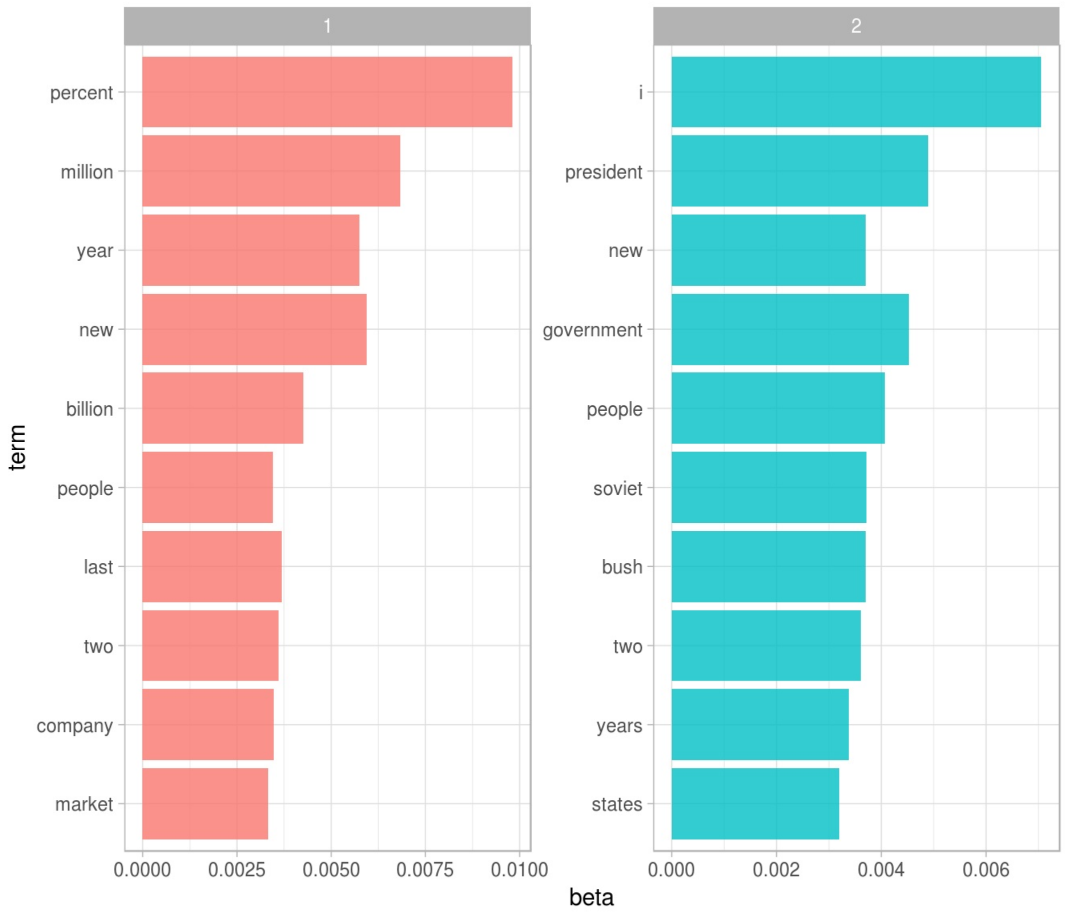


Figure 7.1: The terms that are most common within each topic

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 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 often have some overlap.

As an alternative, we could consider the terms that had the *greatest difference* between topic 1 and topic 2. This can be estimated based on the log ratio of the two: $\log_2(\frac{\beta_2}{\beta_1})$. (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).

```
beta_spread <- ap_lda_td %>%
  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 .

```
beta_spread %>%
  group_by(direction = log_ratio > 0) %>%
  top_n(10, abs(log_ratio)) %>%
  mutate(term = reorder(term, log_ratio)) %>%
  ggplot(aes(term, log_ratio)) +
  geom_col() +
  labs(y = "Log2 ratio of beta in topic 2 / topic 1") +
  coord_flip()
```

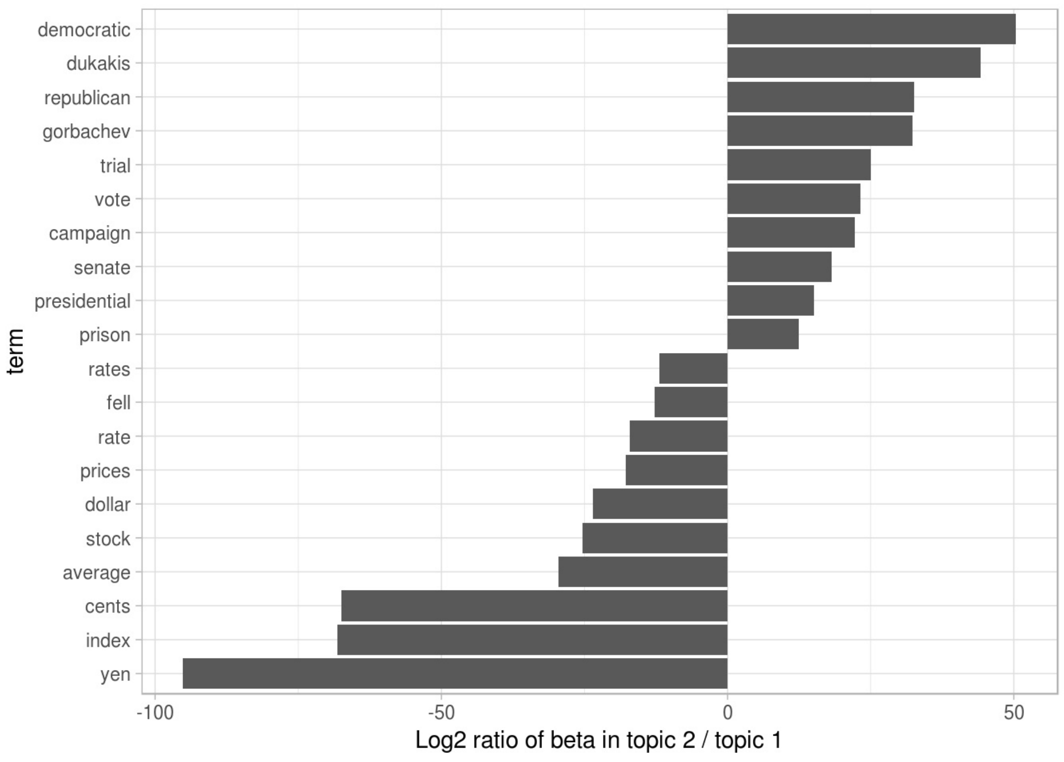


Figure 7.2: Words with the greatest difference in ‘beta’ between topic 2 and topic 1

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 was 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 were political and financial news.

7.1.2 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

$$\gamma$$

(“gamma”), with the `matrix = "gamma"` argument to `tidy()` .

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

```
## # 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 each word in the first document has only a 24.8% probability of coming from topic 1.

We can see that many of these documents were drawn from a mix of the two topics, but that document 6 was drawn almost entirely from topic 2, having a `gamma` of only 5.910^{-4} from topic 1. To check this answer, we could see what the most common words in that document were.

```
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).

7.2 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.

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 representing one of the books.

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

```
library(dplyr)

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 pre-processing, 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 practice, each document might be one newspaper article, or one blog post).

```
library(tidytext)
library(stringr)
library(tidyr)

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

7.2.1 LDA on chapters

Right now our data frame `word_counts` is in a tidy form, with one-term-per-document-per-row. However, the topicmodels package requires a `DocumentTermMatrix` (from the tm package). As described in Chapter , we can cast a one-token-per-row table into a `DocumentTermMatrix` with tidytext's `cast_dtm()` .

```r
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)
```

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`.

```r
library(topicmodels)

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.

```r
chapters_lda_td <- tidy(chapters_lda)
chapters_lda_td
```

```
## # 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% chance of being generated from topic 4.

We could use dplyr's `top_n()` to find the top 5 terms within each topic:

```
top_terms <- chapters_lda_td %>%
  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
```

This output lends itself well to a ggplot2 visualization (Figure).

```
library(ggplot2)

top_terms %>%
  mutate(term = reorder(term, beta)) %>%
  ggplot(aes(term, beta, fill = factor(topic))) +
  geom_bar(alpha = 0.8, stat = "identity", show.legend = FALSE) +
  facet_wrap(~ topic, scales = "free") +
  coord_flip()
```

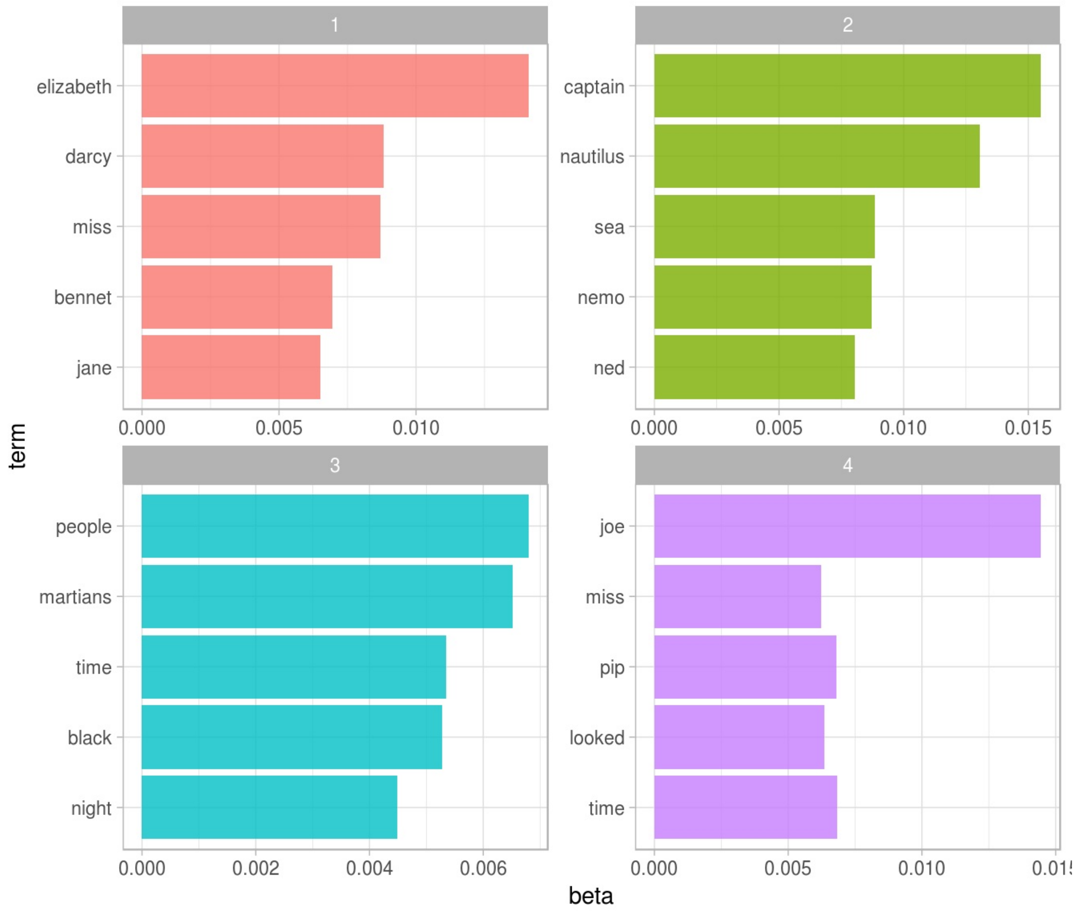


Figure 7.3: The terms that are most common within each topic

These topics are pretty clearly associated with the four books! There's no question that the topic of "nemo", "sea", and "nautilus" 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 there can be words in common between multiple topics, such as "miss" in topics 1 and 4, and "time" in topics 3 and 4. This shows how LDA is a "fuzzy clustering" method: rather than giving particular words to each topic, it treats them as a mixture of all words, with different proportions.

7.3 Per-document classification

Each chapter was a "document" in this analysis. 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

```
chapters_lda_gamma <- tidy(chapters_lda, matrix = "gamma")
chapters_lda_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.

Now that we have these document classifiations, 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).

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

chapters_lda_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
```

```r
ggplot(chapters_lda_gamma, aes(factor(topic), gamma)) +
  geom_boxplot() +
  facet_wrap(~ title)
```

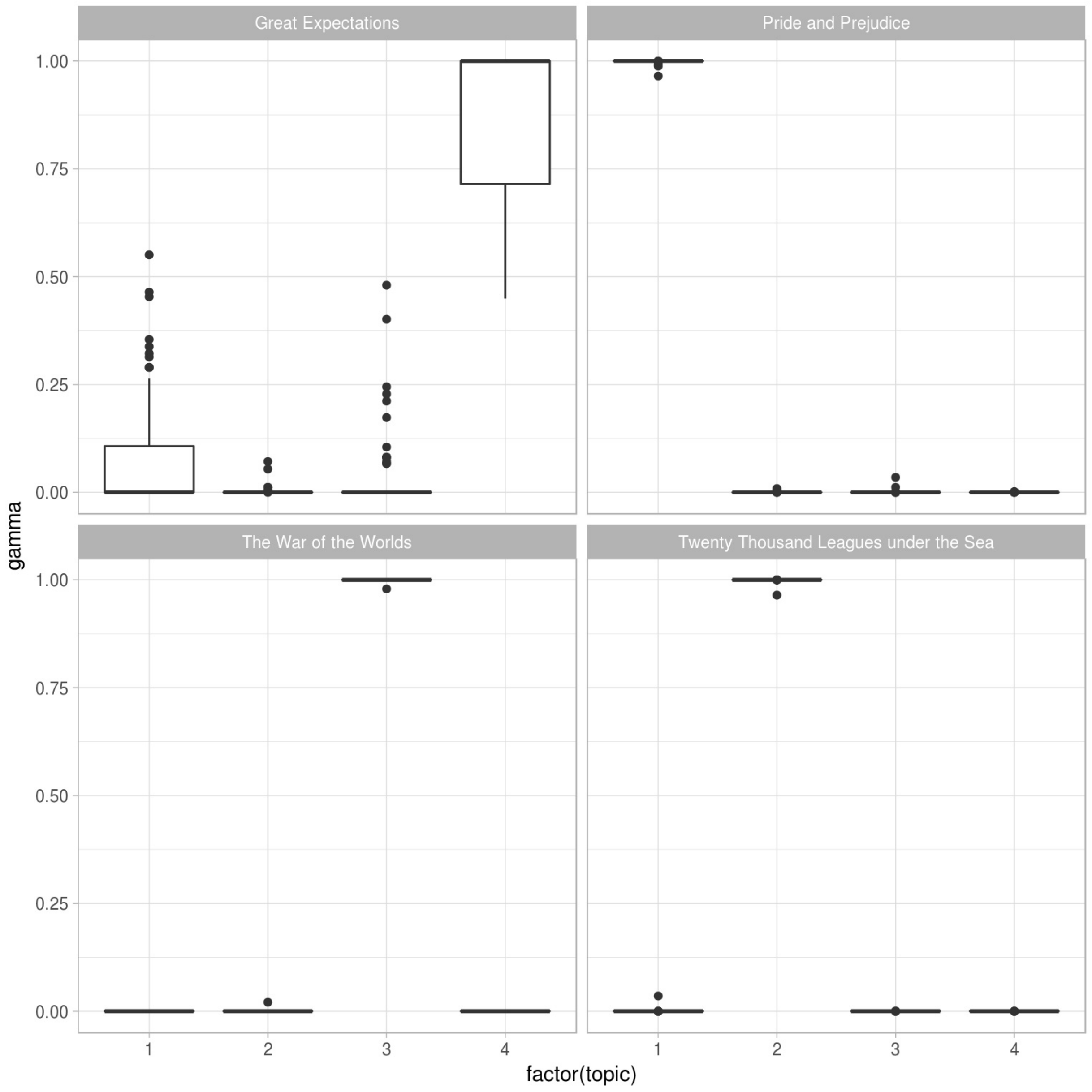


Figure 7.4: The gamma probabilities for each chapter within each book

We notice that almost all of the chapters from *Pride and Prejudice*, *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_lda_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) %>%
  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!

7.4 By word assignments: `augment`

One step that LDA performs 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 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
```

We can, for example, create a “confusion matrix,” showing how often words from one book were assigned to another, using dplyr’s `count()` and tidyr’s `spread()`.

```
assignments %>%
  count(title, consensus, wt = count) %>%
  spread(consensus, n, fill = 0)
```

```
## Source: local data frame [4 x 5]
## Groups: title [4]
##
##                                    title `Great Expectations` `Pride and Prejudice`
## *                                  <chr>                <dbl>                 <dbl>
## 1                     Great Expectations                49770                  3876
## 2                    Pride and Prejudice                    1                 37229
## 3                  The War of the Worlds                    0                     0
## 4 Twenty Thousand Leagues under the Sea                    0                     5
##   `The War of the Worlds` `Twenty Thousand Leagues under the Sea`
## *                   <dbl>                                   <dbl>
## 1                    1845                                      77
## 2                       7                                       5
## 3                   22561                                       7
## 4                       0                                   39629
```

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

What were the most commonly mistaken words?

```r
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
```

```r
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 algorithm is stochastic and iterative, and it can accidentally land on a topic that spans multiple books.

7.5 Alternative LDA implementations

TODO: this section is not complete.

The `LDA` function in the topicmodels package is only one implementation of the latent Dirichlet allocation algorithm. For example, the mallet package implements a wrapper around the MALLET package for text classification tools.

The way an algorithm is run with this algorithm is very different from LDA: it takes non-tokenized documents and performs the tokenization itself, and requires a separate file of stopwords.

```r
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 = " "))

# The mallet package requires a file of stopwords
# Since we've already filtered them, we can give it an empty file
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, are almost identical to the tidiers described in this rest of this chapter.

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

```
## # A tibble: 71,064 × 3
##    topic    term         beta
##    <int>   <chr>        <dbl>
## 1      1 limping 2.761847e-07
## 2      2 limping 1.107103e-04
## 3      3 limping 1.852309e-07
## 4      4 limping 3.454630e-07
## 5      1  pirate 2.761847e-07
## 6      2  pirate 1.107103e-04
## 7      3  pirate 1.852309e-07
## 8      4  pirate 3.454630e-07
## 9      1  gibbet 8.313158e-05
## 10     2  gibbet 2.760855e-07
## # ... with 71,054 more rows
```

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

```
## # A tibble: 772 × 3
##                 document topic     gamma
##                    <chr> <int>     <dbl>
## 1   Great Expectations_1     1 0.3959413
## 2  Great Expectations_10     1 0.2715783
## 3  Great Expectations_11     1 0.4302147
## 4  Great Expectations_12     1 0.3161451
## 5  Great Expectations_13     1 0.3373596
## 6  Great Expectations_14     1 0.2469262
## 7  Great Expectations_15     1 0.3470206
## 8  Great Expectations_16     1 0.2863790
## 9  Great Expectations_17     1 0.3216292
## 10 Great Expectations_18     1 0.3005751
## # ... with 762 more rows
```

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

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

8 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 of the 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.

