Analysis of Unstructured Text

SSC Data Science and Analytics Workshop

Nathan Taback

Statistical Sciences and Computer Science, University of Toronto

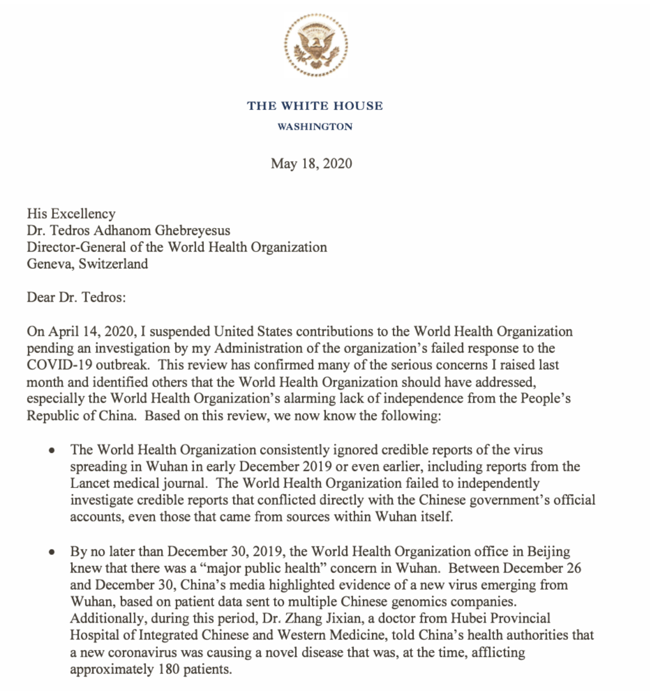
June 4, 2020

Outline

- R Libraries for Working with Text
- Importing unstructured text into R: copy/paste, external API, webscraping, R libraries
- Regular expressions and text normalization (e.g., tokenization)
- N-Grams
- Word Vectors and Matricies

Introduction

Example 1: Trump's Letter to WHO



THE WHITE HOUSE
WASHINGTON

May 18, 2020

His Excellency
Dr. Tedros Adhanom Ghebreyesus
Director-General of the World Health Organization
Geneva, Switzerland

Dear Dr. Tedros:

On April 14, 2020, I suspended United States contributions to the World Health Organization pending an investigation by my Administration of the organization's failed response to the COVID-19 outbreak. This review has confirmed many of the serious concerns I raised last month and identified others that the World Health Organization should have addressed, especially the World Health Organization's alarming lack of independence from the People's Republic of China. Based on this review, we now know the following:

- The World Health Organization consistently ignored credible reports of the virus spreading in Wuhan in early December 2019 or even earlier, including reports from the Lancet medical journal. The World Health Organization failed to independently investigate credible reports that conflicted directly with the Chinese government's official accounts, even those that came from sources within Wuhan itself.

- By no later than December 30, 2019, the World Health Organization office in Beijing knew that there was a "major public health" concern in Wuhan. Between December 26 and December 30, China's media highlighted evidence of a new virus emerging from Wuhan, based on patient data sent to multiple Chinese genomics companies. Additionally, during this period, Dr. Zhang Jixian, a doctor from Hubei Provincial Hospital of Integrated Chinese and Western Medicine, told China's health authorities that a new coronavirus was causing a novel disease that was, at the time, afflicting approximately 180 patients.

- Donald Trump's letter to the Director General of the World Health Organization, Dr. Tedros, is an example of unstructured data.
- There are dates, numbers, and text that does not have a pre-defined data structure.

Example 2: Analysis of p-values

This Issue | Views 75,408 | Citations 131 | Altmetric 528

PDF | More | Cite | Permissions

Original Investigation — FREE

March 15, 2016

Evolution of Reporting *P* Values in the Biomedical Literature, 1990-2015

David Chavalarias, PhD[1]; Joshua David Wallach, BA[2]; Alvin Ho Ting Li, BHSc[3]; John P. A. Ioannidis, MD, DSc[4]

Author Affiliations | Article Information

JAMA. 2016;315(11):1141-1148. doi:10.1001/jama.2016.1952

We defined a P value report as a string starting with either “p,” “P,” “p-value(s),” “P-value(s),” “P value(s),” or “p value(s),” followed by an equality or inequality expression (any combination of =, <, >, ≤, ≥, “less than,” or “of <” and then by a value, which could include also exponential notation (for example, 10-4, 10(-4), E-4, (-4), or e-4).

p-values were extracted from papers using a regular expression

Programming Languages for Working With Unstructured Text

- Two popular languages for computing with data are R and Python.
- We chose R for this workshop, but we could have selected Python.

R Libraries for Working with Text

Some very useful R libraries for working with unstructured text that are used in this workshop:

- `base`
- `tidyverse`
- `janitor`
- `tidytext`
- `rvest`
- `RSelenium`

Importing text into R

Copy/Paste

- For one-time this can work well.


```
tweet_txt <- "I'm starting the week with an
update on the Canada Emergency Commercial
Rent Assistance and the work we're doing
to get you the support you need. Tune in
now for the latest:"

tweet_link <- "https://www.cpac.ca/en/programs/
tweet_replies <- 164
tweet_rt <- 116
tweet_likes <- 599
tweet_url <- "https://twitter.com/JustinTrudeau
```

How many words in the tweet text?

```r
library(tidyverse)
```

```
## ── Attaching packages ──────────────────────────────

## ✓ ggplot2 3.3.0     ✓ purrr   0.3.4
## ✓ tibble  3.0.1     ✓ dplyr   0.8.5
## ✓ tidyr   1.0.3     ✓ stringr 1.4.0
## ✓ readr   1.3.1     ✓ forcats 0.5.0

## ── Conflicts ──────────────────────────────────────
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Tokenization

```
library(tidytext)
# a data frame (tibble) that separates the text into words and stores
# the result in a tibble column called out
tibble(tweet_txt) %>% unnest_tokens(out, tweet_txt, token = "words")
```

```
## # A tibble: 32 x 1
##    out
##    <chr>
##  1 i’m
##  2 starting
##  3 the
##  4 week
##  5 with
##  6 an
##  7 update
##  8 on
##  9 the
## 10 canada
## # … with 22 more rows
```

```
# count the number of words use summarise with n()

tibble(tweet_txt) %>%
  unnest_tokens(out, tweet_txt, token = "words") %>%
  summarise(Num_words = n())
```

```
## # A tibble: 1 x 1
##   Num_words
##       <int>
## 1        32
```

Brief `tidyverse` detour

- A tibble is the `tidyverse` version of a data frame, and in most use-cases the two can be used interchangeably.
- The `%>%` is similar to function composition: $(f \circ g \circ h)(x)$ is analagous to `x %>% h() %>% g() %>% f()`

Brief `tidyverse` detour

```
summarise(unnest_tokens(tibble(tweet_txt),
                        out, tweet_txt,
                        token = "words"), Num_words = n())
```

is the same as

```
tibble(tweet_txt) %>%
  unnest_tokens(out, tweet_txt, token = "words") %>%
  summarise(Num_words = n())
```

We could have done the same using base R functions:

```
length(unlist(strsplit(tweet_txt, split = " ")))
```

Brief `tidyverse` detour

In the "tidy tools manifesto" (see vignettes in `tidyverse`) Hadley Wickham states tht there are four basic principles to a tidy API:

- Reuse existing data structures.
- Compose simple functions with the pipe.
- Embrace functional programming.
- Design for humans.

Using an External API to Access Data

R has several libraries where the objective is to import data into R from an external website such as twitter or PubMed.

rtweet

Twitter has an API that can be accessed via the R library `rtweet`.

