Reading course notes

Haoluan Chen

9/11/2020

Chapter 1

1 The tidy text format

Tidy data structure:

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

Some definition:
Tidy text format - a table with one-token-per-row
Token - a meaningful unit of text (e.g a word that we are interested in analysis)
Tokenization - a process of splitting text into tokens

1.1 Contrasting tidy text with other data structures

Other text data structures:

- String: character vectors
- Corpus: contain raw strings annotated with additional metadata and details
- Document-term matrix: a sparse matrix describing a collection of documents with one row for each document and one column for each term. (e.g word count and tf-idf)

1.2 The unnest_tokens function

https://toronto.ctvnews.ca/ontario-s-covid-19-case-count-surges-to-highest-level-since-june-1.5100868

This is a news updating the current COVID-19 case count in Ontario

```
text <- c("TORONTO -- Ontario's daily COVID-19 case count has surpassed 200 for the first time in almos

text
```

```
## [1] "TORONTO -- Ontario's daily COVID-19 case count has surpassed 200 for the first time in almost tw
## [2] "On Friday, provincial health officials logged 213 new patients infected with the novel coronavi
```

This text is the first two paragraph in the article and we want to turn it into a tidy test dataset to do further analyze. First, we need to put it into a data frame

```
text_df <- tibble(line = 1:2, text = text)

text_df
```

```
## # A tibble: 2 x 2
##    line text
##   <int> <chr>
## 1     1 TORONTO -- Ontario's daily COVID-19 case count has surpassed 200 f~
## 2     2 On Friday, provincial health officials logged 213 new patients inf~
```

Note that this output is a tibble data fram, it has a convenient print method, will not convert strings to factors, and does not user row names.

Now, we can convert this data frame into tidy format using unnest_tokens() function

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

```
## # A tibble: 32 x 2
##     line word
##    <int> <chr>
##  1     1 toronto
##  2     1 ontario's
##  3     1 daily
##  4     1 covid
##  5     1 19
##  6     1 case
##  7     1 count
##  8     1 has
##  9     1 surpassed
## 10     1 200
## # ... with 22 more rows
```

unnest_tokens() takes in two arguments, the first one is the output column name (word in this case) and the second one is the input column (takes the text column in the text_df data frame)

From the output, we notice that:

- the line column represents the line number each word came from
- punctuation has been stripped
- converted to lower case (we can use to_lower = FALSE argument to turn off this behavior)

1.3 Tidying the works of Jane Austen (news in my case)

```
news <- c("TORONTO -- Ontario's daily COVID-19 case count has surpassed 200 for the first time in almos
```

```
news
```

```
##  [1] "TORONTO -- Ontario's daily COVID-19 case count has surpassed 200 for the first time in almost
##  [2] "On Friday, provincial health officials logged 213 new patients infected with the novel coronav
##  [3] "The other 26 local public health units in Ontario reported five or fewer COVID-19 cases on Fri
##  [4] "The last time Ontario saw a daily case count climb above 200 was on July 21 and the last time
##  [5] "The province's daily count had hovered above the 100 mark for majority of the past three weeks
##  [6] "Most recently, Ontario saw 170 new cases of the disease confirmed on Thursday, 149 on Wednesda
##  [7] "The new patients logged on Friday bring Ontario's total case count to 44,068, including deaths
##  [8] "There were no new deaths linked to the novel coronavirus recorded by the province on Friday, bu
##  [9] "Health officials deemed 124 more COVID-19 cases to be resolved in the province as of Friday. O
## [10] "There are now 1,657 active cases in the province."
## [11] "As of Friday, 49 COVID-19 patients are in Ontario hospitals. Eighteen of those patients are in
## [12] "Since the start of the pandemic, more than 3.2 million COVID-19 tests have been conducted in O
## [13] "In the last recorded 24-hour period, 32,501 tests were conducted."
## [14] "On July 21, nearly 23,000 COVID-19 tests were conducted and just more than 16,000 were conducte
## [15] "There are currently 31,384 tests under investigation in the province."
```

```
news_df <- tibble(line = 1:15, text = news)

news_df <- news_df %>%
  unnest_tokens(word, text)
```

Now we have tokenized the news, we can manipulate it with tidy tools. Often we want to revmoe stop words which are not userful for an analysis. We may remove the stop words by using anti_join

```
news_df <- anti_join(news_df, stop_words, by = "word")
```

Then, we can use count() to find the most common words in this news article

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

```
## # A tibble: 105 x 2
##    word          n
##    <chr>     <int>
##  1 19            6
##  2 covid         6
##  3 friday        6
##  4 patients      6
##  5 ontario       5
##  6 province      5
##  7 conducted     4
##  8 count         4
##  9 health        4
## 10 ontario's     4
## # ... with 95 more rows
```

Plot to show the most common words

```
plot1 <- news_df %>%
  count(word, sort = TRUE) %>%
  filter(n > 3) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word,n)) +
  geom_col() +
  xlab(NULL) +
  coord_flip()
```

Here is the most common words in the news article

1.4 The gutenbergr package

a package privides access to the data set used in the book. For me I will use another new article related to COVID-19. https://www.cp24.com/news/ontario-reports-decline-in-new-covid-19-cases-today-sixth-day-in-a-row-new-infectio
5056474?cache=yes%3FclipId%3D89531

1.5 Word frequencies

A common task in text mining is to look at word frequencies. In this section, I will compare the two news article above. The first article is a recent one, which the COVID cases is increasing. The second article in 1.4 was a news article in August 8, which the COVID case is decreasing.

```
news2 <- c("Ontario is reporting a decline in new COVID-19 cases today, marking the sixth day in a row

Provincial health officials say 70 new cases of the virus were confirmed today, down from the 88 reporte

Officials recorded 95 new cases on Thursday, 86 on Wednesday, and 91 on Tuesday, bringing the rolling f
Ontario Health Minister Christine Elliott confirmed Saturday that recoveries outpaced new cases once ag

According to the province's latest  epidemiological summary, 107 more cases are now considered to be re
```

```
More than 26,000 tests were completed over the past 24 hours, resulting in a case positivity rate of 0.1

'Locally, 29 of Ontario's 34 public health units are reporting five or fewer cases with 15 of them repo

Ottawa and Peel Region each reported 13 new infections and nine new cases were confirmed in Chatham-Ken

Toronto recorded the fewest number of new infections in the GTA with just one new case confirmed over t

Nearly 60 per cent of all new cases in the province were in people under the age of 40.

Elliott noted that hospitalizations and intensive care admissions both declined.

The latest data indicates that 53 people infected with COVID-19 are currently receiving treatment at On

One new death was reported today, bringing the total number of virus-related deaths in the province to 1

The total number of lab-confirmed cases of COVID-19 in Ontario now stands at 39,967 but only 1,052 are s
```

```
news2
```

```
## [1] "Ontario is reporting a decline in new COVID-19 cases today, marking the sixth day in a row new i
```

```
news2_df <- tibble(line = 1, text = news2)

news2_df <- news2_df %>%
  unnest_tokens(word, text)

news2_df <- news2_df %>%
  anti_join(stop_words)
```

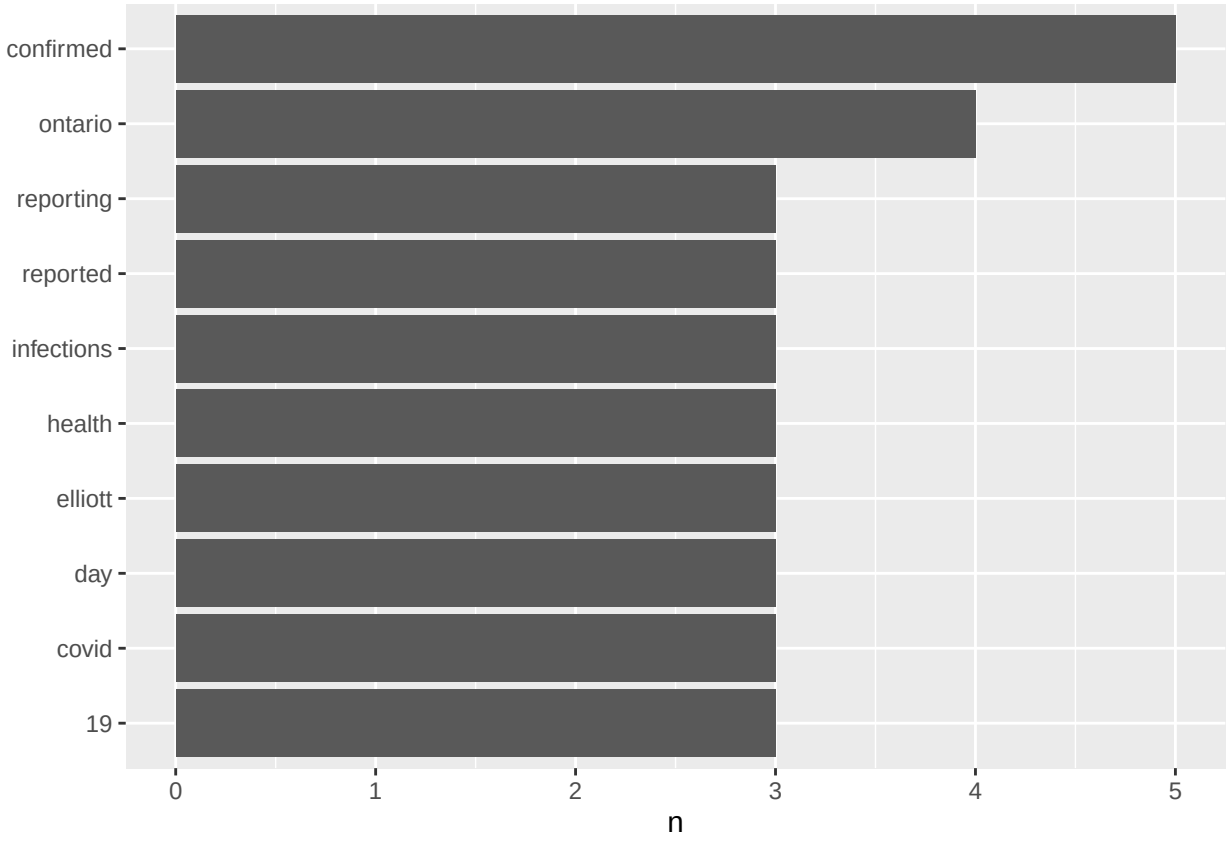
```
## Joining, by = "word"
```

```
news2_df %>% count(word, sort = TRUE)
```

