Data Wrangling and Data Analysis
Text mining #1

DL Oberski

Department of Methodology & Statistics

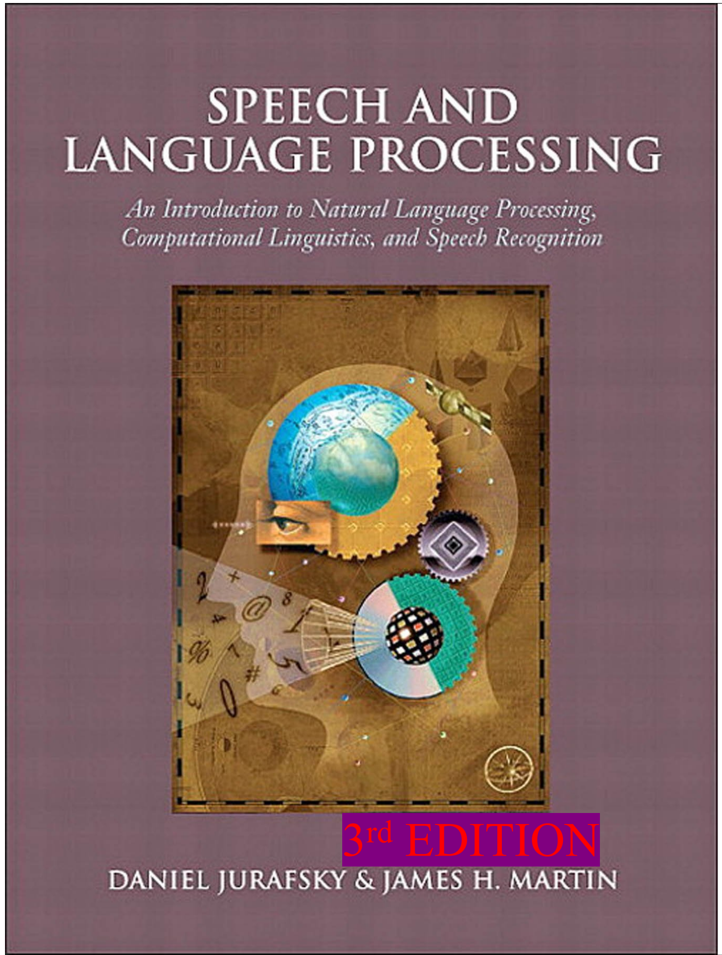
Utrecht University

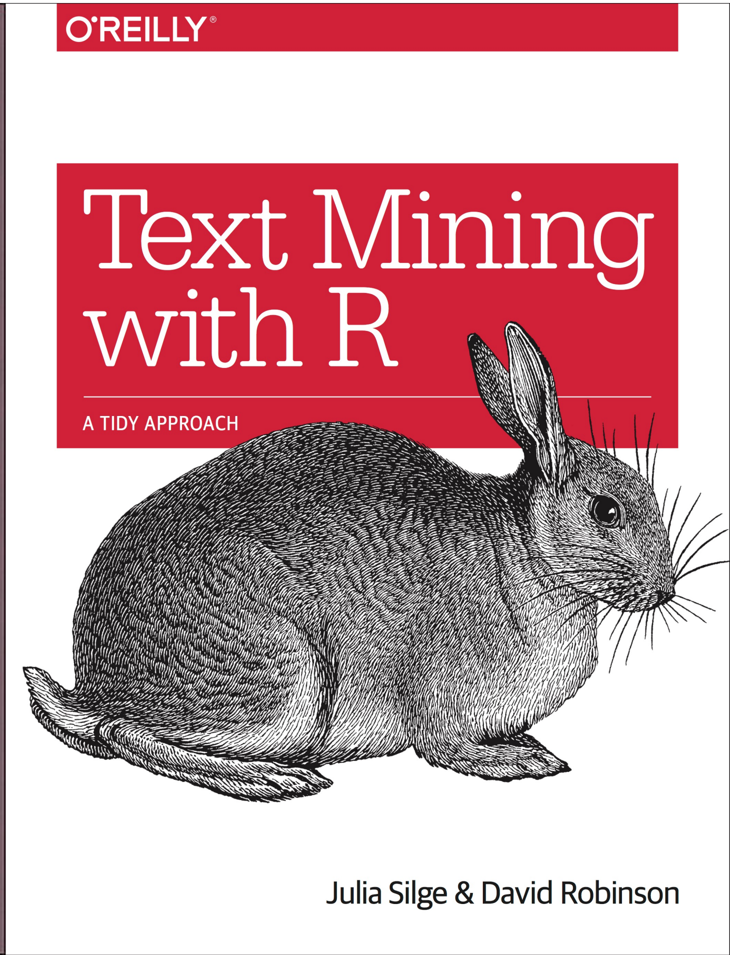
This week

- Day 1: Clustering #2: Model-based clustering
- **Day 2: Text mining #1**
- Day 3: Text mining #2

Readings about text mining

- Jurafsky & Martin (2021). *Speech and language processing (3rd ed draft)* https://web.stanford.edu/~jurafsky/slp3/
  - Sections
  - 2.1, 2.4, (regular expressions)
  - 6.2, 6.3, 6.4, 6.5, 6.8
- *Silge & Robinson (2021). Text mining with R: A tidy approach.* https://www.tidytextmining.com/
  - Chapter 3
- More accessible (?) intro to regular expressions:
  - **R4 data science** ch. 14
  - https://r4ds.had.co.nz/strings.html#matching-patterns-with-regular-expressions





Why learn text mining

Text data is everywhere:

- websites (e.g., news), social media (e.g., twitter), databases (e.g., doctors' notes), digital scans of printed materials, …
- Applications in industry: search, machine translation, sentiment analysis, question answering, …
- Applications in science: cognitive modelling, understanding bias in language, automated systematic literature reviews, …

Regular expressions

http://xkcd.com/208/

Regular expressions (regex)

- Powerful and very very useful tool for text (pre)processing
- Used in pretty much every pipeline involving text
- Typical applications:
  - Extracting numbers, emails, IP-addresses, etc.
  - Validating text inputs in GUIs
  - Reformatting annoying incorrect dates (everything not `yyyy-mm-dd`)
  - Scrubbing names and addresses for pseudonimization
- Powerful: e.g. J&M implement (part of) ELIZA (see link) in regex!
- Cryptic & takes a **lot** of practice!

Basic matching using `stringr`

```r
x <- c("apple", "banana", "pear")
str_view(x, "an")
```

```
apple

banana

pear
```

Basic matching

. matches any character

```
str_view(x, ".a.")
```

```
apple
banana
pear
```

Anchors

By default, regular expressions will match any part of a string. It's often useful to **anchor** the regular expression so that it matches from the start or end of the string. You can use:

- ^ to match the start of the string.
- $ to match the end of the string.

```
x <- c("apple", "banana", "pear")
str_view(x, "^a")
```

```
apple
banana
pear
```

Game

- Match the word “monarch”
- Match all 8-letter words

```
Barbados is moving from a parliamentary
constitutional monarchy under the hereditary
monarch of Barbados (currently Queen
Elizabeth II) to a parliamentary republic with
a ceremonial elected president as head of
state.
```

Examples

Regex	Matches any string that
`hello`	contains {hello}
`gray|grey`	contains {gray, grey}
`gr(a|e)y`	contains {gray, grey}
`gr[ae]y`	contains {gray, grey}
`b[aeiou]bble`	contains {babble, bebble, bibble, bobble, bubble}
`[b-chm-pP]at|ot`	contains {bat, cat, hat, mat, nat, oat, pat, Pat, ot}
`colou?r`	contains {color, colour}

More complicated examples

Regex	**Matches any string that**
`\d`	contains {0,1,2,3,4,5,6,7,8,9}
`1\d{10}`	contains an 11-digit string starting with a 1
`\d+(\.\d\d)?`	contains a positive integer or a floating point number with exactly two characters after the decimal point.
`^dog`	begins with "dog"
`dog$`	ends with "dog"
`^dog$`	is exactly "dog"

Matching and Extracting Data

- The function `str_detect()` returns a True/False depending on whether the string matches the regular expression
- If we actually want the matching strings to be extracted, we use `str_extract()`

[0-9]+

One or more digits

```
> library(stringr)
> s = 'My 2 favorite numbers are 19 and 42'
> str_extract_all(s, '[0-9]+')
[[1]]
[1] "2"  "19" "42"
```

Regular expression conclusion

- You have now *heard of* regular expressions
- And might have a basic idea of what you might do with them
- The only way to really learn, however, is **practice**
- Read the set texts (J&M ch 2 and/or R4DS ch 14)
- Next time you encounter some text you need to work on think "can I do this using regular expressions?"
- The answer is probably "yes".

