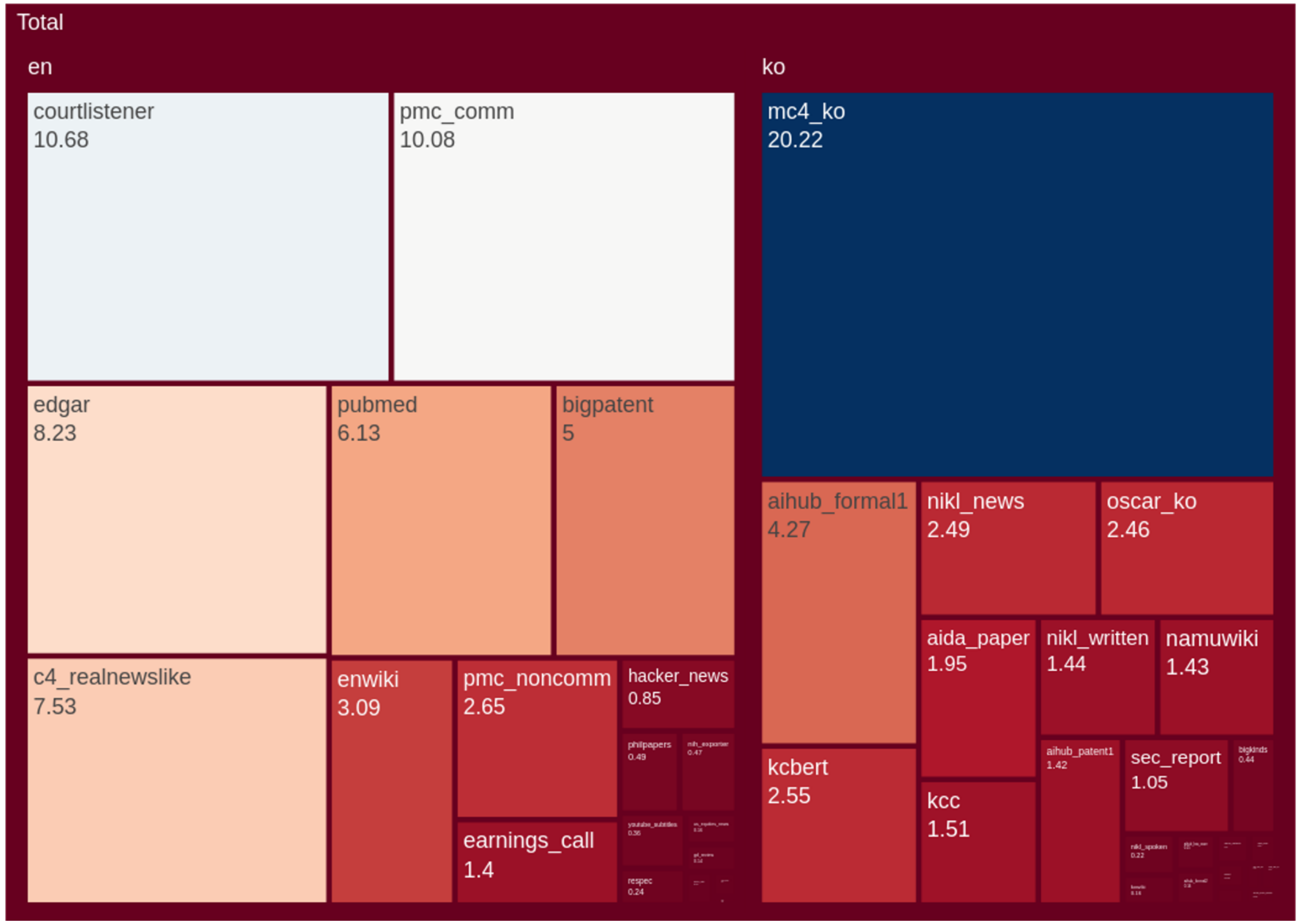
Lecture 3: Corpora, Web Scraping, and RegEx

Corpora

- A corpus is a collection of text
    - Often annotated in some way
    - Sometimes just lots of raw text
- Examples
    - Penn Treebank: 1M words of parsed Wall Street Journal
    - Canadian Hansards: 10M+ words of aligned French/English sentences
    - Yelp reviews
    - The Web / Common Crawl: billions of words of who knows what

The eKorpkit Corpus



The eKorpkit Corpus is a large, diverse, multilingual (ko/en) language modelling dataset.