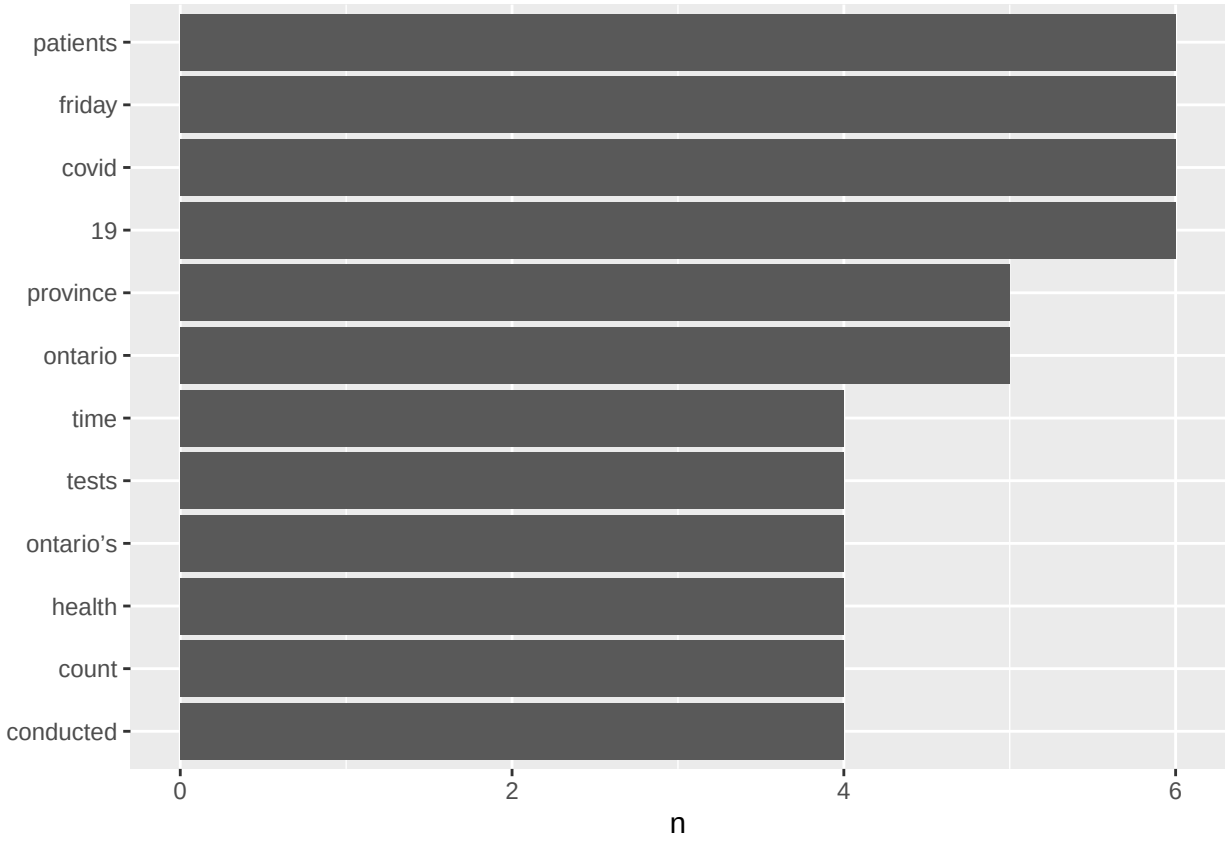
```
## # A tibble: 109 x 2
##    word           n
##    <chr>      <int>
##  1 confirmed      5
##  2 ontario        4
##  3 19             3
##  4 covid          3
##  5 day            3
##  6 elliott        3
##  7 health         3
##  8 infections     3
##  9 reported       3
## 10 reporting      3
## # ... with 99 more rows
```

```
news2_df %>%
  count(word, sort = TRUE) %>%
  filter(n > 2) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word,n)) +
  geom_col() +
```

```
  xlab(NULL) +
  coord_flip()
```




```
plot1
```



Lets compare the most common word between two article.

Note: I had some problem producing the plot in this seesion in comparing frequency, So I am comparing the most common words between two article.

1.6 Summary

In this chapter, I learning that when data is organized into tidy text data(one token per row), tasks like removing stop words and ploting are simple within the tidy tool ecosystem. The one-token-per-row framework may be extended from words to n-grams and other meaningful unit of text.

2 Sentiment analysis with tidy data

We can use the tools of text mining to approach the emotional conents of text programmatically. One way to analyze the sentiment of a text is to consider the text as a combinations of individual words and the sentiment content of the text is the sum of the sentiment content of each words.

2.1 The sentiments dataset

The following three package provides access to several sentiment lexicons(the vocabulary of a person, language, or branch of knowledge):

- AFINN (from AFINN from Nielsen)
- bing (from Bing Liu and collaborators)
- nrc (from Saif Mohammad and Peter Turney)

All these lexicons are based on unigrams(single words).

In the AFINN package, the lexicon assigns each words with a score between -5 to 5, where negatives socres indicating negative sentiment and postive socres means the postive sentiment.

In the bing package, the lexicon also categorizes words in binary fashion but only postive and negative categories.

In the nrc package, the lexicon categorizes words into categories(e.g postive, negative, anger, disgust, fear, sadness) in a binary fashion.

Note: cite them when using these sentiment lexicon,

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

```
## # 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
```

```
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("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
##  8 abandoned   sadness
```

```
##  9 abandonment anger
## 10 abandonment fear
## # ... with 13,891 more rows
```

These sentiment lexicons were constructed and validated by crowdsourcing, restauant or movie reviews, or Twitter data. We should be careful when applying these sentiment lexicons on the text that is very different from what they were validated on. There are also some domain-specific sentiment lexicons, which are constructed to be used with text from a specific content area.(e.g some sentiment lexicons are specifically for finance)

Something to keep in mind:

- Not every English words is in the lexicon, because many words are neutral.
- these methods do not take account qualifers before a word. For example, "no good" or "not true".
- The size of the text can have an effect on our analysis result. A text with many paragraphs can have both postive and negative sentiment and averaged out to be close to zero. Thus, sentence-sized or paragraph-sized text often works better.

2.2 Sentiment analysis with inner join

With tidy text data format, we can perform sentiment analysis by using inner join.

Using NRC lexicon, What are the most common sad words in our first article? (COVID-19 cases increasing)

```
nrc_negative <- get_sentiments("nrc") %>%
  filter(sentiment == "negative")

news_df %>%
  inner_join(nrc_negative) %>%
  count(word, sort = TRUE)
```

```
## Joining, by = "word"
```

```
## # A tibble: 5 x 2
##   word         n
##   <chr>    <int>
## 1 death        1
## 2 deceased     1
## 3 disease      1
## 4 pandemic     1
## 5 retract      1
```

Since we have a small data set, there are only four negative words in our first article.

```
news2_df %>%
  inner_join(nrc_negative) %>%
  count(word, sort = TRUE)
```

```
## Joining, by = "word"
```

```
## # A tibble: 4 x 2
##   word        n
##   <chr>   <int>
## 1 virus       2
## 2 death       1
## 3 decline     1
## 4 row         1
```

These are the negative words in the second article.

We can also find the sentiment score for each word in the article using Boing lexicon and inner_join

```
news_sentiment <- news_df %>%
  inner_join(get_sentiments("bing"))
```

```
## Joining, by = "word"
```

```
news_sentiment <- news_df %>%
  inner_join(get_sentiments("bing")) %>%
  count(index = line, sentiment) %>%
  spread(sentiment, n, fill = 0) %>%
  mutate(sentiment = positive - negative)
```

