Computational Sociology

Introduction to Natural Language Processing

Dr. Thomas Davidson

Rutgers University

February 29, 2024

Plan

1. Course updates
2. What is NLP?
3. Preprocessing texts
4. The bag-of-words representation
5. TF-IDF weighting
6. The vector-space model
7. Cosine similarity

Course updates

- Homework 2 due tomorrow at 5pm
- Project proposals due next week Friday at 5pm
  - 3-4 pages double-spaced (see details from last week)
  - *Submit PDF via email*

Introduction to NLP

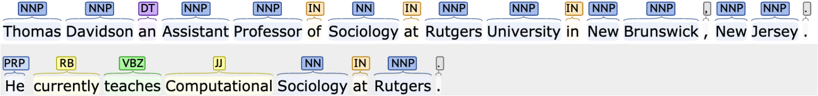
What is natural language processing?

- 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

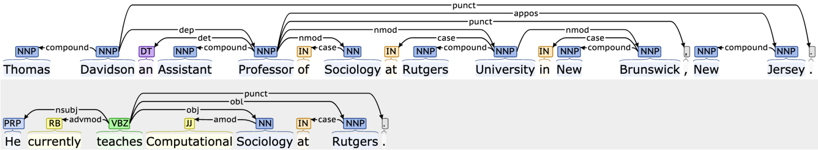
Introduction to NLP

NLP tasks: Part-of-speech tagging



Examples created using https://corenlp.run/

NLP tasks: Dependency-parsing



Examples created using https://corenlp.run/

NLP tasks: Co-reference resolution

CorefEntity8

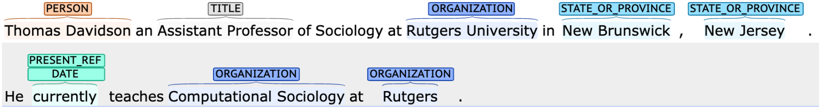
Thomas Davidson an Assistant Professor of Sociology at Rutgers University in New Brunswick , New Jersey .

CorefEntity8

He currently teaches Computational Sociology at Rutgers .

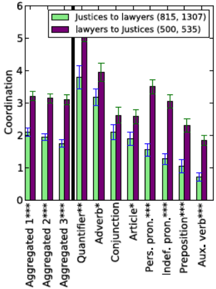
Examples created using https://corenlp.run/

NLP tasks: Named-entity recognition



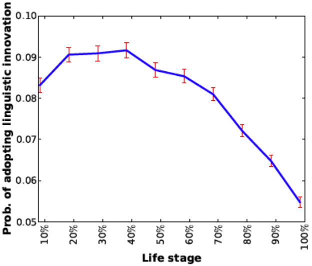
Examples created using https://corenlp.run/

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.

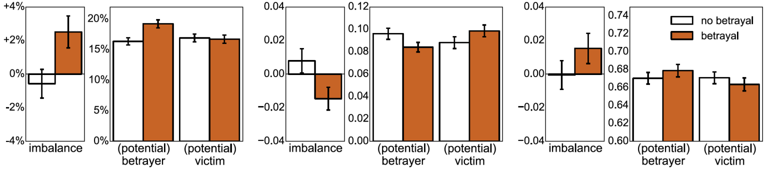
Applications: Identity and group membership



(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.

Applications: Trust and betrayal



(a) Positive sentiment (percentage of sentences)

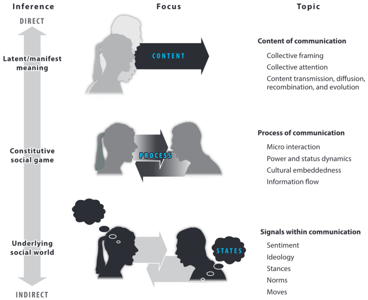
(b) Planning discourse markers (avg. number per sentence)

(c) Politeness (avg. message score)

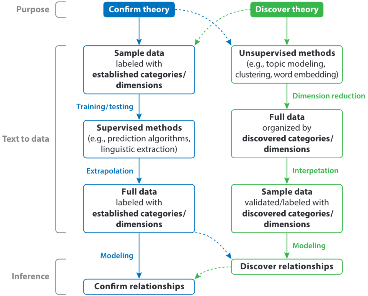
Figure 3: Friendships that will end in betrayal are imbalanced. The eventual betrayer is more positive, more polite, but plans less than the victim. The white bars correspond to matched lasting friendships, where the roles of potential betrayer and victim are arbitrarily assigned; in these cases, the 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.

