DS3_Lecture_Slides

Bruno Helmeczy

15/05/2021

Sentiment Analysis with R

Let's address the topic of opinion mining or sentiment analysis. When human readers approach a text, we use our understanding of the emotional intent of words to infer whether a section of text is positive or negative, or perhaps characterized by some other more nuanced emotion like surprise or disgust. We can use the tools of text mining to approach the emotional content of text programmatically

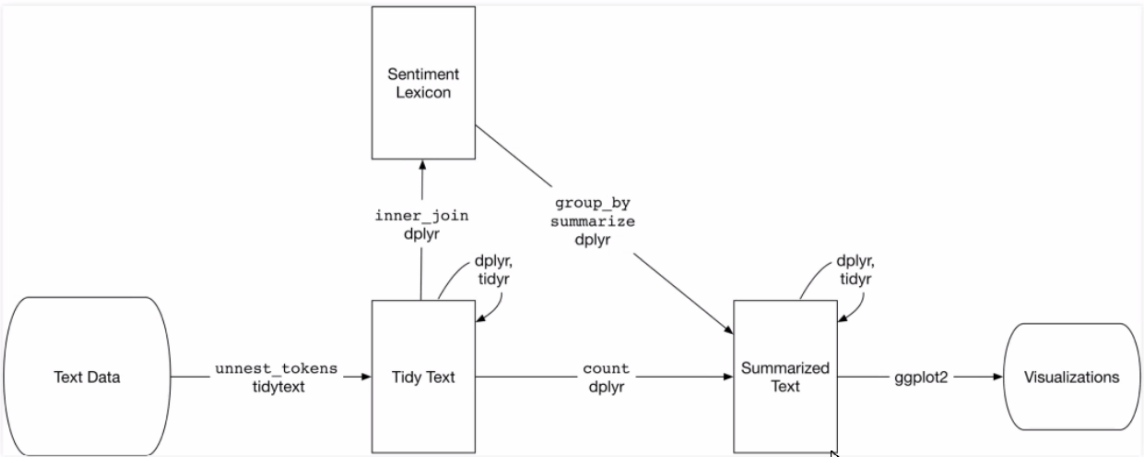
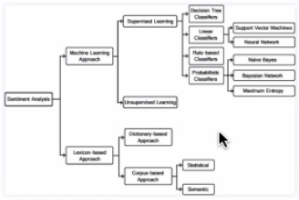


Figure 1: b

Thinking about Sentiment

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

There are other approaches machine learning approaches and lexicon based. Supervised, unsupervised, dictionary and corpus. For the purposes of this course we are applying a simple dictionary based approach. Though the limitations of this approach are well known, it is far and away the most interpretable and simplest.



The sentiments dataset

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

```
library(tidytext)

sentiments
## # A tibble: 27,314 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
```

The three general-purpose lexicons are

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

The Lexicons

All three of these lexicons are based on unigrams, i.e., single words. These lexicons contain many English words and the words are assigned scores for positive/negative sentiment, and also possibly emotions like joy, anger, sadness, and so forth.

All of this information is tabulated in the sentiments dataset, and tidytext provides a function get_sentiments() to get specific sentiment lexicons without the columns that are not used in that lexicon.

```
get_sentiments("nrc")
get_sentiments("bing")
get_sentiments("afinn")
```

There are also some domain-specific sentiment lexicons available, constructed to be used with text from a specific content area. Later in the course we will study an analysis using a sentiment lexicon specifically for finance.

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.

The NRC Lexicon

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.

```
get_sentiments("nrc")

## # A tibble: 13,901 x 2
##           word sentiment
##          <chr>     <chr>
##  1      abacus     trust
##  2     abandon      fear
##  3     abandon  negative
##  4     abandon   sadness
##  5   abandoned     anger
##  6   abandoned      fear
##  7   abandoned  negative
```

The Bing Lexicon

The bing lexicon categorizes words in a binary fashion into positive and negative categories.

```
get_sentiments("bing")

## # A tibble: 6,788 x 2
##           word sentiment
##          <chr>     <chr>
##  1     2-faced  negative
##  2     2-faces  negative
##  3          a+  positive
##  4    abnormal  negative
##  5     abolish  negative
##  6  abominable  negative
##  7  abominably  negative
```

The AFINN Lexicon

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.

```
get_sentiments("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
```

Sentiment Analysis using inner_join

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

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

Sentiment Analysis using inner_join

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

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

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

Sentiment Analysis using inner_join - results

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

Comparing the three sen

With several options for sentiment le
all three sentiment lexicons and exam
choose only the words from the one

```
pride_prejudice <- tidy_b
  filter(book == "Pride &

## # A tibble: 122,204 x
##                 book
##               <fctr>
##  1 Pride & Prejudice
##  2 Pride & Prejudice
##  3 Pride & Prejudice
##  4 Pride & Prejudice
```

Now, we can use inner_join() to calcu

Comparing the three sentiment dictionaries

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

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

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

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

Comparing the three sentiment dictionaries

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

Why The Differences

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

```
get_sentiments("nrc") %>%
     filter(sentiment %in% c("positive",
                             "negative")) %>%
  count(sentiment)
## # A tibble: 2 x 2
##   sentiment     n
##       <chr> <int>
## 1  negative  3324
## 2  positive  2312
get_sentiments("bing") %>%
  count(sentiment)
## # A tibble: 2 x 2
##   sentiment     n
##       <chr> <int>
## 1  negative  4782
## 2  positive  2006
```

Both lexicons have more negative than positive words, but the ratio of negative to positive words is higher in the Bing lexicon than the NRC lexicon.

Most common positive and negative words

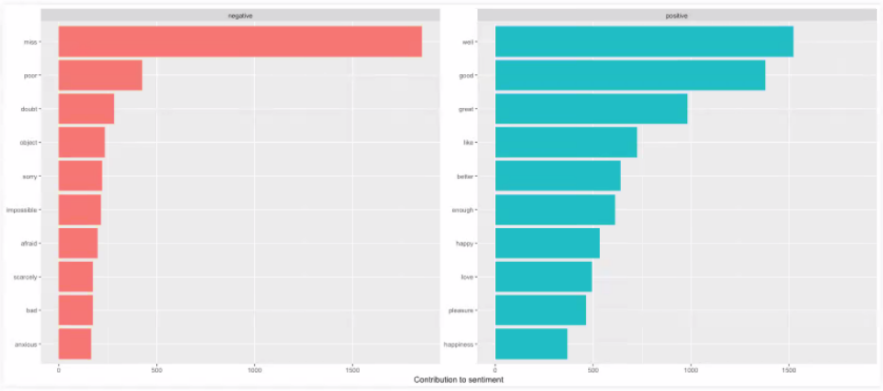
One advantage of having the data frame with both sentiment and word is that we can analyze word counts that contribute to each sentiment. By implementing `count()` here with arguments of both word and sentiment, we find out how much each word contributed to each sentiment.

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

bing_word_counts
## # A tibble: 2,585 x 3
##         word sentiment     n
##        <chr>     <chr> <int>
##  1     miss  negative  1855
##  2     well  positive  1523
##  3     good  positive  1380
##  4    great  positive   981
##  5     like  positive   725
```

Most common positive and negative words

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

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



SPOT THE ERROR

Dealing With Errors

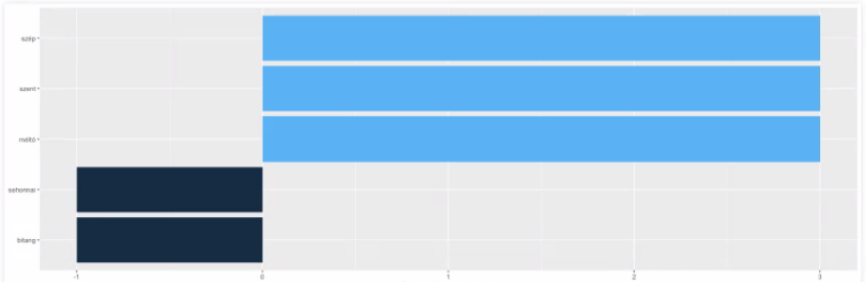
This visualization lets us spot an anomaly in the sentiment analysis; the word "miss" is coded as negative but it is used as a title for young, unmarried women in Jane Austen's works. If it were appropriate for our purposes, we could easily add "miss" to a custom stop-words list using bind_rows(). We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

```
custom_stop_words <- bind_rows(data_frame(word = c("miss"),
                                          lexicon = c("custom")),
                               stop_words)

custom_stop_words
## # A tibble: 1,150 x 2
##           word lexicon
##          <chr>   <chr>
##  1        miss  custom
##  2           a   SMART
##  3         a's   SMART
```

So now we can do a simple analysis...

- Or so I thought. It turns out these dictionaries are ok, but as far as I can tell, hungarian text is:
  - Very seldom positive
  - At best neutral
- Please, actually take this code and prove me wrong!

```
tidy_poem <- poem_df %>%
  unnest_tokens(word, body) %>%
  anti_join(hu_stop_word) %>%
  inner_join(hungarian_sentiment)

tidy_poem %>%
  count(word, sentiment) %>% mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n * sentiment, fill = sentiment)) + geom_col(show.legend = FALSE) +
   labs(y = "Contribution to sentiment", x = NULL) + coord_flip()
```



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. R packages included coreNLP (T. Arnold and Tilton 2016), cleanNLP (T. B. Arnold 2016), and sentimentr (Rinker 2017) are examples of such sentiment analysis algorithms.

```
library(sentimentr)

print(sentimentr::sentiment("I am not having a good day."))
```

```
   element_id sentence_id word_count  sentiment
1:          1           1          7 -0.2834734
```

```
print(sentimentr::sentiment("I am not not having a good day."))
```

```
   element_id sentence_id word_count sentiment
1:          1           1          8  0.265165
```

However don't be fooled, this won't stop the following from failing:

```
print(sentiment("The idiot said that he was having the very best most amazing truly wonderful day of his
life!"))
```

```
   element_id sentence_id word_count sentiment
1:          1           1         18  1.072445
```

Looking at units beyond just words

Perhaps we event want to do chapter by chapter analysis. We could then split all of Jane Austen's books into chapters (you remember that wonderful regular expression from Tueday?)

```
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
```

Looking at units beyond just words

Then, let's find the number of negative words in each chapter and divide by the total words in each chapter. For each book, which chapter has the highest proportion of negative words?

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

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

tidy_books %>%
  semi_join(bingnegative) %>%  group_by(book, chapter) %>%
  summarize(negativewords = n()) %>%
  left_join(wordcounts, by = c("book", "chapter")) %>%
  mutate(ratio = negativewords/words) %>%  filter(chapter != 0) %>%
  top_n(1) %>% ungroup()
## # A tibble: 6 x 5
##                 book chapter negativewords words      ratio
##               <fctr>   <int>         <int> <int>      <dbl>
## 1 Sense & Sensibility      43           161  3405 0.04728341
## 2   Pride & Prejudice      34           111  2104 0.05275665
## 3      Mansfield Park      46           173  3685 0.04694708
## 4                Emma      15           151  3340 0.04520958
## 5    Northanger Abbey      21           149  2982 0.04996647
## 6           Persuasion       4            62  1807 0.03431101
```

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

Analyzing word and document frequency: tf-idf

A central question in text mining and natural language processing is how to quantify what a document is about. Can we do this by looking at the words that make up the document? One measure of how important a word may be is its term frequency (tf), how frequently a word occurs in a document. Another approach is to look at a term's inverse document frequency (idf), which decreases the weight for commonly used words and increases the weight for words that are not used very much in a collection of documents. This can be combined with term frequency to calculate a term's tf-idf (the two quantities multiplied together), the frequency of a term adjusted for how rarely it is used.

