Tidy Text Mining

Julia Silge and David Robinson

2016-07-06

Contents

Chapter 1

Introduction

This intro will be changed a lot to serve as a general and friendly intro to the topic.

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

Tidy data sets allow manipulation with a standard set of "tidy" tools, including popular packages such as dplyr(**?**), ggplot2(**?**), and broom(**?**). These tools do not yet, however, have the infrastructure to work fluently with text data and natural language processing tools. In developing this package, we provide functions and supporting data sets to allow conversion of text to and from tidy formats, and to switch seamlessly between tidy tools and existing text mining packages.

We define the tidy text format as being one-token-per-document-per-row, and provide functionality to tokenize by commonly used units of text including words, n-grams, and sentences. 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 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 specific to a particular document. (Other document stuff in this chapter perhaps?)

Text won't be tidy at all stages of an analysis.

- **Chapter 5** 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 6** introduces the concept of topic modeling, and uses the `tidy` method for interpreting and visualizing the output of the topicmodels package.
- **Chapter 7** (TODO) introduces tidying methods for the glove package, which offer an interface to word2vec models. (*These methods are still being implemented so this chapter is far from written!*)

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

- **Chapter 8** demonstrates an application of a tidy text analysis on the Yelp restaurant review dataset. We show a few approaches to predicting a star rating from a review's text, and see how well sentiment analysis (from Chapter 3) does at this task.
- **Chapter 9** TODO: find at least one other in-depth exploration of text data. Optional but I think would conclude the book well.

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
```

```
## # A tibble: 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:

```
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 Example: 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
```

```
## # A tibble: 73,422 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
```

```
## # ... with 73,412 more rows
```

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:

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

tidy_books
```

```
## # A tibble: 725,054 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
## # ... with 725,044 more rows
```

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`.

```
data("stop_words")

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

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

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

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

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

```
library(ggplot2)

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

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

2.2.1 The gutenbergr package

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

Chapter 3

Sentiment Analysis with Tidy Data

3.1 The sentiments dataset

The

```
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
```

3.2 Sentiment analysis with inner join

Sentiment analysis can be done as an inner join. Three sentiment lexicons are in the tidytext package in the `sentiment` dataset. 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>
```

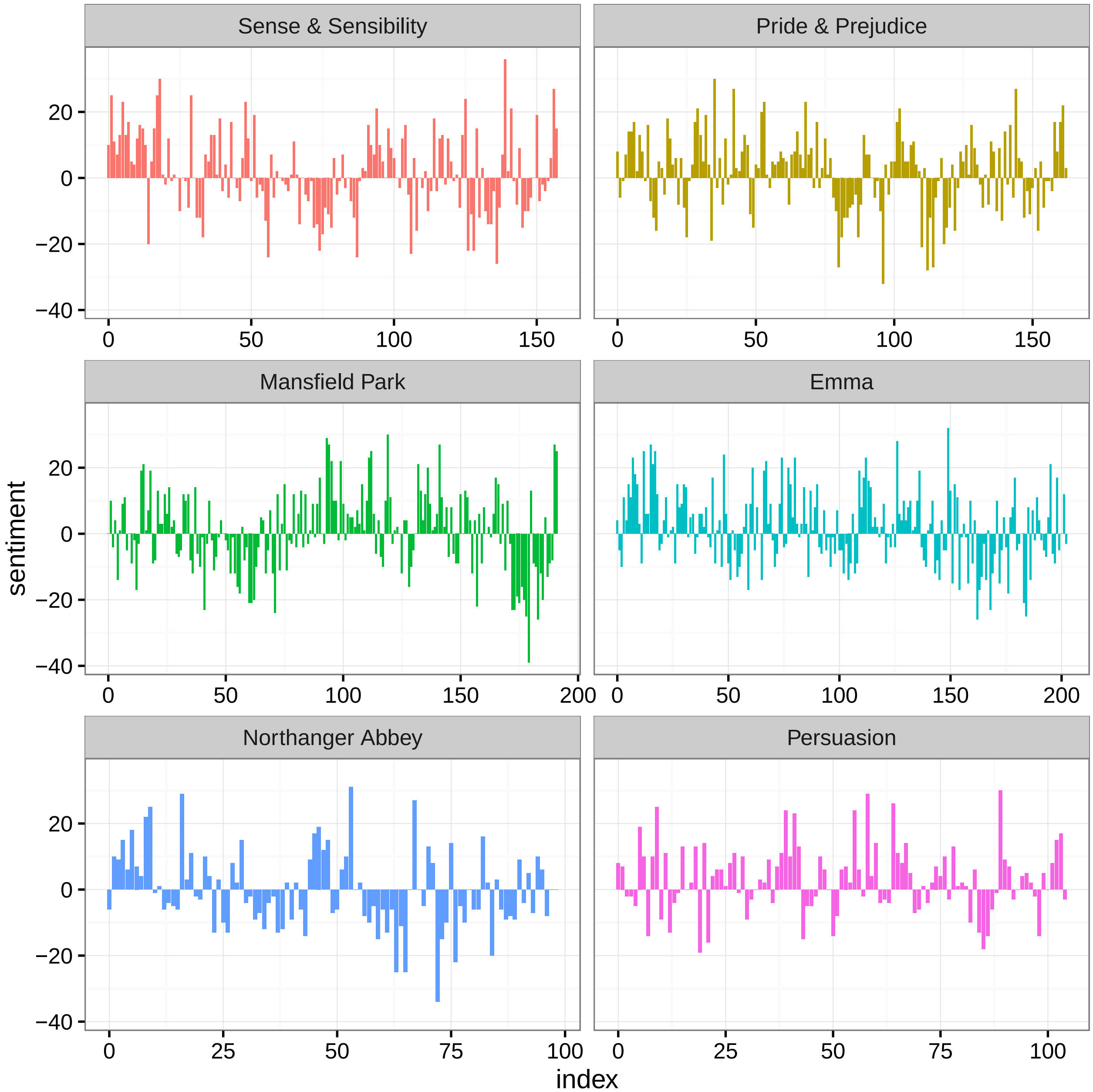
```
## 1     friend   166
## 2       hope   143
## 3      happy   125
## 4       love   117
## 5       deal    92
## 6      found    92
## 7  happiness    76
## 8     pretty    68
## 9       true    66
## 10   comfort    65
## # ... with 288 more rows
```

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)

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



3.2.1 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.

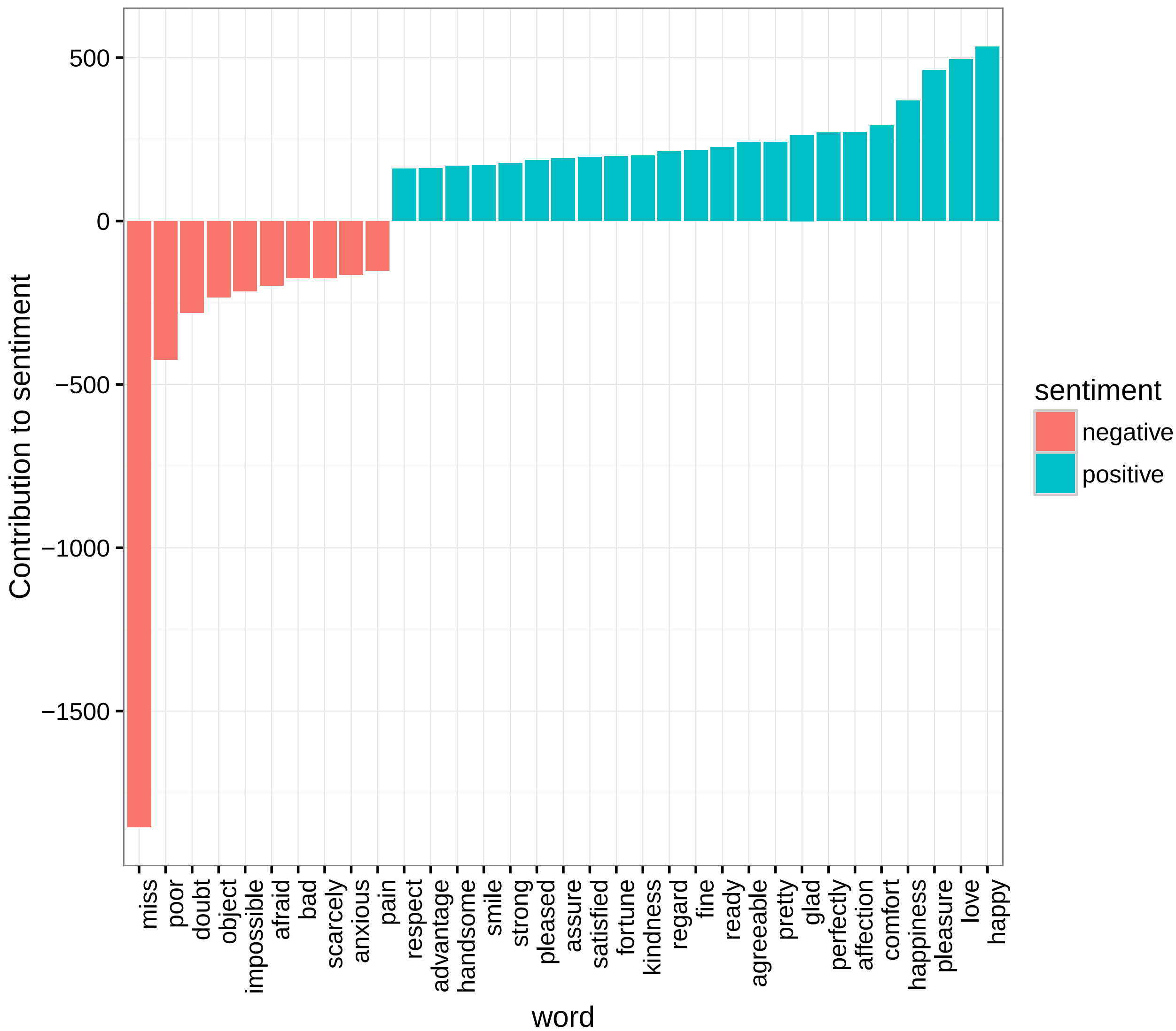
```
bing_word_counts <- tidy_books %>%
  inner_join(bing) %>%
  count(word, sentiment, sort = TRUE) %>%
  ungroup()

bing_word_counts
```

```
## # A tibble: 2,555 x 3
##          word sentiment     n
```

```
##          <chr>     <chr> <int>
## 1  abominable  negative    17
## 2  abominably  negative     7
## 3   abominate  negative     3
## 4      abound  positive     1
## 5      abrupt  negative     5
## 6    abruptly  negative    12
## 7     absence  negative   111
## 8      absurd  negative    19
## 9   absurdity  negative    12
## 10  abundance  positive    14
## # ... with 2,545 more rows
```

This can be shown visually, and we can pipe straight into ggplot2 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`.