```
library(rtweet)

  source("twitter_creds.R")

token <- create_token(
  app = "nlpandstats",
  consumer_key = api_key,
  consumer_secret = api_secret_key)
```

see rtweet article on tokens.

```r
# read csv file
JTtw <- rtweet::read_twitter_csv("JT20tweets_may25.csv")
```

Count the number of words in each tweet.

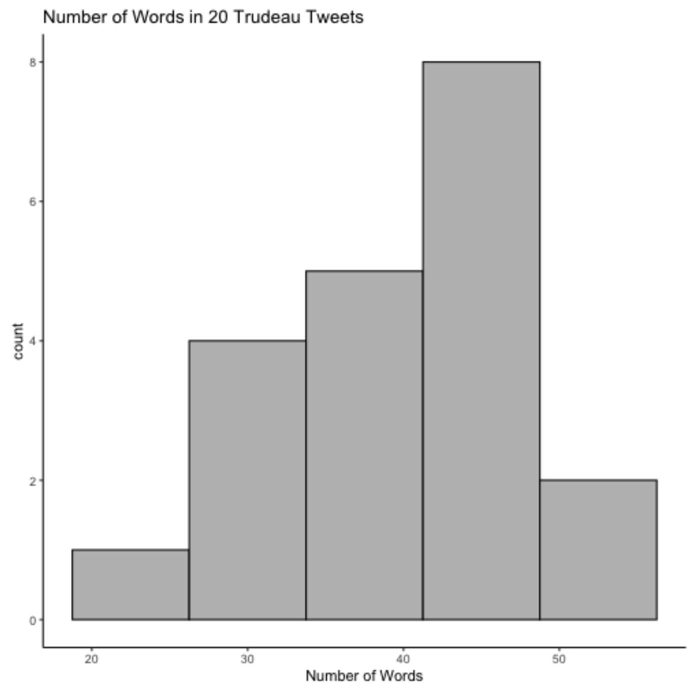
```r
# index each tweet with rowid then count
# rowids
rowid_to_column(JTtw) %>%
  unnest_tokens(out, text, token = "words") %>%
  count(rowid) %>% head()
```

```
## # A tibble: 6 x 2
##   rowid     n
##   <int> <int>
## 1     1    42
## 2     2    47
## 3     3    30
## 4     4    42
## 5     5    38
## 6     6    44
```

Plot the distribution of word counts in JT's tweets.

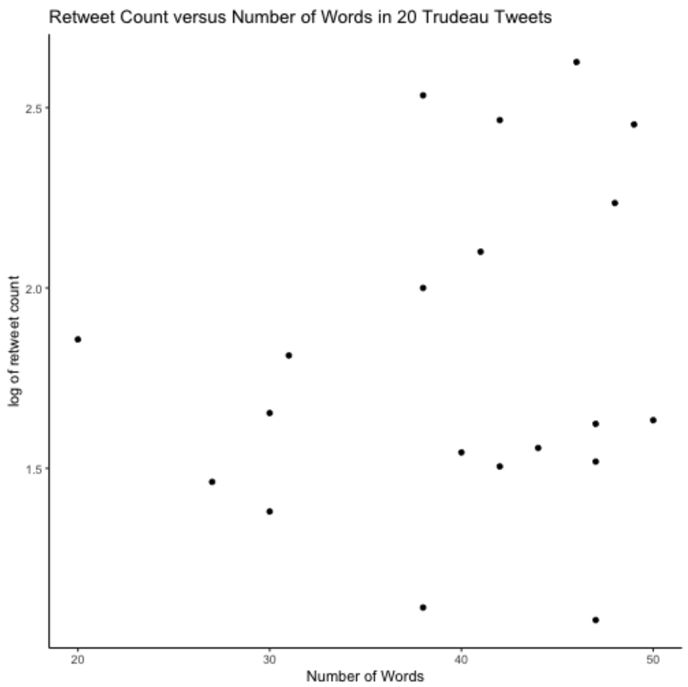
```
# index each tweet with rowid
# then count rowids
text_counts <-
  rowid_to_column(JTtw) %>%
  unnest_tokens(out, text,
                token = "words") %>%
  count(rowid)
```

```
text_counts %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 5,
                 colour = "black",
                 fill = "grey") +
  xlab("Number of Words") +
  ggtitle("Number of Words in 20 Trude
  theme_classic()
```



What about the relationship between retweet count and word count?

```
#merge original data frame to data
# frame with counts by rowid
rowid_to_column(JTtw) %>%
  left_join(text_counts, by = "rowid")
  select(rowid, n, retweet_count) %>%
  ggplot(aes(n, log10(retweet_count)))
  geom_point() + xlab("Number of Words
  ggtitle("Retweet Count versus Number
```



Twitter suggests that hashtags use all caps.

```
JTtw$hashtags
```

```
##  [1] "PKDay"            "PKDay"            NA                 NA
##  [5] NA                 NA                 NA                 NA
##  [9] NA                 NA                 NA                 NA
## [13] NA                 NA                 "COVID19"          "COVID19"
## [17] NA                 NA                 "GlobalGoalUnite" "GlobalGoalUnite"
```

```
# check if hastags are all upper case then count the number
sum(toupper(JTtw$hashtags) == JTtw$hashtags, na.rm = TRUE)
```

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

```
length(JTtw$hashtags)
```

```
## [1] 20
```

```
sum(toupper(JTtw$hashtags) == JTtw$hashtags, na.rm = TRUE)/length(JTtw
```

```
## [1] 0.1
```

R Libraries with Unstructured Text

There are many R libraries with unstructured text available. A few examples include: `geniusr` for song lyrics, and `gutenbergr` for lit.

gutenbergr

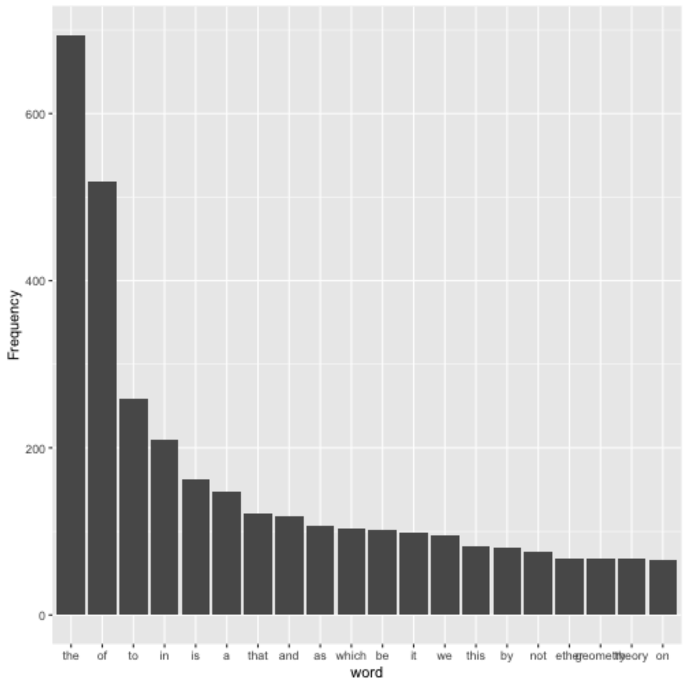
- Project Gutenberg is a free library of mostly older literary works (e.g., Plato and Jane Austen), although there are several non-literary works. There are over 60,000 books.
- `gutenbergr` is a package to help download and process these works.

