

Computational Sociology

Introduction to Natural Language Processing

Dr. Thomas Davidson

Rutgers University

March 1, 2021

Plan

1. Course updates
2. What is NLP?
3. Working with texts
4. Vector representations of texts
5. Document-similarity measures

Course updates

- ▶ Homework 2 and project proposal due Friday (3/5) at 5pm
 - ▶ Submit proposal using form (see #general on Slack)
 - ▶ Submit homework via Github

Introduction to NLP

What is natural language processing?

- ▶ Three components of NLP*:
 - ▶ Natural language / “text as data”
 - ▶ A corpus of text (e.g. books, reviews, tweets, e-mails)
 - ▶ (Computational) linguistics
 - ▶ Linguistic theory to guide analysis and computational approaches to handle data
 - ▶ Statistics
 - ▶ Statistical methods to make inferences

*Not *that* NLP: https://en.wikipedia.org/wiki/Neuro-linguistic_programming

Introduction to NLP

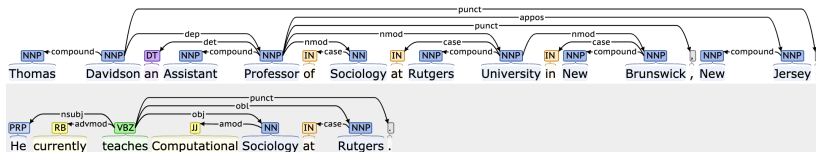
NLP tasks: Part-of-speech tagging

Thomas Davidson an Assistant Professor of Sociology at Rutgers University in New Brunswick , New Jersey .
He currently teaches Computational Sociology at Rutgers .

Examples created using <https://corenlp.run/>

Introduction to NLP

NLP tasks: Dependency-parsing



Examples created using <https://corenlp.run/>

Introduction to NLP

NLP tasks: Co-reference resolution

Thomas Davidson an Assistant Professor of Sociology at CorefEntity8 Rutgers University in New Brunswick , New Jersey .

He currently teaches Computational Sociology at CorefEntity8 Rutgers .

Examples created using <https://corenlp.run/>

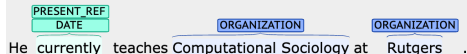
Introduction to NLP

NLP tasks: Named-entity recognition

Thomas Davidson an Assistant Professor of Sociology at Rutgers University in New Brunswick , New Jersey .



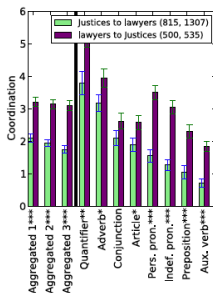
He currently teaches Computational Sociology at Rutgers .



Examples created using <https://corenlp.run/>

Introduction to NLP

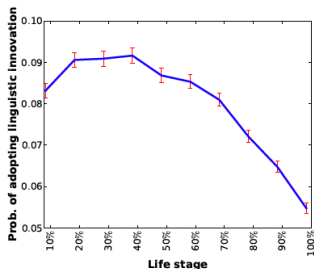
Applications: Power dynamics



Danescu-Niculescu-Mizil, Cristian, Lillian Lee, Bo Pang, and Jon Kleinberg. 2012. "Echoes of Power: Language Effects and Power Differences in Social Interaction." In Proceedings of the 21st International Conference on World Wide Web, 699–708. ACM. <http://dl.acm.org/citation.cfm?id=2187931>.

Introduction to NLP

Applications: Identity and group membership



(c) Adoption of lexical innovations

Danescu-Niculescu-Mizil, Cristian, Robert West, Dan Jurafsky, Jure Leskovec, and Christopher Potts. 2013. "No Country for Old Members: User Lifecycle and Linguistic Change in Online Communities." In Proceedings of the 22nd International Conference on World Wide Web, 307–18. ACM. <http://dl.acm.org/citation.cfm?id=2488416>.

Introduction to NLP

Applications: Trust and betrayal

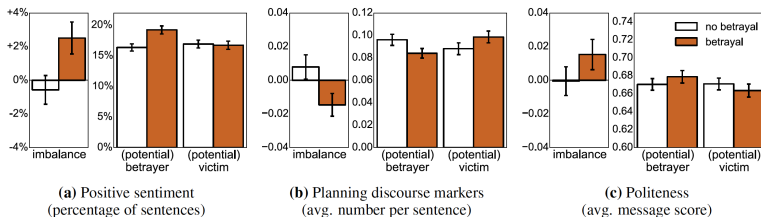


Figure 3: Friendships that will end in betrayal are imbalanced. The eventual betrayer is more positive, more polite, but plans less than the victim. The white bars correspond to matched lasting friendships, where the roles of potential betrayer and victim are arbitrarily assigned; in these cases, the imbalances disappear. Error bars mark bootstrapped standard errors (Efron, 1979).

Niculae, Vlad, Srijan Kumar, Jordan Boyd-Graber, and Cristian Danescu-Niculescu-Mizil. 2015. "Linguistic Harbingers of Betrayal: A Case Study on an Online Strategy Game." In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing. Beijing, China: ACL. <http://arxiv.org/abs/1506.04744>.

