ETC1010: Introduction to Data Analysis
Week 8, part B

Text analysis and linear models

Lecturer: *Nicholas Tierney*

Department of Econometrics and Business Statistics

nicholas.tierney@monash.edu

May 2020

Recap

- tidying up text
- unnest_tokens
- stop words - (I, am, be, the, this, what, we, myself)
- sentiment analysis

Upcoming Assessment

- Project
- Practical Exam
- Final Exam

Project

- Complete ED quiz before Thursday
- Focus on narrowing down some interesting questions and datasets

Practice Exams

- Practice exams are up for the final exam and the practical exam

Overview

- Tidy Text continued
- Term Frequency
- Inverse Document Frequency
- More practice

How do we measure the importance of a word to a document in a collection of documents?

i.e a novel in a collection of novels or a review in a set of reviews...

We combine the following statistics:

- Term frequency
- Inverse document frequency

Term frequency

The raw frequency of a word w in a document d. It is a function of the word and the document.

$$tf(w, d) = \frac{\text{count of w in d}}{\text{total count in d}}$$

Harry Potter books

Using data from Harry potter:

```
## # A tibble: 200 x 2
##    book                text
##    <fct>               <chr>
##  1 Philosopher's S… "THE BOY WHO LIVED    Mr. and Mrs. Dursley, of number four, Priv
##  2 Philosopher's S… "THE VANISHING GLASS    Nearly ten years had passed since the Du
##  3 Philosopher's S… "THE LETTERS FROM NO ONE    The escape of the Brazilian boa cons
##  4 Philosopher's S… "THE KEEPER OF THE KEYS    BOOM. They knocked again. Dudley jerk
##  5 Philosopher's S… "DIAGON ALLEY    Harry woke early the next morning. Although he
##  6 Philosopher's S… "THE JOURNEY FROM PLATFORM NINE AND THREE-QUARTERS    Harry's la
##  7 Philosopher's S… "THE SORTING HAT    The door swung open at once. A tall, black-h
##  8 Philosopher's S… "THE POTIONS MASTER    There, look.\"    \"Where?\"    \"Next to
##  9 Philosopher's S… "THE MIDNIGHT DUEL    Harry had never believed he would meet a b
## 10 Philosopher's S… "HALLOWEEN    Malfoy couldn't believe his eyes when he saw that
## # … with 190 more rows
```

Harry Potter books

Unnest tokens, and use `count` to count up the words within each book:

```
book_words <- hp_books %>%
  unnest_tokens(word, text) %>%
  count(book, word, sort = TRUE)
```

```
book_words
## # A tibble: 67,881 x 3
##    book                word      n
##    <fct>               <chr> <int>
##  1 Order of the Phoenix the   11740
##  2 Deathly Hallows      the   10335
##  3 Goblet of Fire       the    9305
##  4 Half-Blood Prince    the    7508
##  5 Order of the Phoenix to     6518
##  6 Order of the Phoenix and    6189
##  7 Deathly Hallows      and    5510
##  8 Order of the Phoenix of     5332
##  9 Prisoner of Azkaban  the    4990
## 10 Goblet of Fire       and    4959
## # … with 67,871 more rows
```

Term frequency

Let's calculate frequency of words for The Philosopher's Stone

```
stopwords_smart <- get_stopwords(source = "smart")

document <- book_words %>%
  anti_join(stopwords_smart) %>%
  filter(book == "Philosopher's Stone")
document
## # A tibble: 5,547 x 3
##    book                word           n
##    <fct>               <chr>      <int>
##  1 Philosopher's Stone harry       1213
##  2 Philosopher's Stone ron          410
##  3 Philosopher's Stone hagrid       336
##  4 Philosopher's Stone back         261
##  5 Philosopher's Stone hermione     257
##  6 Philosopher's Stone professor    181
##  7 Philosopher's Stone looked       169
##  8 Philosopher's Stone snape        145
##  9 Philosopher's Stone dumbledore   143
```

