Introduction to Applied Data Science

Lecture 7: Text as Data and Text Mining

Bas Machielsen
Utrecht University
2024-04-22

Introduction

Introduction

- Overview of this class:
    - Introduction to Data Science
    - Introduction to R & Programming
    - Getting Data: API's and Databases
    - Getting Data: Web Scraping
    - Transforming and Cleaning Data
    - Spatial & Network Data
    - **This lecture**: Text as Data and Mining
    - Data Science Project

Introduction

- As you know by now, there are many kinds of different data
- (Silge and Robinson, 2021) note:

> Analysts are often trained to handle tabular or rectangular data that is mostly numeric, but much of the data proliferating today is unstructured and text-heavy. Many of us who work in analytical fields are not trained in even simple interpretation of natural language.

- One approach is to treat text as `data.frames` of individual words:
- We can **manipulate, summarize, and visualize** the characteristics of text easily and integrate **natural language processing** into effective workflows we were already using.
- One possible goal of this: an attempt to transform text into numbers

Today's Program

- We will talk about the equivalent of **tidy data** specially for text data, called **tidy text**
- We will explore **sentiment analysis**, a naive and basic way to analyze and quantify text data
- We generalize the practice of sentiment analysis to **frequency analysis** and *N*-grams
- Finally, we'll talk about a slightly more sophisticated method of analyzing text data called **topic modeling**

Tidy Text

Tidy Text

- Remember the **tidy data format**:
    - Each variable is a column
    - Each observation is a row
    - Each type of observational unit is a table
- We define the **tidy text format** as being a `data.frame` with **one-token-per-row.**
- A token is a meaningful unit of text, such as a word, that we are interested in using for analysis, and tokenization is the process of splitting text into tokens.
- For now, you can think of a **token** as a word, but there are several reservations: e.g. is a stop word a token? How about a dot?

Example Tidy Text

- Suppose we want to turn this text into a tidy format:

```r
library(tidytext)
text_df <- tibble(text = c("To be, or not to be; that is the question;",
                           "Whether 'tis nobler in the mind to suffer",
                           "The slings and arrows of outrageous fortune",
                           "Or to take arms against a sea of troubles"),
                  lines=1:4)

text_df
```

```
## # A tibble: 4 × 2
##   text                                        lines
##   <chr>                                       <int>
## 1 To be, or not to be; that is the question;      1
## 2 Whether 'tis nobler in the mind to suffer       2
## 3 The slings and arrows of outrageous fortune     3
## 4 Or to take arms against a sea of troubles       4
```

Example Tidy Text

- We can do so using the `unnest_tokens()` function:
  - Where the syntax is `unnest_tokens(data.frame, output, input)`

```
text_df ▷
  unnest_tokens(word, text) ▷
  head(5)
```

```
## # A tibble: 5 × 2
##   lines word
##   <int> <chr>
## 1     1 to
## 2     1 be
## 3     1 or
## 4     1 not
## 5     1 to
```

Other Text Data Structures

- It is insightful to compare **tidy text** to other data structures (which we will also use)
- **Simple string format**: Easiest format. In R: character vectors
  - Often raw data is first read into memory in this form (See previous example)
  - E.g.: you copy or extract a text from a Wikipedia page
- **Corpus**: These types of objects typically contain raw strings annotated with additional metadata and details (In our example: line numbers)
- **Document-term matrix**: A `data.frame` (i.e., a corpus) of documents with one row for each document and one column for each term.
  - The value in the matrix is typically a **word count** or the **tf-idf** (we'll get to this later)

Unnest Tokens

- How to proceed from one format to another?
- We have already seen how to go from string format to tidy text:

```
text ← c("Lorem Ipsum is simply dummy text of the printing and typesetting indust
          Lorem Ipsum has been the industry's standard dummy text ever since the 1
          when an unknown printer took a galley of type and scrambled it to make a
          specimen book. It has survived not only five centuries, but also the lea
          electronic typesetting, remaining essentially unchanged.")

tokens ← tibble(text = text) ▷
  unnest_tokens(word, text)

tokens ▷ head(5)
```

```
## # A tibble: 5 × 1
##   word
##   <chr>
## 1 lorem
## 2 ipsum
## 3 is
## 4 simply
## 5 dummy
```