Term Frequency

In the case of the term frequency tf(t,d), the simplest choice is to use the raw count of a term in a document, i.e. the number of times that term t occurs in document d. If we denote the raw count by ft,d, then the simplest tf scheme is tf(t,d) = ft,d Other possibilities include:

- Boolean "frequencies": tf(t,d) = 1 if t occurs in d and 0 otherwise;
- term frequency adjusted for document length : ft,d ÷ (number of words in d)
- logarithmically scaled frequency: tf(t,d) = log (1 + ft,d), (or zero if ft,d is zero);[6]
- augmented frequency, to prevent a bias towards longer documents, e.g. raw frequency divided by the raw frequency of the most occurring term in the document:

$$\mathrm{tf}(t,d) = 0.5 + 0.5 \cdot \frac{f_{t,d}}{\max\{f_{t',d} : t' \in d\}}$$

Inverse Document Frequency

The inverse document frequency is a measure of how much information the word provides, that is, whether the term is common or rare across all documents. It is the logarithmically scaled inverse fraction of the documents that contain the word, obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient.

$$\mathrm{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

- N - total number of documents in the corpus
- Denominator is number of documents where the term t appears (i.e., tf(t,d) != 0}). If the term is not in the corpus, this will lead to a division-by-zero, so often it is adjusted with a + 1

We can use tidy data principles, as described earlier, 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.

Term frequency in Jane Austen's novels

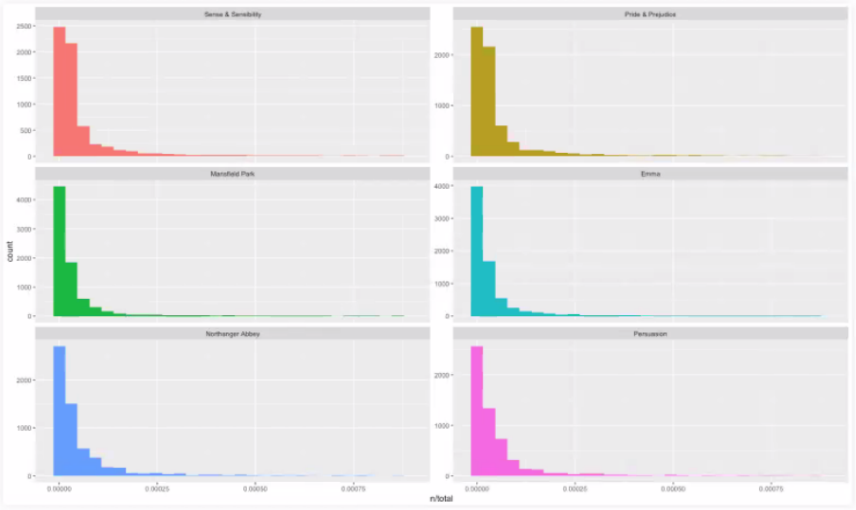
Let's start by looking at the published novels of Jane Austen and examine first term frequency, then tf-idf. We can start just by using dplyr verbs such as group_by() and join(). What are the most commonly used words in Jane Austen's novels? (Let's also calculate the total words in each novel here, for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

```
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
```

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

Visualizing Term Frequency

```
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

Looking at Zipf's Law

Since we have the data frame we used to plot term frequency, we can examine Zipf's law for Jane Austen's novels with just a few lines of dplyr functions.

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

freq_by_rank
## # A tibble: 40,379 x 6
## # Groups:   book [6]
##                book  word     n  total  rank `term frequency`
##              <fctr> <chr> <int>  <int> <int>            <dbl>
##  1 Mansfield Park   the  6206 160460     1       0.03867631
##  2 Mansfield Park    to  5475 160460     2       0.03412065
##  3 Mansfield Park   and  5438 160460     3       0.03389007
##  4             Emma    to  5239 160996     1       0.03254118
##  5             Emma   the  5201 160996     2       0.03230515
##  6             Emma   and  4896 160996     3       0.03041069
```

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 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
```

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.

The bind_tf_idf function

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

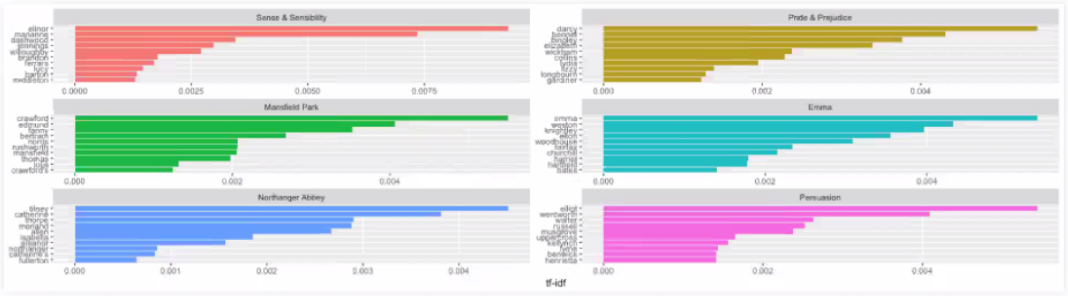
```
book_words %>%
  select(-total) %>%
  arrange(desc(tf_idf))
## # A tibble: 40,379 x 6
##                  book      word     n          tf      idf      tf_idf
##                <fctr>     <chr> <int>       <dbl>    <dbl>       <dbl>
##  1 Sense & Sensibility    elinor   623 0.005193528 1.791759 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 0.003036171 1.791759 0.005440088
##  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 0.002283105 1.791759 0.004090775
## # ... with 40,369 more rows
```

Here we see all proper nouns, names that are in fact important in these novels. None of them occur in all of novels, and they are important, characteristic words for each text within the corpus of Jane Austen's novels.

Visualizing High TF-IDF Words

So, let's look at a visualization for these high tf-idf words!

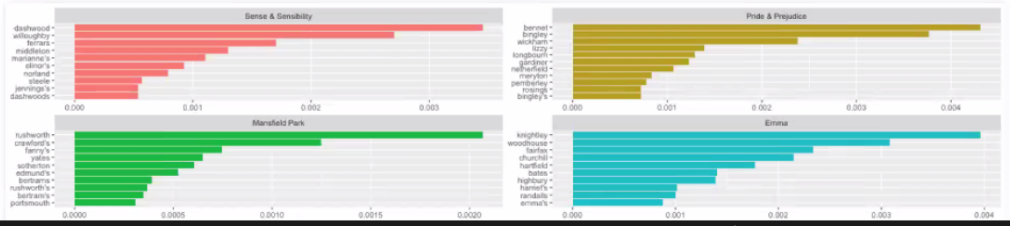
```
book_words %>%
  arrange(desc(tf_idf)) %>%
  mutate(word = factor(word, levels = rev(unique(word)))) %>%
  group_by(book) %>%
  top_n(10) %>%
  ungroup %>%
  ggplot(aes(word, tf_idf, fill = book)) +
  geom_col(show.legend = FALSE) +
  labs(x = NULL, y = "tf-idf") +
  facet_wrap(~book, ncol = 2, scales = "free") +
  coord_flip()
```



Removing Names

What happens if we remove all the names we can? (Using Hadley's Baby Names dataset)...

```
names <- read_csv("data/baby-names.csv") %>% select(word=name) %>%
mutate(word=tolower(word))
book_words %>%
  anti_join(names) %>%
  arrange(desc(tf_idf)) %>%
  mutate(word = factor(word, levels = rev(unique(word)))) %>%
  group_by(book) %>%
  top_n(10) %>%
  ungroup %>%
  ggplot(aes(word, tf_idf, fill = book)) +
  geom_col(show.legend = FALSE) +
  labs(x = NULL, y = "tf-idf") +
  facet_wrap(~book, ncol = 2, scales = "free") +
  coord_flip()
```

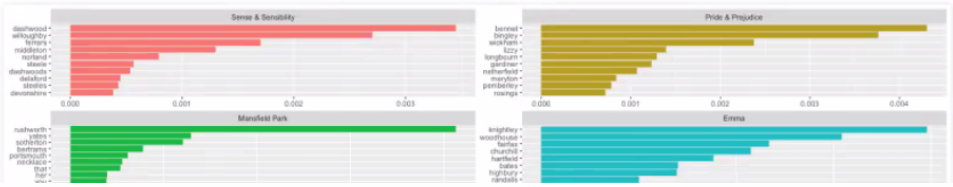


Removing Possessives

Well that wasn't enough, let's now remove all the posessives (apostrophe words)...

What happens if we remove all the names we can? (Using Hadley's Baby Names dataset)...

```
names <- read_csv("data/baby-names.csv") %>% select(word=name) %>% mutate(word=tolower(word))
book_words %>%
  mutate(word = str_extract(word, "[a-z']+")) %>%
  anti_join(names) %>%
  filter(! str_detect(word, "'")) %>%
  arrange(desc(tf_idf)) %>%
  mutate(word = factor(word, levels = rev(unique(word)))) %>%
  group_by(book) %>%
  top_n(10) %>%
  ungroup %>%
  ggplot(aes(word, tf_idf, fill = book)) +
  geom_col(show.legend = FALSE) +
  labs(x = NULL, y = "tf-idf") +
  facet_wrap(~book, ncol = 2, scales = "free") +
  coord_flip()
```



A Bit of Failure

It is OK to fail here. We have learned something rather interesting actually... books of fiction, at least these books, are about the characters and the places!

- The algorithm is shockingly robust
- The algorithm really does find what makes a text unique
- This is worth celebrating, and we should be happy about it.

We have learned a lot.

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.

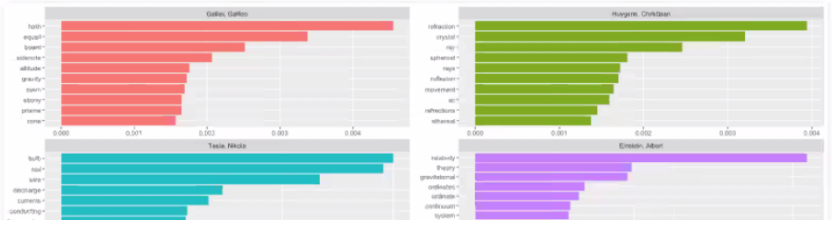
```
library(gutenbergr)
physics <- gutenberg_download(c(37729, 14725, 13476, 5001),
                              meta_fields = "author")
physics_words <- physics %>%
  unnest_tokens(word, text) %>%
  count(author, word, sort = TRUE) %>%
  ungroup()

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
```

Why don't we anti_join with stop_words?!

Visualizing Physics Texts

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

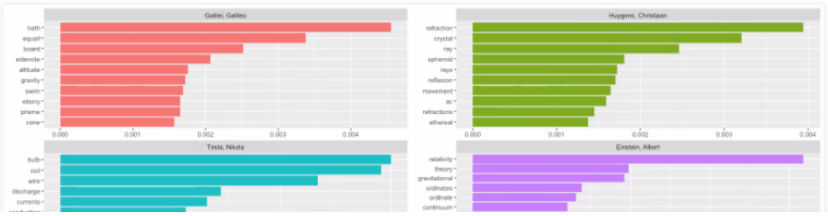
```
plot_physics <- physics_words %>%
  mutate(word = str_extract(word, "[a-z']+")) %>%
  bind_tf_idf(word, author, n) %>%
  arrange(desc(tf_idf)) %>%
  mutate(word = factor(word, levels = rev(unique(word)))) %>%
  mutate(author = factor(author, levels = c("Galilei, Galileo", "Huygens, Christiaan",
                                            "Tesla, Nikola", "Einstein, Albert")))

plot_physics %>%
  group_by(author) %>%  top_n(10, tf_idf) %>%
  ungroup() %>%  mutate(word = reorder(word, tf_idf)) %>%
  ggplot(aes(word, tf_idf, fill = author)) +
  geom_col(show.legend = FALSE) + labs(x = NULL, y = "tf-idf") +
  facet_wrap(~author, ncol = 2, scales = "free") + coord_flip()
```



Visualizing Physics Texts

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