Term frequency

The term frequency for each word is the number of times that word occurs divided by the total number of words in the document.

```
tbl_tf <- document %>%
  mutate(tf = n / sum(n))

tbl_tf %>%
  arrange(desc(tf))
## # A tibble: 5,547 x 4
##    book                word           n      tf
##    <fct>               <chr>      <int>   <dbl>
##  1 Philosopher's Stone harry       1213 0.0385
##  2 Philosopher's Stone ron          410 0.0130
##  3 Philosopher's Stone hagrid       336 0.0107
##  4 Philosopher's Stone back         261 0.00829
##  5 Philosopher's Stone hermione     257 0.00817
##  6 Philosopher's Stone professor    181 0.00575
##  7 Philosopher's Stone looked       169 0.00537
##  8 Philosopher's Stone snape        145 0.00461
##  9 Philosopher's Stone dumbledore   143 0.00454
```

Inverse-document frequency

We can instead look at a term's inverse document frequency (idf), which:

- Decreases weight for commonly used words, while
- Increasing weight for those words not used much in a collection of documents.

This effectively tells us **how common or rare a word is accross a collection of documents**.

It is a function of a word w, and the collection of documents D.

$$idf(w, D) = \log\left(\frac{\text{Number of } D}{\text{Number of documents containing } w}\right)$$

Inverse-document frequency: Example

Let's say that we had 20 documents:

- Out of 20 documents D
- How many documents contain the word, "the". (All 20 contain "the")

$$idf(w = 20, D = 20) = \log\left(\frac{20}{20}\right)$$

$$idf(w = 20, D = 20) = \log(1)$$

$$idf(w = 20, D = 20) = 0$$

Inverse-document frequency: Example

Let's say that we had 20 documents:

- Out of 20 documents D
- How many documents contain the word, "Deciduous". (Only 1 contains the word "Deciduous")

$$idf(w = 1, D = 20) = \log\left(\frac{20}{1}\right)$$

$$idf(w = 1, D = 20) = \log(20)$$

$$idf(w = 1, D = 20) = 2.995$$

Inverse-document frequency: Example

Let's say that we had 20 documents:

- Out of 20 documents D
- How many documents contain the word, "Banana". (10 contain the word "Banana")

$$idf(w = 10, D = 20) = \log\left(\frac{20}{10}\right)$$

$$idf(w = 10, D = 20) = \log(2)$$

$$idf(w = 1, D = 2) = 0.693$$

Inverse document frequency

- When it is higher: Word is not used much in a collection of documents
  - E.g., 1 document uses "deciduous"
- When it is lower: Word is not commonly used much in a collection of documents
  - E.g., all documents use "the", not as many use "bananas"

Inverse document frequency

For the Harry Potter books, we could compute this in a somewhat roundabout as follows:

```
tbl_idf <- book_words %>%
    anti_join(stopwords_smart) %>%
    mutate(collection_size = n_distinct(book)) %>%
    group_by(collection_size, word) %>%
    summarise(times_word_used = n_distinct(book)) %>%
    mutate(freq = collection_size / times_word_used,
           idf = log(freq))
arrange(tbl_idf, idf)
## # A tibble: 23,945 x 5
## # Groups:   collection_size [1]
##    collection_size word         times_word_used  freq   idf
##              <int> <chr>                  <int> <dbl> <dbl>
##  1               7 absolutely                 7     1     0
##  2               7 absurd                     7     1     0
##  3               7 accept                     7     1     0
##  4               7 accepted                   7     1     0
##  5               7 accident                   7     1     0
```

Putting it together

term frequency, inverse document frequency

Multiply tf and idf together. This is a function of a word w, a document d, and the collection of documents D:

$$tf_i df(w, d, D) = tf(w, d) \times idf(w, D)$$

- A **High value** of tf_idf means a word has a high frequency within a document but is quite rare over all documents.
- Likewise if a word occurs in a lot of documents idf will be close to zero, so tf_idf will be small.