Introduction to NLP

Text as data

Table 1 Four principles of quantitative text analysis

-
- (1) All quantitative models of language are wrong—but some are useful.
 - (2) Quantitative methods for text amplify resources and augment humans.
 - (3) There is no globally best method for automated text analysis.
 - (4) Validate, Validate, Validate.
-

► Justin Grimmer and Brandon Stewart, 2013

Introduction to NLP

“Computational approaches are sometimes less subtle and deep than the reading of a skillful analyst, who interprets text in context. Nevertheless, . . . recent advances in NLP and ML are being used to enhance qualitative analysis in two ways. First, supervised ML prediction tools can “learn” and reliably extend many sociologically interesting textual classifications to massive text samples far beyond human capacity to read, curate, and code. Second, unsupervised ML approaches can “discover” unnoticed, surprising regularities in these massive samples of text that may merit sociological consideration and theorization.” - James Evans and Pedro Aceves, 2016

Introduction to NLP

NLP class timeline

- ▶ Week 7 (today)
 - ▶ Pre-processing, bag-of-words, and the vector-space model
- ▶ Week 8
 - ▶ Word embeddings
- ▶ Week 9
 - ▶ Topic models
- ▶ Week 11
 - ▶ Supervised text classification

Working with text

Pre-processing

- ▶ There are several steps we need to take to “clean” or “pre-process” texts for analysis
 - ▶ Tokenization
 - ▶ Stemming/lemmatization
 - ▶ Stop-word removal
 - ▶ Handling punctuation and special characters

Working with text

Tokenization

- ▶ Tokenization is the process of splitting a document into words
 - ▶ e.g. “Cognito, ergo sum” \Rightarrow (“Cognito,”, “ergo”, “sum”)
- ▶ In English this is pretty trivial, we just split using white-space
- ▶ Tokenization is more difficult in languages like Mandarin
 - ▶ It requires more complex parsing to understand grammatical structures

Working with text

Stemming/lemmatization

- ▶ We often want to reduce sparsity by reducing words to a common root
 - ▶ e.g. ("school", schools", "schooling", "schooled") ⇒ "school"
- ▶ Stemming is a simple, heuristic-based approach
- ▶ Lemmatization is a more rigorous approach based on morphology, but is more computationally-intensive and often unnecessary

Working with text

Stop-word removal

- ▶ Stop-words are frequently occurring words that are often removed
- ▶ The intuition is that they add little meaning and do not help us to distinguish between documents
 - ▶ e.g. Virtually all texts in English will contain the words “and”, “the”, “of”, etc.
- ▶ Most NLP packages have lists of stop-words to easily facilitate removal.

Working with text

Handling punctuation and special characters

- ▶ In many cases we may want to remove punctuation and other special characters (e.g. HTML, unicode)
 - ▶ This is often done using regular expressions
 - ▶ Words are typically set to lowercase

Working with text

Pre-process with caution!

- ▶ Researchers often apply these techniques before starting an analysis, but it may affect our results*
 - ▶ There is no one-size-fits-all solution, so think carefully before removing anything
 - ▶ It's often useful to experiment to see if pre-processing steps affect results

Working with text

Word counts

- ▶ Now we have done some pre-processing, one of the most basic ways we can start to analyze texts is by counting the frequency of words.

- ▶ e.g. "I think, therefore I am" \Rightarrow

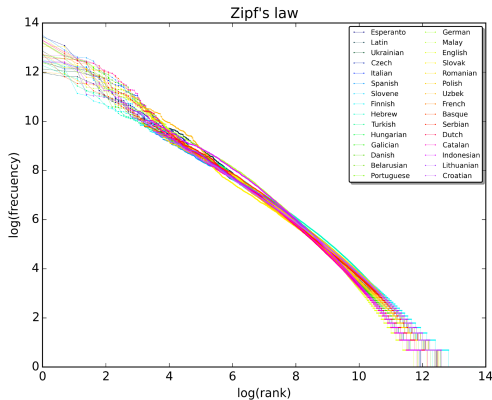
Word	Count
I	2
think	1
therefore	1
am	1

Working with text

Frequency distributions

- ▶ *Zipf's law*: A word's frequency is inversely proportional to its rank order in the frequency distribution.
 - ▶ “the” is the most common word in the English language, accounting for 7% of all words in the *Brown Corpus of American English*
 - ▶ “and” and “of” compete for second place, each accounting for ~3.5% of words in the corpus
 - ▶ The most frequent 135 words account for approximately half the 1 million words in the corpus
 - ▶ Around 50,000 words, representing half the total unique words in the corpus, are *hapax legomena*, words which only occur once

Working with text: Zipf's law



A plot of the rank versus frequency for the first 10 million words in 30 Wikipedias (dumps from October 2015) in a log-log scale (Source: Wikipedia).