8.1 Getting the data and distribution of tweets

An individual can download their 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.

```r
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(alpha = 0.5, position = "identity", bins = 20)
```

Loading [MathJax]/jax/output/HTML-CSS/fonts/TeX/fontdata.js

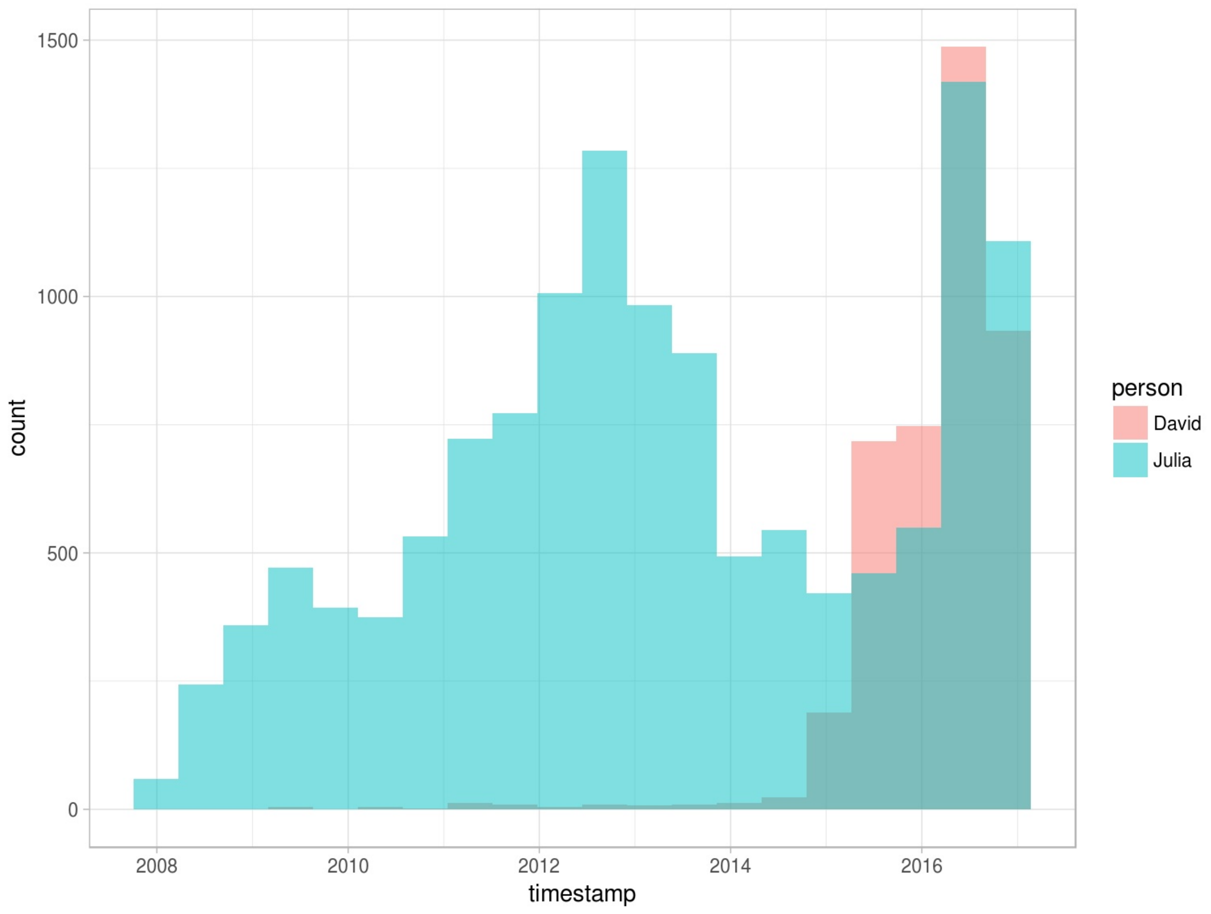


Figure 8.1: All tweets from our accounts

David and Julia tweet at about the same rate currently and joined Twitter about a year apart from each other, but there were about 5 years where David was not active on Twitter and Julia was. In total, Julia has about 4 times as many tweets as David.

8.2 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 library.

```
library(tidytext)
library(stringr)

reg <- "([^A-Za-z_\\d#@']|'(?![A-Za-z_\\d#@]))"
tidy_tweets <- tweets %>%
  filter(!str_detect(text, "^RT")) %>%
  mutate(text = str_replace_all(text, "https://t.co/[A-Za-z\\d]+|http://[A-Za-z\\d]+|&amp;|&lt
, "")) %>%
  unnest_tokens(word, text, token = "regex", pattern = 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 make a differently shaped data frame.

```r
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.

```r
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")
```

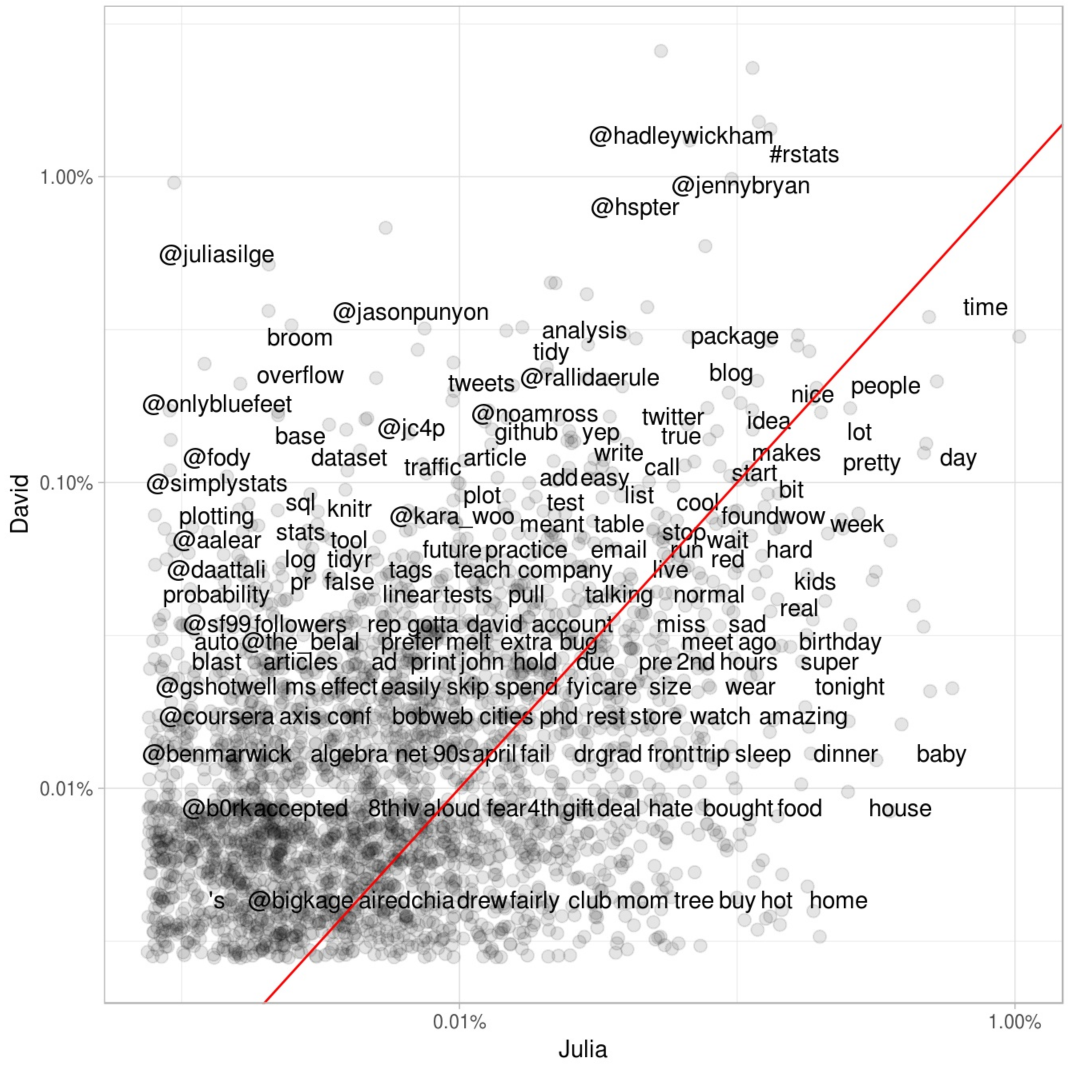


Figure 8.2: Comparing the frequency of words used by Julia and David

Words near the red line in Figure 8.2 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.

8.3 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) %>%
  spread(person, n, fill = 0) %>%
  ungroup() %>%
  mutate_each(funs((. + 1) / sum(. + 1)), -word) %>%
  mutate(logratio = log(David / Julia)) %>%
  arrange(desc(logratio))
```

What are some words that have been about equally likely to come from David 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.

```
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_bar(alpha = 0.8, stat = "identity") +
  coord_flip() +
  ylab("log odds ratio (David/Julia)") +
  scale_fill_discrete(name = "", labels = c("David", "Julia"))
```

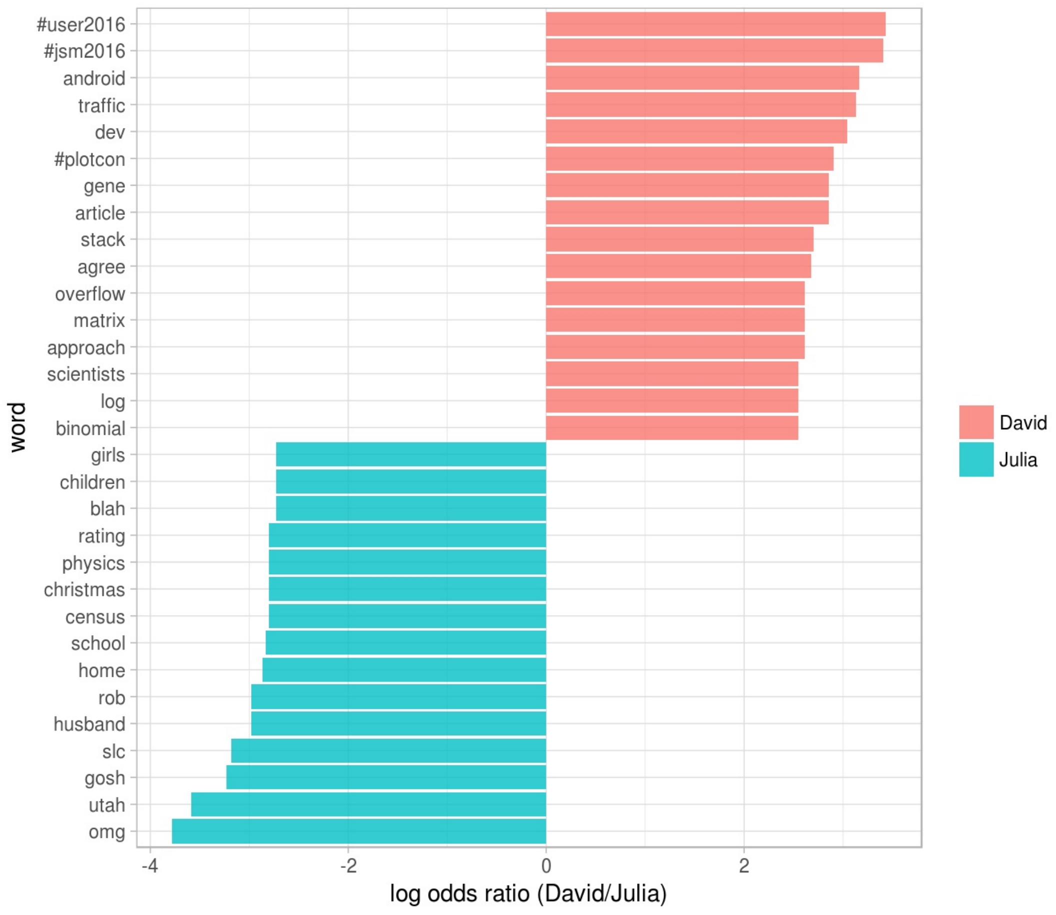


Figure 8.3: Comparing the odds ratios of words from our accounts

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.

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

```
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
```

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)

top_slopes
```

```
## # A tibble: 6 × 8
##    person      word       term      estimate    std.error statistic      p.value adjusted.p.
##     <chr>     <chr>      <chr>         <dbl>        <dbl>     <dbl>        <dbl>
## 1   David   ggplot2 time_floor -8.262540e-08 1.969448e-08 -4.195359 2.724397e-05     2.99683
## 2   Julia   #rstats time_floor -4.496395e-08 1.119780e-08 -4.015427 5.933815e-05     6.46785
## 3   Julia      post time_floor -4.818545e-08 1.454440e-08 -3.312990 9.230419e-04     9.78424
## 4   Julia      read time_floor -9.327168e-08 2.542485e-08 -3.668524 2.439548e-04     2.63471
## 5   David     stack time_floor  8.041202e-08 2.193375e-08  3.666132 2.462468e-04     2.63484
## 6   David #user2016 time_floor -8.175896e-07 1.550152e-07 -5.274253 1.332976e-07     1.47960
```

To visualize our results, we can plot these words' use for both David and Julia over this year of tweets.

```
words_by_time %>%
  inner_join(top_slopes, by = c("word", "person")) %>%
  filter(person == "David") %>%
  ggplot(aes(time_floor, count/time_total, color = word)) +
  geom_line(alpha = 0.8, size = 1.3) +
  labs(x = NULL, y = "Word frequency")
```

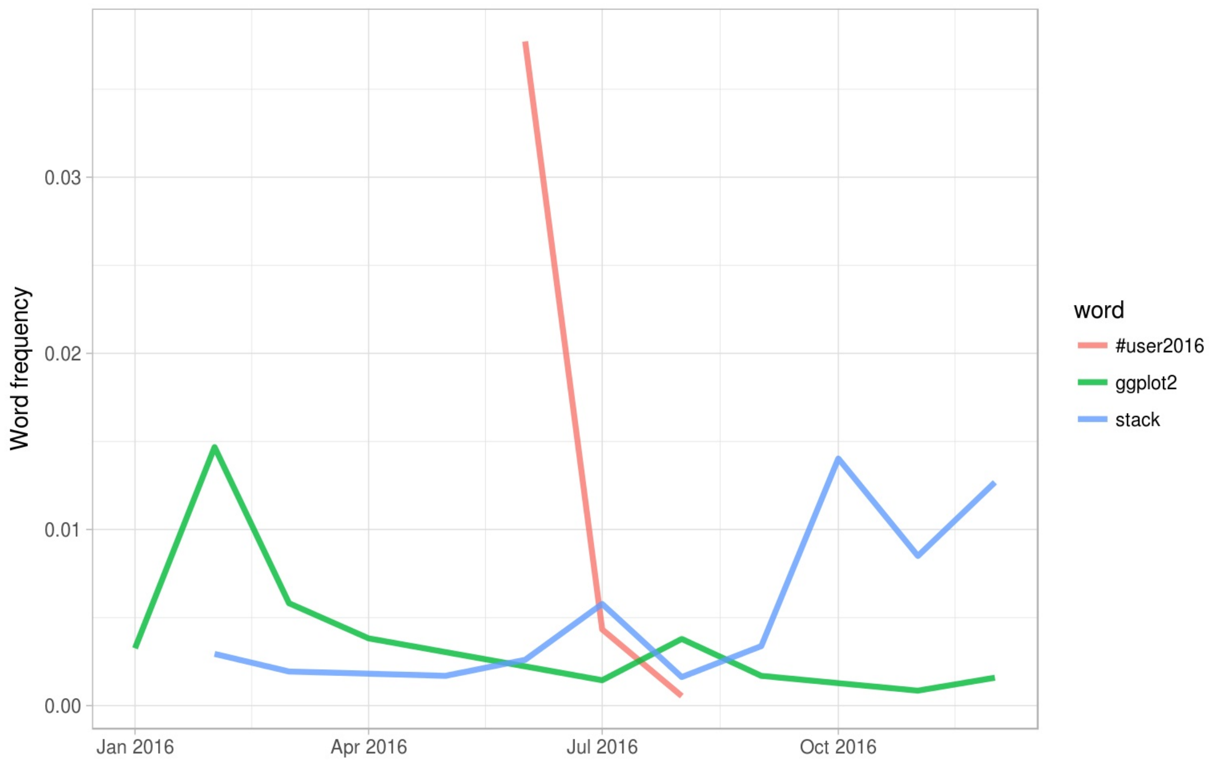
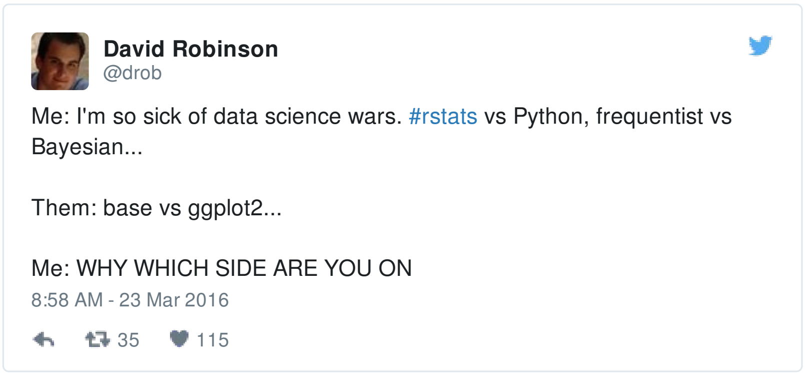


Figure 8.4: Trending words in David's tweets

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.



David Robinson
@drob

Me: I'm so sick of data science wars. #rstats vs Python, frequentist vs Bayesian...

Them: base vs ggplot2...

Me: WHY WHICH SIDE ARE YOU ON

8:58 AM - 23 Mar 2016

35 115

Now let's plot words that have changed frequency in Julia's tweets.

```
words_by_time %>%
  inner_join(top_slopes, by = c("word", "person")) %>%
  filter(person == "Julia") %>%
  ggplot(aes(time_floor, count/time_total, color = word)) +
  geom_line(alpha = 0.8, size = 1.3) +
  labs(x = NULL, y = "Word frequency")
```

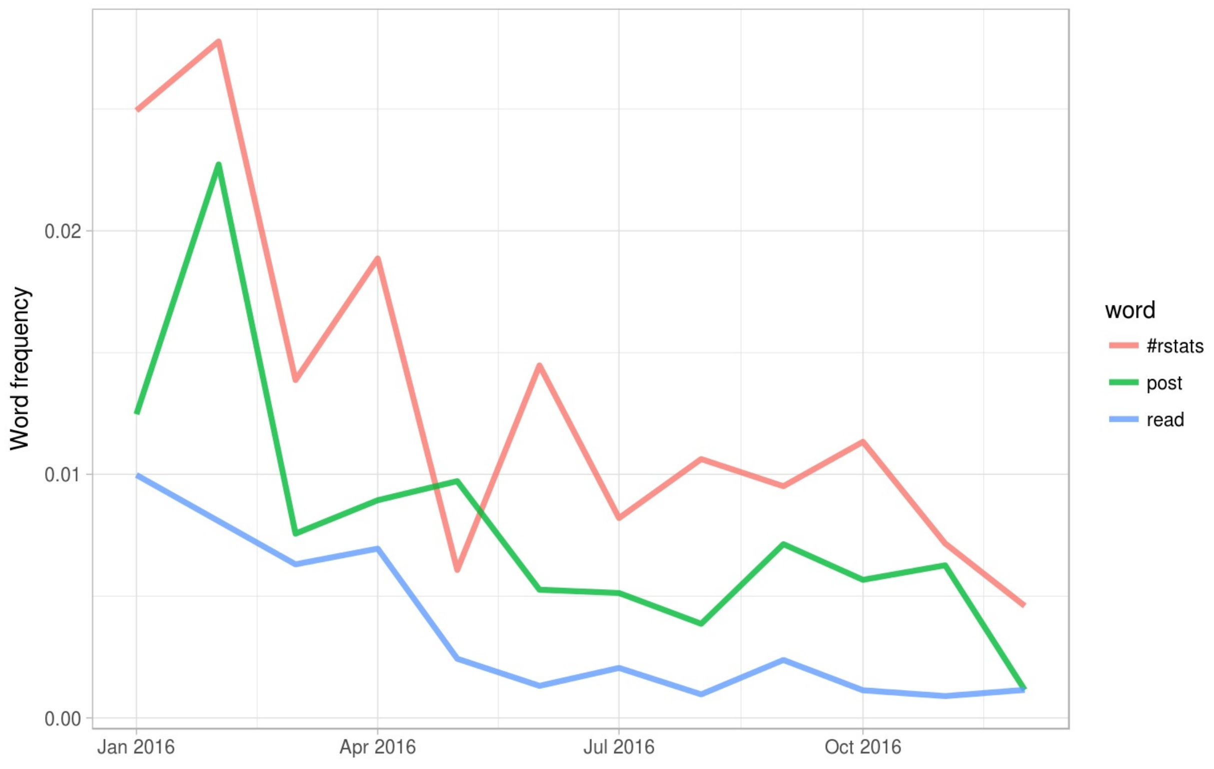


Figure 8.5: Trending words in Julia's tweets

All the significant slopes for Julia are negative. This means she has not tweeted at a higher rate using any specific words, but instead using 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.

