Web Scraping and Text Data

Thomas Bryan Smith*

February 07, 2025

Contents

1 Introduction

We are going to be working with **Miranda v. Arizona, 384 U.S. 436 (1966)**, the case that mandated suspects are advised of their rights (right to counsel, to remain silent, etc.) clearly and explicitly before questioning. This is a good example to work with on account of its length. We want a good number of paragraphs to work with in order to demonstrate how to work with text data.

Follow this link to the document:

Miranda v. Arizona, 384 U.S. 436 (1966)

We are going to scrape the text from this page, store it as an R object, and then talk about the different ways we can represent the document as numbers (rather than words). This document, divided into its constituent paragraphs (each treated as an individual document), will be our proxy for a more traditional ‘corpus’. A corpus is just a collection of written text documents.

First, we need to load the **rvest** package, and then scrape the raw HTML code from the website.

Although, it should be noted that R is a restrictive software when performing natural language processing. At a certain point of working with text data, you will need to graduate from R to Python. The majority of advanced natural language processing tools are built-in and (sometimes only) available in Python.

*University of Mississippi, tbsmit10@olemiss.edu

2 Web Scraping

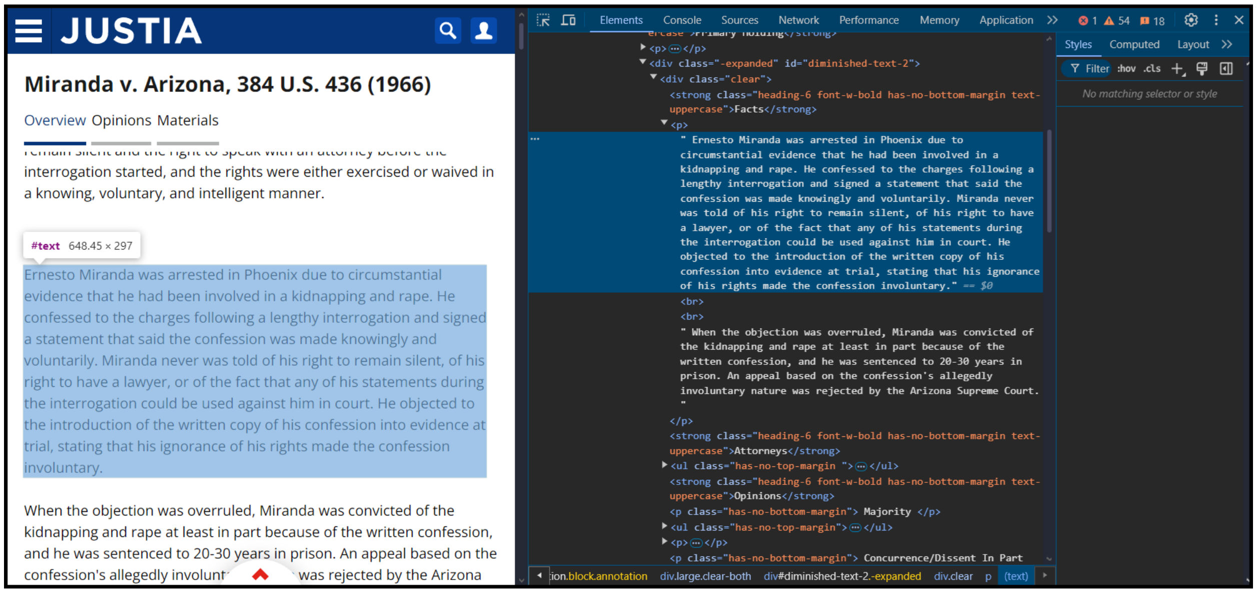
2.1 rvest and reading HTML code

The `read_html()` function, when targeting the URL of the page you wish to scrape, will 'harvest' the HTML code for that page. You then assign, `<-`, that to an appropriately named object (e.g., `html`). The `minimal_html()` function will then minimize the amount of HTML code you will need to remove in order to work with the text data. Finally, printing the `html` object shows us what we are working with.

```r
library(rvest)

html <- read_html("https://supreme.justia.com/cases/federal/us/384/436/")
html <- minimal_html(html)
html
```

```
## {html_document}
## <html>
## [1] <head>\n<meta charset="utf-8">\n<title></title>\n<meta http-equiv="Conten ...
## [2] <body class="  content-left-sidebar sticky-footer bg-white sticky-sidebar ...
```



Miranda v. Arizona, 384 U.S. 436 (1966)

Overview Opinions Materials

interrogation started, and the rights were either exercised or waived in a knowing, voluntary, and intelligent manner.

Ernesto Miranda was arrested in Phoenix due to circumstantial evidence that he had been involved in a kidnapping and rape. He confessed to the charges following a lengthy interrogation and signed a statement that said the confession was made knowingly and voluntarily. Miranda never was told of his right to remain silent, of his right to have a lawyer, or of the fact that any of his statements during the interrogation could be used against him in court. He objected to the introduction of the written copy of his confession into evidence at trial, stating that his ignorance of his rights made the confession involuntary.