3.2.2 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)
```

3.2.3 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.

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 the main vignette, 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.

Let’s look 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.)

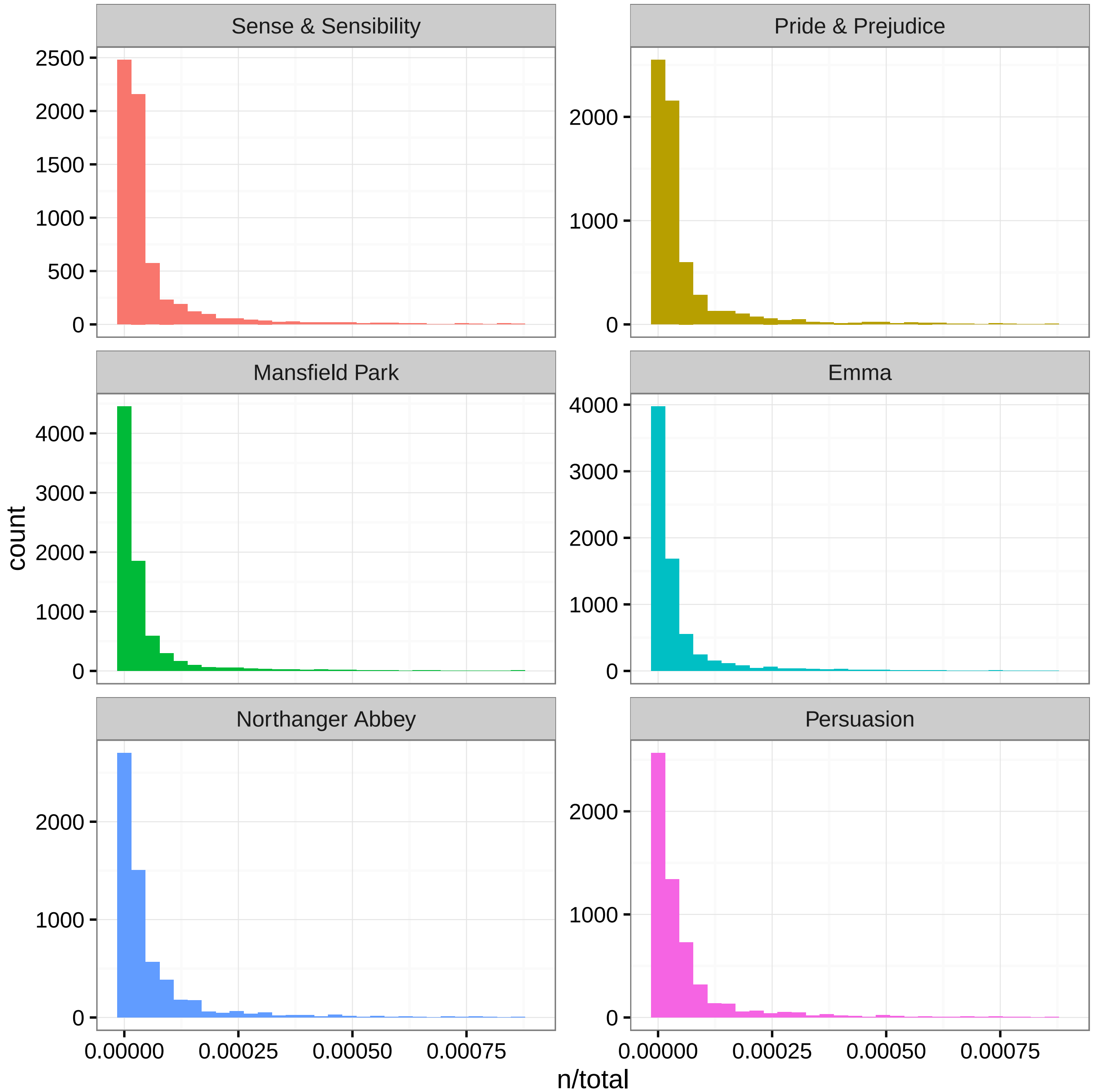
```
library(dplyr)
library(janeaustenr)
library(tidytext)
book_words <- austen_books() %>%
  unnest_tokens(word, text) %>%
```

```
  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  Sense & Sensibility    to  4116 119957
## 2  Sense & Sensibility   the  4105 119957
## 3  Sense & Sensibility    of  3571 119957
## 4  Sense & Sensibility   and  3490 119957
## 5  Sense & Sensibility   her  2543 119957
## 6  Sense & Sensibility     a  2092 119957
## 7  Sense & Sensibility     i  1998 119957
## 8  Sense & Sensibility    in  1979 119957
## 9  Sense & Sensibility   was  1861 119957
## 10 Sense & Sensibility    it  1755 119957
## # ... 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)
ggplot(book_words, aes(n/total, fill = book)) +
  geom_histogram(show.legend = FALSE) +
  xlim(NA, 0.0009) +
  facet_wrap(~book, ncol = 2, scales = "free_y")
```



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

4.2 The bind_tf_idf function

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

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

```
book_words
```

```
## # A tibble: 40,379 x 7
##                   book  word     n  total         tf   idf tf_idf
##                 <fctr> <chr> <int>  <int>      <dbl> <dbl>  <dbl>
## 1  Sense & Sensibility    to  4116 119957 0.03431230     0      0
## 2  Sense & Sensibility   the  4105 119957 0.03422060     0      0
## 3  Sense & Sensibility    of  3571 119957 0.02976900     0      0
## 4  Sense & Sensibility   and  3490 119957 0.02909376     0      0
## 5  Sense & Sensibility   her  2543 119957 0.02119926     0      0
## 6  Sense & Sensibility     a  2092 119957 0.01743958     0      0
## 7  Sense & Sensibility     i  1998 119957 0.01665597     0      0
## 8  Sense & Sensibility    in  1979 119957 0.01649758     0      0
## 9  Sense & Sensibility   was  1861 119957 0.01551389     0      0
## 10 Sense & Sensibility    it  1755 119957 0.01463024     0      0
## # ... with 40,369 more rows
```

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

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

```
## # A tibble: 40,379 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 specifically at *Pride and Prejudice*.

```
book_words %>%
  filter(book == "Pride & Prejudice") %>%
  select(-total) %>%
  arrange(desc(tf_idf))
```

```
## # A tibble: 6,538 x 6
##                 book      word     n           tf       idf      tf_idf
##               <fctr>     <chr> <int>        <dbl>     <dbl>       <dbl>
## 1  Pride & Prejudice     darcy   373 0.0030522732 1.7917595 0.005468939
```

```
## 2  Pride & Prejudice    bennet   294 0.0024058132 1.7917595 0.004310639
## 3  Pride & Prejudice   bingley   257 0.0021030408 1.7917595 0.003768143
## 4  Pride & Prejudice elizabeth   597 0.0048852738 0.6931472 0.003386214
## 5  Pride & Prejudice   wickham   162 0.0013256522 1.7917595 0.002375250
## 6  Pride & Prejudice   collins   156 0.0012765540 1.7917595 0.002287278
## 7  Pride & Prejudice     lydia   133 0.0010883441 1.7917595 0.001950051
## 8  Pride & Prejudice     lizzy    95 0.0007773886 1.7917595 0.001392893
## 9  Pride & Prejudice longbourn    88 0.0007201074 1.7917595 0.001290259
## 10 Pride & Prejudice  gardiner    84 0.0006873752 1.7917595 0.001231611
## # ... with 6,528 more rows
```

These words are, as measured by tf-idf, the most important to *Pride and Prejudice* and most readers would likely agree.

Chapter 5

Tidying and casting document-term matrices

Intro text here.

5.1 Tidying a document-term matrix

Many existing text mining datasets are in the form of a `DocumentTermMatrix` class (from the tm package). For example, consider the corpus of 2246 Associated Press articles from the topicmodels package:

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

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

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 `tidy` function does this. (For more on the tidy verb, see the broom package).

```
library(dplyr)
library(tidytext)

ap_td <- tidy(AssociatedPress)
```

Just as shown in this vignette, having the text in this format is convenient for analysis with the tidytext package. 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
```

```
## # A tibble: 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
## # ... with 30,084 more rows
```

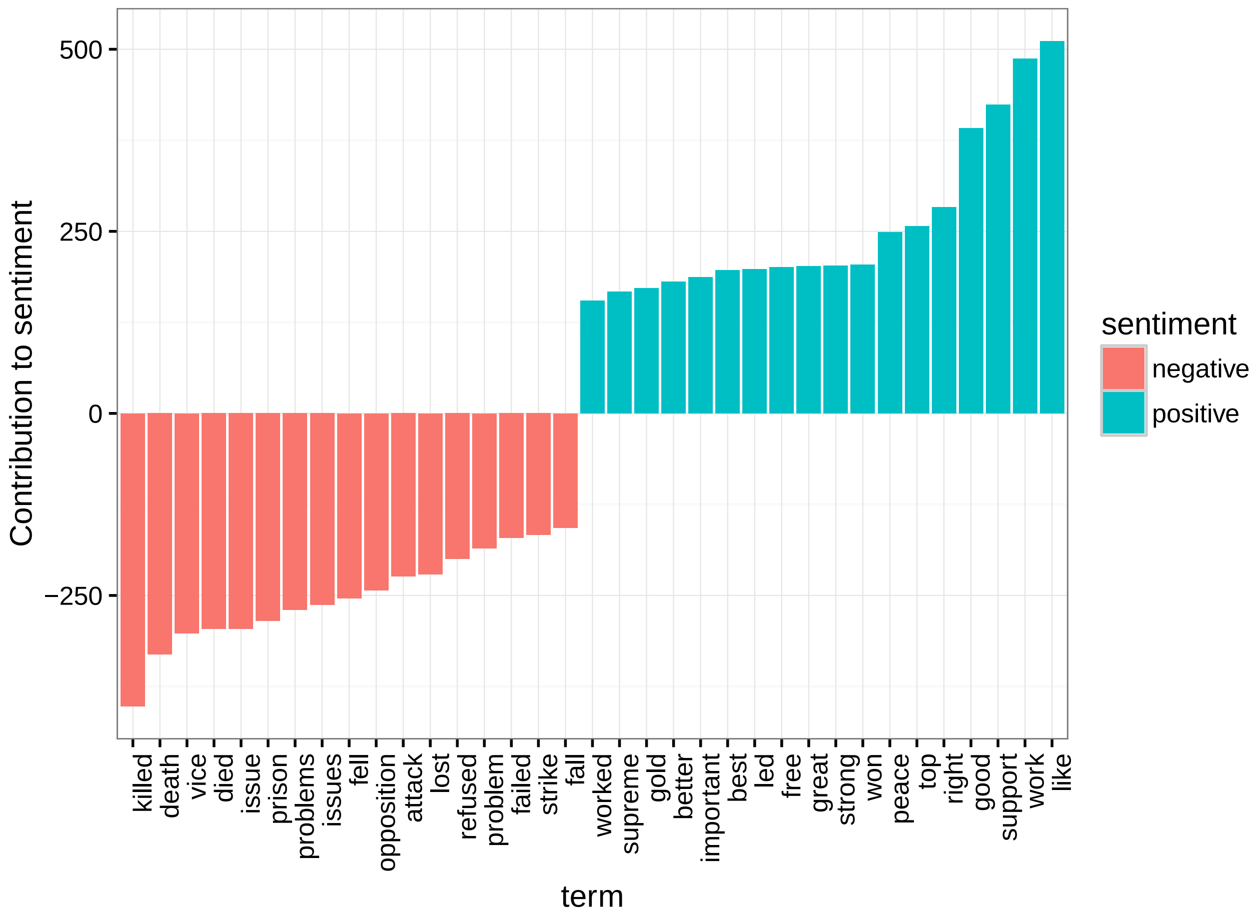
We can find the most negative documents:

```
library(tidyr)

ap_sentiments %>%
  count(document, sentiment, wt = count) %>%
  ungroup() %>%
  spread(sentiment, n, fill = 0) %>%
  mutate(sentiment = positive - negative) %>%
  arrange(sentiment)
```

```
## # A tibble: 2,190 x 4
##    document negative positive sentiment
##       <int>    <dbl>    <dbl>     <dbl>
## 1      1251       54        6       -48
## 2      1380       53        5       -48
## 3       531       51        9       -42
## 4        43       45       11       -34
## 5      1263       44       10       -34
## 6      2178       40        6       -34
## 7       334       45       12       -33
## 8      1664       38        5       -33
## 9      2147       47       14       -33
## 10      516       38        6       -32
## # ... with 2,180 more rows
```

Or visualize which words contributed to positive and 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") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("Contribution to sentiment")
```



Note that a tidier is also available for the `dfm` class from 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: 9,214 feature types
##    ... created a 57 x 9215 sparse dfm
##    ... complete.
## Elapsed time: 0.131 seconds.
```

```
d
```

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

```
tidy(d)
```

```
## # A tibble: 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
## # ... with 43,709 more rows
```

5.2 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.

```
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
```

```
# cast into a Document-Term Matrix
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)
```

```
# 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.
```

```
# 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.

5.3 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
```

```
## # A tibble: 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
## # ... with 10 more variables: topics <chr>, lewissplit <chr>, cgisplit <chr>, oldid <chr>,
## #   topics_cat <list>, places <list>, people <chr>, orgs <chr>, exchanges <chr>, text <chr>
```

Similarly, you can `tidy` a `corpus` object from the quanteda package:

```r
library(quanteda)

data("inaugCorpus")

inaugCorpus
```

```
## Corpus consisting of 57 documents and 3 docvars.
```

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

```
## # A tibble: 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 distingu
## 10  In compliance with an usage coeval with the existence of our Federal Constitution, and sanctioned by t
## # ... with 47 more rows, and 3 more variables: 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
```

```
## # A tibble: 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
## # ... with 49,611 more rows
```

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

```
inaug_freq <- inaug_words %>%
  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
```

```
## # A tibble: 490,200 x 5
##     Year        word     n year_total     percent
##    <int>       <chr> <dbl>      <dbl>       <dbl>
## 1   1789           1     0        529 0.000000000
## 2   1789       1,000     0        529 0.000000000
## 3   1789         100     0        529 0.000000000
## 4   1789 100,000,000     0        529 0.000000000
## 5   1789 120,000,000     0        529 0.000000000
## 6   1789         125     0        529 0.000000000
## 7   1789          13     0        529 0.000000000
## 8   1789        14th     1        529 0.001890359
## 9   1789        15th     0        529 0.000000000
## 10  1789          16     0        529 0.000000000
## # ... with 490,190 more rows
```

For example, we can use the broom package to perform logistic regression on each word.

```
models <- inaug_freq %>%
  group_by(word) %>%
  filter(sum(n) > 50) %>%
  do(tidy(glm(cbind(n, year_total - n) ~ Year, .,
              family = "binomial"))) %>%
```

```
  ungroup() %>%
  filter(term == "Year")

models
```

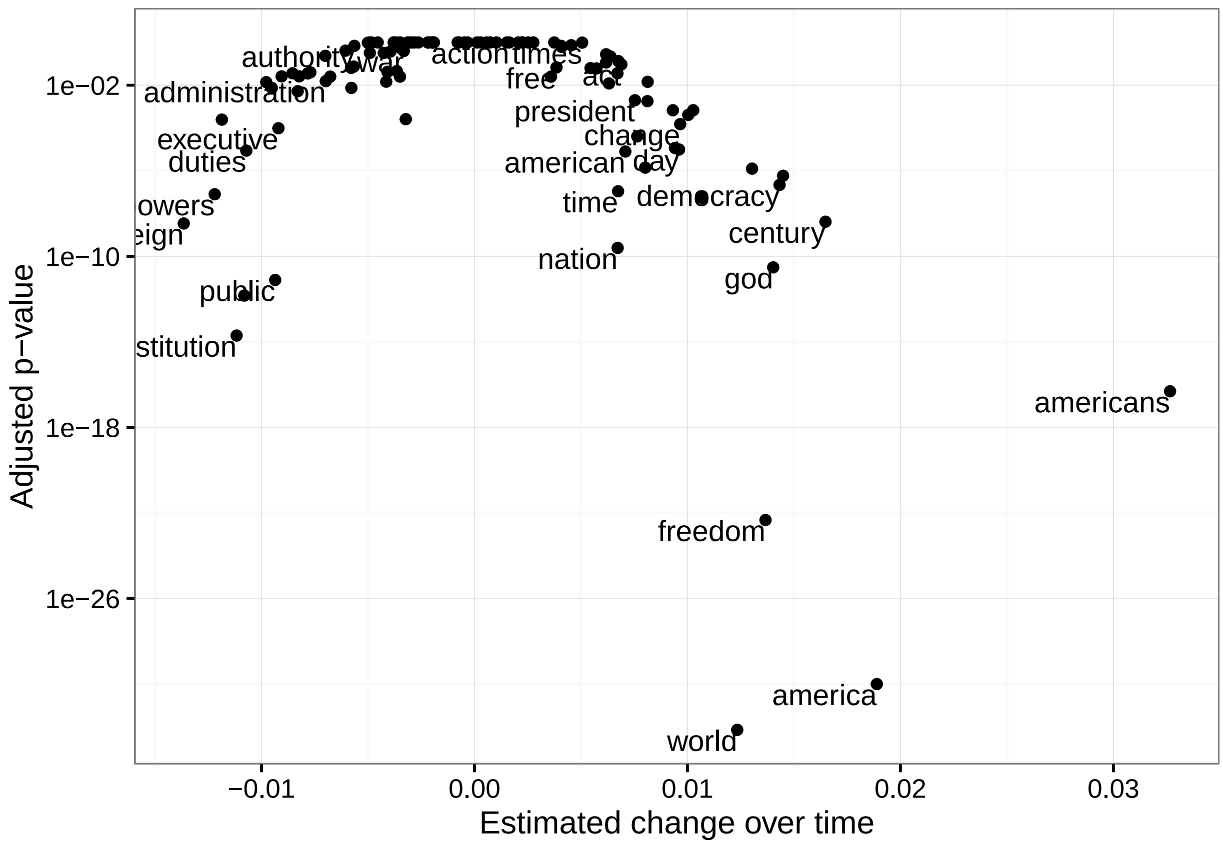
```
## # A tibble: 113 x 6
##               word  term     estimate   std.error  statistic      p.value
##              <chr> <chr>        <dbl>       <dbl>      <dbl>        <dbl>
## 1              act  Year  0.006894234 0.002191596  3.1457591 1.656564e-03
## 2           action  Year  0.001634417 0.001959204  0.8342250 4.041542e-01
## 3  administration  Year -0.006979577 0.001882474 -3.7076616 2.091819e-04
## 4          america  Year  0.018890081 0.001584306 11.9232506 8.954525e-33
## 5         american  Year  0.007084142 0.001321897  5.3590709 8.365105e-08
## 6        americans  Year  0.032657656 0.003659114  8.9250184 4.456252e-19
## 7        authority  Year -0.005640373 0.002336159 -2.4143787 1.576207e-02
## 8         business  Year  0.003745929 0.002016455  1.8576801 6.321445e-02
## 9           called  Year -0.001935068 0.002088388 -0.9265844 3.541423e-01
## 10         century  Year  0.016480566 0.002495844  6.6032027 4.023687e-11
## # ... with 103 more rows
```

