Tidy Text Mining in R

Julia Silge and David Robinson

2016-08-23

Contents

Chapter 1

Introduction

- There is lots of unstructured data proliferating, including text. Analysts are often trained on numeric data, but not in even simple interpretation of natural language.
- The authors developed the tidytext package 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 the tidy data philosophy. By treating text as data frames of words, we can manipulate, summarize, and visualize it easily
- 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 (**?**), 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 worth contrasting with the ways text is often stored in current analyses (TODO: move this to chapter 2?)

- **Raw strings**
- **Corpus** These types of objects typically annotate the raw string content with additional metadata and details
- **Document-term matrix** This is a sparse matrix with one row for each document and one column for each term

Tidy data sets allow manipulation with a standard set of "tidy" tools, including popular packages such as dplyr (**?**), tidyr (**?**), ggplot2 (**?**), and broom (**?**). 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 analyses and explorations.

In the tidytext package provide functionality to tokenize by commonly used units of text including words, n-grams, and sentences. This lets someone convert efficiently from a data frame containing documents into a one-term-per-row format. 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 (**?**)) from popular text mining R packages such as tm (**?**) and quanteda (**?**).

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, and 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. We're confident that even a user early in their career can pick up these For users who don't have this background, we recommend books such R for Data Science [TODO]. 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 method of TF-IDF (term frequency times inverse document frequency), for identifying terms that are especially important to a particular document. (Other document stuff in this chapter perhaps?)
- **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.

Most notably CRAN Task View on Natural Language Processing

- **Supervised 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 through Latent Dirichlet Allocation) in Chapter 6
- **More complex tokenization.** We hand tokenization off to the tokenizers package [cite], which itself wraps a variety of tokenizers with a consistent interface, but many others exist for specific applications.
- **More here**

We feel that the tools . We also believe strongly that the tidy data philosophy is well suited to extensions

Chapter 2

The Tidy Text Format

Intro text may go here about the one-token-per-document-per-row and about what is explored in the chapter.

2.1 The unnest_tokens function

```
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
```

```
## Source: local data frame [4 x 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 coimbined tokens. We need to turn this into **one-token-per-document-per-row**.

To do this, we use tidytext's `unnest_tokens` function:

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

```
## # A tibble: 20 x 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's one token (word) in each row of the new data frame. Also notice:

- Other columns, such as the line number each word came from, are retained
- Punctuation has been stripped
- By default, `unnest_tokens` turns the tokens 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, and transform them into a tidy format. janeaustenr provides them as a one-row-per-line format:

```
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
```

```
## Source: local data frame [70,942 x 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
## ..                     ...                   ...        ...      ...
```

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

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

tidy_books
```

```
## Source: local data frame [724,971 x 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
## ..                 ...        ...     ...         ...
```

This function uses the tokenizers package to separate each line into words. The default tokenizing is for words, but other options include characters, ngrams, 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`.

```r
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.

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

```
## Source: local data frame [13,896 x 2]
##
##      word     n
##     <chr> <int>
## 1    miss  1854
## 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
## ..    ...   ...
```

For example, this allows us to visualize the popular words using ggplot2:

```
library(ggplot2)

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

```
## Source: local data frame [13,896 x 2]
##
##      word     n
##     <chr> <int>
## 1    miss  1854
## 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
## ..    ...   ...
```

2.3 The gutenbergr package

TODO: Now that we've introduced the janeaustenr package, also include a brief intro to the gutenberg package.

2.4 Word frequencies

A common task in text mining is to look at word frequencies 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 x 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 bit of a 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 x 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
```

Well, Jane Austen is not going around talking about people's *hearts* this much; I can tell you that right now. Those Brontë sisters, SO DRAMATIC. Interesting that "time" and "door" are in the top 10 for both H.G. Wells and the Brontë sisters. "Door"?!

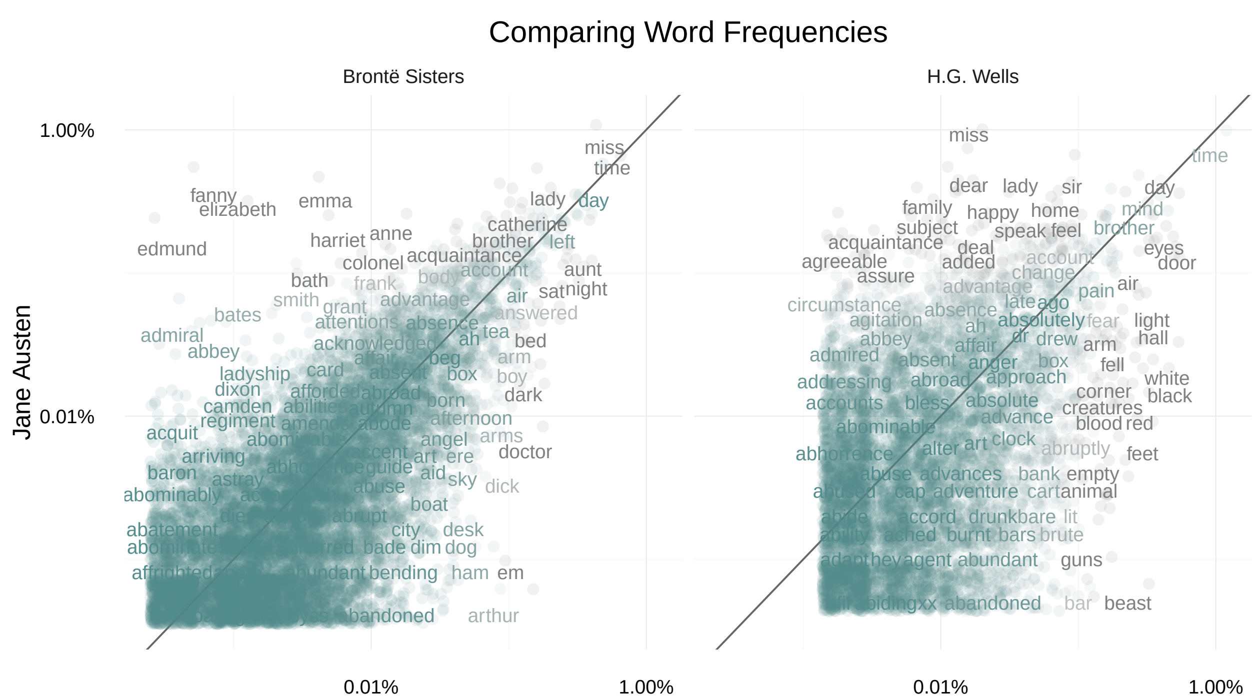
Anyway, let's calculate the frequency for each word for the works of Jane Austen, the Brontë sisters, and H.G. Wells.

```
tidy_both <- bind_rows(
        mutate(tidy_bronte, author = "Brontë Sisters"),
        mutate(tidy_hgwells, author = "H.G. Wells"))
frequency <- tidy_both %>%
        mutate(word = str_extract(word, "[a-z]+")) %>%
        count(author, word) %>%
        rename(other = n) %>%
        inner_join(count(tidy_books, word)) %>%
        rename(Austen = n) %>%
```

```
        mutate(other = other / sum(other),
               Austen = Austen / sum(Austen)) %>%
        ungroup()
```

I'm using `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.

```
library(scales)
ggplot(frequency, aes(x = other, y = Austen, color = abs(Austen - other))) +
        geom_abline(color = "gray40") +
        geom_jitter(alpha = 0.1, size = 2.5, width = 0.4, height = 0.4) +
        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_minimal(base_size = 14) +
        theme(legend.position="none") +
        labs(title = "Comparing Word Frequencies",
             subtitle = "Word frequencies in Jane Austen's texts are closer to the Brontë sisters' than
             y = "Jane Austen", x = NULL)
```



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", "lady", "day" at the upper frequency end) or in both Austen and Wells texts ("time", "day", "mind", "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 plot, words like "elizabeth", "emma", "captain", and "bath" (all proper nouns) are found in Austen's texts but not much in the Brontë texts, while words like "arthur", "dark", "dog", and "doctor" are found in the Brontë texts but not the Austen texts. In comparing H.G. Wells with Jane Austen, Wells uses words like "beast", "guns", "brute", and "animal" that Austen does not, while Austen uses words like "family", "friend", "letter", and "agreeable" that Wells does not.

Overall, notice that the words in the Austen-Brontë plot are closer to the zero-slope line than in the Austen-Wells plot and also extend to lower frequencies; Austen and the Brontë sisters use more similar words than Austen and H.G. Wells. Also, we might notice the percent frequencies for individual words are different in one plot when compared to another because of the inner join; not all the words are found in all three sets of texts so the percent frequency is a different quantity.

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 = 122.45, df = 10611, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.7572399 0.7730119
## sample estimates:
##       cor
## 0.7652408
```

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

```
##
##  Pearson's product-moment correlation
##
## data:  other and Austen
## t = 36.043, df = 5958, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.4020291 0.4437216
## sample estimates:
##       cor
## 0.4230993
```

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

Chapter 3

Sentiment Analysis with Tidy Data

3.1 The `sentiments` dataset

There are a variety of methods and dictionaries that exist for evaluating the opinion or emotion in text. The tidytext package contains three sentiment lexicons in the `sentiments` dataset.

```
sentiments
```

```
## # A tibble: 23,165 x 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 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 (or 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.

These dictionary-based methods 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.

One last caveat is that the size of the chunk of text that we add up unigram sentiment scores for can have an important effect for an analysis. A paragraph-sized text can often have positive and negative sentiment averaged out to about zero, while sentence-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. Let's look at the words with a joy score from the NRC lexicon. What are the most common joy words in *Emma*?

```
nrcjoy <- sentiments %>%
  filter(lexicon == "nrc", sentiment == "joy")

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

```
## # A tibble: 298 x 2
##         word     n
##        <chr> <int>
## 1     friend   166
## 2       hope   143
## 3      happy   125
## 4       love   117
## 5       deal    92
## 6      found    92
## 7  happiness    76
## 8     pretty    68
## 9       true    66
## 10   comfort    65
## # ... with 288 more rows
```

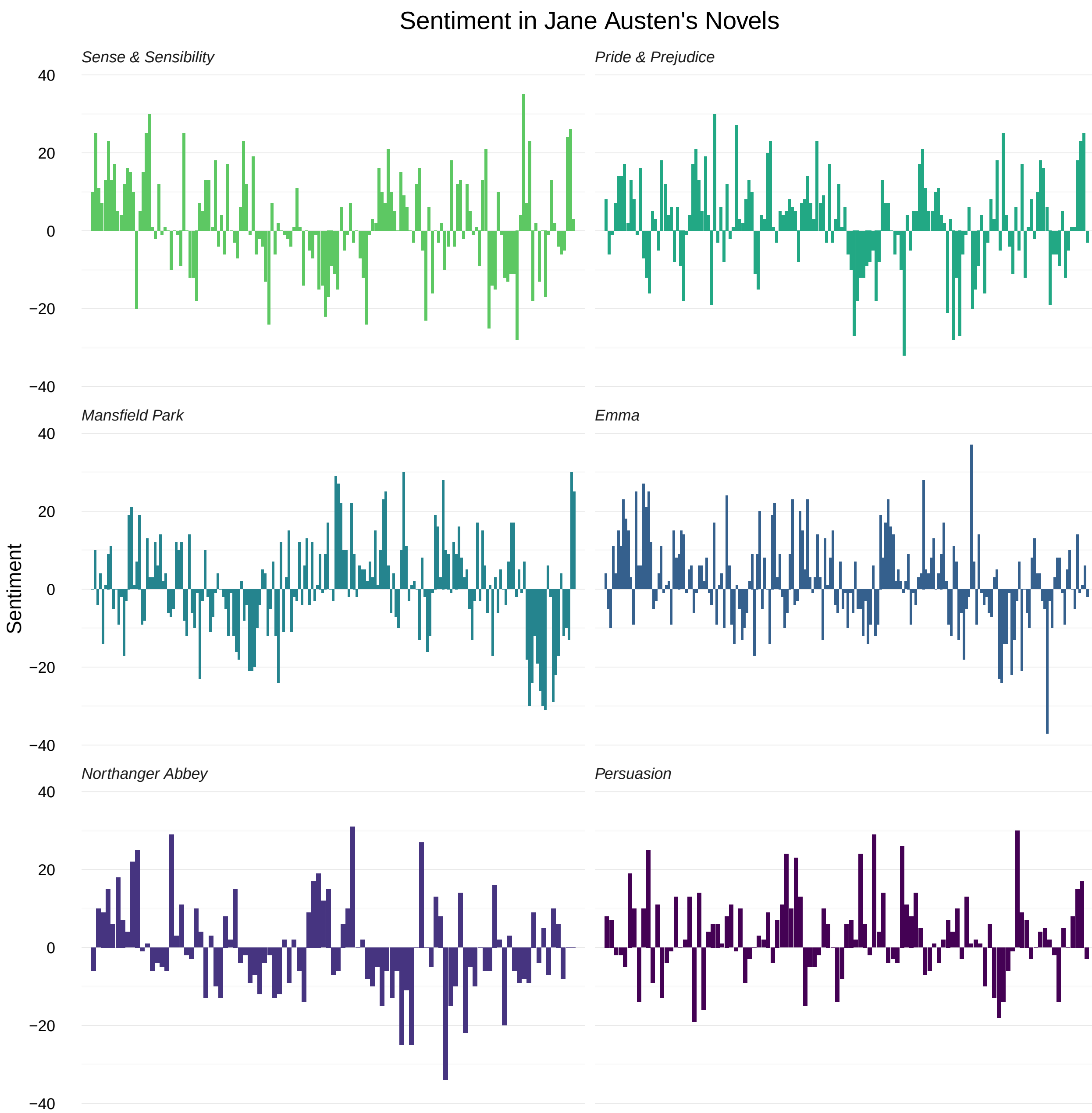
Or instead we could examine how sentiment changes during each novel. Let's find a sentiment score for each word using the Bing lexicon, then count the number of positive and negative words in defined sections of each novel.

```
library(tidyr)
bing <- sentiments %>%
  filter(lexicon == "bing") %>%
  select(-score)

janeaustensentiment <- tidy_books %>%
  inner_join(bing) %>%
  count(book, index = linenumber %/% 80, sentiment) %>%
  spread(sentiment, n, fill = 0) %>%
  mutate(sentiment = positive - negative)
```

Now we can plot these sentiment scores across the plot trajectory of each novel.