NLP and Social Theory (Evans and Aceves 2016)

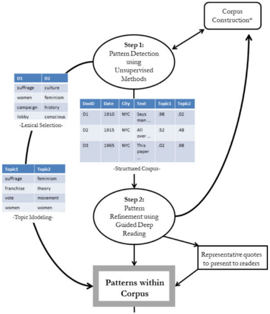


NLP and Social Theory (Evans and Aceves 2016)



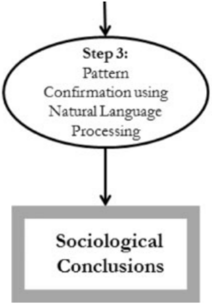
Introduction to NLP

Computational Grounded Theory (Nelson 2020)



Introduction to NLP

Computational Grounded Theory (Nelson 2020)



Introduction to NLP

Computational and qualitative research

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

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

NLP class timeline

- Week 7
  - Pre-processing, bag-of-words, and the vector-space model
- Week 8
  - Word embeddings
- Week 9 (after spring break)
  - Topic models
- Week 11 (Week 10 introduces machine learning)
  - 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”) $\Rightarrow$ “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 punctution 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

Pre-process with caution!

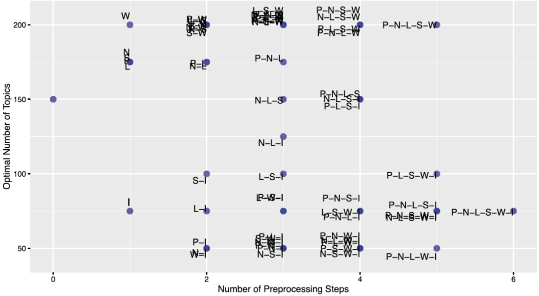


Figure 2. Plot depicting the optimal number of topics (as selected via perplexity) for each of 64 preprocessing specifications not including trigrams. On the x-axis is the number of preprocessing steps, and the y-axis is the number of topics. Each point is labeled according to its specification.

Denny, Matthew J., and Arthur Spirling. 2018. "Text Preprocessing For Unsupervised Learning" Political Analysis 26 (02): 168–89. https://doi.org/10.1017/pan.2017.44.

Working with text

Word counts

- Now we have done some pre-processing, one of the most basic ways we can start to analyze tests is by counting the frequency of words.
  - e.g. "I think, therefore I am" $\Rightarrow$

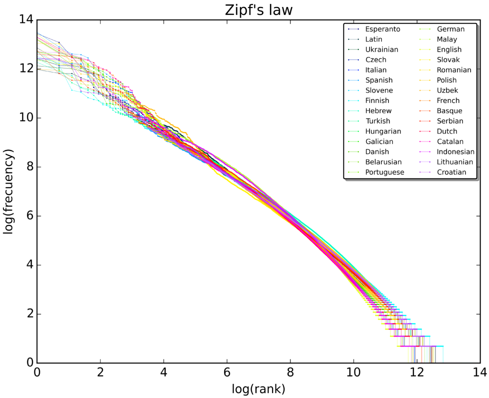
Word	Count
I	2
think	1
therefore	1
am	1

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

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)
library(ggplot2)
library(stringr)
#install.packages("tm") # Dependency for tidytext, required for cast_dt