When the objection was overruled, Miranda was convicted of the kidnapping and rape at least in part because of the written confession, and he was sentenced to 20-30 years in prison. An appeal based on the confession's allegedly involunt was rejected by the Arizona

```
▶<p>⋯</p>
▼<div class="-expanded" id="diminished-text-2">
  ▼<div class="clear">
     <strong class="heading-6 font-w-bold has-no-bottom-margin text-uppercase">Facts</strong>
    ▼<p>
        " Ernesto Miranda was arrested in Phoenix due to circumstantial evidence that he had been involved in a kidnapping and rape. He confessed to the charges following a lengthy interrogation and signed a statement that said the confession was made knowingly and voluntarily. Miranda never was told of his right to remain silent, of his right to have a lawyer, or of the fact that any of his statements during the interrogation could be used against him in court. He objected to the introduction of the written copy of his confession into evidence at trial, stating that his ignorance of his rights made the confession involuntary." == $0
        <br>
        <br>
        " When the objection was overruled, Miranda was convicted of the kidnapping and rape at least in part because of the written confession, and he was sentenced to 20-30 years in prison. An appeal based on the confession's allegedly involuntary nature was rejected by the Arizona Supreme Court.
        "
     </p>
     <strong class="heading-6 font-w-bold has-no-bottom-margin text-uppercase">Attorneys</strong>
    ▶<ul class="has-no-top-margin ">⋯</ul>
     <strong class="heading-6 font-w-bold has-no-bottom-margin text-uppercase">Opinions</strong>
     <p class="has-no-bottom-margin"> Majority </p>
    ▶<ul class="has-no-top-margin">⋯</ul>
    ▶<p>⋯</p>
     <p class="has-no-bottom-margin"> Concurrence/Dissent In Part
```

```
ion.block.annotation  div.large.clear-both  div#diminished-text-2.-expanded  div.clear  p  (text)
```

Figure 1: Inspecting HTML elements in your web browser.

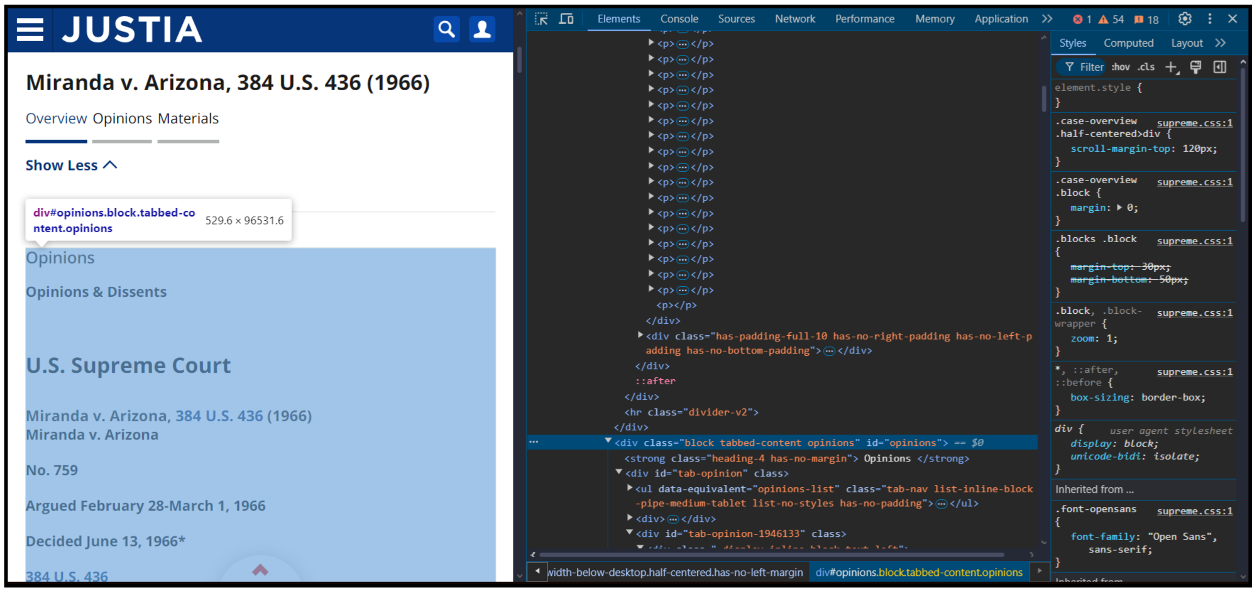
2.2 Extracting the 'right' information

The next goal is to extract the information you are seeking to analyze. The amount of effort here depends heavily on on if the target web page is well-constructed. Poorly constructed web pages will be much more difficult to scrape than well-constructed web pages.

Inspecting the HTML elements of law.justia.com shows us that the text chunks are typically formatted using the paragraph, `<p>`, html tag (see Figure 1). This suggestsa that law.justia was reasonably well constructed, but still not perfect!

