Introduction to Web Scraping

STA 325, Supplemental Material

Rebecca C. Steorts

Agenda

Load libraries

```r
library(magrittr)
library(rvest)
```

```
## Loading required package: xml2
```

```
## Warning: package 'xml2' was built under R version 3.4.3
```

```r
library(stringr)
```

```
## Warning: package 'stringr' was built under R version 3.4
```

```r
library(xml2)
```

Web scraping

Web scraping is a technique for converting data present in an unstructured format (HTML tags) over the web to a structured format (such as a table or file) that can be easily accessed, used, and analyzed.

Why web scrape

1. Companies often `scrape` data to make recommendations to its users.
2. Companies/Researchers often `scrape` picture data and train it (as a match or non-match) to make predictions.

Example: When you click on a Company X picture and tag someone, you are giving the data a label. Company X uses this to then help identify that new pictures that appear on the webpage may be you by comparing to all the pictures in their training dataset.

3. Social media webpages `scrape` data to perform tasks such as sentiment analysis or opinion mining. (Think elections.)

How can we scrape data?

1. Human copy and paste

- This involves humans themselves analyzing and copying the data to local storage.

2. Text pattern matching

- One extracts information from the web using regular expression matching

3. API Interface

- Many webpages have public or private APIs which can be called to retrieve data in a standard format

4. DOM Parsing

- By using the web browsers, programs can retrieve the dynamic content generated by client-side scripts. It is also possible to parse web pages into a DOM tree, based on which programs

DOM Parsing

We'll be looking at scraping via DOM parsing.

In our toolbox, we'll need to understand/use the following:

1. piping
2. rvest package
3. Download the selector Gadget (`https://selectorgadget.com/`) as an extension to your browser

Piping

There is a helpful R trick (used in many other languages) known as piping that we will use. Piping is available through R in the `magrittr` package and using the `{%>%}` command.

How does one use the pipe command? The pipe operator is used to insert an argument into a function. The pipe operator takes the left-hand side (LHS) of the pipe and uses it as the first argument of the function on the right-hand side (RHS) of the pipe.

Why are we learning this? It's required in many newer packages in R, such as `rvest`

Examples of Piping

We give two simple examples

```r
# Example One
1:50 %>% mean
```

```
## [1] 25.5
```

```r
mean(1:50)
```

```
## [1] 25.5
```

Examples of Piping

```r
# Example Two
years <- factor(2008:2012)
# nesting
as.numeric(as.character(years))
```

```
## [1] 2008 2009 2010 2011 2012
```

```r
# piping
years %>% as.character %>% as.numeric
```

```
## [1] 2008 2009 2010 2011 2012
```

Selector Gadget

Selector Gadget is a Google Chrome extension that will allow us to examine the features of the HTML text so that we can input these features into rvest.

rvest

rvest helps you scrape information from web pages.

It is designed to work with `magrittr` to make it easy to express common web scraping tasks.

Functions of rvest

- Create an html document from a url, a file on disk or a string containing html with `read_html()`.
- Select parts of a document using css selectors: `html_nodes(doc, "table td")` (or if you've a glutton for punishment, use xpath selectors with `html_nodes(doc, xpath = "//table//td")`). If you haven't heard of selectorgadget, make sure to read `vignette("selectorgadget")` to learn about it.
- Extract components with `html_tag()` (the name of the tag), `html_text()` (all text inside the tag), `html_attr()` (contents of a single attribute) and `html_attrs()` (all attributes).

Functions of rvest

- You can also use rvest with XML files: parse with `xml()`, then extract components using `xml_node()`, `xml_attr()`, `xml_attrs()`, `xml_text()` and `xml_tag()`.
- Parse tables into data frames with `html_table()`.
- Extract, modify and submit forms with `html_form()`, `set_values()` and `submit_form()`.
- Detect and repair encoding problems with `guess_encoding()` and `repair_encoding()`.
- Navigate around a website as if you're in a browser with `html_session()`, `jump_to()`, `follow_link()`, `back()`, `forward()`, `submit_form()` and so on.