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] "declare that their ends can be attained only by the forcible ov
## [2] "of all existing social conditions. Let the ruling classes tremb
## [3] "Communistic revolution. The proletarians have nothing to lose b
## [4] "chains. They have a world to win."
## [5] ""
## [6] "WORKING MEN OF ALL COUNTRIES, UNITE!"
```

Working with text

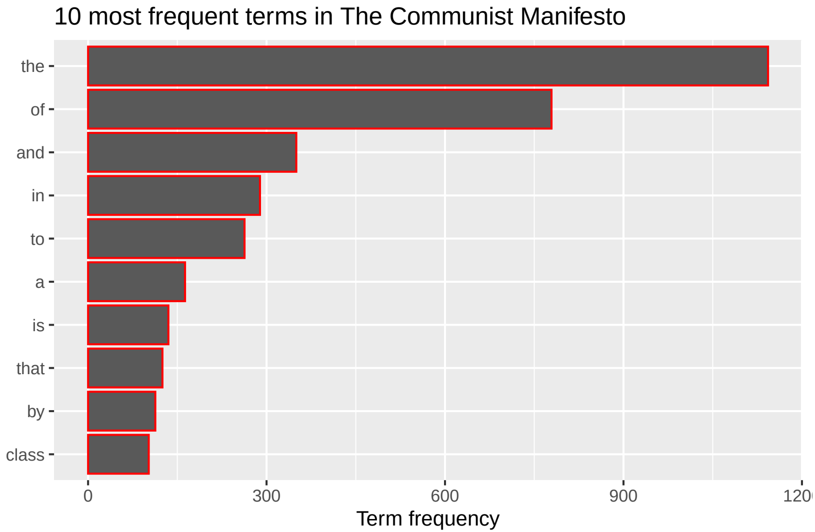
Tokenizing using `tidytext`

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.

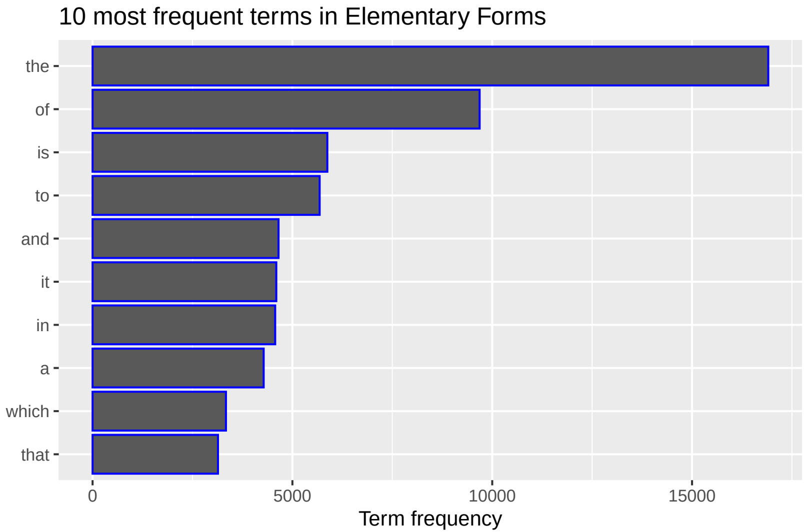
```
tidy.text <- texts %>% unnest_tokens(word, text)
tail(tidy.text$word)
```

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

Term frequency in *The Communist Manifesto*



Term frequency in *Elementary Forms*



Working with text

Removing stopwords

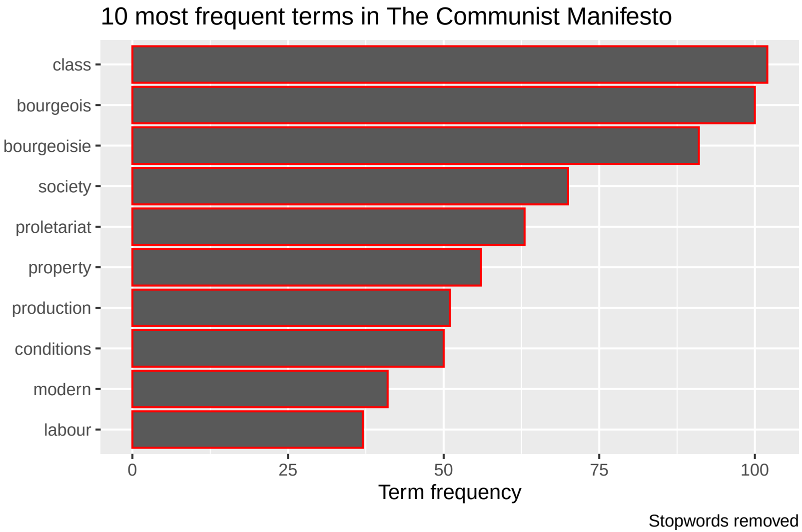
We can load a corpus of stop words contained in `tidytext` and use `anti_join` to filter our texts. This retains all records without a match in stopwords.

```
data(stop_words)
head(stop_words)
```

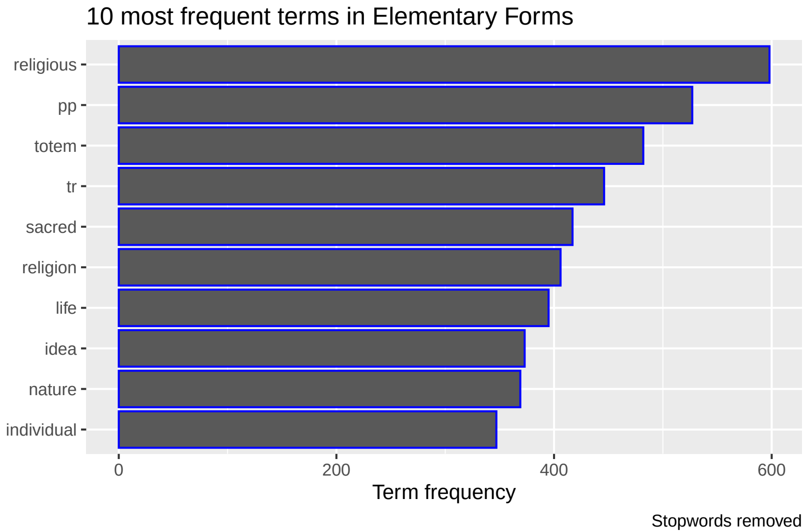
```
## # A tibble: 6 x 2
##   word      lexicon
##   <chr>     <chr>
## 1 a         SMART
## 2 a's       SMART
## 3 able      SMART
## 4 about     SMART
## 5 above     SMART
## 6 according SMART
```

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

Term frequency in *The Communist Manifesto*



Term frequency in *Elementary Forms*



Working with text

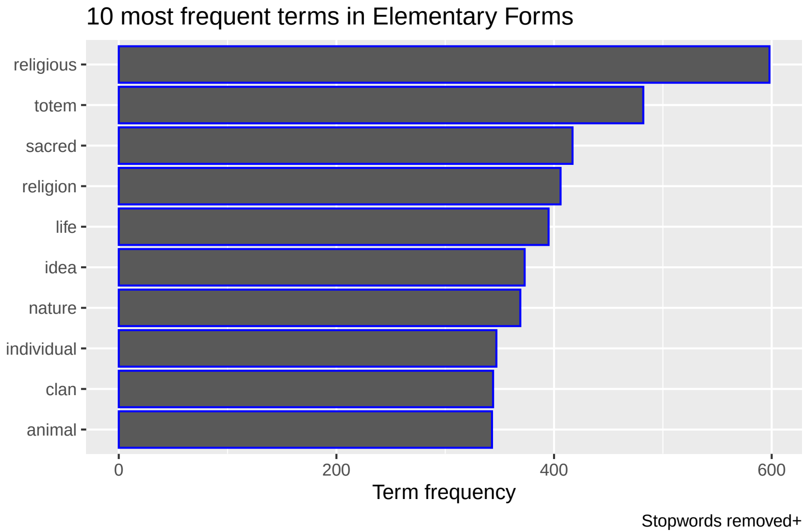
Removing new stopwords

The last example shows how there is still some “junk” in the Durkheim text. We can try to remove this by adding to our stopword list.