It is important that you inspect all of the different sections of a web page so you can verify that *all* of the information you are wanting to extract is accessible under the identified html tag. If you look at figure 2, you will notice that this is absolutely not the case. When you inspect the HTML element associated with the complete supreme court opinion, it has been stored under the following html tag:

`<div id="tab-opinion-1946133" class>`


JUSTIA
Overview Opinions Materials
Show Less
div#opinions.block.tabbed-content.opinions 529.6 × 96531.6
Elements Console Sources Network Performance Memory Application
1 54 18
Styles Computed Layout
Filter :hov .cls

Miranda v. Arizona, 384 U.S. 436 (1966)

Opinions

Opinions & Dissents

U.S. Supreme Court

Miranda v. Arizona, 384 U.S. 436 (1966)
Miranda v. Arizona

No. 759

Argued February 28-March 1, 1966

Decided June 13, 1966*

384 U.S. 436

```
<p>…</p>
<p>…</p>
<p>…</p>
<p>…</p>
<p>…</p>
<p>…</p>
<p>…</p>
<p>…</p>
<p>…</p>
<p>…</p>
<p>…</p>
<p>…</p>
<p>…</p>
<p>…</p>
<p>…</p>
<p>…</p>
<p>…</p>
<p></p>
</div>
<div class="has-padding-full-10 has-no-right-padding has-no-left-padding has-no-bottom-padding">…</div>
</div>
::after
</div>
<hr class="divider-v2">
</div>
<div class="block tabbed-content opinions" id="opinions"> == $0
<strong class="heading-4 has-no-margin"> Opinions </strong>
<div id="tab-opinion" class>
<ul data-equivalent="opinions-list" class="tab-nav list-inline-block-pipe-medium-tablet list-no-styles has-no-padding">…</ul>
<div>…</div>
<div id="tab-opinion-1946133" class>
```

width-below-desktop.half-centered.has-no-left-margin div#opinions.block.tabbed-content.opinions

```
element.style {
}
.case-overview          supreme.css:1
.half-centered>div {
  scroll-margin-top: 120px;
}
.case-overview          supreme.css:1
.block {
  margin: ▶ 0;
}
.blocks .block          supreme.css:1
{
  ~~margin-top: 30px;~~
  ~~margin-bottom: 50px;~~
}
.block, .block-         supreme.css:1
wrapper {
  zoom: 1;
}
*, ::after,             supreme.css:1
::before {
  box-sizing: border-box;
}
div {       user agent stylesheet
  display: block;
  unicode-bidi: isolate;
}
Inherited from ...
.font-opensans          supreme.css:1
{
  font-family: "Open Sans",
    sans-serif;
}
```

Figure 2: It is important to be thorough when inspecting HTML code, ensuring that you are identifying all of the HTML tags you want to target.

So, we want to do the following:

- Extract all text from code with the `<p>` HTML tag.
- Extract all text from the html element with the specific tag, `<div id="tab-opinion-1946133" class>`.

Let's write some code to do this:

```r
syllabus <- html |>
  html_elements("p") |>
  html_text2()

opinion <- html |>
  html_nodes("div[id='tab-opinion-1946133']") |>
  html_text()
```

Breaking this code down, we are doing the following:

- Identify the `html` object, which contains the raw html data scraped from our target URL.
- Pipe this object, `|>`, to the function that will target the HTML elements (and node) from which we want to extract the text.
  - `html_elements()` should be used to 'target' all HTML elements with the `<p>` tag.
  - `html_nodes()` should be used to 'target' the HTML node with the `<div>` tag and the `tab-opinion-1946133` id.

- Append an additional pipe, `|>`, to the `html_text()` or the `html_text2()` function. This will extract the text data from each.
  - `html_text()` extracts text with formatting notation (e.g., `\n` for new lines).
  - `html_text2()` extracts text without formatting notation.

Note that we apply html_text2() to the `<p>` elements, because `<p>` denotes a 'paragraph' on the web page. We want our data to be split into paragraphs, so we do not need to worry about retaining text formatting. However, since the `<div id="tab-opinion-1946133" class>` HTML node is associated with an *entire* Supreme Court opinion, and there are no formatting tags in the underlying HTML code, we need the formatting information to split the data into paragraphs.

2.3 Cutting and pasting

Now that we have the text chunks we want to work with, we need to do a little bit of cleaning. Here, I am only going to show you how to *split* text chunks. Typically, you would need to perform many more additional steps to ensure your text are 'clean' enough for pre-processing and analysis.