8.5 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 their 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 3200 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_tweets <- tweets %>%
  filter(!str_detect(text, "^(RT|@)")) %>%
  mutate(text = str_replace_all(text, "https://t.co/[A-Za-z\\d]+|http://[A-Za-z\\d]+|&amp;|&lt
, "")) %>%
  unnest_tokens(word, text, token = "regex", pattern = reg) %>%
  anti_join(stop_words)

tidy_tweets
```

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

To start with, let's look at retweets. 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. In the second `summarise()` statement, we can also count the number of times each word was used ever 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 5 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.

```
word_by_rts %>%
  filter(uses >= 5) %>%
  group_by(person) %>%
  top_n(10, retweets) %>%
  arrange(retweets) %>%
  mutate(word = factor(word, unique(word))) %>%
  ungroup() %>%
  ggplot(aes(word, retweets, fill = person)) +
  geom_bar(stat = "identity", alpha = 0.8, show.legend = FALSE) +
  facet_wrap(~ person, scales = "free", ncol = 2) +
  coord_flip() +
  labs(x = NULL,
       y = "Median # of retweets for tweets containing each word")
```

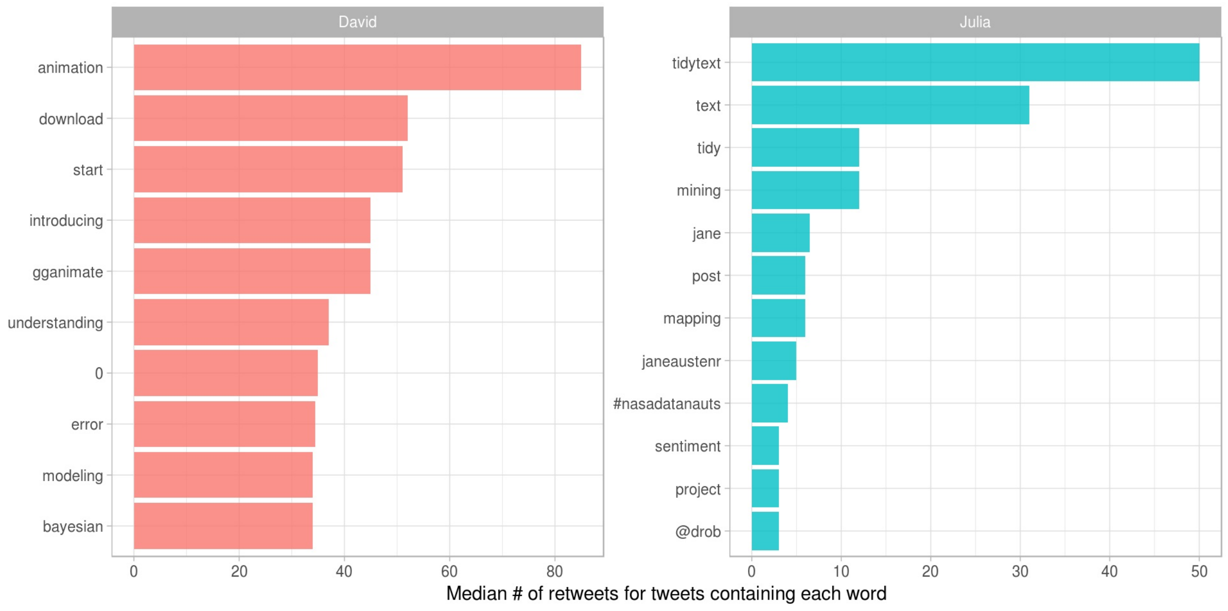


Figure 8.6: Words with highest median retweets

We see lots of word 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?

```r
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.

```
word_by_favs %>%
  filter(uses >= 5) %>%
  group_by(person) %>%
  top_n(10, favorites) %>%
  arrange(favorites) %>%
  mutate(word = factor(word, unique(word))) %>%
  ungroup() %>%
  ggplot(aes(word, favorites, fill = person)) +
  geom_bar(stat = "identity", alpha = 0.8, show.legend = FALSE) +
  facet_wrap(~ person, scales = "free", ncol = 2) +
  coord_flip() +
  labs(x = NULL,
       y = "Median # of favorites for tweets containing each word")
```

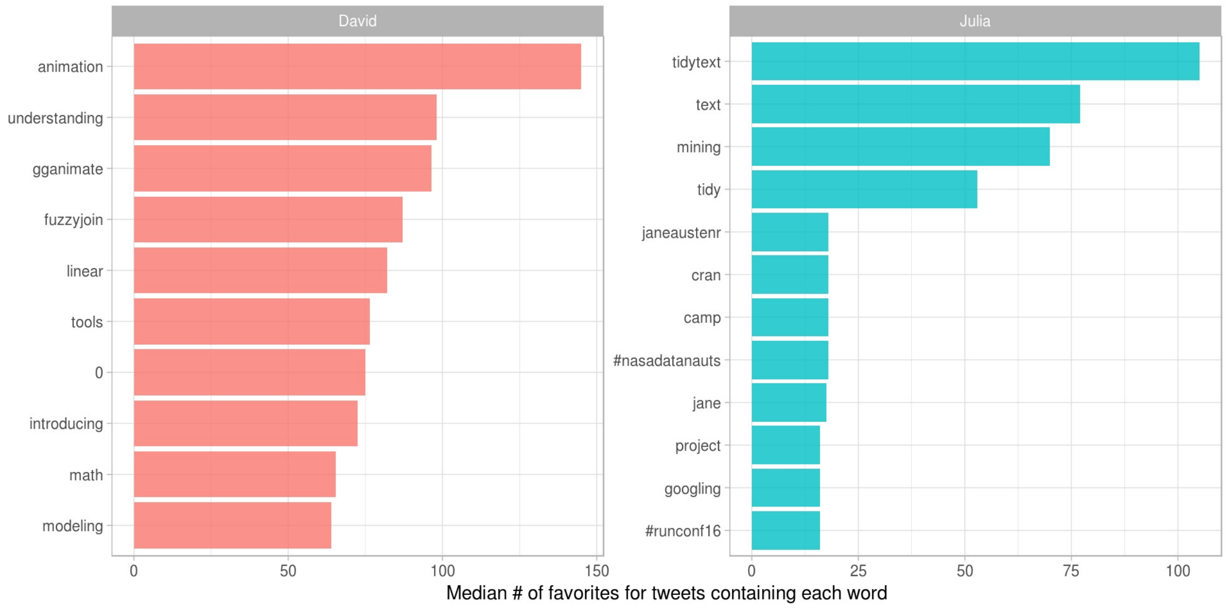


Figure 8.7: Words with highest median favorites

We see some minor differences between Figures 8.6 and 8.7, 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 9 to learn more about NASA data and what we can learn from text analysis of NASA datasets.

9 Case study: mining NASA metadata

There are over 32,000 datasets hosted and/or maintained by NASA; these datasets cover topics from Earth science to aerospace engineering to management of NASA itself. We can use the metadata for these datasets to understand the connections between them.

What is metadata? Metadata is a term that refers to data that gives information about other data; in this case, the metadata informs users about what is in these numerous NASA datasets but does not include the content of the datasets themselves. The metadata includes information like the title of the dataset, a description field, what organization(s) within NASA is responsible for the dataset, keywords for the dataset that have been assigned by a human being, and so forth. NASA places a high priority on making its data open and accessible, even requiring all NASA-funded research to be openly accessible online. The metadata for all its datasets is publicly available online in JSON format.

In this chapter, we will treat the NASA metadata as a text dataset and show how to implement several tidy text approaches with this real-life text. We will use word co-occurrences and correlations, tf-idf, and topic modeling to explore the connections between the datasets. Can we find datasets that are related to each other? Can we find clusters of similar datasets? Since we have several text fields in the NASA metadata, most importantly the title, description, and keyword fields, we can explore the connections between the fields to better understand the complex world of data at NASA. This type of approach can be extended to any domain that deals with text, so let’s take a look at this metadata and get started.

9.1 How data is organized at NASA

First, let’s download the JSON file and take a look at the names of what is stored in the metadata.

```r
library(jsonlite)
metadata <- fromJSON("https://data.nasa.gov/data.json")
names(metadata$dataset)
```

```
##  [1] "_id"                "@type"              "accessLevel"        "accrualPeriodicity"
##  [5] "bureauCode"         "contactPoint"       "description"        "distribution"
##  [9] "identifier"         "issued"             "keyword"            "landingPage"
## [13] "language"           "modified"           "programCode"        "publisher"
## [17] "spatial"            "temporal"           "theme"              "title"
## [21] "license"            "isPartOf"           "references"         "rights"
## [25] "describedBy"
```

We see here that we could extract information from who publishes each dataset to what license they are released under.

It seems likely that the title, description, and keywords for each dataset may be most fruitful for drawing connections between datasets. Let’s check them out.

```
class(metadata$dataset$title)
```

```
## [1] "character"
```

```
class(metadata$dataset$description)
```

```
## [1] "character"
```

```
class(metadata$dataset$keyword)
```

```
## [1] "list"
```

The title and description fields are stored as character vectors, but the keywords are stored as a list of character vectors.

9.1.1 Wrangling and tidying the data

Let's set up separate tidy data frames for title, description, and keyword, keeping the dataset ids for each so that we can connect them later in the analysis if necessary.

```
library(dplyr)

nasa_title <- data_frame(id = metadata$dataset$`_id`$`$oid`,
                         title = metadata$dataset$title)
nasa_title
```

```
## # A tibble: 32,089 × 2
##                          id                                             title
##                       <chr>                                             <chr>
## 1  55942a57c63a7fe59b495a77            15 Minute Stream Flow Data: USGS (FIFE)
## 2  55942a57c63a7fe59b495a78            15 Minute Stream Flow Data: USGS (FIFE)
## 3  55942a58c63a7fe59b495a79            15 Minute Stream Flow Data: USGS (FIFE)
## 4  55942a58c63a7fe59b495a7a 2000 Pilot Environmental Sustainability Index (ESI)
## 5  55942a58c63a7fe59b495a7b 2000 Pilot Environmental Sustainability Index (ESI)
## 6  55942a58c63a7fe59b495a7c 2000 Pilot Environmental Sustainability Index (ESI)
## 7  55942a58c63a7fe59b495a7d       2001 Environmental Sustainability Index (ESI)
## 8  55942a58c63a7fe59b495a7e       2001 Environmental Sustainability Index (ESI)
## 9  55942a58c63a7fe59b495a7f       2001 Environmental Sustainability Index (ESI)
## 10 55942a58c63a7fe59b495a80       2001 Environmental Sustainability Index (ESI)
## # ... with 32,079 more rows
```

These are just a few example titles from the datasets we will be exploring. Notice that we have the NASA-assigned ids here, and also that there are duplicate titles on separate datasets.

```
nasa_desc <- data_frame(id = metadata$dataset$`_id`$`$oid`,
                        desc = metadata$dataset$description)

nasa_desc %>%
  select(desc) %>%
  sample_n(5)
```

```
## # A tibble: 5 × 1
##
##
## 1 A WiMAX networked UAV Telemetry System (WNUTS) is designed for\nnet-centric remote sensin
## 2  ML3DZMBRO is the EOS Aura Microwave Limb Sounder (MLS) daily zonal mean product for brom
## 3  This Level 2 data collection contains derived precipitable column water vapor amounts, d
## 4  To address the NASA GSFC need for significant improvements in wide bandgap materials and
## 5  This data set includes brightness temperature data measured over the Sea of Japan, the W
```

Here we see the first part of several selected description fields from the metadata.

Now we can build the tidy data frame for the keywords. For this one, we need to use `unnest()` from tidyr, because they are in a list-column.

```
library(tidyr)

nasa_keyword <- data_frame(id = metadata$dataset$`_id`$`$oid`,
                           keyword = metadata$dataset$keyword) %>%
  unnest(keyword)

nasa_keyword
```

```
## # A tibble: 126,814 × 2
##                           id       keyword
##                        <chr>         <chr>
## 1  55942a57c63a7fe59b495a77 EARTH SCIENCE
## 2  55942a57c63a7fe59b495a77   HYDROSPHERE
## 3  55942a57c63a7fe59b495a77 SURFACE WATER
## 4  55942a57c63a7fe59b495a78 EARTH SCIENCE
## 5  55942a57c63a7fe59b495a78   HYDROSPHERE
## 6  55942a57c63a7fe59b495a78 SURFACE WATER
## 7  55942a58c63a7fe59b495a79 EARTH SCIENCE
## 8  55942a58c63a7fe59b495a79   HYDROSPHERE
## 9  55942a58c63a7fe59b495a79 SURFACE WATER
## 10 55942a58c63a7fe59b495a7a EARTH SCIENCE
## # ... with 126,804 more rows
```

This is a tidy data frame because we have one row for each keyword; this means we will have multiple rows for each dataset because a dataset can have more than one keyword.

Now it is time to use tidytext's `unnest_tokens()` for the title and description fields so we can do the text analysis. Let's also remove common English words from the titles and descriptions. We will not remove stop words from the keywords, because those are short, human-assigned keywords like **RADIATION** or **CLIMATE INDICATORS**.

```
library(tidytext)

nasa_title <- nasa_title %>%
  unnest_tokens(word, title) %>%
  anti_join(stop_words)

nasa_desc <- nasa_desc %>%
  unnest_tokens(word, desc) %>%
  anti_join(stop_words)
```

These are now in the tidy text format that we have been working with throughout this book, with one token (word, in this case) per row; let's take a look before we move on in our analysis.

```
nasa_title
```

```
## # A tibble: 210,914 × 2
##                           id        word
##                        <chr>       <chr>
## 1  56d07ee5a759fdadc44e5923      marble
## 2  56d07ee5a759fdadc44e5923        epic
## 3  56d07c16a759fdadc44e5922      fitara
## 4  56d07c16a759fdadc44e5922        ocio
## 5  56cf5b00a759fdadc44e5849 implementing
## 6  56cf5b00a759fdadc44e5846    receding
## 7  56cf5b00a759fdadc44e5846   recursive
## 8  56cf5b00a759fdadc44e5840  complaints
## 9  56cf5b00a759fdadc44e583b       score
## 10 56cf5b00a759fdadc44e583a         fix
## # ... with 210,904 more rows
```

```
nasa_desc
```

```
## # A tibble: 2,677,811 × 2
##                           id           word
##                        <chr>          <chr>
## 1  56d07c16a759fdadc44e5922          fitara
## 2  56d07c16a759fdadc44e5922            ocio
## 3  56cf5b00a759fdadc44e584a   degradation's
## 4  56cf5b00a759fdadc44e5847         dchwp1s
## 5  56cf5b00a759fdadc44e5847        dchwp1sp
## 6  56cf5b00a759fdadc44e5847          dchwdp
## 7  56cf5b00a759fdadc44e5847         dchwsnf
## 8  56cf5b00a759fdadc44e5847         dchwssf
## 9  56cf5b00a759fdadc44e5847        bursting
## 10 56cf5b00a759fdadc44e5847 consequentially
## # ... with 2,677,801 more rows
```

9.1.2 Some initial simple exploration

What are the most common words in the NASA dataset titles? We can use `count()` from dplyr to check this out.

```
nasa_title %>%
  count(word, sort = TRUE)
```

```
## # A tibble: 11,614 × 2
##       word     n
##      <chr> <int>
## 1  project  7735
## 2     data  3354
## 3        1  2841
## 4    level  2400
## 5   global  1809
## 6       v1  1478
## 7    daily  1397
## 8        3  1364
## 9     aura  1363
## 10      l2  1311
## # ... with 11,604 more rows
```

What about the descriptions?

```
nasa_desc %>%
  count(word, sort = TRUE)
```

```
## # A tibble: 35,940 × 2
##          word     n
##         <chr> <int>
## 1        data 68871
## 2       modis 24420
## 3      global 23028
## 4           2 16599
## 5           1 15770
## 6      system 15480
## 7     product 14780
## 8        aqua 14738
## 9       earth 14373
## 10 resolution 13879
## # ... with 35,930 more rows
```

Words like “data” and “global” are used very often in NASA titles and descriptions. We may want to remove digits and some “words” like “v1” from these data frames for many types of analyses; they are not too meaningful for most audiences. We can do this by making a list of custom stop words and using `anti_join()` to remove them from the data frame, just like we removed the default stop words that are in the tidytext package. This approach can be used in many instances and is a great tool to bear in mind.

```
my_stopwords <- data_frame(word = c(as.character(1:10),
                                    "v1", "v03", "l2", "l3", "l4", "v5.2.0",
                                    "v003", "v004", "v005", "v006", "v7"))
nasa_title <- nasa_title %>%
  anti_join(my_stopwords)
nasa_desc <- nasa_desc %>%
  anti_join(my_stopwords)
```

What are the most common keywords?

```
nasa_keyword %>%
  group_by(keyword) %>%
  count(sort = TRUE)
```

```
## # A tibble: 1,774 × 2
##                   keyword     n
##                     <chr> <int>
## 1            EARTH SCIENCE 14362
## 2                  Project  7452
## 3               ATMOSPHERE  7321
## 4              Ocean Color  7268
## 5             Ocean Optics  7268
## 6                   Oceans  7268
## 7                completed  6452
## 8  ATMOSPHERIC WATER VAPOR  3142
## 9                   OCEANS  2765
## 10            LAND SURFACE  2720
## # ... with 1,764 more rows
```

Many NASA datasets have **Project completed** as a set of keywords. We likely want to change all of the keywords to either lower or upper case to get rid of duplicates like **OCEANS** and **Oceans**. Let's do that here.

```
nasa_keyword <- nasa_keyword %>%
  mutate(keyword = toupper(keyword))
```

9.2 Word co-ocurrences and correlations

As a next step, let's examine which words commonly occur together in the titles, descriptions, and keywords of NASA datasets, as described in Chapter 5. We can then examine word networks for these fields; this may help us see, for example, which datasets are related to each other.

9.2.1 Networks of Description and Title Words

We can use `pairwise_count()` from the widyr package to count how many times each pair of words occurs together in a title or description field.

```
library(widyr)

title_words <- nasa_title %>%
  pairwise_count(word, id, sort = TRUE, upper = FALSE)

title_words
```

```
## # A tibble: 156,689 × 3
##     item1   item2     n
##     <chr>   <chr> <dbl>
## 1  system project   796
## 2     lba     eco   683
## 3    airs    aqua   641
## 4   level    aqua   623
## 5   level    airs   612
## 6    aura     omi   607
## 7  global    grid   597
## 8  global   daily   574
## 9    data  boreas   551
## 10 ground     gpm   550
## # ... with 156,679 more rows
```

These are the pairs of words that occur together most often in title fields. Some of these words are obviously acronyms used within NASA, and we see how often words like "project" and "system" are used.

```
desc_words <- nasa_desc %>%
  pairwise_count(word, id, sort = TRUE, upper = FALSE)

desc_words
```

```
## # A tibble: 10,889,084 × 3
##         item1      item2     n
##         <chr>      <chr> <dbl>
## 1        data     global  9864
## 2        data resolution  9302
## 3  instrument resolution  8189
## 4        data    surface  8180
## 5      global resolution  8139
## 6        data instrument  7994
## 7        data     system  7870
## 8  resolution      bands  7584
## 9        data      earth  7576
## 10      orbit resolution  7462
## # ... with 10,889,074 more rows
```

These are the pairs of words that occur together most often in descripton fields. "Data" is a very common word in description fields; there is no shortage of data in the datasets at NASA!

