Gov 2002 - Section 5 - Theory & Practice of Web Scraping

Connor Jerzak

October 12, 2016

Harvard University, Department of Government

Theory of Web Scraping

What is the Internet?

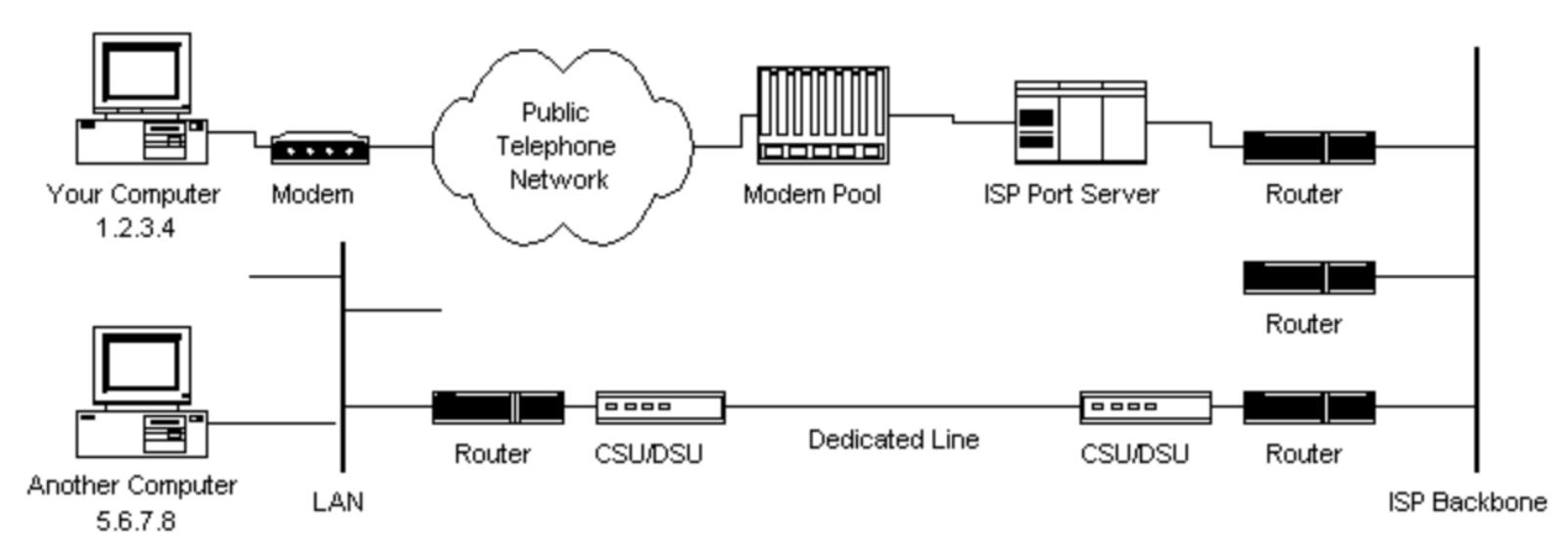
- **History**. The Internet has origins in US Department of Defense projects such as the ARPANET.
  - The ARPANET project aimed at sharing digital information across geographically separated computers.
  - These projects were funded in the 1960s by the Department of Defense. Paul Baran (and others) were trying to construct a communications system that would survive a nuclear attack.

    Their answer was that the communications system should not be centralized, but instead distributed, with many interconnected nodes all able to share information with one another in a swift manner.

Components of the Internet

- **Description**. The Internet is a network of interconnected computers, which transmit data to one another through a type of packet switching called the Internet protocol (IP).
  - *Packet switching*. Packet switching is a method for transferring data across a network in which the data is broken up into smaller blocks (called "packets"), which are then transmitted over some medium. The packet size is designed to minimize the time it takes for data to pass across a network, and to increase the resilience of the system to noise.
  - *Router*. A router forwards the data packets along the network.
  - *Modem*. A modem is a network hardware device that adjusts the out-going data signals to encode digital information for transmission and decodes incoming signals to extract the transmitted information.