```
library(ggplot2)
library(viridis)
ggplot(janeaustensentiment, aes(index, sentiment, fill = book)) +
        geom_bar(stat = "identity", show.legend = FALSE) +
```

```
  facet_wrap(~book, ncol = 2, scales = "free_x") +
  theme_minimal(base_size = 13) +
  labs(title = "Sentiment in Jane Austen's Novels",
       y = "Sentiment") +
  scale_fill_viridis(end = 0.75, discrete=TRUE, direction = -1) +
  scale_x_discrete(expand=c(0.02,0)) +
  theme(strip.text=element_text(hjust=0)) +
  theme(strip.text = element_text(face = "italic")) +
  theme(axis.title.x=element_blank()) +
  theme(axis.ticks.x=element_blank()) +
  theme(axis.text.x=element_blank())
```



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

TODO: The three different methods of calculating sentiment give results that are different in an absolute sense but have similar relative trajectories through the novel.

3.3 Most common positive and negative words

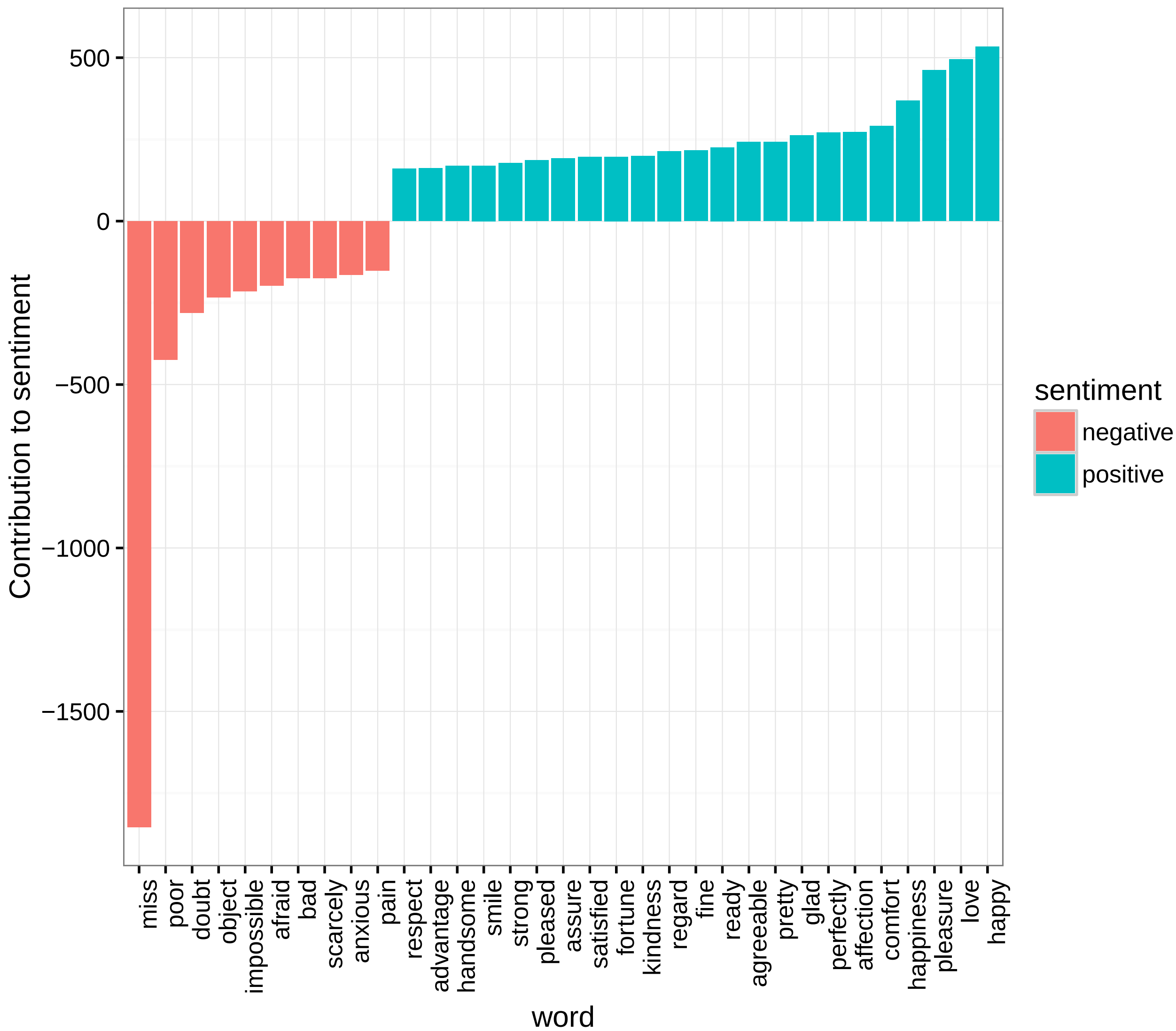
One advantage of having the data frame with both sentiment and word is that we can analyze word counts that contribute to each sentiment.

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

bing_word_counts
```

```
## # A tibble: 2,554 x 3
##         word sentiment     n
##        <chr>     <chr> <int>
## 1       miss  negative  1854
## 2      happy  positive   534
## 3       love  positive   495
## 4   pleasure  positive   462
## 5       poor  negative   424
## 6  happiness  positive   369
## 7    comfort  positive   292
## 8      doubt  negative   281
## 9  affection  positive   272
## 10 perfectly  positive   271
## # ... with 2,544 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 %>%
  filter(n > 150) %>%
  mutate(n = ifelse(sentiment == "negative", -n, n)) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n, fill = sentiment)) +
  geom_bar(stat = "identity") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("Contribution to sentiment")
```



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`.

3.4 Wordclouds

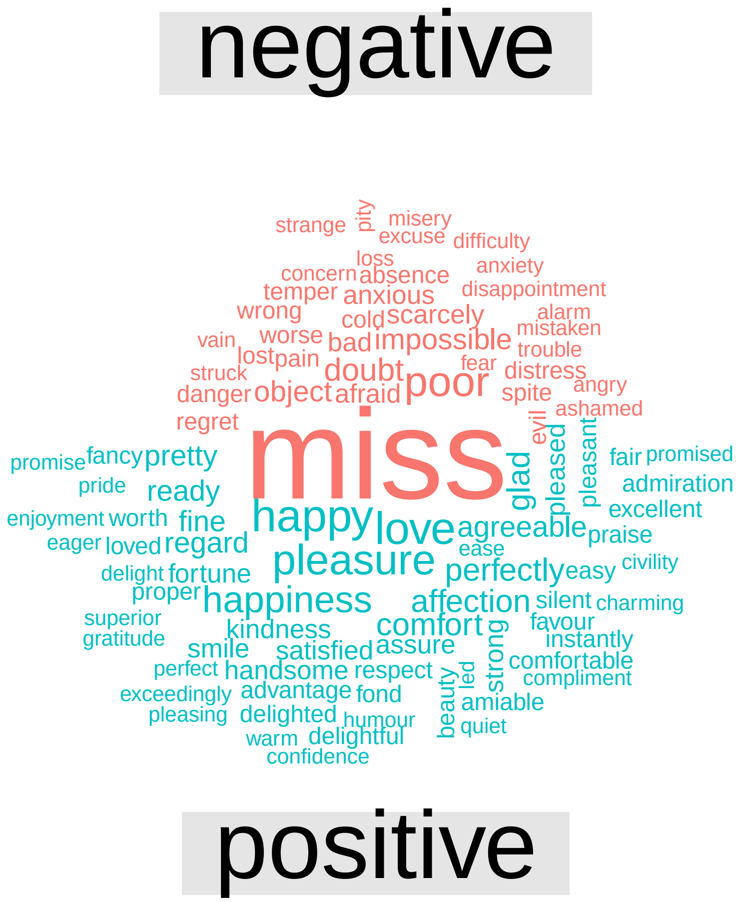
We've seen that this tidy text mining approach works well with ggplot2, but having our data in a tidy format is useful for other plots as well.

For example, consider the wordcloud package. Let's look at the most common words in Jane Austen's works as a whole again.

```
library(wordcloud)

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

In other functions, such as `comparison.cloud`, you may need to turn it 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(bing) %>%
  count(word, sentiment, sort = TRUE) %>%
  acast(word ~ sentiment, value.var = "n", fill = 0) %>%
  comparison.cloud(colors = c("#F8766D", "#00BFC4"),
                   max.words = 100)
```



3.5 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 a neighbourhood,
```

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.

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 x 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 dataframe 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 <- sentiments %>%
  filter(lexicon == "bing", 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)
```

```
## Source: local data frame [6 x 5]
## Groups: book [6]
##
##                  book chapter negativewords words     ratio
##                <fctr>   <int>         <int> <int>     <dbl>
## 1 Sense & Sensibility      29           172  1135 0.1515419
## 2   Pride & Prejudice      34           108   646 0.1671827
## 3      Mansfield Park      45           132   884 0.1493213
## 4                Emma      15           147  1012 0.1452569
## 5    Northanger Abbey      27            55   337 0.1632047
## 6          Persuasion      21           215  1948 0.1103696
```

These are the chapters with the most negative words in each book, normalized for number of words in the chapter. What is happening in these chapters? In Chapter 29 of *Sense and Sensibility* Marianne finds out what an awful person Willoughby is by letter, and in Chapter 34 of *Pride and Prejudice* Mr. Darcy proposes for the first time (so badly!). Chapter 45 of *Mansfield Park* is almost the end, when Tom is sick with consumption and Mary is revealed as mercenary and uncaring, Chapter 15 of *Emma* is when horrifying Mr. Elton proposes, and Chapter 27 of *Northanger Abbey* is a short chapter where Catherine gets a terrible letter from her inconstant friend Isabella. Chapter 21 of *Persuasion* is when Anne's friend tells her all about Mr. Elliott's immoral past.

Interestingly, many of those chapters are very close to the ends of the novels; things tend to get really bad for Jane Austen's characters before their happy endings, it seems. Also, these chapters largely involve terrible revelations about characters through letters or conversations about past events, rather than some action happening directly in the plot. All that, just with dplyr verbs, because the data is tidy.

Chapter 4

TF-IDF: Analyzing word and document frequency

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. 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 sophisticated approach to adjusting term frequency for commonly used words.

4.1 Term frequency and inverse document frequency

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 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 #tidytext, 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.

4.2 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.)

```
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 x 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
```

The usual suspects are here, "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.

```
library(ggplot2)
library(viridis)
ggplot(book_words, aes(n/total, fill = book)) +
        geom_histogram(alpha = 0.8, show.legend = FALSE) +
        xlim(NA, 0.0009) +
        labs(title = "Term Frequency Distribution in Jane Austen's Novels",
             y = "Count") +
        facet_wrap(~book, ncol = 2, scales = "free_y") +
        theme_minimal(base_size = 13) +
        scale_fill_viridis(end = 0.85, discrete=TRUE) +
        theme(strip.text=element_text(hjust=0)) +
        theme(strip.text = element_text(face = "italic"))
```

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.

4.3 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.

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

```
## # A tibble: 40,379 x 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.

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

```
## # A tibble: 40,379 x 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. 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.

```r
library(ggstance)
library(ggthemes)
plot_austen <- book_words %>%
        arrange(desc(tf_idf)) %>%
        mutate(word = factor(word, levels = rev(unique(word))))
ggplot(plot_austen[1:20,], aes(tf_idf, word, fill = book, alpha = tf_idf)) +
```

```
        geom_barh(stat = "identity") +
        labs(title = "Highest tf-idf words in Jane Austen's Novels",
             y = NULL, x = "tf-idf") +
        theme_tufte(base_family = "Arial", base_size = 13, ticks = FALSE) +
        scale_alpha_continuous(range = c(0.6, 1), guide = FALSE) +
        scale_x_continuous(expand=c(0,0)) +
        scale_fill_viridis(end = 0.85, discrete=TRUE) +
        theme(legend.title=element_blank()) +
        theme(legend.justification=c(1,0), legend.position=c(1,0))
```

Let's look at the novels individually.

```
plot_austen <- plot_austen %>% group_by(book) %>% top_n(15) %>% ungroup
ggplot(plot_austen, aes(tf_idf, word, fill = book, alpha = tf_idf)) +
        geom_barh(stat = "identity", show.legend = FALSE) +
        labs(title = "Highest tf-idf words in Jane Austen's Novels",
             y = NULL, x = "tf-idf") +
        facet_wrap(~book, ncol = 2, scales = "free") +
        theme_tufte(base_family = "Arial", base_size = 13, ticks = FALSE) +
        scale_alpha_continuous(range = c(0.6, 1)) +
        scale_x_continuous(expand=c(0,0)) +
        scale_fill_viridis(end = 0.85, discrete=TRUE) +
        theme(strip.text=element_text(hjust=0)) +
        theme(strip.text = element_text(face = "italic"))
```

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

via GIPHY

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

```
## # A tibble: 12,592 x 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, and of course these documents are all very different lengths. Let's go ahead and calculate tf-idf.

```
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(tf_idf, word, fill = author, alpha = tf_idf)) +
        geom_barh(stat = "identity") +
        labs(title = "Highest tf-idf words in Classic Physics Texts",
             y = NULL, x = "tf-idf") +
        theme_tufte(base_family = "Arial", base_size = 13, ticks = FALSE) +
        scale_alpha_continuous(range = c(0.6, 1), guide = FALSE) +
        scale_x_continuous(expand=c(0,0)) +
        scale_fill_viridis(end = 0.6, discrete=TRUE) +
        theme(legend.title=element_blank()) +
        theme(legend.justification=c(1,0), legend.position=c(1,0))
```

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(tf_idf, word, fill = author, alpha = tf_idf)) +
        geom_barh(stat = "identity", show.legend = FALSE) +
        labs(title = "Highest tf-idf words in Classic Physics Texts",
             y = NULL, x = "tf-idf") +
        facet_wrap(~author, ncol = 2, scales = "free") +
        theme_tufte(base_family = "Arial", base_size = 13, ticks = FALSE) +
        scale_alpha_continuous(range = c(0.6, 1)) +
        scale_x_continuous(expand=c(0,0)) +
        scale_fill_viridis(end = 0.6, discrete=TRUE) +
        theme(strip.text=element_text(hjust=0))
```

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

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

```
##  [1] "                       Fig. 01: file fig01.gif"
```

```
##  [2] "                               eq. 1: file eq01.gif"
##  [3] "                               eq. 2: file eq02.gif"
##  [4] "                               eq. 3: file eq03.gif"
##  [5] "                               eq. 4: file eq04.gif"
##  [6] "                             eq. 05a: file eq05a.gif"
##  [7] "                             eq. 05b: file eq05b.gif"
##  [8] "                              eq. 07: file eq07.gif"
##  [9] "                              eq. 08: file eq08.gif"
## [10] "                              eq. 09: file eq09.gif"
```

Some cleaning up of the text might be in order. The same thing is true for "eq", obviously here. "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"
```

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. "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.

```
mystopwords <- data_frame(word = c("gif", "eq", "co", "rc", "ac", "ak", "bn",
                                  "fig", "file", "cg", "cb"))
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(tf_idf, word, fill = author, alpha = tf_idf)) +
        geom_barh(stat = "identity", show.legend = FALSE) +
        labs(title = "Highest tf-idf words in Classic Physics Texts",
             y = NULL, x = "tf-idf") +
        facet_wrap(~author, ncol = 2, scales = "free") +
        theme_tufte(base_family = "Arial", base_size = 13, ticks = FALSE) +
        scale_alpha_continuous(range = c(0.6, 1)) +
        scale_x_continuous(expand=c(0,0)) +
        scale_fill_viridis(end = 0.6, discrete=TRUE) +
        theme(strip.text=element_text(hjust=0))
```

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

Chapter 5

Working with combinations of words using n-grams and widyr

5.1 Tokenizing by n-gram

We've been using the `unnest_tokens` function to tokenize by word, or sometimes by sentence or paragraph. But we can also tokenize into consecutive sequences of words, called **n-grams**.

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

# Set n = 2 to divide into pairs of words
austen_digrams <- austen_books() %>%
  unnest_tokens(digram, text, token = "ngrams", n = 2)

austen_digrams
```

```
## # A tibble: 725,048 x 2
##                   book          digram
##                 <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 is still tidy: it's one-row-per-token, but now each token represents a digram. Notice that these digrams are overlapping: "sense and" is one token, "and sensibility" is another.

5.1.1 Counting and filtering n-grams

We can examine the most common digrams using `count`:

```
austen_digrams %>%
  count(digram, sort = TRUE)
```

```
## # A tibble: 211,237 x 2
##      digram     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 expected, a lot of them are pairs of common (relatively uninteresting) words. 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", which we can remove stop-words from individually:

```
library(tidyr)

digrams_separated <- austen_digrams %>%
  separate(digram, c("word1", "word2"), sep = " ")

digrams_filtered <- digrams_separated %>%
  filter(!word1 %in% stop_words$word) %>%
  filter(!word2 %in% stop_words$word)

digrams_filtered
```

```
## # A tibble: 44,784 x 3
##                   book       word1        word2
##                 <fctr>       <chr>        <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
```

We can now count the most common pairs of words:

```
digrams_filtered %>%
  count(word1, word2, sort = TRUE)
```

```
## 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.

We may want to work with the recombined words. tidyr's `unite()` is the opposite of `separate()`, and lets us recombine the columns into one.