Let's plot networks of these co-occurring words so we can see these relationships better in Figure 9.1. We will again use the ggraph package for visualizing our networks.

```r
library(ggplot2)
library(igraph)
library(ggraph)

set.seed(1234)
title_words %>%
  filter(n >= 250) %>%
  graph_from_data_frame() %>%
  ggraph(layout = "fr") +
  geom_edge_link(aes(edge_alpha = n, edge_width = n)) +
  geom_node_point(color = "darkslategray4", size = 5) +
  geom_node_text(aes(label = name), repel = TRUE) +
  theme_void()
```

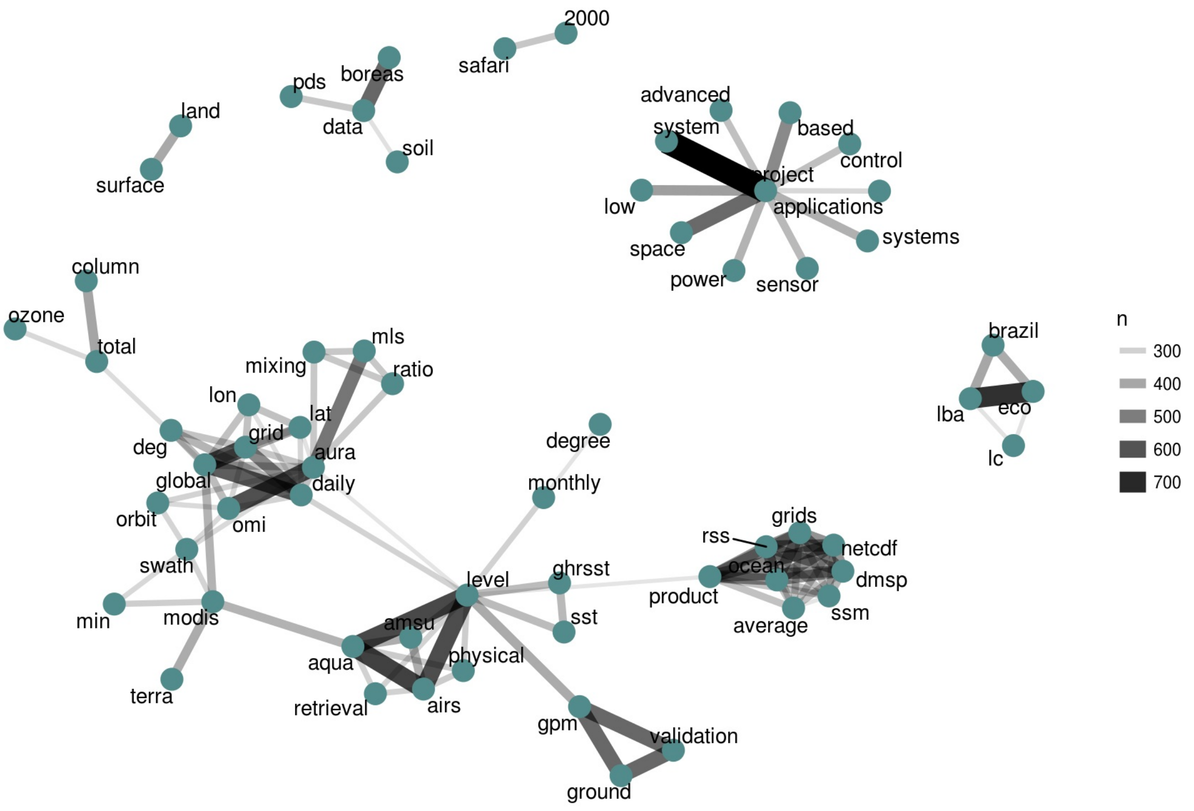


Figure 9.1: Word network in NASA dataset titles

We see some clear clustering in this network of title words; words in NASA dataset titles are largely organized into several families of words that tend to go together.

What about the words from the description fields?

```
set.seed(1234)
desc_words %>%
  filter(n >= 5000) %>%
  graph_from_data_frame() %>%
  ggraph(layout = "fr") +
  geom_edge_link(aes(edge_alpha = n, edge_width = n)) +
  geom_node_point(color = "indianred4", size = 5) +
  geom_node_text(aes(label = name), repel = TRUE) +
  theme_void()
```

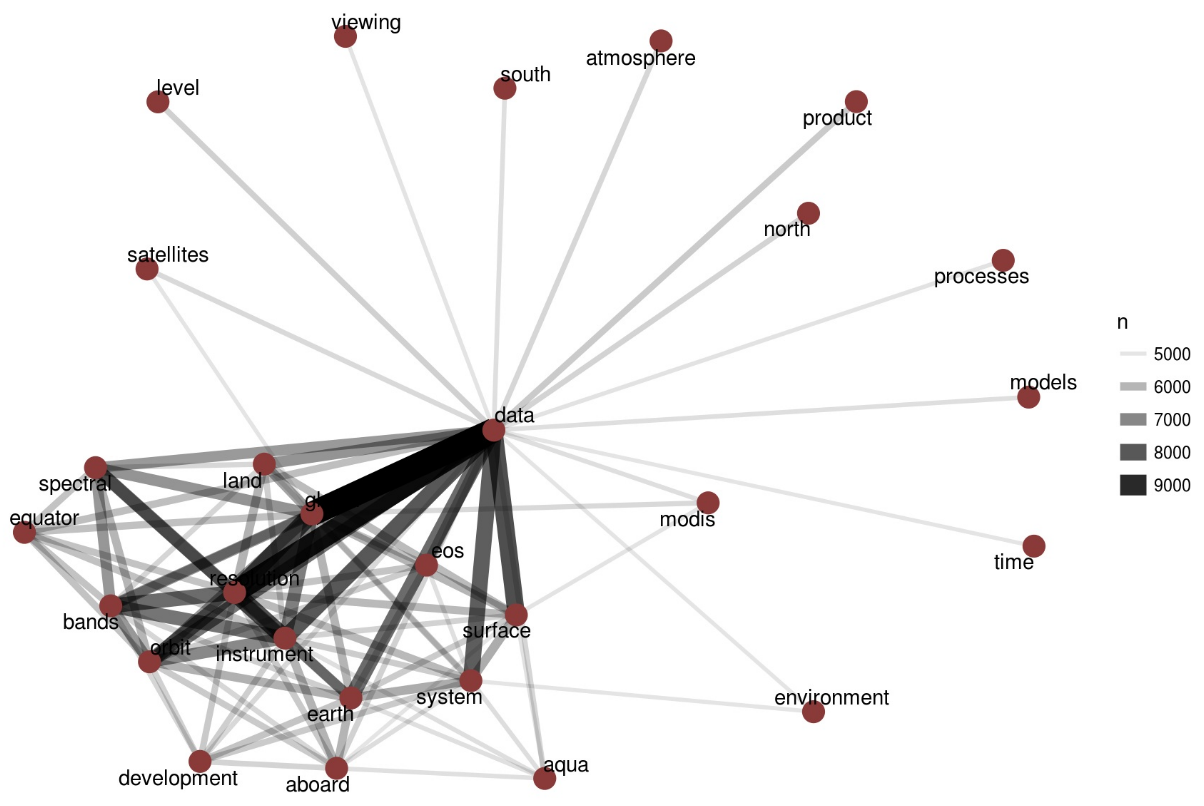


Figure 9.2: Word network in NASA dataset descriptions

Figure 9.2 shows such *strong* connections between the top dozen or so words (words like “data”, “global”, “resolution”, and “instrument”) that we do not see clear clustering structure in the network. We may want to use tf-idf (as described in detail in Chapter 4) as a metric to find characteristic words for each description field, instead of looking at counts of words.

9.2.2 Networks of Keywords

Next, let’s make a network of the keywords to see which keywords commonly occur together in the same datasets.

```
keyword_counts <- nasa_keyword %>%
  pairwise_count(keyword, id, sort = TRUE, upper = FALSE)

keyword_counts
```

```
## # A tibble: 13,390 × 3
##            item1                   item2     n
##            <chr>                   <chr> <dbl>
## 1         OCEANS            OCEAN OPTICS  7324
## 2  EARTH SCIENCE              ATMOSPHERE  7318
## 3         OCEANS             OCEAN COLOR  7270
## 4   OCEAN OPTICS             OCEAN COLOR  7270
## 5        PROJECT               COMPLETED  6450
## 6  EARTH SCIENCE ATMOSPHERIC WATER VAPOR  3142
## 7     ATMOSPHERE ATMOSPHERIC WATER VAPOR  3142
## 8  EARTH SCIENCE                  OCEANS  2762
## 9  EARTH SCIENCE            LAND SURFACE  2718
## 10 EARTH SCIENCE               BIOSPHERE  2448
## # ... with 13,380 more rows
```

```
set.seed(1234)
keyword_counts %>%
  filter(n >= 700) %>%
  graph_from_data_frame() %>%
  ggraph(layout = "fr") +
  geom_edge_link(aes(edge_alpha = n, edge_width = n)) +
  geom_node_point(color = "royalblue3", size = 5) +
  geom_node_text(aes(label = name), repel = TRUE) +
  theme_void()
```

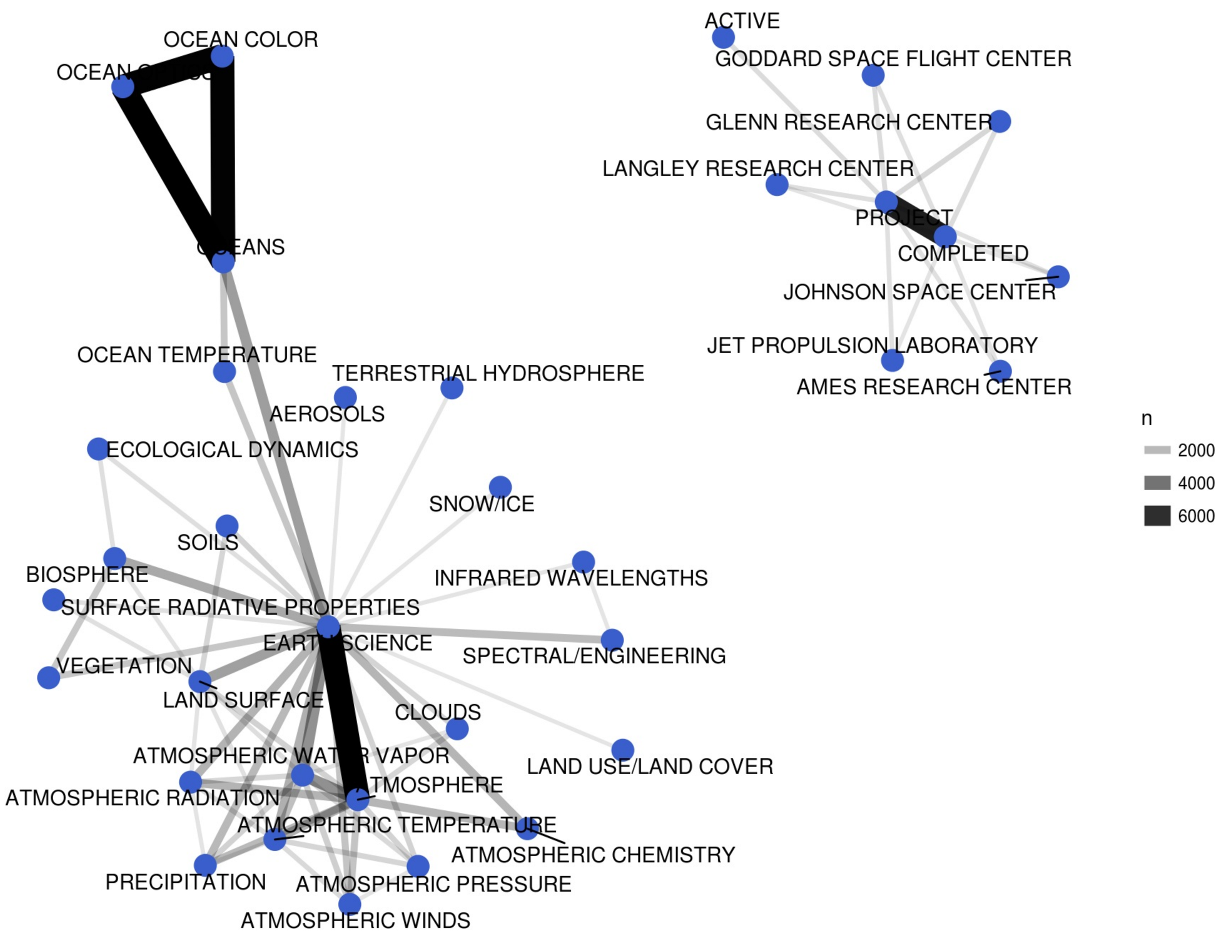


Figure 9.3: Co-occurrence network in NASA dataset keywords

We definitely see clustering here, and strong connections between keywords like **OCEANS**, **OCEAN OPTICS**, and **OCEAN COLOR**, or **PROJECT* and *COMPLETED**. These are the most commonly co-occurring words, but also just the most common keywords in general. To examine the relationships among keywords in a different way, we can find the correlation among the keywords as described in Chapter 5. This looks for those keywords that are more likely to occur together than with other keywords in a description field.

```
keyword_cors <- nasa_keyword %>%
  group_by(keyword) %>%
  filter(n() >= 50) %>%
  pairwise_cor(keyword, id, sort = TRUE, upper = FALSE)

keyword_cors
```

```
## # A tibble: 7,875 × 3
##                   item1       item2 correlation
##                   <chr>       <chr>       <dbl>
## 1             KNOWLEDGE     SHARING   1.0000000
## 2              DASHLINK        AMES   1.0000000
## 3              SCHEDULE  EXPEDITION   1.0000000
## 4            TURBULENCE      MODELS   0.9971871
## 5                 APPEL   KNOWLEDGE   0.9967945
## 6                 APPEL     SHARING   0.9967945
## 7          OCEAN OPTICS OCEAN COLOR   0.9952123
## 8  ATMOSPHERIC SCIENCE       CLOUD   0.9938681
## 9                LAUNCH    SCHEDULE   0.9837078
## 10               LAUNCH  EXPEDITION   0.9837078
## # ... with 7,865 more rows
```

Notice that these keywords at the top of this sorted data frame have correlation coefficients equal to 1; they always occur together. This means these are redundant keywords. It may not make sense to continue to use both of the keywords in these sets of pairs; instead, just one keyword could be used.

Let’s visualize the network of keyword correlations, just as we did for keyword co-occurences.

```r
set.seed(1234)
keyword_cors %>%
  filter(correlation > .6) %>%
  graph_from_data_frame() %>%
  ggraph(layout = "fr") +
  geom_edge_link(aes(edge_alpha = correlation, edge_width = correlation)) +
  geom_node_point(color = "royalblue3", size = 5) +
  geom_node_text(aes(label = name), repel = TRUE) +
  theme_void()
```

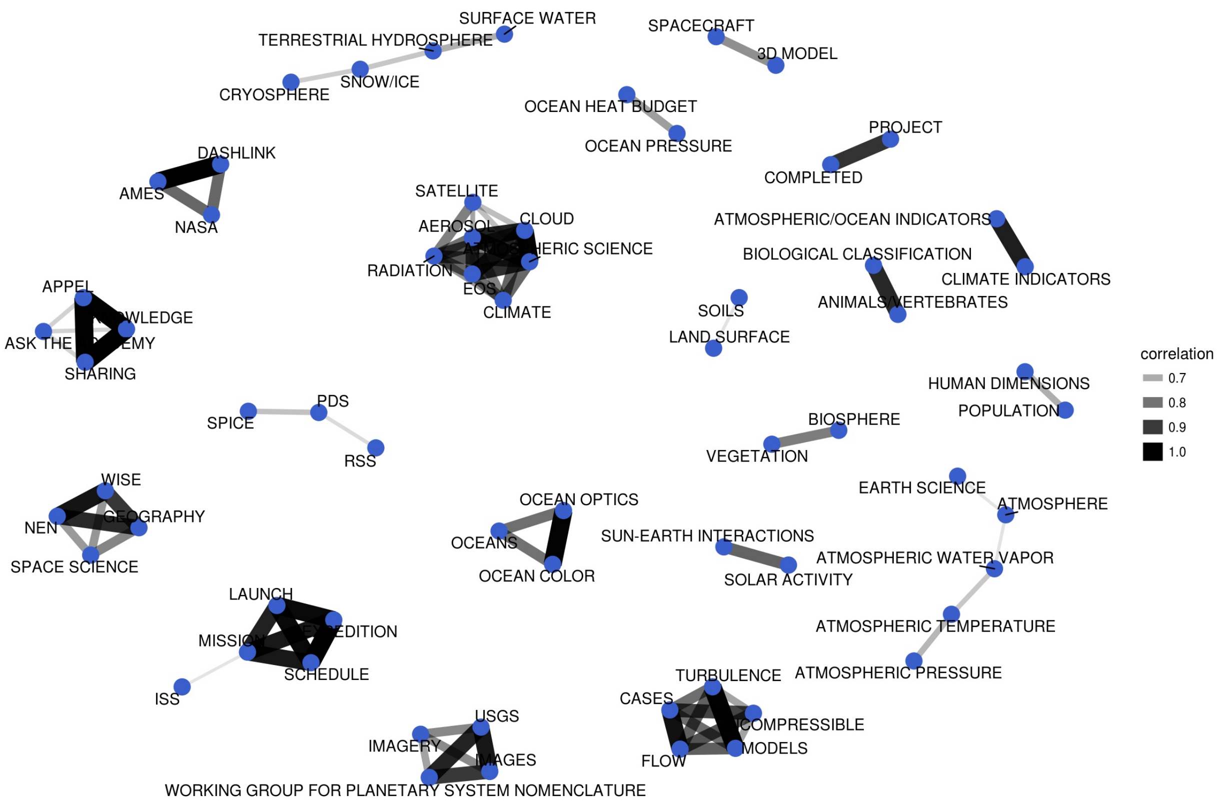


Figure 9.4: Correlation network in NASA dataset keywords

This network appears much different than the co-occurence network. The difference is that the co-occurrence network asks a question about which keyword pairs occur most often, and the correlation network asks a question about which keywordsoccur more often together than with other keywords. Notice here the high number of small clusters of keywords; the network structure can be extracted (for further analysis) from the `graph_from_data_frame()` function above.

9.3 Calculating tf-idf for the description fields

The network graph in Figure 9.2 showed us that the description fields are dominated by a few common words like "data", "global", and "resolution"; this would be an excellent opportunity to use tf-idf as a statistic to find characteristic words for individual description fields. As discussed in Chapter 4, we can use tf-idf, the term frequency times inverse document frequency, to identify words that are especially important to a document within a collection of documents. Let's apply that approach to the description fields of these NASA datasets.