Example: Cleaning Data from Books

- We have a dataset `austen_books()` from the `janeaustenr` package, with raw text data from Jane Austen books, from which we will make a corpus:

```
library(janeaustenr)
original_books ← austen_books() ▷
  group_by(book) ▷
  mutate(linenumber = row_number(),
         chapter = cumsum(str_detect(text,  # Regex detects Roman Numerals
                                     regex("^chapter [\\divxlc]",
                                           ignore_case = TRUE)))) ▷
  ungroup()

original_books ▷ head(5)
```

```
## # A tibble: 5 × 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
```

Example: Cleaning Data from Books

- From this corpus, we now make tokens using the `unnest_tokens()` function:

```
book_tokens <- original_books |>
  unnest_tokens(word, text)

book_tokens |> head(8)
```

```
## # A tibble: 8 × 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          5       0 1811
## 8 Sense & Sensibility         10       1 chapter
```

Stop Words

- We have already seen various words, like "chapter", which we intuitively do not accord relevance
- A similar argument pertains to words like "the, it, and," etc. , i.e. **stop words**
- These words often **lack relevance**
  - But you should be careful in deciding whether to remove them
- An easy way (in English) to remove them is provided by the `tidytext` package in the dataset `stop_words`:

```
stop_words ▷ head(5)
```

```
## # A tibble: 5 × 2
##   word  lexicon
##   <chr> <chr>
## 1 a     SMART
## 2 a's   SMART
## 3 able  SMART
## 4 about SMART
## 5 above SMART
```

Removing Stop Words

- An easy way to remove stop words is to use the `anti_join()` function
- `anti_join()` removes all words in the "left" dataset that also exist in the "right" dataset

```
clean_tokens ← book_tokens ▷ anti_join(stop_words)

clean_tokens ▷
  head(8)
```

```
## # A tibble: 8 × 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          5       0 1811
## 6 Sense & Sensibility         10       1 chapter
## 7 Sense & Sensibility         10       1 1
## 8 Sense & Sensibility         13       1 family
```

Adding Words to Stop Words

- You can also **add words** to a stop words list manually
- For future reference, let's add numbers to the `stop_words` data.frame:

```r
numbers ← tibble(word = 0:3000, lexicon = "custom") ▷
  mutate(word=as.character(word))

stop_words ← bind_rows(numbers, stop_words)
```

- The R package `stopwords` contains many lists of stop words in many different languages

Word Frequency

- The first basic analysis we can perform on this tidy text data is computing the **word frequency**
- We can obtain a word frequency list by:

```r
clean_tokens ▷
  count(word, sort = TRUE) ▷
  head(10)
```

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

Another Example: Wikipedia

- We use web scraping to extract a text from an article on Wikipedia
  - Remove stop words
  - And then count word frequency

```r
library(rvest)
link <- "https://en.wikipedia.org/wiki/2022_FIFA_World_Cup"

# Go to the wikipedia page, right click, look at the structure of the page
# Extract all p's in a <div> with class "mw-parser-output"
wc2022 <- read_html(link) |>
  html_elements("div.mw-parser-output p") |>
  html_text()

data <- tibble(section = seq(wc2022), text = wc2022) |>
  unnest_tokens(word, text) |>
  anti_join(stop_words)
```

Another Example: Wikipedia

- The tidy text data looks like this:

```
data ▷
  head(5)
```

```
## # A tibble: 5 × 2
##   section word
##     <int> <chr>
## 1       3 fifa
## 2       3 world
## 3       3 cup
## 4       3 22nd
## 5       3 fifa
```

- Now, we can make a **word frequency count**:

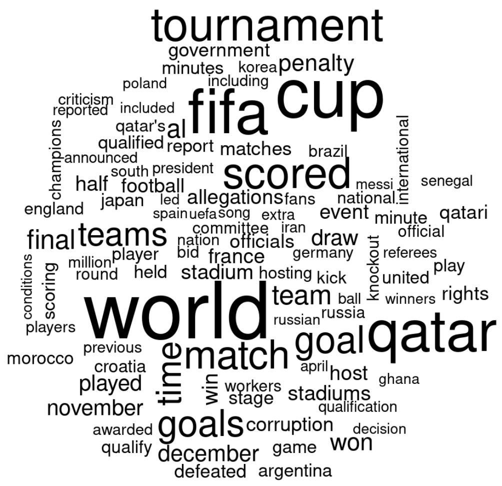
```
data ▷
  count(word, sort = TRUE) ▷
  head(5)
```

```
## # A tibble: 5 × 2
##   word           n
##   <chr>      <int>
## 1 world        118
## 2 cup           99
## 3 fifa          88
## 4 qatar         81
## 5 tournament    62
```

Wordcloud

- This is also the basis for a visualization technique called a **wordcloud**

```r
library(wordcloud)

data |>
  count(word, sort = TRUE) |>
  with(wordcloud(word, n, max.words = 100))
```



Sentiment Analysis

Sentiment Analysis

- A more nuanced and productive way of analyzing text data is called **opinion mining or sentiment analysis**
- We want to infer whether a section of text is **positive or negative**, or perhaps characterized by some other more **nuanced emotion** like surprise or disgust.
- The basics are really simple: we map a word to a number, e.g.:
    - Positive number: positive sentiment
    - Negative number: negative sentiment
- Some important obvious drawbacks:
    - Not every word is in the lexicon because many words are pretty neutral.
    - The methods do not take into account **qualifiers** before a word, such as in “no good” or “not true”
    - Doesn't understand **sarcasm or negated text**

Sentiment Analysis

- The `tidytext` package contains three automatic sentiment maps:
    - The `bing` lexicon categorizes words in a **binary fashion** into positive and negative categories.
    - 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 `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.

Example: Wikipedia Data

- Let's try to find out the sentiment according to the `bing` lexicon of each paragraph on the 2022 World Cup page:

```
bing ← get_sentiments("bing")
# Merge sentiment with your data with inner_join
sect_sent ← data ▷
  inner_join(bing) ▷
  count(section, sentiment) ▷
  pivot_wider(names_from = sentiment, values_from = n, values_fill = 0)

sect_sent ▷ head(5)
```

```
## # A tibble: 5 × 3
##   section positive negative
##     <int>    <int>    <int>
## 1       3        1        0
## 2       4        1        0
## 3       5       14        1
## 4       6        2        6
## 5       7        1        1
```

Example: Wikipedia Data

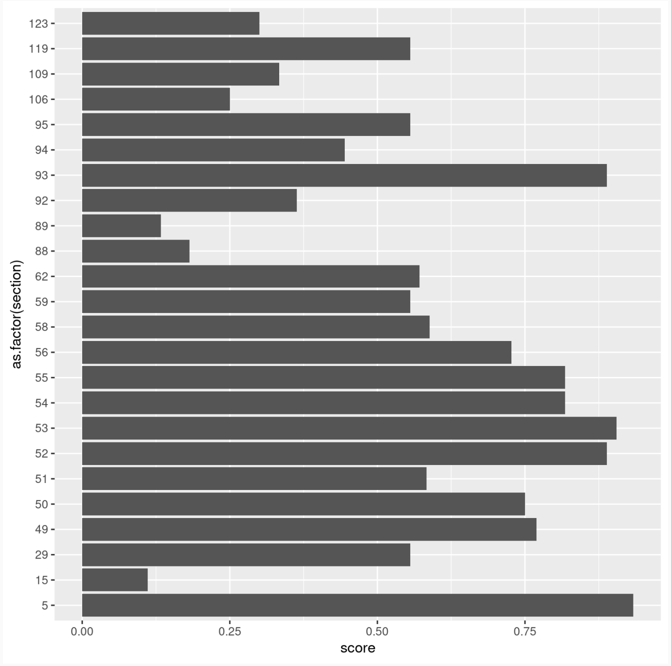
- Let's now calculate a **sentiment score**
- Merge it to the **most occurring words** in the section
    - Get a rough idea what the section is about and what the sentiment is:

```
# Most occurring words
mow ← data ▷
  group_by(section) ▷
  count(word) ▷
  slice_max(order_by = word, n = 10) ▷
  select(-n) ▷
  nest(data = word)

# Calculate a net sentiment score
sect_sent ← sect_sent ▷
  mutate(score = positive / (positive + negative)) ▷
  filter(positive + negative > 8)
```