Challenge problem: regex crossword

	[COBRA]+	(AB|O|OR)+
(.)+\1		
[^ABRC]+		

Why text mining

- Text data is everywhere – websites (e.g., news), social media (e.g., twitter), databases (e.g., doctors' notes), digital scans of printed materials, …
- A lot of world's data is in **unstructured** text format
- Applications in industry: search, machine translation, sentiment analysis, question answering, …
- Applications in science: cognitive modeling, understanding bias in language, automated systematic literature reviews, …

Basic idea of text mining

- Text is “unstructured data”
- How do we get to something structured that we can compute with?
- $\rightarrow$ text has to be **represented** somehow

Basic plan:

1. Represent the text as something that makes sense to a computer;
2. Continue life as normal.

Step 2 might involve prediction (“text classification”, “sentiment analysis”), visualization (e.g. word clouds), etc.

Example representations: “time series”

“And the evening and the morning were the third day.”

Token time series:

- Label each token 1-8 (including “.”)
- $1 \to 2 \to 3 \to 1 \to 2 \to 4 \to 5 \to 2 \to 6 \to 7 \to 8$

Part-of-speech time series:

- CON $\to$ DET $\to$NOUN $\to$CON $\to$DET $\to$NOUN $\to$VERB $\to$DET $\to$ADJ$\to$ NOUN

Etc.

Can do statistics as on any categorical time series data.

Example representations: bag-of-words

“And the evening and the morning were the third day.”

Word count:

```
## s_tok
##       .     and     day evening morning     the   third    were 
##       1       2       1       1       1       3       1       1 
```

Word proportions:

```
## s_tok
##      .    and    day evening morning    the  third   were 
## 0.0909 0.1818 0.0909 0.0909 0.0909 0.2727 0.0909 0.0909 
```

Etc.

Can do statistics as on any rectangular data set

Text representations

- Other examples: tf-idf, topic models, embeddings, transformers
- From very simple (word count) to very complex (encoder-decoder neural networks)
- (One of) the main foci of current research in natural language processing
- Can usually get quite far with very simple!

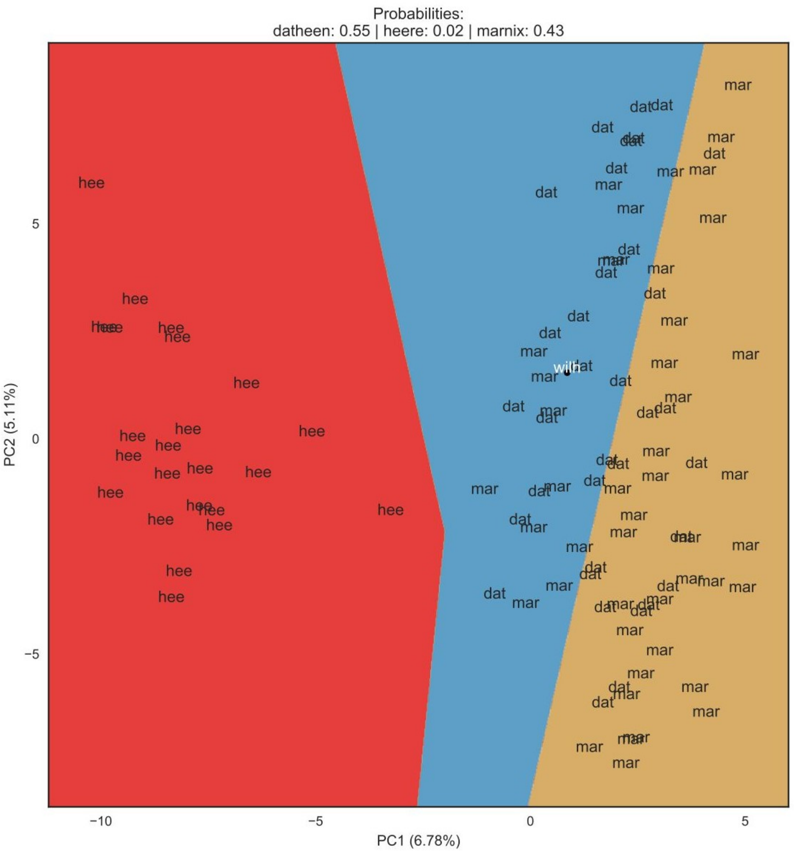
Language is hard

- Different things can mean more or less the same (“data science” vs. “statistics”)
- Context dependency (“You have very nice shoes”);
- Same words with different meanings (“to sanction”);
- Lexical ambiguity (“we saw her duck”)
- Irony, sarcasm (“You should swallow disinfectant”?)
- Figurative language (“He has a heart of stone”)
- Negation (“not good” vs. “good”), spelling variations, jargon, abbreviations
- All the above is different over languages, 99% of work is on English!

Language is hard

- We won’t solve linguistics today…
- In spite of the problems, text mining can be quite effective!

Who wrote the Wilhelmus?



https://dh2017.adho.org/abstracts/079/079.pdf

Which ICD-10 codes should I give this doctor's note?

Bovengenoemde patiënt was opgenomen op op de voor het specialisme **Cardiologie.**

Cardiovasculaire risicofactoren: Roken(-) Diabetes(-) Hypertensie(?) Hypercholesterolemie (?)

Anamnese. Om 18.30 pijn op de borst met uitstraling naar de linkerarm, zweten, misselijk. Ambulance gebeld en bij aansluiten monitor beeld van acuut onderwandinfarct. AMBU overdracht:.500mg aspegic iv, ticagrelor 180mg oraal, heparine, zofran eenmalig, 3x NTG spray. HD stabiel gebleven. . .Medicatie bij presentatie.Geen..

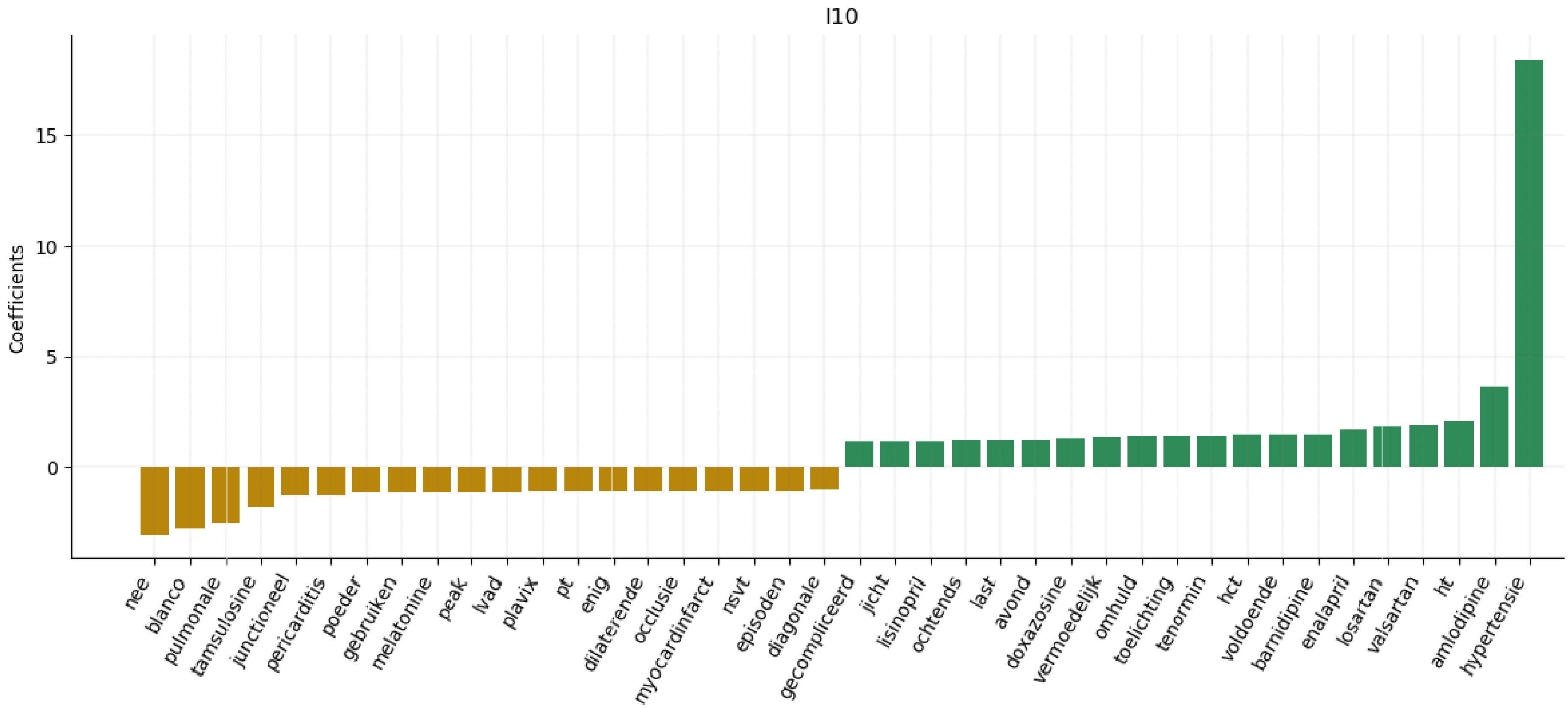
Lichamelijk onderzoek. Grauw, vegetatief, Halsvenen niet gestuwd. Cor s1 s2 geen souffles.Pulm schoon. Extr warm en slank .