English: 258.83 GiB, Korean: 190.04 GiB, Total: 448.87 GiB

The eKorpkit Corpus

Name	Language	Size	Weight	# Docs	# Sents	# Words
mc4_ko	ko	90.76 GiB	20.22%	15,618,718	665,858,888	8,007,674,274
courtlistener	en	47.92 GiB	10.68%	3,489,298	335,079,871	8,324,277,457
pmc_comm	en	45.26 GiB	10.08%	51,276,102	297,884,818	7,365,607,900
edgar	en	36.94 GiB	8.23%	213,376	177,270,203	6,053,677,897
c4_realnewslike	en	33.79 GiB	7.53%	13,813,090	155,883,681	6,040,207,703
pubmed	en	27.51 GiB	6.13%	22,498,747	190,907,356	4,281,121,705
bigpatent	en	22.46 GiB	5.00%	1,244,053	2,488,106	4,613,882,925
aihub_formal1	ko	19.16 GiB	4.27%	1,073,944	93,148,022	1,993,574,713
enwiki	en	13.85 GiB	3.09%	6,200,658	129,066,417	2,400,717,561

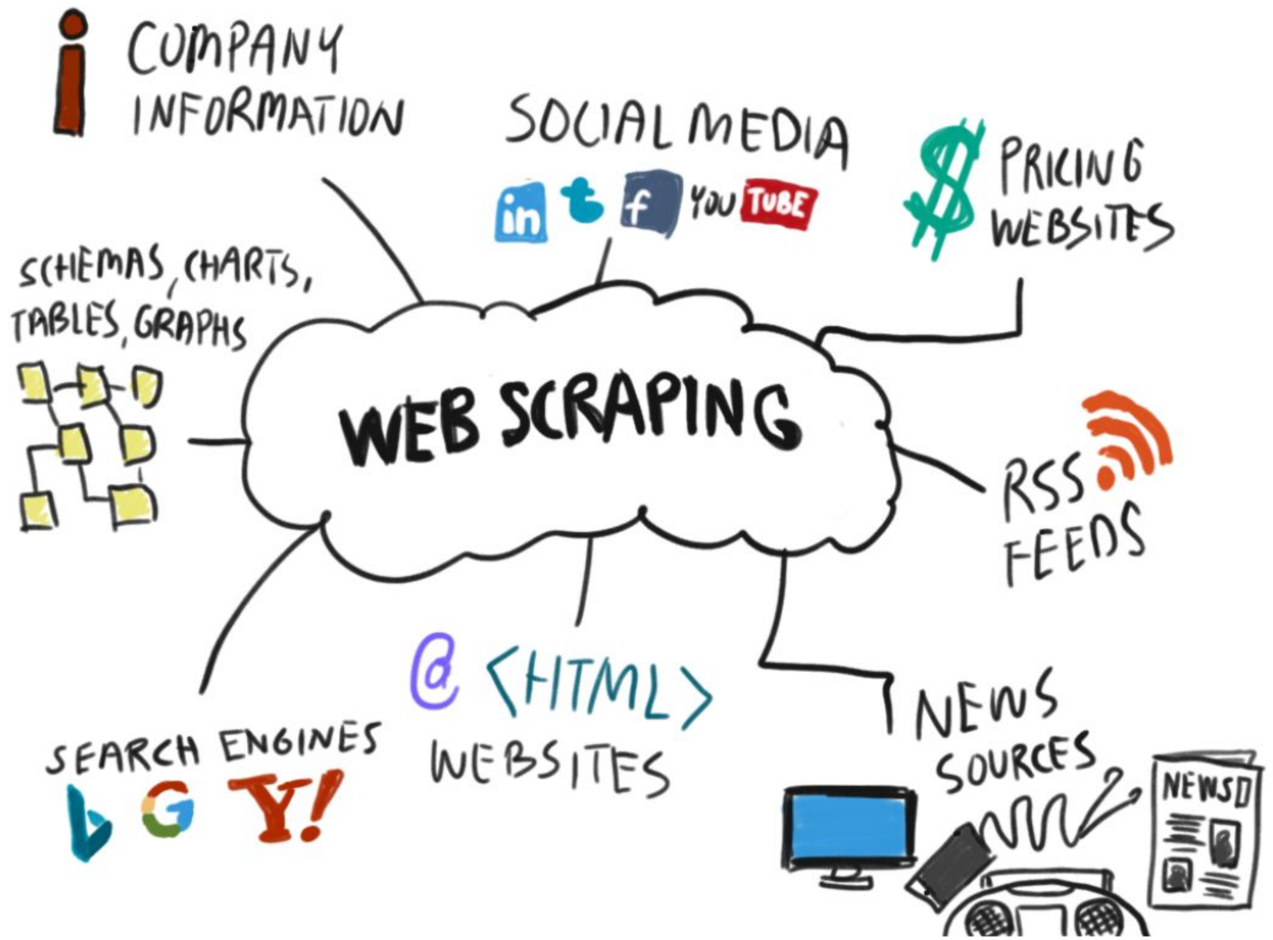
Name	Language	Size	Weight	# Docs	# Sents	# Words
pmc_noncomm	en	11.88 GiB	2.65%	14,142,294	79,748,279	1,923,415,913
kcbert	ko	11.45 GiB	2.55%	82,990,213	82,990,213	1,088,177,367
nikl_news	ko	11.19 GiB	2.49%	4,104,534	42,527,395	1,138,897,337
oscar_ko	ko	11.05 GiB	2.46%	3,673,262	61,833,262	1,122,638,494
aida_paper	ko	8.77 GiB	1.95%	481,389	38,808,105	1,025,422,060
kcc	ko	6.80 GiB	1.51%	46,529,987	46,529,987	703,222,627
nikl_written	ko	6.45 GiB	1.44%	20,128	27,231,846	679,547,033
namuwiki	ko	6.43 GiB	1.43%	571,026	67,315,244	691,537,393
aihub_patent1	ko	6.40 GiB	1.42%	155,939	29,206,198	673,134,598
earnings_call	en	6.30 GiB	1.40%	159,380	32,391,491	1,160,525,933
sec_report	ko	4.70 GiB	1.05%	817,040	32,644,657	495,245,547