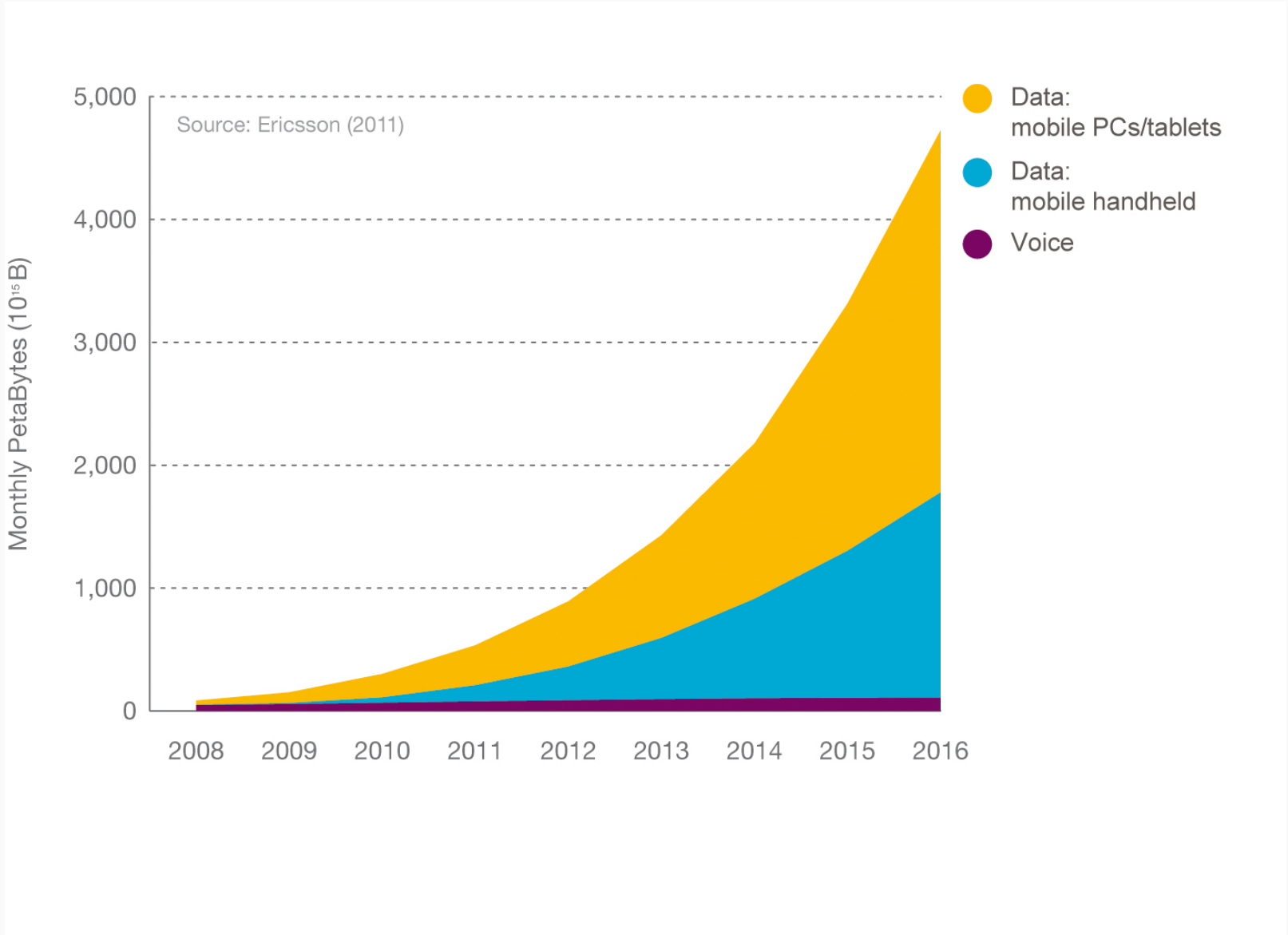
The Internet



Data and the Internet

- The Internet is now the medium of choice for virtually every kind of communication.
  - Every minute, there are an estimated 4,166,667 Facebook likes, 600,000 Tinder swipes, 347,222 Tweets.
  - "Construction of the Empire State Building: 7 million human-hours. The Panama Canal: 20 million human-hours. Estimated number of human-hours spent playing solitaire around the world in one year: 9 billion." – Manuel Blum
  - The amount of data on the web is growing approximately at an exponential rate.
- **Implications**. A huge wealth of information is freely available on the Internet. Web scraping is the technology involved with converting this information into useable knowledge.