Aanvullend onderzoek. AMBU ECG: Sinusritme, STEMI inferior III)II C/vermoedelijk RCA. Coronair angiografie. (…) .Conclusie angio: 1-vatslijden..PCI

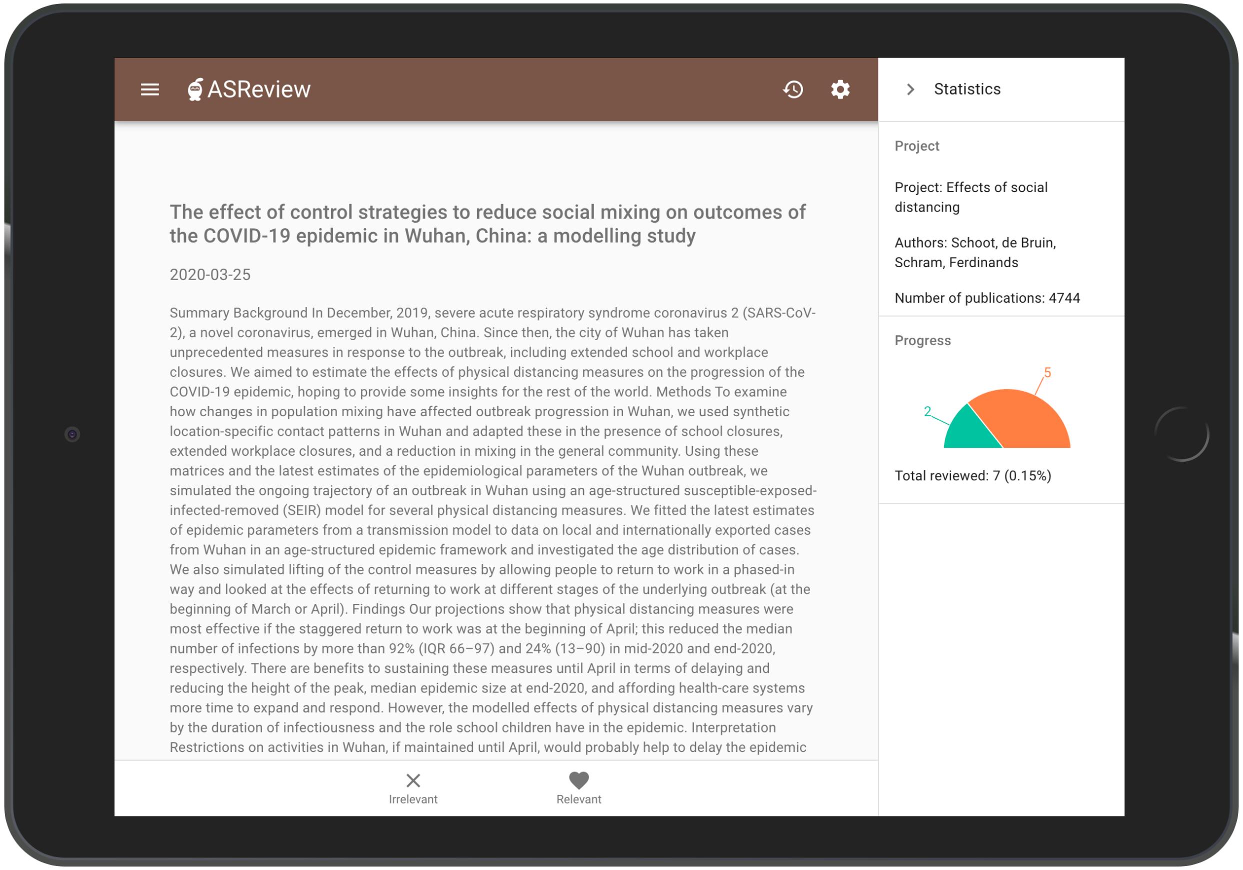
Conclusie en beleid Bovengenoemde jarige man, blanco cardiale voorgeschiedenis, werd gepresenteerd vanwege een STEMI inferior waarvoor een spoed PCI werd verricht van de mid-RCA. Er bestaan geen relevante nevenletsels. Hij kon na de procedure worden overgeplaatst naar de CCU van het . ..Dank voor de snelle overname. ..Medicatie bij overplaatsing. Acetylsalicylzuur dispertablet 80mg ; oraal; 1 x per dag 80 milligram ; .Ticagrelor tablet 90mg ; oraal; 2 x per dag 90 milligram ; .Metoprolol tablet 50mg ; oraal; 2 x per dag 25 milligram ; .Atorvastatine tablet 40mg (als ca-zout-3-water) ; oraal; 1 x per dag 40 milligram ;

Samenvatting Hoofddiagnose: STEMI inferior wv PCI RCA. Geen nevenletsels. Nevendiagnoses: geen. Complicaties: geen Ontslag naar: CCU .

Which ICD-10 codes should I give this doctor's note?



Which studies go in in my systematic review?



https://asreview.nl/

Main points for today

1. Workflow of text mining
2. Pre-processing text data
3. Word and document frequency
4. Sentiment analyisis
5. conclusion

Some useful definitions

- **Document**: a sequence of words and punctuation, following the grammatical rules of a language
- **Term**: usually a word, but can be a word-pair or phrase
- **Corpus**: a collection of documents
- **Lexicon**: set of all unique words in a corpus

Basic workflow for text analysis

1. Get some text
2. Organize text into ‘corpus’
3. Pre-process: e.g., remove punctuation, stopwords, lowercase
4. Create representation $\rightarrow$ actual dataset
5. Perform analysis as usual

Step 1. Get some text

Typical sources:

- Existing corpora, e.g. newspapers, libraries, etc. - examples: https://www.clarin.eu/portal, https://new.linguistlist.org/studentportal/
- Web scraping

```
library('rvest')
webpage <- read_html('https://en.wikipedia.org/wiki/COVID-19_pandemic')
```

- Social media APIs (e.g. https://rtweet.info/)
- …

Step 2. Organize text into ‘corpus’

Text corpus: typically stores the text as a **raw character string with metadata** and details stored with the text

Example: 50 Years of Pop Music Lyrics (Kaylin Walker)

```
## Rows: 5,100
## Columns: 6
## $ Rank   <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 1...
## $ Song   <chr> "wooly bully", "i cant help myself sugar pie honey bunch", "...
## $ Artist <chr> "sam the sham and the pharaohs", "four tops", "the rolling s...
## $ Year   <dbl> 1965, 1965, 1965, 1965, 1965, 1965, 1965, 1965, 1965, 1965, ...
## $ Lyrics <chr> "sam the sham miscellaneous wooly bully wooly bully sam the ...
## $ Source <dbl> 3, 1, 1, 1, 1, 1, 3, 5, 1, 3, 3, 1, 3, 1, 3, 3, 3, 3, 1, 1, ...
```

Step 2. Organize text into ‘corpus’

Text corpus: typically stores the text as a **raw character string** with metadata and details stored with the text

Example: 50 Years of Pop Music Lyrics (Kaylin Walker)

[1] “is this the real life is this just fantasy caught in a landslide no escape from reality open your eyes look up to the skies and see im just a poor boy i need no sympathy because im easy come easy go a little high little low anyway the wind blows doesnt really matter to me to memama just killed a man put a gun against his head pulled my trigger now hes dead mama life had just begun but now ive gone and thrown it all away mama ooo didnt mean to make you cry if im not back again this time tomorrow

Step 3. Preprocessing

“And the evning and the morning were the third day.”

Typical steps:

- Stemming (“running”$\rightarrow$“run”) or Lemmatization (“were”$\rightarrow$“is”)
- Lowercasing (“And”$\rightarrow$“and”)
- Stopword removal (“evning morning is third day.”)
- Punctuation removal (“evning morning is third day”)
- Number removal (“day 3”$\rightarrow$“day”)
- Spell correction (“evning”$\rightarrow$“evening”)
- Tokenization (“evening”, “morning”, “is”, “third”, “day”)

Not all of these are appropriate at all times!

