ISOM 3390: Business Programming in R

Instructor: Bingjie Qian (PhD candidate)

Summer 2020, HKUST

Topic 11: Text Analytics

11.1 Overview of Text Analytics

Text analytics is to teach computers to understand text, for example, learning the general public's opinions on the firm's products or services from customers' online reviews.

Basic Workflow for Text Analytics

- Obtain the text sources
- Extract documents and move into a corpus
- Transformation, which typically involves:
  - **Case folding** - usually convert to lower case
  - **Punctuation and number removal**
  - **Stop words removal** - remove common words that are not informative, e.g., "the", "of", "to" in English
  - **Stemming** - reduce words to their word stem, e.g., "Fishing", "fished", and "fisher" -> "fish"
- Extract features - convert the text string into some sort of quantifiable measures
- Perform analysis - e.g., text classification, topic modeling, etc.

`tidytext`

The `tidytext` package provides useful functions (e.g., `unnest_tokens()` , `bind_tf_idf()` , `cast_dtm()`) and datasets (e.g., `stop_words` , `sentiments`) for text analytics, and allows conversion of text to and from tidy formats.

```
# install.packages("tidytext")
library(tidytext)
```

Arranging text in the tidy format allows us to use the `tidyverse` functions to explore and visualize text data coherently.

```
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.0 ──
## ✓ ggplot2 3.3.2     ✓ purrr   0.3.4
## ✓ tibble  3.0.1     ✓ dplyr   1.0.0
## ✓ tidyr   1.1.0     ✓ stringr 1.4.0
## ✓ readr   1.3.1     ✓ forcats 0.5.0
```

```
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

11.2 Text Analytics of Jane Austen's Books

We will use Jane Austen's books as an example of text analytics.

The `janeaustenr` package provides the text of Jane Austen's 6 published novels in a one-row-per-line format.

```
# install.packages("janeaustenr")
library(janeaustenr)
```

The function `austen_books()` returns a tidy data frame of all 6 novels.

```
austen_books()
```

```
## # A tibble: 73,422 x 2
##    text                    book
##  * <chr>                   <fct>
##  1 "SENSE AND SENSIBILITY" Sense & Sensibility
##  2 ""                      Sense & Sensibility
##  3 "by Jane Austen"        Sense & Sensibility
##  4 ""                      Sense & Sensibility
##  5 "(1811)"                Sense & Sensibility
##  6 ""                      Sense & Sensibility
##  7 ""                      Sense & Sensibility
##  8 ""                      Sense & Sensibility
##  9 ""                      Sense & Sensibility
## 10 "CHAPTER 1"             Sense & Sensibility
## # … with 73,412 more rows
```

```
unique(austen_books()$book)   # 6 books
```

```
## [1] Sense & Sensibility Pride & Prejudice   Mansfield Park
## [4] Emma                Northanger Abbey    Persuasion
## 6 Levels: Sense & Sensibility Pride & Prejudice Mansfield Park ... Persuasion
```

Indexing and Annotating Lines

Use `mutate()` in `dplyr` to annotate each line with its line number and chapter:

```
original_books <- austen_books() %>% group_by(book) %>% mutate(linenumber = row_numbe
r(), chapter = cumsum(str_detect(text, regex("^chapter [\\divxlc]", ignore_case = TRU
E)))) %>% ungroup()
original_books
```

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

`regex()` is a function in `stringr` that controls the actual matching behavior defined by a regular expression.

Tokenizing Text

Tokenization is the process of tokenizing or splitting a string or text into a list of tokens. You can think of tokens as parts, like a word is a token in a sentence, and a sentence is a token in a paragraph.

`tidytext` provides a useful function `unnest_tokens()` for tokenizing text:

```
str(unnest_tokens)
```

```
## function (tbl, output, input, token = "words", format = c("text", "man",
##     "latex", "html", "xml"), to_lower = TRUE, drop = TRUE, collapse = NULL,
##     ...)
```

- `unnest_tokens()` splits a column into tokens, splitting the table into one-token-per-row.
- `tbl` : A data frame
- `output` : Output column to be created as string or symbol.
- `input` : Input column that gets split as string or symbol.
- `token` : Unit for tokenizing, or a custom tokenizing function. Built-in options include "words" (default), "characters", "ngrams", "sentences", "paragraphs", "regex", "tweets", etc.
- `to_lower` : Whether to convert tokens to lowercase. Defaults to `TRUE` .
- `...` : Extra arguments passed on to tokenizers, such as `strip_punct` for "words" and "tweets", `n` and `k` for "ngrams" and "skip_ngrams", `strip_url` for "tweets", and `pattern` for "regex".

Use `unnest_tokens()` to tokenize the `text` column in `original_books` , and remove all tokens which are strictly numbers:

```
tidy_books <- original_books %>% unnest_tokens(word, text) %>% filter(!str_detect(wor
d, "^[0-9]*$"))
tidy_books
```

```
## # A tibble: 724,852 x 4
##    book                linenumber chapter word
##    <fct>                    <int>   <int> <chr>
##  1 Sense & Sensibility          1       0 sense
##  2 Sense & Sensibility          1       0 and
##  3 Sense & Sensibility          1       0 sensibility
##  4 Sense & Sensibility          3       0 by
##  5 Sense & Sensibility          3       0 jane
##  6 Sense & Sensibility          3       0 austen
##  7 Sense & Sensibility         10       1 chapter
##  8 Sense & Sensibility         13       1 the
##  9 Sense & Sensibility         13       1 family
## 10 Sense & Sensibility         13       1 of
## # … with 724,842 more rows
```

The above code also strips all punctuations (`strip_punct = TRUE`), and converts each word to lowercase (`to_lower = TRUE`) for easy comparability.

Try other options of `token` .

Tokenize text into sentences:

```
original_books %>% unnest_tokens(sentence, text, token = "sentences") %>% head(5)
```

```
## # A tibble: 5 x 4
##   book             linenumber chapter sentence
##   <fct>                 <int>   <int> <chr>
## 1 Sense & Sensibi…          1       0 sense and sensibility
## 2 Sense & Sensibi…          3       0 by jane austen
## 3 Sense & Sensibi…          5       0 (1811)
## 4 Sense & Sensibi…         10       1 chapter 1
## 5 Sense & Sensibi…         13       1 the family of dashwood had long been sett…
```

Split text into chapters using a regular expression:

```
austen_books() %>% group_by(book) %>% unnest_tokens(chapter, text, token = "regex", p
attern = "Chapter|CHAPTER [\\dIVXLC]") %>% ungroup() %>% head(5)
```

```
## # A tibble: 5 x 2
##   book             chapter
##   <fct>            <chr>
## 1 Sense & Sensibi… "sense and sensibility\n\nby jane austen\n\n(1811)\n\n\n\n\n"
## 2 Sense & Sensibi… "\n\n\nthe family of dashwood had long been settled in susse…
## 3 Sense & Sensibi… "\n\n\nmrs. john dashwood now installed herself mistress of …
## 4 Sense & Sensibi… "\n\n\nmrs. dashwood remained at norland several months; not…
## 5 Sense & Sensibi… "\n\n\n\"what a pity it is, elinor,\" said marianne, \"that …
```

Removing Stop Words

Stop words are the common words that are not informative, e.g., "and", "the", "of", "to" in English.

`tidytext` has a dataset named `stop_words` for English stop words.

Use `anti_join()` or `filter()` to remove them from further analysis:

```
tidytext::stop_words
```

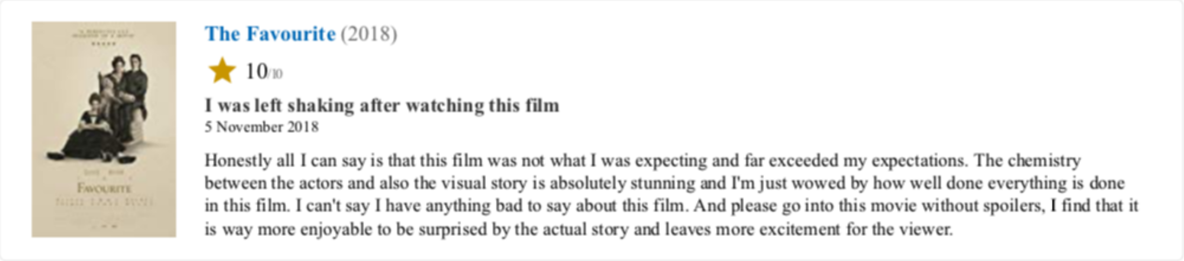
```
## # A tibble: 1,149 x 2
##    word        lexicon
##    <chr>       <chr>
##  1 a           SMART
##  2 a's         SMART
##  3 able        SMART
##  4 about       SMART
##  5 above       SMART
##  6 according   SMART
##  7 accordingly SMART
##  8 across      SMART
##  9 actually    SMART
## 10 after       SMART
## # … with 1,139 more rows
```

```
tidy_books <- tidy_books %>% anti_join(stop_words, by = "word")
tidy_books
```