TF IDF summary

- TF IDF helps us find those words that are important in the content of documents
- It does this by increasing the weight of words not used very much in a collection, since the IDF is higher when a word isn't used often.
- So a **higher** TF IDF means the word is more important if it is both used a lot (has a high term frequency), and is uncommon (higher IDF).
- And a **lower** TF IDF means the word is less important, since it might be really common (high term frequency), but be really common (lower IDF).

Putting it together, tf-idf

We can calculate TF IDF using `bind_tf_idf()`

```
book_words_counts <- book_words %>%
      anti_join(stopwords_smart) %>%
      bind_tf_idf(term = word, document = book, n = n)

book_words_counts
## # A tibble: 64,582 x 6
##    book                 word          n     tf   idf tf_idf
##    <fct>                <chr>     <int>  <dbl> <dbl>  <dbl>
##  1 Order of the Phoenix harry      3730 0.0352     0      0
##  2 Goblet of Fire       harry      2936 0.0369     0      0
##  3 Deathly Hallows      harry      2770 0.0345     0      0
##  4 Half-Blood Prince    harry      2581 0.0374     0      0
##  5 Prisoner of Azkaban  harry      1824 0.0408     0      0
##  6 Chamber of Secrets   harry      1503 0.0409     0      0
##  7 Order of the Phoenix hermione   1220 0.0115     0      0
##  8 Philosopher's Stone  harry      1213 0.0385     0      0
##  9 Order of the Phoenix ron        1189 0.0112     0      0
## 10 Deathly Hallows      hermione   1077 0.0134     0      0
```

What words were important to the books?

Your Turn

Explore uncommon / important words in Jane Austen's books!

- Complete "8b-jane-austen-tf-idf.Rmd"

Sentiment analysis

Sentiment analysis tags words or phrases with an emotion, and summarises these, often as the positive or negative state, over a body of text.

Sentiment analysis: examples

- Examining effect of emotional state in twitter posts
- Determining public reactions to government policy, or new product releases
- Trying to make money in the stock market by modeling social media posts on listed companies
- Evaluating product reviews on Amazon, restaurants on zomato, or travel options on TripAdvisor

Lexicons

The `tidytext` package has a lexicon of sentiments, based on four major sources: AFINN, bing, Loughran, nrc

emotion

What emotion do these words elicit in you?

- summer
- hot chips
- hug
- lose
- stolen
- smile

Different sources of sentiment

- The `nrc` lexicon categorizes words in a binary fashion ("yes"/"no") into categories of positive, negative, anger, anticipation, disgust, fear, joy, sadness, surprise, and trust.
- The `bing` lexicon categorizes words in a binary fashion into positive and negative categories.
- The `AFINN` lexicon assigns words with a score that runs between -5 and 5, with negative scores indicating negative sentiment and positive scores indicating positive sentiment.

Different sources of sentiment

```
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
```

Sentiment analysis

- Once you have a bag of words, you need to join the sentiments dictionary to the words data.
- Particularly the lexicon `nrc` has multiple tags per word, so you may need to use an "inner_join".
- `inner_join()` returns all rows from x where there are matching values in y, and all columns from x and y.
- If there are multiple matches between x and y, all combination of the matches are returned.

Exploring sentiment in Harry Potter

```
book_words
## # A tibble: 67,881 x 3
##    book                 word      n
##    <fct>                <chr> <int>
##  1 Order of the Phoenix the   11740
##  2 Deathly Hallows      the   10335
##  3 Goblet of Fire       the    9305
##  4 Half-Blood Prince    the    7508
##  5 Order of the Phoenix to     6518
##  6 Order of the Phoenix and    6189
##  7 Deathly Hallows      and    5510
##  8 Order of the Phoenix of     5332
##  9 Prisoner of Azkaban  the    4990
## 10 Goblet of Fire       and    4959
## # … with 67,871 more rows
```