```
to.remove <- tibble(text=c("pp", "tr")) %>% unnest_tokens(word, text)

tidy.text <- tidy.text %>%
  anti_join(to.remove)
```

Term frequency in *Elementary Forms*



Working with text

Stemming

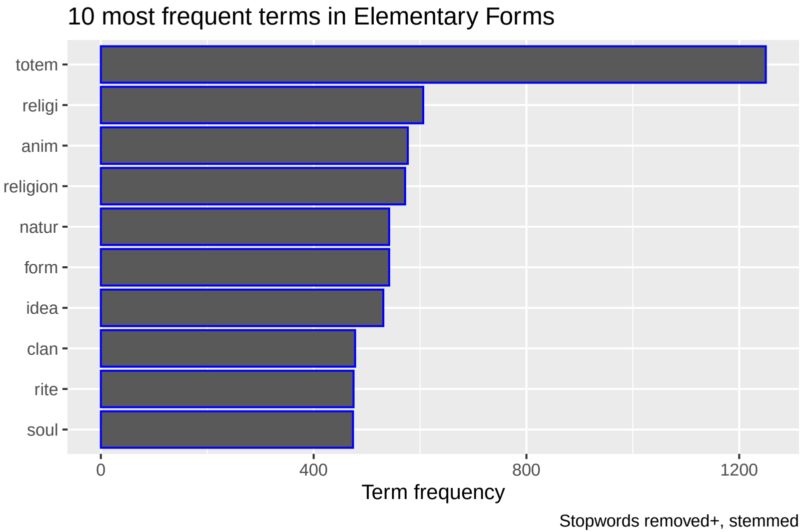
We can stem the terms using a function from the package `SnowballC`, which is a wrapper for a commonly used stemmer called the Porter Stemmer, written in C.

```
library(SnowballC)

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

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

Term frequency in *Elementary Forms*



Working with text

Counting words

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

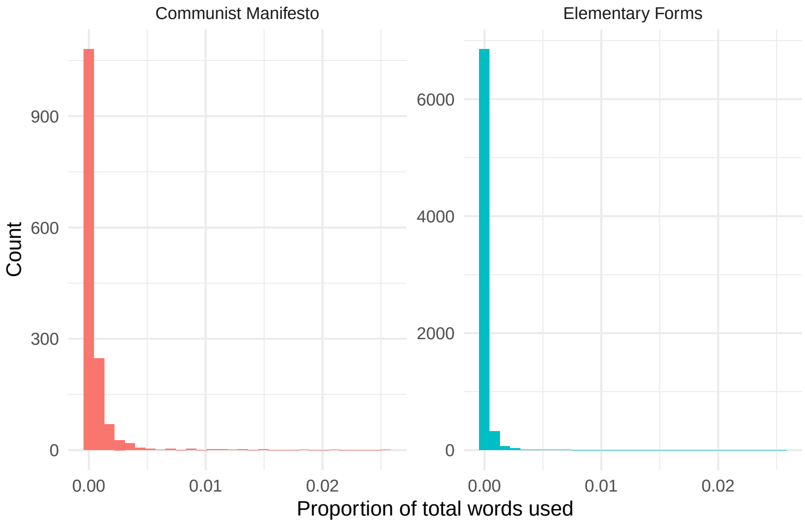
```
# Count words by text
text.words <- tidy.text %>% count(title, word, sort = TRUE)

# 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
```