Working with text

Bag-of-words

- ▶ Documents are often treated as “bags of words”, i.e. we treat a document as a collection of words without retaining information about the order
 - ▶ e.g. “This is a document” \Rightarrow (“document”, “This”, “a”, “is”)

Working with text

Example: Loading data

```
library(tidyverse)
library(tidytext)
library(gutenbergr)

ef <- gutenberg_download(41360) # Download Elementary Forms
cm <- gutenberg_download(61) # Download Communist Manifesto

ef$title <- "Elementary Forms"
cm$title <- "Communist Manifesto"

texts <- bind_rows(ef, cm)
```

Working with text

In this example, each text is represented as a table, where the first column is the ID in the Project Gutenberg database and the text field contains each sentence as a new row.

```
print(tail(texts$text))
```

```
## [1] "Let the ruling classes tremble at a Communistic revolution."  
## [2] "The proletarians have nothing to lose but their chains."  
## [3] "They have a world to win."  
## [4] ""  
## [5] ""  
## [6] "          WORKING MEN OF ALL COUNTRIES, UNITE!"
```

Working with text

Tokenizing using tidytext

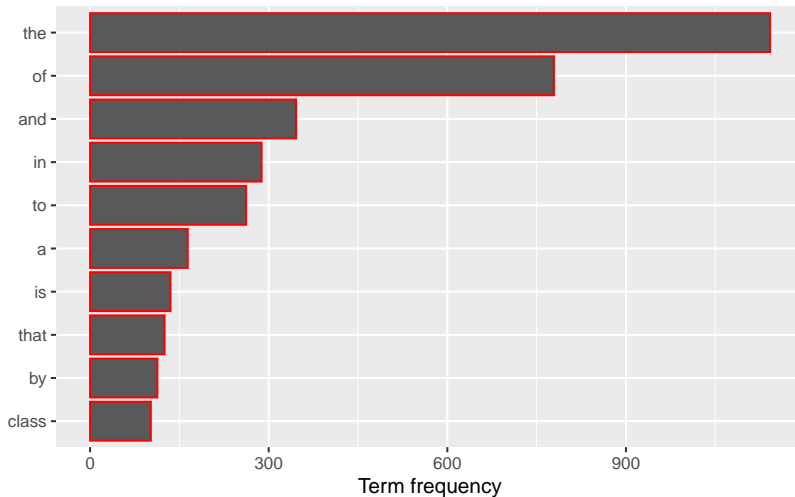
We are going to be using the tidytext package to conduct our analyses. The `unnest_tokens` function is used to tokenize the text, resulting in a table containing the original book ID and each token as a separate row.

```
tidy.text <- texts %>% unnest_tokens(word, text)
tail(tidy.text$word)
```

```
## [1] "working" "men" "of" "all" "countries" "uni
```

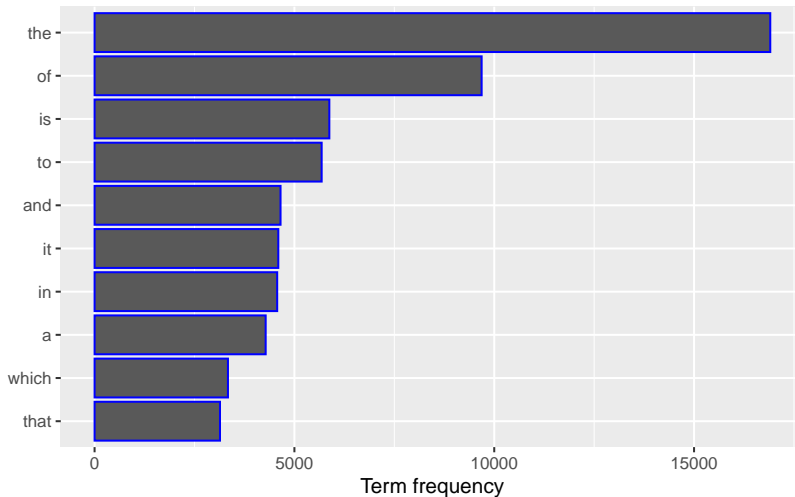
Working with text

10 most frequent terms in The Communist Manifesto



Working with text

10 most frequent terms in The Elementary Forms of Religious Life



Working with text

Removing stopwords

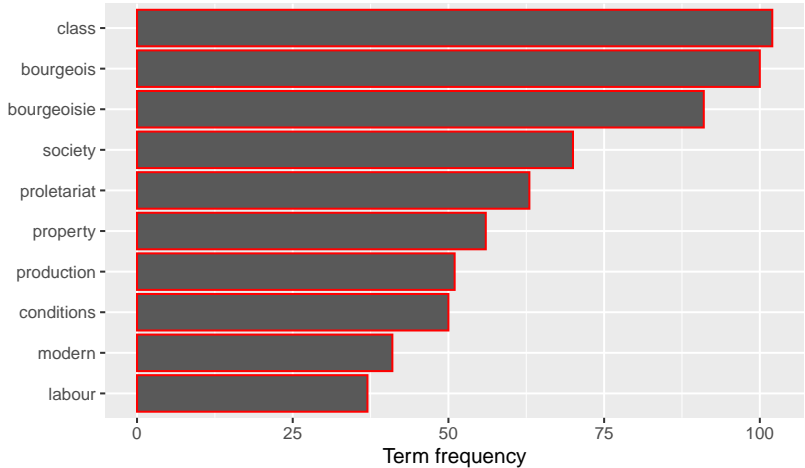
We can load a corpus of stop words contained in `tidytext` and use `anti_join` to filter our texts. This retains all records without a match in stopwords.

```
data(stop_words)

tidy.text <- tidy.text %>%
  anti_join(stop_words)
```