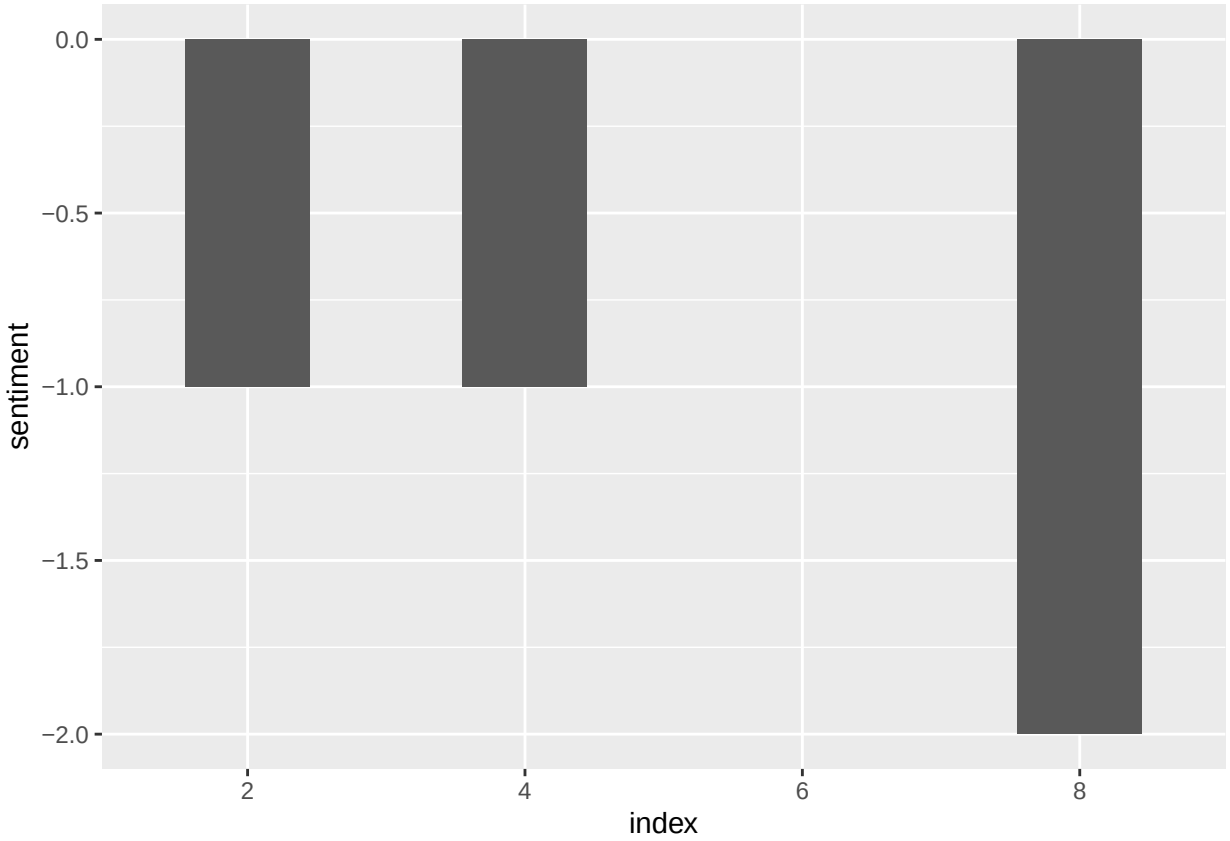
```
## Joining, by = "word"
```

```
news_sentiment
```

```
## # A tibble: 3 x 4
##   index negative positive sentiment
##   <int>    <dbl>    <dbl>     <dbl>
## 1     2        1        0        -1
## 2     4        1        0        -1
## 3     8        3        1        -2
```

```
ggplot(news_sentiment, aes(index, sentiment)) +
  geom_col(show.legend = FALSE)+
  scale_x_discrete(limits = c(0, 2, 4, 6, 8))
```

```
## Warning: Continuous limits supplied to discrete scale.
## Did you mean `limits = factor(...)` or `scale_*_continuous()`?
```



As the result shown, our article are likely to be negative. (Again we have a very small dataset)

2.3 Comparing the three sentiment dictionaries

Let's use all three sentiment lexicons on the article.

```
afinn <- news_df %>%
  inner_join(get_sentiments("afinn")) %>%
  group_by(index = line) %>%
  summarise(sentiment = sum(value)) %>%
  mutate(method = "AFINN")
```

```
## Joining, by = "word"
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
afinn
```

```
## # A tibble: 5 x 3
##   index sentiment method
##   <int>     <dbl> <chr>
## 1     2        -2 AFINN
## 2     8        -2 AFINN
## 3     9         2 AFINN
## 4    10         1 AFINN
## 5    11         2 AFINN
```

```
bing <- news_df %>%
  inner_join(get_sentiments("bing")) %>%
```

```
  mutate(method = "bing")
```

```
## Joining, by = "word"
```

```
bing
```

```
## # A tibble: 6 x 4
##    line word       sentiment method
##   <int> <chr>      <chr>     <chr>
## 1     2 infected   negative  bing
## 2     4 infections negative  bing
## 3     8 retract    negative  bing
## 4     8 patient    positive  bing
## 5     8 death      negative  bing
## 6     8 toll       negative  bing
```

```
nrc <- news_df %>%
  inner_join(get_sentiments("nrc")) %>%
  filter(sentiment %in% c("positive", "negative")) %>%
  mutate(method = "NRC")
```

```
## Joining, by = "word"
```

```
nrc
```

```
## # A tibble: 16 x 4
##     line word       sentiment method
##    <int> <chr>      <chr>     <chr>
##  1     1 count      positive  NRC
##  2     3 public     positive  NRC
##  3     4 count      positive  NRC
##  4     4 confirmed  positive  NRC
##  5     5 count      positive  NRC
##  6     5 majority   positive  NRC
##  7     6 disease    negative  NRC
##  8     6 confirmed  positive  NRC
##  9     7 count      positive  NRC
## 10     7 including  positive  NRC
## 11     8 retract    negative  NRC
## 12     8 deceased   negative  NRC
## 13     8 patient    positive  NRC
## 14     8 death      negative  NRC
## 15    11 assistance positive  NRC
## 16    12 pandemic   negative  NRC
```

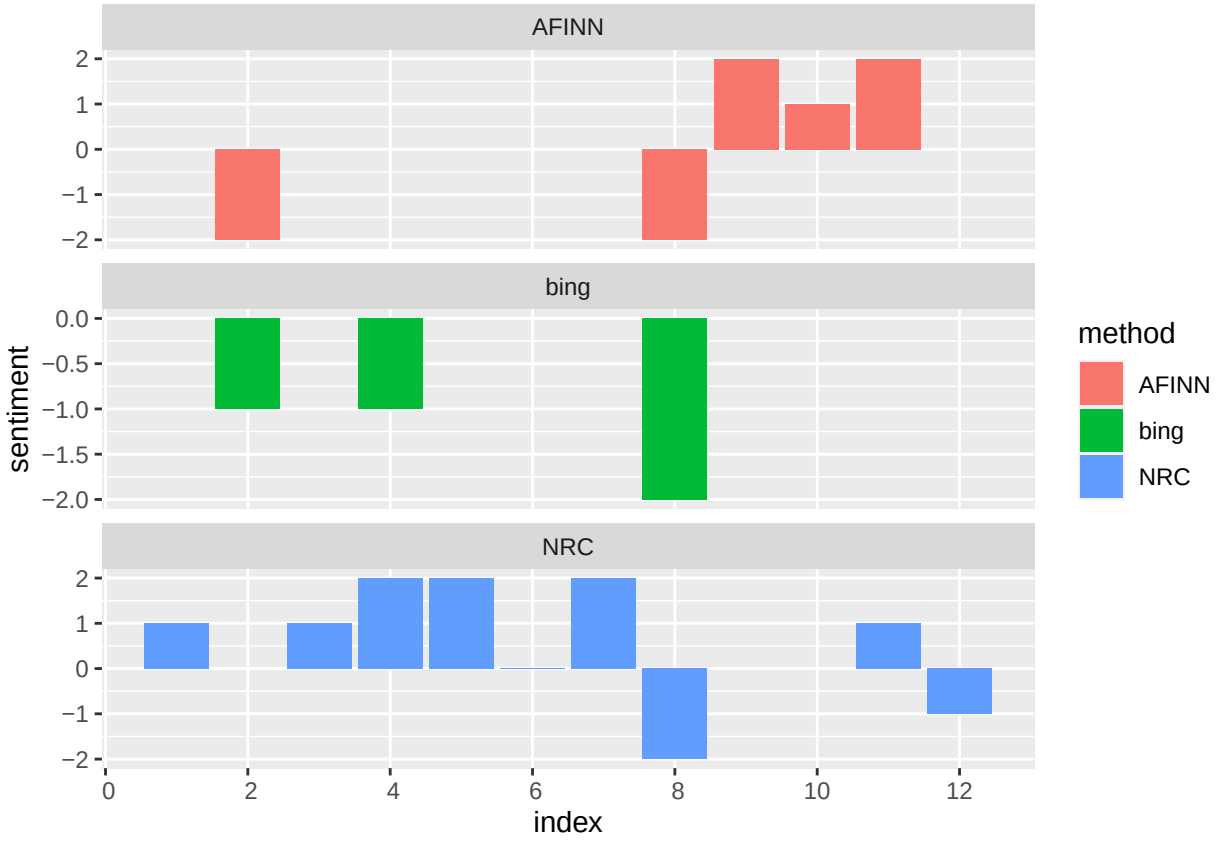
```
bing_and_nrc <- bind_rows(bing, nrc) %>%
  count(method, index = line, sentiment) %>%
  spread(sentiment, n, fill = 0) %>%
  mutate(sentiment = positive - negative)
bing_and_nrc
```

```
## # A tibble: 12 x 5
##    method index negative positive sentiment
##    <chr>  <int>    <dbl>    <dbl>     <dbl>
##  1 bing       2        1        0        -1
##  2 bing       4        1        0        -1
##  3 bing       8        3        1        -2
```

```
##  4 NRC        1        0        1         1
##  5 NRC        3        0        1         1
##  6 NRC        4        0        2         2
##  7 NRC        5        0        2         2
##  8 NRC        6        1        1         0
##  9 NRC        7        0        2         2
## 10 NRC        8        3        1        -2
## 11 NRC       11        0        1         1
## 12 NRC       12        1        0        -1
```

```
# plotting
bind_rows(afinn, bing_and_nrc) %>%
  ggplot(aes(index, sentiment, fill = method)) +
  geom_col() +
  facet_wrap(~method, ncol = 1, scales = "free_y") +
  scale_x_discrete(limits = c(0, 2, 4, 6, 8, 10, 12))
```

```
## Warning: Continuous limits supplied to discrete scale.
## Did you mean `limits = factor(...)` or `scale_*_continuous()`?
```



As we can see in the plot, three different lexicons for calculating sentiment in our article gives very different result due to the small data size.

However, according to the book, we see similar dips and peaks in sentiment at about the same places in the novel, but absolute values are significantly different. AFINN lexicon gives largest absolute value, Bing gives lower absolute values, and the NRC labled the text more positively.

"the NRC sentiment is high, the AFINN sentiment has more variance, the Bing et al. sentiment appears to

find longer stretches of similar text, but all three agree roughly on the overall trends in the sentiment through a narrative arc."

Why is NRC lexicon provided higher result compared to the Bing?

```
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   4781
## 2 positive   2005
```

As we can see, both lexicons ahve more negative words than postive words, but the ratio of negative to postive words is higher in the Bing than NRC. This is the reason that we see higher sentiment scores in the NRC.

Also, if the negative words in the NRC do not match the words we are analysing very well, then "whatever the source of these differences, we see similar relative trajectories across the narrative arc, with similar changes in slope, but marked differences in absolute sentiment from lexicon to lexicon."