Word frequency distribution



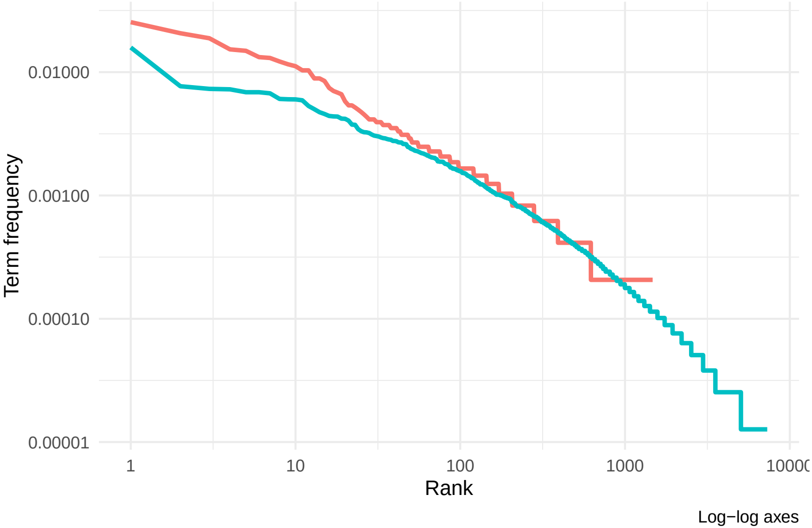
Working with text

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()
```

Zipf's law



0.01000
0.00100
0.00010
0.00001
Term frequency
1
10
100
1000
10000
Rank
Log−log axes

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)

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 obtain trigrams from the texts.

Trigrams in *The Communist Manifesto*

```
tidy.trigrams %>% filter(gutenberg_id == 61) %>%
  count(word, sort = TRUE) %>%
  slice(1:10) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(n, word)) +
  geom_col(color='red') +
  labs(y = NULL, x='Term frequency',
       title="10 most frequent trigrams in The Communist Manifesto",
       caption="Stopwords removed+, stemmed")
```

Trigrams in *Elementary Forms*

```
tidy.trigrams %>% filter(gutenberg_id == 41360) %>%
  count(word, sort = TRUE) %>%
  slice(1:10) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(n, word)) +
  geom_col(color='blue') +
  labs(y = NULL, x='Term frequency',
       title="10 most frequent trigrams in The Elementary Forms of Reli
       caption="Stopwords removed+, stemmed")
```

Working with text

Comparing documents

- Building upon the previous analyses, we will now consider how to compare documents
  - Re-weighting word counts to find distinctive words
  - Representing documents as vectors of word counts
  - Geometric interpretations of document vectors

Working with text

Limitations of word counts

- Word counts alone are an imperfect measure for comparing documents
  - Some words occur in most documents, providing little information about the document (recall Zipf's law)
  - Similarly, some words are very rare, providing little generalizable insight
  - We want to find words that help distinguish between documents

Working with text

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

- Term-frequency inverse document-frequency (TF-IDF) is a way to weight word counts ("term frequencies") to give higher weights to words that help distinguish between documents
  - Intuition: Adjust word counts to take into account how many documents a word appears in.

Working with text

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

- $D =$ number of documents in the corpus
- $tf_{t,d} = t$ as proportion of all words in d / number of times term t used in document d
- $df_t =$ number of documents containing term t
- $idf_t = log(\frac{D}{df_t}) =$ log of fraction of all documents containing t
  - $\frac{N}{df_t}$ is larger for terms occurring in fewer documents
  - The logarithm is used to penalize very high values
  - If a word occurs in all documents $df_t = N \rightarrow idf_t = log\frac{D}{D} = log(1) = 0$.
- We then use these values to calculate $TFIDF_{t,d} = tf_{t,d} * idf_t$

Working with text

Computing TF-IDF in `tidytext`

We can easily compute TF-IDF weights using `tidy.text` by using the word-count object we created earlier. Note the two document example is trivial. Many words have IDF scores equal to zero because they occur in both documents.

```
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
```

Working with text

Take the stem “countri” for example (short for country, country’s, countries).

```
## # A tibble: 2 x 7
##   title                word          n total        tf   idf tf_idf
##   <chr>                <chr>     <int> <int>     <dbl> <dbl>  <dbl>
## 1 Communist Manifesto countri      26  4835 0.00538       0      0
## 2 Elementary Forms    countri      16 78851 0.000203      0      0
```

Working with text

The term "australia" has a relatively low term frequency but a higher IDF score, since it only occurs in *Elementary Forms*.