Stemming

- Unifies variations in the text data:
  - e.g., ’walking’, ‘walks’, ‘walked’ $\rightarrow$ walk
- Inflectional stemming:
  - Remove plurals
  - Normalize verb tenses
  - Remove other affixes
- Stemming to root:
  - Reduce word to most basic element
  - More aggressive than inflictional
  - e.g., ‘denormalization’ $\rightarrow$ norm;
  - e.g., ‘Apply’, ‘applications’, ‘reapplied’ $\rightarrow$ apply

Tokenization with tidytext

function `unnest_tokens()` $\rightarrow$ one-term-per-row (automatically removes punctuation)

```
## # A tibble: 566 x 6
##     Rank Song        Artist                           Year Source token
##    <dbl> <chr>       <chr>                           <dbl>  <dbl> <chr>
##  1     1 uptown funk mark ronson featuring bruno mars  2015      1 this
##  2     1 uptown funk mark ronson featuring bruno mars  2015      1 hit
##  3     1 uptown funk mark ronson featuring bruno mars  2015      1 that
##  4     1 uptown funk mark ronson featuring bruno mars  2015      1 ice
##  5     1 uptown funk mark ronson featuring bruno mars  2015      1 cold
##  6     1 uptown funk mark ronson featuring bruno mars  2015      1 michelle
##  7     1 uptown funk mark ronson featuring bruno mars  2015      1 pfeiffer
##  8     1 uptown funk mark ronson featuring bruno mars  2015      1 that
##  9     1 uptown funk mark ronson featuring bruno mars  2015      1 white
## 10     1 uptown funk mark ronson featuring bruno mars  2015      1 gold
## # ... with 556 more rows
```

Removal of stop words - song lyrics

In `tidytext`: `anti_join(stop_words)` on `unnest_tokens()` object.

Still including stop words:

```
## # A tibble: 42,157 x 2
##    token     n
##    <chr> <int>
##  1 you   64606
##  2 i     56472
##  3 the   53451
##  4 to    35752
##  5 and   32555
##  6 me    31170
##  7 a     29282
##  8 it    25688
##  9 my    22821
## 10 in    18553
## # ... with 42,147 more rows
```

With stopwords removed

```
## # A tibble: 41,561 x 2
##    token     n
##    <chr> <int>
##  1 love  15283
##  2 im    14279
##  3 dont  11587
##  4 baby   9098
##  5 youre  6592
##  6 yeah   6259
##  7 time   5176
##  8 girl   4803
##  9 wanna  4767
## 10 gonna  4550
## # ... with 41,551 more rows
```

Also note the removed punctuation by `unnest_tokens`.

Step 4. Create representation $\longrightarrow$ actual dataset

Bag of words

- d1: "And God said, Let there be light: and there was light."
- d2: "And God saw the light, that it was good: and God divided the light from the darkness."
- d3: "And God called the light Day, and the darkness he called Night. And the evening and the morning were the first day."

"Document - Term matrix" (DTM)

	light	god	darkness	called	day	let	said	divided	good	saw	evening	first	morning	night
d1	2	1	0	0	0	1	1	0	0	0	0	0	0	0
d2	2	2	1	0	0	0	0	1	1	1	0	0	0	0
d3	1	1	1	2	2	0	0	0	0	0	1	1	1	1

DTM in R

```
lyricsCorpus <- Corpus(VectorSource(song_lyrics$Lyrics))
lyricsDTM <- DocumentTermMatrix(lyricsCorpus)
```

```
## <<DocumentTermMatrix (documents: 5100, terms: 41762)>>
## Non-/sparse entries: 473454/212512746
## Sparsity           : 100%
## Maximal term length: 46
## Weighting          : term frequency (tf)
## Sample             :
##       Terms
## Docs   all and dont know like love that the you your
##   2993   5  44    2    0    6    0    6  44   5    1
##   3319   6  18   18    7    7    1   20  44  30    3
##   3378   0  52    3    3    7    0    5  48  44    4
##   3551  13  26    5    4   12    1    2  38  30   11
##   3762   4  24   31    0    6    0    7  31  16    7
##   3840   7  24    3    2    8    2    4  39  22    1
##   3959  20  27   10   15   11    1    7  54  16    3
##   4249   6  29   17    7   14    4   12  32  23   12
##   4488  10  22    6    5   11    0    6  48  14    2
##   4571  10  22    6    5   11    0    6  48  14    2
```

Summarizing tokens per document

- Bag of words model:
  - Ignores word order
  - **Document-term matrix** (dtm): One-document-per-row and one-term-per-column
  - Cells can contain counts, proportions, or (more common) scaled proportions (tf-idf)
- The tidy text format and the dtm can be converted to and from one another:
  - `tidy()` turns a document-term matrix into a tidy data frame
  - `cast()` turns a tidy one-term-per-row data frame into a matrix

Word frequency

- Option: quantify how frequently a word occurs in a document (tf), and inspect most frequent words within a document
- However, many words that appear often do not seem very informative, even after removing stop words
- For example, words in the different books of the Harry Potter series:

```
## Warning in NextMethod(): number of items to replace
## replacement length
```

```
## # A tibble: 63,651 x 4
##    book                word         n total
##    <fct>               <chr>    <int> <int>
##  1 order_of_the_phoenix harry     3730 96777
##  2 goblet_of_fire      harry     2936 72663
##  3 deathly_hallows     harry     2770 73406
##  4 half_blood_prince   harry     2581 63098
##  5 prisoner_of_azkaban harry     1824 41188
##  6 chamber_of_secrets  harry     1503 33621
##  7 order_of_the_phoenix hermione  1220 96777
##  8 philosophers_stone  harry     1213 28585
##  9 order_of_the_phoenix ron       1189 96777
## 10 deathly_hallows     hermione  1077 73406
## # ... with 63,641 more rows
```

Word frequency: Zipf’s law

Most words occur rarely and only very few words occur frequently.

“Zipf’s law”: $\mathrm{tf}(\mathrm{rank}) \propto \frac{1}{\mathrm{rank}^c}$ [https://en.wikipedia.org/wiki/Zipf%27s_law]

“Term frequency-inverse document frequency” (tf-idf)

- IDEA 1: Add unimportant frequent words to the list of stop words, but some of these words might be more important in some documents than others
- IDEA 2: decrease the weight for commonly used words and increases the weight for words that are not used very much in a collection of documents
- IDEA 2 is called the **inverse document frequency** (idf):

$$\mathrm{idf}(\mathrm{term}) = \ln\left(\frac{n_{\mathrm{documents}}}{n_{\mathrm{documents\ containing\ term}}}\right)$$

- When idf is combined with tf, we get the $\mathrm{tf\text{-}idf}$, which is intended as importance of a word to a document in a corpus

tf “Document - Term matrix” (DTM)

Bag of words

- d1: “And God said, Let there be light: and there was light.”
- d2: “And God saw the light, that it was good: and God divided the light from the darkness.”
- d3: “And God called the light Day, and the darkness he called Night. And the evening and the morning were the first day.”

“Document - Term matrix” (DTM) (raw word counts)

	light	god	darkness	called	day	let	said	divided	good	saw	evening	first	morning	night
d1	2	1	0	0	0	1	1	0	0	0	0	0	0	0
d2	2	2	1	0	0	0	0	1	1	1	0	0	0	0
d3	1	1	1	2	2	0	0	0	0	0	1	1	1	1

tf-idf “Document - Term matrix” (DTM)

Bag of words

- d1: “And God said, Let there be light: and there was light.”
- d2: “And God saw the light, that it was good: and God divided the light from the darkness.”
- d3: “And God called the light Day, and the darkness he called Night. And the evening and the morning were the first day.”