```
library(gutenbergr)
gutenberg_works(str_detect(author, "Einstein"))
```

```
## # A tibble: 2 x 8
##   gutenberg_id title author gutenberg_autho… language gutenberg_books… rights
##          <int> <chr> <chr>                 <int> <chr>    <chr>            <chr>
## 1         5001 Rela… Einst…                 1630 en       Physics          Publi…
## 2         7333 Side… Einst…                 1630 en       <NA>             Publi…
## # … with 1 more variable: has_text <lgl>
```

```
einstein_talk <- gutenberg_download(7333)
```

```r
einstein_talk %>%
  unnest_tokens(out, text) %>%
  count(out) %>%
  top_n(20) %>%
  ggplot(aes(reorder(out,-n), n)) +
  geom_col() +
  xlab("word") + ylab("Frequency")
```

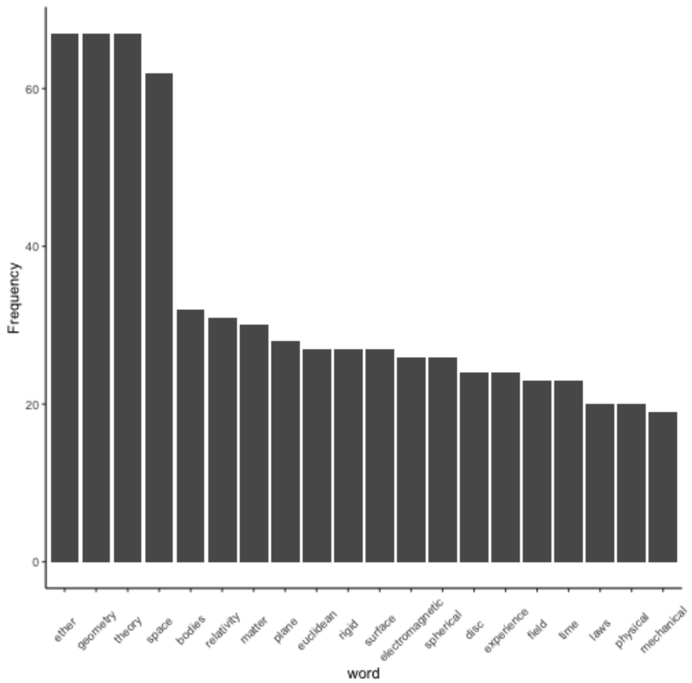


Let's remove stop words such as "the" and "it". `stop_words` is a dataframe of stop words in `tidytext`.