```
## # A tibble: 217,406 x 4
##    book                linenumber chapter word
##    <fct>                    <int>   <int> <chr>
##  1 Sense & Sensibility          1       0 sense
##  2 Sense & Sensibility          1       0 sensibility
##  3 Sense & Sensibility          3       0 jane
##  4 Sense & Sensibility          3       0 austen
##  5 Sense & Sensibility         10       1 chapter
##  6 Sense & Sensibility         13       1 family
##  7 Sense & Sensibility         13       1 dashwood
##  8 Sense & Sensibility         13       1 settled
##  9 Sense & Sensibility         13       1 sussex
## 10 Sense & Sensibility         13       1 estate
## # … with 217,396 more rows
```

Sentiment Analysis

Human readers use their understandings of the emotional intent of words to infer whether a section of text is positive or negative, or perhaps characterize it by some more nuanced emotions like exciting, boring, surprising, etc.



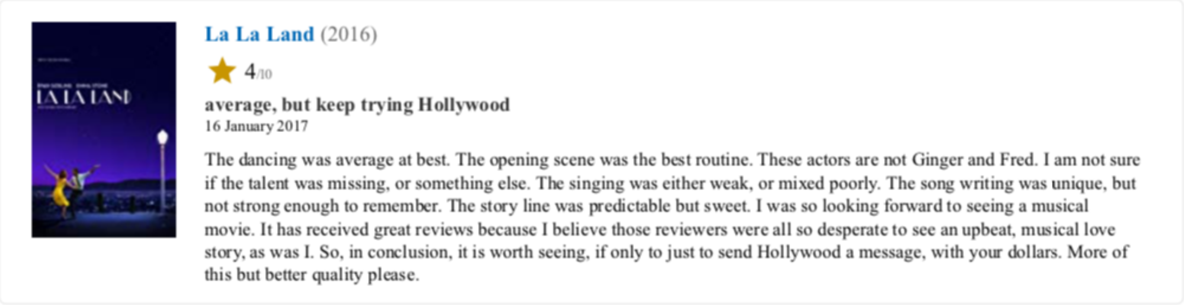
The Favourite (2018)

★ 10/10

I was left shaking after watching this film

5 November 2018

Honestly all I can say is that this film was not what I was expecting and far exceeded my expectations. The chemistry between the actors and also the visual story is absolutely stunning and I'm just wowed by how well done everything is done in this film. I can't say I have anything bad to say about this film. And please go into this movie without spoilers, I find that it is way more enjoyable to be surprised by the actual story and leaves more excitement for the viewer.



La La Land (2016)

★ 4/10

average, but keep trying Hollywood

16 January 2017

The dancing was average at best. The opening scene was the best routine. These actors are not Ginger and Fred. I am not sure if the talent was missing, or something else. The singing was either weak, or mixed poorly. The song writing was unique, but not strong enough to remember. The story line was predictable but sweet. I was so looking forward to seeing a musical movie. It has received great reviews because I believe those reviewers were all so desperate to see an upbeat, musical love story, as was I. So, in conclusion, it is worth seeing, if only to just to send Hollywood a message, with your dollars. More of this but better quality please.

Figure 1. Movie Reviews

Sentiment analysis draws upon positive and negative word sets (called sentiment lexicons or dictionaries) that convey human emotion or feeling.

The `tidytext` package provides the function `get_sentiments()` to get several sentiment lexicons ("bing", "afinn", "loughran", and "nrc") in the tidy format.

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

```
## # A tibble: 6,786 x 2
##    word        sentiment
##    <chr>       <chr>
##  1 2-faces     negative
##  2 abnormal    negative
##  3 abolish     negative
##  4 abominable  negative
##  5 abominably  negative
##  6 abominate   negative
##  7 abomination negative
##  8 abort       negative
##  9 aborted     negative
## 10 aborts      negative
## # … with 6,776 more rows
```

```
get_sentiments("afinn")  # need to install the `textdata` package to access this data
set
```

```
## # A tibble: 2,477 x 2
##    word       value
##    <chr>      <dbl>
##  1 abandon       -2
##  2 abandoned     -2
##  3 abandons      -2
##  4 abducted      -2
##  5 abduction     -2
##  6 abductions    -2
##  7 abhor         -3
##  8 abhorred      -3
##  9 abhorrent     -3
## 10 abhors        -3
## # … with 2,467 more rows
```

We can draw upon a lexicon to identify a list of positive and negative words, and count their numbers to derive *sentiment scores* for text.

Identify positive and negative words:

```
tidy_books_bing <- tidy_books %>% inner_join(get_sentiments("bing"), by = "word")
tidy_books_bing
```

```
## # A tibble: 44,171 x 5
##    book                linenumber chapter word        sentiment
##    <fct>                    <int>   <int> <chr>       <chr>
##  1 Sense & Sensibility         16       1 respectable positive
##  2 Sense & Sensibility         18       1 advanced    positive
##  3 Sense & Sensibility         20       1 death       negative
##  4 Sense & Sensibility         21       1 loss        negative
##  5 Sense & Sensibility         25       1 comfortably positive
##  6 Sense & Sensibility         28       1 goodness    positive
##  7 Sense & Sensibility         28       1 solid       positive
##  8 Sense & Sensibility         29       1 comfort     positive
##  9 Sense & Sensibility         30       1 relish      positive
## 10 Sense & Sensibility         33       1 steady      positive
## # … with 44,161 more rows
```

Count the number of positive words and negative words for each chunk of text in each book (here we define a chunck as 80 lines of text):