```
plot_physics <- physics_words %>%
  mutate(word = str_extract(word, "[a-z']+")) %>%
  bind_tf_idf(word, author, n) %>%
  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")))

plot_physics %>%
  group_by(author) %>%  top_n(10, tf_idf) %>%
  ungroup() %>%  mutate(word = reorder(word, tf_idf)) %>%
  ggplot(aes(word, tf_idf, fill = author)) +
  geom_col(show.legend = FALSE) + labs(x = NULL, y = "tf-idf") +
  facet_wrap(~author, ncol = 2, scales = "free") + coord_flip()
```



Tokenizing by n-gram

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

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

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

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

austen_bigrams
## # A tibble: 725,049 x 2
##                   book          bigram
##                 <fctr>           <chr>
##  1 Sense & Sensibility       sense and
##  2 Sense & Sensibility and sensibility
##  3 Sense & Sensibility  sensibility by
##  4 Sense & Sensibility         by jane
##  5 Sense & Sensibility     jane austen
##  6 Sense & Sensibility     austen 1811
```

Counting and filtering n-grams

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

```
austen_bigrams %>%
  count(bigram, sort = TRUE)
## # A tibble: 211,236 x 2
##      bigram     n
##       <chr> <int>
##  1   of the  3017
##  2    to be  2787
##  3   in the  2368
##  4   it was  1781
##  5     i am  1545
```

As one might expect, a lot of the most common bigrams are pairs of common (uninteresting) words, such as of the and to be: what we call "stop-words".

Counting and filtering n-grams

This is a useful time to use tidyr's separate(), which splits a column into multiple based on a delimiter. This lets us separate it into two columns, "word1" and "word2", at which point we can remove cases where either is a stop-word.

```
library(tidyr)

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

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

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

bigram_counts
## # A tibble: 33,421 x 3
##      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
```

Analyzing bigrams

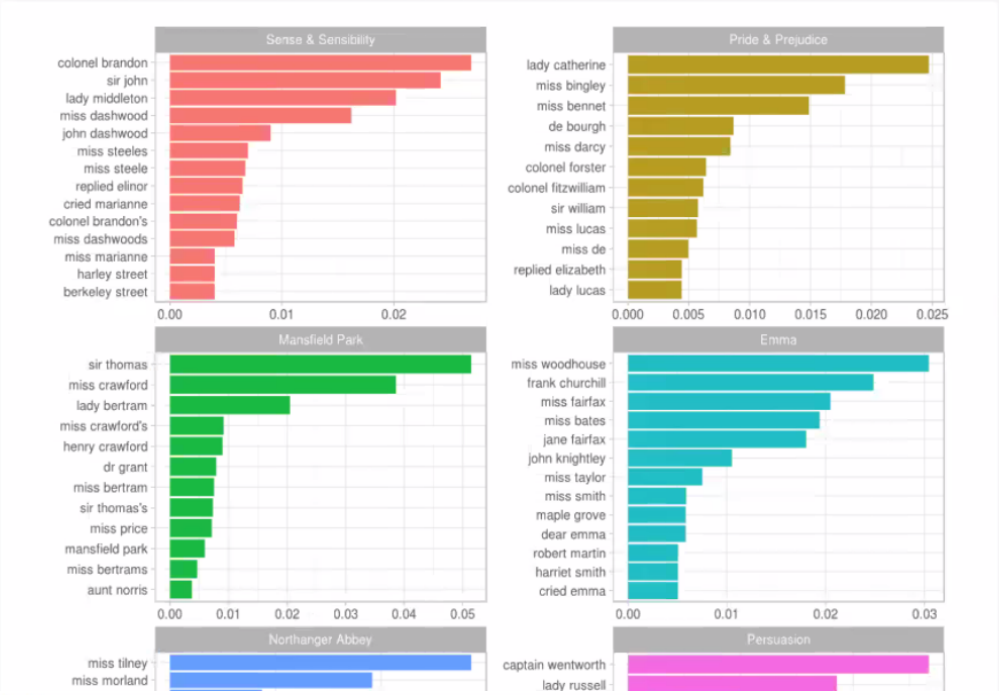
This one-bigram-per-row format is helpful for exploratory analyses of the text. As a simple example, we might be interested in the most common "streets" mentioned in each book:

```
bigrams_filtered %>%
  filter(word2 == "street") %>%
  count(book, word1, sort = TRUE)
## # A tibble: 34 x 3
##                  book       word1     n
##                <fctr>       <chr> <int>
##  1 Sense & Sensibility    berkeley    16
##  2 Sense & Sensibility      harley    16
##  3    Northanger Abbey    pulteney    14
##  4    Northanger Abbey      milsom    11
##  5      Mansfield Park     wimpole    10
##  6   Pride & Prejudice gracechurch     9
##  7 Sense & Sensibility     conduit     6
```

A bigram can also be treated as a term in a document in the same way that we treated individual words. For example, we can look at the tf-idf (Chapter 3) of bigrams across Austen novels. These tf-idf values can be visualized within each book, just as we did for words (Figure 4.1).

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

bigram_tf_idf
## # A tibble: 36,217 x 6
##                  book            bigram     n         tf      idf     tf_idf
##                <fctr>             <chr> <int>      <dbl>    <dbl>      <dbl>
##  1          Persuasion captain wentworth   170 0.02985599 1.791759 0.05349475
##  2      Mansfield Park        sir thomas   287 0.02873160 1.791759 0.05148012
##  3      Mansfield Park     miss crawford   215 0.02152368 1.791759 0.03856525
```

Visualizing Bigrams



Tradeoffs with Bigrams

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

Using bigrams to provide context in sentiment analysis

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

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

```
bigrams_separated %>%
  filter(word1 == "not") %>%
  count(word1, word2, sort = TRUE)
## # A tibble: 1,246 x 3
##    word1 word2     n
##    <chr> <chr> <int>
##  1   not    be   610
##  2   not    to   355
##  3   not  have   327
##  4   not  know   252
##  5   not     a   189
...
```

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

Using bigrams to provide context in sentiment analysis

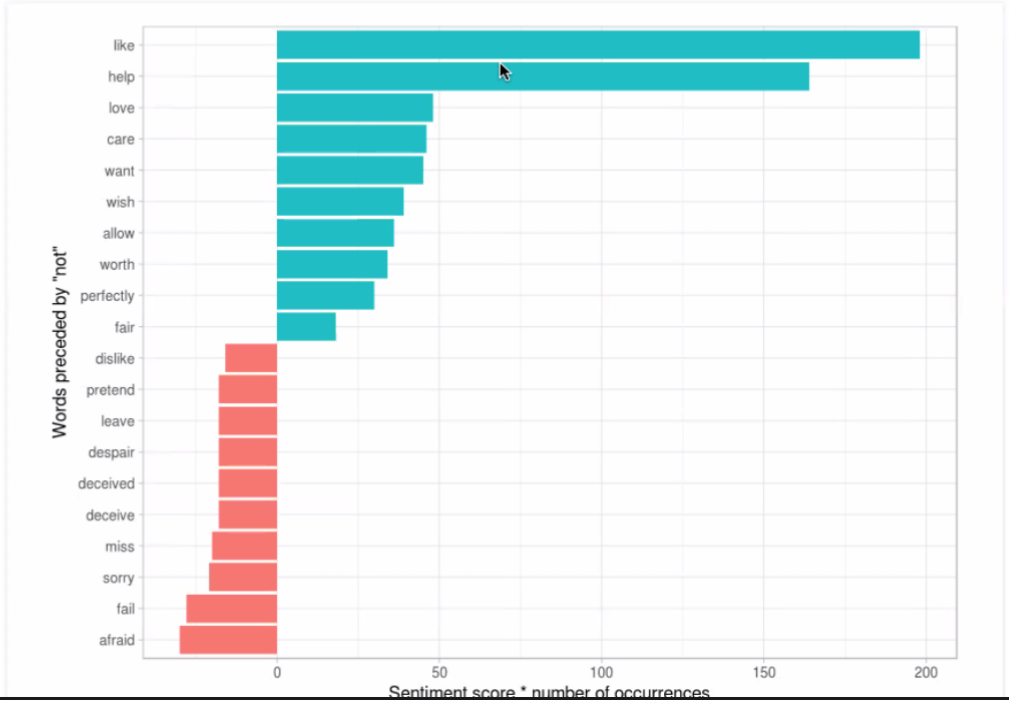
We can then examine the most frequent words that were preceded by "not" and were associated with a sentiment.

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

not_words
## # A tibble: 245 x 3
##       word2 score     n
##       <chr> <int> <int>
##  1    like     2    99
##  2    help     2    82
##  3    want     1    45
##  4    wish     1    39
```

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

```
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_col(show.legend = FALSE) +
  xlab("Words preceded by \"not\"") +
  ylab("Sentiment score * number of occurrences") +
  coord_flip()
```

Misidentified Sentiment



Visualizing a network of bigrams with ggraph

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

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

The igraph package has many powerful functions for manipulating and analyzing networks. One way to create an igraph object from tidy data is the graph_from_data_frame() function, which takes a data frame of edges with columns for "from", "to", and edge attributes (in this case n):

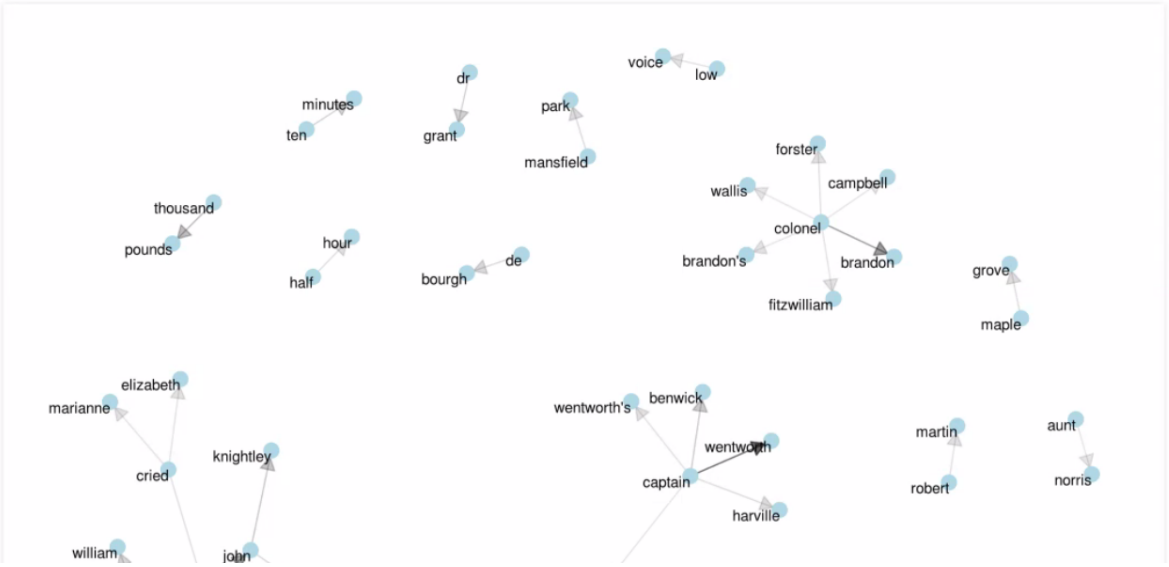
```
library(igraph)

# original counts
bigram_counts
## # A tibble: 33,421 x 3
##      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

bigram_graph <- bigram_counts %>%
  select(from=word1, to=word2)
  filter(n > 20) %>%
  graph_from_data_frame()

bigram_graph
## IGRAPH 21eb65f DN-- 91 77 --
```