```
stop_words %>% head()
```

```
## # 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
```

```
einstein_talk %>%
  unnest_tokens(out, text) %>%
  anti_join(stop_words,
            by = c("out" = "word")) %>
  count(out) %>%
  top_n(20) %>%
  ggplot(aes(reorder(out,-n), n)) +
  geom_col() +
  xlab("word") + ylab("Frequency") +
  theme_classic() +
  theme(axis.text.x =
          element_text(angle = 45, vju
```



Webscraping

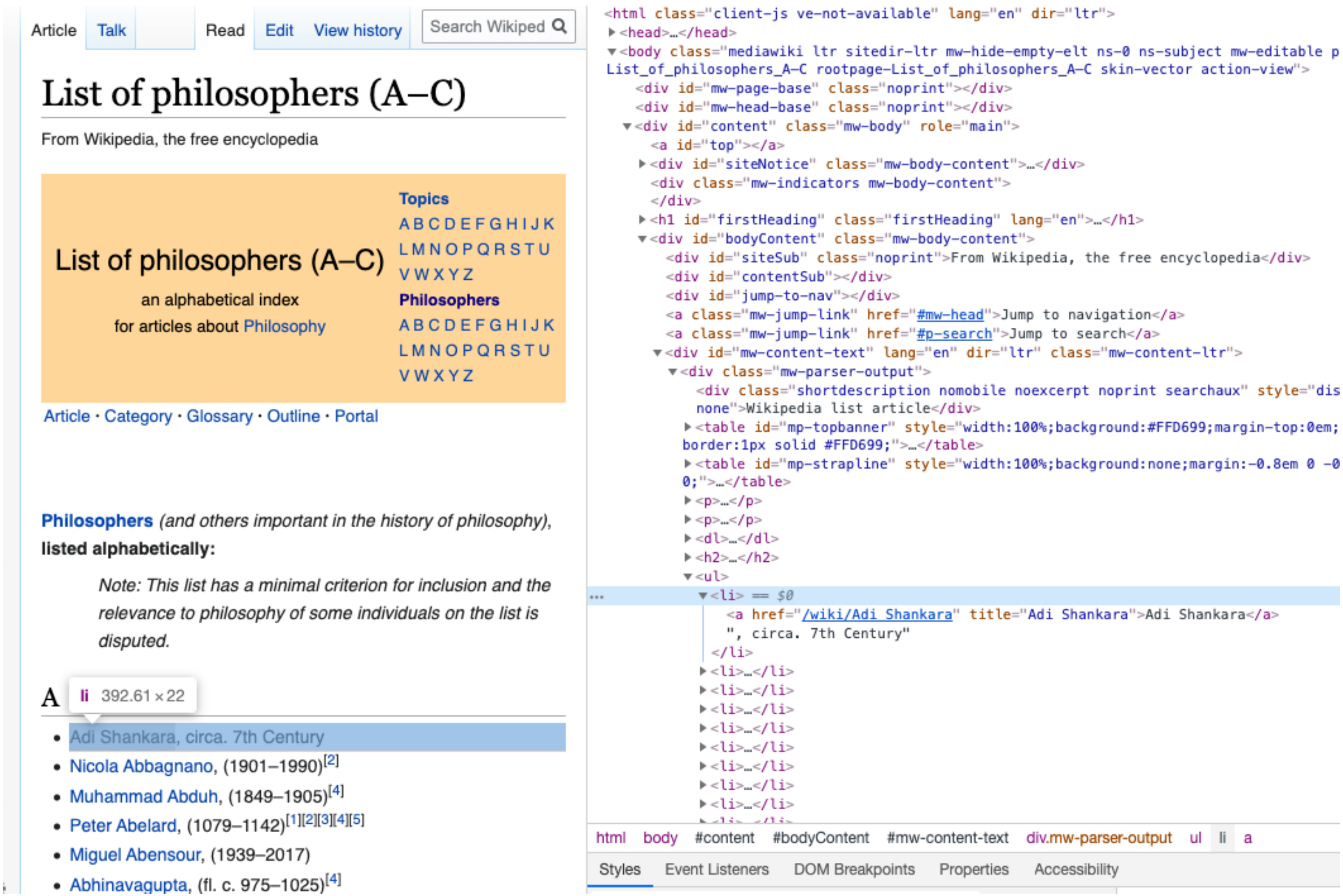
- The Stanford Encyclopedia of Philosophy (SEP) is a "dynamic reference work [that] maintains academic standards while evolving and adapting in response to new research" on philosophical topics. "Entries should focus on the philosophical issues and arguments rather than on sociology and individuals, particularly in discussions of topics in contemporary philosophy. In other words, entries should be "idea-driven" rather than "person-driven".

- Which philosophers appear most frequently in SEP entries?

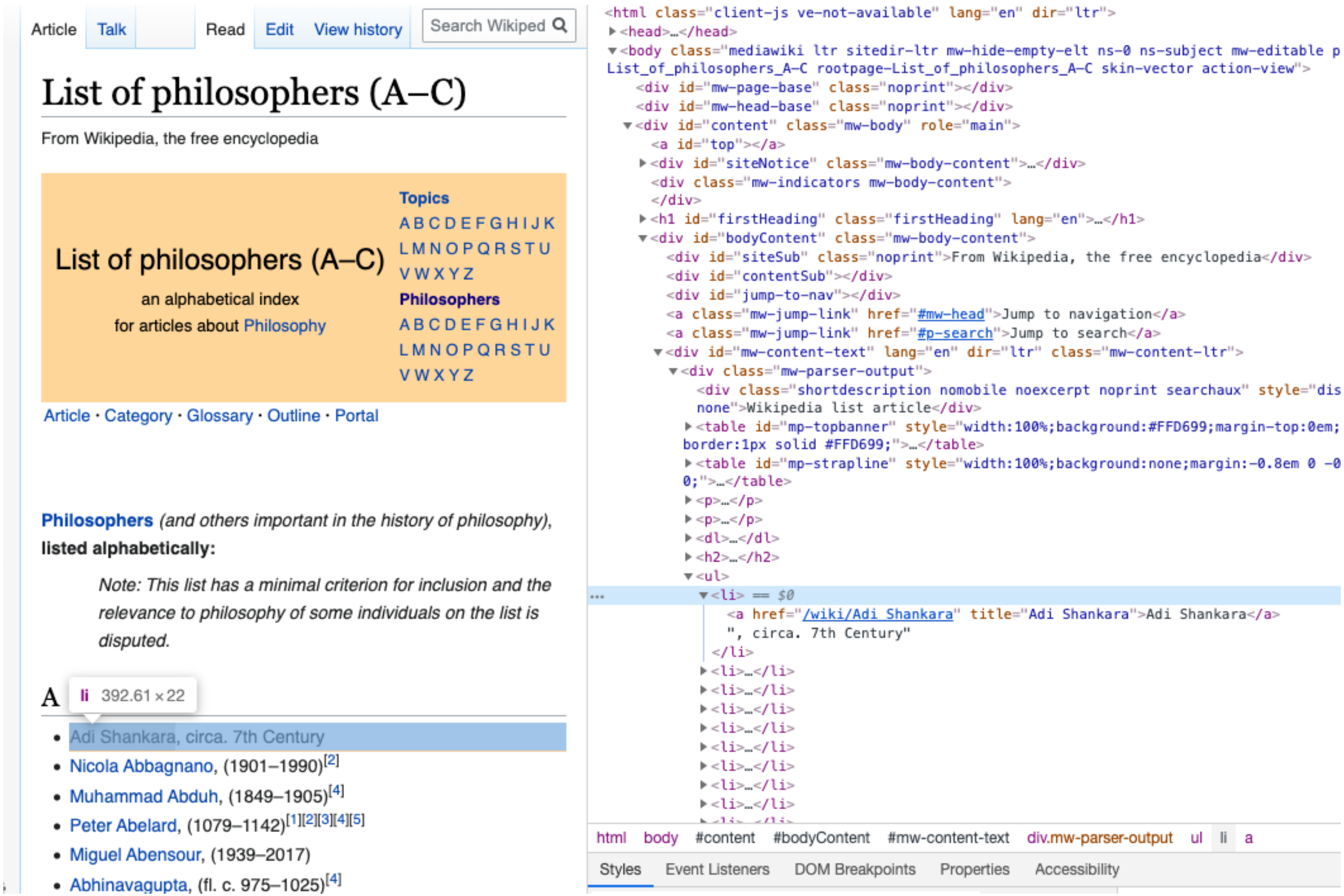
  A list of philosopehrs can be obtained by scraping a few Wikipedia pages using `rvest`.

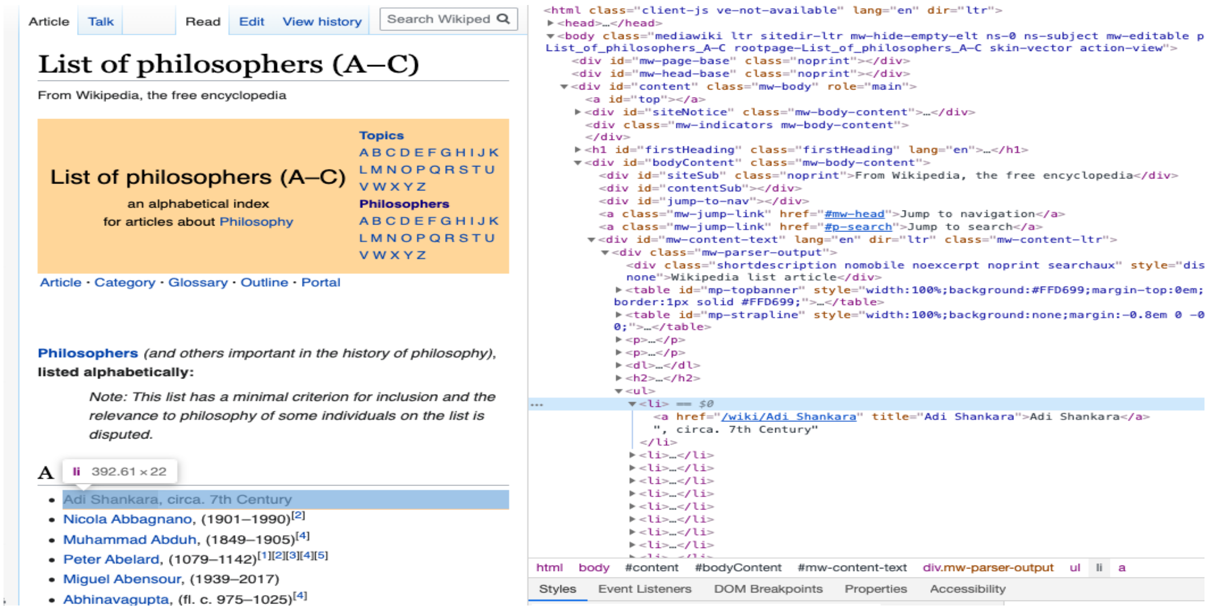
The basic idea is to inspect the page and figure out which part of the page contains the information you want.

- Start by inspecting the element



- The names are stored in an html unordered list `<ul>` as items `<li>`. So, extract these nodes using `html_nodes()` then extract the text using `html_text()`.