```
models %>%
  filter(term == "Year") %>%
  arrange(desc(abs(estimate)))
```

```
## # A tibble: 113 x 6
##          word  term    estimate   std.error statistic      p.value
##         <chr> <chr>       <dbl>       <dbl>     <dbl>        <dbl>
## 1  americans  Year  0.03265766 0.003659114  8.925018 4.456252e-19
## 2    america  Year  0.01889008 0.001584306 11.923251 8.954525e-33
## 3    century  Year  0.01648057 0.002495844  6.603203 4.023687e-11
## 4       live  Year  0.01448914 0.002490610  5.817506 5.973212e-09
## 5  democracy  Year  0.01432438 0.002394738  5.981606 2.209489e-09
## 6        god  Year  0.01402582 0.001921362  7.299935 2.879058e-13
## 7    freedom  Year  0.01366336 0.001320242 10.349129 4.223092e-25
## 8    foreign  Year -0.01364998 0.002058045 -6.632497 3.300543e-11
## 9      earth  Year  0.01303351 0.002291996  5.686532 1.296449e-08
## 10     world  Year  0.01233715 0.001000739 12.328042 6.398240e-35
## # ... with 103 more rows
```

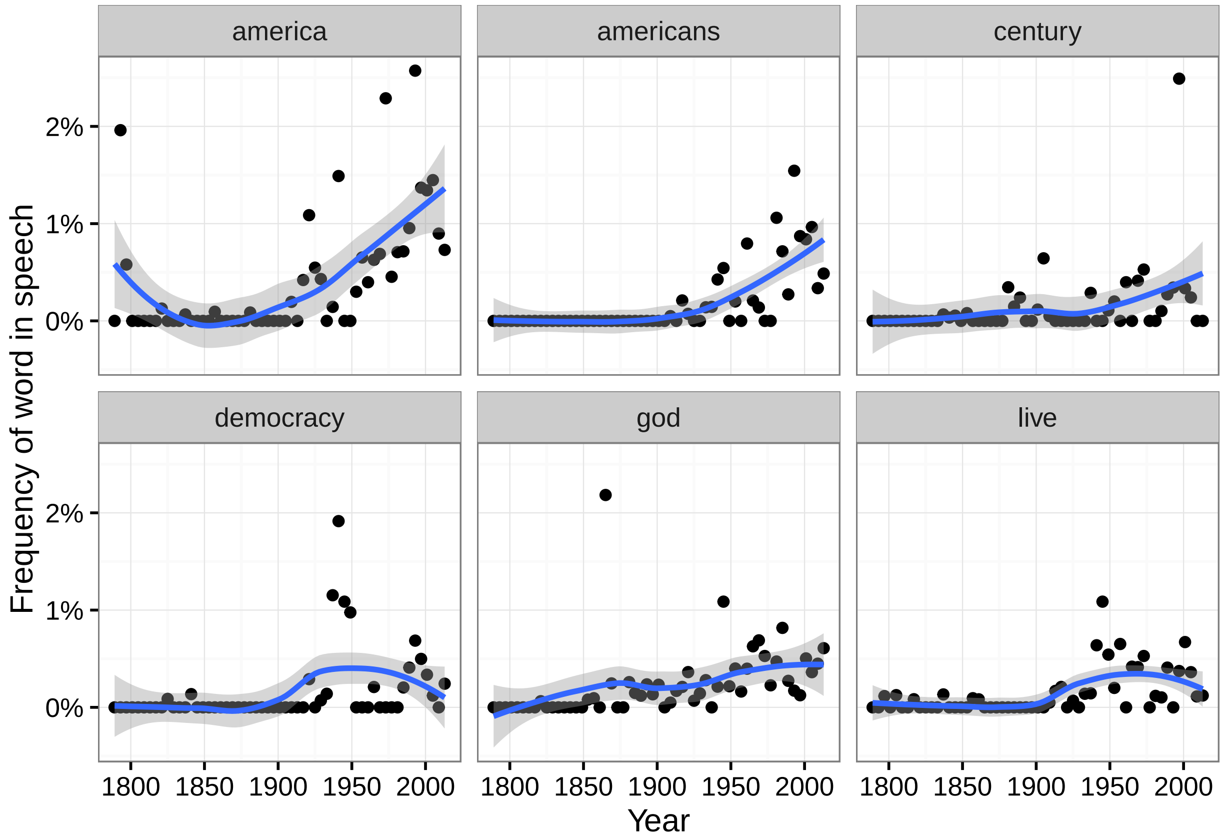
You can show these models as a volcano plot, which compares the effect size with the significance:

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

models %>%
  mutate(adjusted.p.value = p.adjust(p.value)) %>%
  ggplot(aes(estimate, adjusted.p.value)) +
  geom_point() +
  scale_y_log10() +
  geom_text(aes(label = word), vjust = 1, hjust = 1,
            check_overlap = TRUE) +
  xlab("Estimated change over time") +
  ylab("Adjusted p-value")
```



We can also use the ggplot2 package to display the top 6 terms that have changed in frequency over time.

```r
library(scales)

models %>%
  top_n(6, abs(estimate)) %>%
  inner_join(inaug_freq) %>%
  ggplot(aes(Year, percent)) +
  geom_point() +
  geom_smooth() +
  facet_wrap(~ word) +
  scale_y_continuous(labels = percent_format()) +
  ylab("Frequency of word in speech")
```


america
americans
century
democracy
god
live
2%
1%
0%
Frequency of word in speech
1800 1850 1900 1950 2000
Year

Chapter 6

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 the main vignette, to approach topic modeling using consistent and effective tools. In particular, we'll be using tidying functions for LDA objects from the topicmodels package.

6.1 Can we tell the difference between Dickens, Wells, Verne, and Austen?

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?

6.2 Setup

We'll retrieve 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
```

```
## # A tibble: 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
## # ... with 51,653 more rows
```

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`.

```
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
```

```
## # A tibble: 104,721 x 3
##           title_chapter   word     n
##                   <chr>  <chr> <int>
## 1  Great Expectations_1    sir    13
## 2  Great Expectations_1 church     7
## 3  Great Expectations_1 looked     7
## 4  Great Expectations_1    pip     7
## 5  Great Expectations_1  river     6
## 6  Great Expectations_1 tilted     6
## 7  Great Expectations_1  black     5
## 8  Great Expectations_1   head     5
## 9  Great Expectations_1   live     5
## 10 Great Expectations_1 mother     5
## # ... with 104,711 more rows
```

6.3 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
```

```
## # A tibble: 72,860 x 3
##    topic   term         beta
##    <int>  <chr>        <dbl>
## 1      1    sir 2.084216e-03
## 2      2    sir 4.985743e-03
## 3      3    sir 2.483177e-03
## 4      4    sir 2.935244e-04
## 5      1 church 2.936533e-04
## 6      2 church 7.641174e-05
## 7      3 church 7.494668e-04
## 8      4 church 7.008334e-04
## 9      1 looked 1.950223e-03
## 10     2 looked 1.533431e-03
## # ... with 72,850 more rows
```

Notice that this has turned the model into a one-topic-per-term-per-row format. For each combination the model 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:

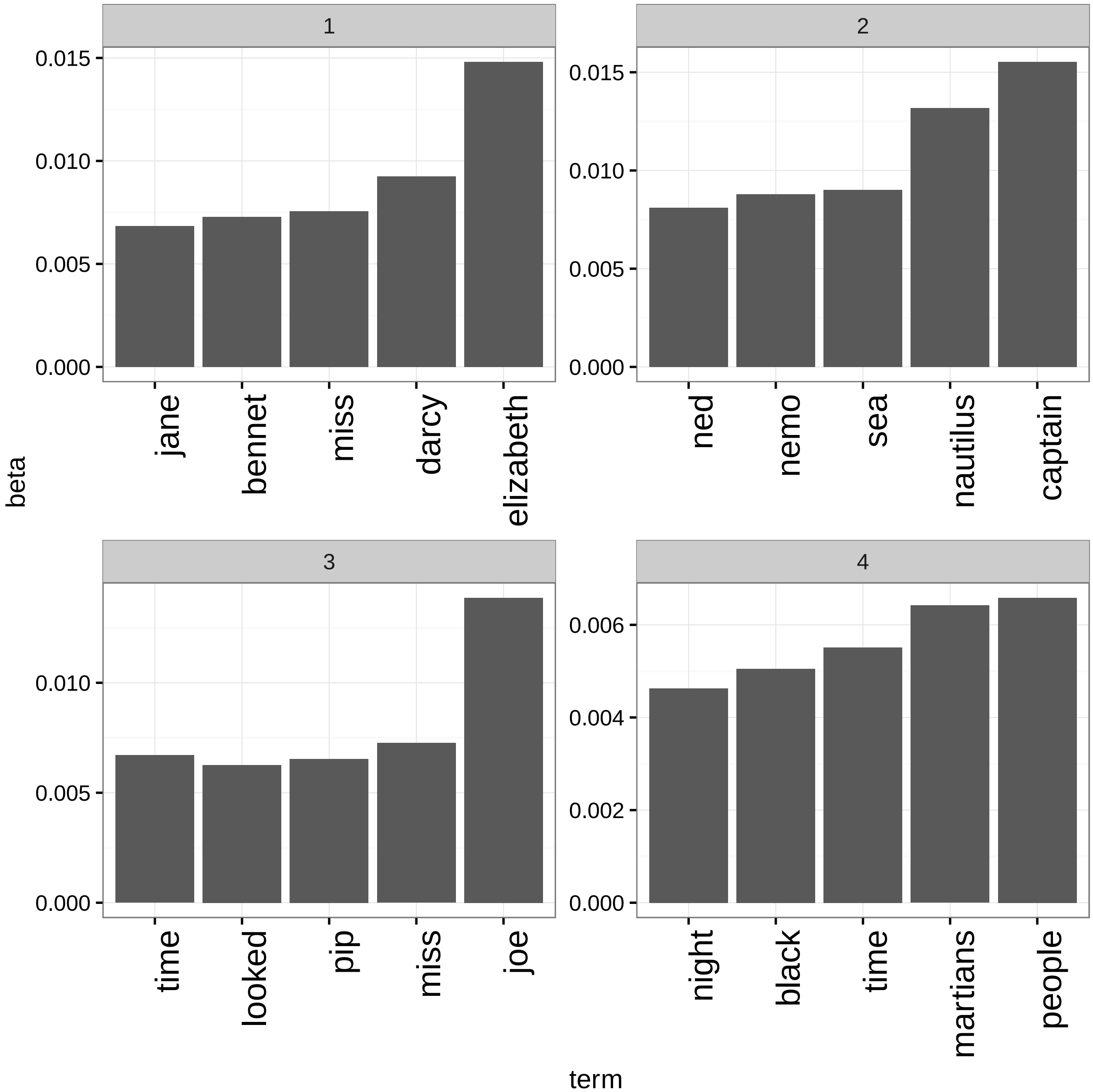
```
top_terms <- chapters_lda_td %>%
  group_by(topic) %>%
  top_n(5, beta) %>%
  ungroup() %>%
  arrange(topic, -beta)
```

```r
top_terms
```

```
## # A tibble: 20 x 3
##    topic      term        beta
##    <int>     <chr>       <dbl>
## 1      1 elizabeth 0.014806162
## 2      1     darcy 0.009250751
## 3      1      miss 0.007560066
## 4      1    bennet 0.007291477
## 5      1      jane 0.006845060
## 6      2   captain 0.015526574
## 7      2  nautilus 0.013176023
## 8      2       sea 0.009007086
## 9      2      nemo 0.008792461
## 10     2       ned 0.008108322
## 11     3       joe 0.013866397
## 12     3      miss 0.007262017
## 13     3      time 0.006709286
## 14     3       pip 0.006541889
## 15     3    looked 0.006262112
## 16     4    people 0.006584429
## 17     4  martians 0.006419187
## 18     4      time 0.005508969
## 19     4     black 0.005051646
## 20     4     night 0.004629963
```