Name	Language	Size	Weight	# Docs	# Sents	# Words
hacker_news	en	3.80 GiB	0.85%	818,299	41,573,998	662,524,112
philpapers	en	2.19 GiB	0.49%	31,016	139,518	365,576,851
nih_exporter	en	2.10 GiB	0.47%	1,017,230	13,540,126	326,974,102
bigkinds	ko	1.99 GiB	0.44%	871,304	7,759,115	197,746,184
youtube_subtitles	en	1.61 GiB	0.36%	150,749	16,074,289	303,286,377
respec	en	1.08 GiB	0.24%	1,119,640	7,083,257	169,590,880
nikl_spoken	ko	1002.49 MiB	0.22%	25,614	19,042,013	116,067,432
kowiki	ko	715.39 MiB	0.16%	563,959	5,671,388	70,263,451
us_equities_news	en	714.16 MiB	0.16%	220,976	1,834,664	131,179,752
aihub_law_case	ko	689.96 MiB	0.15%	77,202	1,095,140	66,686,761
aihub_formal2	ko	650.03 MiB	0.14%	95,990	1,650,141	64,523,191

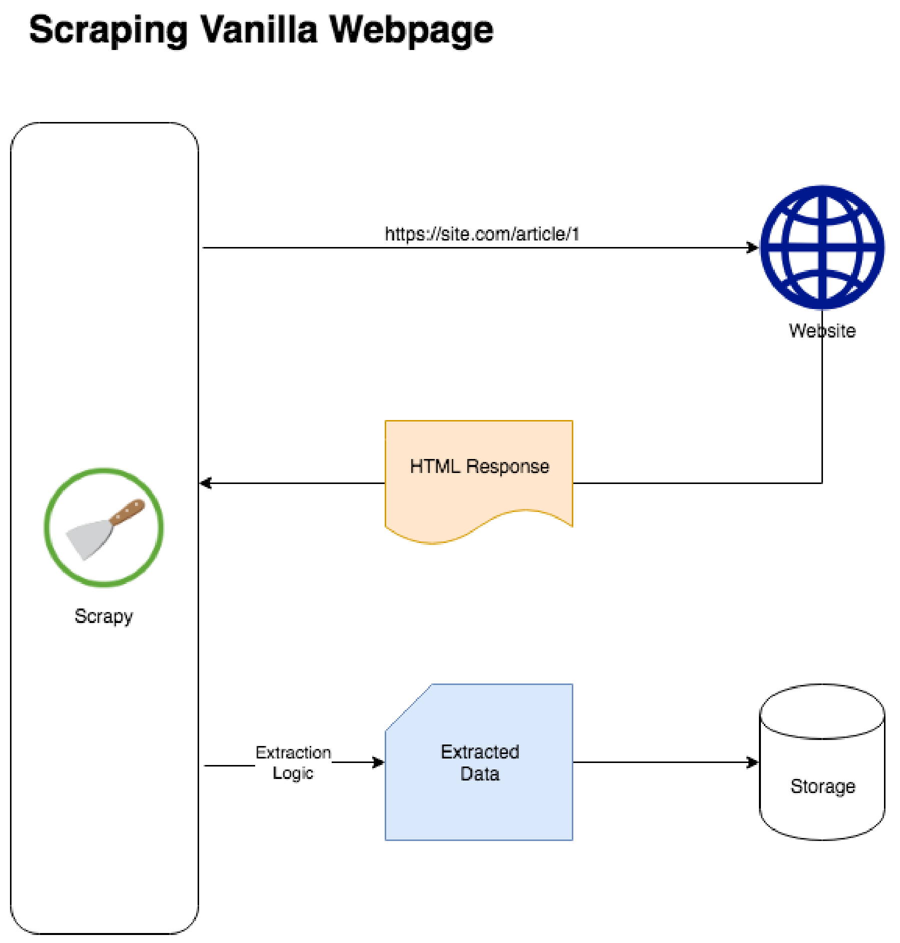
Name	Language	Size	Weight	# Docs	# Sents	# Words
gd_review	en	642.76 MiB	0.14%	1,929,910	6,733,680	112,977,678
aihub_patent2	ko	457.18 MiB	0.10%	147,674	1,879,909	46,045,036
enron_mail	en	428.36 MiB	0.09%	247,586	7,908,959	65,258,456
aihub_paper	ko	370.11 MiB	0.08%	98,344	1,802,883	35,556,261
kaist	ko	304.92 MiB	0.07%	11,157	1,926,901	30,929,508
reuters_financial	en	288.63 MiB	0.06%	101,055	1,983,069	49,495,061
aihub_book	ko	236.66 MiB	0.05%	180,001	1,201,956	23,052,720
aihub_koen_formal	ko	206.37 MiB	0.04%	1,350,000	1,350,000	20,659,619
aihub_koen_ssci	ko	186.49 MiB	0.04%	1,361,845	1,361,845	19,104,237
aihub_koen_sci	ko	164.42 MiB	0.04%	1,344,631	1,344,631	17,720,448
fomc	en	112.66 MiB	0.02%	2,822	950,620	18,640,148