```
tidy_books_bing %>% count(book, index = linenumber %/% 80, sentiment)
```

```
## # A tibble: 1,840 x 4
##    book                index sentiment     n
##    <fct>               <dbl> <chr>     <int>
##  1 Sense & Sensibility     0 negative     16
##  2 Sense & Sensibility     0 positive     26
##  3 Sense & Sensibility     1 negative     19
##  4 Sense & Sensibility     1 positive     44
##  5 Sense & Sensibility     2 negative     12
##  6 Sense & Sensibility     2 positive     23
##  7 Sense & Sensibility     3 negative     15
##  8 Sense & Sensibility     3 positive     22
##  9 Sense & Sensibility     4 negative     16
## 10 Sense & Sensibility     4 positive     29
## # … with 1,830 more rows
```

In the above code, we use `count()` to count the number of observations in each group (based by `book`, `index` and `sentiment`). `df %>% count(a, b, c)` is roughly equivalent to `df %>% group_by(a, b, c) %>% summarise(n = n())`.

Calculate the sentiment score as the difference between the number of positive words and negatvie words:

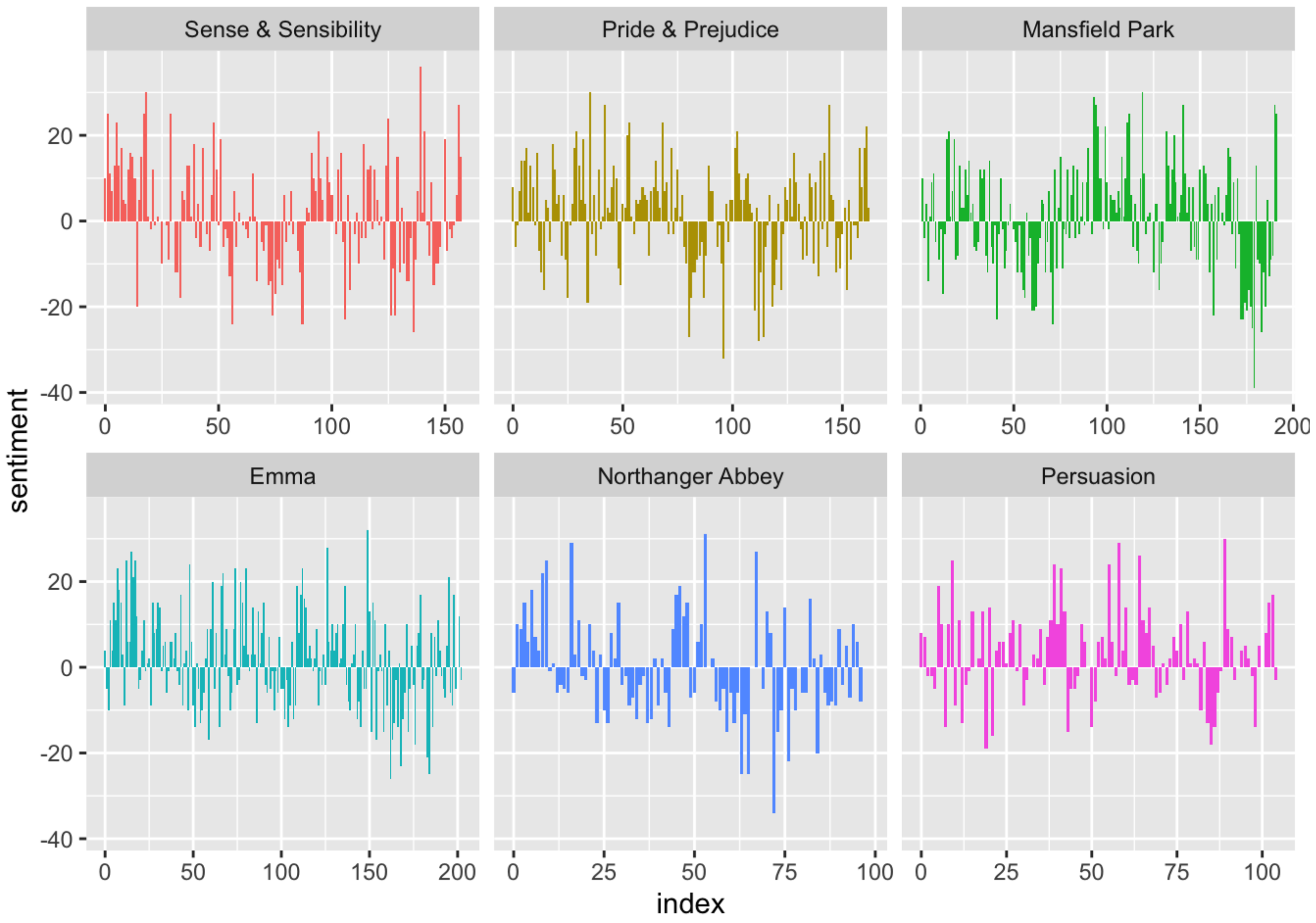
```
sentiment_bing <- tidy_books_bing %>% count(book, index = linenumber %/% 80, sentimen
t) %>%
pivot_wider(names_from = sentiment, values_from = n, values_fill = 0) %>% mutate(sent
iment = positive - negative)
sentiment_bing
```