Wikipedia Example

- We merge the two data frames and plot the section keywords against the sentiment:

```
score_df ← sect_sent ▷
  left_join(mow)

score_df ▷
  ggplot(aes(x = score, y = as.factor(section))) + geom_col()
```



Wikipedia Example

- Let us finally plot the score and the most important keywords, so as to get a sense which sections are very negative, and which are positive:

```
score_df ▷
  unnest() ▷
  group_by(section) ▷
  summarize(score = mean(score), text = stringr::str_c(word, collapse=", ")) ▷
  arrange(score) ▷
  head(8)
```

```
## # A tibble: 8 × 3
##   section score text
##     <int> <dbl> <chr>
## 1      15 0.111 zürich, world, votes, vote, united, uefa, tournaments, switzerland,
## 2      89 0.133 world, won, violation, senior, selection, representations, report, r
## 3      88 0.182 world, workers, women, wider, tv, treatment, ten, study, strong, sta
## 4     106 0.25  wrongdoing, vote, visa, vice, times, support, sunday, stated, sponso
## 5     123 0.3   water, waits, village, transportation, tourists, tents, tent, tap, s
## 6     109 0.333 summary, submitted, stating, significant, russia, reviewed, represen
## 7      92 0.364 world, workers, water, wage, violation, violated, treatment, time, s
## 8      94 0.444 zones, zone, world, wide, western, supreme, stated, social, sobering
```

Frequency Analysis

Frequency Analysis

- You might have realized that a lot of sections contain words like `world, cup`, etc.
  - You could look at the *term frequency* defined as the amount of times a word occurs / total amount of words in a document
- Ideally, you would also want to focus on the *uniqueness* of each word
- Another approach is to look at a term's **inverse document frequency** (idf)
  - This decreases the weight for commonly used words and increases the weight for words that are not used very much in a collection of documents.

$$\mathrm{idf(term)} = \log\left[\frac{N_{\mathrm{documents}}}{N_{\mathrm{documents\ containing\ term}}}\right]$$

Tf-Idf

- If you multiply the **term frequency** (the count of the word in a document) by its **inverse document frequency**, you get a statistic called the `tf-idf`
- The statistic `tf-idf` is intended to measure how important a word is to a document in a collection (a corpus) of documents
- For example, to one novel in a collection of novels or to one website in a collection of websites.
    - The higher the `tf-idf`, the more *relevance* a word has.

$$\text{tf-idf}(w, d) = \text{tf}(w, d) \times \text{idf}(w)$$

where $\text{tf}(w, d)$ represents term frequency or the number of occurrences of term w in document d, and the inverse document frequency of word w defined as before.

Tf-Idf

- The function `bind_tf_idf` calculates the `tf_idf` ratio for you:
- The syntax is `bind_tf_idf(frequency_df, term_var, section_var, count_var)`

```r
# Start with the World Cup data, calculate tf_idf on the basic of frequency count
tf_idf ← data ▷
  group_by(section) ▷
  count(word) ▷
  bind_tf_idf(word, section, n)

tf_idf ▷ head(5)
```

```
## # A tibble: 5 × 6
## # Groups:   section [1]
##   section word             n     tf   idf tf_idf
##     <int> <chr>        <int>  <dbl> <dbl>  <dbl>
## 1       3 22nd             1 0.0294  4.78 0.141
## 2       3 arab             1 0.0294  3.17 0.0932
## 3       3 asia             1 0.0294  4.78 0.141
## 4       3 awarded          1 0.0294  2.99 0.0879
## 5       3 championship     1 0.0294  4.09 0.120
```

Tf-Idf

- Let's see if we can get a more accurate description of the sections if we use the `tf-itf` rather than the 10 most occurring words in each paragraph:

- Potentially, you could now do **sentiment analysis** while weighting the words by their `tf_idf`