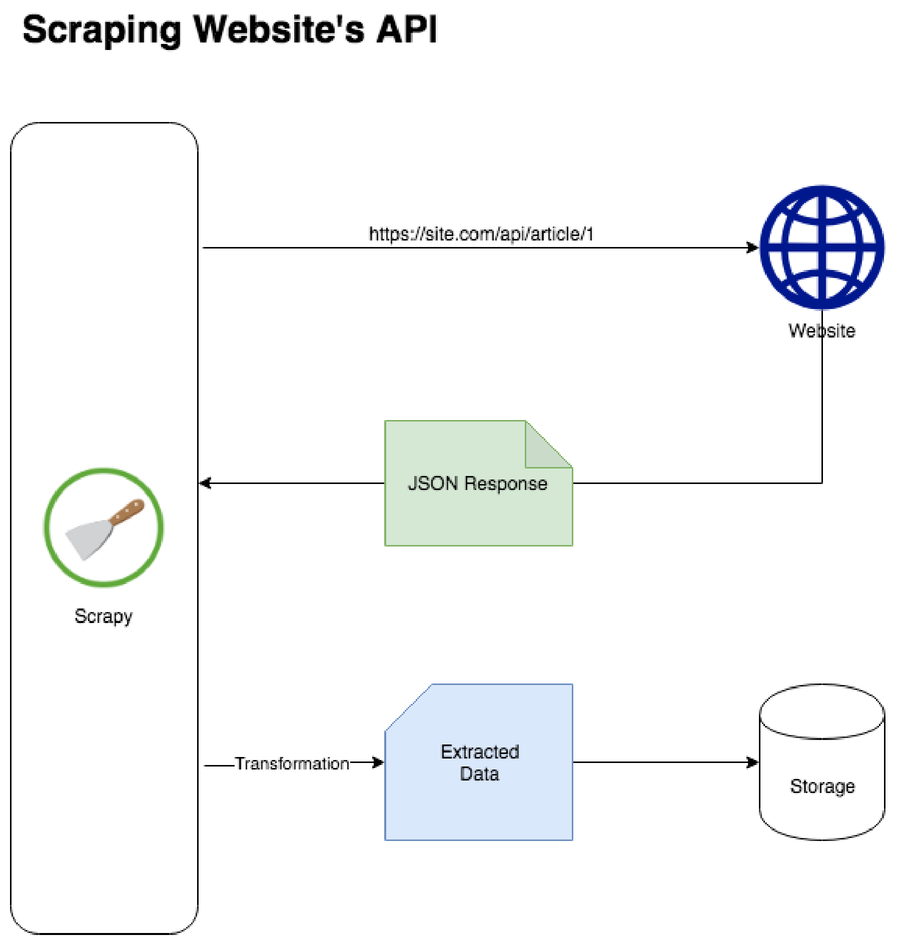
Name	Language	Size	Weight	# Docs	# Sents	# Words
esg_report	ko	24.17 MiB	0.01%	15,561	119,031	2,488,545
aihub_law_kb	ko	9.99 MiB	0.00%	17,373	46,140	934,632
bok_minutes	ko	9.54 MiB	0.00%	163	33,027	918,203
pathobook	en	4.28 MiB	0.00%	28	33,603	648,221
English	en	258.83 GiB	57.66%			
Korean	ko	190.04 GiB	42.34%			
Total		448.87 GiB	100.00%			

Web Scraping



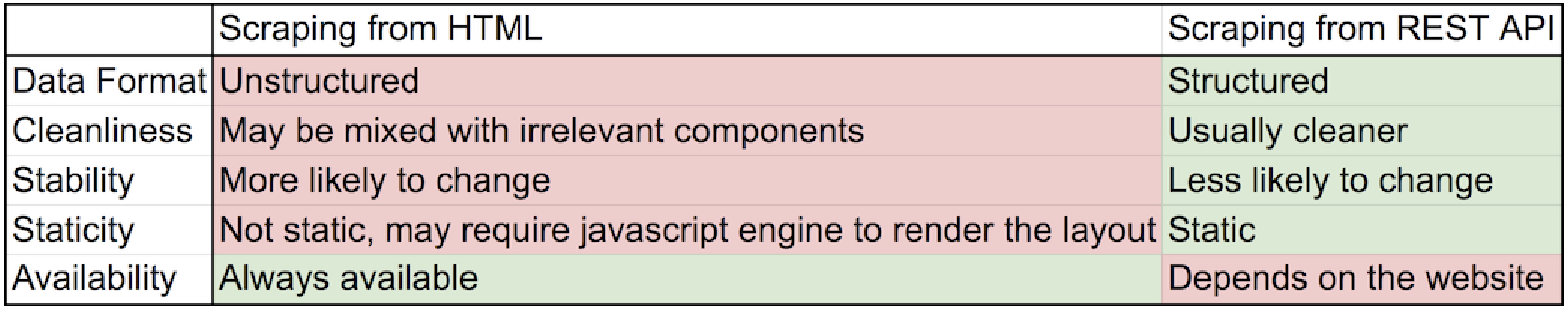
Web Scraping: Vanilla Web Page vs. REST API





Vanilla Web Page vs. REST API

- Web scraping, by the definition itself, is to extract structured data from unstructured sources. Most of the data displayed from the websites are only good for human eyes. The underlying data are raw, complex, and unfriendly to machines.
- Giving raw data a structure is difficult, but what if the source data itself are already structured?
- With the rise of modern web app frameworks like React and Vue.js, more and more sites are using REST API to send and receive data, then render the final layout in the client side.