Visualizing the Graph

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

```
library(ggraph)

set.seed(2016)

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

ggraph(bigram_graph, layout = "fr") +
  geom_edge_link(aes(edge_alpha = n), show.legend = FALSE,
                 arrow = a, end_cap = circle(.07, 'inches')) +
  geom_node_point(color = "lightblue", size = 5) +
  geom_node_text(aes(label = name), vjust = 1, hjust = 1) +
  theme_void()
```

Visualizing the Graph



An Update

Tidytext now supports stopwords in hungarian OUT OF THE BOX! Yes, it's true, Julia Silge went ahead and integrated a new interesting package from CRAN, the stopwords package. If you install tidytext from cran now (or have very recently) you can do the following:

```
library(tidytext)

head(get_stopwords('hu'), 5)
```

```
# A tibble: 5 x 2
  word  lexicon
  <chr> <chr>
1 a     snowball
2 ahogy snowball
3 ahol  snowball
4 aki   snowball
5 akik  snowball
```

It now supports a ton of languages out of the box!

```
library(stopwords)
map(stopwords_getsources(), stopwords_getlanguages) %>% flatten %>% unique %>% str_sort
```

```
 [1] "af" "ar" "bg" "bn" "br" "ca" "cs" "da" "de" "el" "en" "eo" "es" "et"
[15] "eu" "fa" "fi" "fr" "ga" "gl" "ha" "he" "hi" "hr" "hu" "hy" "id" "ir"
[29] "it" "ja" "ko" "ku" "la" "lt" "lv" "mr" "ms" "nl" "no" "pl" "pt" "ro"
[43] "ru" "sk" "sl" "so" "st" "sv" "sw" "th" "tl" "tr" "uk" "ur" "vi" "yo"
[57] "zh" "zu"
```

Example: mining financial articles

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

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

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

Sys.setenv(TZ='Europe/Budapest') #IMPORTANT

company <- c("Microsoft", "Apple", "Google", "Facebook", "Twitter", "Snapchat")
symbol <- c("MSFT", "AAPL", "GOOG", "FB", "TWTR", "SNAP")

download_articles <- function(symbol) {
  WebCorpus(YahooFinanceSource(paste0(symbol)))
}

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

Mining Financial Articles

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

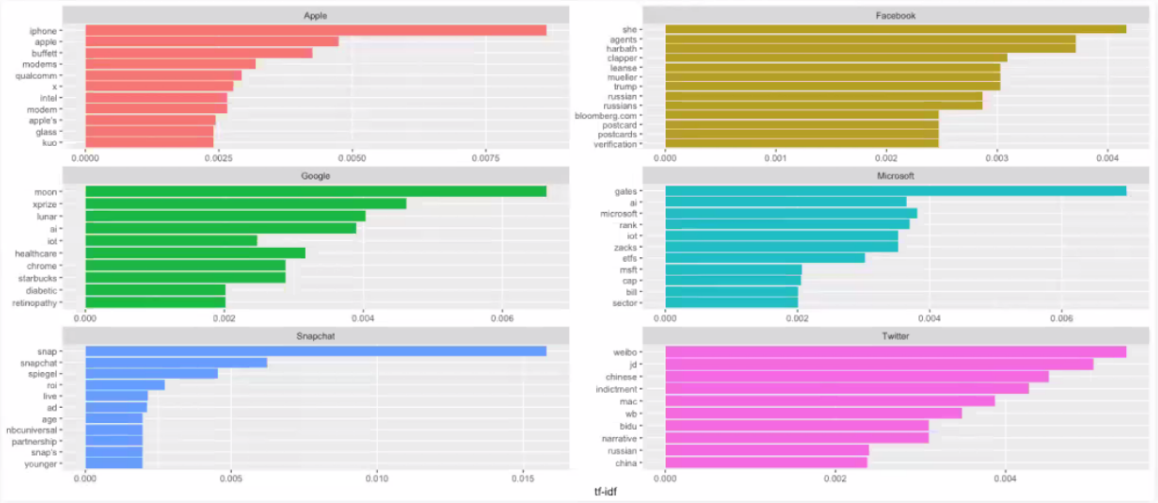
```
stock_tokens <- stock_articles %>%
  unnest(map(corpus, tidy)) %>%
  unnest_tokens(word, text) %>%
  select(company, datetimestamp, word, id, heading)

stock_tokens
## # A tibble: 105,054 x 5
##      company       datetimestamp        word        id
##        <chr>              <dttm>       <chr>       <chr>
##  1 Microsoft 2017-01-17 12:07:24      amazon tag:finance.google.com,cluster
##  2 Microsoft 2017-01-17 12:07:24      passes tag:finance.google.com,cluster
##  3 Microsoft 2017-01-17 12:07:24   microsoft tag:finance.google.com,cluster
##  4 Microsoft 2017-01-17 12:07:24          to tag:finance.google.com,cluster
##  5 Microsoft 2017-01-17 12:07:24      become tag:finance.google.com,cluster
##  6 Microsoft 2017-01-17 12:07:24         the tag:finance.google.com,cluster
```

Mining Financial Articles

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

```
library(stringr)

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

stock_tf_idf %>%
  arrange(desc(tf_idf)) %>%
  mutate(word = factor(word, levels = rev(unique(word)))) %>%
  group_by(company) %>%
  top_n(10) %>%
  ungroup %>%
  ggplot(aes(word, tf_idf, fill = company)) +
  geom_col(show.legend = FALSE) +
  labs(x = NULL, y = "tf-idf") +
  facet_wrap(~company, ncol = 2, scales = "free") +
  coord_flip()
```

Top Company Terms

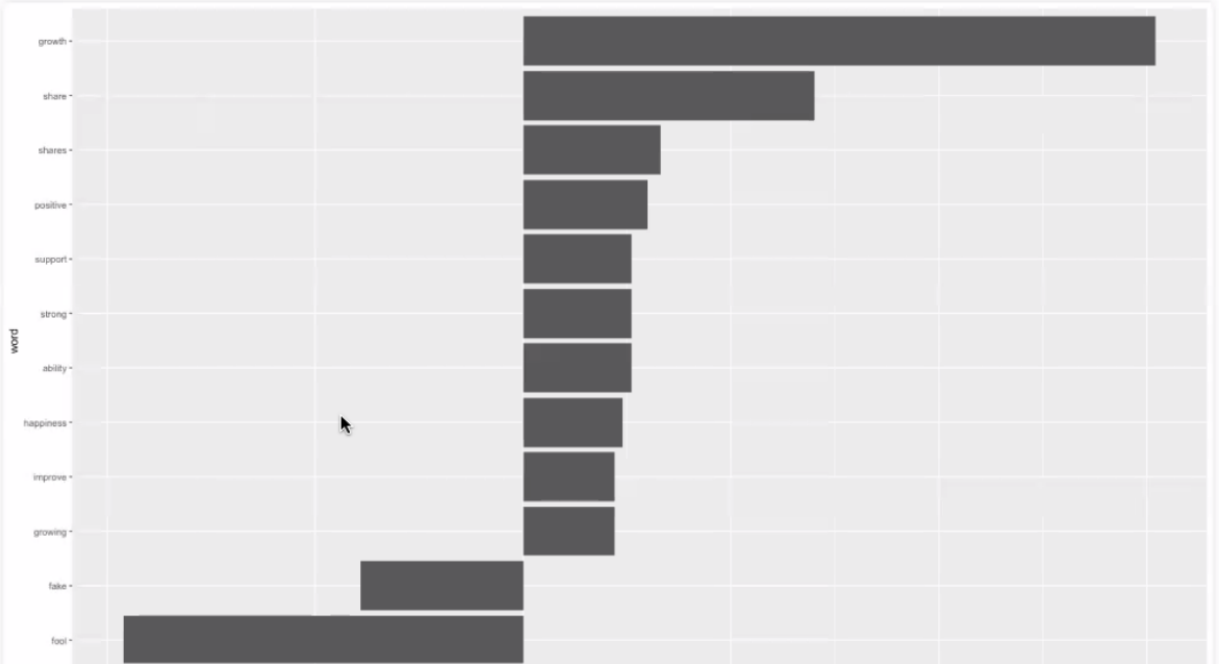


Sentiment Analysis

If we were interested in using recent news to analyze the market and make investment decisions, we'd likely want to use sentiment analysis to determine whether the news coverage was positive or negative. Before we run such an analysis, we should look at what words would contribute the most to positive and negative sentiments. For example, we could examine this within the AFINN lexicon.

```
stock_tokens %>%
  anti_join(stop_words, by = "word") %>%
  count(word, id, sort = TRUE) %>%
  inner_join(get_sentiments("afinn"), by = "word") %>%
  group_by(word) %>%
  summarize(contribution = sum(n * score)) %>%
  top_n(12, abs(contribution)) %>%
  mutate(word = reorder(word, contribution)) %>%
  ggplot(aes(word, contribution)) +
  geom_col() +
  coord_flip() +
  labs(y = "Frequency of word * AFINN score")
```

Sentiment Analysis



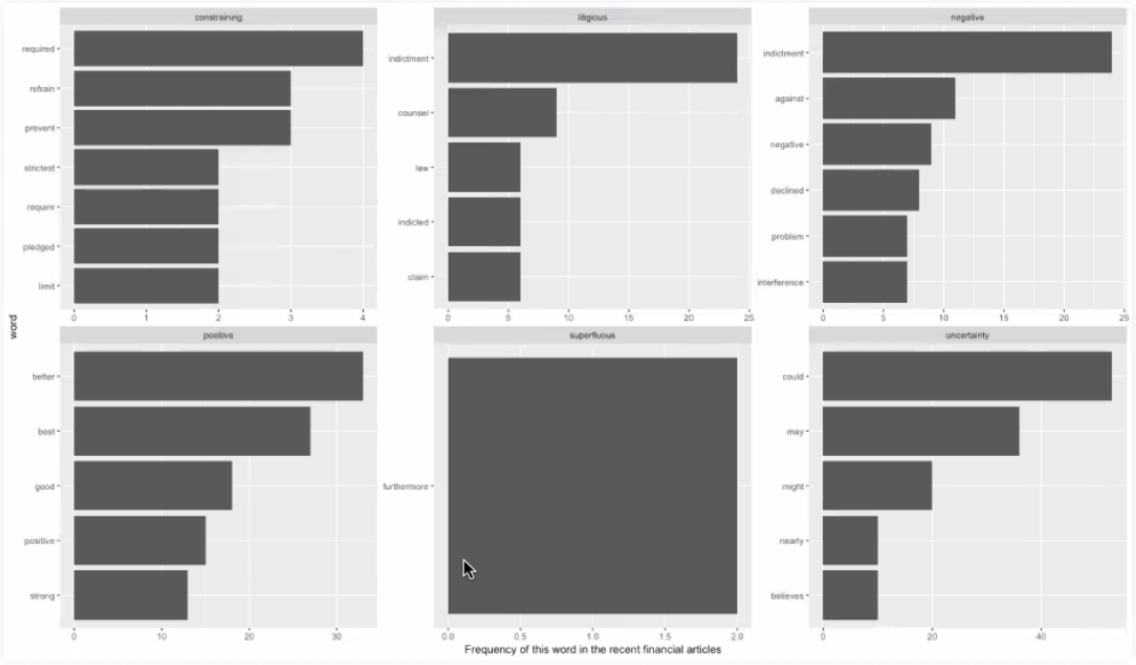
Danger Danger!!!