The text within the `opinion` object consistently (and conveniently) splits the paragraphs using the `\t` special formatting character. We can use this to split the string into many different chunks and paragraphs. Note that the `str_split()` function creates a list object, so we also need to extract the first vector from that list object (i.e., `opinion[[1]]`). Due to sections of poor or incomplete HTML encoding, there are many repeated `\r`, which will create a lot of empty elements in our character vector. so, we also need to filter out all strings which contain no characters (i.e., `nchar() > 0`).

```r
library(stringr)

opinion <- opinion |>
  str_split("\t")

opinion <- opinion[[1]]

opinion <- subset(opinion, nchar(opinion) > 0)

head(opinion, 18)
```

```
##  [1] "\n"
##  [2] "\n"
##  [3] "\nU.S. Supreme Court\nMiranda v. Arizona, 384\nU.S. 436 (1966)\n\n"
##  [4] "Miranda v. Arizona\n\n"
##  [5] "No. 759\n\n"
##  [6] "Argued February 28-March 1, 1966\n\n"
##  [7] "Decided June 13, 1966*\n\n"
##  [8] "384\nU.S. 436\n\n"
##  [9] "CERTIORARI TO THE SUPREME COURT OF ARIZONA\n\n"
## [10] "MR. CHIEF JUSTICE WARREN delivered the opinion of the Court.\n\n"
## [11] "The cases before us raise questions which go to the roots of our\nconcepts of American crimina
## [12] "[440]\n\n"
## [13] "We dealt with certain phases of this problem recently in\nEscobedo v. Illinois, 378 U. S. 478
## [14] "This case has been the subject of judicial interpretation and\nspirited legal debate since it
## [15] "[441]\n\n"
## [16] "have speculated on its range and desirability. [Footnote 3] We granted certiorari in these\nca
## [17] "[442]\n\n"
## [18] "concrete constitutional guidelines for law enforcement agencies\nand courts to follow.\n\n"
```

However, as you can see from the first 18 elements of the `opinion` vector, the `str_split()` function has not perfectly split our text into paragraphs. Any instance where the opinion introduces a citation (e.g. `[440]`) the paragraph has been split. Additionally, note that some of the special formatting code remains, specifically the `\n` (i.e., new line) code.

So, first, we will remove use regular expressions and the `gsub()` function to remove `\n` from every string element of the opinion character vector. We will also use `trimws()` to remove leading and trailing spaces.

```r
# Remove \n special characters from opinion strings
head(opinion <- gsub("\n", " ", opinion))
```

```
## [1] " "
## [2] " "
## [3] " U.S. Supreme Court Miranda v. Arizona, 384 U.S. 436 (1966)  "
## [4] "Miranda v. Arizona  "
## [5] "No. 759  "
## [6] "Argued February 28-March 1, 1966  "
```

```r
# Remove leading and trailing whitespace
head(opinion <- trimws(opinion))
```

```
## [1] ""
## [2] ""
## [3] "U.S. Supreme Court Miranda v. Arizona, 384 U.S. 436 (1966)"
## [4] "Miranda v. Arizona"
## [5] "No. 759"
## [6] "Argued February 28-March 1, 1966"
```

```r
# Remove newly empty elements
head(opinion <- subset(opinion, nchar(opinion) > 0))
```

```
## [1] "U.S. Supreme Court Miranda v. Arizona, 384 U.S. 436 (1966)"
## [2] "Miranda v. Arizona"
## [3] "No. 759"
## [4] "Argued February 28-March 1, 1966"
## [5] "Decided June 13, 1966*"
## [6] "384 U.S. 436"
```

Things are looking better, but we still have to solve the issue where citations have split paragraphs. This is a little more complicated, but we can break it down into steps:

- Use `grep()` to find the location of all citations in the opinion text; any string of one to three numbers encased in brackets (e.g., [000]). Assign the result to the `cites` object, which we will use to *index* the opinion vector.
- Create two additional vectors that contain the index of the pre- and post-citation text. This can be done fairly simply by adding and subtracting 1 from the `cites` object (since the pre- and post-citation text are necessarily immediately above and below the citation).
- Create a list object which uses the `paste()` function to combine the pre- and post-citation text for each triplet.
- Overwrite the citations in the `opinion` vector with the pre-and-post text, index the `opinion` vector with the `cites` vector.

- Remove the original pre- and post-citation text from the opinion vector (you overwrote the citation with the combined text in the previous step). You do this by negative indexing the opinion vector with the combined, `c()`, `pre` and `post` vectors (i.e., `c(pre, post)`).

```r
# Find the citations in the list, following
# the format of [0], [00], or [000],
# where 0 can be any number.
cites <- grep("^\\[[0-9]{1,3}\\]$", opinion)
pre <- cites - 1
post <- cites + 1

# Combine the pre- and post-citation text.
# Omit the citation itself (cites object).
combined <- list()
for (i in seq_along(cites)){
  combined[[i]] <- paste(opinion[pre[i]], opinion[post[i]])
}
combined <- unlist(combined)

# Replace the citations in the original with the combined paragraphs,
# then remove the original pre- and post-citation text elements.
opinion[cites] <- combined
opinion <- opinion[-c(pre, post)]

# Print the first 18 elements of the opinion character vector:
head(opinion, 18)
```

```
##  [1] "U.S. Supreme Court Miranda v. Arizona, 384 U.S. 436 (1966)"
##  [2] "Miranda v. Arizona"
##  [3] "No. 759"
##  [4] "Argued February 28-March 1, 1966"
##  [5] "Decided June 13, 1966*"
##  [6] "384 U.S. 436"
##  [7] "CERTIORARI TO THE SUPREME COURT OF ARIZONA"
##  [8] "MR. CHIEF JUSTICE WARREN delivered the opinion of the Court."
##  [9] "The cases before us raise questions which go to the roots of our concepts of American criminal
## [10] "This case has been the subject of judicial interpretation and spirited legal debate since it wa
## [11] "have speculated on its range and desirability. [Footnote 3] We granted certiorari in these case
## [12] "We start here, as we did in Escobedo, with the premise that our holding is not an innovation in
## [13] "Over 70 years ago, our predecessors on this Court eloquently stated:"
## [14] "\"The maxim nemo tenetur seipsum accusare had its origin in a protest against the inquisitorial
## [15] "Brown v. Walker, 161 U. S. 591, 596-597 (1896). In stating the obligation of the judiciary to a
## [16] "\". . . our contemplation cannot be only of what has been, but of what may be. Under any other
## [17] "This was the spirit in which we delineated, in meaningful language, the manner in which the con
## [18] "Our holding will be spelled out with some specificity in the pages which follow, but, briefly s
```