```r
library(rvest)

# read the webpage
ac_url <- "https://en.wikipedia.org/wiki/List_of_philosophers_(A%E2%80%93C)"
wiki_philnamesAC <- xml2::read_html(ac_url)

wiki_philnamesAC %>%
  html_nodes("div.mw-parser-output ul li") %>%
  html_text() %>%
  head()
```

```
## [1] "Article"                          "Category"
## [3] "Glossary"                         "Outline"
## [5] "Portal"                           "Adi Shankara, circa. 7th Century"
```

Remove the the first 5 rows using `slice(-(1:5))`.

Write a function to do this for the four Wikipedia pages

```
getphilnames <- function(url, removerows){
  philnames <- xml2::read_html(url) %>%
    html_nodes("div.mw-parser-output ul li") %>%
    html_text() %>%
    tibble(names = .) %>% # rename the column name
    slice(removerows)
  return(philnames)
}
```

```
names_ac <- getphilnames("https://en.wikipedia.org/wiki/List_of_philosophers_(A%E2%80%93C)",
                         -(1:5))
names_dh <- getphilnames("https://en.wikipedia.org/wiki/List_of_philosophers_(D%E2%80%93H)",
                         -(1:5))
names_iq <- getphilnames("https://en.wikipedia.org/wiki/List_of_philosophers_(I%E2%80%93Q)",
                         -(1:5))
names_rz <- getphilnames("https://en.wikipedia.org/wiki/List_of_philosophers_(R%E2%80%93Z)",
                         -(1:5))
wiki_names <- rbind(names_ac, names_dh, names_iq, names_rz)
wiki_names %>% head()
```

```
## # A tibble: 6 x 1
##   names
##   <chr>
## 1 Adi Shankara, circa. 7th Century
## 2 Nicola Abbagnano, (1901–1990)[2]
## 3 Muhammad Abduh, (1849–1905)[4]
## 4 Peter Abelard, (1079–1142)[1][2][3][4][5]
## 5 Miguel Abensour, (1939–2017)
## 6 Abhinavagupta, (fl. c. 975–1025)[4]
```

- We need to extract the names from each entry. This is the same as removing all the text after the comma.
- The regular expression `,.*$` matches all text after (and including) the comma then we can remove is with `str_remove()` (`str_remove()` is vectorized).

```
# the regex ,.*$ matches , and any letter . until
# the end $ of the string
# str_remove() removes the matches

wiki_names <- str_remove(wiki_names$names, ",.*$")
wiki_names %>% head()
```

```
## [1] "Adi Shankara"      "Nicola Abbagnano" "Muhammad Abduh"    "Peter Abelard"
## [5] "Miguel Abensour"   "Abhinavagupta"
```

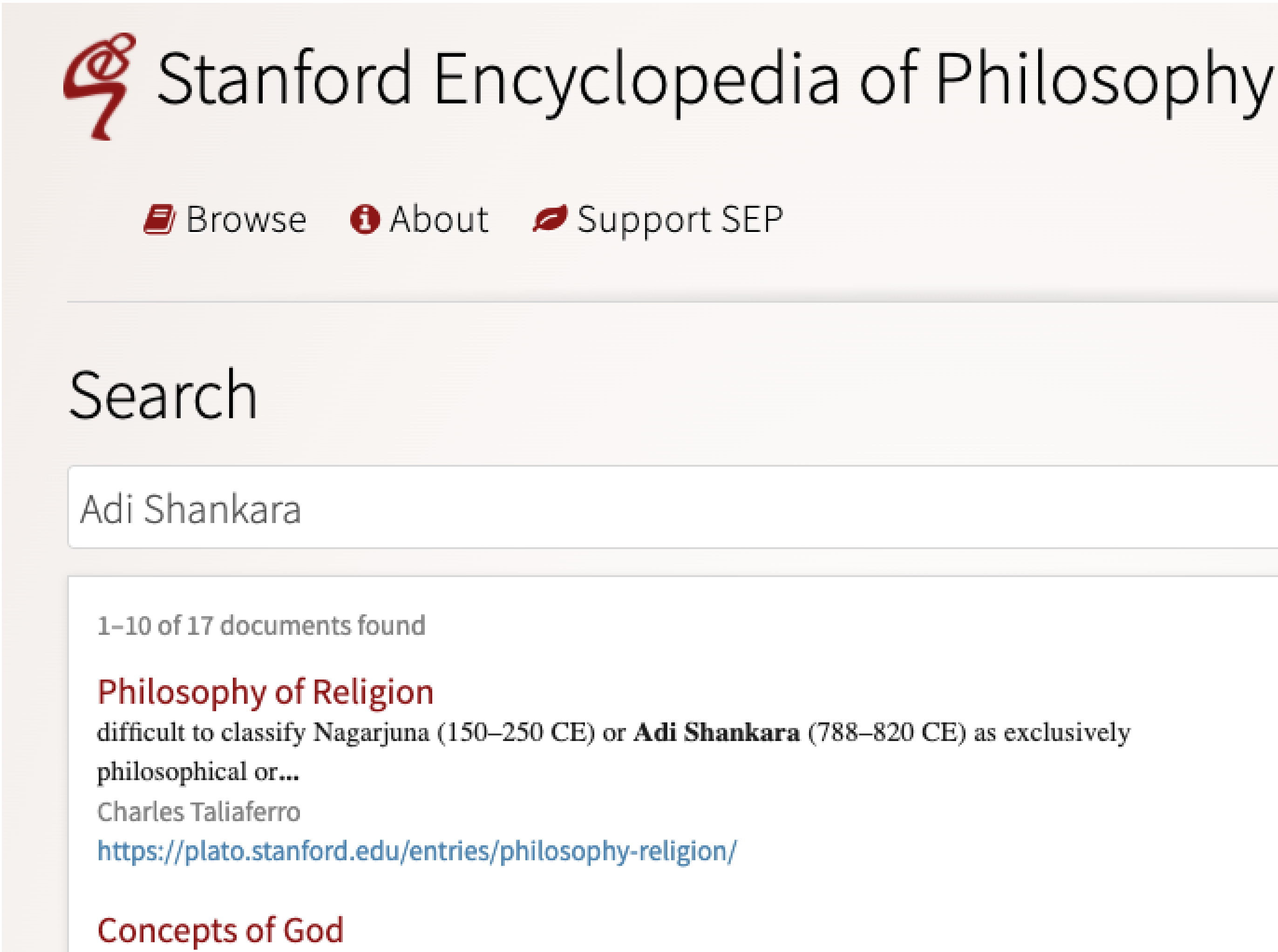
We can use tools in the `RSelenium` library to automate (via programming) web browsing. It is primarily used for testing webapps and webpages across a range of browser/OS combinations.

To run the Selenium Server I'll run the Docker container