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

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

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

Visualizing With Loughran and McDonald



Computing on Sentiment

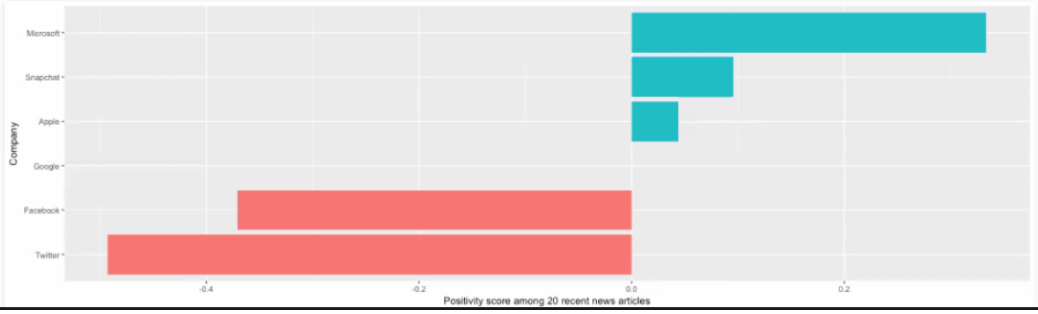
Now that we know we can trust the dictionary to approximate the articles' sentiments, we can use our typical methods for counting the number of uses of each sentiment-associated word in each corpus.

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

stock_sentiment_count
# A tibble: 6 x 7
  company   constraining litigious negative positive superfluous uncertainty
* <chr>            <dbl>     <dbl>    <dbl>    <dbl>       <dbl>       <dbl>
1 Apple             8.00     11.0      97.0     106          2.00        62.0
2 Facebook          9.00     21.0      61.0      28.0        0           21.0
3 Google            6.00     12.0      77.0      77.0        0           60.0
4 Microsoft         2.00      8.00     32.0      64.0        0           28.0
5 Snapchat          1.00      3.00     52.0      63.0        0           35.0
6 Twitter           5.00     49.0     100        34.0        0           50.0
```

Computing on Sentiment

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

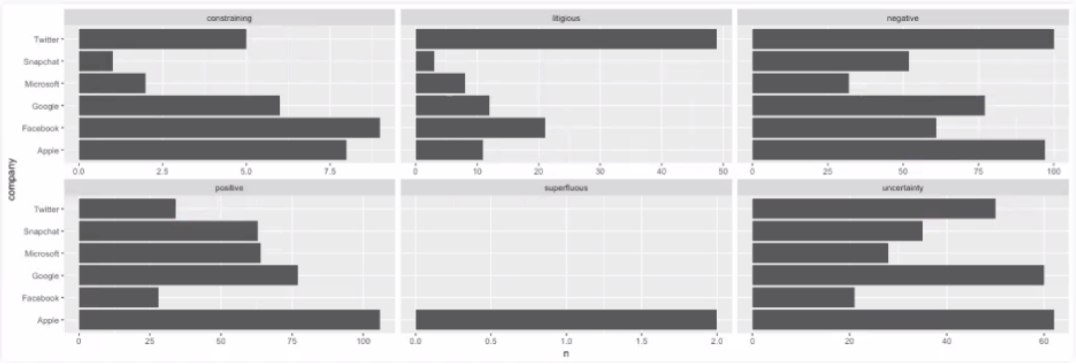
```
stock_sentiment_count %>%
  mutate(score = (positive - negative) / (positive + negative)) %>%
  mutate(company = reorder(company, score)) %>%
  ggplot(aes(company, score, fill = score > 0)) +
  geom_col(show.legend = FALSE) +
  coord_flip() +
  labs(x = "Company",
       y = "Positivity score among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



Looking At Each

Ok, let's actually look at each one by sentiment count...

```
stock_sentiment_count <- stock_tokens %>%
    inner_join(get_sentiments("loughran"), by = "word") %>%
    count(sentiment, company) %>%
    ggplot(aes(x=company, y=n)) +
    geom_col() +
    coord_flip() +
    facet_wrap(~sentiment, scales="free_x")
```



Topic Modeling

In text mining, we often have collections of documents, such as blog posts or news articles, that we'd like to divide into natural groups so that we can understand them separately. Topic modeling is a method for unsupervised classification of such documents, similar to clustering on numeric data, which finds natural groups of items even when we're not sure what we're looking for.

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

Latent Dirichlet allocation

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

- Every document is a mixture of topics. We imagine that each document may contain words from several topics in particular proportions. For example, in a two-topic model we could say "Document 1 is 90% topic A and 10% topic B, while Document 2 is 30% topic A and 70% topic B."
- Every topic is a mixture of words. For example, we could imagine a two-topic model of American news, with one topic for "politics" and one for "entertainment." The most common words in the politics topic might be "President", "Congress", and "government", while the entertainment topic may be made up of words such as "movies", "television", and "actor". Importantly, words can be shared between topics; a word like "budget" might appear in both equally.

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

Topic Modeling the AP

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

```
library(topicmodels)

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

We can use the LDA() function from the topicmodels package, setting k = 2, to create a two-topic LDA model. Why 2? Because it's useful from a training standpoint. IRL you will have more than 2 topics to be sure.

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

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

Fitting the model was the "easy part": the rest of the analysis will involve exploring and interpreting the model using tidying

Word-topic probabilities

In day 1's motivating example we introduced the tidy() method, originally from the broom package (Robinson 2017), for tidying model objects. The tidytext package provides this method for extracting the per-topic-per-word probabilities, called β , from the model.

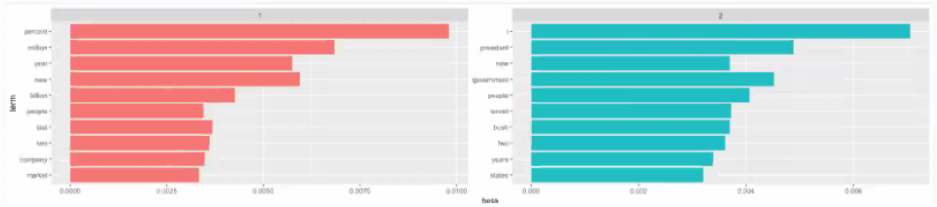
```
library(tidytext)

ap_topics <- tidy(ap_lda, matrix = "beta")
ap_topics
## # A tibble: 20,946 x 3
##    topic       term         beta
##    <int>      <chr>        <dbl>
##  1     1      aaron 1.686917e-12
##  2     2      aaron 3.895941e-05
##  3     1    abandon 2.654910e-05
##  4     2    abandon 3.990786e-05
##  5     1  abandoned 1.390663e-04
...
```

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

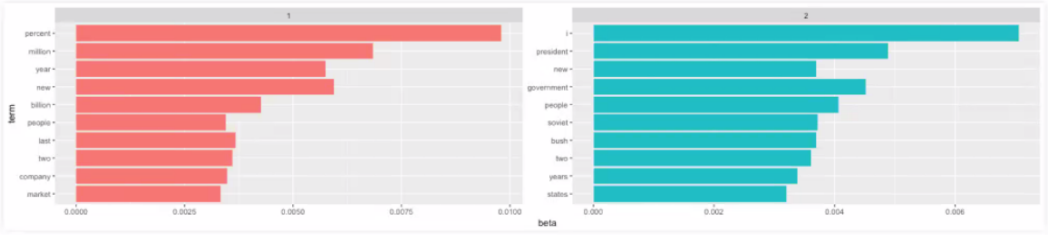
Visualizing Word-topic Probabilities

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

```
library(ggplot2)
library(dplyr)

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

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



Undertanding Word-Topic Frequencies

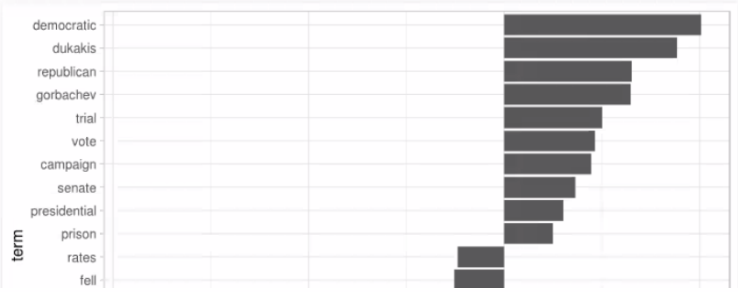


This visualization lets us understand the two topics that were extracted from the articles. The most common words in topic 1 include "percent", "million", "billion", and "company", which suggests it may represent business or financial news. Those most common in topic 2 include "president", "government", and "soviet", suggeting that this topic represents political news. One important observation about the words in each topic is that some words, such as "new" and "people", are common within both topics. This is an advantage of topic modeling as opposed to "hard clustering" methods: topics used in natural language could have some overlap in terms of words.

As an alternative, we could consider the terms that had the greatest difference in β between topic 1 and topic 2. This can be estimated based on the log ratio of the two: log2(β2/β1) (a log ratio is useful because it makes the difference symmetrical: β2 being twice as large leads to a log ratio of 1, while β1 being twice as large results in -1). To constrain it to a set of especially relevant words, we can filter for relatively common words, such as those that have a β greater than 1/1000 in at least one topic.

Visualizing Gretest Differences in Beta

```
library(tidyr)

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

beta_spread
## # A tibble: 198 x 4
##             term       topic1       topic2   log_ratio
##            <chr>        <dbl>        <dbl>       <dbl>
##  1 administration 4.309502e-04 1.382244e-03   1.6814189
##  2            ago 1.065216e-03 8.421279e-04  -0.3390353
##  3      agreement 6.714984e-04 1.039024e-03   0.6297728
```



Analysis of Visualization

We can see that the words more common in Topic 2 include political parties such as

- democratic
- republican

as well as politician's names such as

- dukakis
- gorbachev

Topic 1 was more characterized by currencies like

- yen
- dollar

as well as financial terms such as

- index
- prices
- rates

This helps confirm that the two topics the algorithm identified were political and financial news.