	Scraping from HTML	Scraping from REST API
Data Format	Unstructured	Structured
Cleanliness	May be mixed with irrelevant components	Usually cleaner
Stability	More likely to change	Less likely to change
Staticity	Not static, may require javascript engine to render the layout	Static
Availability	Always available	Depends on the website

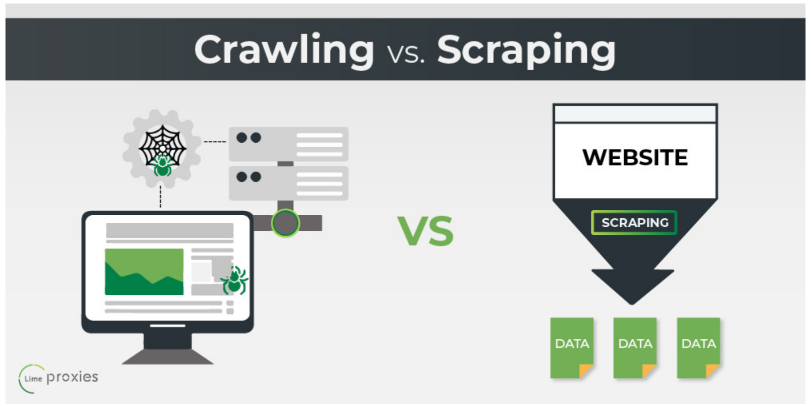
Web Crawling Vs. Web Scraping

Web Scraping

Web Scraping is the process of extracting specific data from web pages. It involves the process of sending a web request and getting a web page returned as a response, then parsing it to extract the required data while every other content is left.

Web Crawling

Web crawling, on the other hand, takes a more generalized approach, visiting web pages and keeping records of what's on them and then extracting the links on the page that meets specific criteria to add to the list of links to be crawled.



Dump Dataset: Wikipedia

Downloading a Wikipedia dump

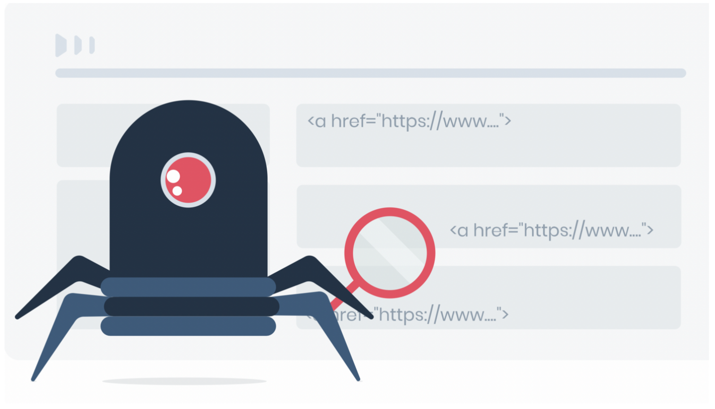
Wikipedia dumps are freely available in multiple formats in many languages. For the English language Wikipedia, a full list of all available formats of the latest dump can be found here

Extracting and cleaning a Wikipedia dump

The Wikipedia dump we've just downloaded is not ready to be pre-processed (sentence-tokenized and one sentence-per-line) just yet. First, we need to extract and clean the dump, which can easily be accomplished with WikiExtractor

Common Crawl

- Common Crawl is a non-profit organization which crawls the web and provides datasets and metadata to the public freely.
- The Common Crawl corpus contains petabytes of data including raw web page data, metadata data and text data collected over 8 years of web crawling.
- Common Crawl data are stored on Public Data sets of Amazon and other cloud platforms around the world.
- So access to the Common Crawl corpus is free.
- You can use Amazon's cloud platform to perform analysis jobs by downloading it.



Choose the Right Tool

BeautifulSoup:

- Beautiful soup is a library for parsing HTML and XML documents.
- Requests (handles HTTP sessions and makes HTTP requests) in combination with BeautifulSoup (a parsing library) are the best package tools for small and quick web scraping.
- For scraping simpler, static, less-JS related complexities, then this tool is probably what you’re looking for.