This is important to keep in mind when choosing a sentiment lexicon for analysis.

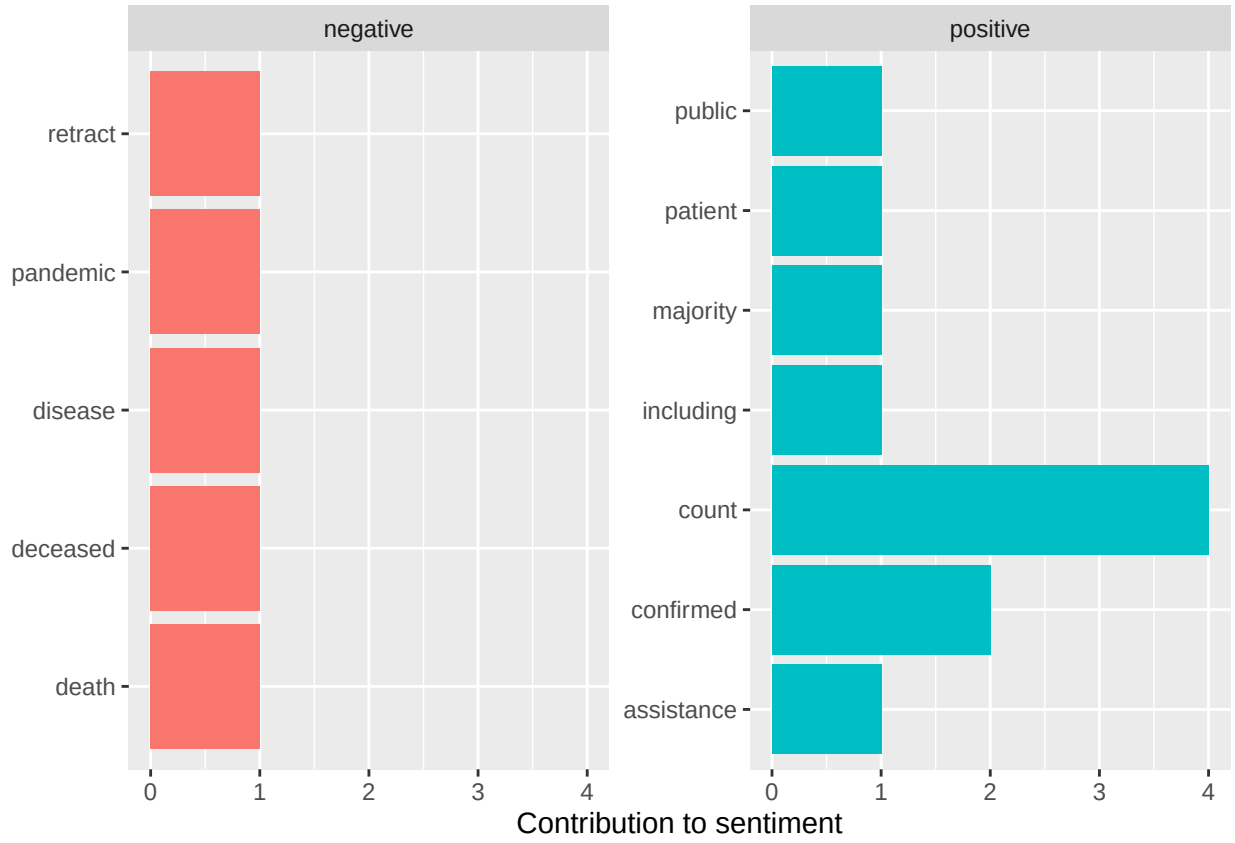
2.4 Most common positive and negative words

```
nrc_word_count <- nrc %>%
  count(word, sentiment, sort = TRUE)

nrc_word_count
```

```
## # A tibble: 12 x 3
##    word       sentiment     n
##    <chr>      <chr>     <int>
##  1 count      positive      4
##  2 confirmed  positive      2
##  3 assistance positive      1
##  4 death      negative      1
##  5 deceased   negative      1
##  6 disease    negative      1
##  7 including  positive      1
##  8 majority   positive      1
##  9 pandemic   negative      1
## 10 patient    positive      1
## 11 public     positive      1
## 12 retract    negative      1
```

```
#plot
```

```
nrc_word_count %>%
  group_by(sentiment) %>%
  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()
```



Problem: Since my article data set is small, we can spot problem in my analysis that patient and confirmed are negative in my case. How do we spot there kinds of problem when we have very big dataset?

2.5 Wordclouds

```
library(janeaustenr)
```

```
## Warning: package 'janeaustenr' was built under R version 3.6.3
```

```
library(stringr)

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

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

```
## Joining, by = "word"
```

```
## Warning in wordcloud(word, n, max.words = 100): elizabeth could not be fit
## on page. It will not be plotted.
```


```
library(reshape2)
```

```
##
## Attaching package: 'reshape2'
```

```
## The following object is masked from 'package:tidyr':
##
##     smiths
```

```
tidy_books %>%
  inner_join(get_sentiments("bing")) %>%
  count(word, sentiment, sort = TRUE) %>%
  acast(word ~ sentiment, value.var = "n", fill = 0) %>%
  comparison.cloud(colors = c("gray20", "gray80"),
                   max.words = 100)
```

```
## Joining, by = "word"
```

Note that since my data set is small, I am using the data set in the text book to create these wordcloud.

To tag positive and negative words, we will have to turn the datafram into matrix with reshape2 (acast()) function and use comparison.cloud().

The size of the word's test in the wordcloud is in proportion to its frequency in the document. We use word cloud to visually see the most frequent postive and negative words, but the size is not comparable across sentiments

2.6 Looking at units beyond just words

Sometimes it is useful to look at different unit of text. For example, try to understand the sentiment of a sentence as a whole. We can tokenize text into sentences.

```
news_sentences <- tibble(line = 1:15, text = news)

news_sentences <- news_sentences %>%
  unnest_tokens(sentence, text, token = "sentences")
news_sentences
```

```
## # A tibble: 19 x 2
##     line sentence
##    <int> <chr>
##  1     1 toronto -- ontario's daily covid-19 case count has surpassed 200 ~
##  2     2 on friday, provincial health officials logged 213 new patients in~
##  3     3 the other 26 local public health units in ontario reported five o~
##  4     4 the last time ontario saw a daily case count climb above 200 was ~
##  5     5 the province's daily count had hovered above the 100 mark for maj~
```

```
##  6     6 most recently, ontario saw 170 new cases of the disease confirmed~
##  7     6 the last time the province dipped into double digits was on aug. ~
##  8     6 20.
##  9     7 the new patients logged on friday bring ontario's total case coun~
## 10     8 there were no new deaths linked to the novel coronavirus recorded~
## 11     9 health officials deemed 124 more covid-19 cases to be resolved in~
## 12     9 ontario's total number of recovered patients is now 39,598.
## 13    10 there are now 1,657 active cases in the province.
## 14    11 as of friday, 49 covid-19 patients are in ontario hospitals.
## 15    11 eighteen of those patients are in the intensive care unit and nin~
## 16    12 since the start of the pandemic, more than 3.2 million covid-19 t~
## 17    13 in the last recorded 24-hour period, 32,501 tests were conducted.
## 18    14 on july 21, nearly 23,000 covid-19 tests were conducted and just ~
## 19    15 there are currently 31,384 tests under investigation in the provi~
```

We can use unnest_tokens() to split into tokens using a regex patter. It is not needed for my case.

Now, we can find the porportion of negative words in the article.

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


news_df %>%
  semi_join(bingnegative)%>%
  summarize(negativewords = n()) %>%
  mutate(ratio = negativewords/word_count)
```

```
## Joining, by = "word"
```

```
## # A tibble: 1 x 2
##   negativewords  ratio
##           <int>  <dbl>
## 1             5 0.0294
```

We can compare the negative wor ration with other documents, since it is normalized by the text length.

2.7 Summary

Sentiment analysis can help understand the attitudes and opnions expressed in text. "We can use sentiment analysis to understand how a narrative arc changes throughout a book or what words with emotional and opinon content are important for a particular text." We can also compare differnt emotional and opinion content between different documents.

3 Analyzing word and document frequency: tf-idf

term frequency(tf) is measure of how frequently a word occurs in a document.

inverse document frequency(idf) which decrease the weight for commonly used words and increases the weight for words that are ot used very often.

These two frequency can be combined to calucate tf-idf. So the frequency of a term is adjusted for how rarely it is used

tf-idf - "intended to measure how important a word is to a document in a collection (or corpus) of documents"

The inverse document frequency for any given term is defined as:

$$idf(term) = ln(\frac{n_{documents}}{n_{documentsContaingTerm}})$$

3.1 Term frequency in Jane Austen's novels

From pervious examples, I realized that my data set was too small for our model. I used more articles as my data set to run the examples blow.

https://www.cbc.ca/news/canada/world-canada-covid-19-sept-15-1.5724311 https://www.ctvnews.ca/health/coronavirus/canada-very-well-might-be-starting-its-second-wave-of-covid-19-doctor-says-1.5105105 https://ottawacitizen.com/news/local-news/covid-19-five-deaths-in-west-end-villa-outbreak-51-new-cases-in-ottawa-elliott-say

We want to examine the term frequency We first tokenized our data, and check the most common used words in our data.

```
news_word <- news_word %>%
  unnest_tokens(word, text) %>%
  count(article, word, sort = TRUE)
news_word
```

```
## # A tibble: 954 x 3
##    article word      n
##      <int> <chr> <int>
##  1       3 the      51
##  2       1 the      42
##  3       3 of       38
##  4       3 to       35
##  5       3 and      32
##  6       1 to       25
##  7       3 in       25
##  8       1 of       21
##  9       1 and      18
## 10       3 have     18
## # ... with 944 more rows
```

```
total_words <- news_word %>%
  group_by(article) %>%
  summarize(total = sum(n))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
total_words
```

```
## # A tibble: 3 x 2
##   article total
##     <int> <int>
## 1       1   728
## 2       2   144
## 3       3  1131
```

```
news_word <- left_join(news_word, total_words)
```

```
## Joining, by = "article"
```

```
news_word
```