```
## # A tibble: 1 x 7
##   title             word            n total      tf   idf   tf_idf
##   <chr>             <chr>       <int> <int>   <dbl> <dbl>    <dbl>
## 1 Elementary Forms australia     108 78851 0.00137 0.693 0.000949
```

Working with text

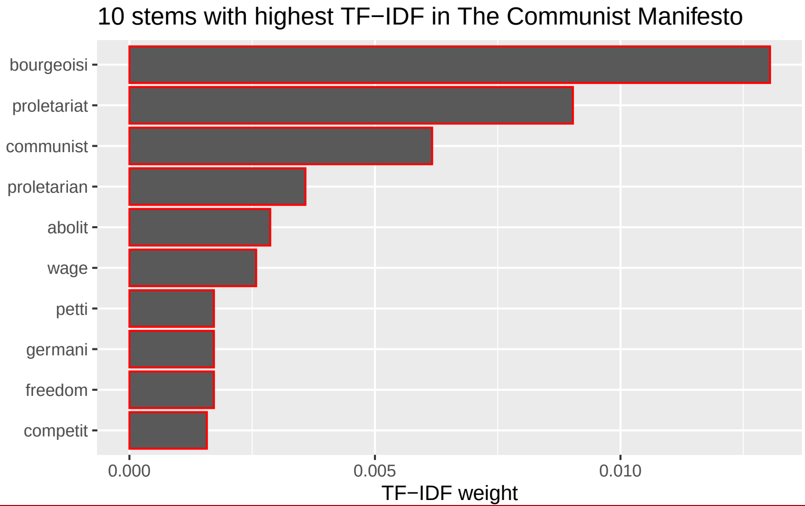
Choose another word and inspect the results.

Working with text

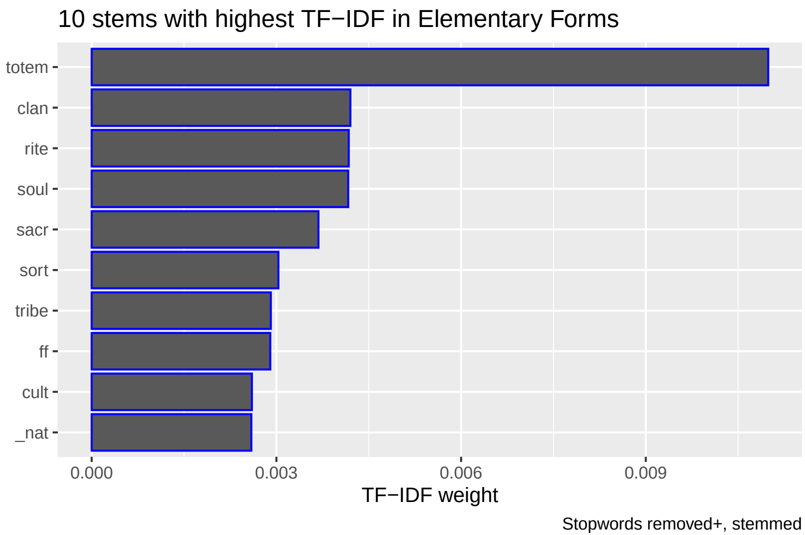
In this case *all* words unique to one document will have the same IDF score. Why?

```
## # 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 clan    478 78851 0.00606 0.693 0.00420
## 3 Elementary Forms rite    475 78851 0.00602 0.693 0.00418
## 4 Elementary Forms soul    474 78851 0.00601 0.693 0.00417
## 5 Elementary Forms sacr    419 78851 0.00531 0.693 0.00368
## 6 Elementary Forms sort    345 78851 0.00438 0.693 0.00303
```

TF-IDF weighted word stems in *The Communist Manifesto*



TF-IDF weighted word stems in *Elementary Forms*



The vector-space model

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, $tf_{i,j}$.
    - This can be the raw term counts or TF-IDF weighted counts.
- Most cells are empty so it is usually stored as a sparse matrix to conserve memory.

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

The vector-space model

Casting a `tidytext` object into a DTM

```
X <- texts %>% unnest_tokens(word, text) %>%
    anti_join(stop_words) %>% count(title, word) %>%
    cast_dtm(title, word, n)
print(X)
```

```
## <<DocumentTermMatrix (documents: 2, terms: 11524)>>
## Non-/sparse entries: 12661/10387
## Sparsity           : 45%
## Maximal term length: 22
## Weighting          : term frequency (tf)
```

Note: This matrix is not weighted by TF-IDF, although we could apply the weights if desired.

The vector-space model

Viewing the DTM

The object created is a class unique to the `tidytext` package. We can inspect this to see what it contains.

```
class(X)
```

```
## [1] "DocumentTermMatrix"    "simple_triplet_matrix"
```

```
dim(X)
```

```
## [1]     2 11524
```

```
X$dimnames[1]
```

```
## $Docs
## [1] "Communist Manifesto" "Elementary Forms"
```

```
X$dimnames[[2]][1:50] # first 50 columns
```

```
##  [1] "1"                 "10"                "1830"              "1846"
##  [5] "1847"              "1888"              "18th"              "2"
##  [9] "3"                 "4"                 "5"                 "6"
## [13] "7"                 "8"                 "9"                 "_a"
## [17] "_b"                "_beaux"            "_c"                "_i.e_"
```

The vector-space model

Viewing the DTM

The easiest way to see the actual DTM is to cast it to a matrix.