“Document - Term matrix” (DTM) (tf-idf)

	light	god	darkness	called	day	let	said	divided	good	saw	evening	first	morning	night
d1	0	0	0.000	0.0	0.0	1.1	1.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
d2	0	0	0.405	0.0	0.0	0.0	0.0	1.1	1.1	1.1	0.0	0.0	0.0	0.0
d3	0	0	0.405	2.2	2.2	0.0	0.0	0.0	0.0	0.0	1.1	1.1	1.1	1.1

tf-idf in R

```
hp_words_count <- hp_words_count %>% bind_tf_idf(word, book, n)
```

```
## # A tibble: 63,651 x 6
##    book                 word          n     tf   idf tf_idf
##    <fct>                <chr>     <int>  <dbl> <dbl>  <dbl>
##  1 order_of_the_phoenix harry      3730 0.0385     0      0
##  2 goblet_of_fire       harry      2936 0.0404     0      0
##  3 deathly_hallows      harry      2770 0.0377     0      0
##  4 half_blood_prince    harry      2581 0.0409     0      0
##  5 prisoner_of_azkaban  harry      1824 0.0443     0      0
##  6 chamber_of_secrets   harry      1503 0.0447     0      0
##  7 order_of_the_phoenix hermione   1220 0.0126     0      0
##  8 philosophers_stone   harry      1213 0.0424     0      0
##  9 order_of_the_phoenix ron        1189 0.0123     0      0
## 10 deathly_hallows      hermione   1077 0.0147     0      0
## # ... with 63,641 more rows
```

Note: idf and thus tf-idf are zero for extremely common words

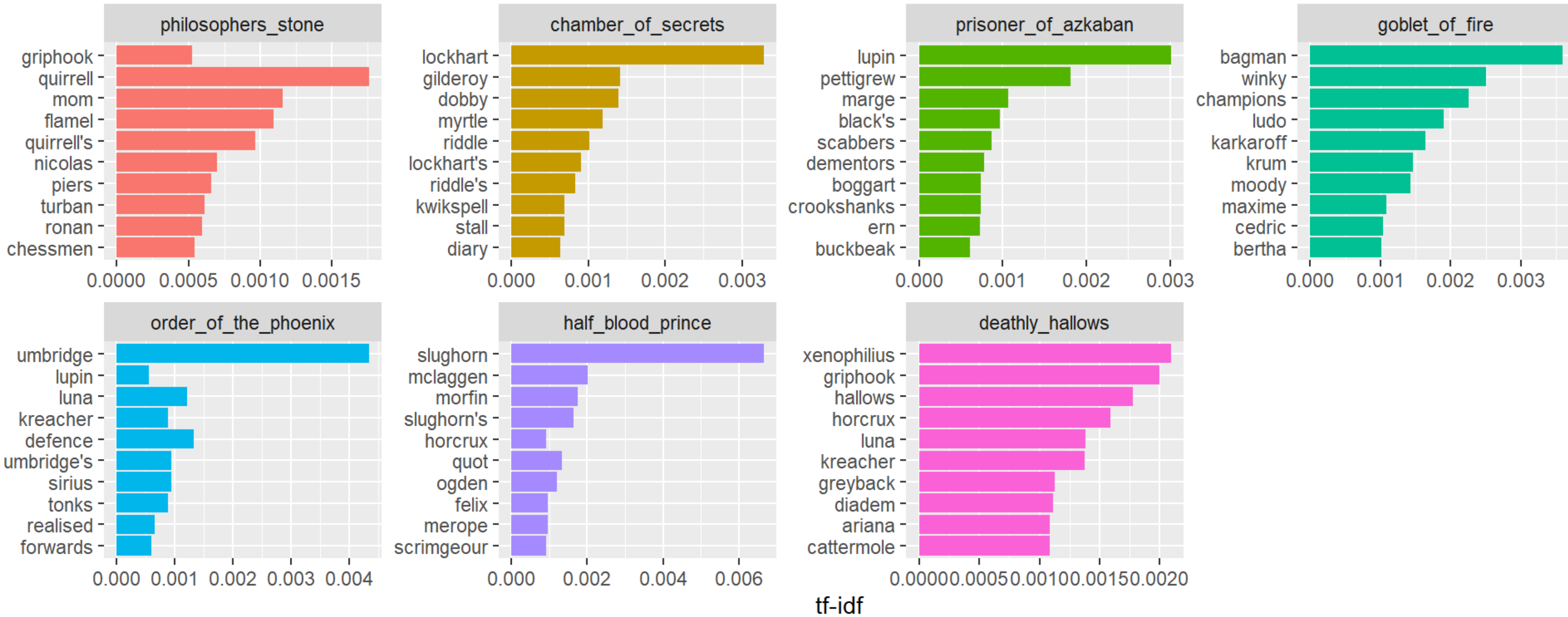
Word frequency: tf-idf in **R**

Inspecting terms with a high $tf-idf$:

```
## # A tibble: 63,651 x 6
##    book                 word            n      tf   idf  tf_idf
##    <fct>                <chr>       <int>   <dbl> <dbl>   <dbl>
##  1 half_blood_prince    slughorn      335 0.00531 1.25  0.00665
##  2 order_of_the_phoenix umbridge      496 0.00513 0.847 0.00434
##  3 goblet_of_fire       bagman        208 0.00286 1.25  0.00359
##  4 chamber_of_secrets   lockhart      197 0.00586 0.560 0.00328
##  5 prisoner_of_azkaban  lupin         369 0.00896 0.336 0.00301
##  6 goblet_of_fire       winky         145 0.00200 1.25  0.00250
##  7 goblet_of_fire       champions      84 0.00116 1.95  0.00225
##  8 deathly_hallows      xenophilius    79 0.00108 1.95  0.00209
##  9 half_blood_prince    mclaggen       65 0.00103 1.95  0.00200
## 10 deathly_hallows      griphook      117 0.00159 1.25  0.00200
## # ... with 63,641 more rows
```

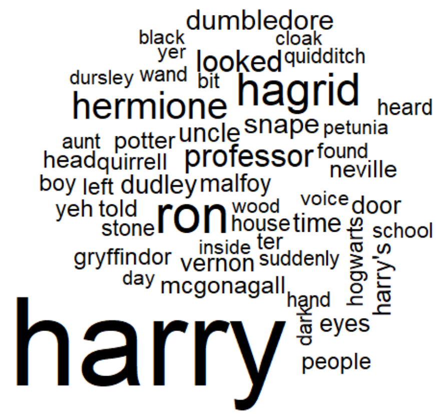
Word frequency

Visualized over all books:



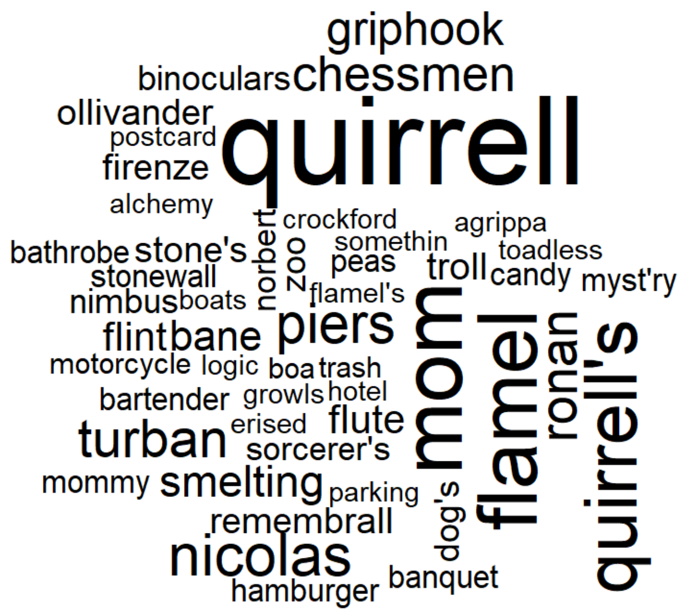
Word frequency

Frequent terms in the first Harry Potter book, the Philosophers stone:



Word frequency

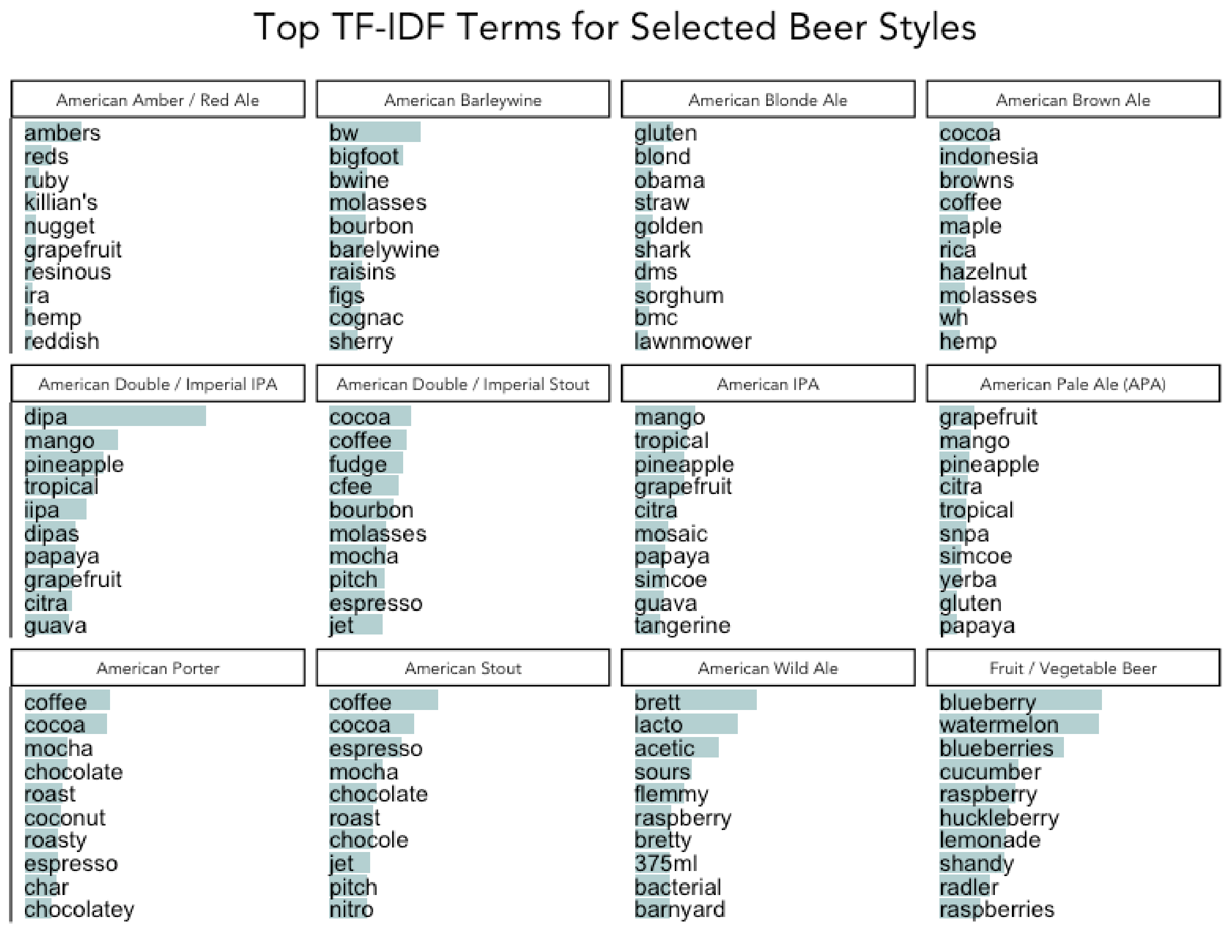
Distinctive terms (uinsg $tf - idf$ in the first Harry Potter book, the Philosophers stone:



Word frequency - another example

What are the most characteristic words used by reviewers to describe beers of different styles?

kaylinpavlik.com/tidy-text-beer



Step 4. Perform an analysis

Basic text summaries

- **Word frequency**
- Collocation: words appearing near each other
- Concordance: the instances and contexts of a given word or set of words
- **Dictionary tagging / sentiment analysis: determining the attitude of a speaker or writer**

More advanced techniques

- Document classification
- Corpora comparison (corpus: group of text documents)
- Language use over time
- Topic modelling: detecting clusters
- Natural language processing

Sentiment analysis

Try to extract and identify positive/negative valence from a text.

Basic idea:

$$\text{Sentiment} = \text{Total no. positive words} - \text{Total no. negative words}$$

- Use 'sentiment dictionaries' (lexicons) to assess a score (positive/negative) or emotion to each term;
- In `tidytext`: `AFINN`, `bing`, `nrc`;
- There are also domain specific sentiment lexicons, for example the Loughran and McDonald dictionary of financial sentiment terms;
- More advanced methods: use classification to predict sentiment from text (e.g. tf-idf).

Sentiment analysis - `AFINN`

`AFINN` lexicon (Finn Årup Nielsen):

- assigns words with a score that runs between -5 and 5, with negative scores indicating negative sentiment and positive scores indicating positive sentiment
- terms manually labelled for valence by Finn Årup Nielsen between 2009 and 2011.
- Specifically created for sentiment analysis of microblogs such as Twitter

```
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 - `bing`

`bing` lexicon (Bing Liu and collaborators):

- categorizes words into positive and negative categories
- Developed for mining and summarizing customer reviews
- First, adjective words were identified using a natural language processing method. Second, for each opinion word, semantic orientation was determined

```
## # 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
```

Sentiment analysis - **nrc**

`nrc` lexicon (Saif Mohammad and Peter Turney):

- categorizes words into categories of positive, negative, anger, anticipation, disgust, fear, joy, sadness, surprise, and trust
- annotations were manually done by crowdsourcing

```
## # 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
```

Example NRC

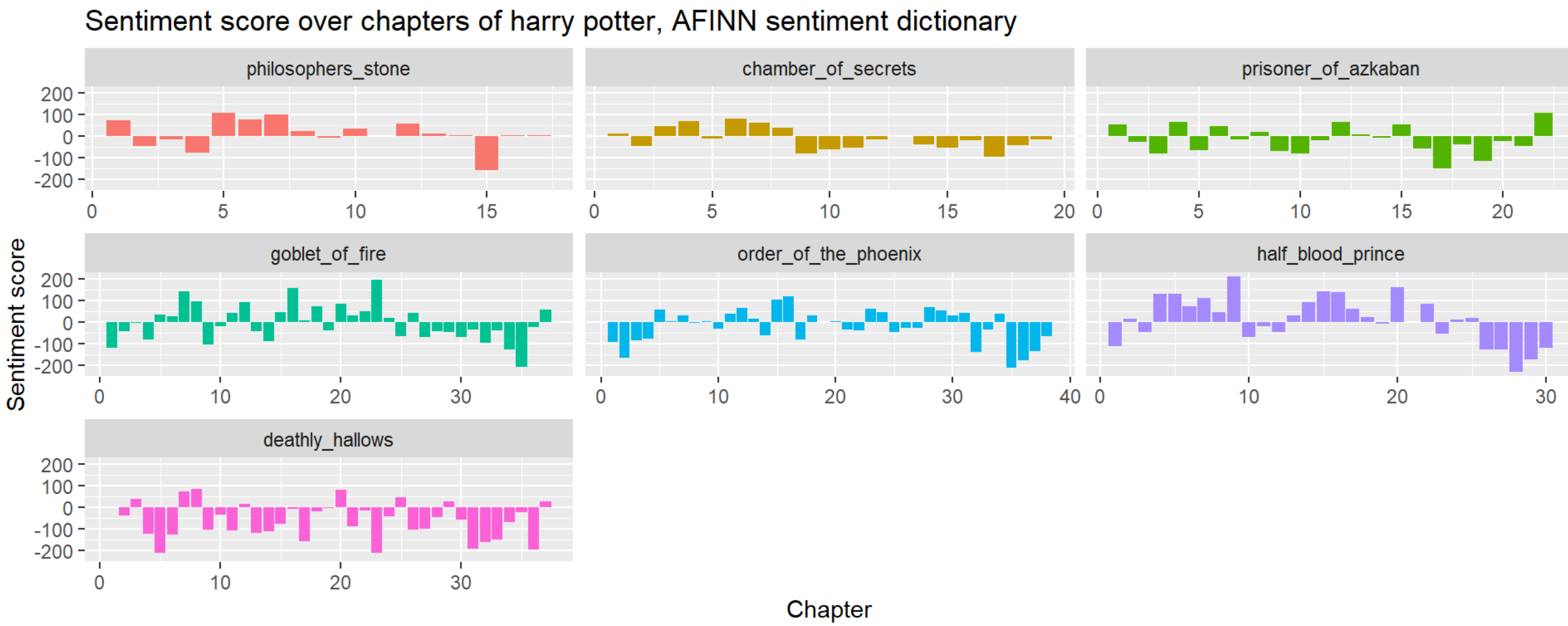
Most common joy words in Harry Potter