  - Alternatively, you could **filter the data frame** conditional on a particular `tf_df` threshold
  - This serves as a filter for words that distinguish sections/chapters/books from all others in your corpus

Tf-Idf

- Here, we display the 10 highest tf-idf words per section:

```
tf_idf ▷
  slice_max(order_by = tf_idf, n = 10) ▷
  select(section, word) ▷
  summarize(text = stringr::str_c(word, collapse=", ")) ▷
  head(5)
```

```
## # A tibble: 5 × 2
##   section text
##     <int> <chr>
## 1       3 held, 22nd, asia, championship, muslim, organized, world, arab, korea, sou
## 2       4 held, teams, cities, extremes, host's, hot, event, alongside, days, determ
## 3       5 player, golden, title, tournament's, winning, awarded, nation, final, goal
## 4       6 attracted, choice, community, scheduling, significant, wider, lack, strong
## 5       7 held, contested, round, teams, distancing, length, masks, negative, profes
```

N-grams and Topic Modeling

N-grams and Topic Modeling

- So far we've considered words as **individual units**, and considered their relationships to **sentiments or to documents**.
- Many interesting text analyses are based on the **relationships between words**, examining which words tend to follow others immediately, or that tend to **co-occur** within the same document
- An N-gram is a token consisting of n consecutive words:
    - We can do this by adding the `token = "ngrams"` option to `unnest_tokens()`, and set the parameter n to the number of consecutive words we pick as our tokens

N-grams and Topic Modeling

- We proceed again from the World Cup Dataset, but now **tokenize** it in terms of **bigrams**:

```r
bigram ← tibble(section = seq(wc2022), text = wc2022) ▷
  unnest_tokens(bigram, text, token = "ngrams", n = 2) ▷
  filter(!is.na(bigram))

bigram ▷ head(5)
```

```
## # A tibble: 5 × 2
##   section bigram
##     <int> <chr>
## 1       3 the 2022
## 2       3 2022 fifa
## 3       3 fifa world
## 4       3 world cup
## 5       3 cup was
```

Using Bigrams For Context

- You can analyze **bigrams** in the same way as you can analyze 1-grams
- For example, you can use it to find which words are preceded by "not"
- Potentially merge this (remember `left_join`, `inner_join`, etc.) to create new sentiment scores based on the relative presence of "not" before a word

```
bigram |>
  separate(bigram, c('word1','word2'), sep = " ") |>
  filter(word1 == "not") |>
  count(word1, word2, sort=TRUE) |>
  head(5)
```

```
## # A tibble: 5 × 3
##   word1 word2       n
##   <chr> <chr>   <int>
## 1 not   be          2
## 2 not   enough      2
## 3 not   provide     2
## 4 not   qualify     2
## 5 not   secure      2
```

Topic Modeling

Topic Modeling

- We often have collections of documents, such as blog posts or news articles, that we'd like to divide into natural groups so that we can understand them separately.
- Topic modeling is a method for **unsupervised classification** of such documents
- **Latent Dirichlet allocation (LDA)** is a particularly popular method for fitting a topic model. It treats each document as a mixture of topics, and each topic as a mixture of words.
- This allows documents to "overlap" each other in terms of content, rather than being separated into discrete groups, in a way that mirrors typical use of natural language.

Latent Dirichlet Allocation

(From Silge & Robinson, 2022)

- In R, we can do Latent Dirichlet Allocation using the package `topicmodels`
  - The input in LDA is a **document-term data.frame**
- The way LDA works:
  - Every document is a mixture of topics. For example, in a two-topic model we could say “Document 1 is 90% topic A and 10% topic B, while Document 2 is 30% topic A and 70% topic B.”
  - Every topic is a mixture of words.
- For example, we could imagine a two-topic model of American news, with one topic for “politics” and one for “entertainment.”
- The most common words in the politics topic might be “President”, “Congress”, and “government”, while the entertainment topic may be made up of words such as “movies”, “television”, and “actor”.