Working with text

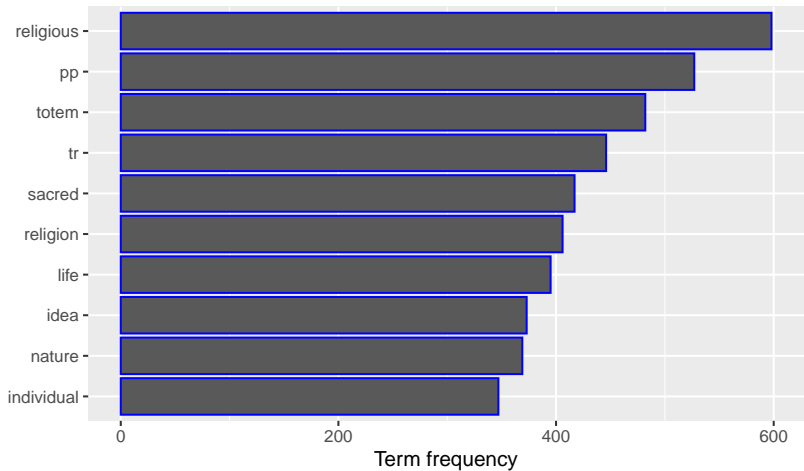
10 most frequent terms in The Communist Manifesto



Stopwords removed

Working with text

10 most frequent terms in The Elementary Forms of Religious Life



Stopwords removed

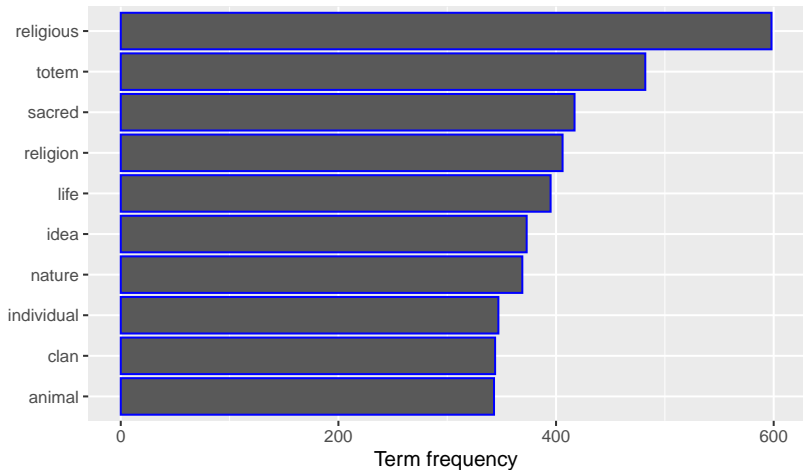
Working with text

Exercise: Removing other junk

The last example shows how there is still some “junk” in the Durkheim text.

Working with text

10 most frequent terms in The Elementary Forms of Religious Life



Stopwords removed+

Working with text

Stemming

We can stem the terms using a function from the package `SnowballC`, which is a wrapper for a commonly used stemmer called the Porter Stemmer, written in C.

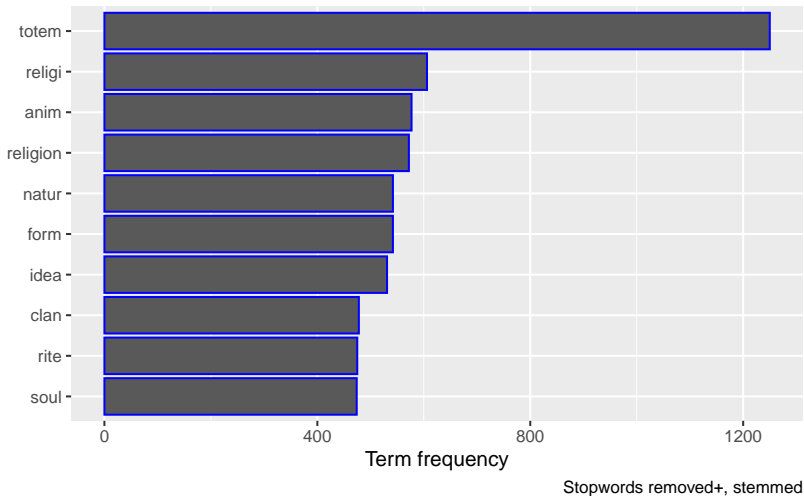
```
library(SnowballC)
```

```
tidy.text <- tidy.text %>% mutate_at("word", funs(wordStem(.), language = "english"))
```

Stemmer solution from https://cbail.github.io/SICSS_Basic_Text_Analysis.html. See for more info on stemming and lemmatizing in R.