```
Xm <- as.matrix(X)
```

The vector-space model

Geometric interpretations

- Each document is a vector in N-dimensional space, where N is the total number of unique words (row of the DTM / column of TDM)
- Each word is a vector in D-dimensional space, where D is the number of documents (columns of the DTM / row of TDM)

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

The vector-space model

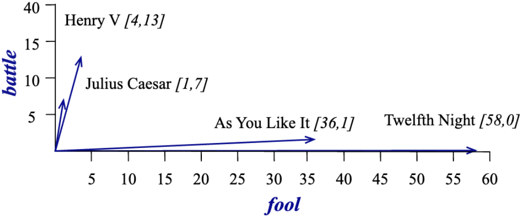
Document vectors

	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

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.

The vector-space model

Document vectors



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.

The vector-space model

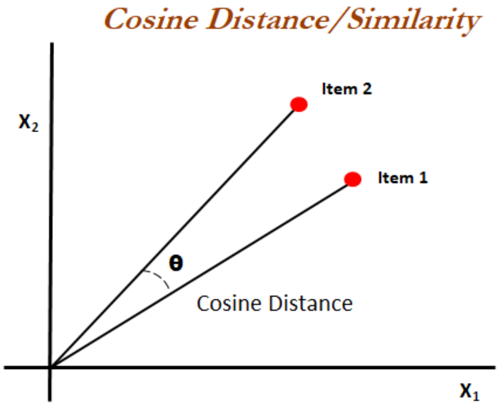
Word vectors

	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

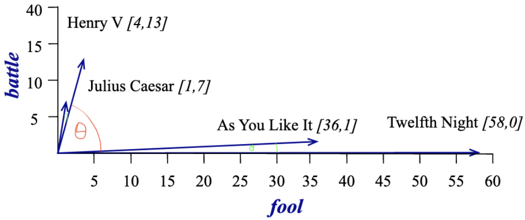
Figure 6.5 The term-document matrix for four words in four Shakespeare plays. The red boxes show that each word is represented as a row vector of length four.

We could also treat the rows of this matrix as vector representations of each word. We will return to this idea next week.

Cosine similarity



Cosine similarity



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:

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

Cosine similarity

Calculation

```r
set.seed(22924)
u <- rnorm(10)
v <- rnorm(10)

sum(u*v) / (sqrt(sum(u^2)) * sqrt(sum(v^2)))
```

```
## [1] 0.09405026
```

```r
# Same result using matrix multiplication
u %*% v / (sqrt(u %*% u) * sqrt(v %*% v))
```

```
##            [,1]
## [1,] 0.09405026
```

Cosine similarity

Making a function

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

cosine.sim(u,v)
```

```
##             [,1]
## [1,] 0.09405026
```

Cosine similarity

Cosine similarity between Marx and Durkheim

We can use the two columns of the DTM matrix defined above as arguments to the similarity function.

```
print(cosine.sim(Xm[1,], Xm[2,]))
```

```
##           [,1]
## [1,] 0.2031065
```

Cosine similarity

Cosine similarity for a larger corpus

Let's consider another example with a larger corpus of texts.

```
m <- gutenberg_metadata %>%
    filter(author == "Shakespeare, William" & language == "en")
plays <- gutenberg_download(2235:2269)

plays <- plays %>% left_join(m, by = "gutenberg_id") %>%
    filter(gutenberg_id != 2240) # Removing a duplicate
```

Cosine similarity

From text to DTM

Exercise: Modify the pipeline to filter out words that occur less than 5 times in the entire corpus.

```
## <<DocumentTermMatrix (documents: 33, terms: 29205)>>
## Non-/sparse entries: 119712/844053
## Sparsity           : 88%
## Maximal term length: 17
## Weighting          : term frequency (tf)
## [1]    33 29205
```

Cosine similarity

Extracting TF-IDF matrix

```
DTMd <- as.matrix(DTM)

# Run line below if using tf-idf weights as
# some columns contain zeros and must be removed
#DTMd <- DTMd[,colSums(DTM) > 0]
```

Cosine similarity

Normalizing columns

We can simplify the cosine similarity calculation if we normalize each column by its length (the denominator in the above calculation.)

```
normalize <- function(v) {
  return (v/sqrt(v %*% v))
}

# Normalizing every column in the matrix
for (i in 1:dim(DTMd)[1]) {
  DTMd[i,] <- normalize(DTMd[i,])
}
```