```
digrams_united <- digrams_filtered %>%
  unite(digram, word1, word2, sep = " ")

digrams_united
```

```
## # A tibble: 44,784 x 2
##                   book                   digram
## *               <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
```

You could also easily work with trigrams (sequences of 3 words) 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
```

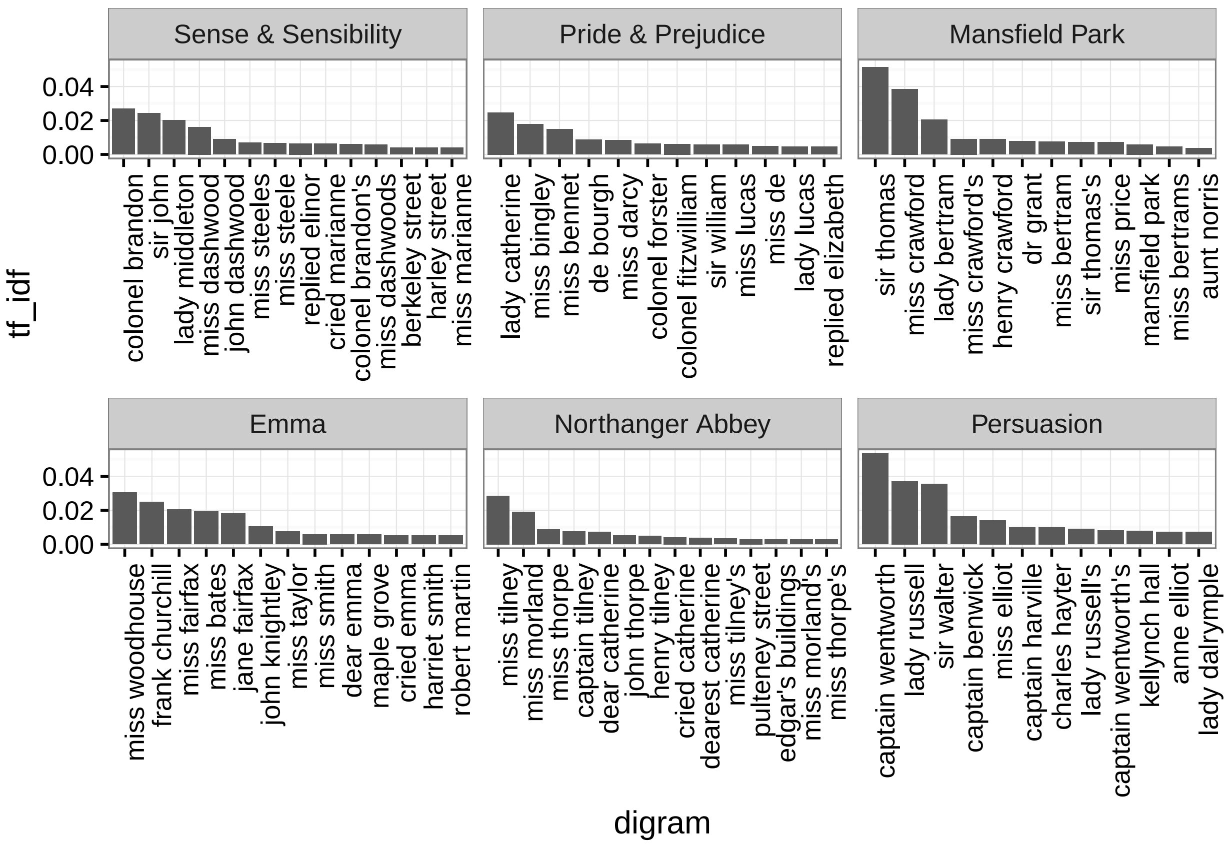
5.1.2 Analyzing digrams

A digram can be treated like a term just as we treated a word. For example, we can look at TF-IDF of digrams:

```
digram_tf_idf <- digrams_united %>%
  count(book, digram) %>%
  bind_tf_idf(digram, book, n) %>%
  arrange(desc(tf_idf))

digram_tf_idf
```

```
## Source: local data frame [36,217 x 6]
## Groups: book [6]
##
##                   book            digram     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
```

This can be visualized within each book, just as we did for words:



Much as we discovered in Chapter **?**{#tfidf}, the units that distinguish each Austen book are almost exclusively names.

5.1.3 Using digrams to provide context in sentiment analysis

Our sentiment analysis approch in Chapter **?**{#sentiment} 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 matters nearly as much as its presence. For example, the words "happy" and "like" will be positive, even in a sentence like "I'm not **happy** and I don't **like** it!"

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

```
## # A tibble: 1,246 x 2
##    word2     n
##    <chr> <int>
## 1     be   610
## 2     to   355
## 3   have   327
## 4   know   252
## 5      a   189
## 6  think   176
## 7   been   160
## 8    the   147
## 9     at   129
## 10    in   118
```

```
## # ... with 1,236 more rows
```

We can use word2 to

Let's use the AFINN lexicon for sentiment analysis, which gives a sentiment score for each word:

```r
AFINN <- sentiments %>%
  filter(lexicon == "AFINN") %>%
  select(word, score)

AFINN
```

```
## # A tibble: 2,476 x 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
```

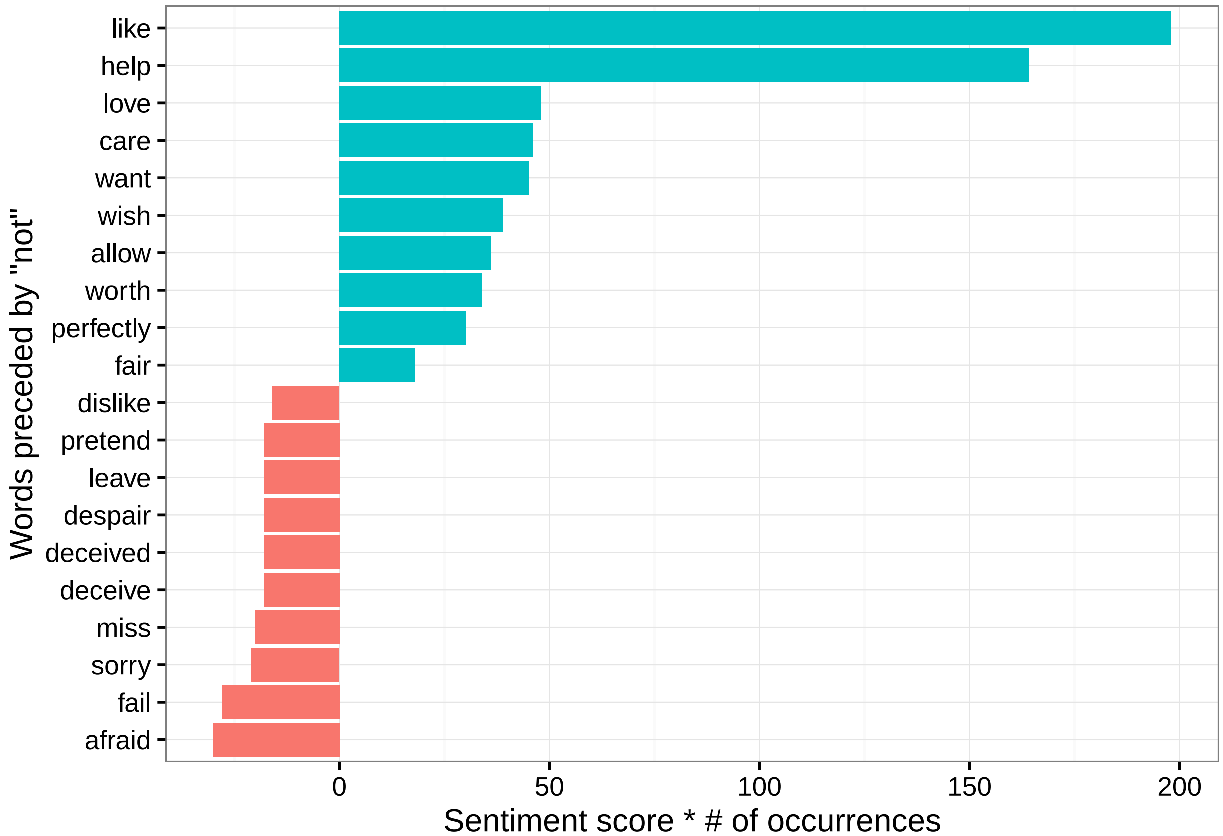
```r
not_words <- digrams_separated %>%
  filter(word1 == "not") %>%
  inner_join(AFINN, by = c(word2 = "word")) %>%
  count(word2, score, sort = TRUE) %>%
  ungroup()

not_words
```

```
## # A tibble: 245 x 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
```

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 sentiment score of +3 occurring 10 times has as much impact as a word with a sentiment score of +1 occurring 30 times).

```r
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 * # of occurrences") +
  coord_flip()
```



The digrams "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.

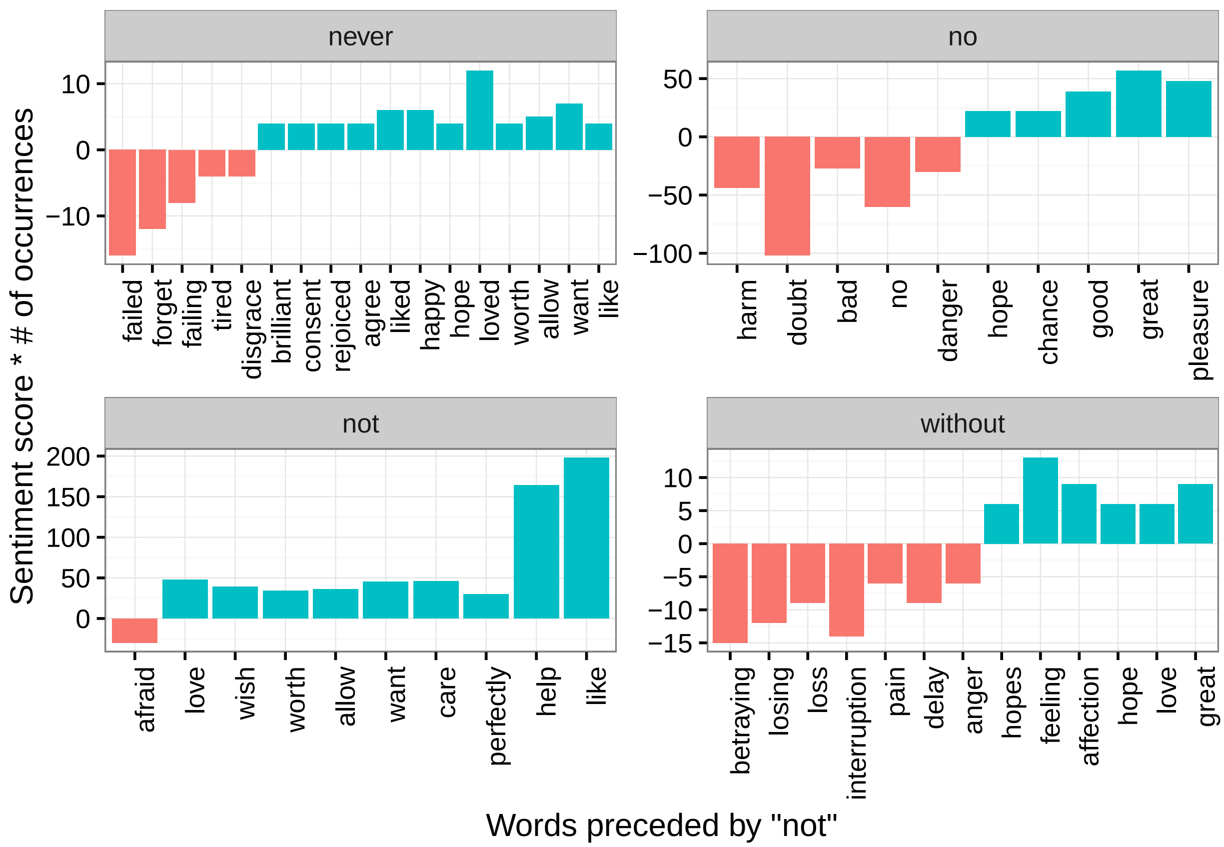
"Not" isn't the only word that provides context. We could make a vector of words that we suspect , and use the same joining and counting approach to examine all of them:

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

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

negated_words
```

```
## # A tibble: 531 x 4
##    word1 word2 score     n
##    <chr> <chr> <int> <int>
## 1     no doubt    -1   102
```

```
## 2    not  like     2    99
## 3    not  help     2    82
## 4     no    no    -1    60
## 5    not  want     1    45
## 6    not  wish     1    39
## 7    not allow     1    36
## 8    not  care     2    23
## 9     no  harm    -2    22
## 10   not sorry    -1    21
## # ... with 521 more rows
```

```
negated_words %>%
  mutate(contribution = n * score) %>%
  mutate(word2 = reorder(word2, contribution)) %>%
  group_by(word1) %>%
  top_n(10, abs(contribution)) %>%
  ggplot(aes(word2, contribution, fill = n * score > 0)) +
  geom_bar(stat = "identity", show.legend = FALSE) +
  facet_wrap(~ word1, scales = "free") +
  xlab("Words preceded by \"not\"") +
  ylab("Sentiment score * # of occurrences") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



5.2 Visualizing digrams as a network with the ggraph package

A

5.2.1 Creating a network with igraph

Now that we have our

```
digram_counts <- digrams_filtered %>%
  count(word1, word2, sort = TRUE)

digram_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
```

Here we'll be referring to a "graph" not in the sense of a visualization, but as a . A graph can be created from a tidy object because a graph 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

```
library(igraph)

digram_graph <- digram_counts %>%
  filter(n > 20) %>%
  graph_from_data_frame()

digram_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
```

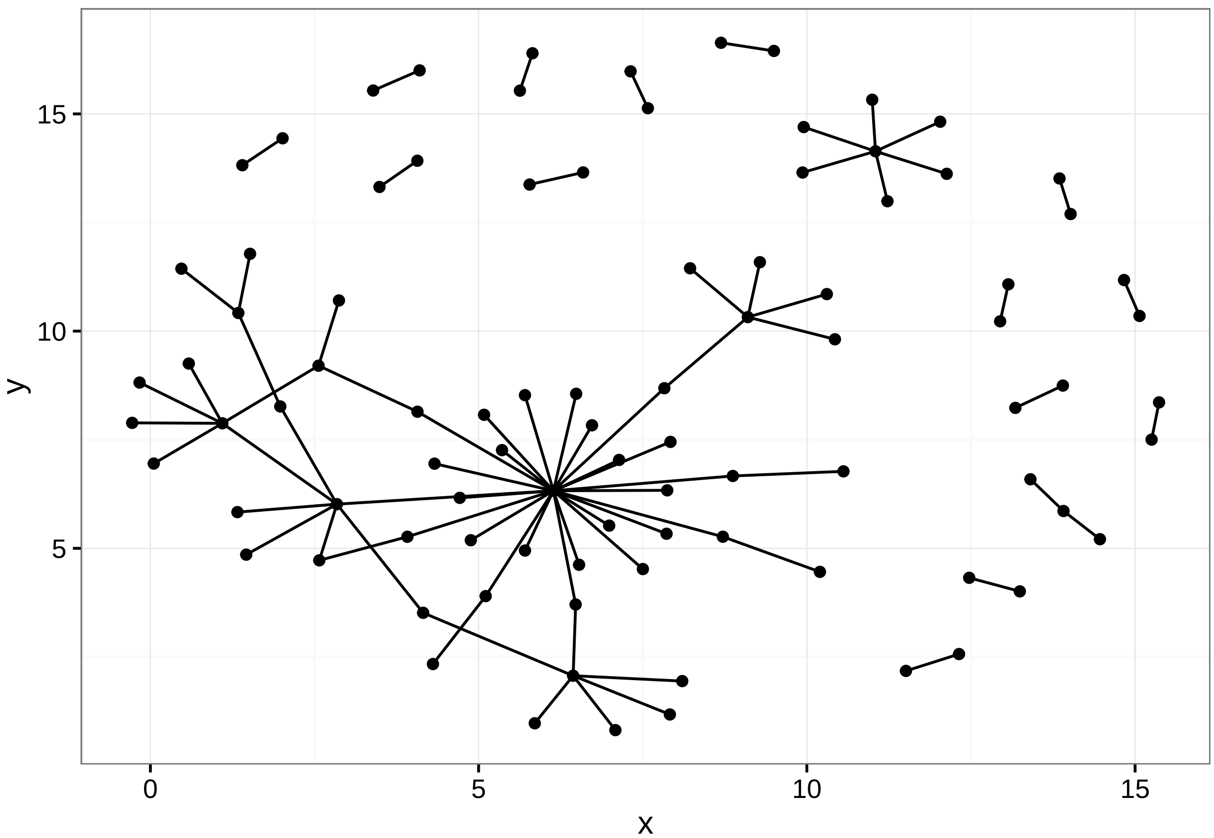
The igraph package has many powerful functions for manipulating and analyzing networks.