9.3.1 What is tf-idf for the description field words?

We will consider each description field a document, and the whole set of description fields the collection or corpus of documents. We have already used `unnest_tokens()` earlier in this chapter to make a tidy data frame of the words in the description fields, so now we can use `bind_tf_idf()` to calculate tf-idf for each word.

```
desc_tf_idf <- nasa_desc %>%
  count(id, word, sort = TRUE) %>%
  ungroup() %>%
  bind_tf_idf(word, id, n)
```

What are the highest tf-idf words in the NASA description fields?

```
desc_tf_idf %>%
  arrange(-tf_idf)
```

```
## # A tibble: 1,913,224 × 6
##                          id                                          word     n    tf     <
##                       <chr>                                         <chr> <int> <dbl>     <
## 1  55942a7cc63a7fe59b49774a                                           rdr     1     1 10.37
## 2  55942ac9c63a7fe59b49b688 palsar_radiometric_terrain_corrected_high_res     1     1 10.37
## 3  55942ac9c63a7fe59b49b689  palsar_radiometric_terrain_corrected_low_res     1     1 10.37
## 4  55942a7bc63a7fe59b4976ca                                          lgrs     1     1  8.76
## 5  55942a7bc63a7fe59b4976d2                                          lgrs     1     1  8.76
## 6  55942a7bc63a7fe59b4976e3                                          lgrs     1     1  8.76
## 7  55942a7dc63a7fe59b497820                                           mri     1     1  8.58
## 8  55942ad8c63a7fe59b49cf6c                      template_proddescription     1     1  8.29
## 9  55942ad8c63a7fe59b49cf6d                      template_proddescription     1     1  8.29
## 10 55942ad8c63a7fe59b49cf6e                      template_proddescription     1     1  8.29
## # ... with 1,913,214 more rows, and 1 more variables: tf_idf <dbl>
```

These are the most important words in the description fields as measured by tf-idf, meaning they are common but not too common. Notice we have run into an issue here; both n and term frequency are equal to 1 for these terms, meaning that these were description fields that only had a single word in them. If a description field only contains one word, the tf-idf algorithm will think that is a very important word. Depending on our analytic goals, it might be a good idea to throw out all description fields that have very few words.

9.3.2 Connecting description fields to keywords

We now know which words in the descriptions have high tf-idf, and we also have labels for these descriptions in the keywords. Let's do a full join of the keyword data frame and the data frame of description words with tf-idf, and then find the highest tf-idf words for a given keyword.

```
desc_tf_idf <- full_join(desc_tf_idf, nasa_keyword, by = "id")
```

Let's plot some of the most important words, as measured by tf-idf, for a few example keywords used on NASA datasets. First, let's use dplyr operations to filter for the keywords we want to examine and take just the top 15 words for each keyword. Then, let's plot those words in Figure 9.5.

```
desc_tf_idf %>%
  filter(!near(tf, 1)) %>%
  filter(keyword %in% c("SOLAR ACTIVITY", "CLOUDS",
                        "SEISMOLOGY", "ASTROPHYSICS",
                        "HUMAN HEALTH", "BUDGET")) %>%
  arrange(desc(tf_idf)) %>%
  group_by(keyword) %>%
  distinct(word, keyword, .keep_all = TRUE) %>%
  top_n(15, tf_idf) %>%
  ungroup() %>%
  mutate(word = factor(word, levels = rev(unique(word)))) %>%
  ggplot(aes(word, tf_idf, fill = keyword)) +
  geom_bar(stat = "identity", alpha = 0.8, show.legend = FALSE) +
  facet_wrap(~keyword, ncol = 3, scales = "free") +
  coord_flip() +
  labs(title = "Highest tf-idf words in NASA metadata description fields",
       caption = "NASA metadata from https://data.nasa.gov/data.json",
       x = NULL, y = "tf-idf")
```

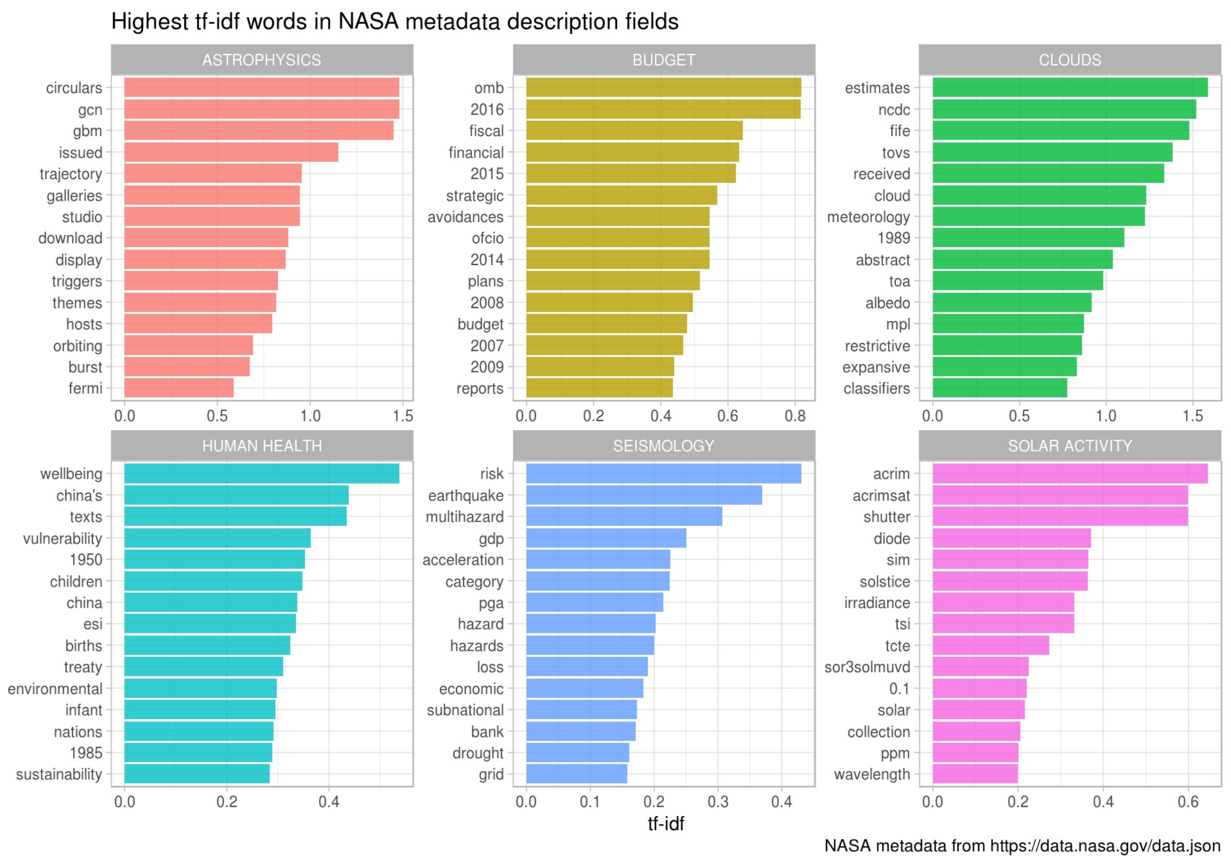


Figure 9.5: Distribution of tf-idf for words from datasets labeled with select keywords

Using tf-idf has allowed us to identify important description words for each of these keywords. Datasets labeled with the keyword **SEISMOLOGY** have words like “earthquake”, “risk”, and “hazard” in their description, while those labeled with **HUMAN HEALTH** have descriptions characterized by words like “wellbeing”, “vulnerability”, and “children.” Most of the combinations of letters that are not English words are certainly acronyms (like OMB for the Office of Management and Budget), and the examples of years and numbers are important for these topics. The tf-idf statistic has identified the kinds of words it is intended to, important words for individual documents within a collection of documents.

9.4 Topic modeling

Using tf-idf as a statistic has already given us insight into the content of NASA description fields, but let's try an additional approach to the question of what the NASA descriptions fields are about. We can use topic modeling as described in Chapter 7 to model each document (description field) as a mixture of topics and each topic as a mixture of words. As in earlier chapters, we will use latent Dirichlet allocation (LDA) for our topic modeling; there are other possible approaches for topic modeling.

9.4.1 Casting to a document-term matrix

To do the topic modeling as implemented here, we need to make a `DocumentTermMatrix`, a special kind of matrix from the tm package (of course, this is just a specific implementation of the general concept of a "document-term matrix"). Rows correspond to documents (description texts in our case) and columns correspond to terms (i.e., words); it is a sparse matrix and the values are word counts.

Let's clean up the text a bit using stop words to remove some of the nonsense "words" leftover from HTML or other character encoding. We can use `bind_rows()` to add our custom stop words to the list of default stop words from the tidytext package, and then all at once use `anti_join()` to remove them all from our data frame.

```
my_stop_words <- bind_rows(stop_words,
                           data_frame(word = c("nbsp", "amp", "gt", "lt",
                                               "timesnewromanpsmt", "font",
                                               "td", "li", "br", "tr", "quot",
                                               "st", "img", "src", "strong",
                                               "http", "file", "files",
                                               as.character(1:12)),
                                      lexicon = rep("custom", 30)))

word_counts <- nasa_desc %>%
  anti_join(my_stop_words) %>%
  count(id, word, sort = TRUE) %>%
  ungroup()

word_counts
```

```
## # A tibble: 1,895,310 × 3
##                           id     word     n
##                        <chr>    <chr> <int>
## 1  55942a8ec63a7fe59b4986ef     suit    82
## 2  55942a8ec63a7fe59b4986ef    space    69
## 3  56cf5b00a759fdadc44e564a     data    41
## 4  56cf5b00a759fdadc44e564a     leak    40
## 5  56cf5b00a759fdadc44e564a     tree    39
## 6  55942a8ec63a7fe59b4986ef pressure    34
## 7  55942a8ec63a7fe59b4986ef   system    34
## 8  55942a89c63a7fe59b4982d9       em    32
## 9  55942a8ec63a7fe59b4986ef       al    32
## 10 55942a8ec63a7fe59b4986ef    human    31
## # ... with 1,895,300 more rows
```

This is the information we need, the number of times each word is used in each document, to make a `DocumentTermMatrix`. We can `cast()` from our tidy text format to this non-tidy format as described in detail in Chapter 6.

```
desc_dtm <- word_counts %>%
  cast_dtm(id, word, n)

desc_dtm
```

```
## <<DocumentTermMatrix (documents: 32003, terms: 35901)>>
## Non-/sparse entries: 1895310/1147044393
## Sparsity           : 100%
## Maximal term length: 166
## Weighting          : term frequency (tf)
```

We see that this dataset contains documents (each of them a NASA description field) and terms (words). Notice that this example document-term matrix is (very close to) 100% sparse, meaning that almost all of the entries in this matrix are zero. There are non-zero entries everywhere in the matrix that corresponds to a certain word appearing in a certain document.

9.4.2 Ready for topic modeling

Now let’s use the topicmodels package to create an LDA model. How many topics will we tell the algorithm to make? This is a question much like in k-means clustering; we don’t really know ahead of time. We tried the following modeling procedure using 8, 16, 24, 32, and 64 topics; we found that at 24 topics, documents are still getting sorted into topics cleanly but going much beyond that caused the distributions of γ, the probability that each document belongs in each topic, to look worrisome. We will show more details on this later.

```r
library(topicmodels)

desc_lda <- LDA(desc_dtm, k = 24, control = list(seed = 1234))
desc_lda
```

```
## A LDA_VEM topic model with 24 topics.
```

This is a stochastic algorithm that could have different results depending on where the algorithm starts, so we need to specify a `seed` for reproducibility as shown here.

9.4.3 Interpreting the topic model

Now that we have built the model, let's `tidy()` the results of the model, i.e., construct a tidy data frame that summarizes the results of the model. The tidytext package includes a tidying method for LDA models from the topicmodels package.

```r
tidy_lda <- tidy(desc_lda)

tidy_lda
```

```
## # A tibble: 861,624 × 3
##    topic  term          beta
##    <int> <chr>         <dbl>
## 1      1  suit 1.003981e-121
## 2      2  suit 2.630614e-145
## 3      3  suit  1.916240e-79
## 4      4  suit  6.715725e-45
## 5      5  suit  1.738334e-85
## 6      6  suit  7.692116e-84
## 7      7  suit  3.283851e-04
## 8      8  suit  3.738586e-20
## 9      9  suit  4.846953e-15
## 10    10  suit  4.765471e-10
## # ... with 861,614 more rows
```

The column β tells us the probability of that term being generated from that topic for that document. It is the probability of that term (word) belonging to that topic. Notice that some of the values for β are very, very low, and some are not so low.

What is each topic about? Let's examine the top 10 terms for each topic.

```
top_terms <- tidy_lda %>%
  group_by(topic) %>%
  top_n(10, beta) %>%
  ungroup() %>%
  arrange(topic, -beta)

top_terms
```

```
## # A tibble: 240 × 3
##    topic        term       beta
##    <int>       <chr>      <dbl>
## 1      1        data 0.04488960
## 2      1        soil 0.03676198
## 3      1    moisture 0.02954555
## 4      1        amsr 0.02437751
## 5      1         sst 0.01684001
## 6      1  validation 0.01322457
## 7      1 temperature 0.01317075
## 8      1     surface 0.01290046
## 9      1    accuracy 0.01225131
## 10     1         set 0.01155372
## # ... with 230 more rows
```

It is not very easy to interpret what the topics are about from a data frame like this so let's look at this information visually in Figure 9.6.

```
top_terms %>%
  mutate(term = reorder(term, beta)) %>%
  group_by(topic, term) %>%
  arrange(desc(beta)) %>%
  ungroup() %>%
  mutate(term = factor(paste(term, topic, sep = "__"), levels = rev(paste(term, topic, sep =
"__")))) %>%
  ggplot(aes(term, beta, fill = as.factor(topic))) +
  geom_bar(stat = "identity", show.legend = FALSE, alpha = 0.8) +
  coord_flip() +
  scale_x_discrete(labels = function(x) gsub("__.+$", "", x)) +
  labs(title = "Top 10 terms in each LDA topic",
       x = NULL, y = expression(beta)) +
  facet_wrap(~ topic, ncol = 4, scales = "free")
```

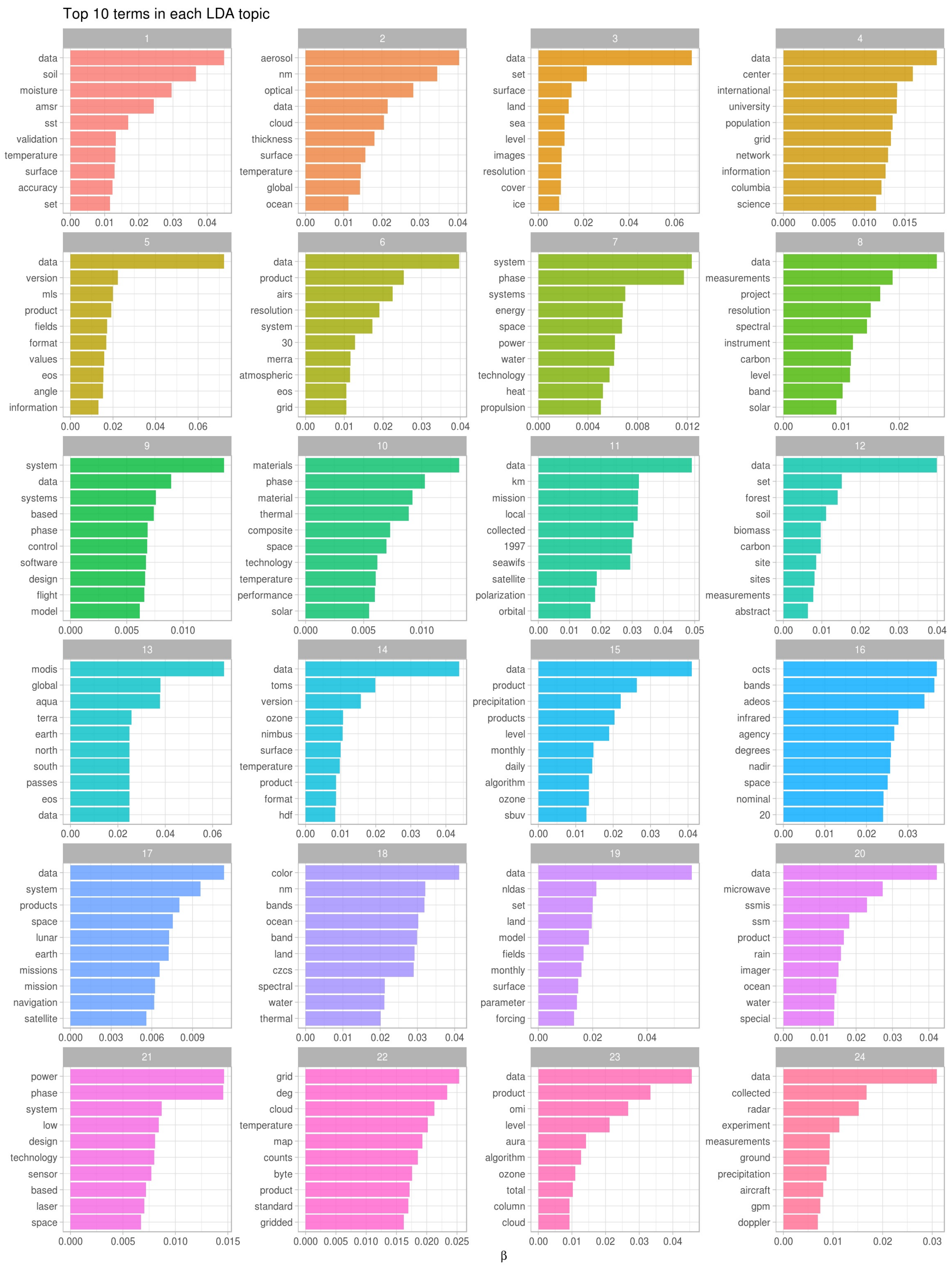


Figure 9.6: Top terms in topic modeling of NASA metadata description field texts

We can see what a dominant word "data" is in these description texts. In addition, there are meaningful differences between these collections of terms, from terms about soil, forests, and biomass in topic 12 to terms about design, systems, and technology in topic 21. The topic modeling process has identified groupings of terms that we can understand as human readers of these description fields.