Document-topic probabilities

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

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

## # A tibble: 4,492 x 3
##    document topic        gamma
##       <int> <int>        <dbl>
##  1        1     1 0.2480616686
##  2        2     1 0.3615485445
##  3        3     1 0.5265844180
##  4        4     1 0.3566530023
##  5        5     1 0.1812766762
##  6        6     1 0.0005883388
```

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

We can see that many of these documents were drawn from a mix of the two topics, but that document 6 was drawn almost entirely from topic 2, having a γ from topic 1 close to zero. To check this answer, we could tidy() the document-term matrix and check what the most common words in that document were.

Introspecting The DTM (the Tidy way)

```
tidy(AssociatedPress) %>%
  filter(document == 6) %>%
  arrange(desc(count))
## # A tibble: 287 x 3
##    document           term count
##       <int>          <chr> <dbl>
##  1        6        noriega    16
##  2        6         panama    12
##  3        6        jackson     6
##  4        6         powell     6
##  5        6 administration     5
##  6        6       economic     5
##  7        6        general     5
##  8        6              i     5
##  9        6     panamanian     5
## 10        6       american     4
## # ... with 277 more rows
```

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

Case study: mining NASA metadata

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

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

```
library(jsonlite)
metadata <- fromJSON("https://data.nasa.gov/data.json")
names(metadata$dataset)
##  [1] "_id"             "@type"           "accessLevel"       "accrualPeriodicity"
##  [5] "bureauCode"      "contactPoint"    "description"       "distribution"
##  [9] "identifier"      "issued"          "keyword"           "landingPage"
## [13] "language"        "modified"        "programCode"       "publisher"
## [17] "spatial"         "temporal"        "theme"             "title"
## [21] "license"         "isPartOf"        "references"        "rights"
## [25] "describedBy"
```

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

It seems likely that the title, description, and keywords for each dataset may be most fruitful for drawing connections between datasets.

Wrangling and tidying the data

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

```
library(dplyr)

nasa_title <- data_frame(id = metadata$dataset$`_id`$`$oid`,
                         title = metadata$dataset$title)
nasa_title
## # A tibble: 32,089 x 2
##                          id                                       title
##                       <chr>                                       <chr>
##  1 55942a57c63a7fe59b495a77           15 Minute Stream Flow Data: USGS (FIFE)
##  2 55942a58c63a7fe59b495a7c 2000 Pilot Environmental Sustainability Index (ESI)
##  3 55942a58c63a7fe59b495a7d       2001 Environmental Sustainability Index (ESI)
```

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

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

nasa_desc %>%
  select(desc) %>% sample_n(5)
## # A tibble: 5 x 1
##                                                                              desc
##                                                                             <chr>
## 1          Version 2.0 Aquarius Level 3 ocean surface wind speed standard mapped Monthly image data
## 2 The Birds of the Americas Family Presence Grids of the Gridded Species Distribution, Version 1  is a
## 3 "The EUMETSAT OSI-SAF NAR SST products are SST fields derived from NOAA/AVHRR data and available ove
```

Building Tidy Data

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

```
library(tidyr)

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

nasa_keyword
## # A tibble: 126,814 x 2
##                          id       keyword
##                       <chr>         <chr>
##  1 55942a57c63a7fe59b495a77 EARTH SCIENCE
##  2 55942a57c63a7fe59b495a77   HYDROSPHERE
##  3 55942a57c63a7fe59b495a77 SURFACE WATER
##  4 55942a57c63a7fe59b495a78 EARTH SCIENCE
##  5 55942a57c63a7fe59b495a78   HYDROSPHERE
##  6 55942a57c63a7fe59b495a78 SURFACE WATER
```

This is a tidy data frame because we have one row for each keyword; this means we will have multiple rows for each dataset because a dataset can have more than one keyword. Now it is time to use tidytext's unnest_tokens() for the title and description fields so we can do the text analysis. Let's also remove stop words from the titles and descriptions. We will not remove stop words from the keywords, because those are short, human-assigned keywords like "RADIATION" or "CLIMATE INDICATORS".

```
library(tidytext)

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

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

This is a tidy data frame because we have one row for each keyword; this means we will have multiple rows for each dataset because a dataset can have more than one keyword. Now it is time to use tidytext's unnest_tokens() for the title and description fields so we can do the text analysis. Let's also remove stop words from the titles and descriptions. We will not remove stop words from the keywords, because those are short, human-assigned keywords like "RADIATION" or "CLIMATE INDICATORS".

```
library(tidytext)

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

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

Some initial simple exploration

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

```
nasa_title %>%
  count(word, sort = TRUE)
## # A tibble: 11,614 x 2
##       word     n
##      <chr> <int>
##  1 project  7735
##  2    data  3354
##  3       1  2841
##  4   level  2400
##  5  global  1809
##  6      v1  1478
```

What about the descriptions?

```
nasa_desc %>%
  count(word, sort = TRUE)
## # A tibble: 35,940 x 2
##          word     n
##         <chr> <int>
##  1       data 68871
##  2      modis 24420
##  3     global 23028
##  6     system 15480
##  7    product 14780
```

Words like "data" and "global" are used very often in NASA titles and descriptions. We may want to remove digits and some "words" like "v1" from these data frames for many types of analyses; they are not too meaningful for most audiences.

What are the most common keywords?

Word co-ocurrences and correlations

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

Networks of Description and Title Words

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

```
library(widyr)

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

title_word_pairs
## # A tibble: 156,689 x 3
##      item1   item2     n
##      <chr>   <chr> <dbl>
##  1 system project   796
##  2    lba     eco   683
##  3   airs    aqua   641
##  4  level    aqua   623
##  5  level    airs   612
##  6   aura     omi   607
```

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

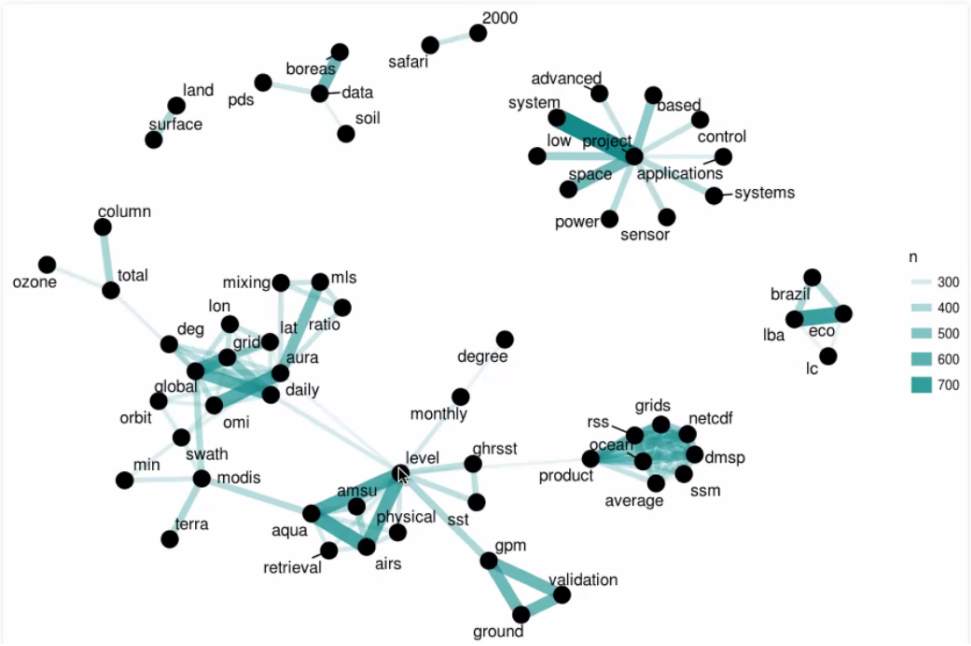
```
desc_word_pairs <- nasa_desc %>%
  pairwise_count(word, id, sort = TRUE, upper = FALSE)

desc_word_pairs
## # A tibble: 10,889,084 x 3
##         item1      item2     n
```

Visualizing Co-Occurances

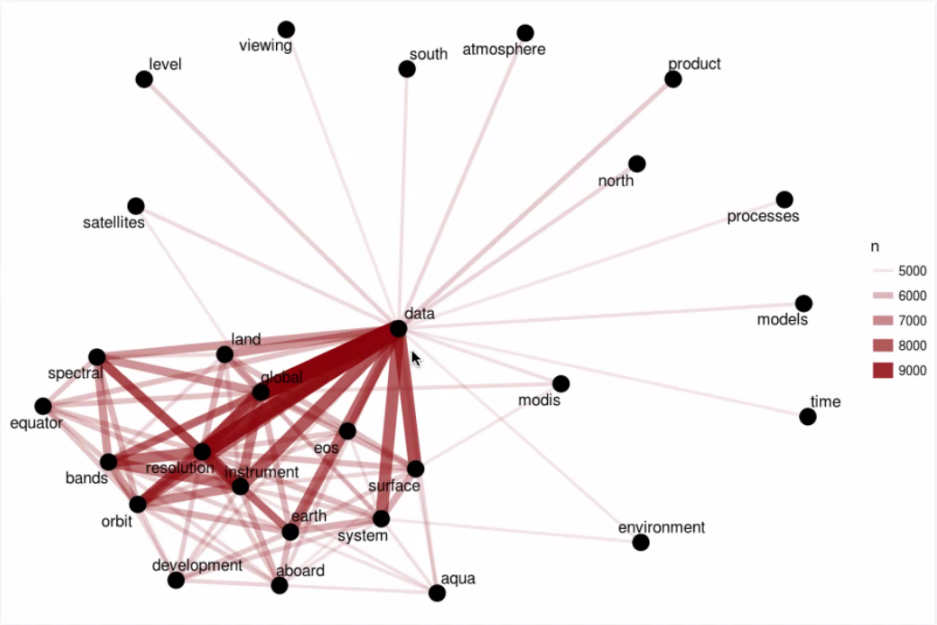
Let's plot networks of these co-occurring words so we can see these relationships better in Figure 8.1. We will again use the ggraph package for visualizing our networks.

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

set.seed(1234)
title_word_pairs %>%
  filter(n >= 250) %>%
  graph_from_data_frame() %>%
  ggraph(layout = "fr") +
  geom_edge_link(aes(edge_alpha = n, edge_width = n), edge_colour = "cyan4") +
  geom_node_point(size = 5) +
  geom_node_text(aes(label = name), repel = TRUE,
                 point.padding = unit(0.2, "lines")) +
  theme_void()
```

Visualizing Titles



Visualizing Descriptions



Case study: analyzing usenet text

In our final case study, we'll use what we've learned in this book to perform a start-to-finish analysis of a set of 20,000 messages sent to 20 Usenet bulletin boards in 1993. The Usenet bulletin boards in this dataset include newsgroups for topics like politics, religion, cars, sports, and cryptography, and offer a rich set of text written by many users. This data set is publicly available at http://qwone.com/~jason/20Newsgroups/ (the 20news-bydate.tar.gz file) and has become popular for exercises in text analysis and machine learning.

Pre-processing

We'll start by reading in all the messages from the 20news-bydate folder, which are organized in sub-folders with one file for each message. We can read in files like these with a combination of read_lines(), map() and unnest().

```
library(dplyr)
library(tidyr)
library(purrr)
library(readr)
training_folder <- "data/20news-bydate/20news-bydate-train/"

# Define a function to read all files from a folder into a data frame
read_folder <- function(infolder) {
  data_frame(file = dir(infolder, full.names = TRUE)) %>%
    mutate(text = map(file, read_lines)) %>%
    transmute(id = basename(file), text) %>%
    unnest(text)
}

# Use unnest() and map() to apply read_folder to each subfolder
raw_text <- data_frame(folder = dir(training_folder, full.names = TRUE)) %>%
  unnest(map(folder, read_folder)) %>%
  transmute(newsgroup = basename(folder), id, text)
raw_text
## # A tibble: 511,655 x 3
##      newsgroup    id                                                      text
##          <chr> <chr>                                                     <chr>
##  1 alt.atheism 49960                                 From: mathew <mathew@mantis.co.uk>
##  2 alt.atheism 49960                             Subject: Alt.Atheism FAQ: Atheist Resources
##  3 alt.atheism 49960    Summary: Books, addresses, music -- anything related to atheism
##  4 alt.atheism 49960 Keywords: FAQ, atheism, books, music, fiction, addresses, contacts
```

Visualizing News Groups

Notice the newsgroup column, which describes which of the 20 newsgroups each message comes from, and id column, which identifies a unique message within that newsgroup. What newsgroups are included, and how many messages were posted in each? We can see that Usenet newsgroup names are named hierarchically, starting with a main topic such as "talk", "sci", or "rec", followed by further specifications.