Count joyful words in "Chamber of Secrets"

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

book_words %>%
  filter(book == "Chamber of Secrets") %>%
  inner_join(nrc_joy) %>%
  arrange(desc(n))
## # A tibble: 205 x 4
##    book               word         n sentiment
##    <fct>              <chr>    <int> <chr>
##  1 Chamber of Secrets good        85 joy
##  2 Chamber of Secrets diary       64 joy
##  3 Chamber of Secrets found       53 joy
##  4 Chamber of Secrets smile       29 joy
##  5 Chamber of Secrets white       25 joy
##  6 Chamber of Secrets green       24 joy
##  7 Chamber of Secrets feeling     21 joy
##  8 Chamber of Secrets kind        18 joy
##  9 Chamber of Secrets magical     18 joy
## 10 Chamber of Secrets pleased     18 joy
```

Count joyful words in "Chamber of Secrets"

```
## # A tibble: 6 x 4
##   book                word      n sentiment
##   <fct>               <chr> <int> <chr>
## 1 Chamber of Secrets good     85 joy
## 2 Chamber of Secrets diary    64 joy
## 3 Chamber of Secrets found    53 joy
## 4 Chamber of Secrets smile    29 joy
## 5 Chamber of Secrets white    25 joy
## 6 Chamber of Secrets green    24 joy
```

"Good" is the most common joyful word, followed by "diary", "found", and "smile".

These make sense ... except for "diary", and "found",

and ... "white" and "green" ?

Your turn: go to rstudio.cloud

Go to "8b-jane-austen-sentiment.Rmd"

- What are the most common "anger" words used in Emma?
- What are the most common "surprise" words used in Emma?

Comparing lexicons

- All of the lexicons have a measure of positive or negative.
- We can tag the words in Emma by each lexicon, and see if they agree.

```
nrc_pn <- get_sentiments("nrc") %>%
  filter(sentiment %in% c("positive",
                          "negative"))

secrets_nrc <- book_words %>%
  filter(book == "Chamber of Secrets") %>%
  inner_join(nrc_pn)

secrets_bing <- book_words %>%
  filter(book == "Chamber of Secrets") %>%
  inner_join(get_sentiments("bing"))

secrets_afinn <- book_words %>%
  filter(book == "Chamber of Secrets") %>%
  inner_join(get_sentiments("afinn"))
```

Comparing lexicons

```
secrets_nrc
## # A tibble: 1,291 x 4
##    book              word          n sentiment
##    <fct>             <chr>     <int> <chr>
##  1 Chamber of Secrets harry      1503 negative
##  2 Chamber of Secrets professor   190 positive
##  3 Chamber of Secrets sir          88 positive
##  4 Chamber of Secrets good         85 positive
##  5 Chamber of Secrets diary        64 positive
##  6 Chamber of Secrets black        61 negative
##  7 Chamber of Secrets found        53 positive
##  8 Chamber of Secrets small        51 negative
##  9 Chamber of Secrets boy          49 negative
## 10 Chamber of Secrets wizard       45 positive
## # … with 1,281 more rows
```

Comparing lexicons

```
secrets_afinn
## # A tibble: 768 x 4
##    book               word        n value
##    <fct>              <chr>   <int> <dbl>
##  1 Chamber of Secrets no        221    -1
##  2 Chamber of Secrets like      184     2
##  3 Chamber of Secrets good       85     3
##  4 Chamber of Secrets great      67     3
##  5 Chamber of Secrets want       66     1
##  6 Chamber of Secrets better     54     2
##  7 Chamber of Secrets hard       47    -1
##  8 Chamber of Secrets reached    43     1
##  9 Chamber of Secrets stop       42    -1
## 10 Chamber of Secrets help       40     2
## # … with 758 more rows
```

Comparing lexicons