We just explored which words are associated with which topics. Next, let's examine which topics are associated with which description fields (i.e., documents). We will look at a different probability for this, γ, the probability that each document belongs in each topic, again using the `tidy` verb.

```
lda_gamma <- tidy(desc_lda, matrix = "gamma")

lda_gamma
```

```
## # A tibble: 768,072 × 3
##                     document topic        gamma
##                        <chr> <int>        <dbl>
## 1  55942a8ec63a7fe59b4986ef     1 6.453820e-06
## 2  56cf5b00a759fdadc44e564a     1 1.158393e-05
## 3  55942a89c63a7fe59b4982d9     1 4.917441e-02
## 4  56cf5b00a759fdadc44e55cd     1 2.249043e-05
## 5  55942a89c63a7fe59b4982c6     1 6.609442e-05
## 6  55942a86c63a7fe59b498077     1 5.666520e-05
## 7  56cf5b00a759fdadc44e56f8     1 4.752082e-05
## 8  55942a8bc63a7fe59b4984b5     1 4.308534e-05
## 9  55942a6ec63a7fe59b496bf7     1 4.408626e-05
## 10 55942a8ec63a7fe59b4986f6     1 2.878188e-05
## # ... with 768,062 more rows
```

Notice that some of the probabilites visible at the top of the data frame are low and some are higher. Our model has assigned a probability to each description belonging to each of the topics we constructed from the sets of words. How are the probabilities distributed? Let's visualize them.

```
ggplot(lda_gamma, aes(gamma, fill = as.factor(topic))) +
  geom_histogram(show.legend = FALSE, alpha = 0.8) +
  facet_wrap(~ topic, ncol = 4) +
  scale_y_log10() +
  labs(title = "Distribution of probability for each topic",
       y = "Number of documents", x = expression(gamma))
```

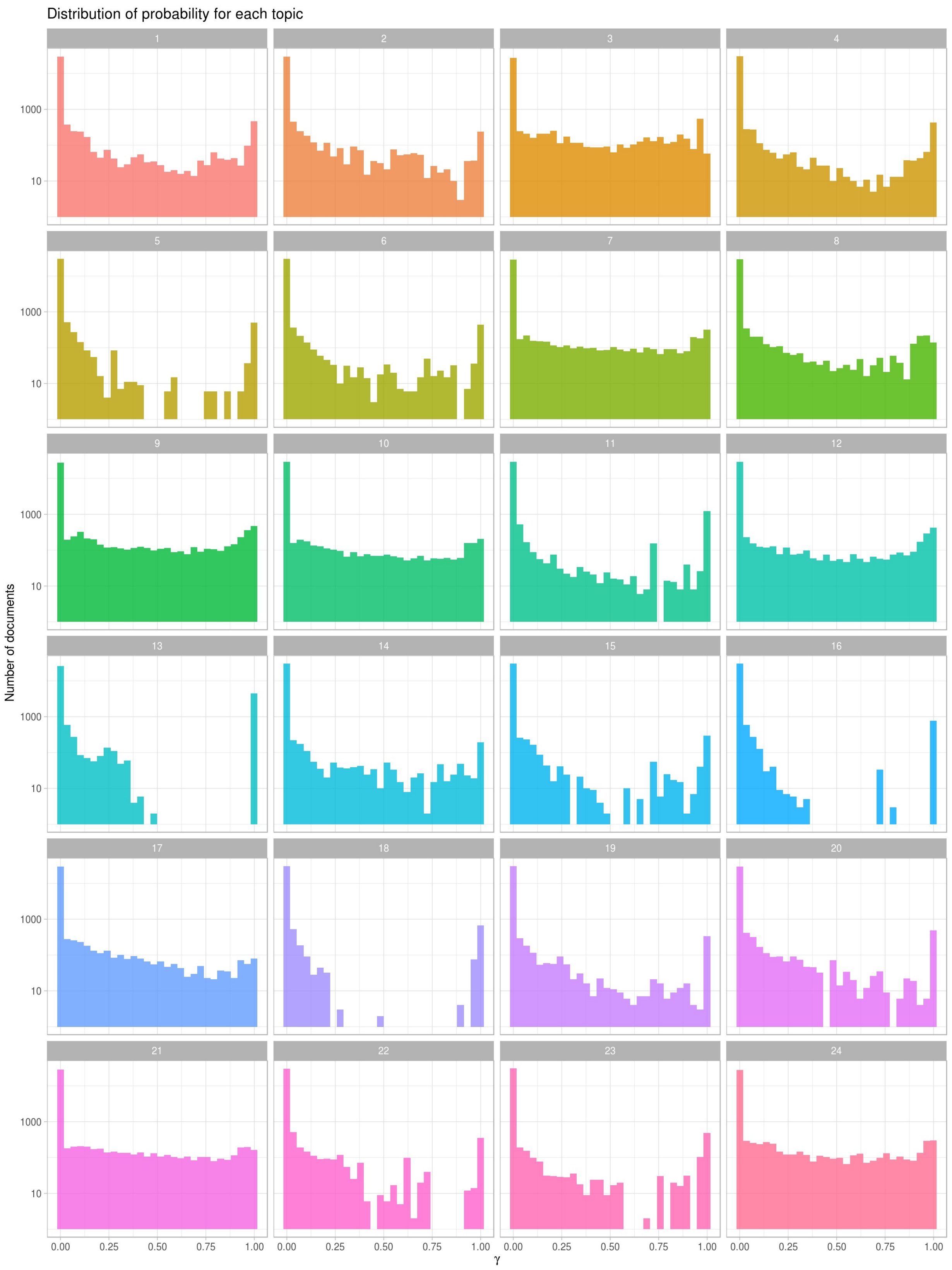


Figure 9.7: Probability distribution in topic modeling of NASA metadata description field texts

First notice that the y-axis is plotted on a log scale; otherwise it is difficult to make out any detail in the plot. Next, notice that γ runs from 0 to 1 in each panel. Remember that this is the probability that a given document belongs in a given topic. There are many values near zero, which means there are many documents that do not belong in each topic. Also, most of these panels show a higher number of documents near $\gamma = 1$; these are the documents that *do* belong in those topics. For example, let's look specifically at topic 18 in Figure 9.7, a topic that had documents cleanly sorted in and out of it. There are many documents with γ close to 1; these are the documents that *do* belong to topic 18 according to the model. There are also many documents with γ close to 0; these are the documents that do *not* belong to topic 18. Each document appears in each panel in this plot, and its γ for that topic tells us that document's probability of belonging in that topic.

This plot displays the type of information we used to choose how many topics for our topic modeling procedure. When we tried options higher than 24 (such as 32 or 64), the distributions for γ started to look very flat toward $\gamma = 1$; documents were not getting sorted into topics very well.

9.4.4 Connecting topic modeling with keywords

Let's connect these topic models with the keywords and see what relationships we can find. We can `full_join()` this to the human-tagged keywords and discover which keywords are associated with which topic.

```
lda_gamma <- full_join(lda_gamma, nasa_keyword, by = c("document" = "id"))

lda_gamma
```

```
## # A tibble: 3,037,671 × 4
##                     document topic        gamma                  keyword
##                        <chr> <int>        <dbl>                    <chr>
## 1  55942a8ec63a7fe59b4986ef     1 6.453820e-06      JOHNSON SPACE CENTER
## 2  55942a8ec63a7fe59b4986ef     1 6.453820e-06                   PROJECT
## 3  55942a8ec63a7fe59b4986ef     1 6.453820e-06                 COMPLETED
## 4  56cf5b00a759fdadc44e564a     1 1.158393e-05                  DASHLINK
## 5  56cf5b00a759fdadc44e564a     1 1.158393e-05                      AMES
## 6  56cf5b00a759fdadc44e564a     1 1.158393e-05                      NASA
## 7  55942a89c63a7fe59b4982d9     1 4.917441e-02 GODDARD SPACE FLIGHT CENTER
## 8  55942a89c63a7fe59b4982d9     1 4.917441e-02                   PROJECT
## 9  55942a89c63a7fe59b4982d9     1 4.917441e-02                 COMPLETED
## 10 56cf5b00a759fdadc44e55cd     1 2.249043e-05                  DASHLINK
## # ... with 3,037,661 more rows
```

Now we can use `filter()` to keep only the document-topic entries that have probabilities (γ) greater than some cut-off value; let's use 0.9.

```
top_keywords <- lda_gamma %>%
  filter(gamma > 0.9) %>%
  group_by(topic, keyword) %>%
  count(keyword, sort = TRUE)

top_keywords
```

```
## Source: local data frame [1,022 x 3]
## Groups: topic [24]
##
##    topic       keyword     n
##    <int>         <chr> <int>
## 1     13   OCEAN COLOR  4480
## 2     13  OCEAN OPTICS  4480
## 3     13        OCEANS  4480
## 4     11   OCEAN COLOR  1216
## 5     11  OCEAN OPTICS  1216
## 6     11        OCEANS  1216
## 7      9       PROJECT   926
## 8     12 EARTH SCIENCE   909
## 9      9     COMPLETED   834
## 10    16   OCEAN COLOR   768
## # ... with 1,012 more rows
```

What are the top keywords for each topic?

```
top_keywords %>%
  top_n(5, n) %>%
  group_by(topic, keyword) %>%
  arrange(desc(n)) %>%
  ungroup() %>%
  mutate(keyword = factor(paste(keyword, topic, sep = "__"),
                          levels = rev(paste(keyword, topic, sep = "__")))) %>%
  ggplot(aes(keyword, n, fill = as.factor(topic))) +
  geom_bar(stat = "identity", show.legend = FALSE, alpha = 0.8) +
  labs(title = "Top keywords for each LDA topic",
       x = NULL, y = "Number of documents") +
  coord_flip() +
  scale_x_discrete(labels = function(x) gsub("__.+$", "", x)) +
  facet_wrap(~ topic, ncol = 4, scales = "free")
```

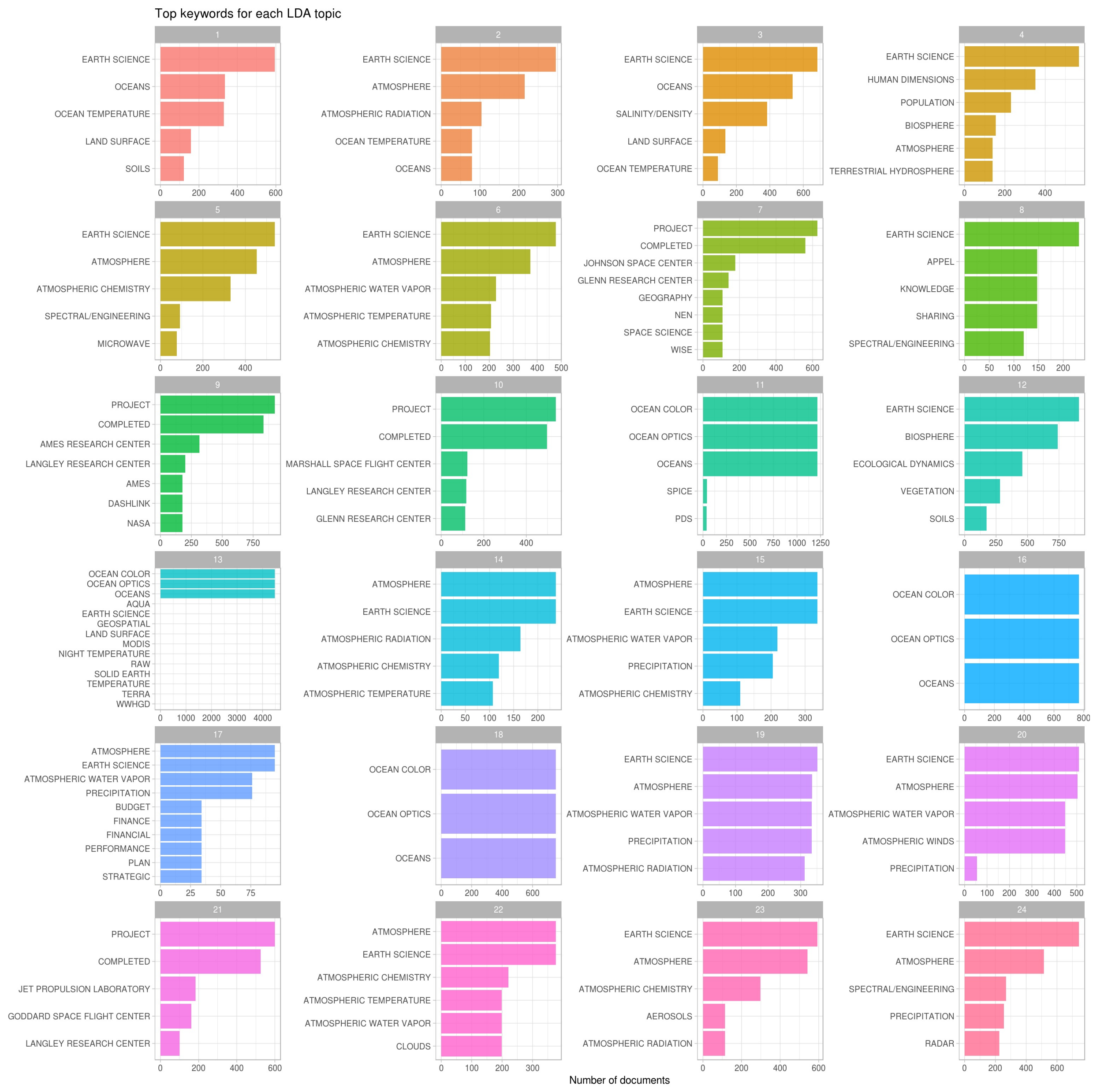


Figure 9.8: Top keywords in topic modeling of NASA metadata description field texts

Let's take a step back and remind ourselves what Figure 9.8 is telling us. NASA datasets are tagged with keywords by human beings, and we have built an LDA topic model (with 24 topics) for the description fields of the NASA datasets. This plot answers the question, "For the datasets with description fields that have a high probability of belonging to a given topic, what are the most common human-assigned keywords?"

It's interesting that the keywords for topics 13, 16, and 18 are essentially duplicates of each other, because the top terms in those topics do exhibit meaningful differences. Also note that by number of documents, the combination of 13, 16, and 18 is quite a large percentage of the total number of datasets represented in this plot, and even more if we were to include topic 11. By number, there are *many* datasets at NASA that deal with oceans, ocean color, and ocean optics. We see **PROJECT COMPLETED** in topics 9, 10, and 21, along with the names of NASA laboratories and research centers. Other important subject areas that stand out are groups of keywords about atmospheric science, budget/finance, and population/human dimensions. We can go back to Figure 9.6 on terms and topics to see which words in the description fields are driving datasets being assigned to these topics. For example, topic 4 is associated with keywords about population and human dimensions, and some of the top terms for that topic are "population", "international", "center", and "university".

By using a combination of network analysis, tf-idf, and topic modeling, we have come to a greater understanding of how datasets are related at NASA. Specifically, we have more information now about how keywords are connected to each other and which datasets are likely to be related. The topic model could be used to suggest keywords based on the words in the description field, or the work on the keywords could suggest the most important combination of keywords for certain areas of study.

10 Case study: analyzing usenet text

In our final chapter, we’ll use what we’ve learned in this book to perform a start-to-finish analysis of a set of 20,000 messages sent to 20 Usenet bulletin boards in 1993. The Usenet bulletin boards in this data set include boards for topics like politics, autos, “for sale”, atheism, etc. This data set is publicly available and has become popular for testing and exercises in text analysis and machine learning.

10.1 Wrangling the data

We’ll start by reading in all the messages. (Note that this step takes several minutes).

```r
library(dplyr)
library(tidyr)
library(purrr)
library(readr)
library(stringr)
```

```r
training_folder <- "data/20news-bydate/20news-bydate-train/"

read_folder <- function(infolder) {
  print(infolder)
  data_frame(file = dir(infolder, full.names = TRUE)) %>%
    mutate(text = map(file, read_lines)) %>%
    transmute(id = basename(file), text) %>%
    unnest(text)
}

raw_text <- data_frame(folder = dir(training_folder, full.names = TRUE)) %>%
  unnest(map(folder, read_folder)) %>%
  transmute(board = basename(folder), id, text)
```

Each email has structure we need to remove. For starters:

- Every email has one or more headers (e.g. “from:”, “in_reply_to:”)
- Many have signatures, which (since they’re constant for each user) we wouldn’t want to examine alongside the content

We need to remove headers and signatures.

```r
# remove headers and signatures
cleaned_text <- raw_text %>%
  group_by(id) %>%
  filter(cumsum(text == "") > 0,
         cumsum(str_detect(text, "^--")) == 0) %>%
  ungroup()

# remove nested text (starting with ">") and lines that note the author
# of those
cleaned_text <- cleaned_text %>%
  filter(str_detect(text, "^[^>]+[A-Za-z\\d]") | text == "",
         !str_detect(text, "writes(:|\\.\\.\\.)$"),
         !str_detect(text, "^In article <"),
         !id %in% c(9704, 9985))
```

Now it is time to use `unnest_tokens` to identify the words in this data set.

```r
library(tidytext)

usenet_words <- cleaned_text %>%
  unnest_tokens(word, text) %>%
  filter(str_detect(word, "^[a-z]"),
         str_detect(word, "[a-z]$"),
         !word %in% stop_words$word)
```

What are the most common words?

```r
usenet_words %>%
  count(word, sort = TRUE)
```

```
## # A tibble: 63,937 × 2
##        word     n
##       <chr> <int>
## 1    people  3397
## 2      time  2569
## 3       god  1611
## 4    system  1571
## 5   subject  1312
## 6     lines  1188
## 7   program  1086
## 8   windows  1085
## 9       bit  1070
## 10    space  1062
## # ... with 63,927 more rows
```

Or perhaps more sensibly, we could examine the most common words by board.

```
words_by_board <- usenet_words %>%
  count(board, word) %>%
  ungroup()
```

```
words_by_board %>%
  group_by(board) %>%
  top_n(3)
```

```
## Source: local data frame [60 x 3]
## Groups: board [20]
##
##                       board     word     n
##                       <chr>    <chr> <int>
## 1               alt.atheism      god   268
## 2               alt.atheism    jesus   129
## 3               alt.atheism   people   276
## 4             comp.graphics graphics   217
## 5             comp.graphics    image   169
## 6             comp.graphics  program   134
## 7   comp.os.ms-windows.misc      dos   194
## 8   comp.os.ms-windows.misc     file   232
## 9   comp.os.ms-windows.misc  windows   625
## 10 comp.sys.ibm.pc.hardware     card   237
## # ... with 50 more rows
```

These look sensible and illuminating so far; let’s move on to some more sophisticated analysis!