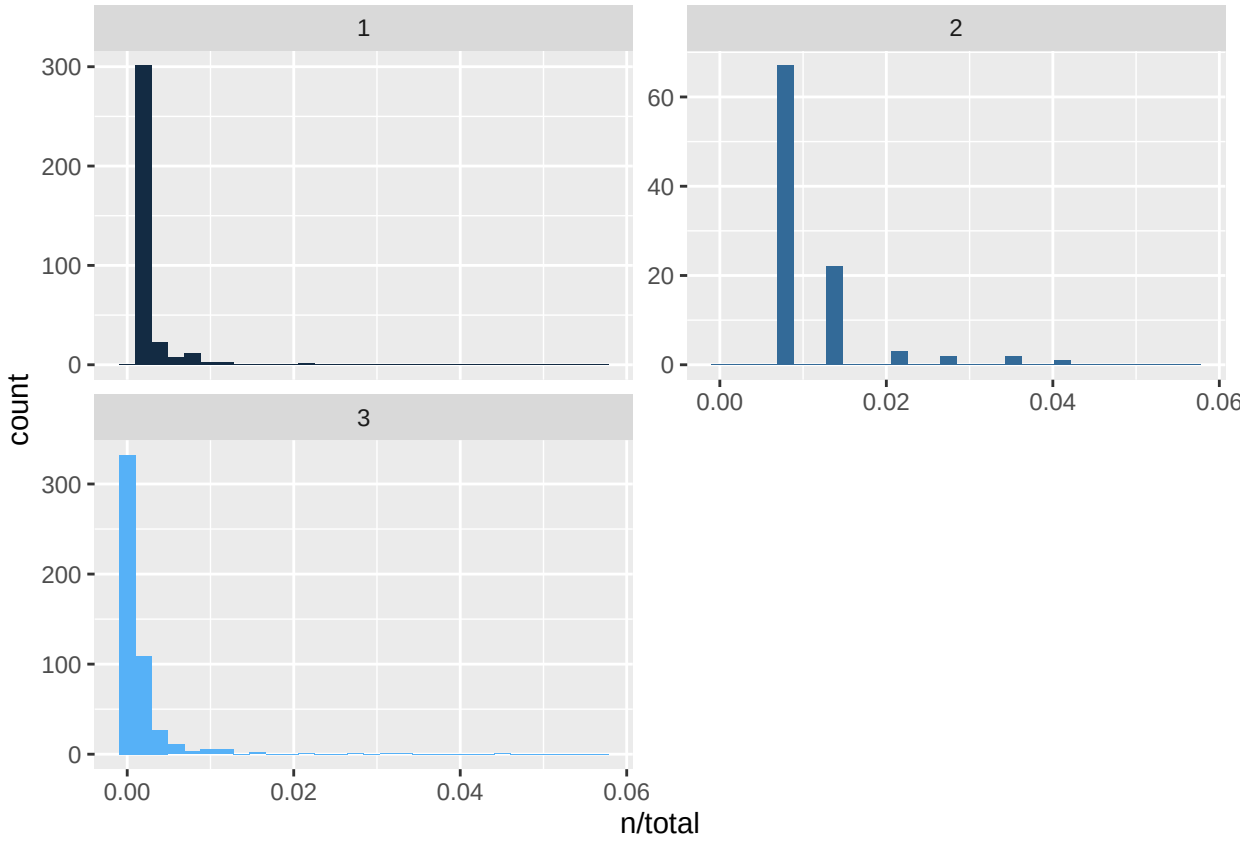
```
## # A tibble: 954 x 4
##    article word      n total
##      <int> <chr> <int> <int>
```

```
##  1       3 the      51  1131
##  2       1 the      42   728
##  3       3 of       38  1131
##  4       3 to       35  1131
##  5       3 and      32  1131
##  6       1 to       25   728
##  7       3 in       25  1131
##  8       1 of       21   728
##  9       1 and      18   728
## 10       3 have     18  1131
## # ... with 944 more rows
```

Here in the news_word data frame, we have each word_article combination per row. n is the number of occurance of that word in the article. Total is the total words in that article. We can look at the distributioin of n/total for each article.

```
ggplot(news_word, aes(n/total, fill = article)) +
  geom_histogram(show.legend = FALSE) +
   facet_wrap(~article, ncol = 2, scales = "free_y")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



All three article have very long tails to the right(extremely rare words). These plot exhibit similar distributions for our article and the book example used in our textbook, with many words that occur rarely and fewer words occur frequently.

3.2 Zipf's law

The distribution shown in 3.1 are typical in language. It is so common that given any corpus of natural language the relationship between the frequency that a word appears is inversely proportional to its rank. - This is the Zipf's law.

We can examine Zipf's law using our article data.

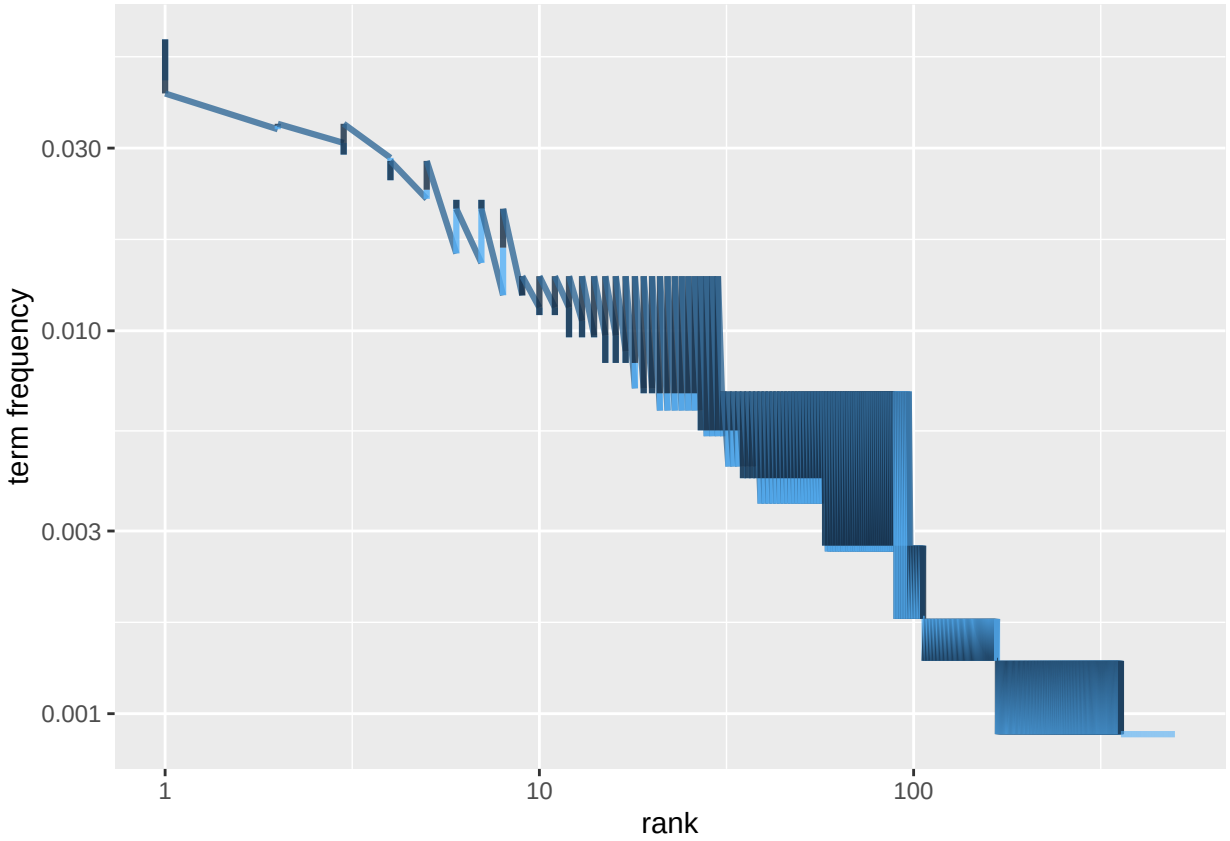
```
freq_by_rank <- news_word %>%
  group_by(article) %>%
  mutate(rank = row_number(),
         `term frequency` = n/total)
freq_by_rank
```

```
## # A tibble: 954 x 6
## # Groups:   article [3]
##    article word      n total  rank `term frequency`
##      <int> <chr> <int> <int> <int>            <dbl>
##  1       3 the      51  1131     1           0.0451
##  2       1 the      42   728     1           0.0577
##  3       3 of       38  1131     2           0.0336
##  4       3 to       35  1131     3           0.0309
##  5       3 and      32  1131     4           0.0283
##  6       1 to       25   728     2           0.0343
##  7       3 in       25  1131     5           0.0221
##  8       1 of       21   728     3           0.0288
##  9       1 and      18   728     4           0.0247
## 10       3 have     18  1131     6           0.0159
## # ... with 944 more rows
```

The rank column represents the rank of each word in each article.

Zipf's law is often visualized by plotting rank on the x-axis and term frequency on the y-axis, on logarithmic scales. Plotting this way, an inversely proportional relationship will have a constant, negative slope.