Exponential growth



What is a webpage?

- *URL*. A webpage is defined by a Uniform Resource Locator (URL). This specifies where to find a given file. Web page URLs were previously static, but now are often dynamic, and are updated continually in interactions with the user.
- *Content*. There are various languages that tell the web browser how to display the files stored in the web host's server for the user. One of the more common languages is HyperText Markup Language (HTML), and each individual markup code in HTML is called a tag or element.
    - All webpages are actually text files, with a series of pointers telling the browser how to present the files saved on the webpage servers.
    - We'll talk more about this later.

URL example


http://www.computerhope.com/jargon/u/url.htm
Protocol
Subdomain
Domain and domain suffix
Directories
Web page

Practice of Web Scraping

Web Scraping, Introduced

Web scraping involves two related tasks.

- *Download phase*. In the download phase, we need to find a way to download a webpage. This is often very easy, but can be more difficult if the webpage is behind a password wall.
- *Parsing phase*. In the parsing phase, we need to find a way to extract the specific information we need from the downloaded webpage. To perform this task, we often use regular expressions. If the website is rendered using JavaScript, our task is made more difficult.

Download Phase

Download Phase - Simple Cases

Methods

- **In R**.

```
my_page = readLines('http://www.harvard.edu/')
print(my_page)
```

- **In Python**.

```
import urllib2
my_url = 'http://www.harvard.edu/'
response = urllib2.urlopen(my_url)
webContent = response.read()
print(webContent[0:300])
```

Download Phase - Recursive Crawling

- **Wget**. Wget is a terminal-based application. It will allow us to recursively download an entire website.
  - *Terminal*. The terminal is a special interface that allows you to communicate with your computer.
  - *Recursive crawling*. Wget can easily be programmed to download a webpage, to follow all links on that webpage, and then to download all the linked webpages, and so on. This allows one to download a non-trivial fraction of the web with little overhead.

Download Phase - Principles of Crawling

- When using tools such as Wget, it is important to keep a few things in mind.
  - *Download rate*. If you retrieve files as fast as possible from a webpage, you may get blacklisted. This is because you will be using the webpage's resources (and things may slow down for other users). You may also get blacklisted if you download things at the maximum possible bandwith rate.
  - *Wget code*. In terminal, type:

```
wget --wait=20 --limit-rate=20K -r -p -U Mozilla
 http://www.harvard.edu/

where -r activates the recursive retrieval mode
and -p causes Wget to download all the files that
are needed to properly display a given HTML page.
See https://www.gnu.org/software/wget/manual/html_node/Recursi
for more details
```

Download Phase - Getting behind password walls

- **With Wget**:
  - Download your browser's cookie file as a .txt after you have logged into the desired webpage using your name and password. For Mozilla, use: `https://addons.mozilla.org/en-US/firefox/addon/cookie-exporter/`.
  - Save the cookie .txt file in the desired directory.
  - Set your terminal working directory to the place where your .txt file is stored.
  - If you're using UNIX, type the following into your terminal (provided that your .txt is named `my_cookies.txt`):

```
wget --user-agent=agent-
-erobots=off --wait 1 --load-cookies
my_cookies.txt -p
http://hollis.harvard.edu/primo_library/libweb/action/myAccountMenu.do?vid=HVD&filler=&ticke
-O savedoutputfile.html -nv
```

  - Note that this solution may violate your user agreement, which often will ban the use of crawlers such as Wget.

Parsing Phase

Parsing, Introduced

- Web scraping allows us to download a large volume of data. However, we need to extract meaningful information from this data.
- Fortunately, many webpages are structured in way that allows us to extract specific elements of the webpage. Specifically, websites are written using HTML, which provides a consistent structure you can use to download your target information.

Principles of HTML

- The browser translates the webpage code into its visible form by reading the HTML tags.
- Angled brackets separate HTML code from normal text. The tags don't appear, but their effects do:

```
<b>These words will be bold</b>, and these will not.
```

- **Key point**: Raw HTML can be examined to determine how the website is organized, and what kind of data are stored therein.