Working with text

10 most frequent terms in The Elementary Forms of Religious Life



Working with text: Zipf's law

Let's get counts of words across both texts to analyze their distribution.

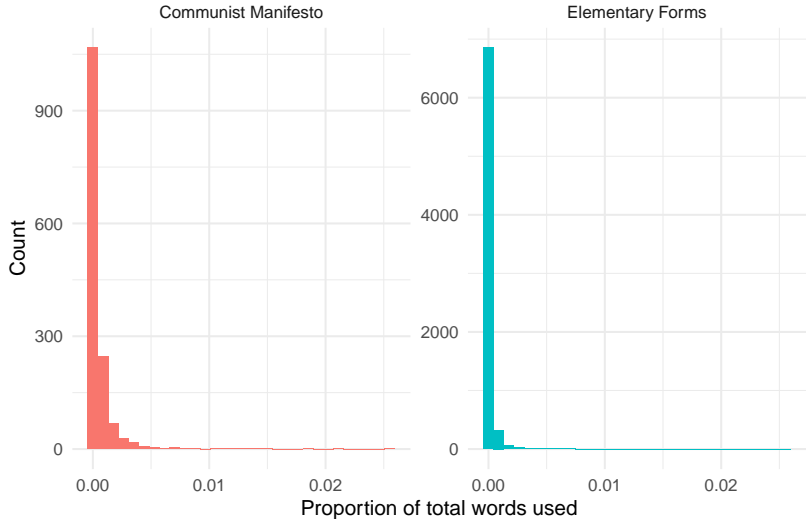
```
# Count words by text
text.words <- tidy.text %>% count(title, word, sort = TRUE)

# Get total number of words in each text
total.words <- text.words %>% group_by(title) %>%
  summarize(total = sum(n))

# Merge
words <- left_join(text.words, total.words)
head(words)
```

```
## # A tibble: 6 x 4
##   title          word      n total
##   <chr>         <chr>  <int> <int>
## 1 Elementary Forms totem    1250 78851
## 2 Elementary Forms religi    606 78851
## 3 Elementary Forms anim     577 78851
```

Working with text: Zipf's law

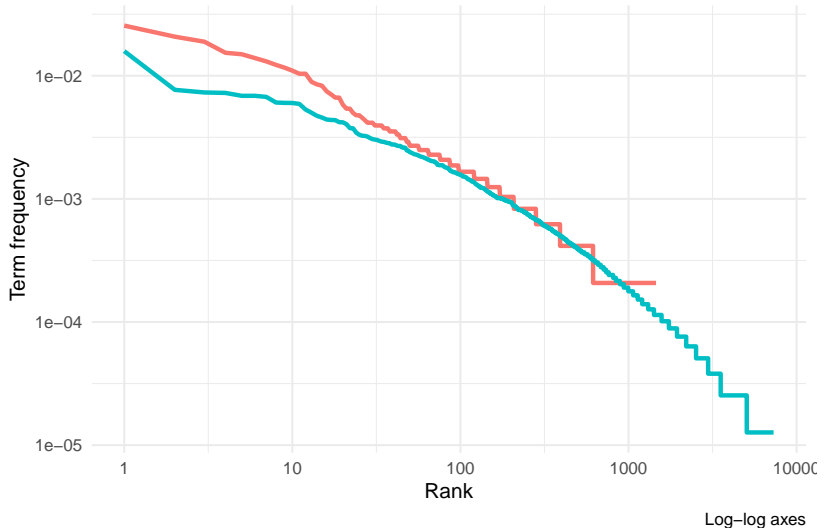


Working with text: Zipf's law

Calculating rank and frequency for each word in each text.

```
freq_by_rank <- words %>%  
  group_by(title) %>%  
  mutate(rank = row_number(),  
         `term frequency` = n/total) %>%  
  ungroup()
```


Working with text: Zipf's law



Working with text

N-grams

- ▶ So far we have just considered treating a text as a “bag-of-words”
- ▶ One way to maintain some information about word order (and hence syntax) is to use N-grams
- ▶ An *N-gram** is a sequence of N words
- ▶ We often split texts into N-grams to capture basic syntactic units like phrases
 - ▶ N is usually small.
 - ▶ $N = 2$ is called a “bigram”; $N = 3$ is a “trigram”
 - ▶ e.g. “I like peanut butter” contains the following bigrams: “I like”, “like peanut”, & “peanut butter”.