10.2 Term frequency and inverse document frequency: tf-idf

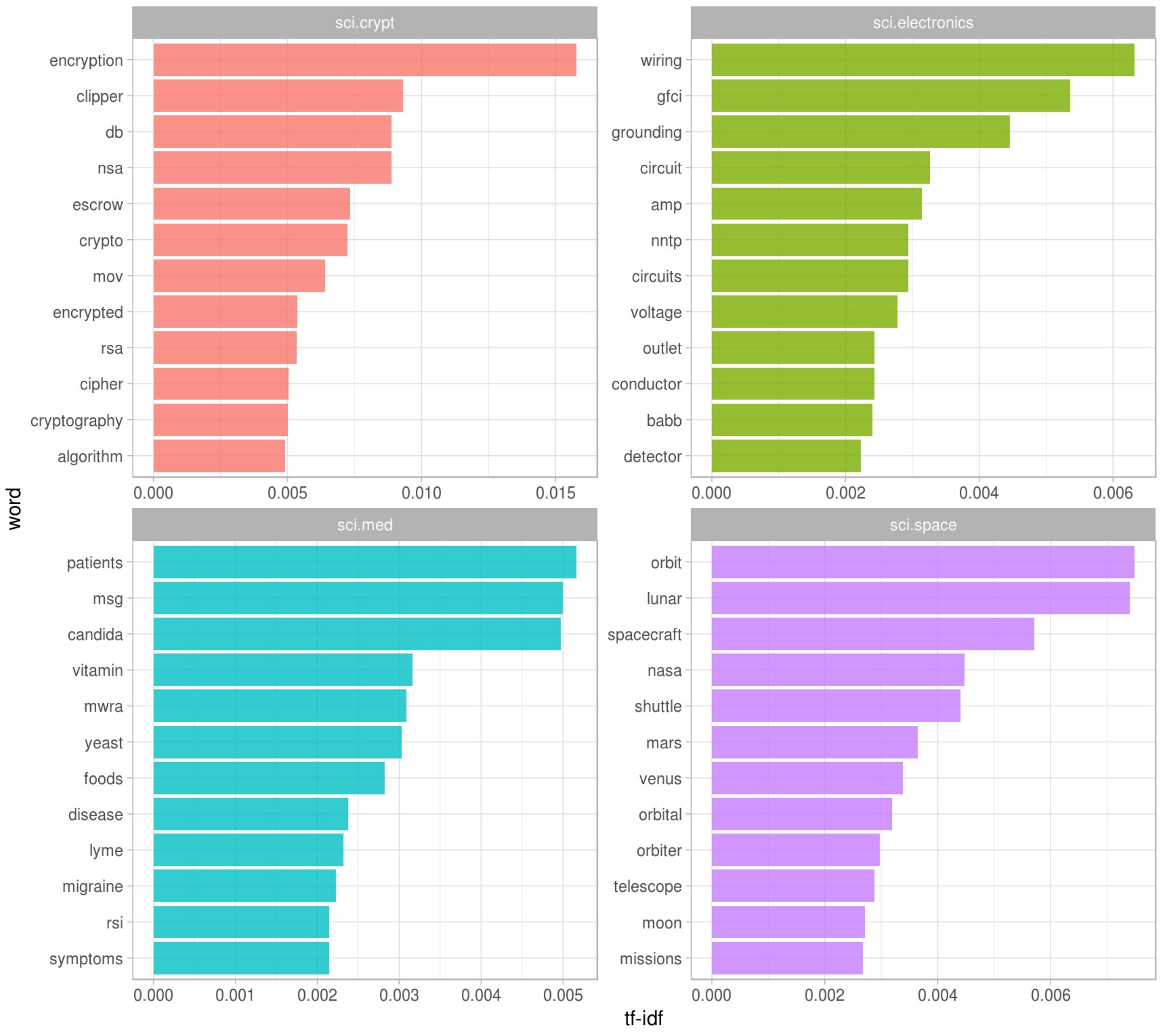
Some words are likely to be more common on particular boards. Let’s try quantifying this using the tf-idf metric we learned in Chapter 4.

```
tf_idf <- words_by_board %>%
  bind_tf_idf(word, board, n) %>%
  arrange(desc(tf_idf))

tf_idf
```

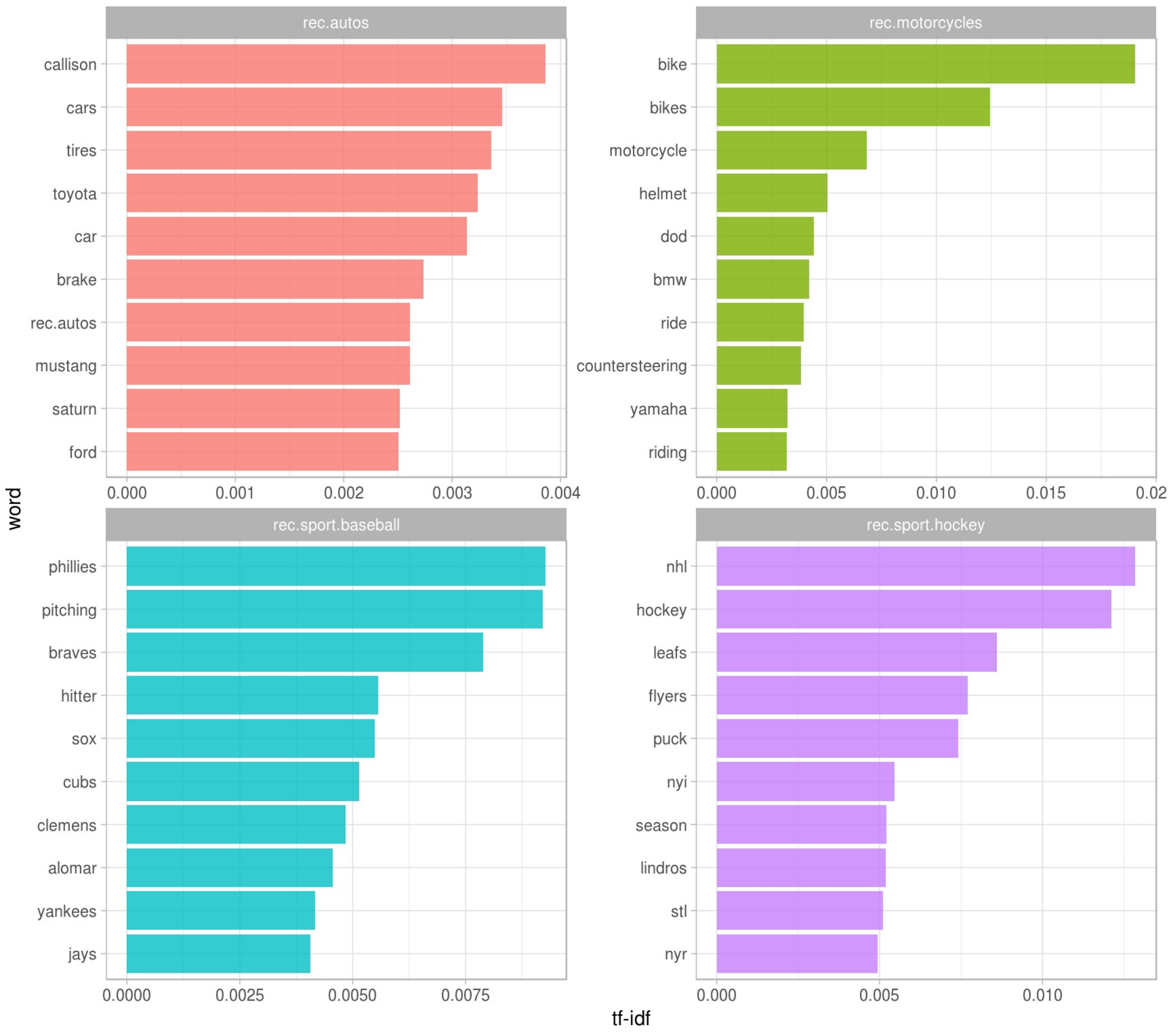
```
## # A tibble: 166,528 × 6
##                       board           word     n          tf      idf     tf_idf
##                       <chr>          <chr> <int>       <dbl>    <dbl>      <dbl>
## 1  comp.sys.ibm.pc.hardware           scsi   483 0.018138801 1.203973 0.02183862
## 2           rec.motorcycles           bike   321 0.013750268 1.386294 0.01906192
## 3     talk.politics.mideast       armenian   440 0.007348275 2.302585 0.01692003
## 4                 sci.crypt     encryption   410 0.008311878 1.897120 0.01576863
## 5     talk.politics.mideast      armenians   396 0.006613447 2.302585 0.01522803
## 6          rec.sport.hockey            nhl   151 0.004291114 2.995732 0.01285503
## 7  comp.sys.ibm.pc.hardware            ide   208 0.007811326 1.609438 0.01257184
## 8        talk.politics.misc stephanopoulos   158 0.004175145 2.995732 0.01250762
## 9           rec.motorcycles          bikes    97 0.004155065 2.995732 0.01244746
## 10         rec.sport.hockey         hockey   265 0.007530762 1.609438 0.01212029
## # ... with 166,518 more rows
```

We can visualize this for a few select boards. First, let’s look at all the `sci.` boards.

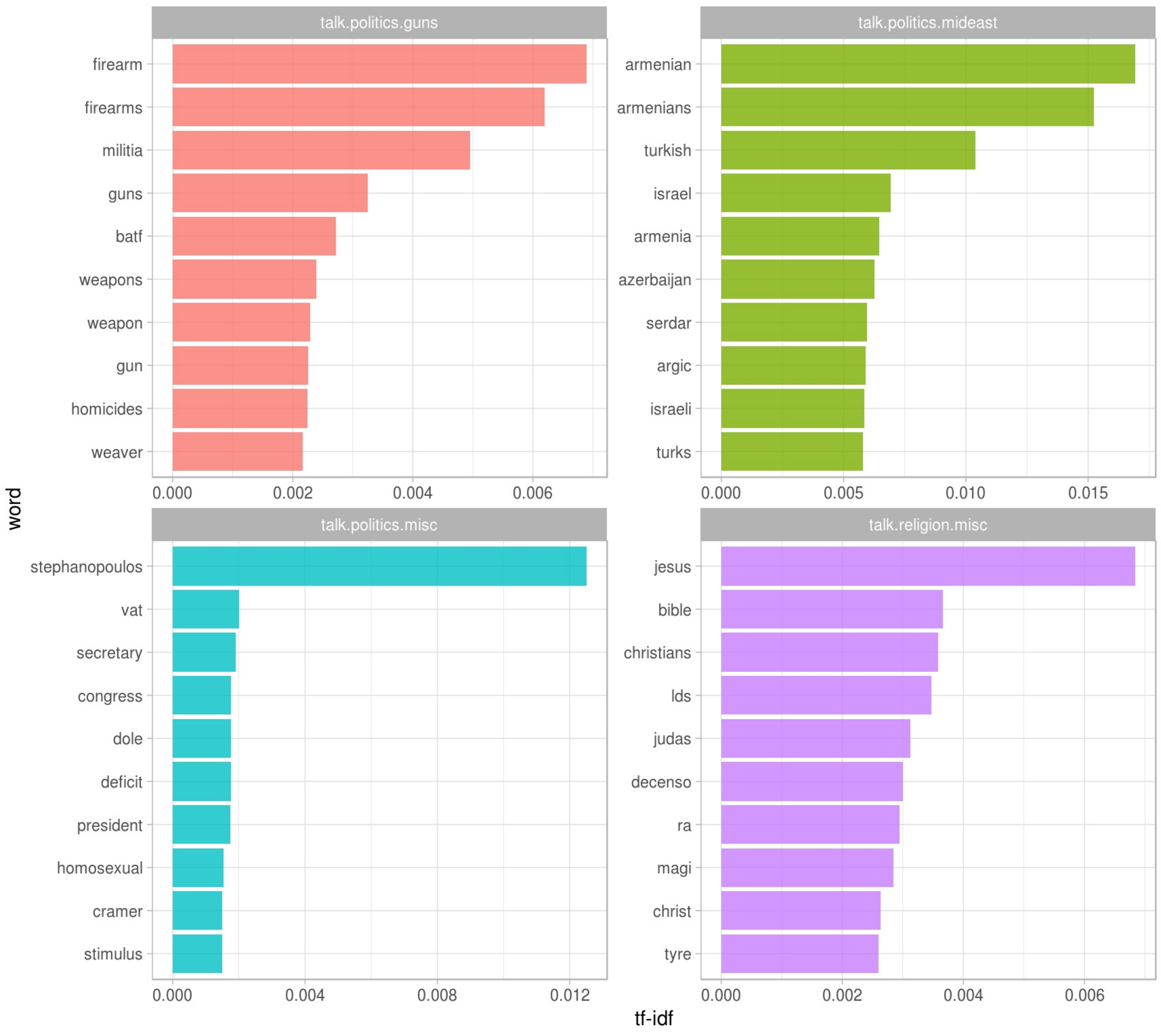
```r
library(ggplot2)

tf_idf %>%
  filter(str_detect(board, "^sci\\.")) %>%
  group_by(board) %>%
  top_n(12, tf_idf) %>%
  mutate(word = reorder(word, tf_idf)) %>%
  ggplot(aes(word, tf_idf, fill = board)) +
  geom_bar(alpha = 0.8, stat = "identity", show.legend = FALSE) +
  facet_wrap(~ board, scales = "free") +
  ylab("tf-idf") +
  coord_flip()
```



We could use almost the same code (not shown) to compare the "rec." (recreation) or "talk." boards:



rec.autos
callison
cars
tires
toyota
car
brake
rec.autos
mustang
saturn
ford
0.000
0.001
0.002
0.003
0.004
rec.motorcycles
bike
bikes
motorcycle
helmet
dod
bmw
ride
countersteering
yamaha
riding
0.000
0.005
0.010
0.015
0.02
rec.sport.baseball
phillies
pitching
braves
hitter
sox
cubs
clemens
alomar
yankees
jays
0.0000
0.0025
0.0050
0.0075
rec.sport.hockey
nhl
hockey
leafs
flyers
puck
nyi
season
lindros
stl
nyr
0.000
0.005
0.010
word
tf-idf

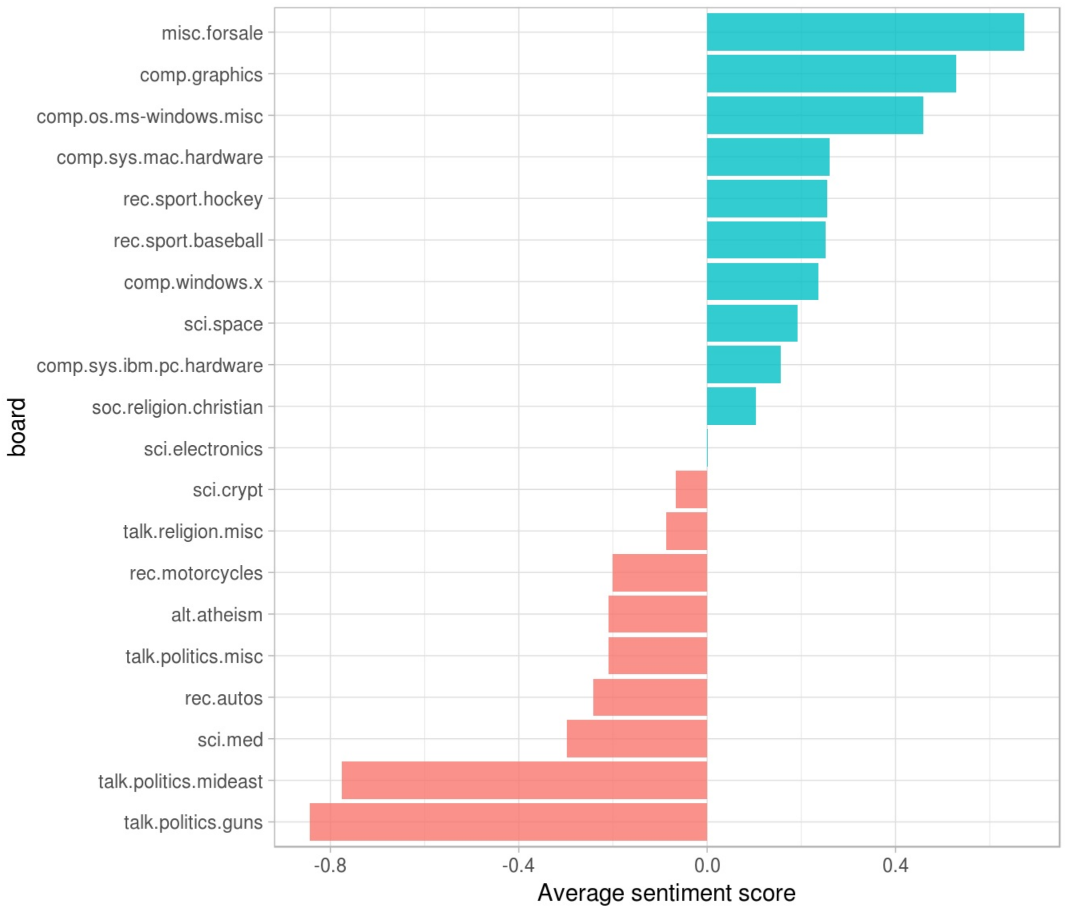


We see lots of characteristic words for these boards, from “pitching” and “hitter” for the baseball board to “firearm” and “militia” on the guns board. Notice how high tf-idf is for words like “Stephanopoulos” or “Armenian”; this means that these words are very unique among the documents as a whole and important to those particular boards.

10.3 Sentiment analysis

We can use the sentiment analysis techniques we explored in Chapter 3 to examine how positive and negative words were used in these Usenet posts. Which boards used the most positive and negative words?

```r
AFINN <- get_sentiments("afinn")

word_board_sentiments <- words_by_board %>%
  inner_join(AFINN, by = "word")

board_sentiments <- word_board_sentiments %>%
  group_by(board) %>%
  summarize(score = sum(score * n) / sum(n))

board_sentiments %>%
  mutate(board = reorder(board, score)) %>%
  ggplot(aes(board, score, fill = score > 0)) +
  geom_bar(alpha = 0.8, stat = "identity", show.legend = FALSE) +
  coord_flip() +
  ylab("Average sentiment score")
```



10.4 Sentiment analysis by word

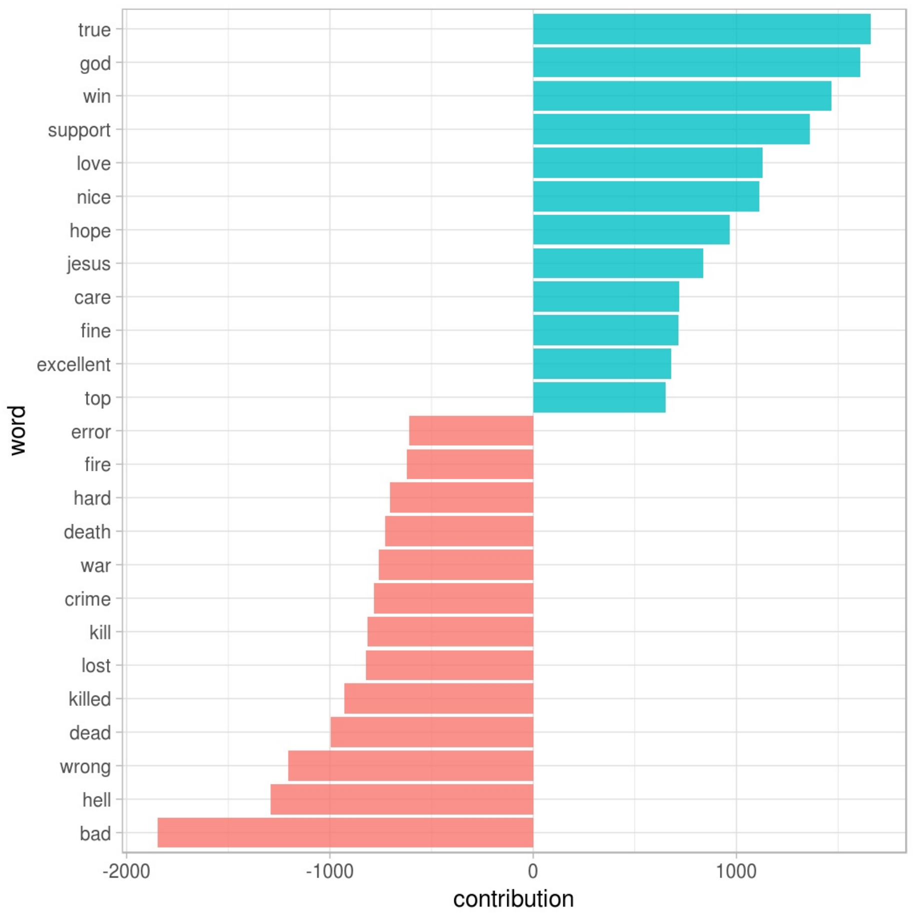
It’s worth looking deeper to understand *why* some boards ended up more positive than others. For that, we can examine the total positive and negative contributions of each word.

```
contributions <- usenet_words %>%
  inner_join(AFINN, by = "word") %>%
  group_by(word) %>%
  summarize(occurences = n(),
            contribution = sum(score))

contributions
```

```
## # A tibble: 1,891 × 3
##          word occurences contribution
##         <chr>      <int>        <int>
## 1    abandon         12          -24
## 2   abandoned        18          -36
## 3    abandons         3           -6
## 4   abduction         1           -2
## 5       abhor         3           -9
## 6    abhorred         1           -3
## 7   abhorrent         2           -6
## 8   abilities        16           32
## 9     ability       160          320
## 10    aboard          8            8
## # ... with 1,881 more rows
```

Which words had the most effect?