```
freq_by_rank %>%
  ggplot(aes(rank, `term frequency`, color = article)) +
  geom_line(size = 1.1, alpha = 0.8, show.legend = FALSE) +
  scale_x_log10() +
  scale_y_log10()
```

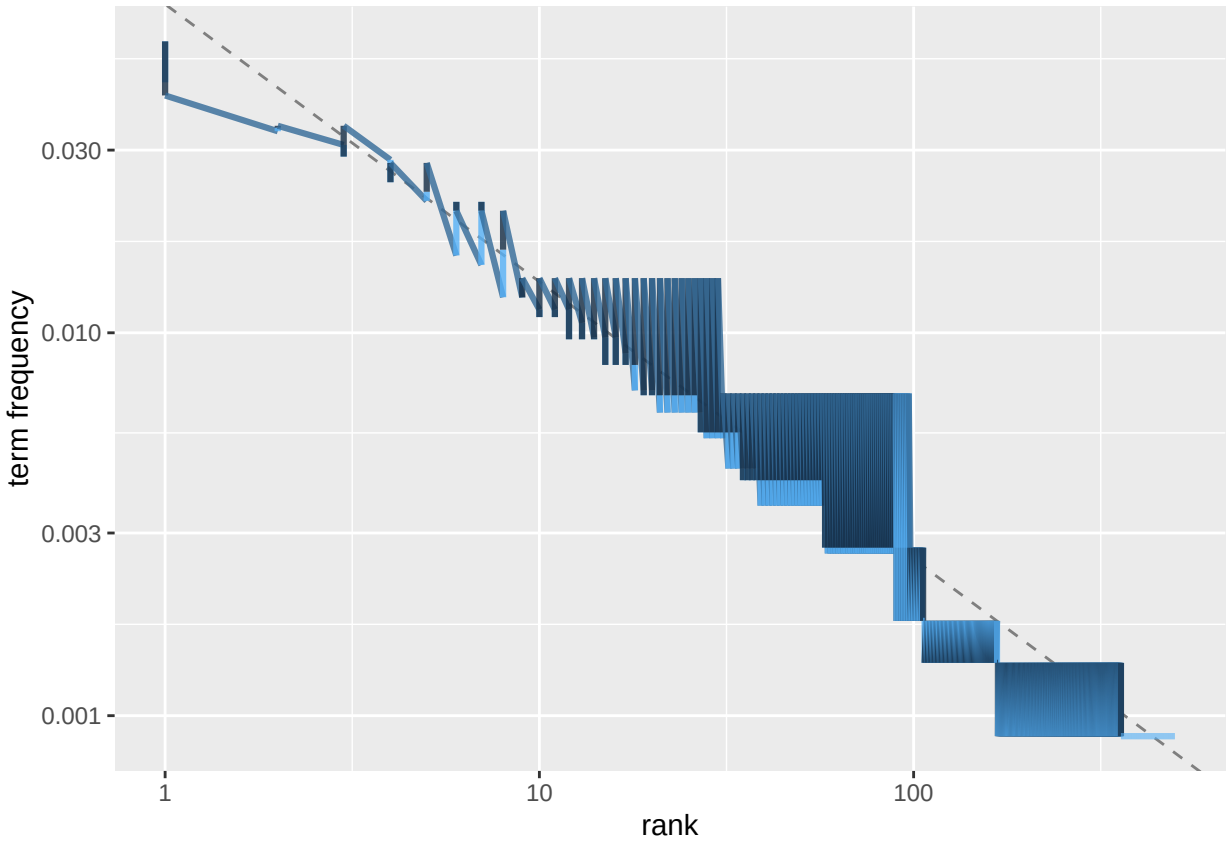


We see that the relationship between rank and frequency does have a negative slope

```
rank_subset <- freq_by_rank %>%
  filter(rank < 80,
         rank > 20)

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

```
##
## Call:
## lm(formula = log10(`term frequency`) ~ log10(rank), data = rank_subset)
##
## Coefficients:
## (Intercept)  log10(rank)
##     -1.1816      -0.6818
```

```
freq_by_rank %>%
  ggplot(aes(rank, `term frequency`, color = article)) +
  geom_abline(intercept = -1.1419, slope = -0.7247, color = "gray50", linetype = 2) +
  geom_line(size = 1.1, alpha = 0.8, show.legend = FALSE) +
  scale_x_log10() +
  scale_y_log10()
```



3.3 The bind_tf_idf function

The idea of tf-idf is to find the important words in our document by decreasing the weight for common words and increase the weight for rare words. Calculating tf-idf to find important words but not too common.

```
news_word <- news_word %>%
  bind_tf_idf(word, article, n)
```

Notice that idf and tf-idf are zeon for extremely common words. The inverse document frequency will be higher for words that occur in fewer of the documents in the collections. In opposite, the inverse document frequency and tf-idf is very low (close to zero) for words that occur in many of the articles.

Now, let's look at a visualization for the ti-idf words in our article

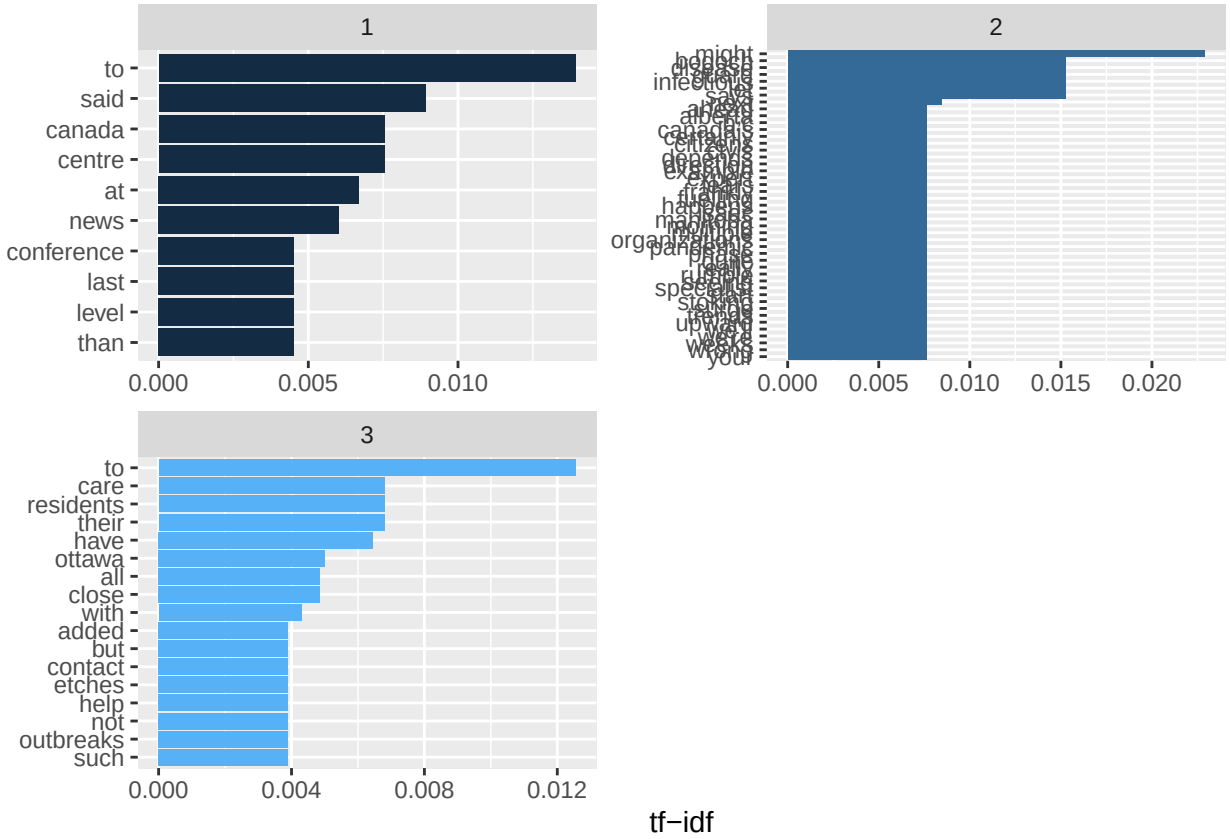
```
news_word <-news_word%>%
  arrange(desc(tf_idf)) %>%
  mutate(word = factor(word, levels = rev(unique(word))))
news_word
```

```
## # A tibble: 954 x 7
##    article word           n total     tf   idf  tf_idf
##      <int> <fct>      <int> <int>  <dbl> <dbl>   <dbl>
##  1       2 might          3   144 0.0208 1.10  0.0229
##  2       2 bogoch         2   144 0.0139 1.10  0.0153
##  3       2 disease        2   144 0.0139 1.10  0.0153
##  4       2 guard          2   144 0.0139 1.10  0.0153
##  5       2 infectious     2   144 0.0139 1.10  0.0153
##  6       2 let            2   144 0.0139 1.10  0.0153
```

```
##  7       2 says           2   144 0.0139 1.10  0.0153
##  8       1 to            25   728 0.0343 0.405 0.0139
##  9       3 to            35  1131 0.0309 0.405 0.0125
## 10       1 said          16   728 0.0220 0.405 0.00891
## # ... with 944 more rows
```

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

```
## Selecting by tf_idf
```



3.4 A corpus of physics texts

In the book, they analysed some classic physics from package Project Gutenberg using tf-idf. For me I will review the steps for tf-idf analysis.

- use unnest_tokens() to tokenzie each words in our data
- use count() to find how many times each word was used
- use bind_tf_idf() to calculate tf-idf (considering all text have different length)

- use ggplot() to create visualizations showing the highest tf-df words in the data
- check our result, check for names and some less meaningful words.
- remove the less meaningful words by using a custom list of stop words and use anti_join to remove them.
- recalculate the tf-idf using bind_tf_idf() and remake the plot!

3.5 summary

Using tf-idf allows us to find words that are special to one document within a collection of documents and how different words are important in documents within a collection or corpus of documents.

4 Ralationships between words: n-grams and correlations

So far we working on word as individual units and considered their relationship to sentiments or to documents. However, we can also analyses based on the relationshops between words. Examining which words tend to follow others immediately, or which words tend to co-occur within the same documents.