This model lends itself to a visualization:

```r
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*.

6.4 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
```

```
## # A tibble: 772 x 3
```

```
##                  document topic        gamma
##                     <chr> <int>        <dbl>
## 1   Great Expectations_1     1 3.012893e-05
## 2  Great Expectations_10     1 2.145733e-05
## 3  Great Expectations_11     1 1.064330e-05
## 4  Great Expectations_12     1 2.477705e-05
## 5  Great Expectations_13     1 1.954208e-05
## 6  Great Expectations_14     1 7.234765e-05
## 7  Great Expectations_15     1 1.327335e-05
## 8  Great Expectations_16     1 2.951203e-05
## 9  Great Expectations_17     1 1.954208e-05
## 10 Great Expectations_18     1 1.174153e-05
## # ... with 762 more rows
```

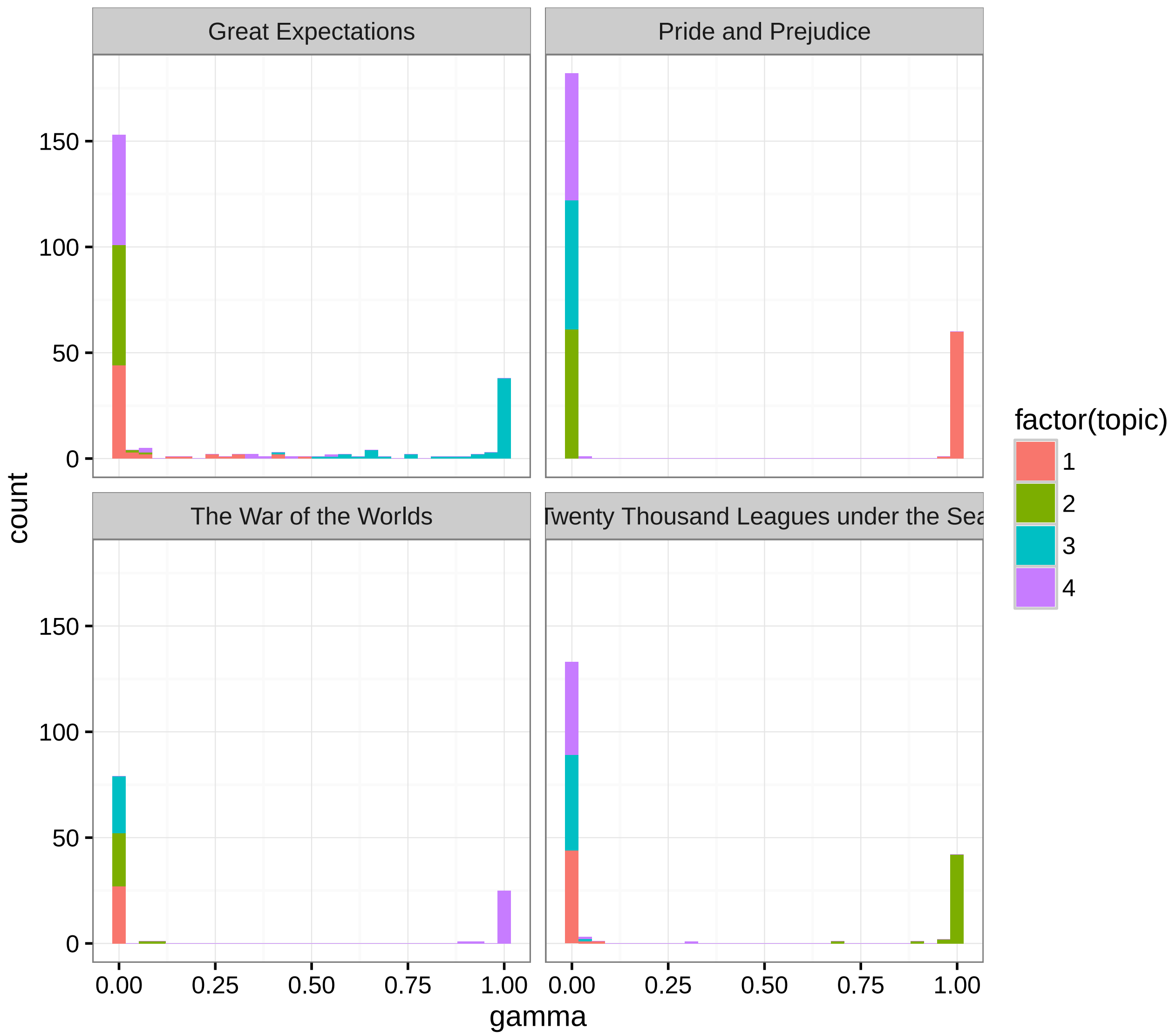
Setting `matrix = "gamma"` returns a tidied version with one-document-per-topic-per-row. Now that we have these document classifiations, we can see how well our unsupervised learning did at distinguishing the four books. First we re-separate the document name into title and chapter:

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

```
## # A tibble: 772 x 4
##                  title chapter topic        gamma
## *                <chr>   <int> <int>        <dbl>
## 1  Great Expectations       1     1 3.012893e-05
## 2  Great Expectations      10     1 2.145733e-05
## 3  Great Expectations      11     1 1.064330e-05
## 4  Great Expectations      12     1 2.477705e-05
## 5  Great Expectations      13     1 1.954208e-05
## 6  Great Expectations      14     1 7.234765e-05
## 7  Great Expectations      15     1 1.327335e-05
## 8  Great Expectations      16     1 2.951203e-05
## 9  Great Expectations      17     1 1.954208e-05
## 10 Great Expectations      18     1 1.174153e-05
## # ... with 762 more rows
```

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
```

```
## # A tibble: 193 x 4
##                  title chapter topic     gamma
##                  <chr>   <int> <int>     <dbl>
## 1  Great Expectations      23     3 0.5187562
## 2  Great Expectations      54     4 0.5522191
## 3  Great Expectations      56     3 0.5523846
## 4  Great Expectations      37     3 0.5724660
## 5  Great Expectations      55     3 0.5813702
```

```
## 6  Great Expectations      46     3 0.6067932
## 7  Great Expectations      25     3 0.6405668
## 8  Great Expectations      21     3 0.6443639
## 9  Great Expectations      20     3 0.6457449
## 10 Great Expectations      53     3 0.6636836
## # ... with 183 more rows
```

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
```

```
## # A tibble: 4 x 2
##                                consensus topic
##                                    <chr> <int>
## 1                     Great Expectations     3
## 2                    Pride and Prejudice     1
## 3                  The War of the Worlds     4
## 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 [5 x 3]
## Groups: title [?]
##
##                                    title                             consensus     n
##                                    (chr)                                 (chr) (int)
## 1                     Great Expectations                    Great Expectations    58
## 2                     Great Expectations                 The War of the Worlds     1
## 3                    Pride and Prejudice                   Pride and Prejudice    61
## 4                  The War of the Worlds                 The War of the Worlds    27
## 5 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!

6.4.1 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.

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

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

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

assignments
```

```
## # A tibble: 104,721 x 6
##                 title chapter  term count .topic         consensus
##                 <chr>   <int> <chr> <dbl>  <dbl>             <chr>
## 1  Great Expectations       1   sir    13      3 Great Expectations
## 2  Great Expectations      10   sir     1      3 Great Expectations
## 3  Great Expectations      11   sir     2      3 Great Expectations
## 4  Great Expectations      18   sir     3      3 Great Expectations
## 5  Great Expectations      19   sir    10      3 Great Expectations
## 6  Great Expectations      20   sir     7      3 Great Expectations
## 7  Great Expectations      23   sir     1      3 Great Expectations
## 8  Great Expectations      25   sir     6      3 Great Expectations
## 9  Great Expectations      26   sir     2      3 Great Expectations
## 10 Great Expectations      27   sir    18      3 Great Expectations
## # ... with 104,711 more rows
```

We can, for example, create a “confusion matrix” using dplyr’s `count` and tidyr’s `spread`:

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

```
## Source: local data frame [4 x 5]
## Groups: title [4]
##
##                                   title Great Expectations Pride and Prejudice
##                                   (chr)              (dbl)               (dbl)
## 1                    Great Expectations              51043                2146
## 2                   Pride and Prejudice                  0               37241
## 3                 The War of the Worlds                  0                   0
## 4 Twenty Thousand Leagues under the Sea                  3                  31
##   The War of the Worlds Twenty Thousand Leagues under the Sea
##                   (dbl)                                 (dbl)
## 1                  2341                                    38
## 2                     1                                     0
## 3                 22492                                    76
## 4                   303                                 39297
```

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?

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

wrong_words
```

```
## # A tibble: 3,701 x 6
##                                   title chapter  term count .topic            consensus
##                                   <chr>   <int> <chr> <dbl>  <dbl>                <chr>
## 1                    Great Expectations      46 river     5      4 The War of the Worlds
```

```
## 2                    Great Expectations      54  river    12      4 The War of the Worlds
## 3                    Great Expectations      20  black     3      4 The War of the Worlds
## 4                    Great Expectations      46  black     2      4 The War of the Worlds
## 5                    Great Expectations      53  black     1      4 The War of the Worlds
## 6                    Great Expectations      54  black     2      4 The War of the Worlds
## 7  Twenty Thousand Leagues under the Sea      18  black     1      4 The War of the Worlds
## 8                    Great Expectations      23   live     1      1   Pride and Prejudice
## 9                    Great Expectations      54   live     1      4 The War of the Worlds
## 10                   Great Expectations      25 mother     2      1   Pride and Prejudice
## # ... with 3,691 more rows
```

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

```
## # A tibble: 3,027 x 4
##                 title             consensus     term     n
##                 <chr>                 <chr>    <chr> <dbl>
## 1  Great Expectations The War of the Worlds     boat    30
## 2  Great Expectations   Pride and Prejudice skiffins    25
## 3  Great Expectations The War of the Worlds     tide    25
## 4  Great Expectations The War of the Worlds      lay    19
## 5  Great Expectations The War of the Worlds     jack    18
## 6  Great Expectations The War of the Worlds    water    18
## 7  Great Expectations   Pride and Prejudice     lady    17
## 8  Great Expectations The War of the Worlds    river    17
## 9  Great Expectations The War of the Worlds   barley    16
## 10 Great Expectations The War of the Worlds   galley    16
## # ... with 3,017 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 7

Tidying word2vec Models from the glove Package

TODO: still a lot of work to be done on the methods as well as the chapter, may or may not make it in