TODO: examples of igraph package

5.2.2 Visualizing a network with ggraph

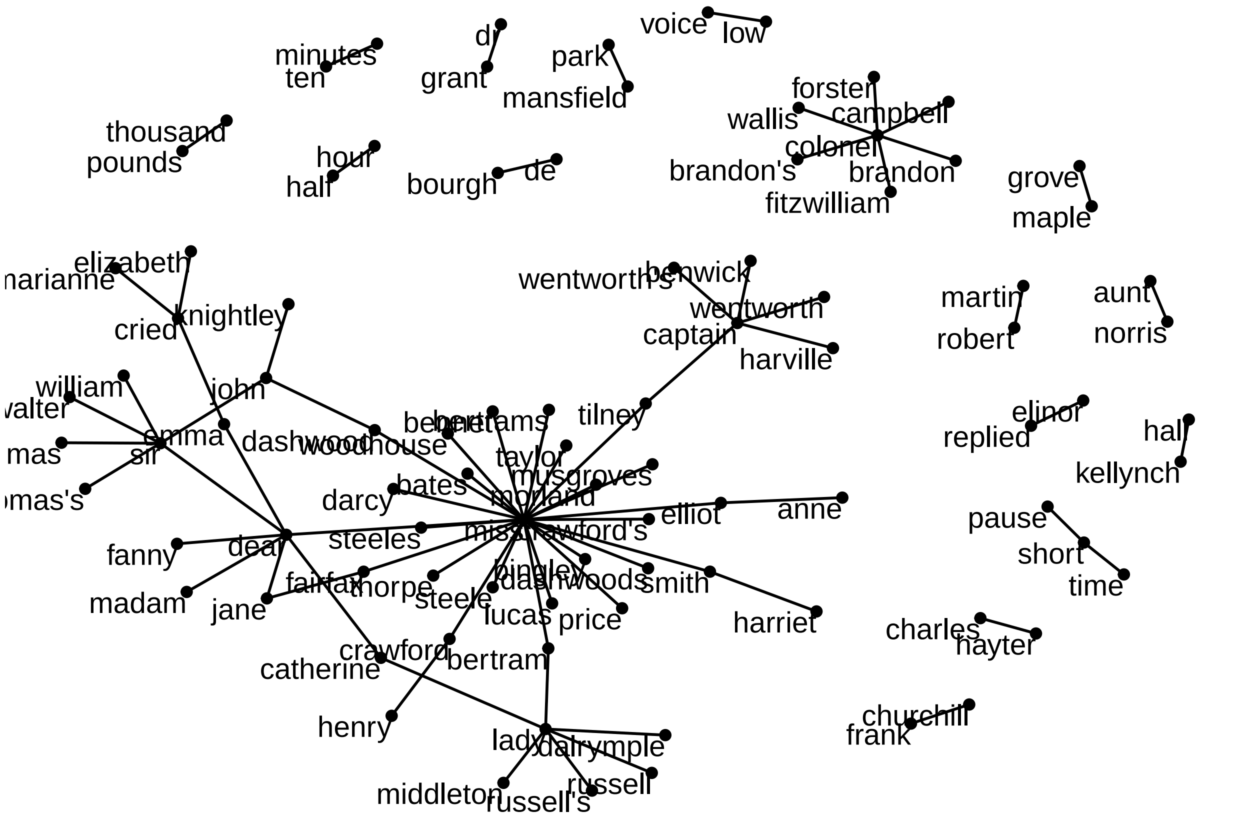
igraph has plotting functions built in, but they're not what the package is designed to do. Many others have developed visualization methods for graphs. But we like the ggraph package, because it implements it in terms of the grammar of graphics.

```
library(ggraph)

set.seed(2016)

ggraph(digram_graph, layout = "fr") +
  geom_edge_link() +
  geom_node_point()
```



This gives an idea by adding the three …

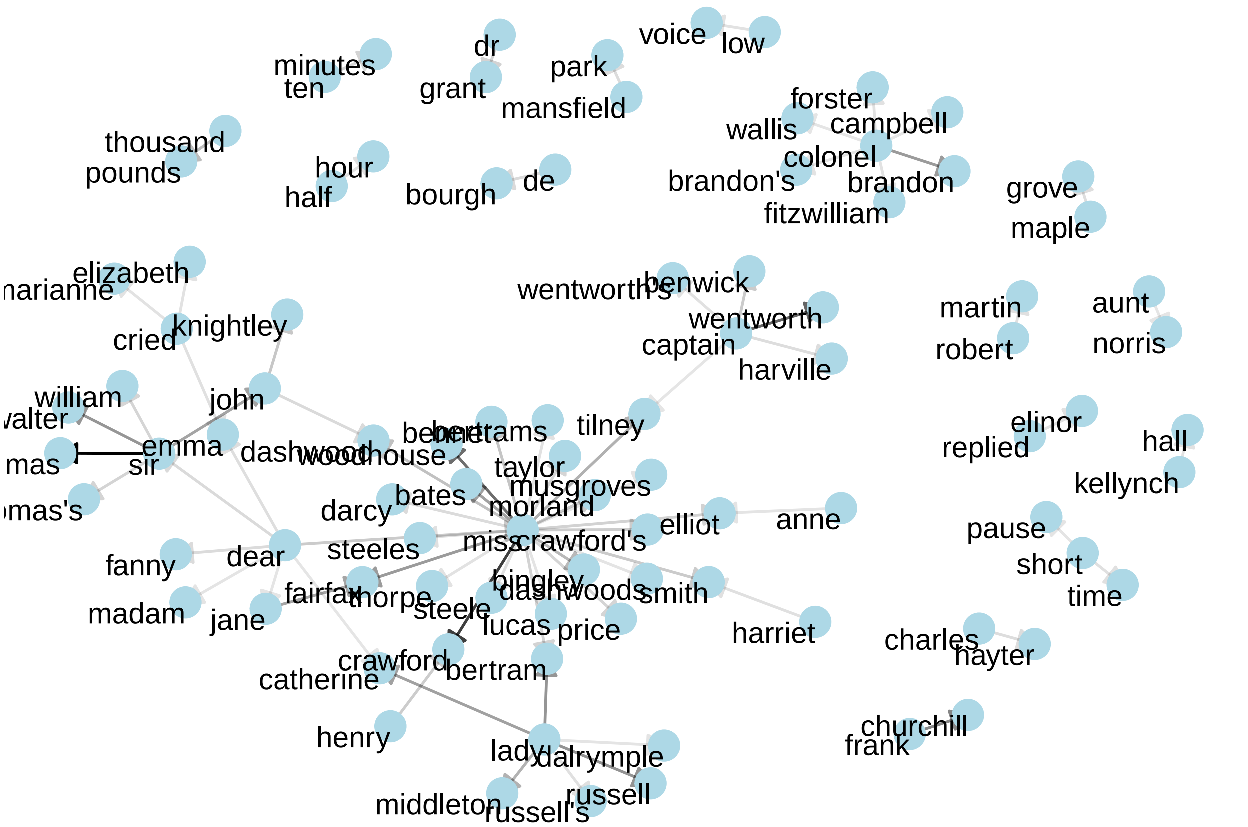
```
set.seed(2016)

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



We can see the graph start to take shape.

- We add the `edge_alpha` aesthetic to the link layer to make links transparent based on how common or rare the digram is
- We add directionality with an arrow
- We tinker with the options to the node layer to make the points more attractive (larger, and blue)

```r
set.seed(2016)

a <- grid::arrow(type = "closed", length = unit(.1, "inches"))

ggraph(digram_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()
```



It may take a little more experimentation with your plots to get these graphs to work, but in the end we can visualize a lot this way.

5.2.3 Visualizing digrams in other texts

We went to a good amount of work setting up this

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

visualize_digrams <- function(digrams) {
  set.seed(2016)

  digrams %>%
    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()
}
```

We could visualize pairs in the King James Bible:

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

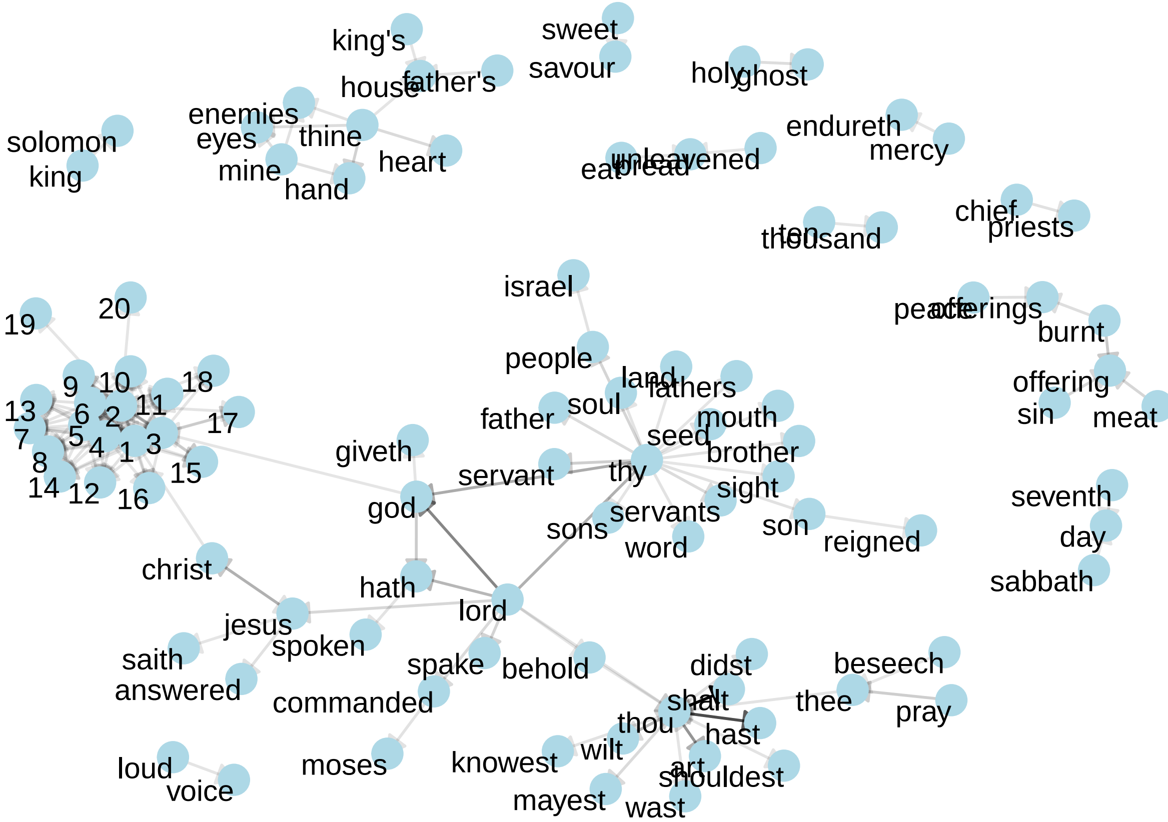
```
library(gutenbergr)

kjv <- gutenberg_download(10)

kjv
```

```
## # A tibble: 99,817 x 2
##    gutenberg_id                                                          text
##           <int>                                                         <chr>
## 1            10    The Old Testament of the King James Version of the Bible
## 2            10
## 3            10
## 4            10
## 5            10
## 6            10                        The First Book of Moses:  Called Genesis
## 7            10
## 8            10
## 9            10 1:1 In the beginning God created the heavens and the earth.
## 10           10
## # ... with 99,807 more rows
```

```
kjv_digrams <- kjv %>%
  count_digrams()

kjv_digrams
```

```
## Source: local data frame [47,551 x 3]
## Groups: word1 [7,265]
##
##    word1    word2     n
##    <chr>    <chr> <int>
## 1   thou    shalt  1250
## 2   thou     hast   768
## 3   lord      god   546
## 4    thy      god   356
## 5   thou      art   320
## 6   lord      thy   316
## 7   lord     hath   291
## 8  shalt     thou   258
## 9  jesus   christ   196
## 10 burnt offering   184
## # ... with 47,541 more rows
```

```
kjv_digrams %>%
  filter(n > 40) %>%
  visualize_digrams()
```



5.3 Counting and correlating pairs of words with the widyr package

We've previously analyzed

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

One example of the widyr pattern 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)

word_pairs <- austen_section_words %>%
  pairwise_count(word, section, sort = TRUE)

word_pairs
```

```
## # A tibble: 796,030 x 3
##        item1     item2     n
##        <chr>     <chr> <dbl>
## 1  elizabeth     darcy   144
## 2      darcy elizabeth   144
```

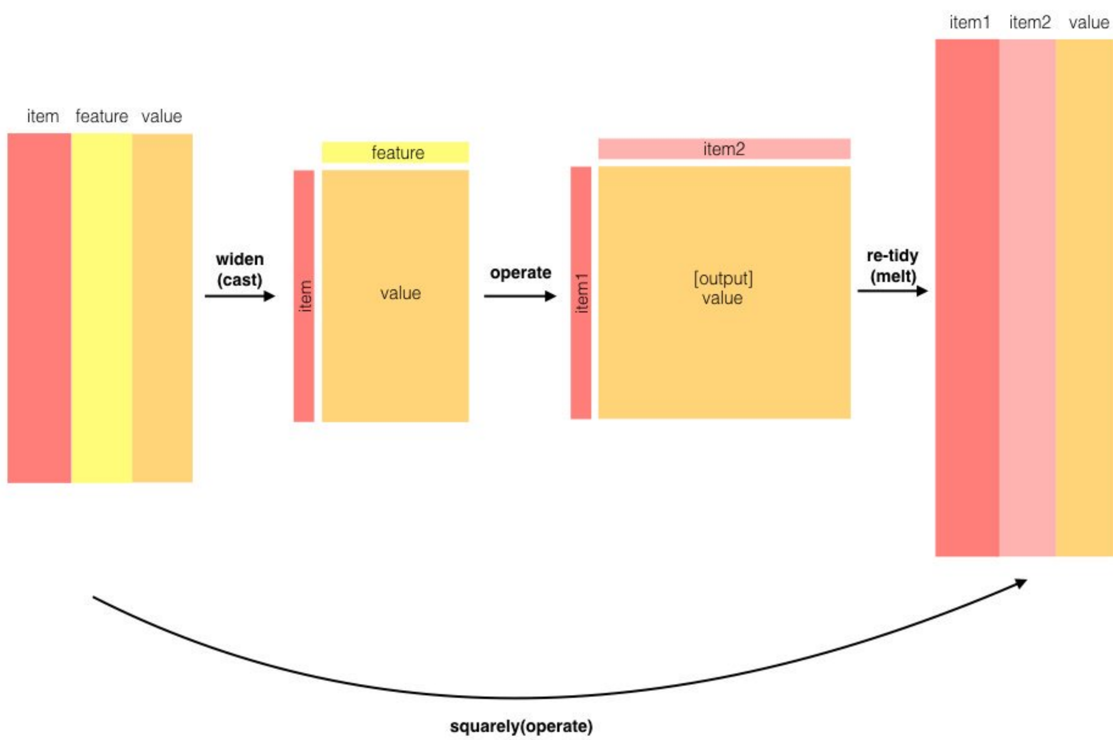


Figure 5.1: The philosophy behind the widyr package, which can operations such as counting and correlating on pairs of values in a tidy dataset.

```
## 3  elizabeth      miss   110
## 4       miss elizabeth   110
## 5       jane elizabeth   106
## 6  elizabeth      jane   106
## 7      darcy      miss    92
## 8       miss     darcy    92
## 9    bingley elizabeth    91
## 10 elizabeth   bingley    91
## # ... with 796,020 more rows
```

For example, we discover that the most common pair of words in a section is "Elizabeth" and "Darcy" (the two main characters).

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

```
## # A tibble: 2,930 x 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    sister    45
## 6  darcy    bennet    45
## 7  darcy      time    41
## 8  darcy      lady    38
## 9  darcy   wickham    37
## 10 darcy    friend    37
## # ... with 2,920 more rows
```

5.3.1 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 words.** We instead want to examine *correlation* among words, which is how often they appear together relative to how often they appear separately.

TODO: formula for Pearson correlation, explanation of phi coefficient

The `pairwise_cor()` function in widyr lets us perform a Pearson correlation across words.