```
secrets_nrc %>%
  count(sentiment, name = "n_sentiment") %>%
  mutate(prop_total = n_sentiment / sum(n_sentiment))
## # A tibble: 2 x 3
##   sentiment n_sentiment prop_total
## * <chr>           <int>      <dbl>
## 1 negative         4524      0.609
## 2 positive         2904      0.391

secrets_bing %>%
  count(sentiment, name = "n_sentiment") %>%
  mutate(prop_total = n_sentiment / sum(n_sentiment))
## # A tibble: 2 x 3
##   sentiment n_sentiment prop_total
## * <chr>           <int>      <dbl>
## 1 negative         2970      0.582
## 2 positive         2133      0.418
```

Comparing lexicons

```
secrets_afinn %>%
  mutate(sentiment = ifelse(value > 0,
                            "positive",
                            "negative")) %>%
  count(sentiment, name = "n_sentiment") %>%
  mutate(prop_total = n_sentiment / sum(n_sentiment))
## # A tibble: 2 x 3
##   sentiment n_sentiment prop_total
## * <chr>           <int>      <dbl>
## 1 negative         2273      0.531
## 2 positive         2010      0.469
```

Your turn:

Continue along with "8b-jane-austen-sentiment.Rmd"

- Using your choice of lexicon (nrc, bing, or afinn) compute the proportion of positive words in each of Austen's books.
- Which book is the most positive? negative?

Example: Simpsons

Data from the popular animated TV series, The Simpsons, has been made available on kaggle.

- `simpsons_script_lines.csv`: Contains the text spoken during each episode (including details about which character said it and where)
- `simpsons_characters.csv`: Contains character names and a character id

The Simpsons

```
scripts <- read_csv("data/simpsons_script_lines.csv")
chs <- read_csv("data/simpsons_characters.csv")
sc <- left_join(scripts, chs, by = c("character_id" = "id"))

sc
## # A tibble: 157,462 x 16
##         id episode_id number raw_text timestamp_in_ms speaking_line character_id
##      <dbl>      <dbl>  <dbl> <chr>              <dbl> <lgl>                <dbl>
##  1   9549         32    209 Miss Ho…           848000 TRUE                   464
##  2   9550         32    210 Lisa Si…           856000 TRUE                     9
##  3   9551         32    211 Miss Ho…           856000 TRUE                   464
##  4   9552         32    212 Lisa Si…           864000 TRUE                     9
##  5   9553         32    213 Edna Kr…           864000 TRUE                    40
##  6   9554         32    214 Martin …           877000 TRUE                    38
##  7   9555         32    215 Edna Kr…           881000 TRUE                    40
##  8   9556         32    216 Bart Si…           882000 TRUE                     8
##  9   9557         32    217 (Apartm…           889000 FALSE                   NA
## 10   9558         32    218 Lisa Si…           889000 TRUE                     9
## # … with 157,452 more rows, and 9 more variables: location_id <dbl>,
## #   raw_character_text <chr>, raw_location_text <chr>, spoken_words <chr>,
```

count the number of times a character speaks

```
sc %>% count(name, sort = TRUE)
## # A tibble: 6,143 x 2
##    name                    n
##    <chr>               <int>
##  1 Homer Simpson       29945
##  2 <NA>                19661
##  3 Marge Simpson       14192
##  4 Bart Simpson        13894
##  5 Lisa Simpson        11573
##  6 C. Montgomery Burns  3196
##  7 Moe Szyslak          2853
##  8 Seymour Skinner      2437
##  9 Ned Flanders         2139
## 10 Grampa Simpson       1952
## # … with 6,133 more rows
```

missing name?