```
library(ggplot2)

raw_text %>%
  group_by(newsgroup) %>%
  summarize(messages = n_distinct(id)) %>%
  ggplot(aes(newsgroup, messages)) +
  geom_col() +
  coord_flip()
```

Pre-processing text

Most of the datasets we've examined in this book were pre-processed, meaning we didn't have to remove, for example, copyright notices from the Jane Austen novels. Here, however, each message has some structure and extra text that we don't want to include in our analysis. For example, every message has a header, containing field such as "from:" or "in_reply_to:" that describe the message. Some also have automated email signatures, which occur after a line like –.

This kind of pre-processing can be done within the dplyr package, using a combination of cumsum() (cumulative sum) and str_detect() from stringr.

```
library(stringr)

# must occur after the first occurrence of an empty line,
# and before the first occurrence of a line starting with --
cleaned_text <- raw_text %>%
  group_by(newsgroup, id) %>%
  filter(cumsum(text == "") > 0,
         cumsum(str_detect(text, "^--")) == 0) %>%
  ungroup()
```

Pre-processing text

Many lines also have nested text representing quotes from other users, typically starting with a line like "so-and-so writes..." These can be removed with a few regular expressions.

```
cleaned_text <- cleaned_text %>%
  filter(str_detect(text, "^[^>]+[A-Za-z\\d]") | text == "",
         !str_detect(text, "writes(:|\\.\\.\\.)$"),
         !str_detect(text, "^In article <"))
```

At that point, we're ready to use unnest_tokens() to split the dataset into tokens, while removing stop-words.

```
library(tidytext)

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

IMPORTANT: Every raw text dataset will require different steps for data cleaning, which will often involve some trial-and-error and exploration of unusual cases in the dataset. It's important to notice that this cleaning can be achieved using tidy tools such as dplyr and tidyr.

Words in newsgroups

Now that we've removed the headers, signatures, and formatting, we can start exploring common words. For starters, we could find the most common words in the entire dataset, or within particular newsgroups.

```
usenet_words %>%
  count(word, sort = TRUE)
## # A tibble: 68,137 x 2
##           word     n
##          <chr> <int>
##  1      people  3655
##  2        time  2705
##  3         god  1626
##  4      system  1595
##  5     program  1103

words_by_newsgroup <- usenet_words %>%
  count(newsgroup, word, sort = TRUE) %>%
  ungroup()

words_by_newsgroup
## # A tibble: 173,913 x 3
##                   newsgroup      word     n
##                       <chr>     <chr> <int>
##  1  soc.religion.christian       god   917
##  2               sci.space     space   840
##  3   talk.politics.mideast    people   728
##  4               sci.crypt       key   704
##  5 comp.os.ms-windows.misc   windows   625
```

Finding tf-idf within newsgroups

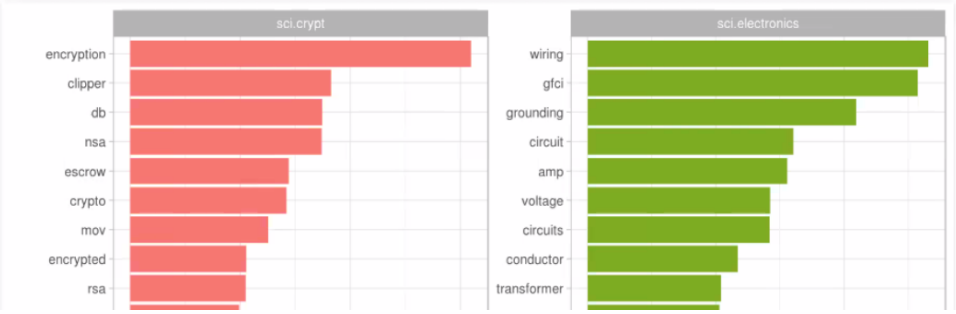
We'd expect the newsgroups to differ in terms of topic and content, and therefore for the frequency of words to differ between them. Let's try quantifying this using the tf-idf metric (day 2).

```
tf_idf <- words_by_newsgroup %>%
  bind_tf_idf(word, newsgroup, n) %>%
  arrange(desc(tf_idf))

tf_idf
## # A tibble: 173,913 x 6
##                   newsgroup           word     n          tf      idf     tf_idf
##                       <chr>          <chr> <int>       <dbl>    <dbl>      <dbl>
##  1 comp.sys.ibm.pc.hardware           scsi   483 0.017616807 1.203973 0.02121016
##  2    talk.politics.mideast       armenian   582 0.008048902 2.302585 0.01853328
##  3          rec.motorcycles           bike   324 0.013898421 1.203973 0.01673332
##  4    talk.politics.mideast      armenians   509 0.007039332 2.302585 0.01620866
##  5               sci.crypt     encryption   410 0.008160990 1.897120 0.01548238
##  6         rec.sport.hockey            nhl   157 0.004396651 2.995732 0.01317119
##  7       talk.politics.misc stephanopoulos   158 0.004162276 2.995732 0.01246906
##  8          rec.motorcycles          bikes    97 0.004160947 2.995732 0.01246508
##  9         rec.sport.hockey         hockey   270 0.007561119 1.609438 0.01216915
## 10           comp.windows.x          oname   136 0.003535498 2.995732 0.01059141
## # ... with 173,903 more rows
```

We can examine the top tf-idf for a few selected groups to extract words specific to those topics. For example, we could look at all the sci. boards

Visualizing The 12 terms with the highest tf-idf within each of the science-related newsgroups

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



Newsgroup Similarity

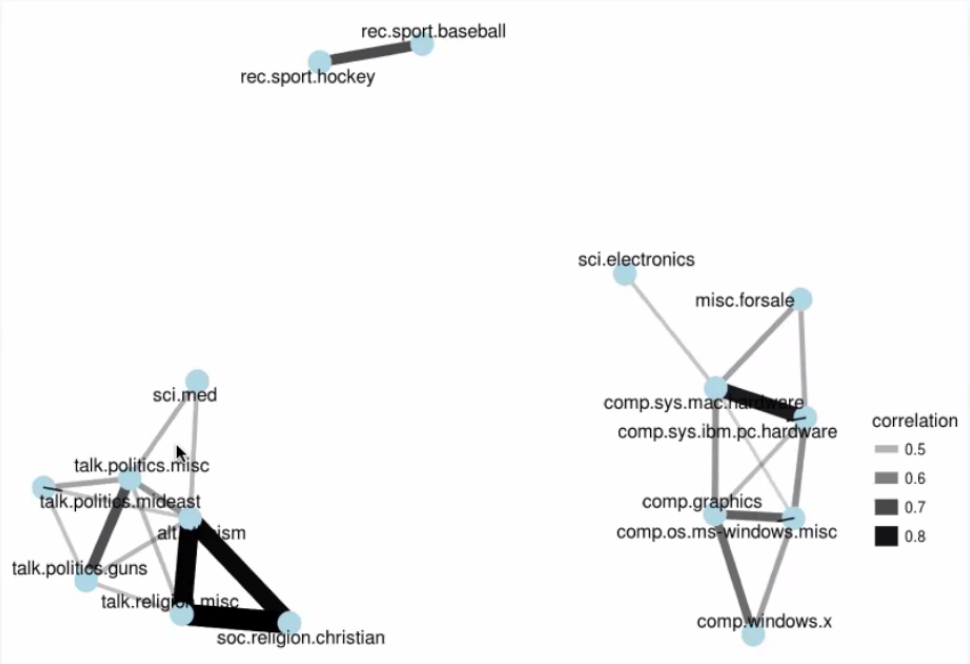
What newsgroups tended to be similar to each other in text content? We could discover this by finding the pairwise correlation of word frequencies within each newsgroup, using the pairwise_cor() function from the widyr package!

```
library(widyr)

newsgroup_cors <- words_by_newsgroup %>%
  pairwise_cor(newsgroup, word, n, sort = TRUE)

newsgroup_cors
## # A tibble: 380 x 3
##                       item1                   item2 correlation
##                       <chr>                   <chr>       <dbl>
##  1       talk.religion.misc   soc.religion.christian   0.8347275
##  2   soc.religion.christian       talk.religion.misc   0.8347275
##  3             alt.atheism       talk.religion.misc   0.7793079
##  4       talk.religion.misc             alt.atheism   0.7793079
```

We could then filter for stronger correlations among newsgroups, and visualize them in a network...

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

newsgroup_cors %>%
  filter(correlation > .4) %>%
  graph_from_data_frame() %>%
  ggraph(layout = "fr") +
  geom_edge_link(aes(alpha = correlation, width = correlation)) +
  geom_node_point(size = 6, color = "lightblue") +
  geom_node_text(aes(label = name), repel = TRUE) +
  theme_void()
```

Newsgroup Similarity



Topic modeling

Earlier today, we used the latent Dirichlet allocation (LDA) algorithm to divide a set of chapters into the books they originally came from. Could LDA do the same to sort out Usenet messages that came from different newsgroups?

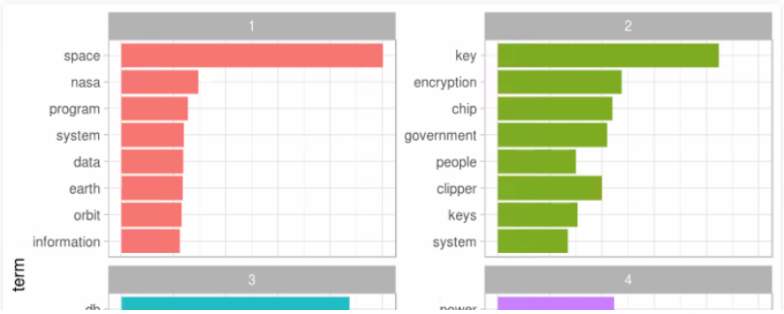
Let's try dividing up messages from the four science-related newsgroups. We first process these into a document-term matrix with cast_dtm(), then fit the model with the LDA() function from the topicmodels package.

```
# include only words that occur at least 50 times
word_sci_newsgroups <- usenet_words %>%
  filter(str_detect(newsgroup, "^sci")) %>%
  group_by(word) %>%
  mutate(word_total = n()) %>%
  ungroup() %>%
  filter(word_total > 50)

# convert into a document-term matrix
# with document names such as sci.crypt_14147
sci_dtm <- word_sci_newsgroups %>%
  unite(document, newsgroup, id) %>%
  count(document, word) %>%
  cast_dtm(document, word, n)
library(topicmodels)
sci_lda <- LDA(sci_dtm, k = 4, control = list(seed = 2016))
```

What four topics did this model extract, and did they match the four newsgroups? This approach will look familiar: we visualize each topic based on the most frequent terms within it.

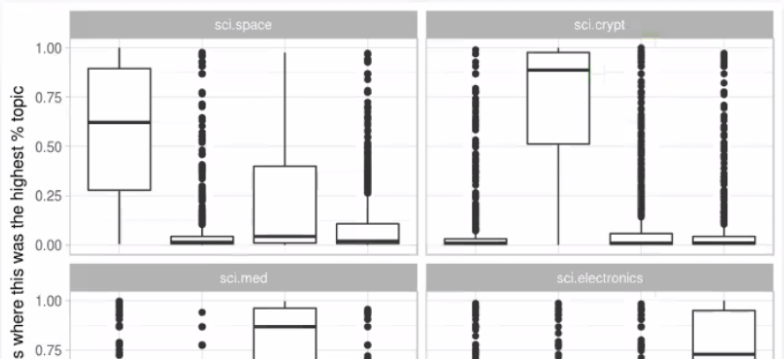
Topic modeling - The top 8 words from each topic fit by LDA

```
sci_lda %>%
  tidy() %>%
  group_by(topic) %>%
  top_n(8, beta) %>%
  ungroup() %>%
  mutate(term = reorder(term, beta)) %>%
  ggplot(aes(term, beta, fill = factor(topic))) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~ topic, scales = "free_y") +
  coord_flip()
```



Confirming Topics - Distribution of gamma for each topic within each Usenet newsgroup

```
sci_lda %>%
  tidy(matrix = "gamma") %>%
  separate(document, c("newsgroup", "id"), sep = "_") %>%
  mutate(newsgroup = reorder(newsgroup, gamma * topic)) %>%
  ggplot(aes(factor(topic), gamma)) +
  geom_boxplot() +
  facet_wrap(~ newsgroup) +
  labs(x = "Topic",
       y = "# of messages where this was the highest % topic")
```

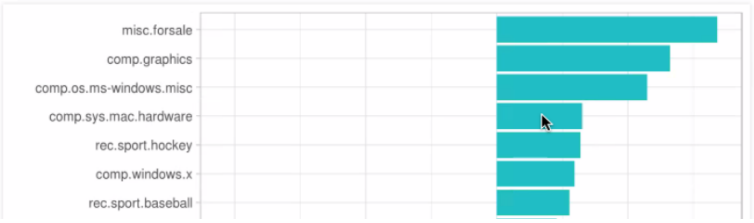


Sentiment Analysis

We can use the sentiment analysis techniques we explored earlier in the class to examine how often positive and negative words occurred in these Usenet posts. Which newsgroups were the most positive or negative overall?

In this example we'll use the AFINN sentiment lexicon, which provides numeric positivity scores for each word, and visualize it with a bar plot.