However, we're not quite there yet. There are still some broken up paragraphs (elements in the `opinion` vector that start with a lower case letter), we can combine these using a similar technique as in the previous code chunk. We can finish off cleaning these data with the following code chunk:

```r
leading_lower <- grep("^[a-z]", opinion)
pre <- leading_lower - 1
```

```r
combined <- list()
for (i in seq_along(leading_lower)){
  combined[[i]] <- paste(opinion[pre[i]], opinion[leading_lower[i]])
}
combined <- unlist(combined)

opinion[pre] <- combined
opinion <- opinion[-leading_lower]

head(opinion, 18)
```

```
##  [1] "U.S. Supreme Court Miranda v. Arizona, 384 U.S. 436 (1966)"
##  [2] "Miranda v. Arizona"
##  [3] "No. 759"
##  [4] "Argued February 28-March 1, 1966"
##  [5] "Decided June 13, 1966*"
##  [6] "384 U.S. 436"
##  [7] "CERTIORARI TO THE SUPREME COURT OF ARIZONA"
##  [8] "MR. CHIEF JUSTICE WARREN delivered the opinion of the Court."
##  [9] "The cases before us raise questions which go to the roots of our concepts of American criminal
## [10] "This case has been the subject of judicial interpretation and spirited legal debate since it wa
## [11] "We start here, as we did in Escobedo, with the premise that our holding is not an innovation i
## [12] "Over 70 years ago, our predecessors on this Court eloquently stated:"
## [13] "\"The maxim nemo tenetur seipsum accusare had its origin in a protest against the inquisitoria
## [14] "Brown v. Walker, 161 U. S. 591, 596-597 (1896). In stating the obligation of the judiciary to
## [15] "\". . . our contemplation cannot be only of what has been, but of what may be. Under any other
## [16] "This was the spirit in which we delineated, in meaningful language, the manner in which the co
## [17] "Our holding will be spelled out with some specificity in the pages which follow, but, briefly
## [18] "The constitutional issue we decide in each of these cases is the admissibility of statements o
```

3 Managing Text Data

3.1 Preprocessing your data

Now we have a string of text data, we can finally start preprocessing our text data. Text data is inherently much more complex than numeric data - hence the long-lasting distinction between qualitative and quantitative research. In order to take a quantitative approach to qualitative data, preparation and pre-processing is incredibly important. Improperly prepared and pre-processed data can severely limit your ability to produce meaningful results and insights.

In this example, I am going to showcase a few steps that are often taken when preprocessing text data using the **Quanteda** package. Quanteda is a fantastic R package, serving as a suite of tools that can support natural language processing projects.

A note on encoding:

Computers are fundamentally mathematical machines. To even present characters on a screen, text and characters need to be *encoded*. There are many different encoding schemes (ASCII, ISO/IEC 8859, UTF-8, UTF-16, etc.). I would strongly advise that you keep track of your own data's encoding to ensure that no special characters are 'lost in translation'. This is especially important when you are working with multilingual data, as different languages may require different encodings.

Preprocessing techniques:

- Tokenization: Process of dividing text into smaller units called 'tokens' (typically words, but can be sub-words, numbers, or other unique character strings).
- Normalization:
  - To lower: Convert all characters in your data to lowercase; `tokens_tolower()`.
  - Stemming: Convert all words in your data into their stem form (e.g., Policing, Policed, Police -> Polic); `tokens_wordstem()`.
  - Lemmatization: Convert all words in your data into their base 'lemma' form (e.g., Policing, Policed -> Police); `tokens_replace()`.
- Pruning:
  - Stopword removal: Remove frequently used words that add little or no meaning to a sentence (the, is, and, etc.) ; `tokens_remote()` and `stopwords("english")`.
  - Relative pruning: Remove highly frequent or infrequent words in your data (as with stopwords, they tend to add little meaning); `dfm_trim()`. You perform this step when creating the document-term matrix, see the next section.

```r
library(quanteda)
```

```
## Warning: package 'quanteda' was built under R version 4.3.3

## Warning in .recacheSubclasses(def@className, def, env): undefined subclass
## "ndiMatrix" of class "replValueSp"; definition not updated

## Package version: 4.2.0
## Unicode version: 15.1
## ICU version: 74.1

## Parallel computing: 16 of 16 threads used.

## See https://quanteda.io for tutorials and examples.
```

```r
library(lexicon)
```

```
## Warning: package 'lexicon' was built under R version 4.3.3
```

```r
# Tokenization
tokens <- tokens(c(syllabus, opinion),
                 what = "word",
                 remove_punct = TRUE,          # removes punctuation
                 remove_numbers = TRUE,        # removes numbers
                 remove_url = TRUE)            # removes URLs
tokens[1]
```

```
## Tokens consisting of 1 document.
## text1 :
##  [1] "Under"      "the"        "Fifth"      "Amendment"  "any"
##  [6] "statements" "that"       "a"          "defendant"  "in"
## [11] "custody"    "makes"
## [ ... and 51 more ]
```

```r
# Lower case
tokens <- tokens_tolower(tokens)
tokens[1]
```

```
## Tokens consisting of 1 document.
## text1 :
##  [1] "under"      "the"        "fifth"      "amendment"  "any"
##  [6] "statements" "that"       "a"          "defendant"  "in"
## [11] "custody"    "makes"
## [ ... and 51 more ]
```

```r
# Stopword removal
stopwords("english")[1:20]
```

```
##  [1] "i"          "me"         "my"         "myself"     "we"
##  [6] "our"        "ours"       "ourselves"  "you"        "your"
## [11] "yours"      "yourself"   "yourselves" "he"         "him"
## [16] "his"        "himself"    "she"        "her"        "hers"
```

```r
tokens <- tokens_remove(tokens, c(stopwords("english"), "also"))
tokens[1]
```

```
## Tokens consisting of 1 document.
## text1 :
##  [1] "fifth"         "amendment"     "statements"    "defendant"
##  [5] "custody"       "makes"         "interrogation" "admissible"
##  [9] "evidence"      "criminal"      "trial"         "law"
## [ ... and 19 more ]
```

```r
# Lemmatization
tokens_replace(tokens,
               pattern = lexicon::hash_lemmas$token,
               replacement = lexicon::hash_lemmas$lemma)[1]
```

```
## Tokens consisting of 1 document.
## text1 :
##  [1] "5"             "amendment"     "statement"     "defendant"
##  [5] "custody"       "make"          "interrogation" "admissible"
##  [9] "evidence"      "criminal"      "trial"         "law"
## [ ... and 19 more ]
```

```r
# Stemming
## In this example, we are going to use stemming.
## However, stemming can reduce the legibility of text.
## So, generally, I advise the use of lemmatization.
tokens <- tokens_wordstem(tokens)
tokens[1]
```

```
## Tokens consisting of 1 document.
## text1 :
##  [1] "fifth"     "amend"     "statement" "defend"    "custodi"   "make"
##  [7] "interrog"  "admiss"    "evid"      "crimin"    "trial"     "law"
## [ ... and 19 more ]
```

3.2 Numeric Representations of Textual Data

3.2.1 Document-Term Matrices

Document-term matrices (often abbreviated as dtm, tdm, or dfm) are adjacency matrices where each axis represents either a document or a word (term) in each of the documents. Each cell of the matrix represents a unique combination of word and document. The number in that cell is either a raw count of the number of times a given word appears in a given document, or a weighted count (see the next section). Put simply, it is a *very* big crosstab of words and documents.

```
# Convert tokenized data into document-term matrix
## Quanteda calls this a "document-feature matrix"
dfm <- dfm(tokens)
dfm
```

```
## Document-feature matrix of: 564 documents, 2,990 features (99.09% sparse) and 0 docvars.
##        features
## docs    fifth amend statement defend custodi make interrog admiss evid crimin
##   text1     1     1         1      2       1    1        2      1    1      1
##   text2     0     0         2      0       0    0        2      0    2      0
##   text3     0     0         0      0       0    0        0      0    0      0
##   text4     1     3         0      2       0    0        2      0    0      1
##   text5     0     0         0      0       0    0        0      0    0      0
##   text6     0     0         0      0       0    1        0      0    0      0
## [ reached max_ndoc ... 558 more documents, reached max_nfeat ... 2,980 more features ]
```

```
# Relative pruning
dfm <- dfm_trim(dfm,
                min_docfreq = 0.01,      # Prunes words appearing in lower 1%
                max_docfreq = 0.99,      # Prunes words appearing in upper 1%
                docfreq_type = "prop",
                verbose = TRUE)
```

```
## dfm_trim() changed from 2,990 features (564 documents) to 639 features (564 documents)
```

```
dfm
```

```
## Document-feature matrix of: 564 documents, 639 features (96.92% sparse) and 0 docvars.
##        features
## docs    fifth amend statement defend custodi make interrog admiss evid crimin
##   text1     1     1         1      2       1    1        2      1    1      1
##   text2     0     0         2      0       0    0        2      0    2      0
##   text3     0     0         0      0       0    0        0      0    0      0
##   text4     1     3         0      2       0    0        2      0    0      1
##   text5     0     0         0      0       0    0        0      0    0      0
##   text6     0     0         0      0       0    1        0      0    0      0
## [ reached max_ndoc ... 558 more documents, reached max_nfeat ... 629 more features ]
```

3.2.2 Term Frequency-Inverse Document Frequency

3.2.2.1 Term Frequency (TF)

Term frequency is a metric that estimates the relative frequency of a given word in a given document. The numerator of the equation represents the number of times a given term, t, appears in a given document, d. The denominator of the equation represents the total number of terms in a given document, d.