```
## # A tibble: 920 x 5
##    book                index negative positive sentiment
##    <fct>               <dbl>    <int>    <int>     <int>
##  1 Sense & Sensibility     0       16       26        10
##  2 Sense & Sensibility     1       19       44        25
##  3 Sense & Sensibility     2       12       23        11
##  4 Sense & Sensibility     3       15       22         7
##  5 Sense & Sensibility     4       16       29        13
##  6 Sense & Sensibility     5       16       39        23
##  7 Sense & Sensibility     6       24       37        13
##  8 Sense & Sensibility     7       22       39        17
##  9 Sense & Sensibility     8       30       35         5
## 10 Sense & Sensibility     9       14       18         4
## # … with 910 more rows
```

Plotting the Results

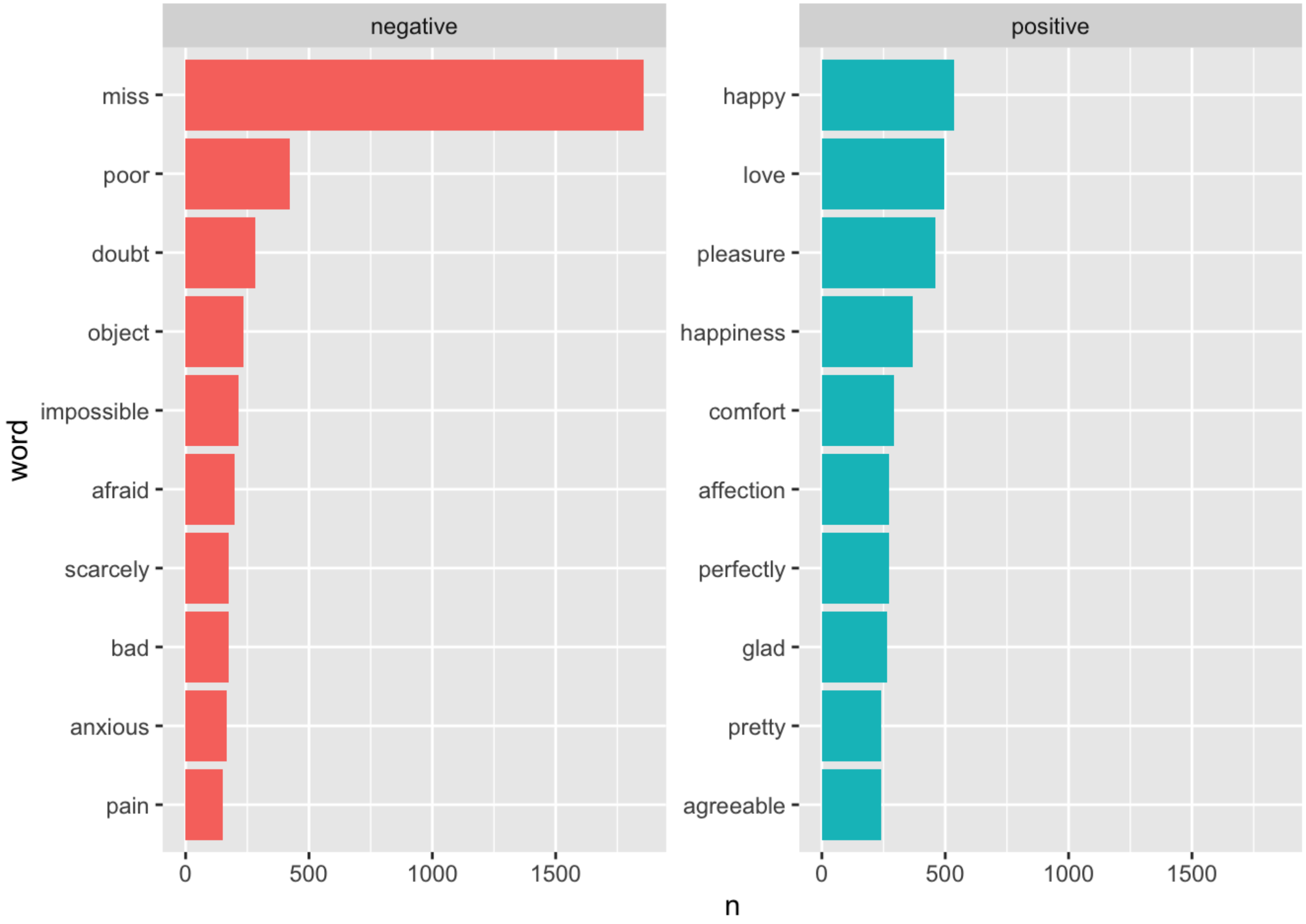
(1) Use bar plots to show the sentiment of each book from the start to the end. `geom_col()` in `ggplot2` creates a bar plot where the heights of the bars represent values in the data.

```
sentiment_bing %>% ggplot(aes(index, sentiment, fill = book)) + geom_col(show.legend
= FALSE) + facet_wrap(~ book, ncol = 3, scales = "free_x")
```



(2) Use bar plots to show the most common positive and negative words.

```
tidy_books_bing %>% count(word, sentiment) %>% ungroup() %>% group_by(sentiment) %>%
top_n(10, n) %>% ungroup() %>% mutate(word = reorder(word, n)) %>% ggplot(aes(word,
 n, fill = sentiment)) + geom_col(show.legend = FALSE) + facet_wrap(~ sentiment, scal
es = "free_y") + coord_flip()
```



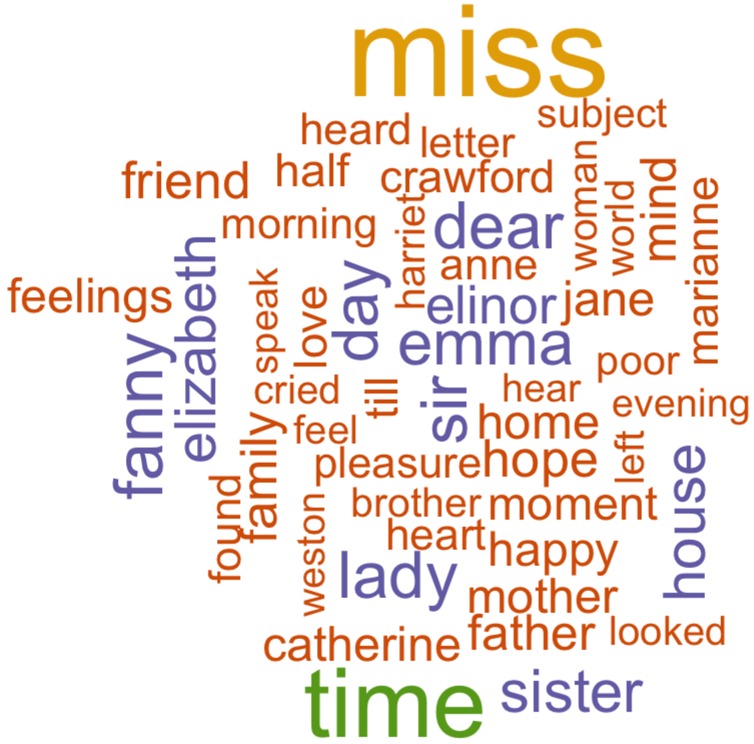
In the above code, `reorder()` reorders levels of `word` based on the values of `n`, so that in the resulting plot, the bars will be ordered.

(3) Use **wordclouds** to visualize word frequencies.

