11-21-2019

Building an Alternative Web Scraper for Big Data Analytics

Kurtis Clark  
_Southwestern Oklahoma State University, clarkkd@student.swosu.edu_

Amanda Evert  
_Southwestern Oklahoma State University, amanda.evert@swosu.edu_

Follow this and additional works at: https://dc.swosu.edu/cpgs_edsbt_bcs_student

Part of the _Computer Sciences Commons_

**Recommended Citation**
https://dc.swosu.edu/cpgs_edsbt_bcs_student/24

This Poster is brought to you for free and open access by the Business & Computer Science at SWOSU Digital Commons. It has been accepted for inclusion in Student Research by an authorized administrator of SWOSU Digital Commons. An ADA compliant document is available upon request. For more information, please contact phillip.fitzsimmons@swosu.edu.
Abstract

Over the summer, I was given access to WebScraper3000 and asked to web scrape, collect internet data, from ratemyprofessors.com. I spent several months working with the program to accomplish this. It only ever seemed to work properly on websites where it has tutorials about scraping them. Even then, the program would only scrape the most rudimentary data in small amounts. If either of these two conditions were missing, the software would lose large quantities of data or just simply crash along the way. After collecting enough data to act as a sample for the research, I decided to abandon the software and begin looking into building my own alternative. I decided on using the Python programming language and its beautifulsoup4 import. The result ended with a far smoother and more complete data collection and set.

Web Scrapping Summarized:

Now find how many times the word 'superfluous' appears.

Issues found with WebScraper3000:

- The interface tries to be intuitive and user friendly; this only made it restrictive.
- The program interface is cluttered with multiple windows, yet these windows cannot be separated into multiple monitors.
- When building a workflow, it is overly complicated to edit any desired data item.
- There are servers; these are advertised to run smoother and faster; they do not.
- The server scrapes crash more often.
- The server runs slower.
- The server ‘loses’ more data sets.
- It is common for the scrapes to ‘lose’ data from one scrape to another.
- Many times, the program would forget the URL list size and only iterate for the first 100 URLs.

My Alternative

- When I decided to build my own web scraper, I did some light research and discovered that Python has a library called Beatifulsoup4. This library is specifically designed to support web scraping. It uses regular expressions and html tags to locate data within an html code. The html code is imported using a popular library called requests. After everything was completed, the total time needed for the scrape and parse was only a couple of days for all 1.6 million data sets, compared to taking just as long to only collect 150,000 with WebScraper3000.
- Creating my own scraper was not the simplest walk in the park. I ran into many issues that I had to work through. Some of these may even be some of the reasons WebScraper3000 was having so many issues.
- Because of the massive scale of the data I was dealing with, and the technical limitations of my PC, I had to separate the entire process into 8 parts.
- On several occasions my scraper would break; after some investigations I found that there were several ‘blank’ professor entries on the website. This was remedied with the addition of the try method. Nearly 200,000 of these were found by deleting the duplicates from the database I imported the data set into.
- I ended up removing the Name section from my data set as it was causing error with special characters not being recognized within the code structure. Even uncode was not recognizing these characters.

What is Beatifulsoup4?

- Beatifulsoup4 is a library import for Python.
- It is specifically made for going through html code, finding certain key words and phrases and the pulling the associated information from the code. It, combined with another library, requests, that pulls the html code from the internet in its raw form, are commonly used to build custom web scrapers like the one I have built.

Ethics of Scraping

There are ethics of use for every tool. Web scraping is no exception. For a lot of websites, web scraping can be seen as nothing more than a simple DOS attack. Because of this, many sites create an API that can be scraped rather than their main site. These APIs are also better equipped and much friendlier to scrape, as all the data is plainly identified and not hidden deep in the html code. Sadly, not all websites have an API like this, including ratemyprofessor.com. In cases like this, it is vital to contact the website first, before beginning large scale scrapes. If the site is not equipped for scrapes, it can cause the site to crash, which in turn would also crash the scrape. If the site representative says not to scrape their site, one alternative is to look around in the open source market and see if there isn’t already a full scrape of the website available. In the case of ratemyprofessor.com, there was already a scrape available, but it did not have a totally complete data set and was several years old, and I already had permission form the site to scrape them, so I decided to continue with my own scrape.

Ethics of Data Analysis

By analyzing large contents of data, we can learn a lot. The question is though, is this analysis ethically sound. Just like with any kind of research, big data analysis can produce any result you want it to. With very simple sorting of data, the results of an analysis can be skewed dramatically. That is why it is imperative to be impartial during the analysis and use as much of the data as possible with as least sorting as possible. Otherwise the trends shown, are the trends made, not the trends found.