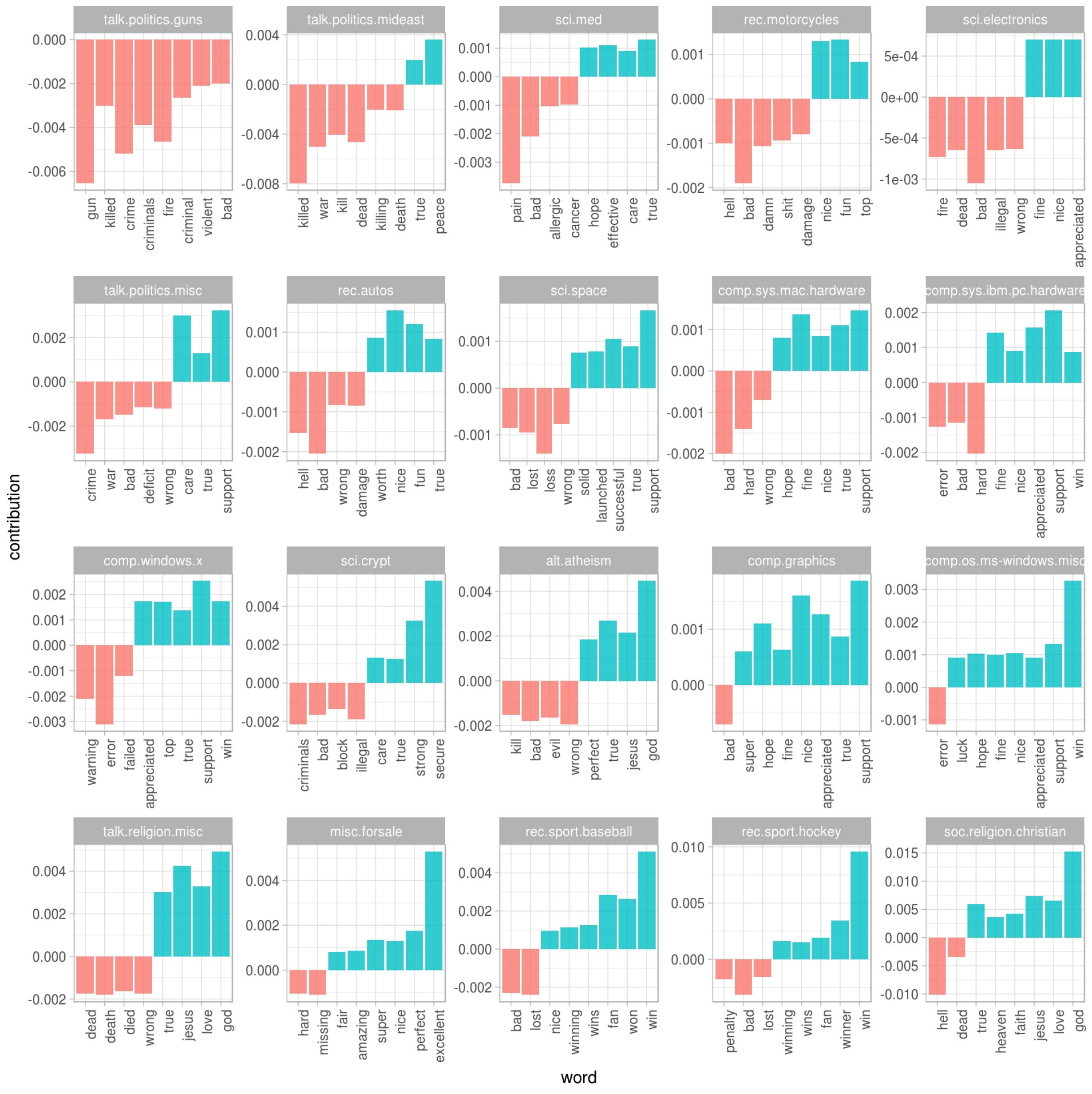
```
contributions %>%
  top_n(25, abs(contribution)) %>%
  mutate(word = reorder(word, contribution)) %>%
  ggplot(aes(word, contribution, fill = contribution > 0)) +
  geom_bar(alpha = 0.8, stat = "identity", show.legend = FALSE) +
  coord_flip()
```



These words look generally reasonable as indicators of each message's sentiment, but we can spot possible problems with the approach. "True" could just as easily be a part of "not true" or a similar negative expression, and the words "God" and "Jesus" are apparently very common on Usenet but could easily be used in many contexts, positive or negative.

The important point is that we may also care about which words contributed the most *within each board*. We can calculate each word's contribution to each board's sentiment score from our `word_board_sentiments` variable:

```
top_sentiment_words <- word_board_sentiments %>%
  mutate(contribution = score * n / sum(n))

top_sentiment_words %>%
  group_by(board) %>%
  top_n(8, abs(contribution)) %>%
  ungroup() %>%
  mutate(board = reorder(board, contribution),
         word = reorder(word, contribution)) %>%
  ggplot(aes(word, contribution, fill = contribution > 0)) +
  geom_bar(alpha = 0.8, stat = "identity", show.legend = FALSE) +
  facet_wrap(~ board, scales = "free") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



We can see here how much sentiment is confounded with topic in this particular approach. An atheism board is likely to discuss “god” in detail even in a negative context, and we can see it makes the board look more positive. Similarly, the negative contribution of the word “gun” to the “talk.politics.guns” board would occur even if the board members were discussing guns positively.

10.5 Sentiment analysis by message

We can also try finding the most positive and negative *messages*.

```r
sentiment_messages <- usenet_words %>%
  inner_join(AFINN, by = "word") %>%
  group_by(board, id) %>%
  summarize(sentiment = mean(score),
            words = n()) %>%
  ungroup() %>%
  filter(words >= 5)
```

As a simple measure to reduce the role of randomness, we filtered out messages that had fewer than five words that contributed to sentiment.

What were the most positive messages?

```
sentiment_messages %>%
  arrange(desc(sentiment))
```

```
## # A tibble: 3,385 × 4
##                       board     id sentiment words
##                       <chr>  <chr>     <dbl> <int>
## 1         rec.sport.hockey  53560  3.888889    18
## 2         rec.sport.hockey  53602  3.833333    30
## 3         rec.sport.hockey  53822  3.833333     6
## 4         rec.sport.hockey  53645  3.230769    13
## 5                rec.autos 102768  3.200000     5
## 6             misc.forsale  75965  3.000000     5
## 7             misc.forsale  76037  3.000000     5
## 8       rec.sport.baseball 104458  2.916667    12
## 9  comp.os.ms-windows.misc   9620  2.857143     7
## 10            misc.forsale  74787  2.833333     6
## # ... with 3,375 more rows
```

Let’s check this by looking at the most positive message in the whole data set.

```
print_message <- function(message_id) {
  cleaned_text %>%
    filter(id == message_id) %>%
    filter(text != "") %>%
    .$text %>%
    cat(sep = "\n")
}

print_message(53560)
```

```
## Everybody.  Please send me your predictions for the Stanley Cup Playoffs!
## I want to see who people think will win.!!!!!!!
## Please Send them in this format, or something comparable:
## 1. Winner of Buffalo-Boston
## 2. Winner of Montreal-Quebec
## 3. Winner of Pittsburgh-New York
## 4. Winner of New Jersey-Washington
## 5. Winner of Chicago-(Minnesota/St.Louis)
## 6. Winner of Toronto-Detroit
## 7. Winner of Vancouver-Winnipeg
## 8. Winner of Calgary-Los Angeles
## 9. Winner of Adams Division (1-2 above)
## 10. Winner of Patrick Division (3-4 above)
## 11. Winner of Norris Division (5-6 above)
## 12. Winner of Smythe Division (7-8 above)
## 13. Winner of Wales Conference (9-10 above)
## 14. Winner of Campbell Conference (11-12 above)
## 15. Winner of Stanley Cup (13-14 above)
## I will summarize the predictions, and see who is the biggest
## INTERNET GURU PREDICTING GUY/GAL.
## Send entries to Richard Madison
## rrmadiso@napier.uwaterloo.ca
## PS:  I will send my entries to one of you folks so you know when I say
## I won, that I won!!!!!
## From: sknapp@iastate.edu (Steven M. Knapp)
## Subject: Re: Radar detector DETECTORS?
## Organization: Iowa State University, Ames, IA
## Lines: 16
## Yes some radar detectors are less detectable by radar detector
## detectors. ;-)
## Look in Car and Driver (last 6 months should do), they had a big
## review of the "better" detectors, and stealth was a factor.
## Steven M. Knapp                             Computer Engineering Student
## sknapp@iastate.edu                  President Cyclone Amateur Radio Club
## Iowa State University; Ames, IA; USA      Durham Center Operations Staff
```

Looks like it’s because the message uses the word “winner” a lot! How about the most negative message? Turns out it’s also from the hockey site, but has a very different attitude.

```
sentiment_messages %>%
  arrange(sentiment)
```

```
## # A tibble: 3,385 × 4
##                    board     id sentiment words
##                    <chr>  <chr>     <dbl> <int>
## 1       rec.sport.hockey  53907 -3.000000     6
## 2        sci.electronics  53899 -3.000000     5
## 3              rec.autos 101627 -2.833333     6
## 4          comp.graphics  37948 -2.800000     5
## 5         comp.windows.x  67204 -2.700000    10
## 6      talk.politics.guns  53362 -2.666667     6
## 7            alt.atheism  51309 -2.600000     5
## 8  comp.sys.mac.hardware  51513 -2.600000     5
## 9              rec.autos 102883 -2.600000     5
## 10       rec.motorcycles  72052 -2.600000     5
## # ... with 3,375 more rows
```

```
print_message(53907)
```

```
## Losers like us? You are the fucking moron who has never heard of the Western
## Business School, or the University of Western Ontario for that matter. Why
## don't you pull your head out of your asshole and smell something other than
## shit for once so you can look on a map to see where UWO is! Back to hockey,
## the North Stars should be moved because for the past few years they have
## just been SHIT. A real team like Toronto would never be moved!!!
## Andrew--
```

Well then.

10.6 N-grams

We can also examine the effect of words that are used in negation, like we did in Chapter 5. Let's start by finding all the bigrams in the Usenet posts.

```
usenet_bigrams <- cleaned_text %>%
  unnest_tokens(bigram, text, token = "ngrams", n = 2)

usenet_bigrams
```

```
## # A tibble: 1,762,089 × 3
##          board    id            bigram
##          <chr> <chr>             <chr>
## 1  alt.atheism 49960      archive name
## 2  alt.atheism 49960      name atheism
## 3  alt.atheism 49960 atheism resources
## 4  alt.atheism 49960     resources alt
## 5  alt.atheism 49960       alt atheism
## 6  alt.atheism 49960   atheism archive
## 7  alt.atheism 49960      archive name
## 8  alt.atheism 49960    name resources
## 9  alt.atheism 49960    resources last
## 10 alt.atheism 49960     last modified
## # ... with 1,762,079 more rows
```

Now let's count how many of these bigrams are used in each board.

```
usenet_bigram_counts <- usenet_bigrams %>%
  count(board, bigram)

usenet_bigram_counts %>%
  arrange(desc(n))
```

```
## Source: local data frame [1,006,415 x 3]
## Groups: board [20]
##
##                     board bigram     n
##                     <chr>  <chr> <int>
## 1  soc.religion.christian of the  1141
## 2   talk.politics.mideast of the  1135
## 3   talk.politics.mideast in the   857
## 4               sci.space of the   684
## 5               sci.crypt of the   671
## 6      talk.politics.misc of the   645
## 7  soc.religion.christian in the   637
## 8      talk.religion.misc of the   630
## 9      talk.politics.guns of the   618
## 10            alt.atheism of the   474
## # ... with 1,006,405 more rows
```

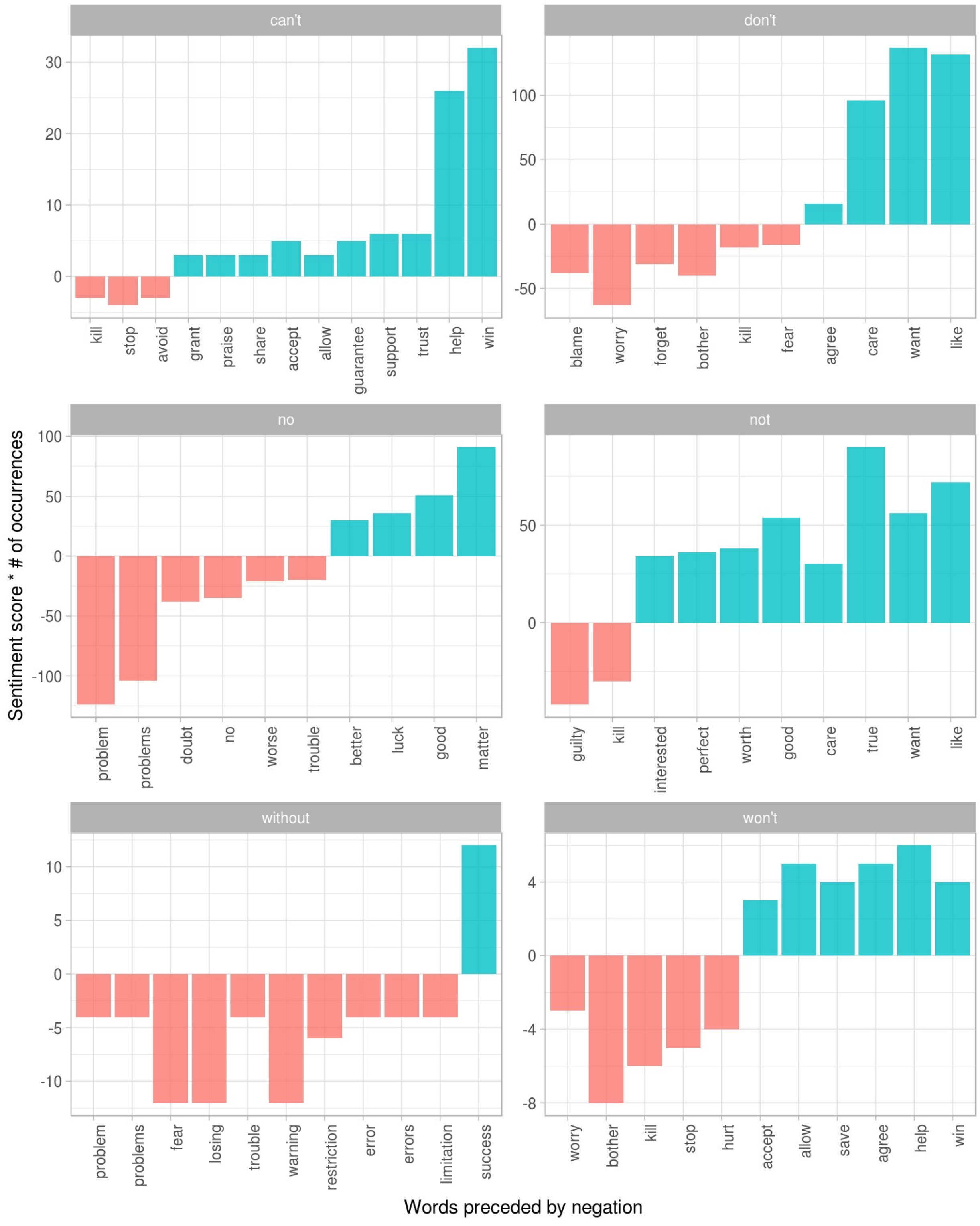
Next, we can calculate tf-idf for the bigrams to find the ones that are important for each board.

```
bigram_tf_idf <- usenet_bigram_counts %>%
  bind_tf_idf(bigram, board, n)

bigram_tf_idf %>%
  arrange(desc(tf_idf))
```

```
## Source: local data frame [1,006,415 x 6]
## Groups: board [20]
##
##                        board            bigram     n          tf      idf      tf_idf
##                        <chr>             <chr> <int>       <dbl>    <dbl>       <dbl>
## 1        talk.politics.misc mr stephanopoulos   155 0.001477344 2.995732 0.004425728
## 2            comp.windows.x                n x   177 0.001917577 2.302585 0.004415384
## 3            comp.windows.x           x printf   130 0.001408390 2.995732 0.004219158
## 4           rec.motorcycles           the bike   104 0.001675663 2.302585 0.003858356
## 5  comp.sys.ibm.pc.hardware             scsi 2   107 0.001478983 2.302585 0.003405485
## 6            comp.windows.x             file x   104 0.001126712 2.995732 0.003375327
## 7     talk.politics.mideast     the armenians   169 0.001111988 2.995732 0.003331220
## 8          rec.sport.hockey                1 0   256 0.002733816 1.203973 0.003291440
## 9            comp.windows.x      output oname   100 0.001083377 2.995732 0.003245506
## 10           comp.windows.x             x char    98 0.001061709 2.995732 0.003180596
## # ... with 1,006,405 more rows
```

Now we come back to the words used in negation that we are interested in examining. Let’s define a vector of words that we suspect are used in negation, and use the same joining and counting approach from Chapter 5 to examine all of them at once.

```
negate_words <- c("not", "without", "no", "can't", "don't", "won't")

usenet_bigram_counts %>%
  ungroup() %>%
  separate(bigram, c("word1", "word2"), sep = " ") %>%
  filter(word1 %in% negate_words) %>%
  count(word1, word2, wt = n, sort = TRUE) %>%
  inner_join(AFINN, by = c(word2 = "word")) %>%
  mutate(contribution = score * nn) %>%
  top_n(10, abs(contribution)) %>%
  ungroup() %>%
  mutate(word2 = reorder(word2, contribution)) %>%
  ggplot(aes(word2, contribution, fill = contribution > 0)) +
  geom_bar(alpha = 0.8, stat = "identity", show.legend = FALSE) +
  facet_wrap(~ word1, scales = "free", nrow = 3) +
  xlab("Words preceded by negation") +
  ylab("Sentiment score * # of occurrences") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



These words are the ones that contribute the most to the sentiment scores in the wrong direction, because they are being used with negation words before them. Phrases like “no problem” and “don’t want” are important sources of misidentification.

11 References

Benoit, Kenneth, and Paul Nulty. 2016. *Quanteda: Quantitative Analysis of Textual Data*. https://CRAN.R-project.org/package=quanteda.

Ingo Feinerer, Kurt Hornik, and David Meyer. 2008. “Text Mining Infrastructure in R.” *Journal of Statistical Software* 25 (5): 1–54. http://www.jstatsoft.org/v25/i05/.

Mullen, Lincoln. 2016. *Tokenizers: A Consistent Interface to Tokenize Natural Language Text*. https://CRAN.R-project.org/package=tokenizers.

Robinson, David. 2016. *Gutenbergr: Download and Process Public Domain Works from Project Gutenberg*. https://cran.rstudio.com/package=gutenbergr.

Robinson, David, Matthieu Gomez, Boris Demeshev, Dieter Menne, Benjamin Nutter, Luke Johnston, Ben Bolker, Francois Briatte, and Hadley Wickham. 2015. *Broom: Convert Statistical Analysis Objects into Tidy Data Frames*. https://CRAN.R-project.org/package=broom.

Silge, Julia. 2016. *Janeaustenr: Jane Austen’s Complete Novels*. https://CRAN.R-project.org/package=janeaustenr.

Silge, Julia, and David Robinson. 2016. “Tidytext: Text Mining and Analysis Using Tidy Data Principles in R.” *JOSS* 1 (3). The Open Journal. doi:10.21105/joss.00037.

Wickham, Hadley. 2007. “Reshaping Data with the reshape Package.” *Journal of Statistical Software* 21 (12): 1–20. http://www.jstatsoft.org/v21/i12/.

———. 2009. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. http://ggplot2.org.

———. 2014. “Tidy Data.” *Journal of Statistical Software* 59 (1): 1–23. doi:10.18637/jss.v059.i10.

———. 2016. *Tidyr: Easily Tidy Data with `spread()` and `gather()` Functions*. https://CRAN.R-project.org/package=tidyr.

Wickham, Hadley, and Romain Francois. 2016. *Dplyr: A Grammar of Data Manipulation*. https://CRAN.R-project.org/package=dplyr.