Scrapy:

- Scrapy is a web crawling framework that provides a complete tool for scraping.
- In Scrapy, we create Spiders which are python classes that define how a particular site/sites will be scrapped.
- So, if you want to build a robust, concurrent, scalable, large scale scraper, then Scrapy is an excellent choice for you.
- Also, Scrapy comes with a bunch of middlewares for cookies, redirects, sessions, caching, etc. that helps you to deal with different complexities that you might come across.

Selenium:

- For heavy-JS rendered pages or very sophisticated websites, Selenium webdriver is the best tool to choose.
- Selenium is a tool that automates the web-browsers, also known as a web-driver.
- With this, you can open a Google Chrome/Mozilla Firefox automated window, which visits a URL and navigates on the links.
- However, it is not as efficient as the tools which we have discussed so far.
- This tool is something to use when all doors of web scraping are being closed, and you still want the data which matters to you.

Web Scraping with Beautiful Soup

Extract data from a web-page using automatic scraping and crawling with Beautiful Soup

Elements

The important thing about HTML is that the markup is represented by elements. An HTML element is a portion of the content that is surrounded by a pair of tags of the same name. Like this:

```
<strong>This is an HTML element.</strong>
```

In this element, strong is the name of the tag; the open tag is `<strong>`, and the matching closing tag is `</strong>`. The way you should interpret this is that the text “This is an HTML element” should be “strong”, i.e., typically this will be bold text.

HTML elements can and commonly do nest:

```
<strong>This is strong, and <u>this is underlined and strong.</u></strong>
```

In addition to the names, opening tags can contain extra information about the element. These are called attributes:

```
<a href='http://www.google.com'>A link to Google's main page</a>
```

In this case, we’re using the `a` element which stood for “anchor”, but now is almost universally used as a “link”. The attribute `href` means “HTML reference”. The meaning given to each attribute changes from element to element.

Other important attributes for our purposes are `id` and `class`. The id attribute gives the attribute a unique identifier, which can then be used to access the element programmatically. Think of it as making the element accessible via a global variable.

The class is similar but is intended to be applied to a whole “class” of elements.

HTML pages require some boilerplate. Here is a minimal page:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
Hello World! What's up?
</body>
</html>
```

The `<head>` contains meta-information such as the title of the site, the `<body>` contains the actual data.

Hierarchy

Data in HTML is often structured hierarchically:

```html
<body>
  <article>
    <span class="date">Published: 1969-10-22</span>
    <span class="author">Led Zeppelin</span>
    <h1>Ramble On</h1>
    <div class="content">
    Leaves are falling all around, It's time I was on my way.
    Thanks to you, I'm much obliged for such a pleasant stay.
    But now it's time for me to go. The autumn moon lights my way.
    For now I smell the rain, and with it pain, and it's headed my way.
    </div>
  </article>
</body>
```

Here, the title of the song is nested three levels deep: `body > article > h1`.

Tables

Data is also often stored in HTML tables, which are enclosed in a `<table>` tag. `<tr>` indicates a row (table row), `<th>` and `<td>` are used to demark cells, either header cells (`<th>`, table header) or regular cells (`<td>`, table data). Here's an example:

```
<table>
    <tr>
        <th></th>
        <th>The Beatles</th>
        <th>Led Zeppelin</th>
    </tr>
    <tr>
        <td># Band Members</td>
        <td>4</td>
        <td>4</td>
    </tr>
</table>
```

The DOM

As we have seen above, a markup document looks a lot like a tree: it has a root, the HTML element, and elements can have children that are containing elements themselves.

While HTML is a textual representation of a markup document, the DOM is a programming interface for it. Also the DOM represents the state of a page as it's rendered, that (nowadays) doesn't mean that there is an underlying HTML document that corresponds to that exactly. Rather, the DOM is dynamically generated with, e.g., JavaScript.

In this class we will use “DOM” to mean the tree created by the web browsers to represent the document.

Inspecting the DOM in a browser

Perhaps the most important habit when scraping is to investigate the source of a page using the Developer Tools. In this case, we'll look at the **element tree**, by clicking on the menu bar: View → Developer → Developer Tools.

Alternatively, you can right click on any part of the webpage, and choose "Inspect Element". Notice that there can be a big difference between what is in the DOM and what is in the source.

Fetching a Website

Scraping a website can cause quite a high load on a server. To avoid that, webmasters usually publish what kinds of scraping they allow on their websites. You should check out a website's terms of service and the `robots.txt` of a domain before crawling excessively. Terms of service are usually broad, so searching for “scraping” or “crawling” is a good idea.

Let's take a look at Google Scholar's robots.txt:

```
User-agent: *
Disallow: /search
Allow: /search/about
Disallow: /sdch
Disallow: /groups
Disallow: /index.html?
Disallow: /?
Allow: /?hl=
...
Disallow: /scholar
Disallow: /citations?
```

Here it specifies that you're not allowed to crawl a lot of the pages. The `/scholar` subdirectory is especially painful because it prohibits you from generating queries dynamically.

It's also common that sites ask you to delay crawiling:

```
Crawl-delay: 30
Request-rate: 1/30
```

You should respect those restrictions. Now, no one can stop you from running a request through a crawler, but sites like google scholar will block you VERY quickly if you request to many pages in a short time-frame. It's also common that bigger sites serve up a CAPTCHA if they think you're using a bot to crawl.

An alternative strategy to dynamically accessing the site you're crawling (as we're doing in the next example) is to download a local copy of the website and crawl that. This ensures that you hit the site only once per page. A good tool to achieve that is wget.

Regular Expressions

> Regular Expressions: A formal language for specifying text strings

How can we search for any of these?

- woodchuck
- woodchucks
- Woodchuck
- Woodchucks

Regular Expressions: Disjunctions

> Letters inside square brackets []

Pattern	Matches
[wW]oodchuck	Woodchuck, woodchuck
[1234567890]	Any digit

> Ranges [A-Z]

Pattern	Matches	
[A-Z]	An upper case letter	`D`renched Blossoms
[a-z]	A lower case letter	`m`y beans were impatient
[0-9]	A single digit	Chapter `1`: Down the Rabbit Hole

Regular Expressions: Negation in Disjunction

Negations `[^Ss]`

- Carat means negation only when first in []

Pattern	Matches	
[^A-Z]	Not an upper case letter	O `y` fn pripetchik
[^Ss]	Neither ‘S’ nor ‘s’	`I` have no exquisite reason”
[^e^]	Neither e nor ^	`L` ook here
a^b	The pattern a carat b	Look up `a^b` now

Regular Expressions: More Disjunction

Woodchuck is another name for groundhog!

The pipe | for disjunction

Pattern	Matches
groundhog	woodchuck
yours|mine	yours
a|b|c	= [abc]
[gG]roundhog|[Ww]oodchuck	Woodchuck

Regular Expressions: ? *+.

Pattern	Matches	
colou?r	Optional previous char	color colour
oo*h!	0 or more of previous char	oh! ooh! oooh! ooooh!
o+h!	1 or more of previous char	oh! ooh! oooh! ooooh!
baa+		baa baaa baaaa baaaaa
beg.n		begin begun begun beg3n

Regular Expressions: Anchors ^ $

Pattern	Matches
^[A-Z]	`P`alo Alto
^[^A-Za-z]	`“`Hello”
\.$	The end`.`
.$	The end`?` The end`!`

Example

Find me all instances of the word “the” in a text.

> the

Misses capitalized examples

> [tT]he

Incorrectly returns other or theology

> [^a-zA-Z][tT]he[^a-zA-Z]

Errors

The process we just went through was based on `fixing two kinds of errors:`

1. Matching strings that we should not have matched (there, then, other)

- False positives (Type I errors)

2. Not matching things that we should have matched (The)

- False negatives (Type II errors)

In NLP we are always dealing with these kinds of errors.

Reducing the error rate for an application often involves two antagonistic efforts:

- Increasing accuracy or precision (minimizing false positives)
- Increasing coverage or recall (minimizing false negatives).

Summary

Regular expressions play a surprisingly large role

- Sophisticated sequences of regular expressions are often the first model for any text processing text

For hard tasks, we use machine learning classifiers

- But regular expressions are still used for pre-processing, or as features in the classifiers
- Can be very useful in capturing generalizations