4.1 Tokenizing by n-gram

We can use unnest_tokens() to tokenize text into consecutive sequences of words, called n-grams. By seeing how often word X is followed by word Y, we can build a model of the relationship between them. We can do it by ading token = "ngrams" when using unnest_tokens(), and setting n to the number of words we want to capture in each n-gram. (when n = 2, examining pairs of two consecutive words, also called "bigrams")

```
news_word <- tibble(article = 1:3, text = newsdata)
news_bigrams <- news_word %>%
  unnest_tokens(bigram, text, token = "ngrams", n = 2)
news_bigrams
```

```
## # A tibble: 2,000 x 2
##    article bigram
##      <int> <chr>
##  1       1 some provinces
##  2       1 provinces are
##  3       1 are scrambling
##  4       1 scrambling to
##  5       1 to increase
##  6       1 increase testing
##  7       1 testing capacity
##  8       1 capacity as
##  9       1 as coronavirus
## 10       1 coronavirus infections
## # ... with 1,990 more rows
```

Note that this data structure is still in tidy text format(one-token-per-row), but each token now represents a bigram.

4.1.1 Counting and filtering n-grams

We can see the most common bigrams using count()

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

```
## # A tibble: 1,715 x 2
##    bigram            n
```

```
##    <chr>         <int>
##  1 covid 19         18
##  2 in the           10
##  3 of covid          7
##  4 of the            7
##  5 that the          7
##  6 on tuesday        6
##  7 the province      6
##  8 and the           5
##  9 at the            5
## 10 ottawa public     5
## # ... with 1,705 more rows
```

A lot of most common bigrams are pairs of common words (of the, to the) these may be uninteresting. So, we can use separate() function to splot a column into multiple based on a delimiter. We can split the bigrams into "word1" and "word2", then we can remove the stop words.

```
bigrams_separated <- news_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)

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

```
## # A tibble: 1,715 x 2
##    bigram            n
##    <chr>         <int>
##  1 covid 19         18
##  2 in the           10
##  3 of covid          7
##  4 of the            7
##  5 that the          7
##  6 on tuesday        6
##  7 the province      6
##  8 and the           5
##  9 at the            5
## 10 ottawa public     5
## # ... with 1,705 more rows
```

```
bigram_counts
```

```
## # A tibble: 376 x 3
##    word1      word2          n
##    <chr>      <chr>      <int>
##  1 covid      19            18
##  2 ottawa     public         5
##  3 public     health         5
##  4 assessment centre         3
##  5 brewer     arena          3
```

```
##  6 close      contacts       3
##  7 doug       ford           3
##  8 news       conference     3
##  9 ontario    premier        3
## 10 premier    doug           3
## # ... with 366 more rows
```

Then, we can recombine the words into bigrams by using unite()

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

bigrams_united
```

```
## # A tibble: 435 x 2
##    article bigram
##      <int> <chr>
##  1       1 increase testing
##  2       1 testing capacity
##  3       1 coronavirus infections
##  4       1 infections spike
##  5       1 covid 19
##  6       1 19 testing
##  7       1 testing sites
##  8       1 significant influx
##  9       1 accommodate demand
## 10       1 ottawa assessment
## # ... with 425 more rows
```

In some cases, we may be interested in the most common trigrams (n = 3), which are consecutive sequences of 3 words.

4.1.2 Analyzing bigrams

A bigram can also be treated as a term in a document. Thus, we can look at the tf-idf of the bigrams in the articles.

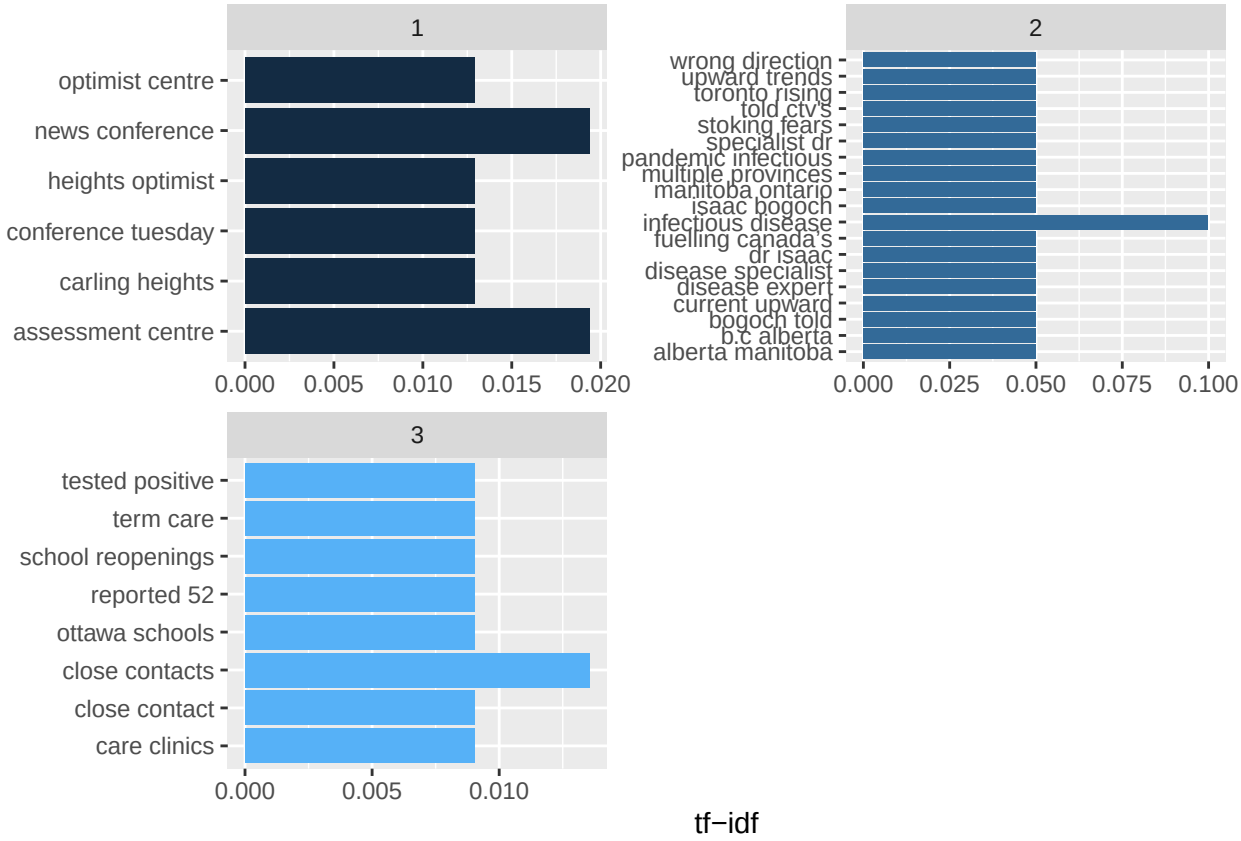
```
bigram_tf_idf <- bigrams_united %>%
  count(article, bigram) %>%
  bind_tf_idf(bigram, article, n) %>%
  arrange(desc(tf_idf))

bigram_tf_idf
```

```
## # A tibble: 391 x 6
##    article bigram                n     tf   idf tf_idf
##      <int> <chr>             <int>  <dbl> <dbl>  <dbl>
##  1       2 infectious disease     2 0.0909  1.10 0.0999
##  2       2 alberta manitoba       1 0.0455  1.10 0.0499
##  3       2 b.c alberta            1 0.0455  1.10 0.0499
##  4       2 bogoch told            1 0.0455  1.10 0.0499
##  5       2 current upward         1 0.0455  1.10 0.0499
##  6       2 disease expert         1 0.0455  1.10 0.0499
##  7       2 disease specialist     1 0.0455  1.10 0.0499
##  8       2 dr isaac               1 0.0455  1.10 0.0499
##  9       2 fuelling canada's      1 0.0455  1.10 0.0499
## 10       2 isaac bogoch           1 0.0455  1.10 0.0499
```

```
## # ... with 381 more rows
```

```
bigram_tf_idf %>%
  group_by(article) %>%
  top_n(3) %>%
  ungroup() %>%
  ggplot(aes(bigram, tf_idf, fill = article)) +
  geom_col(show.legend = FALSE) +
  labs(x = NULL, y = "tf-idf") +
  facet_wrap(~article, ncol = 2, scales = "free") +
  coord_flip()
```

```
## Selecting by tf_idf
```



The advantage to examining the tf-idf of bigrams is that it may capture structure that isn't present when counting single words, and may provide context more understandable. However, the disadvantages are that a typical two-word pair is rarer than single word. Thus, bigrams can be especially useful when we have large dataset. In our case, it is not very useful to use bigram to analysis our article.

4.1.3 Using bigrams to provide context in sentiment analysis

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

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

```
## # A tibble: 4 x 3
##   word1 word2          n
##   <chr> <chr>      <int>
## 1 not   businesses     1
## 2 not   going          1
## 3 not   rule           1
## 4 not   staff          1
```

By performing sentiment analysis on bigram, we can see how often sentiment-associated words are preceded by "not" or other negating words. We may use this to ignore or reverse the sentiment socre.

Let's use AFINN for sentiment analysis.