```
# install.packages("wordcloud")
library(wordcloud)
```

```
## Loading required package: RColorBrewer
```

```
tidy_books %>% count(word) %>% with(wordcloud(word, n, min.freq = 20, max.words = 50,
rot.per = 0.35, colors = brewer.pal(6, "Dark2")))
```



In the above code, the function `with()` evaluates an R expression in an environment constructed from data.

There are many functions that do not have a data argument (e.g., `wordcloud()`), so you have to retype the name of the data frame every time you reference a column. `with()` is a wrapper to let you use any function as if it had a data argument.

Usage of `wordcloud()`:

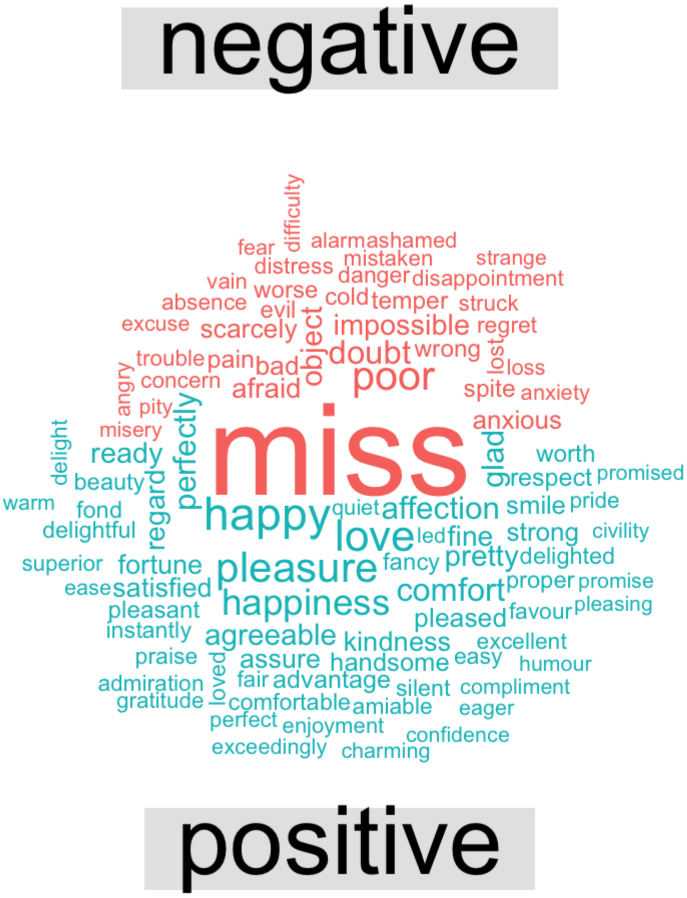
```
str(wordcloud)
```

```
## function (words, freq, scale = c(4, 0.5), min.freq = 3, max.words = Inf,
##     random.order = TRUE, random.color = FALSE, rot.per = 0.1, colors = "black",
##     ordered.colors = FALSE, use.r.layout = FALSE, fixed.asp = TRUE, ...)
```

- `words`: the words
- `freq`: their frequencies
- `min.freq`: words with frequency below `min.freq` will not be plotted.
- `max.words`: maximum number of words to be plotted. Least frequent terms are dropped.
- `rot.per`: proportion of words plotted with 90 degree rotation.
- `colors`: color words from least to most frequent.

We can also plot a cloud comparing the frequencies of positive words and negative words:

```
tidy_books_bing %>% count(word, sentiment) %>% ungroup() %>% pivot_wider(names_from =
sentiment, values_from = n, values_fill = 0) %>% column_to_rownames("word") %>% as.ma
trix() %>% comparison.cloud(colors = c("#F8766D", "#00BFC4"), max.words = 100)
```



Usage of `comparison.cloud()` :