$$tf(t,d) = \frac{f(t,d)}{N(d)}$$

3.2.2.2 Inverse Document Frequency (IDF) A metric that descibes the amount of information information provided by a given word based on its frequency (or rarity). The numerator, N, is the total number documents in the corpus (in our case, this is the number of paragraphs). The denominator, $|\{d \in D : t \in d\}|$, is the number of documents where each given term appears.

$$idf(t,D) = log\frac{|D|}{|\{d \in D : t \in d\}|}$$

3.2.2.3 TF-IDF The term frequency-inverse document frequency metric is a combination of these two metrics. It is designed to scale the raw counts of each combination of word and document to account for the relative importance of that word and document.

$$tf - idf(t,d) = tf(t,d) * idf(t,D)$$

```
# Term frequency-inverse document frequency
dfm_tfidf(dfm)
```

```
## Document-feature matrix of: 564 documents, 639 features (96.92% sparse) and 0 docvars.
##        features
## docs       fifth     amend statement   defend  custodi     make interrog
##   text1 1.052309 0.9878511 0.8704655 1.990808 1.003091 1.079181 1.191886
##   text2 0        0         1.7409310 0        0        0        1.191886
##   text3 0        0         0         0        0        0        0
##   text4 1.052309 2.9635533 0         1.990808 0        0        1.191886
##   text5 0        0         0         0        0        0        0
##   text6 0        0         0         0        0        1.079181 0
##        features
## docs      admiss      evid    crimin
##   text1 1.098067 0.9519386 0.9450991
##   text2 0        1.9038771 0
##   text3 0        0         0
##   text4 0        0         0.9450991
##   text5 0        0         0
##   text6 0        0         0
## [ reached max_ndoc ... 558 more documents, reached max_nfeat ... 629 more features ]
```

3.2.3 Word Co-occurrence Matrix

Word co-occurrence matrices are similar to document-term matrices (both are broadly defined as adjacency matrices), but both rows and columns represent words in the corpus. Each cell of the matrix is representing the number of co-occurrences of two words within a given window. A moving window of variable length (typically 5 words wide centered on a single word) 'slides' along each document in the corpus, and populates the matrix by counting the instances where words appear within the same window. This moving window can be weighted to priortize words that are more proximal to the central 'context' word in the moving window. The data produced by this approach is commonly used in the training of neural network-based natural language processing analyses. This includes word2vec and 'Global Vectors' (GloVe), a pair of very similar word embedding models.

```r
fcm(tokens,
    context = "window",
    count = "weighted",
    weights = 1 / (1:5),
    tri = TRUE)
```

```
## Feature co-occurrence matrix of: 2,990 by 2,990 features.
##            features
## features    fifth    amend statement    defend  custodi     make  interrog
##   fifth         0 67.03333  1.166667 0.5833333 0.700000  0.60000  2.316667
##   amend         0  1.50000  2.150000 0.7000000 1.116667  0.25000  3.183333
##   statement     0  0       18.366667 11.1500000 5.283333 29.61667  7.333333
##   defend        0  0        0        1.8333333 7.283333  0.75000  4.850000
##   custodi       0  0        0        0         0.400000  1.50000 29.866667
##   make          0  0        0        0         0         2.40000  2.850000
##   interrog      0  0        0        0         0         0        7.766667
##   admiss        0  0        0        0         0         0        0
##   evid          0  0        0        0         0         0        0
##   crimin        0  0        0        0         0         0        0
##            features
## features       admiss      evid   crimin
##   fifth      1.083333 0.5000000 0.200000
##   amend      1.833333 1.7333333 1.150000
##   statement 11.266667 2.8666667 0
##   defend     1.950000 1.4000000 5.066667
##   custodi    1.316667 0.4500000 0.200000
##   make       0.500000 1.0000000 0.250000
##   interrog   2.250000 4.0666667 5.583333
##   admiss     0         6.1000000 1.333333
##   evid       0         0.6666667 1.000000
##   crimin     0         0         3.900000
## [ reached max_feat ... 2,980 more features, reached max_nfeat ... 2,980 more features ]
```

3.2.4 Embeddings

Word embeddings are a little harder to explain. Intuitively, they are sets of numbers that represent the *meaning* of a given word based on the patterns it establishes in relation to other words. For example, “sea” and “lake” will co-occur with the words “swim” and “sail”.