A webpage about lies

Our goal for this exercise will be to do the following:

We will scrape an New York Times article that points out “lies” or inconsistencies with President Trump.

The article we’re looking at that was written by the NYT consists of “lies”, where we’ll break these down into four parts to make the exercise easier:

1. Dates of the “lies”
2. The “lie” itself
3. An explaination (from the writers of the NYT) of this is a lie
4. A URL in the text supporting their explaination. (These are embedded in text).

Scraping a webpage

- We're going to start with a simple example where we scrape a webpage.
- This example is taken from the following tutorial, where you can read more about this later and go through this in more detail (`https://towardsdatascience.com/web-scraping-tutorial-in-r-5e71fd107f32`).

Loading the webpage

The first thing we will do is load in the webpage using the `read_html()` function

This returns an XML document that contains informations about the webpage

Loading the webpage

```r
webpage <- read_html("https://www.nytimes.com/interactive/2017/06/23/opinion/trumps-lies.html")
webpage
```

```
## {xml_document}
## <html lang="en" class="no-js page-interactive section-opinion page-theme-standard tone-opinion page-int
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset= ...
## [2] <body>\n<style>\n.lt-ie10 .messenger.suggestions {\n  display: block ...
```

Collect all the records

Every record has the following structure in the HTML code:

```
<span class="short-desc"><strong> DATE </strong> LIE
<span class="short-truth"><a href="URL"> EXPLANATION
</a></span></span>
```

You can find this by either looking at the HTML source file or using the Gadget inspector.

I found it easier/faster to just look at the source.

Collect all the records

- To collect all the “lies”“, we need to identify all the tags that belong to `class="short-desc"`
- The function that will help us to do so is `html_nodes()`
- This function requires the following as inputs:

1. the XML document we want to analyze
2. the nodes we want to select

Using the selector gadget, we can determine that we can find all lies using the selector “.short-desc”

Collect all the records

```r
results <- webpage %>% html_nodes(".short-desc")
results
```

```
## {xml_nodeset (180)}
##  [1] <span class="short-desc"><strong>Jan. 21 </strong>"I wasn't a fan o ...
##  [2] <span class="short-desc"><strong>Jan. 21 </strong>"A reporter for T ...
##  [3] <span class="short-desc"><strong>Jan. 23 </strong>"Between 3 millio ...
##  [4] <span class="short-desc"><strong>Jan. 25 </strong>"Now, the audienc ...
##  [5] <span class="short-desc"><strong>Jan. 25 </strong>"Take a look at t ...
##  [6] <span class="short-desc"><strong>Jan. 25 </strong>"You had millions ...
##  [7] <span class="short-desc"><strong>Jan. 25 </strong>"So, look, when P ...
##  [8] <span class="short-desc"><strong>Jan. 26 </strong>"We've taken in t ...
##  [9] <span class="short-desc"><strong>Jan. 26 </strong>"I cut off hundre ...
## [10] <span class="short-desc"><strong>Jan. 28 </strong>"The coverage abo ...
## [11] <span class="short-desc"><strong>Jan. 29 </strong>"The Cuban-Americ ...
## [12] <span class="short-desc"><strong>Jan. 30 </strong>"Only 109 people  ...
## [13] <span class="short-desc"><strong>Feb. 3 </strong>"Professional anar ...
## [14] <span class="short-desc"><strong>Feb. 4 </strong>"After being force ...
## [15] <span class="short-desc"><strong>Feb. 5 </strong>"We had 109 people ...
## [16] <span class="short-desc"><strong>Feb. 6 </strong>"I have already sa ...
## [17] <span class="short-desc"><strong>Feb. 6 </strong>"It's gotten to a  ...
## [18] <span class="short-desc"><strong>Feb. 6 </strong>"The failing @nyti ...
## [19] <span class="short-desc"><strong>Feb. 6 </strong>"And the previous  ...
## [20] <span class="short-desc"><strong>Feb. 7 </strong>"And yet the murde ...
## ...
```

```r
length(results)
```

```
## [1] 180
```

This returns a list with 180 XML nodes that contain the information for each of the 180 lies in the web page.