```
str(comparison.cloud)
```

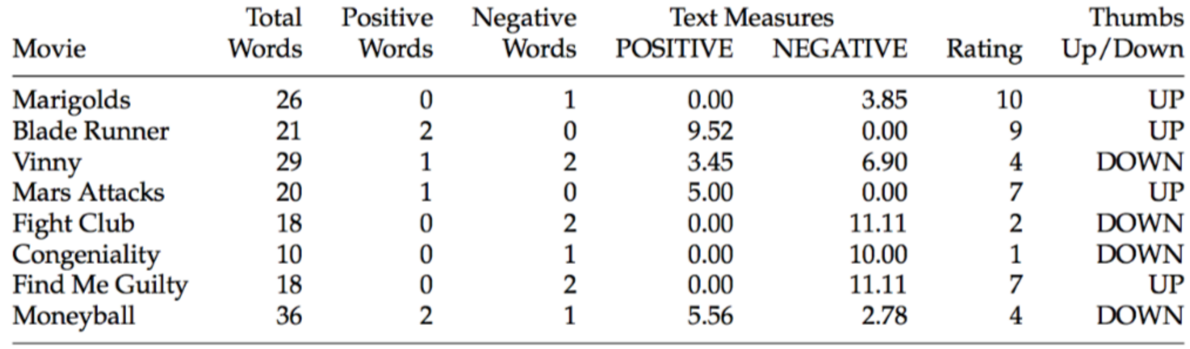
```
## function (term.matrix, scale = c(4, 0.5), max.words = 300, random.order = FALSE, 
##     rot.per = 0.1, colors = brewer.pal(max(3, ncol(term.matrix)), "Dark2"), 
##     use.r.layout = FALSE, title.size = 3, title.colors = NULL, match.colors = FALS
E, 
##     title.bg.colors = "grey90", ...)
```

- `comparison.cloud()` plots a cloud comparing the frequencies of words across documents.
- `term.matrix` : a term frequency matrix whose rows represent words and whose columns represent documents.

In the above code, it's like we create two "documents", one for positive words, and the other for negative words.

N-grams

So far, we have only considered words as text units. A lot of useful work can be done by tokenizing at the word level, but sometimes it is necessary to look at different units of text.



	Total	Positive	Negative	Text Measures			Thumbs
Movie	Words	Words	Words	POSITIVE	NEGATIVE	Rating	Up/Down
Marigolds	26	0	1	0.00	3.85	10	UP
Blade Runner	21	2	0	9.52	0.00	9	UP
Vinny	29	1	2	3.45	6.90	4	DOWN
Mars Attacks	20	1	0	5.00	0.00	7	UP
Fight Club	18	0	2	0.00	11.11	2	DOWN
Congeniality	10	0	1	0.00	10.00	1	DOWN
Find Me Guilty	18	0	2	0.00	11.11	7	UP
Moneyball	36	2	1	5.56	2.78	4	DOWN

Table 1. Unigram-Based Sentiment Scores

The above table shows the unigram-based sentiment scores of movie reviews as well as the corresponding numeric ratings. The sentiment scores (positive vs. negative) of some movies (Marigolds, Find Me Guilty and Moneyball) are inconsistent with their numeric ratings (favorable vs. unfavorable).

The main problem with unigram-based sentiment analysis is that it only focuses on the presence of a word, but ignores the *context* of the word.

The context of a word matters. For example, a positive word can be used in a sarcasm setting, while a negative word preceded by a negating word could mean positive emotions. There is nothing inherently good about the positive words or inherently bad about the negative words. It is context that gives them meaning.

Many interesting text analyses are based on the *relationships between words*. Some words tend to follow others immediately, and some words tend to co-occur within the same document.

For example, the word "unpredictable" has a negative meaning in the phrase "unpredictable steering" in an automotive review, but has a positive meaning in the phrase "unpredictable plot" in a movie review.

Capturing relationships between words require tokenizing by *sequences of adjacent words*, called **n-grams**.

`unnest_tokens()` can tokenize text into n-grams using the `token = "ngrams"` option and setting `n` to the number of words we wish to capture in each `n-gram`.

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

```
## # A tibble: 725,049 x 2
##    book                bigram
##    <fct>               <chr>
##  1 Sense & Sensibility sense and
##  2 Sense & Sensibility and sensibility
##  3 Sense & Sensibility sensibility by
##  4 Sense & Sensibility by jane
##  5 Sense & Sensibility jane austen
##  6 Sense & Sensibility austen 1811
##  7 Sense & Sensibility 1811 chapter
##  8 Sense & Sensibility chapter 1
##  9 Sense & Sensibility 1 the
## 10 Sense & Sensibility the family
## # … with 725,039 more rows
```

Remove bigrams that contain stop words, then count bigram frequencies:

```
bigrams_count <- austen_bigrams %>% separate(bigram, c("word1", "word2"), sep = " ")
 %>% filter(!word1 %in% stop_words$word) %>% filter(!word2 %in% stop_words$word) %>%
 unite(bigram, word1, word2, sep = " ") %>% count(book, bigram, sort = TRUE)
bigrams_count
```