Chapter 8

Predicting ratings from text in the Yelp food reviews dataset

Intro goes here

8.1 Setup

I've downloaded the `yelp_dataset_challenge_academic_dataset` folder from here.[^termsofuse] First I read and process them.

```
library(readr)
library(dplyr)

# You may have used the built-in readLines before, but read_lines from
# readr is faster for large files

# we're reading only 100,000 in this example
# you can try it with the full dataset too, it's just a little slower!
# in the final version of the book we're probably going to read all, it
# just makes this chapter take a while to compile

infile <- "~/Downloads/yelp_dataset_challenge_academic_dataset/yelp_academic_dataset_review.json"
review_lines <- read_lines(infile, n_max = 100000)
```

```
library(stringr)

# Each line is a JSON object- the fastest way to process is to combine into a
# single JSON string and use jsonlite::fromJSON
reviews_combined <- str_c("[", str_c(review_lines, collapse = ", "), "]")

reviews <- jsonlite::fromJSON(reviews_combined) %>%
  jsonlite::flatten() %>%
  tbl_df()
```

```
reviews
```

```
## # A tibble: 100,000 x 10
##                   user_id              review_id stars       date
##                     <chr>                  <chr> <int>      <chr>
```

```
## 1  PUFPaY9KxDAcGqfsorJp3Q Ya85v4eqdd6k9Od8HbQjyA     4 2012-08-01
## 2  Iu6AxdBYGR4A0wspR9BYHA KPvLNJ21_4wbYNctrOwWdQ     5 2014-02-13
## 3  auESFwWvW42h6alXgFxAXQ fFSoGV46Yxuwbr3fHNuZig     5 2015-10-31
## 4  uK8tzraOp4M5u3uYrqIBXg Di3exaUCFNw1V4kSNW5pgA     5 2013-11-08
## 5  I_47G-R2_egp7ME5u_ltew 0Lua2-PbqEQMjD9r89-asw     3 2014-03-29
## 6  PP_xoMSYlGr2pb67BbqBdA 7N9j5YbBHBW6qguE5DAeyA     1 2014-10-29
## 7  JPPhyFE-UE453zA6K0TVgw mjCJR33jvUNt41iJCxDU_g     4 2014-11-28
## 8  2d5HeDvZTDUNVog_WuUpSg Ieh3kfZ-5J9pLju4JiQDvQ     5 2014-02-27
## 9  BShxMIUwaJS378xcrz4Nmg PU28OoBSHpZLkYGCmNxlmg     5 2015-06-16
## 10 fhNxoMwwTipzjO8A9LFe8Q XsA6AojkWjOHA4FmuAb8XQ     3 2012-08-19
## # ... with 99,990 more rows, and 6 more variables: text <chr>, type <chr>, business_id <chr>,
## #   votes.funny <int>, votes.useful <int>, votes.cool <int>
```

8.2 Tidy sentiment analysis

Right now, there is one row for each review. To analyze in the tidy text framework, we need to use the `unnest_tokens` function and turn this into one-row-per-term-per-document:

```r
library(tidytext)

review_words <- reviews %>%
  select(review_id, business_id, stars, text) %>%
  unnest_tokens(word, text) %>%
  filter(!word %in% stop_words$word,
         str_detect(word, "^[a-z']+$"))

review_words
```

```
## # A tibble: 3,971,444 x 4
##                 review_id            business_id stars        word
##                     <chr>                  <chr> <int>       <chr>
## 1  Ya85v4eqdd6k9Od8HbQjyA 5UmKMjUEUNdYWqANhGckJw     4      hoagie
## 2  Ya85v4eqdd6k9Od8HbQjyA 5UmKMjUEUNdYWqANhGckJw     4 institution
## 3  Ya85v4eqdd6k9Od8HbQjyA 5UmKMjUEUNdYWqANhGckJw     4     walking
## 4  Ya85v4eqdd6k9Od8HbQjyA 5UmKMjUEUNdYWqANhGckJw     4   throwback
## 5  Ya85v4eqdd6k9Od8HbQjyA 5UmKMjUEUNdYWqANhGckJw     4         ago
## 6  Ya85v4eqdd6k9Od8HbQjyA 5UmKMjUEUNdYWqANhGckJw     4   fashioned
## 7  Ya85v4eqdd6k9Od8HbQjyA 5UmKMjUEUNdYWqANhGckJw     4        menu
## 8  Ya85v4eqdd6k9Od8HbQjyA 5UmKMjUEUNdYWqANhGckJw     4       board
## 9  Ya85v4eqdd6k9Od8HbQjyA 5UmKMjUEUNdYWqANhGckJw     4      booths
## 10 Ya85v4eqdd6k9Od8HbQjyA 5UmKMjUEUNdYWqANhGckJw     4   selection
## # ... with 3,971,434 more rows
```

Notice that there is now one-row-per-term-per-document: the In this cleaning process we've also removed "stopwords" (such as "I", "the", "and", etc), and removing things things that are formatting (e.g. "—-") rather than a word.

Now I'm going to do sentiment analysis on each review. We'll use the AFINN lexicon, which provides a positivity score for each word, from -5 (most negative) to 5 (most positive).

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

AFINN
```

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

Now as described in this post, our sentiment analysis is just an inner-join operation followed by a summary:

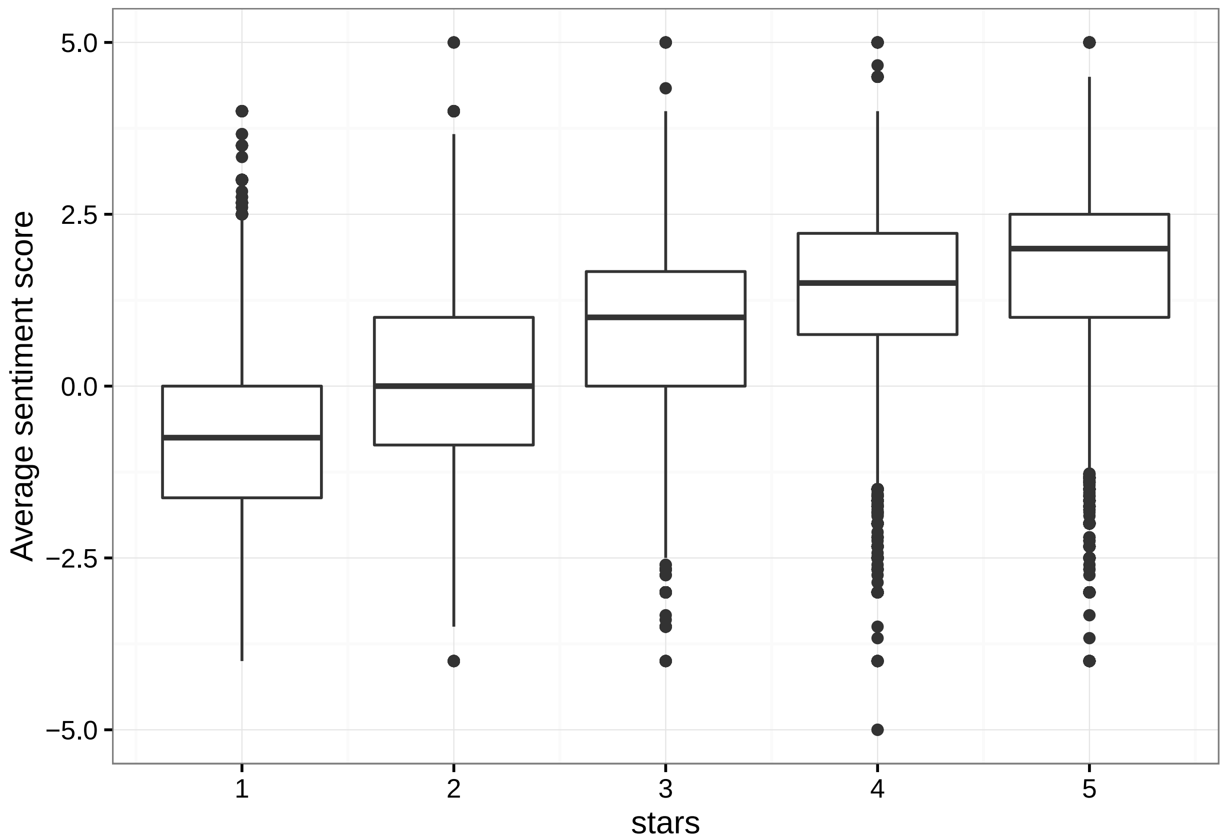
```
reviews_sentiment <- review_words %>%
  inner_join(AFINN, by = "word") %>%
  group_by(review_id, stars) %>%
  summarize(sentiment = mean(afinn_score))

reviews_sentiment
```

```
## Source: local data frame [93,947 x 3]
## Groups: review_id [?]
##
##                 review_id stars  sentiment
##                     (chr) (int)      (dbl)
## 1  __-r0eC3hZlaejvuliC8zQ     5  4.0000000
## 2  __56FUEaW57kZEm56OZk7w     5  0.8333333
## 3  __6tOxx2VcvGR02d2ILkuw     5  1.7500000
## 4  __77nP3Nf1wsGz5HPs2hdw     5  1.6000000
## 5  __B5KInsYxFKIHKXAS6_rA     1 -2.0000000
## 6  __BIQ3tcFZg6_PpdadEfLQ     4  1.6000000
## 7  __DK9Vsmyoo0zJQhIl5cbg     1 -2.1000000
## 8  __ELCJ0wzDM2QNRfVUq26Q     5  3.5000000
## 9  __esH_kgJZeS8k3i6HaG7Q     5  0.2142857
## 10 __GXnNfKFLqFhMtpCTTT2g     3  0.8750000
## ..                    ...   ...        ...
```

Now we can see how our estimates did!

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

```
ggplot(reviews_sentiment, aes(stars, sentiment, group = stars)) +
  geom_boxplot() +
  ylab("Average sentiment score")
```



Well, it's a good start! Our sentiment scores are correlated with positivity ratings. But we do see that there's a large amount of prediction error- some 5-star reviews have a highly negative sentiment score, and vice versa.

8.3 Which words are positive or negative?

We're interested in analyzing the properties of words. Which are suggestive of positive reviews, and which are negative? To do this, we'll create a per-word summary.