*Nothing to do with Scientology [https://en.wikipedia.org/wiki/Engram_\(Dianetics\)](https://en.wikipedia.org/wiki/Engram_(Dianetics))

Working with text

N-grams

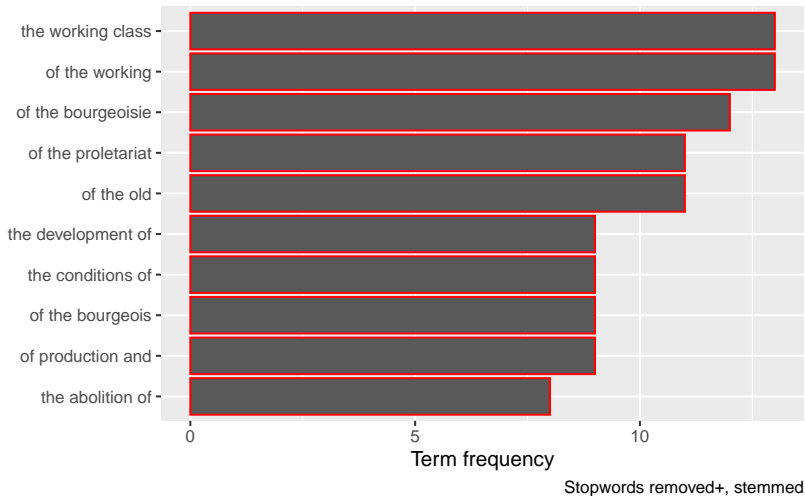
- ▶ We can also use *character N-grams* to split documents into sequences of characters
 - ▶ e.g. “character” can be split into the following triplets (“cha”, “har”, “ara”, “rac”, “act”, “cte”, “ter”)
- ▶ Some recent approaches like BERT combine both character and word N-grams into “word pieces”.
 - ▶ This makes it easy to tokenize new documents since we can always represent them as characters if a word is not in our vocabulary

Exercise:

Modify `unnest_tokens` to construct trigrams.

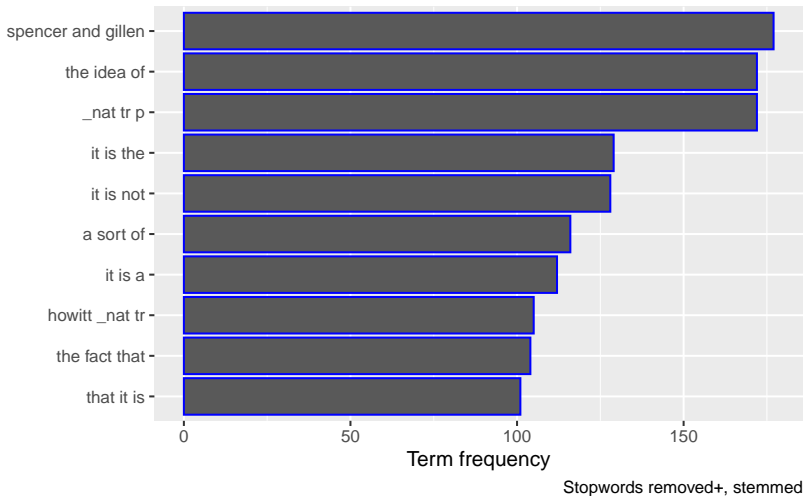
Exercise

10 most frequent trigrams in The Communist Manifest



Exercise

10 most frequent trigrams in The Elementary Forms of Religi



Working with text

Term-frequency inverse document-frequency (TF-IDF)

- ▶ Term frequency alone is an imperfect measure for comparing across documents
 - ▶ Terms common to multiple documents provide us with little information (e.g. stop words)
 - ▶ Terms unique to few documents better allow us to distinguish between them
- ▶ TF-IDF is a way to weight term frequencies to give higher weights to terms which occur infrequently across documents

Working with text

Calculating term-frequency inverse document-frequency (TF-IDF)

- ▶ N = number of documents in the corpus
- ▶ $tf_{t,d}$ = number of times term t used in document d
- ▶ df_t = number of documents containing term t
- ▶ $idf_t = \log\left(\frac{N}{df_t}\right)$ = log of fraction of all documents containing t
 - ▶ Note that $\frac{N}{df_t}$ is larger for less frequent terms
 - ▶ We take the log to penalize very high values
- ▶ We then use these values to calculate $TFIDF_{t,d} = tf_{t,d} * idf_t$

Working with text: TF-IDF

Computing TF-IDF in tidytext

We can easily compute TF-IDF weights using `tidy.text` by using the word-count object we created.