```
docker run -d -p 4445:4444 selenium/standalone-firefox:2.53.1
```

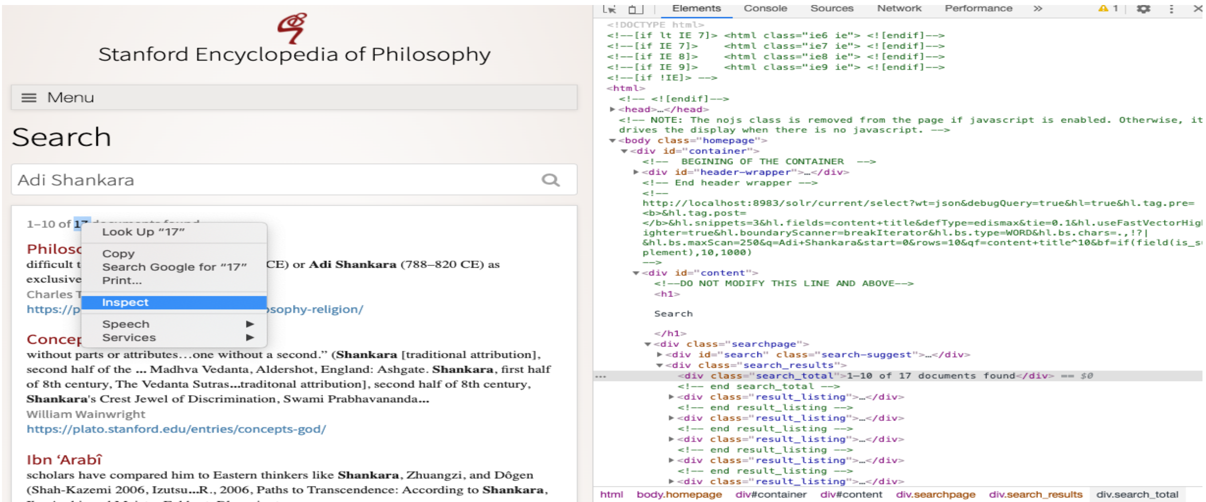
```r
library(RSelenium)

# connect to server
remDr <- remoteDriver(
  remoteServerAddr = "localhost",
  port = 4445L,
  browserName = "firefox"
)

# connect to the server
remDr$open()

#Navigate to <https://plato.stanford.edu/index.html>
remDr$navigate("https://plato.stanford.edu/index.html")

# find search button
webelem <-remDr$findElement(using = "id", value = "search-text")

# input first philosophers name into search
webelem$sendKeysToElement(list(wiki_names[1]))

# find the search button
button_element <- remDr$findElement(using = 'class', value = "icon-search")

# click the search button
button_element$clickElement()
```

There are 17 entries where Adi Shankara is found.



How can we extract 17 from the webpage?

Let's inspect the element that corresponds to 17.



Find the element related to `search_total` then extract the text.

```
# find element
out <- remDr$findElement(using = "class", value="search_total")
# extract text
tot <- out$getElementText()
tot
```

```
## [[1]]
## [1] "1-10 of 17 documents found"
```

Now, use a regular expression to extract the number just before documents.

```
# extract the number of dicuments
str_extract(tot[[1]],"\\d+(?= documents)")
```

```
## [1] "17"
```

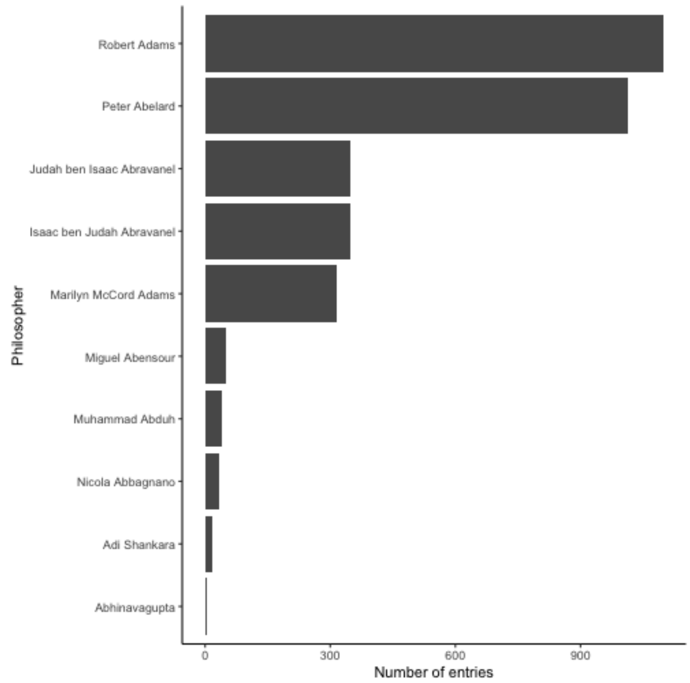
```
# show the match
str_view_all(tot[[1]],"\\d+(?= documents)")
```

1–10 of 17 documents found

Now create a function to do this so that we can interate. It's good practice to add a delay using `Sys.sleep()` so that we don't stretch the SEP server capacity.

```
getcount <- function(i){
  Sys.sleep(0.05)
  remDr$navigate("https://plato.stanford.edu/index.html")
  webelem <-remDr$findElement(using = "id", value = "search-text")
  webelem$sendKeysToElement(list(wiki_names[i]))
  button_element <- remDr$findElement(using = 'class', value = "icon-se
  button_element$clickElement()
  out <- remDr$findElement(using = "class", value="search_total")
  tot <- out$getElementText()
  str_view_all(tot[[1]],"\\d+(?= documents)")
  tot <- str_extract(tot[[1]],"\\d+(?= documents)")
  return(as.numeric(tot))
}
```

```r
# Let's look at the first 10 names
# tidyverse approach
# in base R could use
# sapply(1:10, getcount, simplify = TR

counts <- 1:10 %>%
  map(getcount) %>%
  flatten_dbl()

tibble(names = wiki_names[1:10],
       counts) %>%
  drop_na() %>%
  ggplot(aes(reorder(names,counts),
             counts)) +
  geom_col() +
  coord_flip() + theme_classic() +
  ylab("Number of entries") +
  xlab("Philosopher")
```



Regular Expressions

A formal language for specifying text strings.

How can we search for any of these?

- cat
- cats
- Cat
- Cats

Regular Expressions: Characters

- In R `\\` represents `\`
- `\w` matches any word and `\W` matches any non-word characters such as `.` So to use this as a regular expression

```
str_view_all("Do you agree that stats is popular?", "\\w")
```

```
Do you agree that stats is popular?
```

- `\d` matches any digit,
- `\D` matches any non-digit,
- `.` matches every character except new line.
- `\s` matches any whitespace

```
str_view_all("There are at least 200 people in this zoom!", "\\d")
```

```
There are at least 200 people in this zoom!
```

Regular Expressions: Alternates (Disjunctions)

- Letters inside square brackets `[]` or use pipe `|`.