```
sc %>% filter(is.na(name))
## # A tibble: 19,661 x 16
##       id episode_id number raw_text timestamp_in_ms speaking_line character_id
##    <dbl>      <dbl>  <dbl> <chr>              <dbl> <lgl>                <dbl>
##  1  9557         32    217 (Apartm…          889000 FALSE                   NA
##  2  9565         32    225 (Spring…          918000 FALSE                   NA
##  3 75766        263    106 (Moe's …          497000 FALSE                   NA
##  4  9583         32    243 (Train …          960000 FALSE                   NA
##  5  9604         32    264 (Simpso…         1070000 FALSE                   NA
##  6  9655         33      0 (Simpso…           84000 FALSE                   NA
##  7  9685         33     30 (Simpso…          177000 FALSE                   NA
##  8  9686         33     31 (Simpso…          177000 FALSE                   NA
##  9  9727         33     72 (Simpso…          349000 FALSE                   NA
## 10  9729         33     74 (Simpso…          355000 FALSE                   NA
## # … with 19,651 more rows, and 9 more variables: location_id <dbl>,
## #   raw_character_text <chr>, raw_location_text <chr>, spoken_words <chr>,
## #   normalized_text <chr>, word_count <chr>, name <chr>, normalized_name <chr>,
## #   gender <chr>
```

Simpsons Pre-process the text

```
sc %>%
  unnest_tokens(output = word,
                input = spoken_words)
## # A tibble: 1,355,370 x 16
##       id episode_id number raw_text timestamp_in_ms speaking_line character_id
##    <dbl>      <dbl>  <dbl> <chr>              <dbl> <lgl>                <dbl>
##  1  9549         32    209 Miss Ho…          848000 TRUE                   464
##  2  9549         32    209 Miss Ho…          848000 TRUE                   464
##  3  9549         32    209 Miss Ho…          848000 TRUE                   464
##  4  9549         32    209 Miss Ho…          848000 TRUE                   464
##  5  9549         32    209 Miss Ho…          848000 TRUE                   464
##  6  9549         32    209 Miss Ho…          848000 TRUE                   464
##  7  9549         32    209 Miss Ho…          848000 TRUE                   464
##  8  9549         32    209 Miss Ho…          848000 TRUE                   464
##  9  9549         32    209 Miss Ho…          848000 TRUE                   464
## 10  9549         32    209 Miss Ho…          848000 TRUE                   464
## # … with 1,355,360 more rows, and 9 more variables: location_id <dbl>,
## #   raw_character_text <chr>, raw_location_text <chr>, normalized_text <chr>,
## #   word_count <chr>, name <chr>, normalized_name <chr>, gender <chr>, word <chr>
```

Simpsons Pre-process the text