```
review_words_counted <- review_words %>%
  count(review_id, business_id, stars, word) %>%
  ungroup()

review_words_counted
```

```
## # A tibble: 3,405,173 x 5
##                 review_id            business_id stars      word     n
##                     <chr>                  <chr> <int>     <chr> <int>
## 1  ___XYEos-RIkPsQwplRYyw YxMnfznT3eYya0YV37tE8w     5    batter     1
## 2  ___XYEos-RIkPsQwplRYyw YxMnfznT3eYya0YV37tE8w     5     chips     3
## 3  ___XYEos-RIkPsQwplRYyw YxMnfznT3eYya0YV37tE8w     5  compares     1
## 4  ___XYEos-RIkPsQwplRYyw YxMnfznT3eYya0YV37tE8w     5 fashioned     1
## 5  ___XYEos-RIkPsQwplRYyw YxMnfznT3eYya0YV37tE8w     5  filleted     1
## 6  ___XYEos-RIkPsQwplRYyw YxMnfznT3eYya0YV37tE8w     5      fish     4
## 7  ___XYEos-RIkPsQwplRYyw YxMnfznT3eYya0YV37tE8w     5     fries     1
## 8  ___XYEos-RIkPsQwplRYyw YxMnfznT3eYya0YV37tE8w     5    frozen     1
## 9  ___XYEos-RIkPsQwplRYyw YxMnfznT3eYya0YV37tE8w     5 greenlake     1
```

```
## 10 ___XYEos-RIkPsQwplRYyw YxMnfznT3eYya0YV37tE8w     5      hand     1
## # ... with 3,405,163 more rows
```

```
word_summaries <- review_words_counted %>%
  group_by(word) %>%
  summarize(reviews = n(),
            uses = sum(n),
            average_stars = mean(stars)) %>%
  ungroup()

word_summaries
```

```
## # A tibble: 73,816 x 4
##          word reviews  uses average_stars
##         <chr>   <int> <int>         <dbl>
## 1   a'boiling       1     1          4.00
## 2      a'fare       1     1          4.00
## 3      a'ight       2     2          1.50
## 4        a'la       2     2          4.50
## 5        a'll       1     1          1.00
## 6      a'lyce       1     2          5.00
## 7      a'more       2     2          5.00
## 8    a'orange       1     1          5.00
## 9   a'prowling      1     1          3.00
## 10         aa      20    23          3.25
## # ... with 73,806 more rows
```

We can start by looking only at words that appear in at least 100 (out of 100000) reviews. This makes sense both because words that appear more rarely will have a noisier measurement (a few good or bad reviews could shift the balance), and because they're less likely to be useful in classifying future reviews or text.

```
word_summaries_filtered <- word_summaries %>%
  filter(reviews >= 100)

word_summaries_filtered
```

```
## # A tibble: 4,465 x 4
##          word reviews  uses average_stars
##         <chr>   <int> <int>         <dbl>
## 1         aaa     100   145      3.780000
## 2     ability     210   215      3.580952
## 3    absolute     589   600      3.755518
## 4  absolutely    3195  3401      3.812520
## 5          ac     306   420      3.058824
## 6      accent     112   115      3.446429
## 7      accept     350   370      3.060000
## 8  acceptable     313   319      2.645367
## 9    accepted     162   167      3.030864
## 10     access     530   588      3.541509
## # ... with 4,455 more rows
```

What were the most positive and negative words?

```
word_summaries_filtered %>%
  arrange(desc(average_stars))
```

```
## # A tibble: 4,465 x 4
##             word reviews  uses average_stars
```

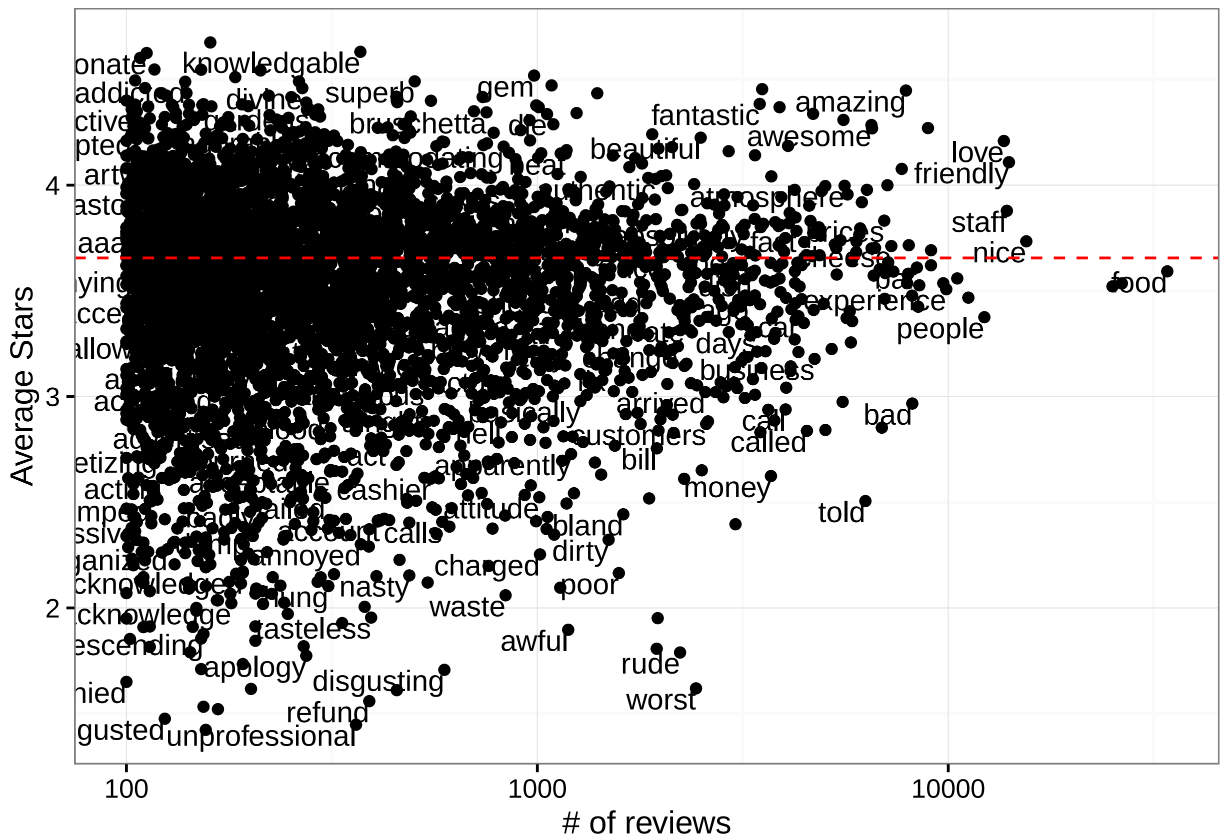
```
##              <chr>   <int> <int>         <dbl>
## 1         exceeded     160   161      4.675000
## 2     knowledgable     371   374      4.630728
## 3    compassionate     112   115      4.625000
## 4        exquisite     108   112      4.601852
## 5          chihuly     117   151      4.547009
## 6         treasure     152   159      4.546053
## 7      compliments     212   215      4.542453
## 8              gem     982   997      4.518330
## 9        botanical     184   241      4.510870
## 10     trustworthy     105   105      4.495238
## # ... with 4,455 more rows
```

```
word_summaries_filtered %>%
  arrange(average_stars)
```

```
## # A tibble: 4,465 x 4
##               word reviews  uses average_stars
##              <chr>   <int> <int>         <dbl>
## 1      incompetent     156   167      1.423077
## 2   unprofessional     362   383      1.447514
## 3        disgusted     124   126      1.475806
## 4           rudely     167   179      1.520958
## 5             lied     154   177      1.532468
## 6           refund     390   497      1.558974
## 7          refused     455   507      1.610989
## 8     unacceptable     201   203      1.616915
## 9            worst    2433  2653      1.619400
## 10          denied     100   111      1.650000
## # ... with 4,455 more rows
```

Makes a lot of sense! We can also plot positivity by frequency:

```
ggplot(word_summaries_filtered, aes(reviews, average_stars)) +
  geom_point() +
  geom_text(aes(label = word), check_overlap = TRUE, vjust = 1, hjust = 1) +
  scale_x_log10() +
  geom_hline(yintercept = mean(reviews$stars), color = "red", lty = 2) +
  xlab("# of reviews") +
  ylab("Average Stars")
```



Note that some of the most common words (e.g. "food") are pretty neutral. There are some common words that are pretty positive (e.g. "amazing", "awesome") and others that are pretty negative ("bad", "told").

8.4 Comparing to sentiment analysis

When we perform sentiment analysis, we're often comparing to a pre-existing lexicon, one that was developed.

The tidytext package also comes with several tidy sentiment analysis lexicons:

```
sentiments
```

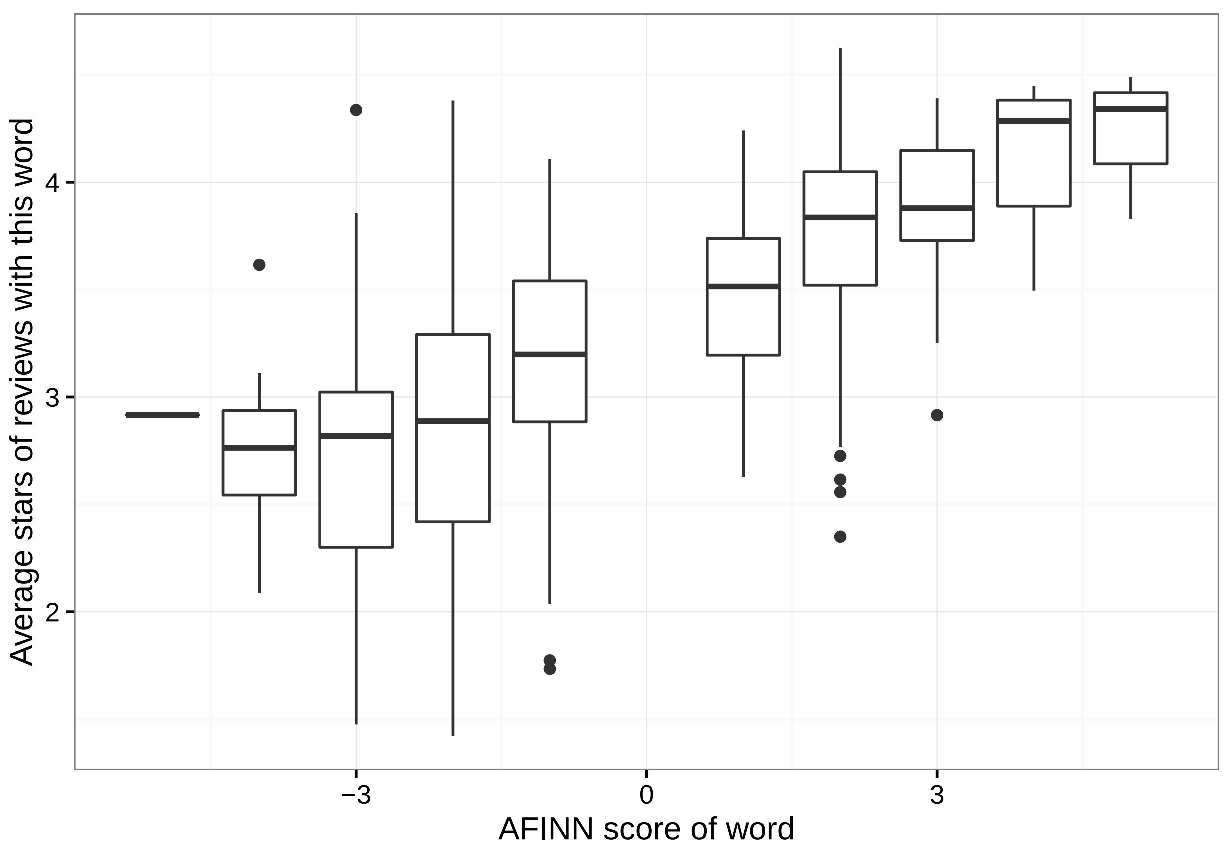
```
## # A tibble: 23,165 x 4
##           word sentiment lexicon score
##          <chr>     <chr>   <chr> <int>
## 1       abacus     trust     nrc    NA
## 2      abandon      fear     nrc    NA
## 3      abandon  negative     nrc    NA
## 4      abandon   sadness     nrc    NA
## 5    abandoned     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
```

We might expect that more positive words are associated with higher star reviews. Does this hold? We can combine and compare the two datasets with `inner_join`.