```
newsgroup_sentiments <- words_by_newsgroup %>%
  inner_join(get_sentiments("afinn"), by = "word") %>%
  group_by(newsgroup) %>%
  summarize(score = sum(score * n) / sum(n))

newsgroup_sentiments %>%
  mutate(newsgroup = reorder(newsgroup, score)) %>%
  ggplot(aes(newsgroup, score, fill = score > 0)) +
  geom_col(show.legend = FALSE) +
  coord_flip() +
  ylab("Average sentiment score")
```



Sentiment analysis by word

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

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

contributions
## # A tibble: 1,909 x 3
##         word occurences contribution
##        <chr>      <int>        <int>
##  1   abandon         13          -26
##  2 abandoned         19          -38
##  3  abandons          3           -6
##  4 abduction          2           -4
```

Which words contriubted the most?

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

Diving Deepper Into the Sentiment

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

We may also care about which words contributed the most within each newsgroup, so that we can see which newsgroups might be incorrectly estimated. We can calculate each word's contribution to each newsgroup's sentiment score, and visualize the strongest contributors from a selection of the groups

```
top_sentiment_words <- words_by_newsgroup %>%
  inner_join(get_sentiments("afinn"), by = "word") %>%
  mutate(contribution = score * n / sum(n))

top_sentiment_words
## # A tibble: 13,063 x 5
##                 newsgroup   word     n score contribution
##                     <chr>  <chr> <int> <int>        <dbl>
##  1 soc.religion.christian    god   917     1  0.014418012
##  2 soc.religion.christian  jesus   440     1  0.006918130
##  3     talk.politics.guns    gun   425    -1 -0.006682285
##  4     talk.religion.misc    god   296     1  0.004654015
##  5           alt.atheism    god   268     1  0.004213770
##  6 soc.religion.christian  faith   257     1  0.004040817
##  7     talk.religion.misc  jesus   256     1  0.004025094
##  8  talk.politics.mideast killed   202    -3 -0.009528152
##  9  talk.politics.mideast    war   187    -2 -0.005880411
## 10 soc.religion.christian   true   179     2  0.005628842
## # ... with 13,053 more rows
```

Conclusions about newsgroups

This confirms our hypothesis about the "misc.forsale" newsgroup: most of the sentiment was driven by positive adjectives such as "excellent" and "perfect". We can also see how much sentiment is confounded with topic. An atheism newsgroup is likely to discuss "god" in detail even in a negative context, and we can see that it makes the newsgroup look more positive. Similarly, the negative contribution of the word "gun" to the "talk.politics.guns" group will occur even when the members are discussing guns positively.

This helps remind us that sentiment analysis can be confounded by topic, and that we should always examine the influential words before interpreting it too deeply.

Also, quite frankly, culture and gender matter. A newsgroup full of American men will use different sentiment baseline than a newsgroup full of Japanese women.

Sentiment analysis by message

We can also try finding the most positive and negative individual messages, by grouping and summarizing by id rather than newsgroup.

```
sentiment_messages <- usenet_words %>%
  inner_join(get_sentiments("afinn"), by = "word") %>%
  group_by(newsgroup, id) %>%
  summarize(sentiment = mean(score),
            words = n()) %>%
  ungroup() %>%
  filter(words >= 5) # if it has less than 5 words, it's noise
```

What were the most positive messages?

```
sentiment_messages %>%
  arrange(desc(sentiment))
## # A tibble: 3,554 x 4
##                 newsgroup     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
```

Let's check this by looking at the most positive message in the whole dataset.

The MOST Positive Message on Usenet

```
## 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!!!!!
```

It was fooled, basically. It just says winner a lot.

The MOST Negative Message on Usenet

```
sentiment_messages %>%
  arrange(sentiment)
## # A tibble: 3,554 x 4
##                newsgroup     id sentiment words
##                    <chr>  <chr>     <dbl> <int>
##  1      rec.sport.hockey  53907 -3.000000     6
##  2       sci.electronics  53899 -3.000000     5
##  3 talk.politics.mideast  75918 -3.000000     7
##  4             rec.autos 101627 -2.833333     6
```

Hockey again! People sure do feel a lot of feelings about sports...

```
print_message("rec.sport.hockey", 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--
```

N-gram analysis

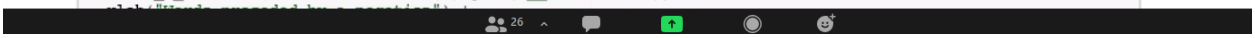
We have considered the effect of words such as "not" and "no" on sentiment analysis of Jane Austen novels, such as considering whether a phrase like "don't like" led to passages incorrectly being labeled as positive. The Usenet dataset is a much larger corpus of more modern text, so we may be interested in how sentiment analysis may be reversed in this text.

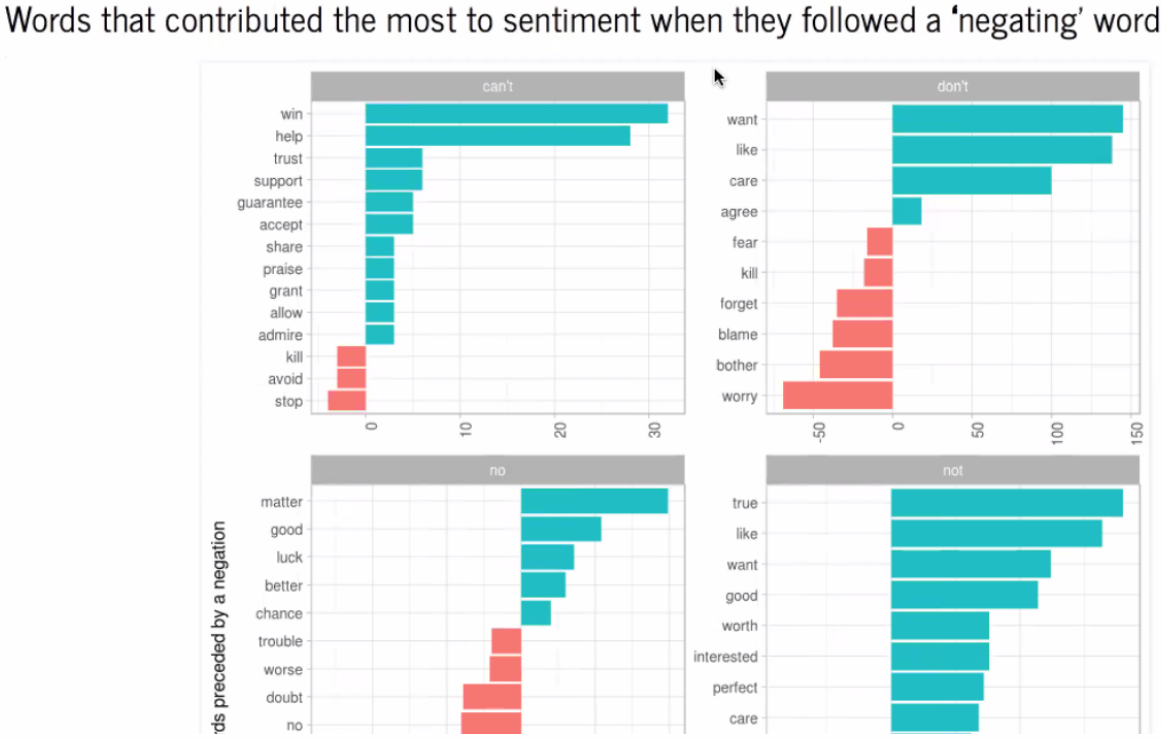
We'd start by finding and counting all the bigrams in the Usenet posts.

```
usenet_bigrams <- cleaned_text %>%
  unnest_tokens(bigram, text, token = "ngrams", n = 2)
usenet_bigram_counts <- usenet_bigrams %>%
  count(newsgroup, bigram, sort = TRUE) %>%
  ungroup() %>%
  separate(bigram, c("word1", "word2"), sep = " ")
```

We could then define a list of six words that we suspect are used in negation, such as "no", "not", and "without", and visualize the sentiment-associated words that most often followed them. This shows the words that most often contributed in the "wrong" direction.

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

usenet_bigram_counts %>%
  filter(word1 %in% negate_words) %>%
  count(word1, word2, wt = n, sort = TRUE) %>%
  inner_join(get_sentiments("afinn"), by = c(word2 = "word")) %>%
  mutate(contribution = score * nn) %>%
  group_by(word1) %>%
  top_n(10, abs(contribution)) %>%
  ungroup() %>%
  mutate(word2 = reorder(paste(word2, word1, sep = "__"), contribution)) %>%
  ggplot(aes(word2, contribution, fill = contribution > 0)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~ word1, scales = "free", nrow = 3) +
  scale_x_discrete(labels = function(x) gsub("__.+$", "", x)) +
```





Summary

NOTE: No matter isn't *actually* negative sentiment! Even our attempt to outsmart negation may be misled. This is tough stuff.

In this analysis of Usenet messages, we've incorporated almost every method for tidy text mining described in this book, ranging from tf-idf to topic modeling and from sentiment analysis to n-gram tokenization. Throughout the chapter, and indeed through all of our case studies, we've been able to rely on a small list of common tools for exploration and visualization. We hope that these examples show how much all tidy text analyses have in common with each other, and indeed with all tidy data analyses.

What We Have Learned

Wow. I must admit, at this point my brain is a little fried. But I think it's important we cover what we've learned...

- What strings are
- How to get text into R from docs, twitter, pdfs, images, the Guardian, the web...
- How to manipulate strings and text
- How to match it with regular expressions
- How to use tidytext to turn text into tidy data structures
- How to use dplyr verbs to do powerful analysis
- How to do correlations on text to find similarities
- How to handle stop words
- How to use inner_join to do sentiment analysis
- How to use tf-idf to understand what makes a document unique
- How to do whole sentence sentiment analysis (if we want)
- How to create ngrams
- How to visualize the relationships between ngrams and words
- How to leverage topic modeling
- How to work with large real world text data

It has been an amazing 3 days. I can not wait to see what your final projects will be like...