```
sc %>%
  unnest_tokens(output = word,
                input = spoken_words) %>%
  anti_join(stop_words)
## # A tibble: 511,869 x 16
##        id episode_id number raw_text timestamp_in_ms speaking_line character_id
##     <dbl>      <dbl>  <dbl> <chr>              <dbl> <lgl>                <dbl>
##  1  9549         32    209 Miss Ho…          848000 TRUE                   464
##  2  9549         32    209 Miss Ho…          848000 TRUE                   464
##  3  9549         32    209 Miss Ho…          848000 TRUE                   464
##  4  9549         32    209 Miss Ho…          848000 TRUE                   464
##  5  9550         32    210 Lisa Si…          856000 TRUE                     9
##  6  9551         32    211 Miss Ho…          856000 TRUE                   464
##  7  9551         32    211 Miss Ho…          856000 TRUE                   464
##  8  9551         32    211 Miss Ho…          856000 TRUE                   464
##  9  9551         32    211 Miss Ho…          856000 TRUE                   464
## 10  9551         32    211 Miss Ho…          856000 TRUE                   464
## # … with 511,859 more rows, and 9 more variables: location_id <dbl>,
## #   raw_character_text <chr>, raw_location_text <chr>, normalized_text <chr>,
## #   word_count <chr>, name <chr>, normalized_name <chr>, gender <chr>, word <chr>
```

Simpsons Pre-process the text

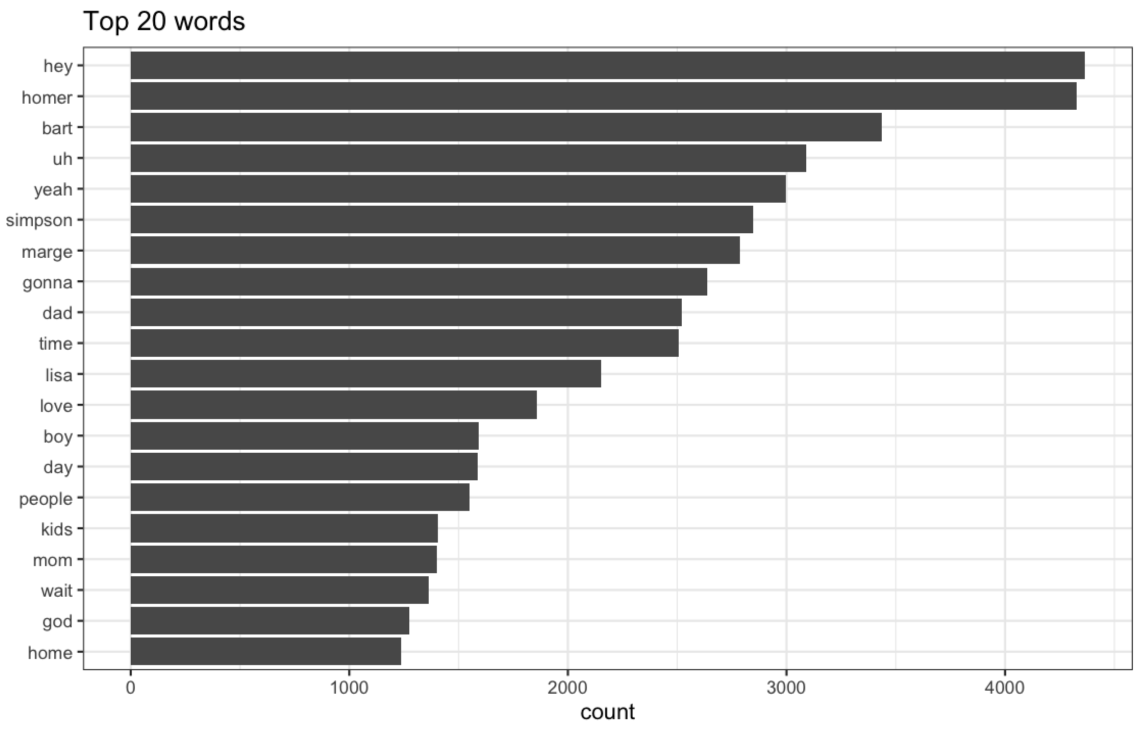
```
sc %>%
  unnest_tokens(output = word,
                input = spoken_words) %>%
  anti_join(stop_words) %>%
  count(word, sort = TRUE) %>%
  filter(!is.na(word))
## # A tibble: 41,891 x 2
##    word        n
##    <chr>   <int>
##  1 hey      4366
##  2 homer    4328
##  3 bart     3434
##  4 uh       3090
##  5 yeah     2997
##  6 simpson  2846
##  7 marge    2786
##  8 gonna    2639
##  9 dad      2521
## 10 time     2508
## # … with 41,881 more rows
```

Simpsons Pre-process the text

```
sc_top_20 <- sc %>%
  unnest_tokens(output = word,
                input = spoken_words) %>%
  anti_join(stop_words) %>%
  count(word, sort = TRUE) %>%
  filter(!is.na(word)) %>%
  mutate(word = factor(word,
                       levels = rev(unique(word)))) %>%
  top_n(20)
```

Simpsons plot most common words

```
ggplot(sc_top_20,
       aes(x = word,
           y = n)) +
  geom_col() +
  labs(x = '',
       y = 'count',
       title = 'Top 20 words') +
  coord_flip() +
  theme_bw()
```



Tag the words with sentiments

Using AFINN words will be tagged on a negative to positive scale of -1 to 5.