```
words_afinn <- word_summaries_filtered %>%
  inner_join(AFINN)

words_afinn
```

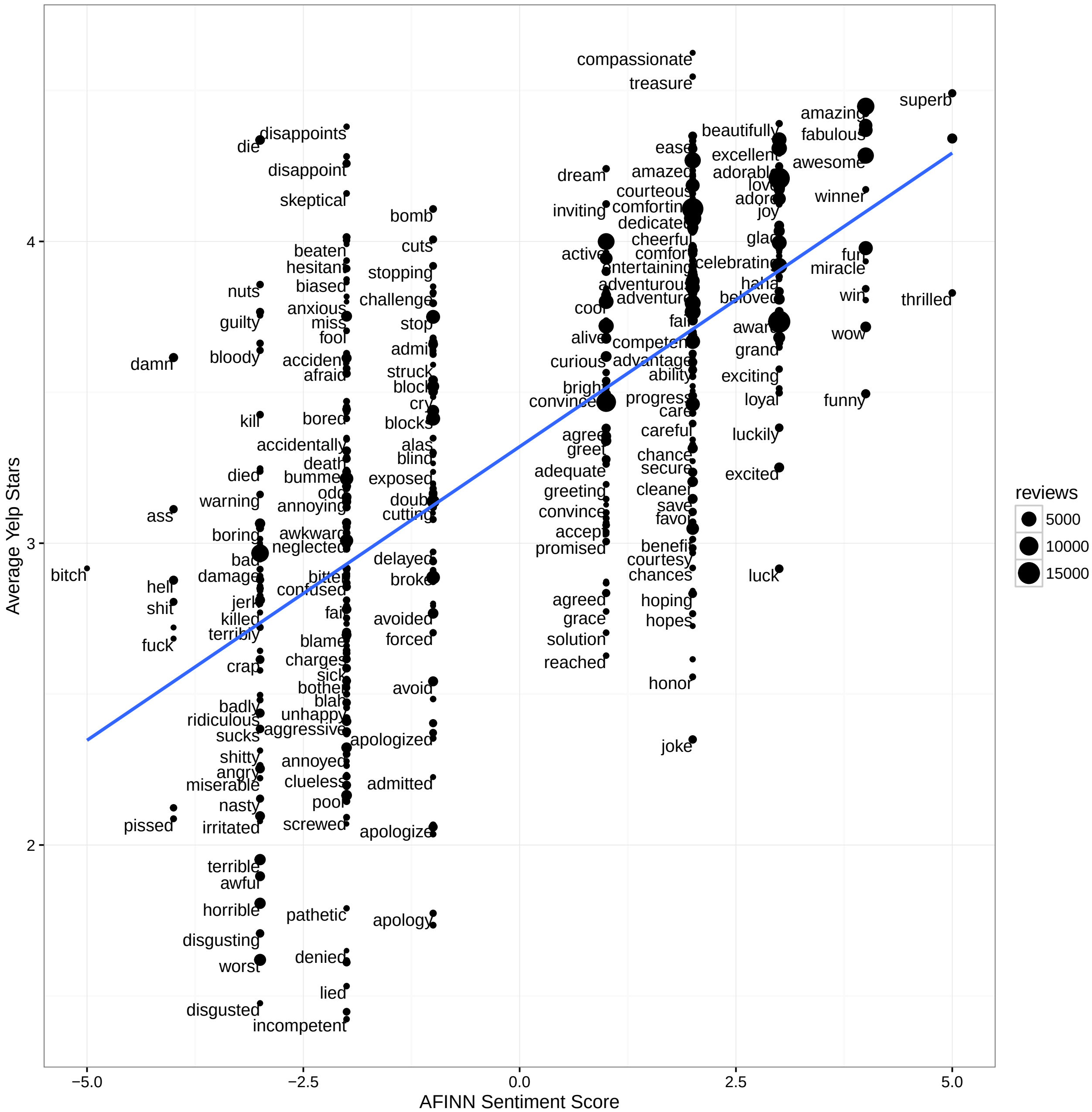
```
## # A tibble: 520 x 5
##             word reviews  uses average_stars afinn_score
##            <chr>   <int> <int>         <dbl>       <int>
## 1        ability     210   215      3.580952           2
## 2         accept     350   370      3.060000           1
## 3       accepted     162   167      3.030864           1
## 4       accident     213   239      3.629108          -2
## 5   accidentally     152   152      3.348684          -2
## 6         active     109   115      3.981651           1
## 7       adequate     290   304      3.262069           1
## 8          admit     740   754      3.666216          -1
## 9       admitted     111   118      2.225225          -1
## 10      adorable     255   266      4.250980           3
## # ... with 510 more rows
```

```
ggplot(words_afinn, aes(afinn_score, average_stars, group = afinn_score)) +
  geom_boxplot() +
  xlab("AFINN score of word") +
  ylab("Average stars of reviews with this word")
```



Just like in our per-review predictions, there's a very clear trend. AFINN sentiment analysis works, at least a little bit!

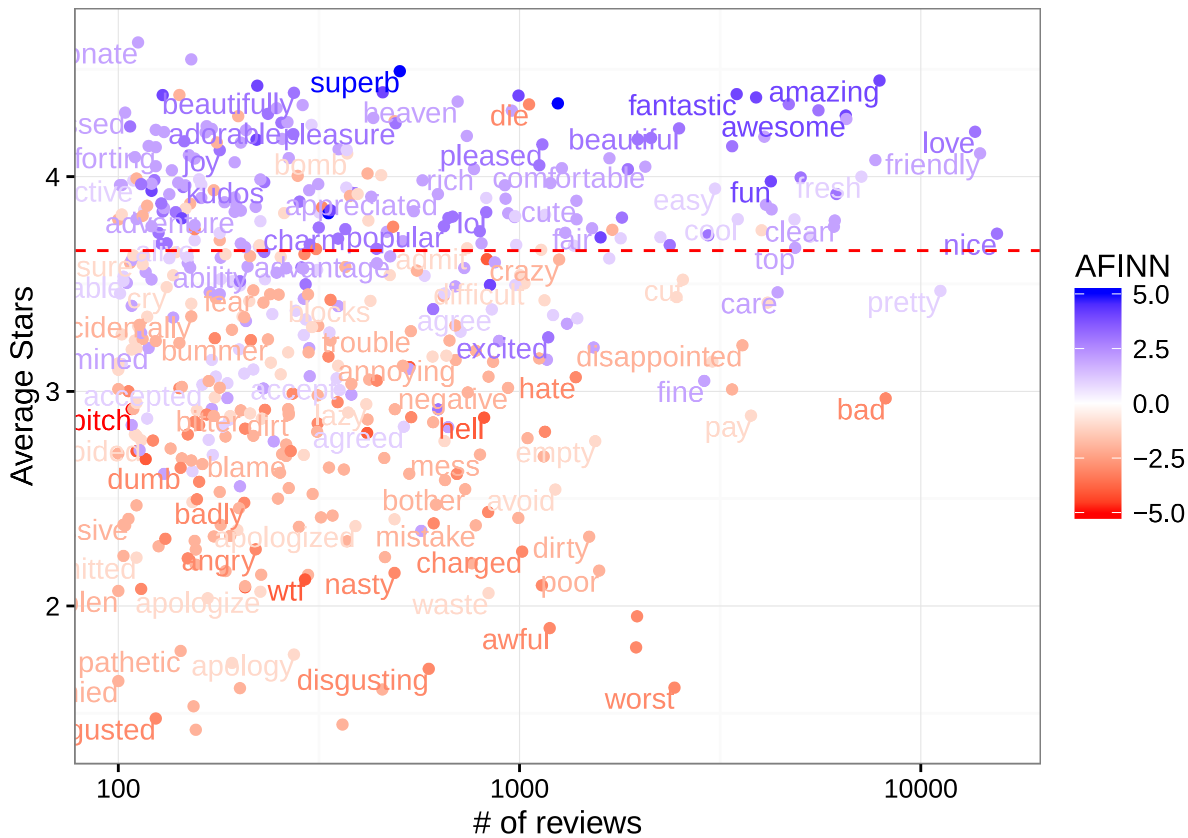
But we may want to see some of those details. Which positive/negative words were most successful in predicting a positive/negative review, and which broke the trend?




```
## mapping: x = x
## geom_blank: na.rm = FALSE
## stat_identity: na.rm = FALSE
## position_identity
```

For example, we can see that most curse words have an AFINN score of -4, and that while some words, like "wtf", successfully predict a negative review, others, like "damn", are often positive. (They're likely part of "damn good", or something similar). Some of the words that AFINN most underestimated included "die" ("the pork chops are to **die** for!"), and one of the words it most overestimated was "joke" ("the service is a complete **joke**!").

One other way we could look at mis


Average Stars
# of reviews
AFINN
5.0
2.5
0.0
−2.5
−5.0
4
3
2
100
1000
10000
superb
beautifully
heaven
die
fantastic
amazing
adorable
pleasure
beautiful
awesome
love
joy
bomb
pleased
friendly
rich
comfortable
fun
fresh
kudos
easy
appreciated
cute
adventure
lol
cool
clear
charm
popular
fair
nice
admit
top
advantage
ability
crazy
cry
fear
difficult
cut
care
pretty
blocks
agree
trouble
excited
bummer
disappointed
annoying
accept
hate
fine
accepted
negative
bad
bitter
dirt
lazy
bitch
hell
pay
agreed
empty
dumb
blame
mess
bother
avoid
badly
apologized
mistake
dirty
angry
charged
wtf
nasty
poor
apologize
waste
awful
pathetic
apology
disgusting
worst

Chapter 9

Some analysis goes here

I don't know what will go here, but I'd like to have one more analysis that touches on all of the areas of tidy text analysis. If we can't find one we'll skip it!