```
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 x 3
##        item1     item2 correlation
##        <chr>     <chr>       <dbl>
## 1         de    bourgh   0.9508510
## 2     bourgh        de   0.9508510
## 3   thousand    pounds   0.7005850
```

```
## 4     pounds  thousand   0.7005850
## 5        sir   william   0.6644804
## 6    william       sir   0.6644804
## 7       lady catherine   0.6633289
## 8  catherine      lady   0.6633289
## 9    colonel   forster   0.6221042
## 10   forster   colonel   0.6221042
## # ... with 154,832 more rows
```

We could find the words most correlated with Elizabeth:

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

```
## # A tibble: 393 x 3
##    item1       item2 correlation
##    <chr>       <chr>       <dbl>
## 1  darcy        miss  0.17989385
## 2  darcy fitzwilliam  0.17899838
## 3  darcy     bingley  0.17878908
## 4  darcy       hurst  0.10658328
## 5  darcy   pemberley  0.09581220
## 6  darcy      friend  0.09281731
## 7  darcy      manner  0.09143830
## 8  darcy       eliza  0.08663187
## 9  darcy       proud  0.07751503
## 10 darcy        eyes  0.07700740
## # ... with 383 more rows
```

5.3.2 Visualizing word correlations

Just as we used ggraph to visualize digrams, we can use it to visualize correlations and clusters among words that we've found through the widyr package.

This graph is an early placeholder, needs to be adjusted:

```
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), vjust = 1, hjust = 1) +
  theme_void()
```

Note that unlike the digram analysis, the relationship here aren't directional.

This kind of correlation network is a very useful and flexible visualization, and we'll examine it further in later chapters.

Chapter 6

Tidying and casting document-term matrices

So far, we've been analyzing data in a tidy text structure: a data frame with one-token-per-document-per-row. This lets us use the popular tidy suite of tools such as dplyr, tidyr, and ggplot2. We've demonstrated that many text analyses can be performed within these.

But many of the existing tools for natural language processing don't work with this kind of structure. The CRAN Task View for Natural Language Processing lists a large selection of packages that take other inputs. One of the most common is the document-term matrix, a sparse matrix of counts with one row for each document and one column for each term. MORE HERE

The tidytext package lets us can integrate these packages into an analysis while still relying on our tidy tools. The two key verbs are:

- `tidy()`:
- `cast_`: Turn a tidy one-term-per-row data frame into a document-term matrix. This includes `cast_sparse()` (sparse Matrix), `cast_dtm()` (`DocumentTermMatrix` objects from tm), and `cast_dfm()` (`dfm` objects from quanteda).

6.1 Tidying a document-term matrix

Many existing text mining datasets expect and provide a **document-term matrix**, or DTM. A DTM is a matrix where:

- Each row represents one document
- Each column represents one term
- Each value 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 interacted with as though they were matrices, but are stored in a more efficient format.

One commonly used implementation of DTMs in R is the `DocumentTermMatrix` class in the tm package. For example, consider the corpus of 2246 Associated Press articles from the topicmodels package:

```
library(tm)

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

```
## [1] "DocumentTermMatrix"    "simple_triplet_matrix"
```

```
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 (words).

If we want to analyze this with tidy tools, we need to turn it into a one-token-per-document-per-row data frame first. The broom package (**?**) introduced the `tidy` verb, which takes a non-tidy object and turns it into a data frame. The `tidytext` package implements that method for `DocumentTermClass` objects:

```
library(dplyr)
library(tidytext)

ap_td <- tidy(AssociatedPress)
ap_td
```

```
## Source: local data frame [302,031 x 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
## ..      ...        ...   ...
```

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

As we've seen in chapters 2-5, this form is convenient for analysis with the dplyr and tidytext packages. For example, you can perform sentiment analysis on these newspaper articles.

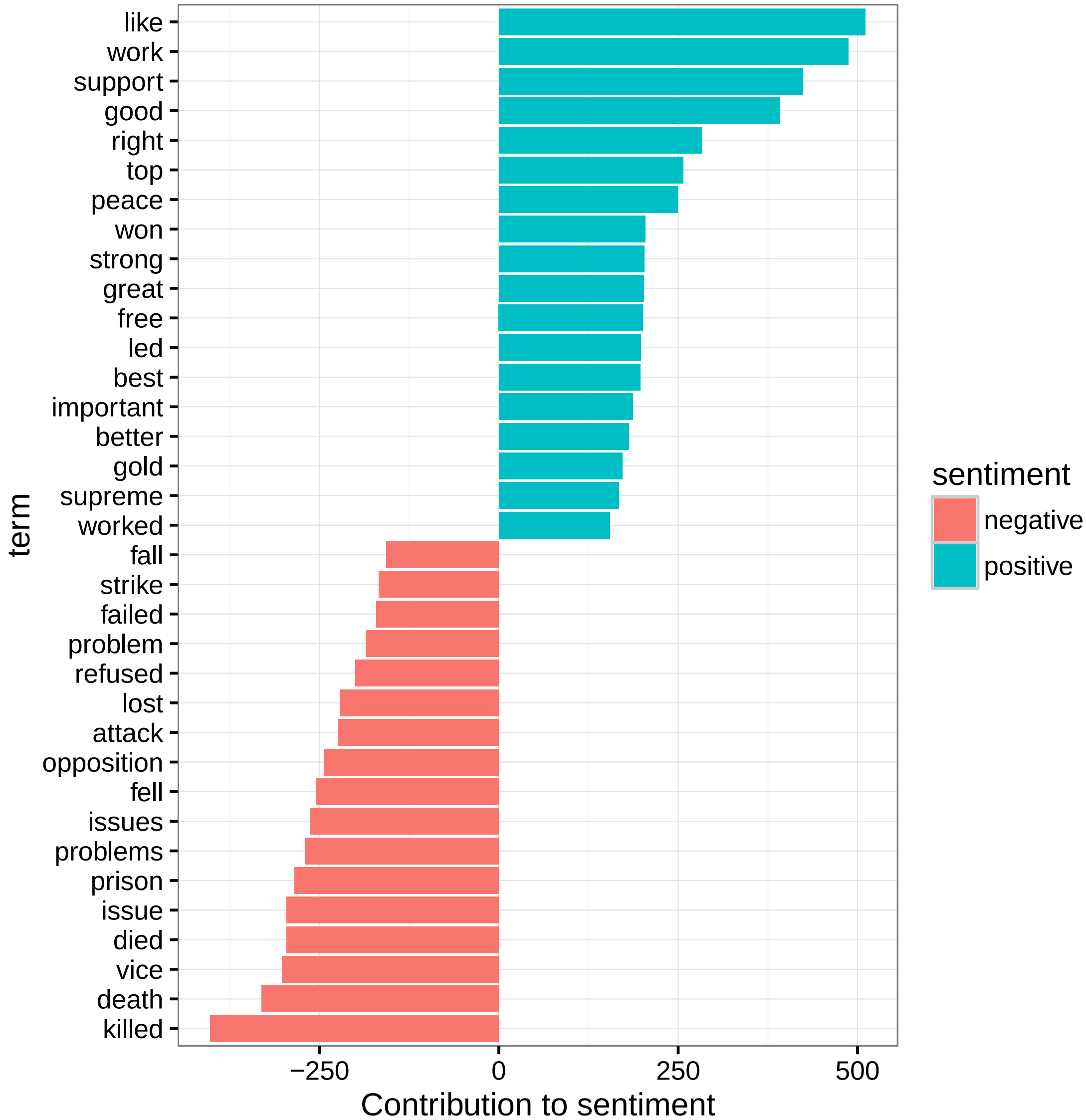
```
bing <- sentiments %>%
  filter(lexicon == "bing") %>%
  select(word, sentiment)

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

ap_sentiments
```

```
## Source: local data frame [30,094 x 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
## ..       ...     ...   ...       ...
```

This could, for example, let us visualize which words from these AP articles most often contributed to positive or negative sentiment:

```
library(ggplot2)

ap_sentiments %>%
  count(sentiment, term, wt = count) %>%
  ungroup() %>%
  filter(n >= 150) %>%
  mutate(n = ifelse(sentiment == "negative", -n, n)) %>%
  mutate(term = reorder(term, n)) %>%
  ggplot(aes(term, n, fill = sentiment)) +
  geom_bar(stat = "identity") +
  ylab("Contribution to sentiment") +
  coord_flip()
```



A tidier is also available for the `dfm` (document-feature matrix) class from the quanteda package (**?**). Consider the corpus of presidential inauguration speeches that comes with the quanteda package:

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

```
## Creating a dfm from a corpus ...
##    ... lowercasing
##    ... tokenizing
##    ... indexing documents: 57 documents
##    ... indexing features: 9,215 feature types
##    ... created a 57 x 9215 sparse dfm
##    ... complete.
## Elapsed time: 0.183 seconds.
```

```
d
```

```
## Document-feature matrix of: 57 documents, 9,215 features.
```

```r
tidy(d)
```

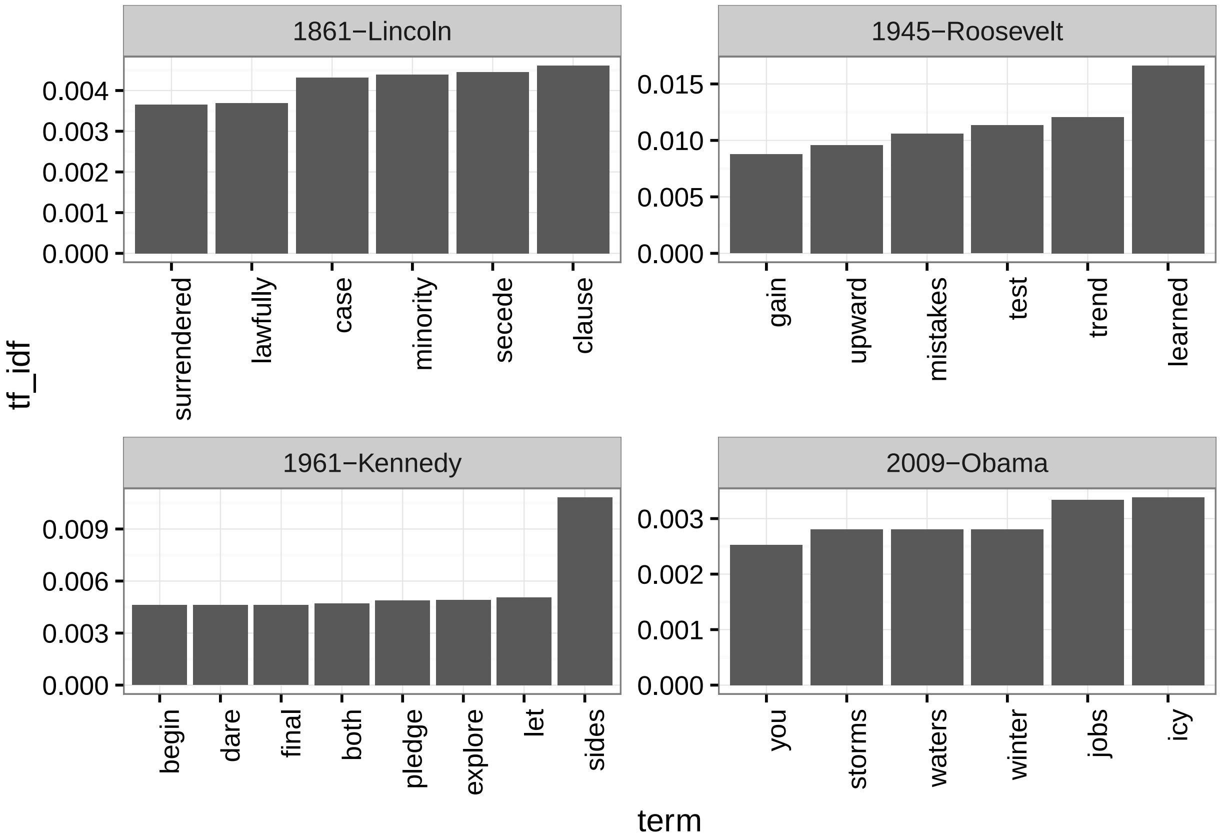
```
## Source: local data frame [43,719 x 3]
##
##           document           term count
##              <chr>          <chr> <dbl>
## 1  1789-Washington fellow-citizens     1
## 2       1797-Adams fellow-citizens     3
## 3   1801-Jefferson fellow-citizens     2
## 4     1809-Madison fellow-citizens     1
## 5     1813-Madison fellow-citizens     1
## 6      1817-Monroe fellow-citizens     5
## 7      1821-Monroe fellow-citizens     1
## 8    1841-Harrison fellow-citizens    11
## 9        1845-Polk fellow-citizens     1
## 10     1849-Taylor fellow-citizens     1
## ..             ...             ...   ...
```

We could find the words most specific to several inaugural speeches using `bind_tf_idf` from chapter 4:

```r
speeches <- c("1861-Lincoln", "1945-Roosevelt",
              "1961-Kennedy", "2009-Obama")

inaug_tf_idf <- tidy(d) %>%
  bind_tf_idf(term, document, count) %>%
  arrange(desc(tf_idf)) %>%
  filter(document %in% speeches)

inaug_tf_idf
```

```
## # A tibble: 2,690 x 6
##          document        term count          tf      idf      tf_idf
##             <chr>       <chr> <dbl>       <dbl>    <dbl>       <dbl>
## 1  1945-Roosevelt     learned     5 0.009009009 1.845827 0.016629069
## 2  1945-Roosevelt       trend     2 0.003603604 3.349904 0.012071726
## 3  1945-Roosevelt        test     3 0.005405405 2.097141 0.011335898
## 4    1961-Kennedy       sides     8 0.005865103 1.845827 0.010825963
## 5  1945-Roosevelt    mistakes     2 0.003603604 2.944439 0.010610591
## 6  1945-Roosevelt      upward     2 0.003603604 2.656757 0.009573899
## 7  1945-Roosevelt        gain     2 0.003603604 2.433613 0.008769778
## 8  1945-Roosevelt  well-being     2 0.003603604 2.251292 0.008112763
## 9  1945-Roosevelt   faintness     1 0.001801802 4.043051 0.007284777
## 10 1945-Roosevelt schoolmaster     1 0.001801802 4.043051 0.007284777
## # ... with 2,680 more rows
```

```r
inaug_tf_idf %>%
  group_by(document) %>%
  top_n(6, tf_idf) %>%
  ungroup() %>%
  mutate(term = reorder(term, tf_idf)) %>%
  ggplot(aes(term, tf_idf)) +
  geom_bar(stat = "identity") +
  facet_wrap(~ document, scales = "free") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



6.2 Casting tidy text data into a DocumentTermMatrix

Some existing text mining tools or algorithms work only on sparse document-term matrices. 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:

```
ap_td
```

```
## # A tibble: 302,031 x 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
```

```
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 it into a Term-Document Matrix with `cast_tdm`, or quanteda's dfm with `cast_dfm`:

```
# cast into a Term-Document Matrix
ap_td %>%
  cast_tdm(term, document, count)
```

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

```
# cast into quanteda's dfm
ap_td %>%
  cast_dfm(term, document, count)
```

```
## Document-feature matrix of: 10,473 documents, 2,246 features.
```

Some tools simply require a sparse matrix:

```
library(Matrix)

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

class(m)
```

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

```
dim(m)
```

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

This allows for 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.

6.3 Tidying corpus objects with metadata

You can also tidy Corpus objects from the tm package. For example, consider a Corpus containing 20 documents:

```
reut21578 <- system.file("texts", "crude", package = "tm")
reuters <- VCorpus(DirSource(reut21578),
                   readerControl = list(reader = readReut21578XMLasPlain))

reuters
```

```
## <<VCorpus>>
## Metadata:  corpus specific: 0, document level (indexed): 0
## Content:  documents: 20
```

The `tidy` verb creates a table with one row per document:

```
reuters_td <- tidy(reuters)
reuters_td
```

```
## Source: local data frame [20 x 17]
##
##                        author       datetimestamp description
##                         <chr>              <time>       <chr>
## 1                          NA 1987-02-26 12:00:56
## 2  BY TED D'AFFLISIO, Reuters 1987-02-26 12:34:11
## 3                          NA 1987-02-26 13:18:00
## 4                          NA 1987-02-26 13:21:01
## 5                          NA 1987-02-26 14:00:57
## 6                          NA 1987-02-28 22:25:46
## 7    By Jeremy Clift, Reuters 1987-02-28 22:39:14
## 8                          NA 1987-03-01 00:27:27
## 9                          NA 1987-03-01 03:22:30
## 10                         NA 1987-03-01 13:31:44
## 11                         NA 1987-03-01 20:05:49
## 12                         NA 1987-03-02 02:39:23
## 13                         NA 1987-03-02 02:43:22
## 14                         NA 1987-03-02 02:43:41
## 15                         NA 1987-03-02 03:25:42
## 16                         NA 1987-03-02 06:20:05
## 17                         NA 1987-03-02 06:28:26
## 18                         NA 1987-03-02 07:13:46
## 19 By BERNICE NAPACH, Reuters 1987-03-02 09:38:34
## 20                         NA 1987-03-02 09:49:06
##                                              heading    id language            origin
##                                                <chr> <chr>    <chr>             <chr>
## 1           DIAMOND SHAMROCK (DIA) CUTS CRUDE PRICES   127       en Reuters-21578 XML
## 2    OPEC MAY HAVE TO MEET TO FIRM PRICES - ANALYSTS   144       en Reuters-21578 XML
## 3          TEXACO CANADA <TXC> LOWERS CRUDE POSTINGS   191       en Reuters-21578 XML
## 4          MARATHON PETROLEUM REDUCES CRUDE POSTINGS   194       en Reuters-21578 XML
## 5          HOUSTON OIL <HO> RESERVES STUDY COMPLETED   211       en Reuters-21578 XML
## 6      KUWAIT SAYS NO PLANS FOR EMERGENCY OPEC TALKS   236       en Reuters-21578 XML
## 7  INDONESIA SEEN AT CROSSROADS OVER ECONOMIC CHANGE   237       en Reuters-21578 XML
## 8                SAUDI RIYAL DEPOSIT RATES REMAIN FIRM   242       en Reuters-21578 XML
## 9             QATAR UNVEILS BUDGET FOR FISCAL 1987/88   246       en Reuters-21578 XML
## 10   SAUDI ARABIA REITERATES COMMITMENT TO OPEC PACT   248       en Reuters-21578 XML
## 11    SAUDI FEBRUARY CRUDE OUTPUT PUT AT 3.5 MLN BPD   273       en Reuters-21578 XML
## 12 GULF ARAB DEPUTY OIL MINISTERS TO MEET IN BAHRAIN   349       en Reuters-21578 XML
## 13 SAUDI ARABIA REITERATES COMMITMENT TO OPEC ACCORD   352       en Reuters-21578 XML
## 14  KUWAIT MINISTER SAYS NO EMERGENCY OPEC TALKS SET   353       en Reuters-21578 XML
## 15          PHILADELPHIA PORT CLOSED BY TANKER CRASH   368       en Reuters-21578 XML
## 16     STUDY GROUP URGES INCREASED U.S. OIL RESERVES   489       en Reuters-21578 XML
## 17     STUDY GROUP URGES INCREASED U.S. OIL RESERVES   502       en Reuters-21578 XML
## 18    UNOCAL <UCL> UNIT CUTS CRUDE OIL POSTED PRICES   543       en Reuters-21578 XML
## 19      NYMEX WILL EXPAND OFF-HOUR TRADING APRIL ONE   704       en Reuters-21578 XML
## 20     ARGENTINE OIL PRODUCTION DOWN IN JANUARY 1987   708       en Reuters-21578 XML
## Variables not shown: topics <chr>, lewissplit <chr>, cgisplit <chr>, oldid <chr>, topics_cat <list>, plac
##   exchanges <chr>, text <chr>.
```

Another variation of a corpus object is `corpus` from the quanteda package:

```
library(quanteda)

data("inaugCorpus")

inaugCorpus
```

```
## Corpus consisting of 57 documents.
```

```
inaug_td <- tidy(inaugCorpus)
inaug_td
```

```
## Source: local data frame [57 x 4]
##
##
##                                                                                                              <
## 1  Fellow-Citizens of the Senate and of the House of Representatives:\n\nAmong the vicissitudes incident t
## 2   Fellow citizens, I am again called upon by the voice of my country to execute the functions of its Chie
## 3   When it was first perceived, in early times, that no middle course for America remained between unlimi
## 4  Friends and Fellow Citizens:\n\nCalled upon to undertake the duties of the first executive office of ou
## 5   Proceeding, fellow citizens, to that qualification which the Constitution requires before my entrance
## 6   Unwilling to depart from examples of the most revered authority, I avail myself of the occasion now pre
## 7   About to add the solemnity of an oath to the obligations imposed by a second call to the station in whic
## 8   I should be destitute of feeling if I was not deeply affected by the strong proof which my fellow-citiz
## 9   Fellow citizens, I shall not attempt to describe the grateful emotions which the new and very distingui
## 10   In compliance with an usage coeval with the existence of our Federal Constitution, and sanctioned by t
## ..
## Variables not shown: Year <int>, President <chr>, FirstName <chr>.
```

This lets us work with tidy tools like `unnest_tokens` to analyze the text alongside the metadata.

```
inaug_words <- inaug_td %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words)

inaug_words
```

```
## Source: local data frame [49,621 x 4]
##
##     Year President FirstName         word
##    <int>     <chr>     <chr>        <chr>
## 1   2013     Obama    Barack        waves
## 2   2013     Obama    Barack     realizes
## 3   2013     Obama    Barack philadelphia
## 4   2013     Obama    Barack          400
## 5   2013     Obama    Barack           40
## 6   2013     Obama    Barack   absolutism
## 7   2013     Obama    Barack      contour
## 8   2013     Obama    Barack      newtown
## 9   2013     Obama    Barack        lanes
## 10  2013     Obama    Barack   appalachia
## ..   ...       ...       ...          ...
```

We could then, for example, see how the appearance of a word changes over time:

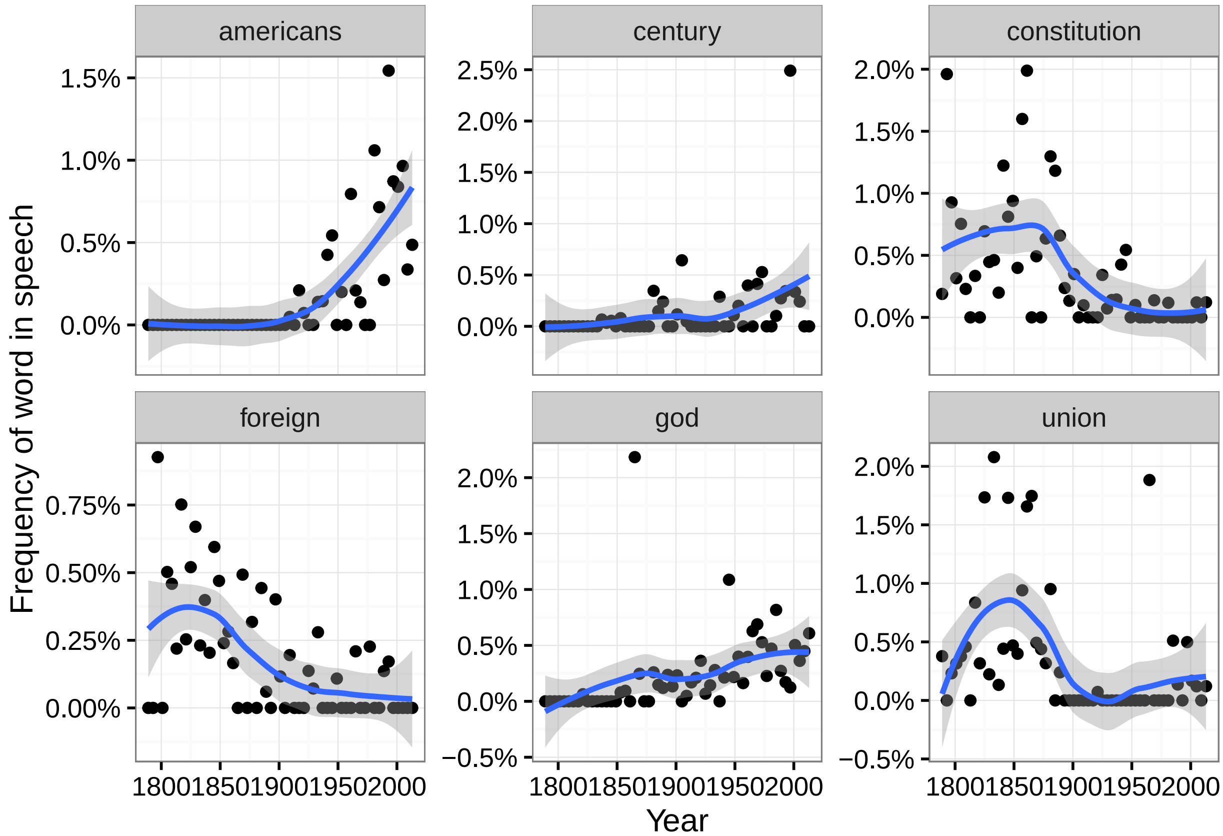
```
library(tidyr)

inaug_freq <- inaug_words %>%
```

```r
  count(Year, word) %>%
  ungroup() %>%
  complete(Year, word, fill = list(n = 0)) %>%
  group_by(Year) %>%
  mutate(year_total = sum(n),
         percent = n / year_total) %>%
  ungroup()

inaug_freq %>%
  filter(word == "america")
```

```
## # A tibble: 57 x 5
##     Year    word     n year_total     percent
##    <int>   <chr> <dbl>      <dbl>       <dbl>
## 1   1789 america     0        529 0.000000000
## 2   1793 america     1         51 0.019607843
## 3   1797 america     5        863 0.005793743
## 4   1801 america     0        634 0.000000000
## 5   1805 america     0        796 0.000000000
## 6   1809 america     0        436 0.000000000
## 7   1813 america     0        456 0.000000000
## 8   1817 america     0       1197 0.000000000
## 9   1821 america     2       1578 0.001267427
## 10  1825 america     0       1153 0.000000000
## # ... with 47 more rows
```

For instance, we could display the top 6 terms that have changed in frequency over time.

```r
library(scales)

inaug_freq %>%
  filter(word %in% c("americans", "century", "foreign", "god",
                     "union", "constitution")) %>%
  ggplot(aes(Year, percent)) +
  geom_point() +
  geom_smooth() +
  facet_wrap(~ word, scales = "free_y") +
  scale_y_continuous(labels = percent_format()) +
  ylab("Frequency of word in speech")
```


americans
century
constitution
foreign
god
union
Frequency of word in speech
Year
0.0%
0.5%
1.0%
1.5%
2.0%
2.5%
0.00%
0.25%
0.50%
0.75%
−0.5%
1800
1850
1900
1950
2000

Chapter 7

Topic Modeling

7.1 Topic modeling

Topic modeling is a method for unsupervised classification of documents, by modeling each document as a mixture of topics and each topic as a mixture of words. Latent Dirichlet allocation is a particularly popular method for fitting a topic model.

We can use tidy text principles, as described in Chapter 2, to approach topic modeling using consistent and effective tools. In particular, we'll be using tidying functions for LDA objects from the topicmodels package. u789

7.2 Setup

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? We'll use topic modeling to discover how chapters are distinguished into distinct topics.

We'll retrieve these four books using the gutenbergr package:

```
library(dplyr)
library(gutenbergr)

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

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

books
```

```
## Source: local data frame [51,663 x 3]
##
##    gutenberg_id                                              text                title
```

```
##         <int>                                               <chr>               <chr>
## 1         36                          The War of the Worlds The War of the Worlds
## 2         36                                                The War of the Worlds
## 3         36                         by H. G. Wells [1898] The War of the Worlds
## 4         36                                                The War of the Worlds
## 5         36                                                The War of the Worlds
## 6         36         But who shall dwell in these worlds if they be The War of the Worlds
## 7         36         inhabited? .   .   .  Are we or they Lords of the The War of the Worlds
## 8         36   World? .   .   .  And how are all things made for man?-- The War of the Worlds
## 9         36            KEPLER (quoted in The Anatomy of Melancholy) The War of the Worlds
## 10        36                                                The War of the Worlds
## ..       ...                                                 ...                 ...
```

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`.

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

by_chapter <- books %>%
  group_by(title) %>%
  mutate(chapter = cumsum(str_detect(text, regex("^chapter ", ignore_case = TRUE)))) %>%
  ungroup() %>%
  filter(chapter > 0)

by_chapter_word <- by_chapter %>%
  unite(title_chapter, title, chapter) %>%
  unnest_tokens(word, text)

word_counts <- by_chapter_word %>%
  anti_join(stop_words) %>%
  count(title_chapter, word, sort = TRUE) %>%
  ungroup()

word_counts
```

```
## Source: local data frame [104,721 x 3]
##
##               title_chapter    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
## ..                      ...     ...   ...
```

7.3 Latent Dirichlet Allocation with the topicmodels package

Right now this data frame 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 this vignette, we can cast a one-token-per-row table into a `DocumentTermMatrix` with tidytext's `cast_dtm`:

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

chapters_dtm
```

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

Now we are ready to use the topicmodels package to create a four topic LDA model.

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

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

(In this case we know there are four topics because there are four books; in practice we may need to try a few different values of `k`).

Now tidytext gives us the option of *returning* to a tidy analysis, using the `tidy` and `augment` verbs borrowed from the broom package. In particular, we start with the `tidy` verb.

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

```
## Source: local data frame [72,860 x 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
## ..   ...     ...          ...
```

Notice that this has turned the model into a one-topic-per-term-per-row format. For each combination the model has β, the probability of that term being generated from that topic.

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

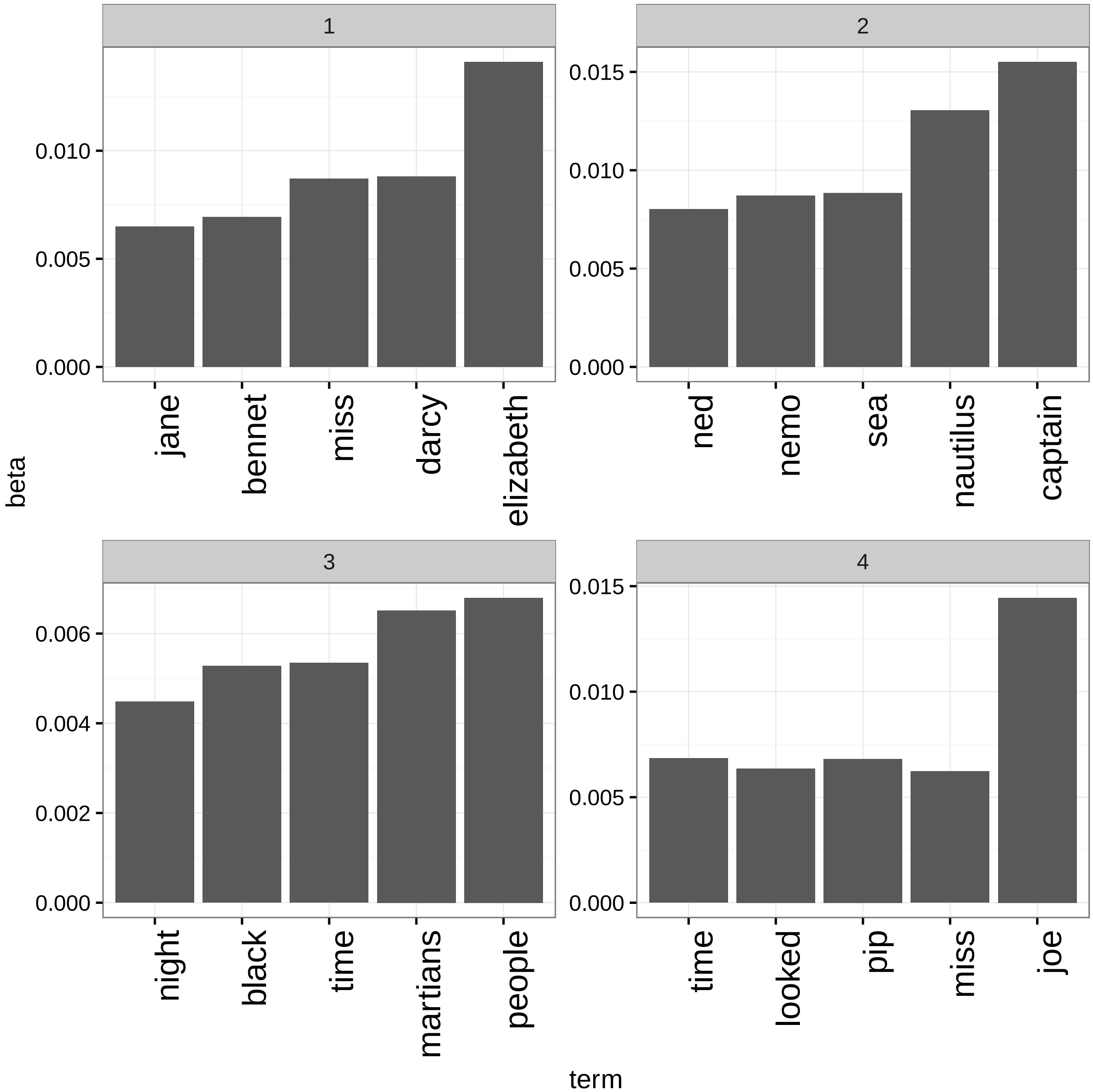
```
top_terms <- chapters_lda_td %>%
  group_by(topic) %>%
  top_n(5, beta) %>%
  ungroup() %>%
```

```
  arrange(topic, -beta)

top_terms
```

```
## Source: local data frame [20 x 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 model lends itself to a visualization:

```
library(ggplot2)
theme_set(theme_bw())

top_terms %>%
  mutate(term = reorder(term, beta)) %>%
  ggplot(aes(term, beta)) +
  geom_bar(stat = "identity") +
  facet_wrap(~ topic, scales = "free") +
  theme(axis.text.x = element_text(size = 15, angle = 90, hjust = 1))
```



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*.