Examples of HTML

- Example 1.

```
<h1>Computing Confidence Intervals for Standard Regression
 Coefficients < /h1> <i> by <b>
 Jeff A. Jones</b> and <b>
Niels G. Waller </b> </i>
 <br> Psychological Methods <br> December 2013 <br>
DOI: 10.1037/a0033269
```

- Example 2.

```
<article>
<title>Computing Confidence Intervals
for Standard Regression Coefficients</title>
<author>Jeff A. Jones</author>
<author>Niels G. Waller</author>
<journal>Psychological Methods</journal>
<year>2013</year>
<doi>DOI: 10.1037/a0033269</doi>
</article>
```

Parsing

HTLM parsing involves two interrelated tasks:

- First, we want to use the HTML structure to store only our target content.
- Second, we want to also strip away HTML tags from our target content while preserving this content.

Parsing, Store Target Content

To find and store the target content, we have a few choices:

- First, we could use regular expressions to extract what we need. We use regular expressions to extract text related to specific patterns. In R, we could use `grep, sub, regexp`, and so on. These commands allow us to find patterns in the HTML text. Your use of these functions will depend on your specific case.

```
my_string <- c("<article>
<title>Computing Confidence Intervals
for Standard Regression Coefficients</title>
<author>Jeff A. Jones</author>
<author>Niels G. Waller</author>
<journal>Psychological Methods</journal>
<year>2013</year>
<doi>DOI: 10.1037/a0033269</doi>
</article>")
my_string <- gsub("[\r\n]", "", my_string)#removes newline characters ("\n")
my_string <- gsub("[\r\n]", "", my_string)#extracts info about title
gregexpr_results <- gregexpr("<title>(.*?)</title>", my_string)
substr(my_string,
       start = gregexpr_results[[1]][1],
       stop = gregexpr_results[[1]][1] +  attr(gregexpr_results[[1]], "match.length"))
```

Parsing, Store Target Content

- Second, we could also use specific pages in `R` or `Pyton` for parsing HTML. The XML package in `R` is excellent:

```
my_string <- c("<article>
<title>Computing Confidence Intervals
for Standard Regression Coefficients</title>
<author>Jeff A. Jones</author>
<author>Niels G. Waller</author>
<journal>Psychological Methods</journal>
<year>2013</year>
<doi>DOI: 10.1037/a0033269</doi>
</article>")
my_string <- gsub("[\r\n]", "", my_string)#removes newline characters ("\n")

#The next two lines extract the title.
#Note the simplicity of this approach.
library("XML")
my_string_new <- htmlParse(my_string, asText = T)
plain.text <- xpathSApply(my_string_new, "//title", xmlValue)

#This code extracts all text in between the "title" tag.
```

Parsing Phase - Dealing with Javascript

- **Problems**. Web hosts know that their data is valuable in monetary terms, and therefore may make it difficult for you to extract what you need. They do this by rendering their webpage in JavaScript, which loads the target page with a delay, making it hard to extract the data in a structured form. Whereas HTML is highly structured, and easily manipulable, JavaScript is much more difficult to parse.
- **Solutions**:
  - *In R*. Use Rvest with PhantomJS.
  - *In Python*. Use Scrapy with Splash. Selenium is another good option, since it automates your interactions with your browser.
- Note that government websites largely do not use JavaScript rendering. However, it is increasingly common in other venues.

Extensions

- **Amazon Elastic Computing**. For large scraping tasks, you can rent large-scale computer resources from Amazon. There are various Amazon EC2 Instance Types, which determine the hardware of the host computers used for your job. For example, there are general purpose types, memory optimized types, accelerated computing, and so on.
- For more information on web scraping, contact the FAS Research Computing Office at `rchelp@fas.harvard.edu`.

Dos and Don'ts

- Do:
  - Do program your scraper to mimic human behavior. Otherwise, websites may identify you as a scraper and may blacklist your IP address.
  - Do store the URLs you have scraped. This avoids redundancies.
  - Do parallelize your scraping.
  - Do save everything that may be relevant for your analysis, but nothing more.
  - Check your user agreement before monetizing or publishing the data from your scraping project.
- Don't:
  - Tailor your code to one website only. Make your code as general as possible, since websites change constantly.

Acknowledgements

- These slides build from material presented by M. Jordan, Jackman, and others.