Latent Dirichlet Allocation

- Mathematically, LDA outputs two objects, Γ and β
- Γ is a map assigning a probability of D documents to K topics
- β is a map assigning a probability of V words to K topics
- We can use the `cast_dtm()` function to convert this into a document-term matrix
- The syntax is: `cast_dtm(frequency_df, section_var, word_var,count_var)`

Example: LDA on our Wikipedia page

- Reminder: our tokenized Wikipedia data looked like this:

```
data ▷ head(5)
```

```
## # A tibble: 5 × 2
##   section word
##     <int> <chr>
## 1       3 fifa
## 2       3 world
## 3       3 cup
## 4       3 22nd
## 5       3 fifa
```

- The data can be converted to a document term matrix using `cast_dtm()`:

```
dtm ← data ▷ count(section, word) ▷ tidytext::cast_dtm(section, word, n)
```

- And then we can use `LDA` from the `topicmodels` package to run LDA:

```
library(topicmodels)
lda ← LDA(dtm, k = 2, control = list(seed = 1234))
```

Results

- Each word gets a β-coefficient for each topic, interpretable as the *probability* of each word belonging to a topic..

```
result_beta ← tidy(lda, matrix = "beta")
result_beta ▷ slice(29:40)
```

```
## # A tibble: 12 × 3
##    topic term            beta
##    <int> <chr>          <dbl>
##  1     1 muslim    1.14e-51
##  2     2 muslim    7.30e- 4
##  3     1 national  4.88e- 4
##  4     2 national  3.93e- 3
##  5     1 november  3.17e- 3
##  6     2 november  4.78e- 3
##  7     1 organized 2.33e-51
##  8     2 organized 7.30e- 4
##  9     1 qatar     7.44e- 3
## 10     2 qatar     2.28e- 2
## 11     1 rights    2.05e- 3
## 12     2 rights    3.96e- 3
```

Results

- We can use some data wrangling and `slice_max` to see which words belong to which topic:
    - For example, by looking at the *ratio* of probabilities
    - This tells us which words are most discriminating

- For topic 2, the most discriminating words are:

```
result_beta ▷
  pivot_wider(names_from = topic, valu
  filter(topic1 > .001 | topic2 > .001
  mutate(log_ratio = log2(topic2 / top
  slice_max(log_ratio, n=8)
```

```
## # A tibble: 8 × 4
##   term          topic1  topic2 log_ratio
##   <chr>          <dbl>   <dbl>     <dbl>
## 1 sale        3.61e-57 0.00109      178.
## 2 reportedly 6.66e-57 0.00109      177.
## 3 public      1.28e-56 0.00146      176.
## 4 gakpo       1.54e-56 0.00109      176.
## 5 rainbow     3.10e-56 0.00146      175.
## 6 wada        2.53e-56 0.00109      175.
## 7 flags       5.30e-56 0.00182      175.
## 8 banned      3.44e-56 0.00109      174.
```

Results

- For topic 1:

```
result_beta ▷
  pivot_wider(names_from = topic, values_from = beta, names_prefix = "topic") ▷
  filter(topic1 > .001 | topic2 > .001) ▷
  mutate(log_ratio = log2(topic2 / topic1)) ▷
  slice_min(log_ratio, n=10)
```

```
## # A tibble: 10 × 4
##    term       topic1   topic2 log_ratio
##    <chr>       <dbl>    <dbl>     <dbl>
##  1 morata    0.00120 4.85e-64     -201.
##  2 costa     0.00240 5.24e-62     -195.
##  3 rica      0.00200 2.82e-60     -189.
##  4 spain     0.00320 8.66e-60     -188.
##  5 germany   0.00400 1.13e-59     -188.
##  6 shootout  0.00120 1.83e-58     -182.
##  7 shot      0.00240 8.00e-58     -181.
##  8 kane      0.00120 9.64e-58     -180.
##  9 equalised 0.00160 4.12e-57     -178.
## 10 music     0.00200 5.49e-57     -178.
```

Recapitulation

- We had a first look at text mining
    - We started out by talking about **different formats** in which text can be organized
    - Subsequently, we performed **sentiment analysis** based on some common sentiment indices
- Then, we switched to frequency analysis and introduced the `tf-idf` metric as a measure of the relevance of a word/token
- We expanded our **understanding** of tokens to n-grams
- Finally, we looked at an example of topic modeling using **Latent Dirichlet Analysis**