Barcelona GSE Data Science Center

Scraping

Terminology: Crawling vs Scraping

Web crawling refers to the act of traversing the internet by a bot, of a piece of software.

Search engines, traditionally, crawl the internet and don't do much with the content except index the text.

Scraping refers to the act of extracting specific data from web pages.

Often, scraping involves some crawling, and uses the same libraries and techniques. Thus, the terms and concepts will overlap.

Scraping

Scraping consists of:

1. Making HTTP requests to servers for HTML content.
2. Parsing that HTML content to:
   - 2.1 Store desired information.
   - 2.2 Find links to follow (for each link, go back to 1.)

Making HTTP Requests

Browsers make HTTP requests.

Every major programming language also has a way to make HTTP requests.

This is sometimes done via a third-party library.

Parsing HTML

HTML parsing is something you will use a trusted library for.

HTML parsers take the raw HTML from an HTTP response, and turn it into a (usually custom) tree-like data structure or class that you can easily traverse, and from which you can easily extract the desired content.

This functionality is conceptually different from that of making the HTTP request. It will usually be included in a separate library, for this reason.

You will need to learn the API, the interface, of the HTML parser you are using!

Read the documentation.

Barcelona GSE Data Science Center

Following Links

Making an HTTP request and parsing it is all you need to scrape a single page.

Usually, however, we want to scrape more than one page.

How do we get all the pages we will scrape?

Often, we get them from links in other pages!

Following Links

What are some websites you might want to scrape?

Which pages?

How can we access all the pages?

Sitemaps

Some sites provide a link to an XML sitemap in their robots.txt file (more on that later).

Other sites provide a sitemap directly as an HTML page, labelled “sitemap”.

Still others provide no sitemap at all.

Sitemaps are generally meant as a way in which crawlers can easily get to all the pages on the site. There might also be some sort of “directory” pages for part of the content.

Rules of the road

What can you scrape?

What should you scrape?

What is legal to scrape?

Public vs. Private

There are two types of content on the web: that which everybody can see (public), and that which only certain individuals can see (private).

When you login to a website, you usually see some private content. You also agreed to a legal document, whether you read it or not, their Terms and Conditions!

Those Terms and Conditions can, and often do, make it illegal for you to scrape private content. And because you have agreed to them, you are bound by them.

Public vs. Private

Public content, on the other hand, is less black and white.

There are websites who are happy with you scraping their content, as long as you do it politely.

There are others that don't want you scraping their content unless they know who you are. Almost everyone wants Google to crawl and index all their pages. But they may not want their competitor doing the same!

Being Polite

Let's assume you have the website's blessing.

How does one act politely?

1. Follow robots.txt file
2. Scrape slowly
3. Identify yourself

Robots.txt

Most major websites will have a robots.txt file. This is just a text file that they create in order to tell bots (web crawlers and scrapers) the rules of their website.

You should obey robots.txt files, it's part of being a good citizen on the web!

Let's look at an example. Canonically, they are always at /robots.text:

https://www.airbnb.com/robots.txt

Mostly, they just describe which paths bots are allowed to access, and which they are not.

Scraping Speed

HTTP requests take time to complete, even at the speed of light, the data might have to go all around the world and back.

While an HTTP request is being made, your computer, and it's processor, is idling. Your processor can prepare many other requests, and handle many other responses, while waiting for its first HTTP request to complete (it could be hundreds of milliseconds!).

Barcelona GSE Data Science Center

Scraping Speed

Scraping in parallel can happen, depending on the language, via processes, threads, or an asynchronous event loop.

Modern machines can thus make many requests very quickly!

However, servers are limited in how many requests they can handle at a given time. For this reason, they prefer to spread the load of requests as evenly as possible, avoiding large spikes in usage.

For this reason, they want you to scrape slowly.

Identify Yourself

One Header that you can send with an HTTP request is that of “user-agent”.

In the case of normal web browsing, “user-agent” refers to the exact browser and version being used.

In the case of scraping, however, it is polite to use a name that refers to your bot and your website, thus that identifies you uniquely so their engineers can know who you are.

Problems

- Getting Blocked
- Javascript

Barcelona GSE Data Science Center

Getting blocked

If you scrape too quickly, scrape from a commercial IP address (AWS), or the website doesn't know you, you might be blocked from crawling. Instead of giving you the page you asked for, they might give you a different page, potentially with a 403 status code, that tells you that you have been denied access.

There are many ways around being blocked. You can lie about your user-agent, use proxies, hold on to cookies, etc.

In general, however, you should be careful. Even if you feel you are ethically justified, this can be a slipper cat and mouse game that eats up a lot of your time!

It might be easier just to ask the website to whitelist you!

Javascript

We have been focusing on the paradigm wherein the server responds to HTTP GET requests with HTML.

Sometimes, however, not all the HTML that we see in our browser is actually sent by the server. The server might send a small amount of HTML, along with some Javascript code, whose responsibility it is to generate the rest of the HTML.

This poses a major problem for scraping, as the content we want isn't returned by the server!

Javascript

The solution is to use a headless browser.

A headless browser is just a browser that does not render the content to a UI.

Headless browsers can be embedded within your scraping program via a library, or run as a separate piece of software and accessed over HTTP.

Popular options: Selenium and Splash

Storing Data

There are many options for storing data from scraping:

- Flat files (json lines, csv, etc.)
- Database

Following Many Links

Let's return to the problem of following links.

Often, the links grow expontentially in number as we scrape.

This is because one page in a directory or search results might link to 10-20 "detail" pages which are often the ones we actually want data from.

In other words, we might want to scrape thousands or millions of individual pages, but we won't have that list of pages ahead of time, we will build it as we go.

Following Many Links

How can we deal with this ever-growing list?

- Loops in loops in loops
- Recursive function calls
- A queue

Which of these will work in a distributed or parallel framework?

Elegantly, only the quee.

Barcelona GSE Data Science Center

Production Scraping

When you actually want to scrape a large site, you will need to make requests in parallel to get the speed needed.

You can build this yourself quite simply, or use a scraping library that gives you this for free.

Scrapy is a Python library that gives you this, and much more, for free. It's a highly opinionated and structured library, so there is a learning curve, but it is well documented and popular.

Barcelona GSE
Data Science Center