7.4 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?

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

```
## Source: local data frame [772 x 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
## ..                      ...   ...          ...
```

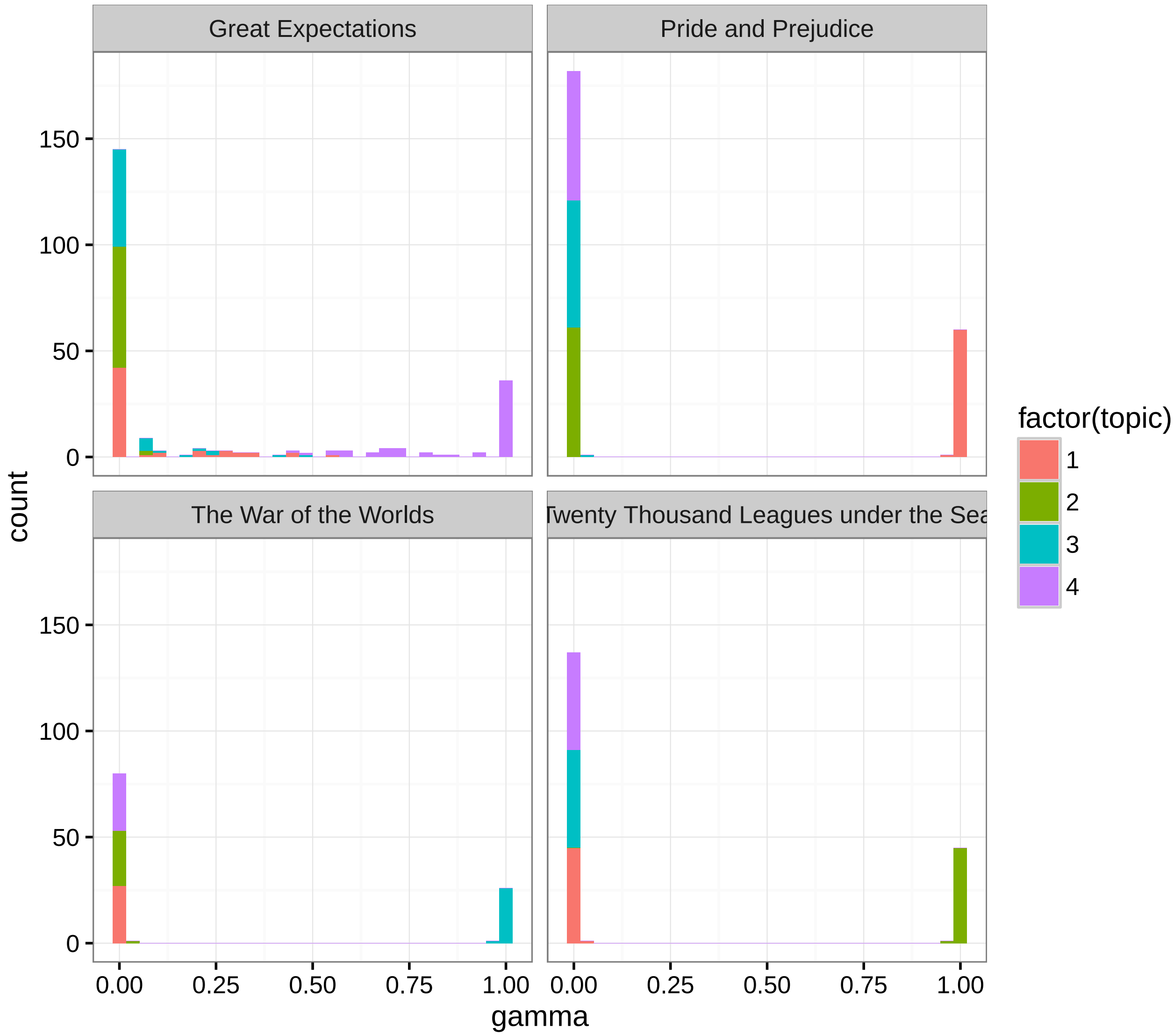
Setting `matrix = "gamma"` returns a tidied version with one-document-per-topic-per-row. Now that we have these document classifiations, we can see how well our unsupervised learning did at distinguishing the four books. First we re-separate the document name into title and chapter:

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

```
## Source: local data frame [772 x 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
## ..                   ...     ...   ...          ...
```

Then we examine what fraction of chapters we got right for each:

```
ggplot(chapters_lda_gamma, aes(gamma, fill = factor(topic))) +
  geom_histogram() +
  facet_wrap(~ title, nrow = 2)
```



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.

```
chapter_classifications <- chapters_lda_gamma %>%
  group_by(title, chapter) %>%
  top_n(1, gamma) %>%
  ungroup() %>%
  arrange(gamma)

chapter_classifications
```

```
## Source: local data frame [193 x 4]
##
##                  title chapter topic     gamma
##                  <chr>   <int> <int>     <dbl>
## 1  Great Expectations      54     3 0.4803234
## 2  Great Expectations      22     4 0.5356506
## 3  Great Expectations      31     4 0.5464851
## 4  Great Expectations      23     1 0.5507241
```

```
## 5  Great Expectations      33     4 0.5700737
## 6  Great Expectations      47     4 0.5802089
## 7  Great Expectations      56     4 0.5984806
## 8  Great Expectations      38     4 0.6455341
## 9  Great Expectations      11     4 0.6689600
## 10 Great Expectations      44     4 0.6777974
## ..                ...     ...   ...       ...
```

We can determine this by finding the consensus book for each, which we note is correct based on our earlier visualization:

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

book_topics
```

```
## Source: local data frame [4 x 2]
##
##                              consensus topic
##                                  <chr> <int>
## 1                   Great Expectations     4
## 2                  Pride and Prejudice     1
## 3                The War of the Worlds     3
## 4 Twenty Thousand Leagues under the Sea     2
```

Then we see which chapters were misidentified:

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

```
## Source: local data frame [6 x 3]
## Groups: title [?]
##
##                                   title                             consensus     n
##                                   <chr>                                 <chr> <int>
## 1                    Great Expectations                    Great Expectations    57
## 2                    Great Expectations                   Pride and Prejudice     1
## 3                    Great Expectations                 The War of the Worlds     1
## 4                   Pride and Prejudice                   Pride and Prejudice    61
## 5                 The War of the Worlds                 The War of the Worlds    27
## 6 Twenty Thousand Leagues under the Sea Twenty Thousand Leagues under the Sea    46
```

We see that only a few chapters from *Great Expectations* were misclassified. Not bad for unsupervised clustering!

7.4.1 By word assignments: `augment`

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

We may want to take the original document-word pairs and find which words in each document were assigned to which topic. This is the job of the `augment` verb.

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

We can combine this with the consensus book titles to find which words were incorrectly classified.

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

assignments
```

```
## # A tibble: 104,721 x 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" using dplyr's `count` and tidyr's `spread`:

```r
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.

What were the most commonly mistaken words?

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

wrong_words
```

```
## # A tibble: 4,535 x 6
```

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

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

```
## # A tibble: 3,500 x 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
```

Notice the word "flopson" here; these wrong words do not necessarily appear in the novels they were misassigned to. Indeed, we can confirm "flopson" appears only in *Great Expectations*:

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

```
## # A tibble: 3 x 3
##          title_chapter    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.

Chapter 8

Case Study: Analyzing Usenet Text

Here we'll use what we've learned in the book to perform a start-to-finish analysis of the Usenet

8.1 Setup

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

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

```
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:

```
# 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))
```

```
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)
```

We could simply find the most common words:

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

```
## # A tibble: 63,937 x 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 we could look at 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
```

8.1.1 TF-IDF

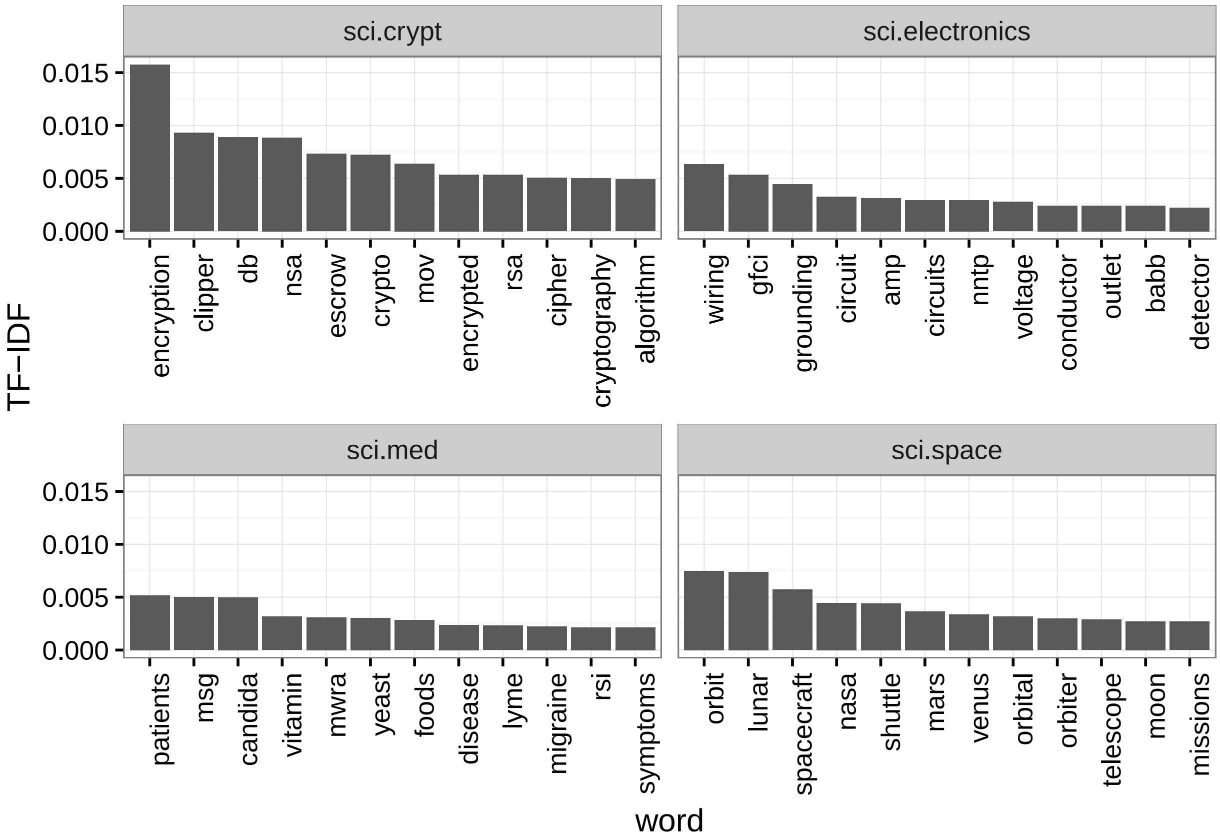
We notice that some words are likely to be more common on particular boards. Let's try quantifying this using the TF-IDF metric we learned in Chapter 4.

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

tf_idf
```

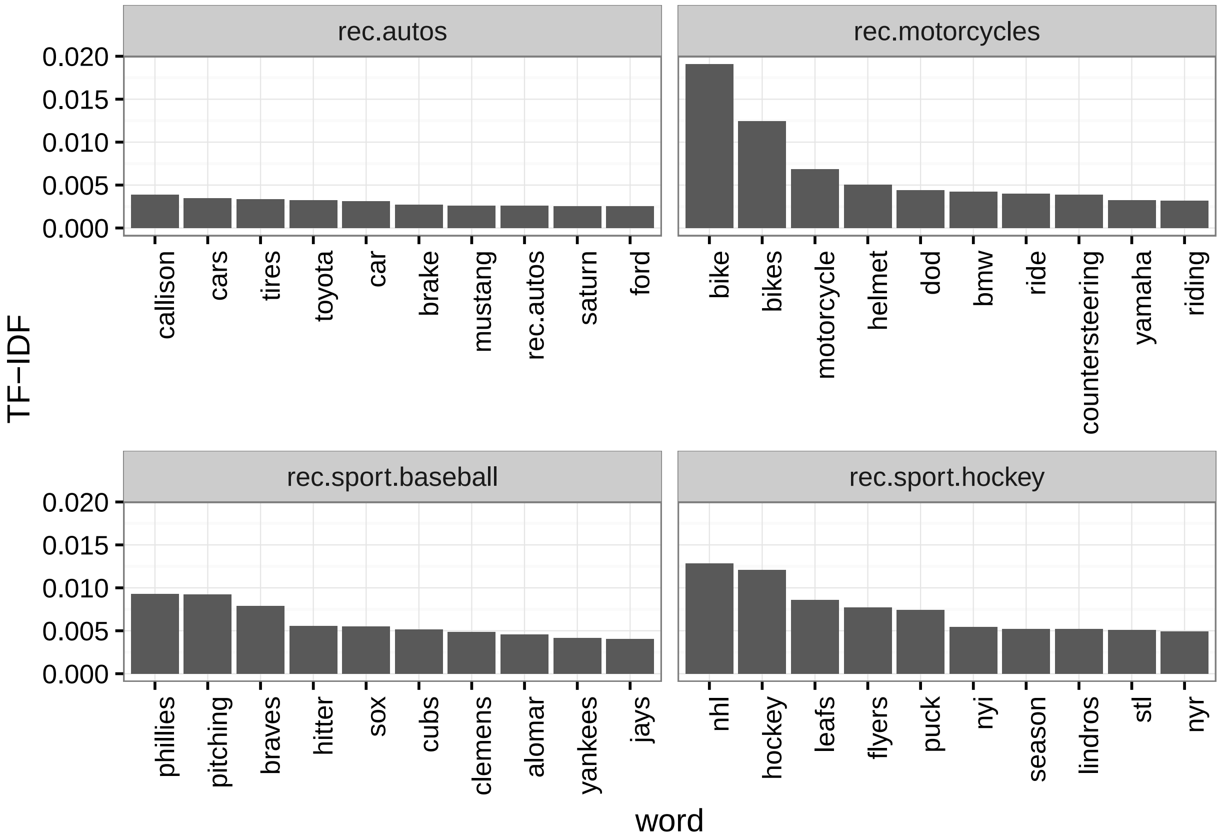
```
## # A tibble: 166,528 x 6
##                       board          word     n          tf      idf     tf_idf
##                       <chr>         <chr> <int>       <dbl>    <dbl>      <dbl>
## 1  comp.sys.ibm.pc.hardware          scsi   483 0.018138801 1.203973 0.02183862
## 2           rec.motorcycles          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
```