With sufficient input data, a simple shallow neural network can learn to ‘recognize’ these patterns, and generate ‘embeddings’ that represent the words as numbers. You can perform mathematical functions on the resulting ‘embeddings’, and represent sementic rules and relationships mathematically. Embeddings provide the foundation for modern computational linguistics.

```r
library(text2vec)

cm <- fcm(tokens,
          context = "window",
          count = "weighted",
          weights = 1 / (1:5),
          tri = TRUE)

glove <- GlobalVectors$new(rank = 50,
```

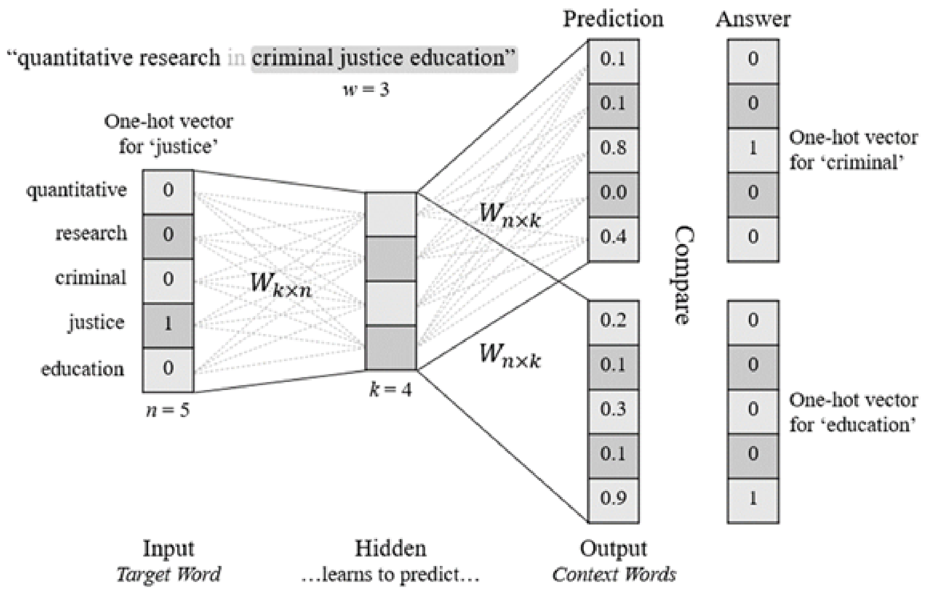


Figure 3: Illustration of the word2vec shallow neural network (Mikolov, 2013). The model learns to predict context words based on target words based on word co-occurrence input.

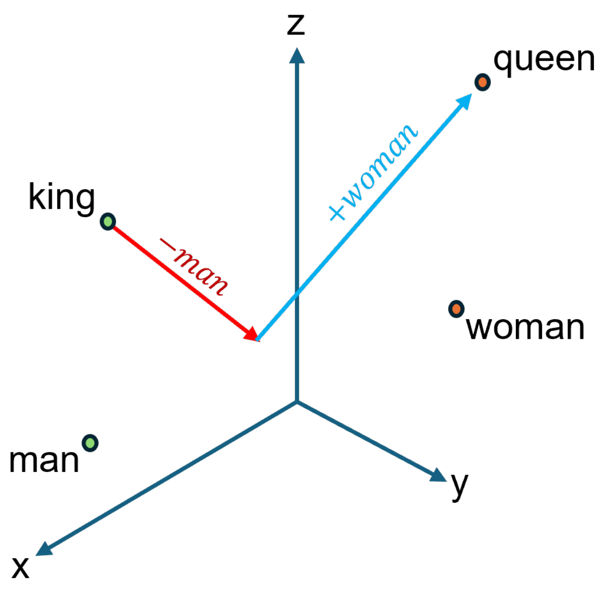


Figure 4: Illustration of the vector mathematics that allow embeddings to represent relationships between words.

```
                          x_max = 10)

wv <- glove$fit_transform(cm,
                          n_iter = 10,
                          convergence_tol = 0.01,
                          n_threads = 8)
```

```
## INFO  [21:33:25.936] epoch 1, loss 0.1398
## INFO  [21:33:25.970] epoch 2, loss 0.0786
## INFO  [21:33:25.997] epoch 3, loss 0.0608
## INFO  [21:33:26.008] epoch 4, loss 0.0503
## INFO  [21:33:26.021] epoch 5, loss 0.0431
## INFO  [21:33:26.035] epoch 6, loss 0.0378
## INFO  [21:33:26.051] epoch 7, loss 0.0337
## INFO  [21:33:26.065] epoch 8, loss 0.0303
## INFO  [21:33:26.077] epoch 9, loss 0.0275
## INFO  [21:33:26.093] epoch 10, loss 0.0252
```

```
miranda <- wv["miranda", , drop = FALSE]

sim2(x = wv,
     y = miranda,
     method = "cosine",
     norm = "l2")[, 1] |>
  sort(decreasing = TRUE) |>
  head(15)
```

```
##    miranda     daytim        6-7       half queensburi     unanim        rel
##  1.0000000  0.4839325  0.4755129  0.4145119  0.4110602  0.3934277  0.3918658
##          j        aug          s    virtual     exceed   restrain       ohio
##  0.3834571  0.3821556  0.3776207  0.3769195  0.3761019  0.3691768  0.3651781
##   unworthi
##  0.3623425
```

4 Bibliography

Benoit K, Watanabe K, Wang H, Nulty P, Obeng A, Müller S, Matsuo A (2018). “quanteda: An R package for the quantitative analysis of textual data.” Journal of Open Source Software, 3(30), 774.

Grolemund, G., & Wickham, H. (2017). R for Data Science. O’Reilly Media.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. Advances in neural information processing systems, 26.