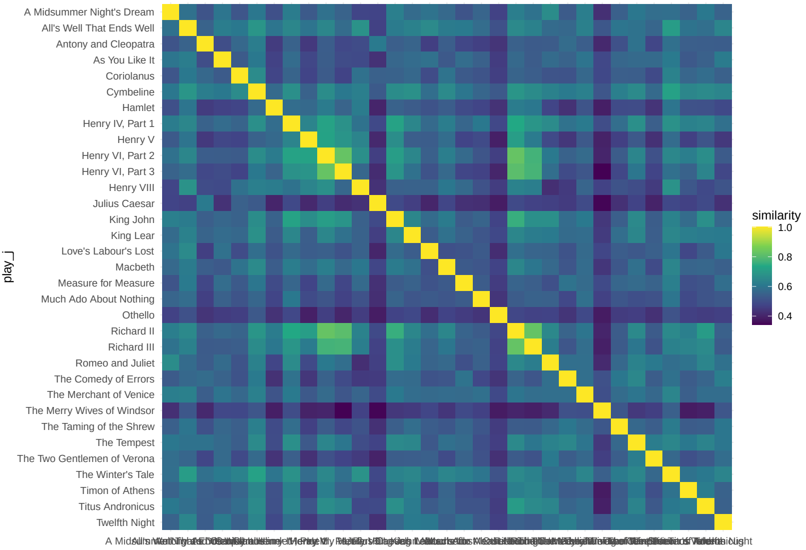
Cosine similarity

Calculating cosine similarity using matrix multiplication

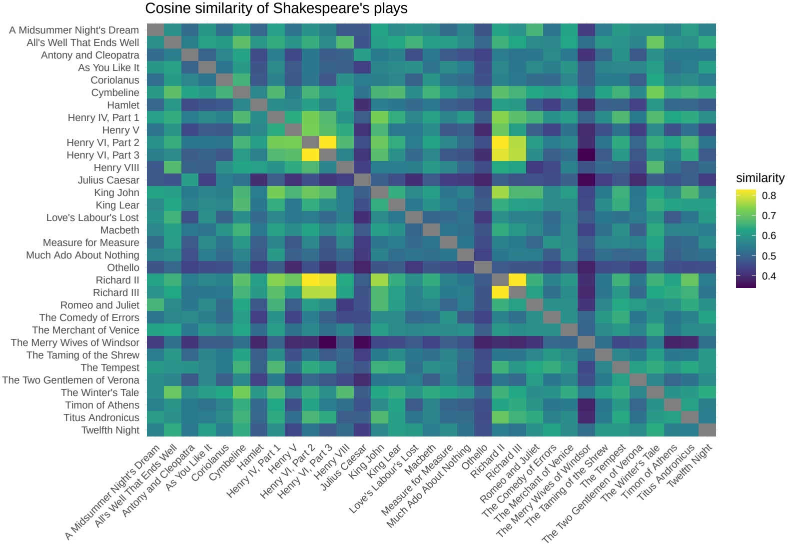
```
sims <- DTMd %*% t(DTMd)
print(sims)
```

```
##                                   Docs
## Docs                                A Midsummer Night's Dream
##   A Midsummer Night's Dream                         1.0000000
##   All's Well That Ends Well                         0.5867144
##   Antony and Cleopatra                              0.5125922
##   As You Like It                                    0.5960917
##   Coriolanus                                        0.5168455
##   Cymbeline                                         0.6029199
##   Hamlet                                            0.5018106
##   Henry IV, Part 1                                  0.6167272
##   Henry V                                           0.5269171
##   Henry VI, Part 2                                  0.5872136
##   Henry VI, Part 3                                  0.5497865
##   Henry VIII                                        0.4750135
##   Julius Caesar                                     0.4730039
```

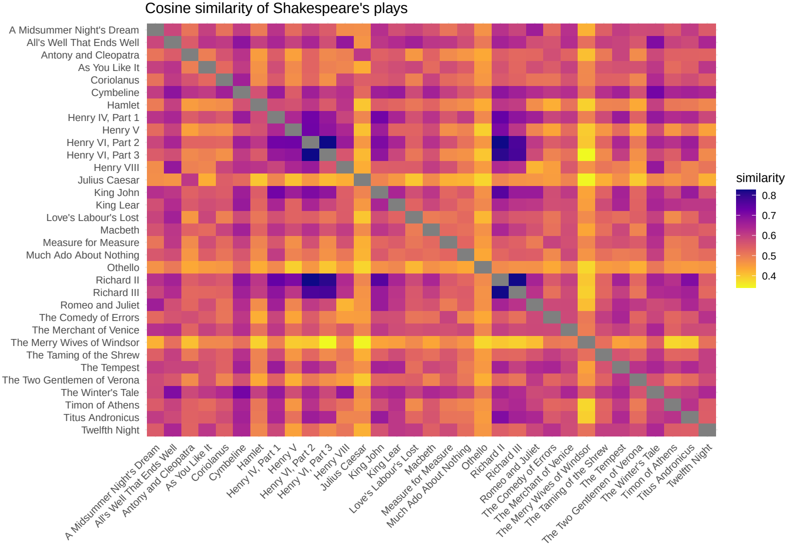
Cosine similarity



Cosine similarity



Cosine similarity



Next week

- Word embeddings