```
## # A tibble: 440 x 2
##    word          n
##    <chr>     <int>
##  1 good       1065
##  2 found       614
##  3 ministry    576
##  4 feeling     391
##  5 magical     380
##  6 white       331
##  7 green       294
##  8 mother      284
##  9 smile       244
## 10 hope        234
## # ... with 430 more rows
```

Most common fear words in Harry Potter

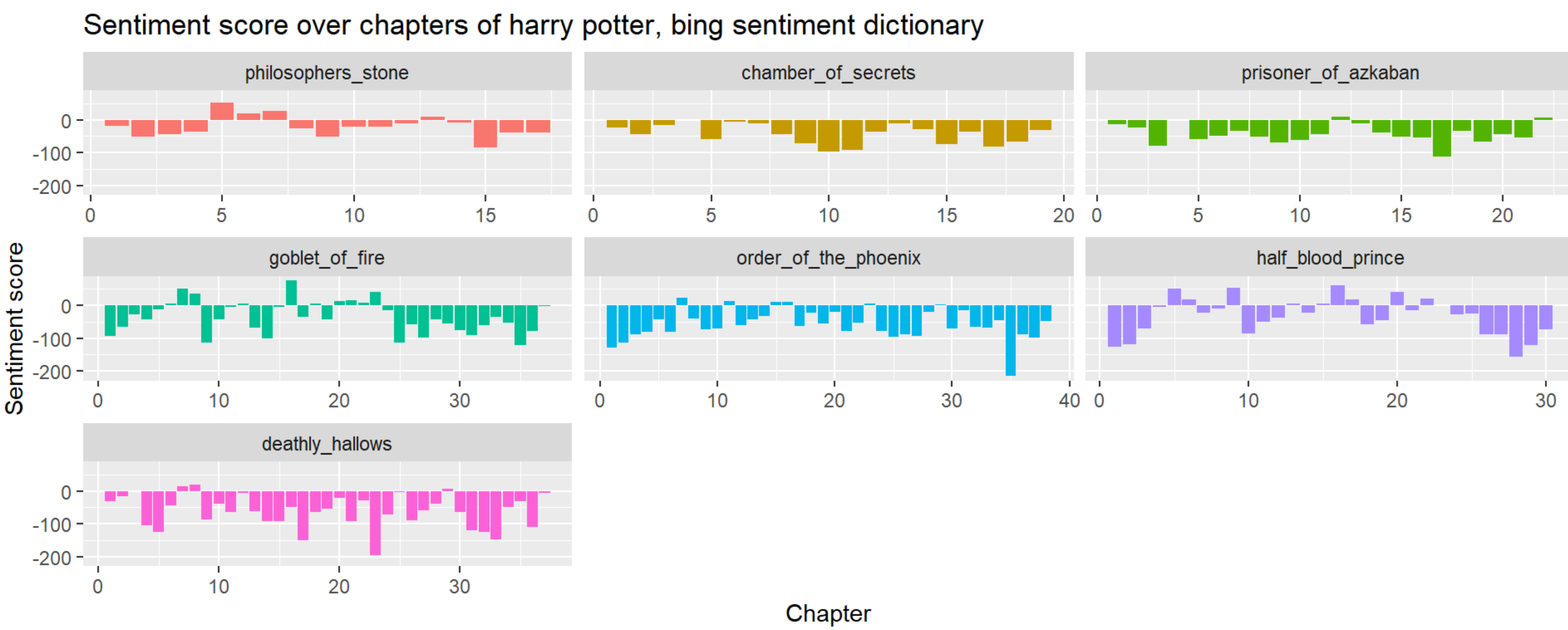
```
## # A tibble: 888 x 2
##    word         n
##    <chr>    <int>
##  1 death      757
##  2 feeling    391
##  3 fire       388
##  4 crouch     297
##  5 shaking    277
##  6 scar       276
##  7 mad        269
##  8 kill       267
##  9 elf        259
## 10 watch      256
## # ... with 878 more rows
```

Example AFINN

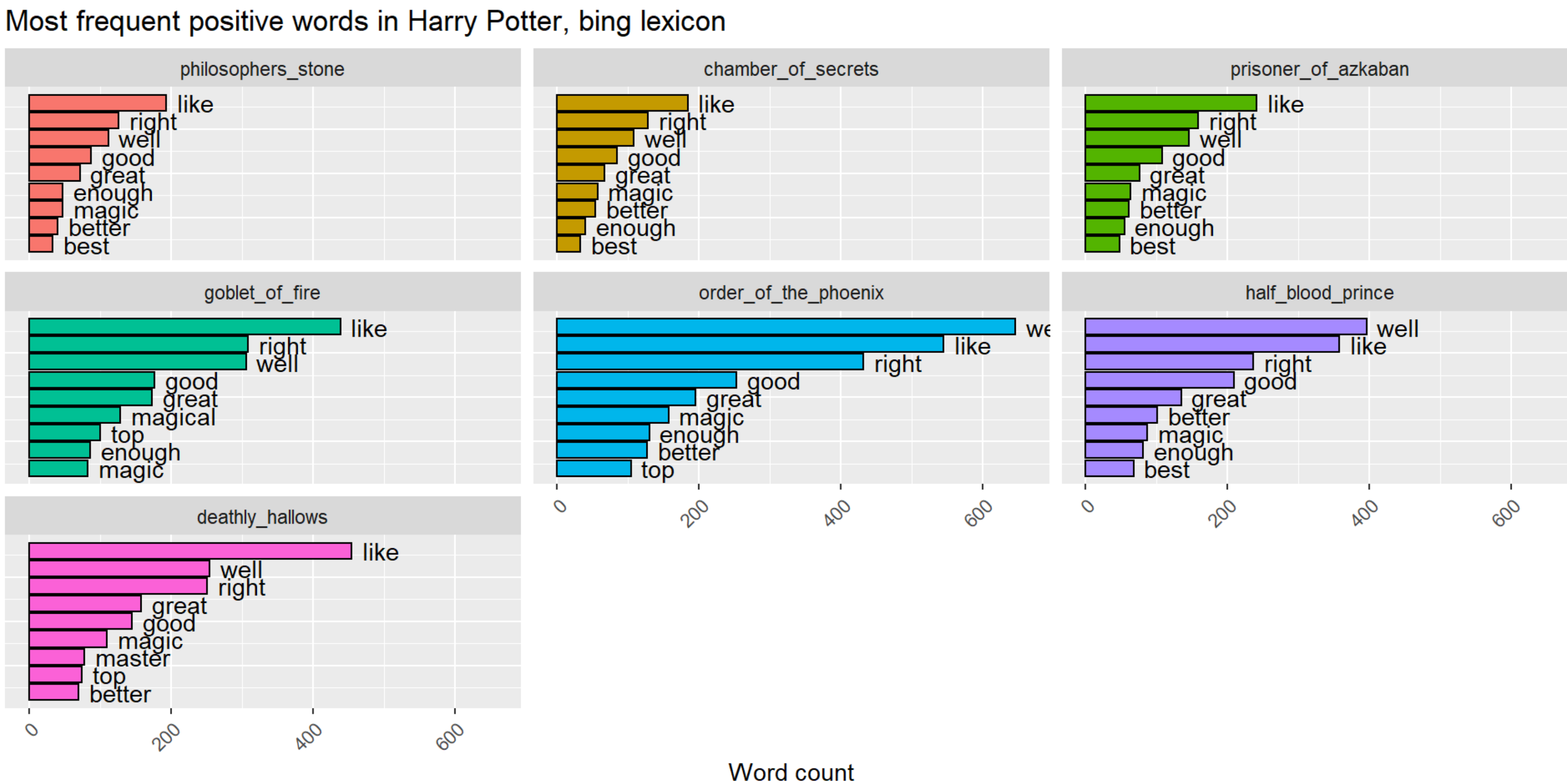


Plot of novel four to six changes towards a negative sentiment towards the end, while the seventh novel has a quite negative sentiment overall.

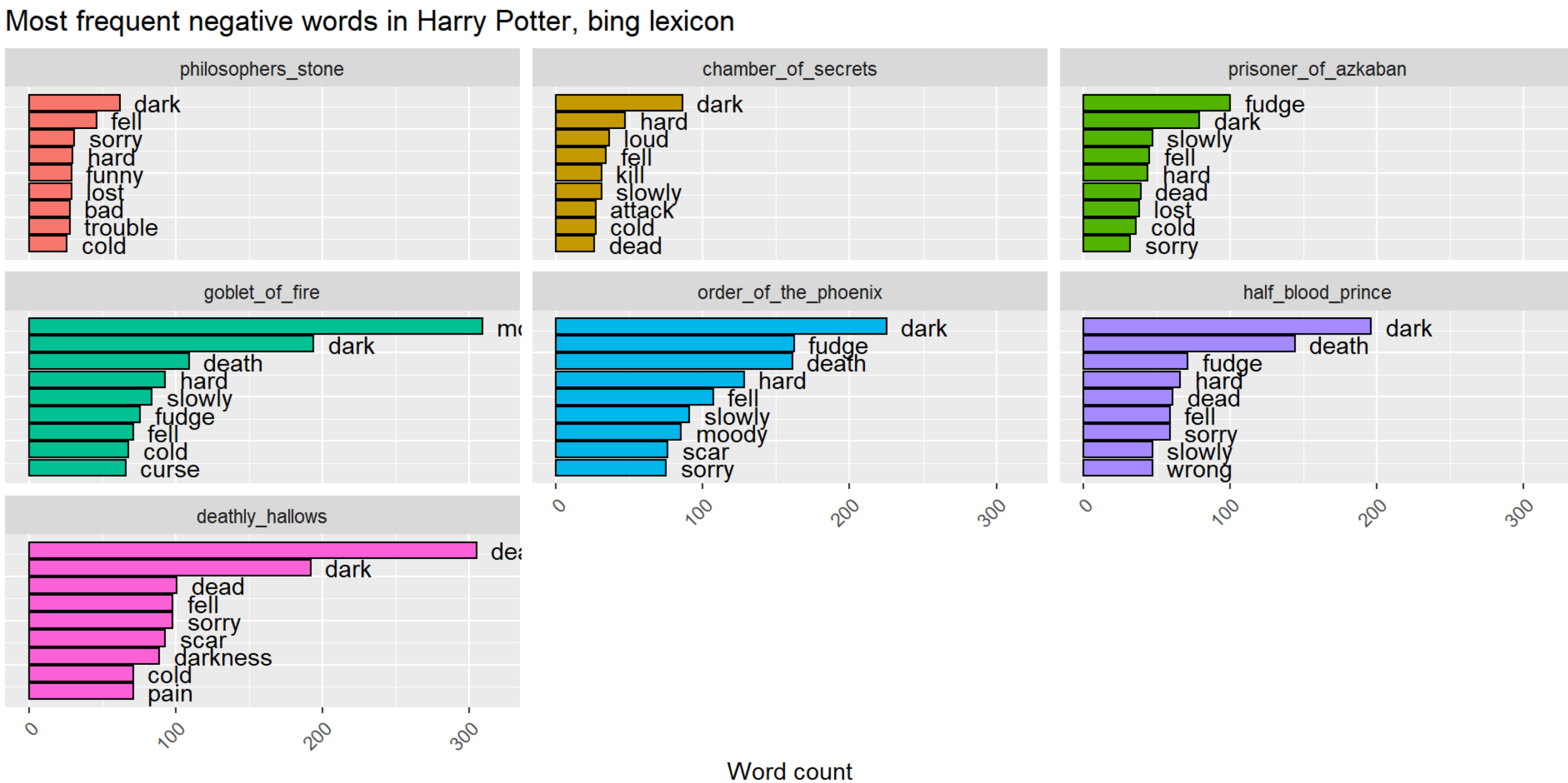
Example bing



Top words contributing to a positive sentiment



Top words contributing to a negative sentiment

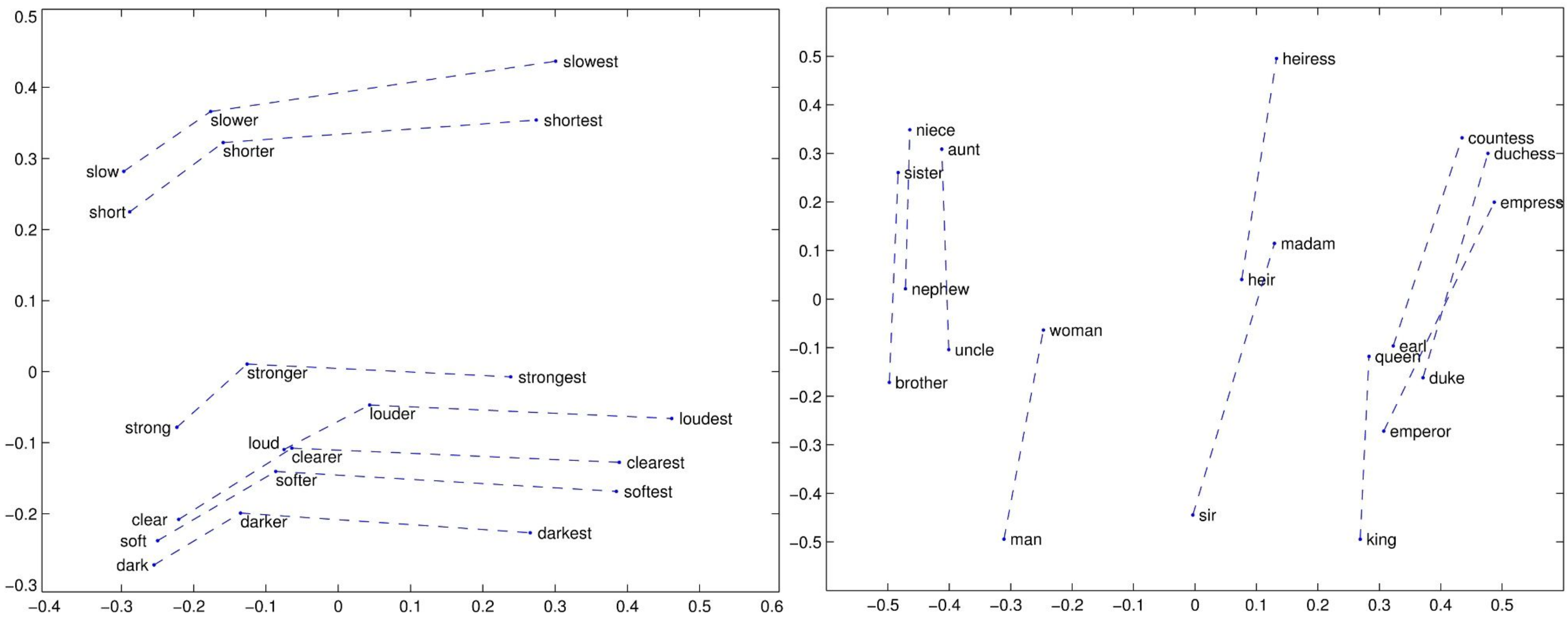


Sentiment analysis

Hurdles:

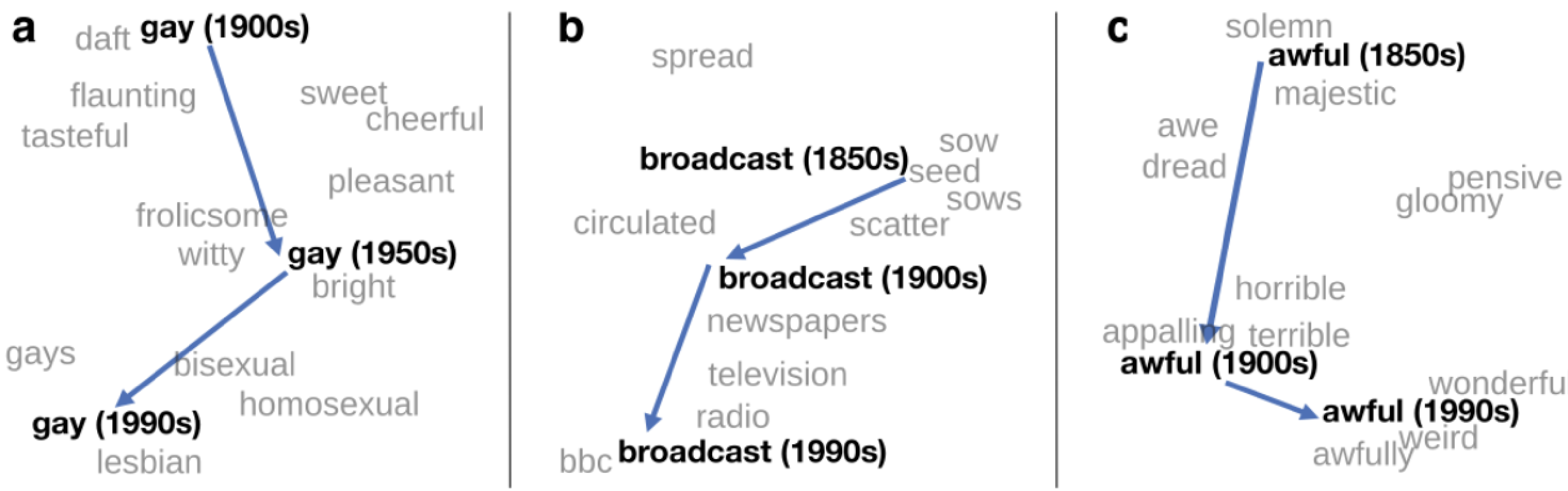
- Consider the sentence ‘I am not happy’. How would this be scored using sentiment analysis as presented?
  - When only investigating one word at the time, qualifiers before a word are not taken into consideration
- Sentiment analysis is language and dictionary dependent. If we would want to label a Dutch text, we are dependent on the availability of a Dutch sentiment lexicon, or have to create one ourselves
- Size: one sentence or small paragraph (like tweets or customer reviews) often have a clear sementic orientation. Long texts often contain positive and negative sentiments, which average out to about zero. Hence, more suited for short texts

Word embeddings: basic idea



Source: https://ruder.io/secret-word2vec/

Word embeddings: change in meaning over time



Source: Hamilton et al. (2016) http://doi.org/10.18653/v1/P16-1141

Conclusion

- The basic **problem** of text mining is that text is not a neat data set
- The **solution** to this problem is preprocessing and representation
- $\rightarrow$ preprocessing & representation determine outcome and its usefulness!
- Harry Potter example:
    - Preprocessing: lowercasing, stopword removal, (what else?)
    - Representation: tf-idf bag-of-words
- Often these very simple choices give a very reasonable baseline,
- surprising amount of insight, even though computers don't know language, **but**
- Many other choices possible… it matters **a lot** $\rightarrow$ plenty left to learn!