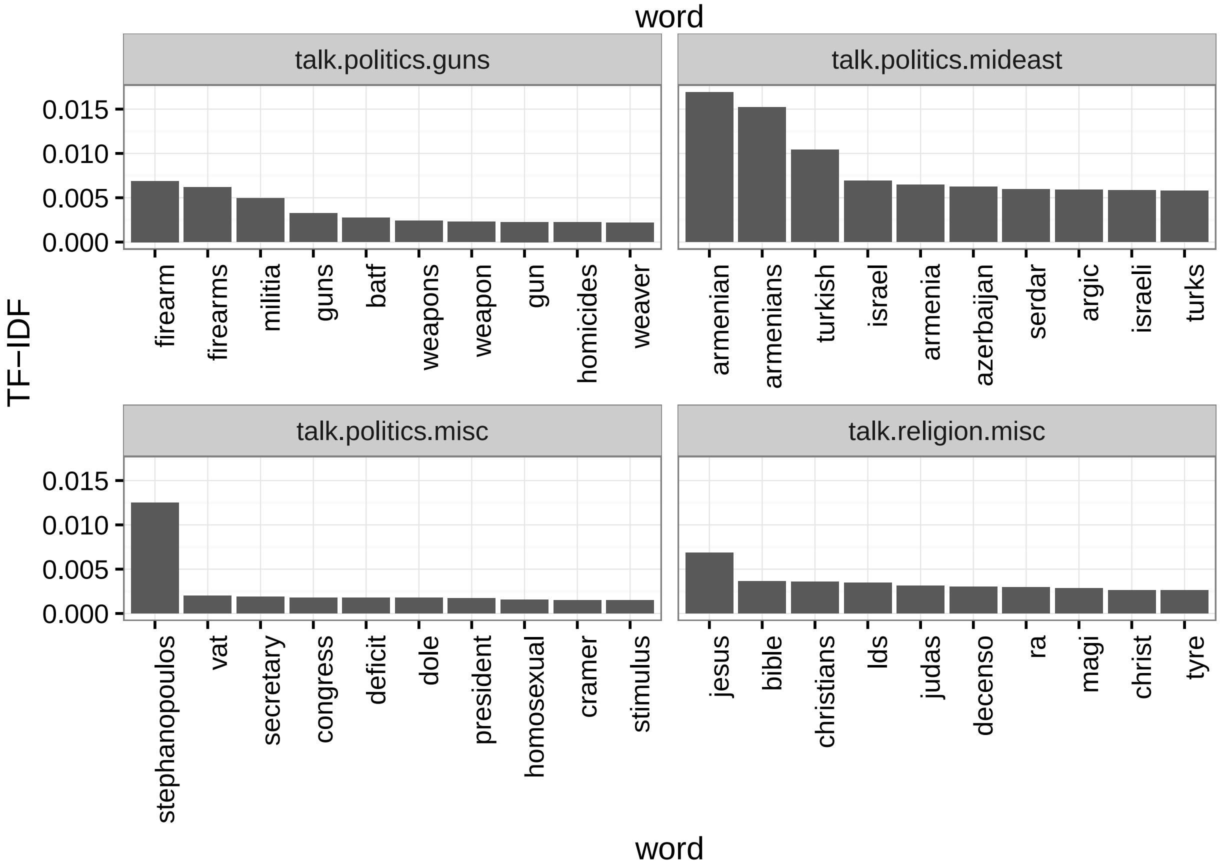
We can visualize this for a few select boards. For example, let's look at all the `sci.` boards:

```
library(ggplot2)
theme_set(theme_bw())

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)) +
  geom_bar(stat = "identity") +
  facet_wrap(~ board, scales = "free_x") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("TF-IDF")
```



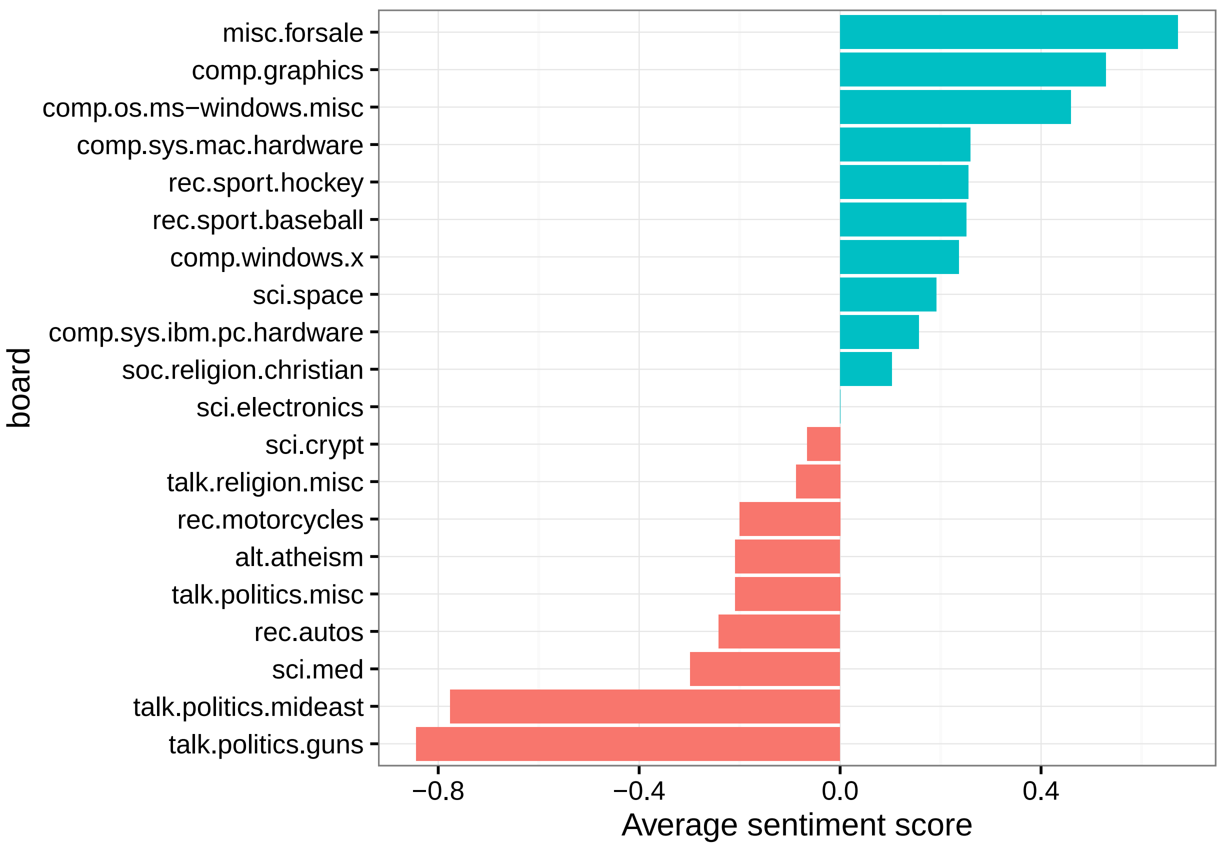
We could use almost the same code (not shown) to compare the "rec." (recreation) or "talk." boards:


TF−IDF
word
rec.autos
0.020
0.015
0.010
0.005
0.000
callison
cars
tires
toyota
car
brake
mustang
rec.autos
saturn
ford
rec.motorcycles
bike
bikes
motorcycle
helmet
dod
bmw
ride
countersteering
yamaha
riding
rec.sport.baseball
phillies
pitching
braves
hitter
sox
cubs
clemens
alomar
yankees
jays
rec.sport.hockey
nhl
hockey
leafs
flyers
puck
nyi
season
lindros
stl
nyr


TF−IDF
word
talk.politics.guns
0.015
0.010
0.005
0.000
firearm
firearms
militia
guns
batf
weapons
weapon
gun
homicides
weaver
talk.politics.mideast
armenian
armenians
turkish
israel
armenia
azerbaijan
serdar
argic
israeli
turks
talk.politics.misc
stephanopoulos
vat
secretary
congress
deficit
dole
president
homosexual
cramer
stimulus
talk.religion.misc
jesus
bible
christians
lds
judas
decenso
ra
magi
christ
tyre

8.1.2 Sentiment Analysis

```
AFINN <- sentiments %>%
  filter(lexicon == "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(stat = "identity", show.legend = FALSE) +
  coord_flip() +
  ylab("Average sentiment score")
```



8.1.3 Looking by word

It's worth discovering *why* some topics ended up more positive then others. For that, we can examine the total positive and negative contributions of each word:

```
contributions <- usenet_words %>%
  inner_join(AFINN, by = "word") %>%
```

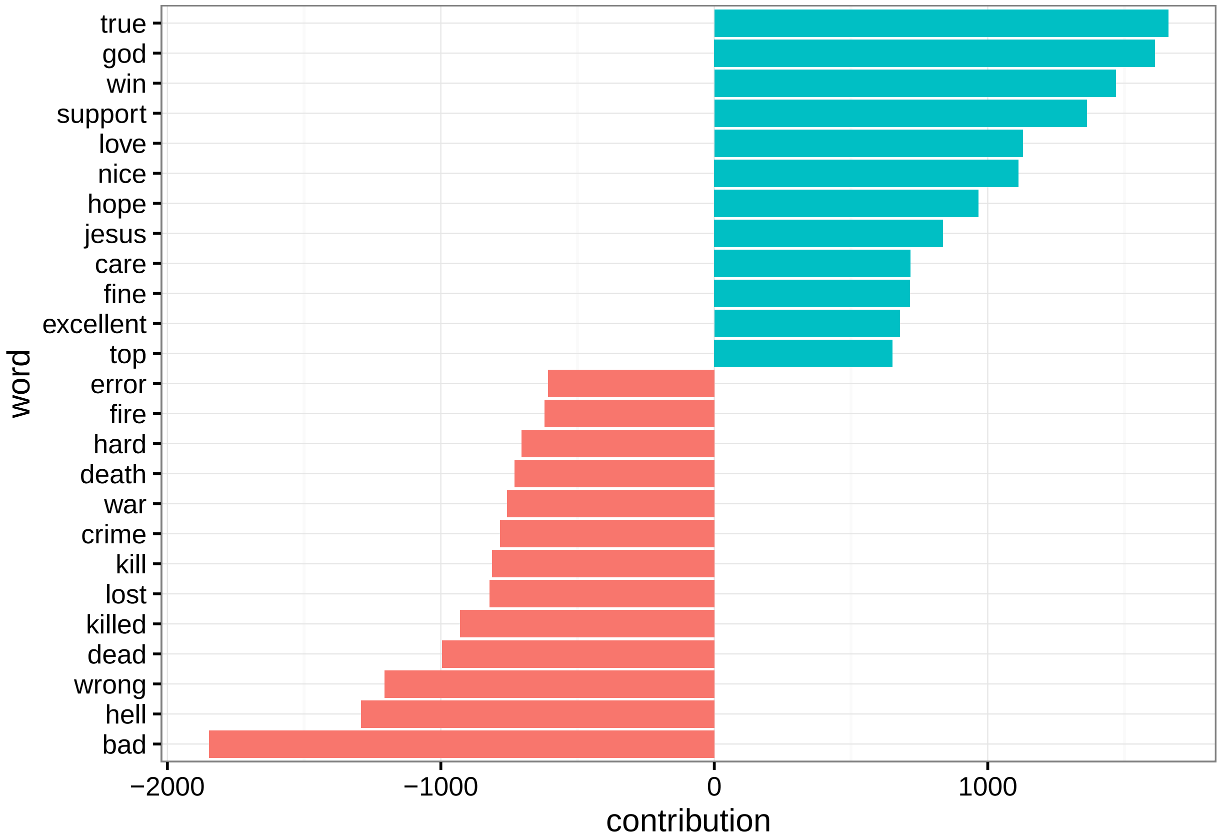
```
  group_by(word) %>%
  summarize(occurences = n(),
            contribution = sum(score))

contributions
```

```
## # A tibble: 1,891 x 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
```

We can visualize which words had the most effect:

```
library(ggplot2)
theme_set(theme_bw())
```

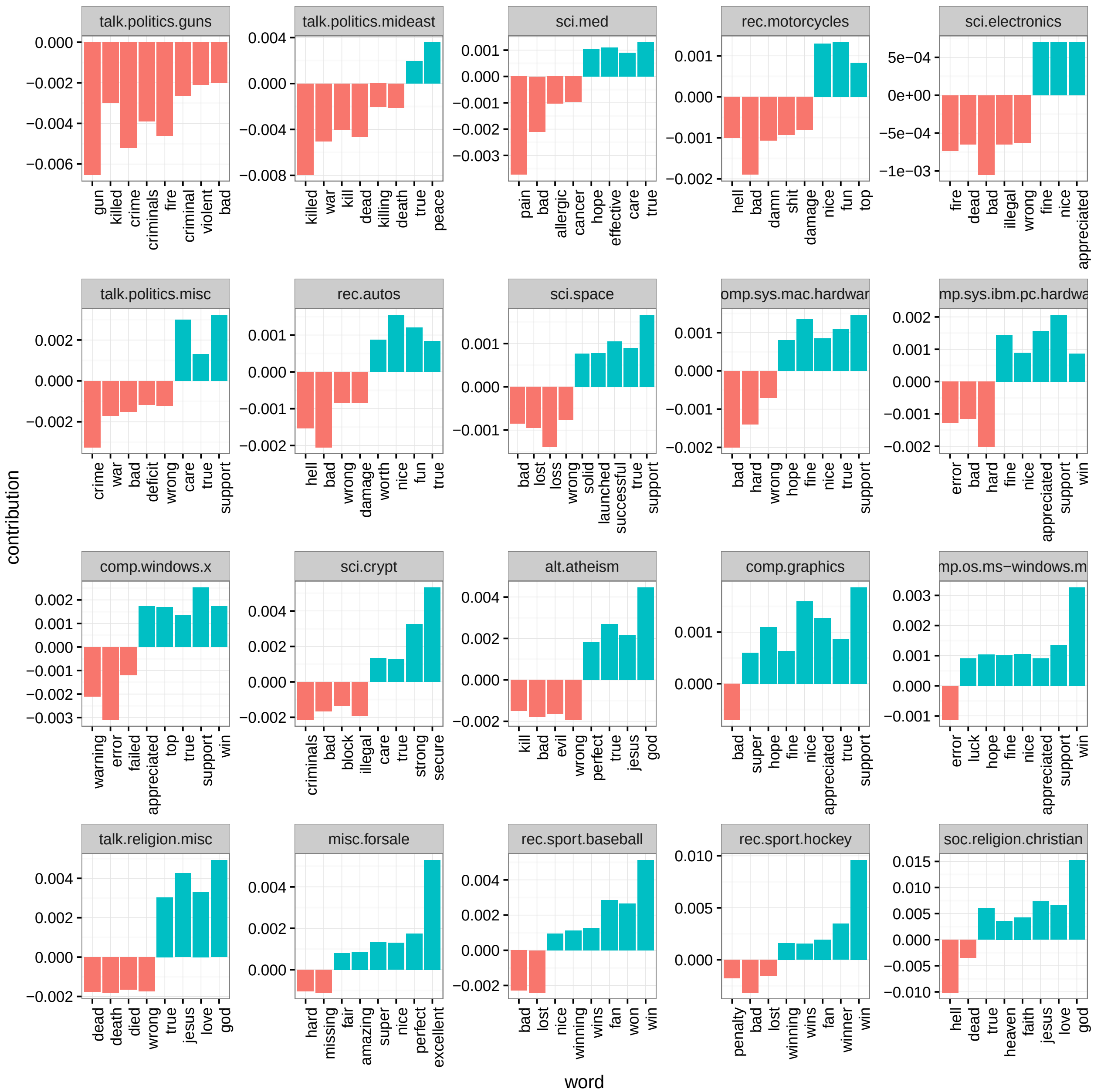
```
contributions %>%
  top_n(25, abs(contribution)) %>%
  mutate(word = reorder(word, contribution)) %>%
  ggplot(aes(word, contribution, fill = contribution > 0)) +
  geom_bar(stat = "identity", show.legend = FALSE) +
  coord_flip()
```



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.

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(stat = "identity", show.legend = FALSE) +
  facet_wrap(~ board, scales = "free") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



We can also see 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 were positive.

8.1.3.1 Sentiment analysis by message

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

```
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 was the most positive messages?

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

```
## # A tibble: 3,385 x 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 message?

```
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:

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

```
## # A tibble: 3,385 x 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
```

```r
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--
```

8.1.4 N-grams

We can also

```r
usenet_digrams <- cleaned_text %>%
  unnest_tokens(digram, text, token = "ngrams", n = 2)
```

```
usenet_digram_counts <- usenet_digrams %>%
  count(board, digram)
```

```
digram_tf_idf <- usenet_digram_counts %>%
  bind_tf_idf(digram, board, n)
```

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

usenet_digram_counts %>%
  ungroup() %>%
  separate(digram, 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(stat = "identity", show.legend = FALSE) +
  facet_wrap(~ word1, scales = "free", nrow = 2) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```