Extracting the first "lie"

Let's start simple and focus on extracting all the necessary details from the first lie.

Recall that the syntax for a single record is:

```
<span class="short-desc"><strong> DATE </strong> LIE <span
```

Date is embedded within the `<strong>` tag.

To select it, we can use the `html_nodes()` function using the selector "strong"

Extracting the first date

```r
first_result <- results[1]
first_result
```

```
## {xml_nodeset (1)}
## [1] <span class="short-desc"><strong>Jan. 21 </strong>"
```

```r
first_result %>% html_nodes("strong")
```

```
## {xml_nodeset (1)}
## [1] <strong>Jan. 21 </strong>
```

Extracting the first lie

- Now to extract the only the text, we use the `html_text()` function
- We will trim leading and trailing spaces using `trim`
- Finally, we will use the `stringr` package to add the year to the date that we extracted

Extracting the first date

```
first_result <- results[1]
first_result
```

```
## {xml_nodeset (1)}
## [1] <span class="short-desc"><strong>Jan. 21 </strong>"I
```

```
date <- first_result %>% html_nodes("strong") %>% html_text
str_c(date, ', 2017')
```

```
## [1] "Jan. 21, 2017"
```

Selecting/Extracting the first lie

- To select the lie, we need to make use of the `xml_contents()` function that is part of the `xml2` package
- The function returns a list with the nodes that are part of `first_result`

Selecting/Extracting the first lie

```
xml_contents(first_result)
```

```
## {xml_nodeset (3)}
## [1] <strong>Jan. 21 </strong>
## [2] "I wasn't a fan of Iraq. I didn't want to go into Ir
## [3] <span class="short-truth"><a href="https://www.buzzf
```

Selecting/Extracting the first lie

We are intersted in the “lie”, which is the text of the second node.

```
xml_contents(first_result)[2]
```

```
## {xml_nodeset (1)}
## [1] "I wasn't a fan of Iraq. I didn't want to go into Iraq."
```

```
xml_contents(first_result)[2] %>% html_text(trim = TRUE)
```

```
## [1] ""I wasn't a fan of Iraq. I didn't want to go into Iraq.""
```

Selecting/Extracting the first lie

Now, we just need to remove the extra pair of quotations using the `str_sub()` function

```
lie <- xml_contents(first_result)[2] %>% html_text(trim = TRUE)
length(lie)
```

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

```
str_sub(lie, 2, -2)
```

```
## [1] "I wasn't a fan of Iraq. I didn't want to go into Iraq."
```

Extracting the explanation

How would you extract the explation? Try this on your own.

Extracting the explanation (Solution)

We need to select the text within the `<span>` tag that belongs to `class=".short-truth"`

Remark: This will extract the text together with the opening and closing parentheses, but we can easily remove them.

Extracting the explanation (Solution)

```r
explanation <- first_result %>% html_node(".short-truth") %>% html_text
explanation
```

```
## [1] "(He was for an invasion before he was against it.)"
```

```r
str_sub(explanation, 2, -2) # remove parans
```

```
## [1] "He was for an invasion before he was against it."
```

Extracting the URL

How would you extract the URL? Try this on your own.

Extracting the URL (Solution)

Finally, to get the URL, notice that this is an attribute within the `<a> tag`

- We simply select this node with the `html_nodes()` function and then select the href attribute with the `html_attr()` function.

Extracting the URL (Solution)

```r
url <- first_result %>% html_node("a") %>% html_attr("href")
url
```

```
## [1] "https://www.buzzfeed.com/andrewkaczynski/in-2002-donald-trump-s
```

Summary

Above, we have extracted the four parts of the first record.

Have them perhaps for homework create the entire dataframe and output things to a .csv file. (Write this up). Extra credit: Use the purr function to replace the for loop.