Luling Huang
October 2016
DSC, Temple University

Demonstration of Web Scraper

The working process is to build a sitemap (like a tree graph) of different types of selectors:

(1) Specify the starting URL for scraping (Web Scraper calls it “_root”).

(2) Choose the appropriate selector and give it a name. The commonly used selectors include link, element, text, and element attribute selectors. The documentation provides straightforward description of each selector.

(3) Think about which selector is parent, and which is child. The basic structure of my data is: root is the parent of thread, and thread is the parent of replying post. My sitemap is illustrated in Figure 1. In my case, each page has 20 threads. Under the root, I created a link selector for all 20 threads (i.e., "thread_link" in Figure 1) and made it a child of the root. Then the scraper click the link and enter the web page for each thread. Next, I created an element selector ("post_info") for each replying post within a thread’s page, and make it a child of "thread_link". At this level, I created a bunch of text selectors to extract the data I want. These text selectors are children of "post_info."

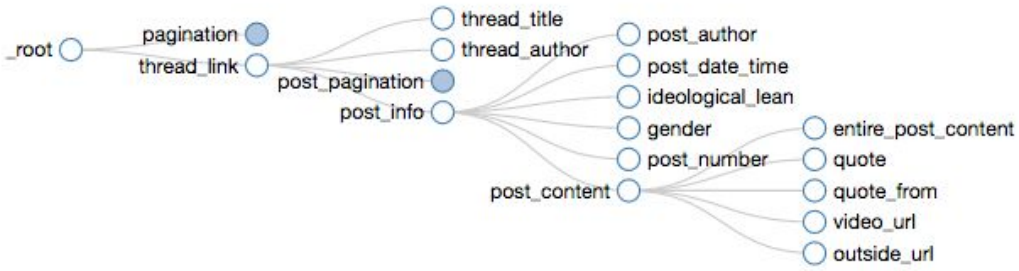

Figure 1. A Web Scraper sitemap.

(4) Deal with multiple pages of the same type of data. For the subforum I am scraping, there are more than 800 threads (by 5:00 PM, 09/14/2016), which means there are more than 40 pages of thread listing. And within some threads, there are more than ten pages of replying posts. The solution is to create a link selector that click the “next page” button at every page of the same level, and also set the link selector as a parent to itself. Therefore, when the scraper finishes scraping at page #1, for example, it will automatically scrapes at page #2. As shown in Figure 1, I created "pagination" at the thread level, and "post_pagination" at the post level to execute the above functions. The

official tutorials introduce two ways (the first and the second) to deal with multiple-page scraping. Choice should be made according to the way page navigation is designed.

(5) Tweak the selector. Sometimes the scraper's selector fails in its guess. For example, when I scrape a post author's profile, some information is not displayed if the author does not disclose it. This creates a problem when the scraper makes its guess. Because the profile information is structured as a description list with a group of <dt> and <dd> tags, if I click to select the row of "Lean", the selector returns "dd:nth-of-type(4)" where 4 indicates that "Lean" occupies the fourth row in the list (Figure 2). But if the location information is missing, "Lean" is in the third row (Figure 3). The solution is to specify which <dt> should be selected for scraping ideological lean. The selector can be manually set as dt:contains('Lean') + dd. Now the scraper knows where to look at and it returns data like "Liberal" or "Moderate" for ideological lean.

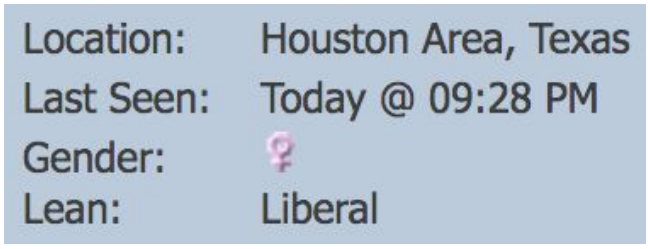

Figure 2.

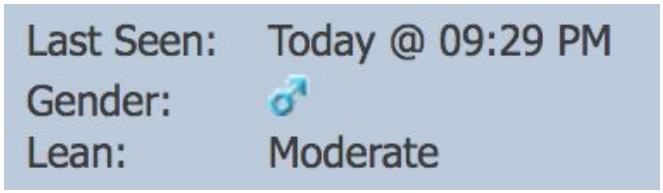

Figure 3.

(6) Give the server some time to respond. For the pagination selector, the tutorial suggests a delay of 2000 ms (2 seconds). It took some time (several hours or more) to scrape the data.

(7) It is a good habit to always use the "data preview" button to check if selectors are correctly set up before scraping.

(8) Web Scraper's Google Groups forum is a great resource for troubleshooting.

(9) Export the data as CSV file and clean the data. The scraper does not return data in the order that appears on web pages. So when cleaning the data, I will use thread name to group replying posts, and use the information of date and time to rebuild the temporary order of posts.