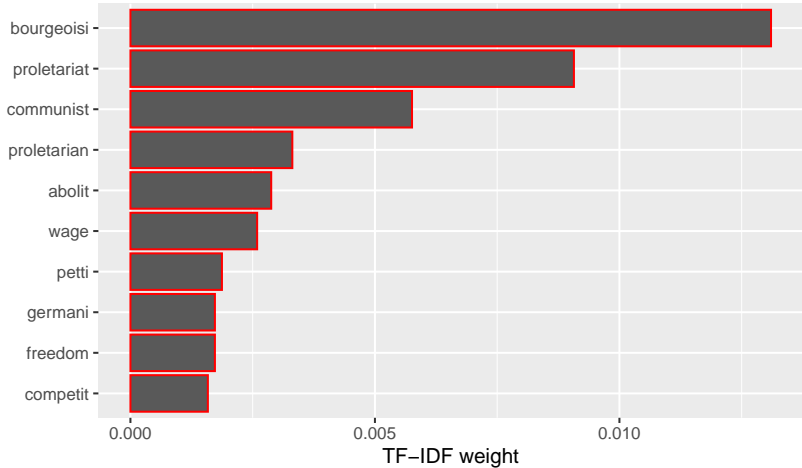
```
tidy.tfidf <- words %>% bind_tf_idf(word, title, n)
head(tidy.tfidf)
```

```
## # A tibble: 6 x 7
##   title          word      n total      tf    idf tf_idf
##   <chr>         <chr>   <int> <int>   <dbl> <dbl> <dbl>
## 1 Elementary Forms totem    1250 78851 0.0159 0.693 0.0110
## 2 Elementary Forms religi    606 78851 0.00769 0      0
## 3 Elementary Forms anim     577 78851 0.00732 0      0
## 4 Elementary Forms religion  572 78851 0.00725 0      0
## 5 Elementary Forms form     542 78851 0.00687 0      0
## 6 Elementary Forms natur    542 78851 0.00687 0      0
```

Note the two document example is quite trivial. Words occurring in both documents get idf weights ~ 0 .

Working with text: TF-IDF

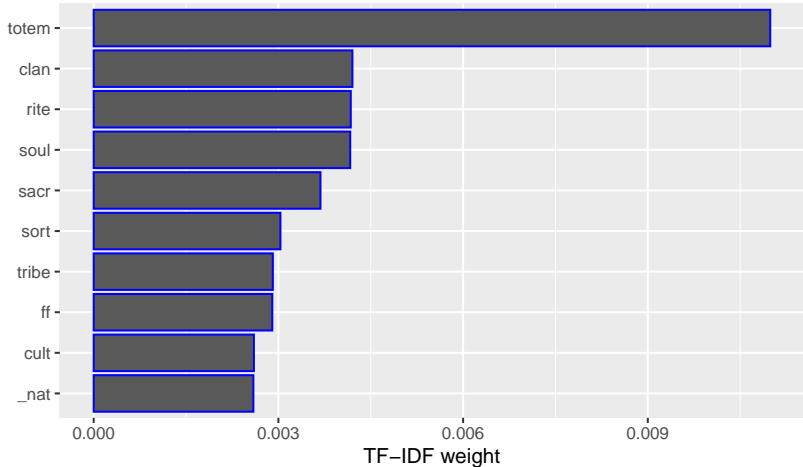
10 stems with highest TF-IDF weight in The Communist Manifesto



Stopwords removed+, stemmed

Working with text: TF-IDF

10 stems with highest TF-IDF weight in The Elementary Forms of Religion



Stopwords removed+, stemmed

Vector representations of texts

The document-term matrix (DTM)

- ▶ A frequently used bag-of-words representation of a text corpus is the *Document-Term Matrix*:
 - ▶ Each row* is a document (a unit of text)
 - ▶ Each column is a term (word)
 - ▶ For a given DTM X , each cell $X_{i,j}$ indicates the number of times a term i occurs in document j , $tf_{i,j}$.
- ▶ Most cells are empty so it is usually stored as a sparse matrix.

*Often the rows and columns reversed, resulting in a *Term-Document Matrix* or *TDM*

Vector representations of texts

Casting a tidytext object into a DTM

```
X <- tidy.text %>% count(title, word) %>% cast_dtm(title, word, n)
print(X)
```

```
## <<DocumentTermMatrix (documents: 2, terms: 7741)>>
## Non-/sparse entries: 8756/6726
## Sparsity           : 43%
## Maximal term length: 22
## Weighting          : term frequency (tf)
```

Vector representations of texts

Geometric interpretation

- ▶ Each text is a vector in N -dimensional space, where N is the total number of unique words (column of the DTM)
- ▶ Each word is a vector in D -dimensional space, where D is the number of documents (rows of the DTM)

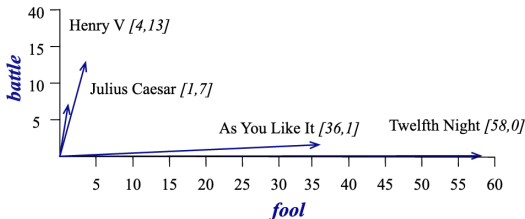
See <https://web.stanford.edu/~jurafsky/slp3/6.pdf> for more details on the vector-space model

Vector representations of texts

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	14	80	62	89
fool	36	58	1	4
wit	20	15	2	3

This example from Jurafsky and Martin shows a Term-Document Matrix (TDM) pertaining to four key words from four Shakespeare plays. The document vectors are highlighted in red.

Vector representations of texts



Here vectors for each play are plotted in two-dimensional space. The y- and x-axes indicate the number of times the words "battle" and "fool" appear in each play. Note how some vectors are closer than others and how they have different lengths.