```
# tidyverse style
str_view_all(string = c("cat","Cat"), pattern = "[cC]") # match c or C
```

cat

Cat

Regular Expressions: Alternates (Disjunctions)

- Ranges `[A-Z]`, `[a-z]`, `[0-9]`, `[:digit:]`, `[:alpha:]`

```
str_view(string = c("cat","Cat 900"), pattern = "[0-9]") # first match
```

```
cat

Cat 900
```

- Negations in disjunction `[^A]`.
- When `^` is first in `[]` it means negation. For example, `[^xyz]` means neither `x` nor `y` nor `z`.

```
str_view_all(string = c("xenophobia causes problems"),
             pattern = "[^cxp]") # neither c nor x nor p
```

```
xenophobia causes problems
```

Regular Expressions: Anchors

- `^a` matches `a` at the start of a string and `a$` matches `a` at the end of a string.

```
str <- c("xenophobia causes problems", "Xenophobia causes problems")
# x at the beginning or s at the end of string
str_view_all(string = str,
             pattern = "^x|s$")
```

```
xenophobia causes problems
Xenophobia causes problems
```

Regular Expressions: Quantifiers

- `a?` matches exactly zero or one `a`
- `a*` matches zero or more `a`
- `a+` matches one or more `a`
- `a{3}` matches exactly 3 `a`'s

```
str_view_all(string = c("colour","color","colouur"),
             pattern = "colou+r")
```

colour

color

colouur

Regular Expressions: Groups/Precedence

- How can I specify both puppy and puppies?

```
# disjuction only applies to suffixes y and ies
str_view_all(string = c("puppy", "puppies"), pattern = "pupp(y|ies)")
```

```
puppy
puppies
```

Example from p-value paper

Original Investigation FREE

March 15, 2016

Evolution of Reporting *P* Values in the Biomedical Literature, 1990-2015

David Chavalarias, PhD[1]; Joshua David Wallach, BA[2]; Alvin Ho Ting Li, BHSc[3]; John P. A. Ioannidis, MD, DSc[4]

Author Affiliations | Article Information

JAMA. 2016;315(11):1141-1148. doi:10.1001/jama.2016.1952

Let's look at extracting all the variations of *p value* from a few sentences using the regular expression provided in the appendix of Chavalarias et al. (2016).

```
"/(\s|\()[Pp]{1}(\s|-)*(value|values)?(\s)*([=<>≤≥]|less than|of<)+(\s
```

Namely,

```
(\s|\()[Pp]{1}(\s|-)*(value|values)?(\s)
```

- `(\s|\()` whitespace or `(`
- `[Pp]{1}` Match either `P` or `p`.
- `(\s|-)*` whitespace or `-` zero or more times
- `(value|values)?` zero or one times
- `(\s)` whitespace

```r
str <- c("The result was significant (p < ",
         "The result was significant P = ",
         "The result was not significant p-value(s) less than ",
         "The result was significant P-value(s) ≤ ",
         "The result was significant P value(s) < ",
         "The result was significant p value(s) < ")

ppat <- "(\\s|\\()[Pp]{1}(\\s|-)*(value|values)?(\\s)"
str_view_all(str, pattern = ppat)
```

```
The result was significant (p <
The result was significant P =
The result was not significant p-value(s) less than
The result was significant P-value(s) ≤
The result was significant P value(s) <
The result was significant p value(s) <
```

But this doesn't capture "p value(s)". Is there a mistake in the data extraction?

What should be added to the regular expression?

```
ppat <- "(\\s|\\()[Pp]{1}(\\s|-)*(value|values|value\\(s\\))?(\\s)"
str_view_all(str, pattern = ppat)
```

```
The result was significant (p <
The result was significant P =
The result was not significant p-value(s) less than
The result was significant P-value(s) ≤
The result was significant P value(s) <
The result was significant p value(s) <
```

N-Grams

- Models that assign probabilities to sequences of words are called language models.
- An n-gram is a sequence of n words.
- A 1-gram or unigram is one word sequence.
- A 2-gram or bigram is a two word sequence.
- A 3-gram or trigram is a three word sequence

- Suppose we want to compute the probability of a word W given some history H, $P(W|H)$
- The sentence "He had successfully avoided meeting his landlady ..." is at the beginning of Crime and Punishment by Fyodor Dostoevsky.
- Let $h = ``\text{He had successfully avoided meeting his}"$ and $w = ``\text{landlady}"$:

  Estimate using the relative frequency of counts:

  $$\frac{\#\ ``\text{He had successfully avoided meeting his landlady}"}{\#\ ``\text{He had successfully avoided meeting his}"}$$

- This could be estimated using counts from searching the web using, say, Google.
- But, new sentences are created all the time so it's difficult to estimate.

- If we want to know the joint probability of an entire sequence of words like "He had successfully avoided meeting his" “out of all possible sequences of six words, how many of them are, "He had successfully avoided meeting his"?

$$P(\text{landlady}|\text{He had successfully avoided meeting his})$$

$$P(X_7 = \text{``landlady''}|X_1 = \text{``He''}, X_2 = \text{``had''},\\ X_3 = \text{``successfully''}, \ldots, X_6 = \text{``his''})$$

The bigram model approximates this probability using the Markov assumption.

$$P(X_7 = \text{``landlady''}|X_1 = \text{``He''}, X_2 = \text{``had''},\\ X_3 = \text{``successfully''}, \ldots,\\ X_6 = \text{``his''}) \approx\\ P(X_7 = \text{``landlady''}|X_6 = \text{``his''})$$

How can this be computed?

- $C_{\text{his landlady}}$ = count the number of bigrams that are "his landlady"
- $C_{\text{his ...}}$ = count the number of bigrams that have first word "his"
- $C_{\text{his landlady}}/C_{\text{his ...}}$

Compute the probability of the bigram "his landlady" in Crime and Punishment.

```
crimeandpun <- gutenberg_download(gutenberg_id = 2554) %>%
  slice(-(1:108))  # remove preface, etc.

crimeandpun %>% unnest_ngrams(output = out, input = text, n = 2) %>%
  mutate(out = tolower(out),
         bigram_xy = str_detect(out, "his landlady"), # Boolean for his landlady
         bigram_x = str_detect(out, "^his")) %>%  # Boolean for his ...
  filter(bigram_x == TRUE) %>%
  group_by(bigram_xy) %>%
  count() %>% # creates the variable n for each group
  ungroup() %>% # ungroup so we can sum n's in each group
  mutate(tot = sum(n), percent=round(n/tot,3))
```

```
## # A tibble: 2 x 4
##   bigram_xy     n   tot percent
##   <lgl>     <int> <int>   <dbl>
## 1 FALSE      2090  2100   0.995
## 2 TRUE         10  2100   0.005
```

Word Vectors and Matrices

- Words that occur in similar contexts tend to have similar meanings. The idea is that "a word is characterized by the company it keeps" (Firth, 1957).
- For example, *oculist* and *eye doctor* tended to occur near words like *eye* or *examined*.
- This link between similarity in how words are distributed and similarity in what they mean is called the **distributional hypothesis**.
- A computational model that deals with different aspects of word meaning is to define a word as a vector in N dimensional space, although the vector can be defined in different ways.

Term Document Matrix

How often do the the words (term), battle, good, fool, and wit occur in a particular Shakespeare play (document)?