```
## # A tibble: 36,217 x 3
##    book                bigram                n
##    <fct>               <chr>             <int>
##  1 Mansfield Park      sir thomas          287
##  2 Mansfield Park      miss crawford       215
##  3 Persuasion          captain wentworth   170
##  4 Emma                miss woodhouse      162
##  5 Emma                frank churchill     132
##  6 Persuasion          lady russell        118
##  7 Mansfield Park      lady bertram        114
##  8 Persuasion          sir walter          113
##  9 Emma                miss fairfax        109
## 10 Sense & Sensibility colonel brandon     108
## # … with 36,207 more rows
```

In the above code, to remove bigrams contaiming stop words, we use `separate()` to separate the column `bigram` into `word1` and `word2` , remove cases where either is a stop word, and use `unite()` to recombine them into one column. Refer to Topic 6 for the usage of `separate()` and `unite()` .

Negating Words

Examine how often sentiment-associated words are preceded by negating words:

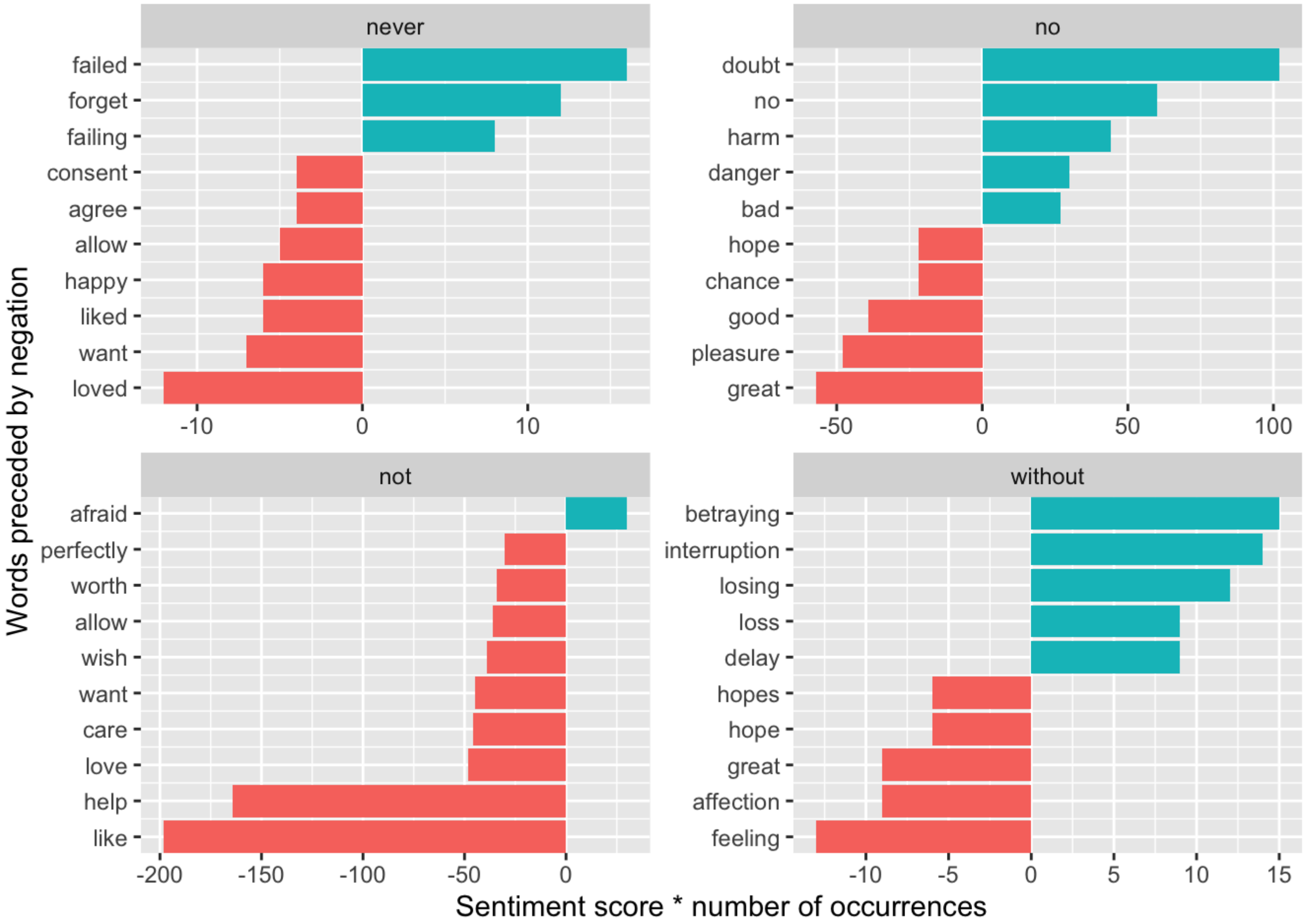
```
neg_bigrams <- austen_bigrams %>% separate(bigram, c("word1", "word2"), sep = " ") %
>% filter(word1 %in% c("not", "no", "never", "without")) %>% inner_join(get_sentiment
s("afinn"), by = c(word2 = "word")) %>% count(word1, word2, value, sort = TRUE) %>% u
ngroup() %>% mutate(value = - value) %>% mutate(contribution = n * value) %>% arrange
(desc(abs(contribution))) %>% split(.$word1) %>% map_dfr(~ head(.x, n = 10)) %>% arra
nge(word1, contribution) %>% mutate(order = row_number())

neg_bigrams
```

```
## # A tibble: 40 x 6
##    word1 word2   value     n contribution order
##    <chr> <chr>   <dbl> <int>        <dbl> <int>
##  1 never loved      -3     4          -12     1
##  2 never want       -1     7           -7     2
##  3 never liked      -2     3           -6     3
##  4 never happy      -3     2           -6     4
##  5 never allow      -1     5           -5     5
##  6 never agree      -1     4           -4     6
##  7 never consent    -2     2           -4     7
##  8 never failing     2     4            8     8
##  9 never forget      1    12           12     9
## 10 never failed      2     8           16    10
## # … with 30 more rows
```

Plotting:

```
neg_bigrams %>% ggplot(aes(order, contribution, fill = n * value > 0)) + geom_bar(sta
t = "identity", show.legend = FALSE) + facet_wrap(~word1, scales = "free") + xlab("Wo
rds preceded by negation") + ylab("Sentiment score * number of occurrences") + scale_
x_continuous(breaks = neg_bigrams$order, labels = neg_bigrams$word2, expand = c(0, 0
)) + coord_flip()
```


never
failed
forget
failing
consent
agree
allow
happy
liked
want
loved
no
doubt
no
harm
danger
bad
hope
chance
good
pleasure
great
not
afraid
perfectly
worth
allow
wish
want
care
love
help
like
without
betraying
interruption
losing
loss
delay
hopes
hope
great
affection
feeling
Words preceded by negation
Sentiment score * number of occurrences