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

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

not_words
```

```
## # A tibble: 0 x 3
## # ... with 3 variables: word2 <chr>, value <dbl>, n <int>
```

unfortunately, because of our small data set, there are only 6 bigrams that contains "not", and the second word following the "not" in these 6 bigrams does not have a sentiment score in AFINN data set.

We can show which words contributed the most in the "wrong direction" in our sentiment analysis by computing the sentiment score and the number of times they appear. Then we can visualized the result with a bar plot.

Looking at the bar plot, we can see the largest causes of misidentification(words that make the text seem much more positive/negative than it is)

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

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

```
## # A tibble: 0 x 4
## # ... with 4 variables: word1 <chr>, word2 <chr>, value <dbl>, n <int>
```

We could then visualize what the most common words to follow each particular negation are. However, again our data set was too small to do it.

4.1.4 Visualizing a network of bigrams with ggraph

We can arrange the words into a network to visualize the relationship among words. It can be constructed from a tidy object with 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

We can use graph_from_data_frame() to produce an igraph object. It takes a data frame of edges with columns for "from", "to", and edge(in this case n)

```
bigram_graph <- bigram_counts %>%
  filter(n >1) %>%
  graph_from_data_frame()

bigram_graph
```

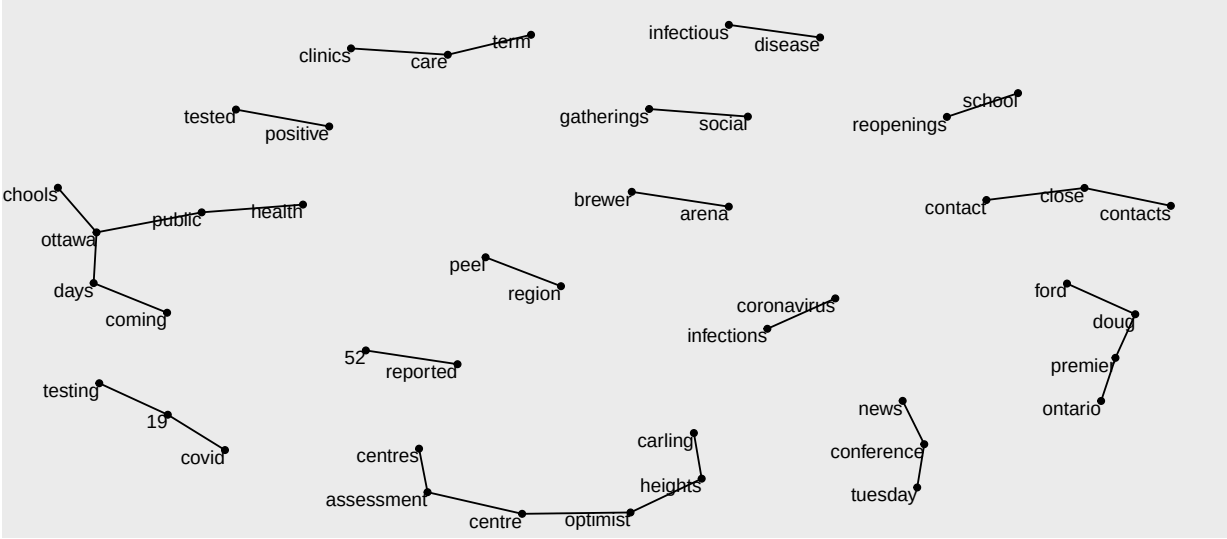
```
## IGRAPH 58b9f28 DN-- 44 29 --
## + attr: name (v/c), n (e/n)
## + edges from 58b9f28 (vertex names):
##  [1] covid      ->19         ottawa     ->public
##  [3] public     ->health     assessment ->centre
##  [5] brewer     ->arena      close      ->contacts
##  [7] doug       ->ford       news       ->conference
##  [9] ontario    ->premier    premier    ->doug
## [11] social     ->gatherings 19         ->testing
## [13] assessment ->centres    care       ->clinics
## [15] carling    ->heights    close      ->contact
## + ... omitted several edges
```

Then, we can convert an igraph object into a ggraph to create the graph.

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

```
## Warning: package 'ggraph' was built under R version 3.6.3
```

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



This graph shows the common bigrams that occurred more than once in our articles.

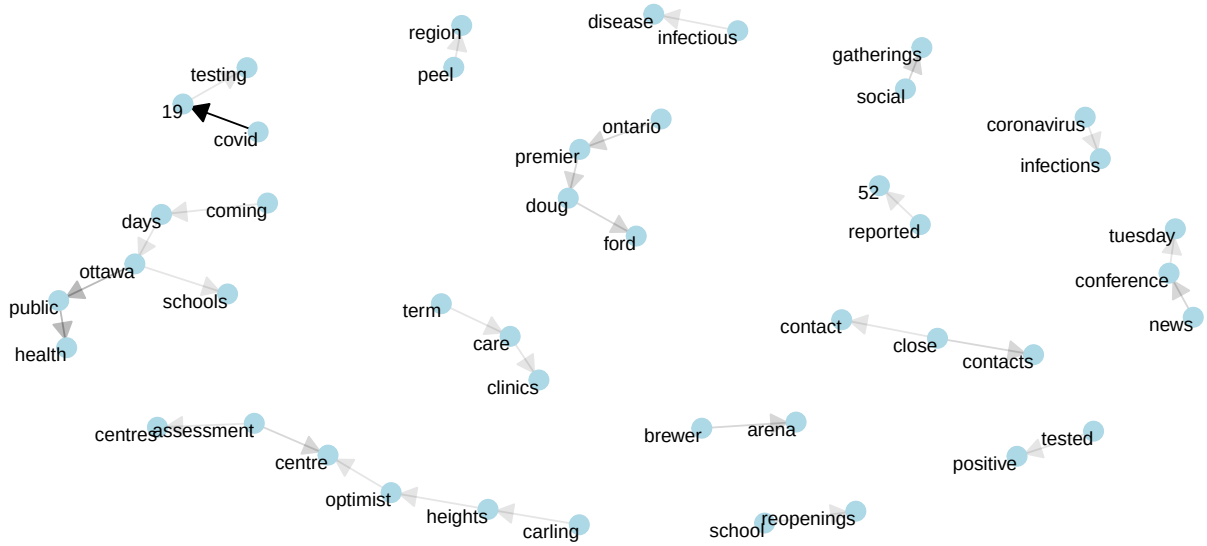
Something we can do to make a better looking graph:

- add edge_alpha aesthetic to the link layer to make links transparent based on how common or rare the bigram is.
- add directional arrow by using grid::arrow(), including an end_cap option that tells the arrow to end

before touching the node.

- add a theme using theme_void()

```
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()
```



"Note that this is a visualization of a Markov chain, a common model in text processing. In a Markov chain, each choice of word depends only on the previous word."

4.1.5 Visualizing bigrams in other texts

We can write a function for clearning and visualizing bigrams on any text dataset.

```
library(dplyr)
library(tidyr)
library(tidytext)
library(ggplot2)
library(igraph)
library(ggraph)
count_bigrams <- function(dataset) {
  dataset %>%
    unnest_tokens(bigram, text, token = "ngrams", n = 2) %>%
    separate(bigram, c("word1", "word2"), sep = " ") %>%
    filter(!word1 %in% stop_words$word,
           !word2 %in% stop_words$word) %>%
    count(word1, word2, sort = TRUE)
}

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

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

Using these count_bigrams and visualize_bigram function, we can easily perform data cleaning and visualizing bigrams on other text dataset.

4.2 Counting and correlating pairs of words with the widyr package

Tokenizing by n-gram is a useful way to explore pairs of adjacent words. However, we may be interested in words that tend to co-occur within particular documents, even if they are not next to each other. It is easy for tidy data to compare between variables or grouping by rows, but it is hard to compare between rows. For example, to count the number of times that two words appear within the same document, or to see how correlated they are. We may use widyr package manipulate the data that can compare between rows.

Counting and correlating among sections

```
news_word <- tibble(article = 1:3, text = newsdata)
news_word <- news_word %>%
  unnest_tokens(word, text) %>%
  filter(!word %in% stop_words$word)

news_word
```

```
## # A tibble: 972 x 2
##    article word
##      <int> <chr>
##  1       1 provinces
##  2       1 scrambling
##  3       1 increase
##  4       1 testing
##  5       1 capacity
##  6       1 coronavirus
##  7       1 infections
##  8       1 spike
##  9       1 canada
## 10       1 lineups
## # ... with 962 more rows
```

```
library(widyr)
```

```
## Warning: package 'widyr' was built under R version 3.6.3
```

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

```
## Warning: `distinct_()` is deprecated as of dplyr 0.7.0.
## Please use `distinct()` instead.
## See vignette('programming') for more help
```

```
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
## Warning: `tbl_df()` is deprecated as of dplyr 1.0.0.
## Please use `tibble::as_tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
word_pairs
```

```
## # A tibble: 181,866 x 3
##    item1      item2          n
##    <chr>      <chr>      <dbl>
##  1 covid      infections     3
##  2 19         infections     3
##  3 ontario    infections     3
##  4 tuesday    infections     3
##  5 toronto    infections     3
##  6 businesses infections     3
##  7 dr         infections     3
##  8 current    infections     3
##  9 infections covid          3
## 10 19         covid          3
## # ... with 181,856 more rows
```

The result from the pairwise_count will count the common pairs of words co-appearing within our the same article

Now, we can see the words that most often occur with "Canada" in our article

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

```
## # A tibble: 258 x 3
##    item1  item2           n
##    <chr>  <chr>       <dbl>
##  1 canada provinces       1
##  2 canada scrambling      1
##  3 canada increase        1
##  4 canada testing         1
##  5 canada capacity        1
##  6 canada coronavirus     1
##  7 canada infections      1
##  8 canada spike           1
##  9 canada lineups         1
## 10 canada covid           1
## # ... with 248 more rows
```

4.2.2 Pairwise correlation

We may also examine correlation among words, which indicates how often they appear together relative to how often they appear separately.

phi coefficient - a common measure for binary correlation

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

where

- n_{11} is the number of times both words appear in the document
- n_{00} is the number of times neither words appear
- n_{10} is the number of times first word appear and second word does not
- n_{01} is the number of times second word appear and first word does not
- $n_{1.}$ is the total number of times first word appear
- $n_{0.}$ is the total number of times first word does not appear
- $n_{.1}$ is the total number of times second word appear
- $n_{1.}$ is the total number of times second word does not appear
- n is the total number of words

The pairwise_cor() function in widyr lets us find the phi coefficient between words based on how often they appear in the same section.

```
word_cors <- news_word %>%
  group_by(word) %>%
  filter(n() >= 5) %>%
  pairwise_cor(word, article, sort = TRUE)


word_cors %>%
  filter(item2 == "wildfires")
```

```
## # A tibble: 0 x 3
## # ... with 3 variables: item1 <chr>, item2 <chr>, correlation <dbl>
```

Because our data set is too small, the correlation is not very interesting. :(

If we have bigger data set, we may use a bar plot to plot the correlation.

4.3 Summary

This chapter, we found relationship and connections between words in a form of n-grams. It enable us to see what words tend to appear after others, or co-occurrences and correlations. I also learn about network visualizations to show the connections between words.