```r
library(janitor)

AsYouLikeIt <- gutenberg_download(1523) %>%
  add_column(book = "AsYouLikeIt") %>%
  slice(-c(1:40))
TwelfthNight <- gutenberg_download(1526) %>%
  add_column(book = "TwelfthNight") %>%
  slice(-c(1:31))
JuliusCaesar <- gutenberg_download(1785) %>%
  add_column(book = "JuliusCaesar") %>%
  slice(-c(1:291))
HenryV <- gutenberg_download(2253) %>%
  add_column(book = "HenryV") %>%
  slice(-c(1:94))

shakespeare_books <- rbind(AsYouLikeIt,TwelfthN

shakespeare_books %>%
  unnest_tokens(out, text) %>%
  mutate(out = tolower(out)) %>%
  filter(out == "battle"|out == "good" | out ==
  group_by(book, out) %>%
  tabyl(out, book) %>% knitr::kable() %>% kable
```

out	AsYouLikeIt	HenryV	JuliusCaesar	TwelfthNight
battle	1	0	8	0
fool	36	0	1	58
good	115	91	71	80
wit	21	3	2	15

- This is an example of a **term-document matrix**: each row represents a word in the volcabulary and each column represents a document from some collection of documents.

out	AsYouLikeIt	HenryV	JuliusCaesar	TwelfthNight
battle	1	0	8	0
fool	36	0	1	58
good	115	91	71	80
wit	21	3	2	15

- The table above is a small selection from the larger term-document matrix.
- A document is represented as a count vector. If $|V|$ is the size of the vocabulary (e.g., all the words in a document) then each document is a point in $|V|$ dimensional space.

TF-IDF

- Simple frequency isn't the best measure of association between words.
- One problem is that raw frequency is very skewed and not very discriminative.
- The dimension for the word good is not very discriminative between Shakespeare plays; good is simply a frequent word and has roughly equivalent high frequencies in each of the plays.
- It's a bit of a paradox. Words that occur nearby frequently (maybe pie nearby cherry) are more important than words that only appear once or twice. Yet words that are too frequent are unimportant. How can we balance these two conflicting constraints?

Term Frequency

- term frequency is the frequency of word t in document d. In the `tidytext` package it's computed as:

$$f_{t,d} / \sum_{t' \in d} f_{t',d}$$

- $f_{t,d}$ is the count of term t in document d.
- $\sum_{t' \in d} f_{t',d}$ is the total number of terms in d.

The Shakespeare example below assumes that each document only has four words. So, if $d =$ "As you like it" and $t =$ "battle" then term frequency is, 1/(1+36+115+21) = 0.0057803.

out	AsYouLikeIt	HenryV	JuliusCaesar	TwelfthNight
battle	1	0	8	0
fool	36	0	1	58
good	115	91	71	80
wit	21	3	2	15

Inverse Document Frequency

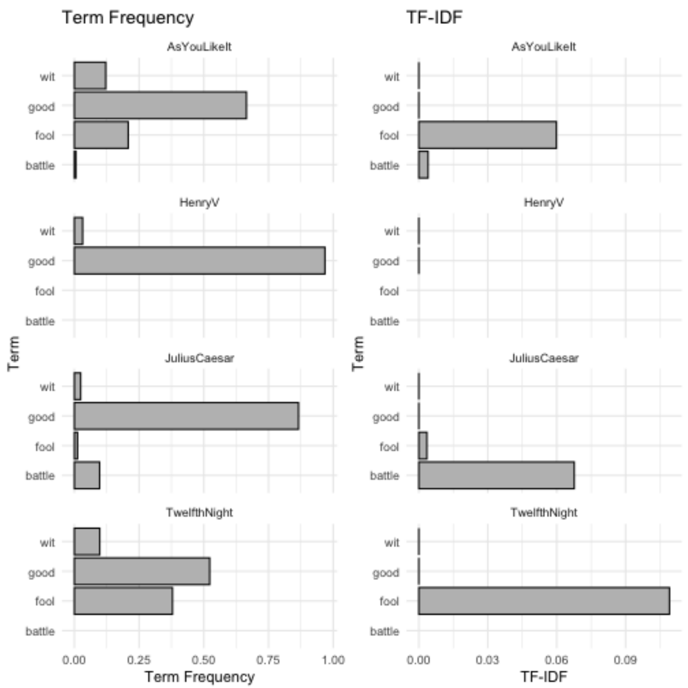
- Terms that are limited to a few documents are useful for discriminating those documents from the rest of the collection.
- Terms that occur frequently across the entire collection aren't as helpful.
- Let $n_{\text{documents}}$ be the number of documents in the collection, and $n_{\text{documents containing term}}$ be the number of documents containg the term. Inverse document frequency is defined as:

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

```r
tf_idf <- shakespeare_books %>%
  unnest_tokens(out, text) %>%
  mutate(out = tolower(out)) %>%
  filter(out == "battle"|out == "good" | out == "fool"|out == "wit") %>
  group_by(book, out) %>%
  count() %>%
  bind_tf_idf(term = out, document = book, n = n)

tf_idf %>% knitr::kable() %>% kableExtra::kable_styling(font_size = 9)
```

book	out	n	tf	idf	tf_idf
AsYouLikeIt	battle	1	0.0057803	0.6931472	0.0040066
AsYouLikeIt	fool	36	0.2080925	0.2876821	0.0598645
AsYouLikeIt	good	115	0.6647399	0.0000000	0.0000000
AsYouLikeIt	wit	21	0.1213873	0.0000000	0.0000000
HenryV	good	91	0.9680851	0.0000000	0.0000000
HenryV	wit	3	0.0319149	0.0000000	0.0000000
JuliusCaesar	battle	8	0.0975610	0.6931472	0.0676241
JuliusCaesar	fool	1	0.0121951	0.2876821	0.0035083
JuliusCaesar	good	71	0.8658537	0.0000000	0.0000000
JuliusCaesar	wit	2	0.0243902	0.0000000	0.0000000
TwelfthNight	fool	58	0.3790850	0.2876821	0.1090559
TwelfthNight	good	80	0.5228758	0.0000000	0.0000000
TwelfthNight	wit	15	0.0980392	0.0000000	0.0000000

```r
library(gridExtra)

p1 <- tf_idf %>% ggplot(aes(out,tf)) +
  geom_col(fill= "grey", colour = "bla
  facet_wrap(~book, nrow = 4) + ggtitl

p2 <- tf_idf %>% ggplot(aes(out,tf_idf
  geom_col(fill= "grey", colour = "bla
  facet_wrap(~book, nrow = 4) + ggtitl

grid.arrange(p1,p2, ncol = 2)
```



Questions?