Vector representations of texts

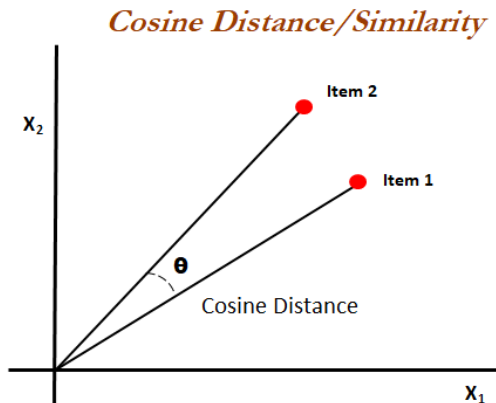
	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Figure 6.5 The term-document matrix for four words in four Shakespeare plays. The red boxes show that each word is represented as a row vector of length four.

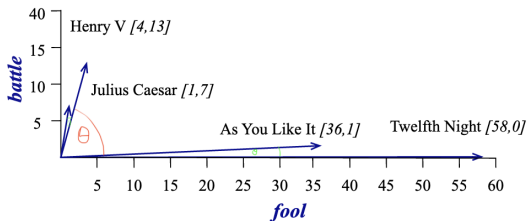
We could also treat the rows of this matrix as vector representations of each word. We will discuss this further next week when we study word embeddings.

Vector representations of texts

Cosine similarity



Vector representations of texts



Vector representations of texts

Calculating cosine similarity

\vec{u} and \vec{v} are vectors representing texts (e.g. rows from a DTM matrix). We can compute the cosine of the angle between these two vectors using the following formula:

$$\cos(\theta) = \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\| \|\vec{v}\|} = \frac{\sum_i \vec{u}_i \vec{v}_i}{\sqrt{\sum_i \vec{u}_i^2} \sqrt{\sum_i \vec{v}_i^2}}$$

The value range from 0 (complete dissimilarity) to 1 (identical), since all values are non-negative.

Vector representations of texts

Calculating cosine similarity

```
cosine.sim <- function(u,v) {  
  
  numerator <- u %*% v  
  denominator <- sqrt(sum(u^2)) * sqrt(sum(v^2))  
  
  return (numerator/denominator)  
}
```

Vector representations of texts

Cosine similarity between Marx and Durkheim

```
X.dense <- as.matrix(X)
print(cosine.sim(X.dense[1,], X.dense[2,]))

##           [,1]
## [1,] 0.2727339
```

Vector representations of texts

Cosine similarity for a larger corpus

The similarity between Marx's *Communist Manifesto* and Durkheim's *Elementary Forms* is rather meaningless without more information. Let's consider another example with a (slightly) larger corpus of texts.

```
#m <- gutenberglmetadata %>% filter(author == "Shakespeare, William" &
rj <- gutenbergldownload(1112)
mnd <- gutenbergldownload(1113)
tn <- gutenbergldownload(1123)
kl <- gutenbergldownload(1128)
mb <- gutenbergldownload(1129)
rj$play <- "Romeo & Juliet"
mnd$play <- "A Midsummer Night's Dream"
tn$play <- "Twelfth Night"
kl$play <- "King Lear"
mb$play <- "Macbeth"

S <- bind_rows(rj, mnd, tn, kl, mb)
```

Vector representations of texts

Exercise: From tidytext to DTM

Convert the plays into tidytext objects, using any preprocessing steps you want and filtering out any words which occur less than 10 times in the corpus. Calculate TF-IDF scores then convert to a DTM called `S.m`.

Vector representations of texts

Normalizing columns

```
S.dense <- as.matrix(S.m)

normalize <- function(v) {
  return (v/sqrt(sum(v^2)))
}

# Normalizing every column
for (i in 1:4) {
  S.dense[i,] <- normalize(S.dense[i,])
}
```

Vector representations of texts

Calculating cosine similarity using matrix multiplication

```
sims <- S.dense %*% t(S.dense)
print(sims)
```

```
##
```

```
Docs
```

```
## Docs
```

```
A Midsummer Night's Dream King Lear
```

```
## A Midsummer Night's Dream
```

```
1.00000000 0.44837496 0.3
```

```
## King Lear
```

```
0.44837496 1.00000000 0.4
```

```
## Macbeth
```

```
0.36835175 0.41323267 1.0
```

```
## Romeo & Juliet
```

```
0.52748241 0.52900587 0.4
```

```
## Twelfth Night
```

```
0.03149266 0.04301891 0.0
```

```
##
```

```
Docs
```

```
## Docs
```

```
Romeo & Juliet Twelfth Night
```

```
## A Midsummer Night's Dream
```

```
0.52748241 0.03149266
```

```
## King Lear
```

```
0.52900587 0.043018912
```

```
## Macbeth
```

```
0.40133981 0.029018825
```

```
## Romeo & Juliet
```

```
1.00000000 0.038207661
```

```
## Twelfth Night
```

```
0.03820766 0.008267727
```