```
sc_word <- sc %>%
  unnest_tokens(output = word, input = spoken_words) %>%
  anti_join(stop_words) %>%
  count(name, word) %>%
  filter(!is.na(word))

sc_word
## # A tibble: 220,838 x 3
##    name             word            n
##    <chr>            <chr>       <int>
##  1 '30s Reporter    burns           1
##  2 '30s Reporter    kinda           1
##  3 '30s Reporter    sensational     1
##  4 1-Year-Old Bart beer            1
##  5 1-Year-Old Bart daddy           5
##  6 1-Year-Old Bart fat             1
##  7 1-Year-Old Bart moustache       1
```

Tag the words with sentiments

```
sc_s <- sc_word %>%
  inner_join(get_sentiments("afinn"), by = "word")

sc_s
## # A tibble: 26,688 x 4
##    name                 word        n value
##    <chr>                <chr>   <int> <dbl>
##  1 1-Year-Old Bart      nice        1     3
##  2 10-Year-Old Homer chance      1     2
##  3 10-Year-Old Homer cool        1     1
##  4 10-Year-Old Homer die         1    -3
##  5 10-Year-Old Homer died        1    -3
##  6 10-Year-Old Homer dreams      1     1
##  7 10-Year-Old Homer happy       1     3
##  8 10-Year-Old Homer heaven      1     2
##  9 10-Year-Old Homer hell        1    -4
## 10 10-Year-Old Homer kiss        1     2
## # … with 26,678 more rows
```

Examine Simpsons characters

```
sc_s %>%
  group_by(name) %>%
  summarise(m = mean(value)) %>%
  arrange(desc(m))
## # A tibble: 3,409 x 2
##    name                     m
##    <chr>                <dbl>
##  1 2nd Sportscaster         4
##  2 4-h Judge                4
##  3 7-Year-Old Brockman      4
##  4 ALEPPO                   4
##  5 All Kids                 4
##  6 Applicants               4
##  7 Australian               4
##  8 Bill James               4
##  9 Canadian Player          4
## 10 Carl Kasell              4
## # … with 3,399 more rows
```

Examine Simpsons characters: Focus characters.

```
keep <- sc %>% count(name,
                     sort=TRUE) %>%
  filter(!is.na(name)) %>%
  filter(n > 999)

sc_s %>%
  filter(name %in% keep$name) %>%
  group_by(name) %>%
  summarise(m = mean(value)) %>%
  arrange(m)
## # A tibble: 16 x 2
##    name                  m
##    <chr>             <dbl>
##  1 Nelson Muntz     -0.519
##  2 Grampa Simpson   -0.429
##  3 Homer Simpson    -0.428
##  4 Bart Simpson     -0.391
##  5 Chief Wiggum     -0.388
##  6 Lisa Simpson     -0.388
##  7 Marge Simpson    -0.344
```

Your turn: "8b-simpsons.Rmd"

1. Bart Simpson is featured at various ages. How has the sentiment of his words changed over his life?
2. Repeat the sentiment analysis with the NRC lexicon. What character is the most "angry"? "joyful"?

Extension: Explore Harry Potter

I've included the harry potter data the code from the harry potter part of the lecture in "8b-harry-potter.Rmd", if you want to have a play around, I've got a few questions there.

Further extension

Text Mining with R has an example comparing historical physics textbooks: *Discourse on Floating Bodies* by Galileo Galilei, *Treatise on Light* by Christiaan Huygens, *Experiments with Alternate Currents of High Potential and High Frequency* by Nikola Tesla, and *Relativity: The Special and General Theory* by Albert Einstein. All are available on the Gutenberg project.

Work your way through the comparison of physics books. It is section 3.4.

Thanks

- Dr. Mine Çetinkaya-Rundel
- Dr. Julia Silge: https://github.com/juliasilge/tidytext-tutorial and https://juliasilge.com/blog/animal-crossing/
- Dr. Julia Silge and Dr. David Robinson: https://www.tidytextmining.com/