CS839 - Project Stage 2

Chakshu Ahuja, cahuja2@wisc.edu
Geetika, geetika@wisc.edu
Swati Anand, sanand@cs.wisc.edu

Data Sources

We are extracting information about the Books from the below e-commerce websites that seemed to contain many matching entities:

Amazon: https://www.amazon.com/b/ref=usbk_surl_books/?node=283155
Barnes&Noble: https://www.barnesandnoble.com/

Wrapper Extraction Procedure

We crawled these two web sites to fetch the book lists from categories like Literary Fiction, Classics, Biographies and Ancient History in the order of their Bestsellers rank. We were able to crawl at most 2000 books from each category at *Barnes&Noble* and 1200 books from each category at *Amazon*. From the book lists page, we crawled each individual book's URL to fetch attributes of the book using *XPath* or *CSS Selectors* based extraction method. We were able to do that since all the pages seemed to follow a similar template. There were a few exceptions wherein the pages didn't follow the exact same template for one or more attributes on the page (eg, the price was listed twice in Used and New books sub-categories). Our crawler has currently ignored those pages.

We have used two different frameworks (Selenium, Scrapy) for extracting data from both the sources owing to the programmer's skillset, although the procedure used was the same as described above.

For fetching entities from Amazon, the starting source URLs (of each sub-category) are supplied and the *Selenium* web driver launches the Firefox browser and visits each page. The driver is then used to extract the attributes of the book using XPath or other HTML/CSS attributes that are identified as following a template on each page. The results are initially maintained as a Python dictionary keyed on the book page URL so that duplicate books are not crawled multiple times. When the crawling of each page is completed, the results are stored in a CSV. This also enables recovery in case of failures, where in if script fails after scraping 1000 books, we can directly

start from 1001 count since the information for the already scraped 1000 books have been stored into persistent CSV file. Since there are clicks involved in the scraping process as normal user would do, it prevents the crawler from being detected as a bot by the web source (in contrast to other scraping libraries available). We have added explicit timeouts and also multiple try/except so as a human need not monitor the process at all times and after a specified timeout, the crawler can move on to next book rather than being halted for a long time.

For fetching entities from Barnes&Noble, we have defined a spider using Scrapy and the data is extracted in CSV format. A list of start URLs is defined to crawl the books in each category. The response returned by the crawler is parsed to fetch book attributes from the page using XPath and CSS Selectors. To ensure crawling at a high speed and prevent blocking, we have used rotating proxies for subsequent requests and have set the maximum attempt count for each page to 10. When 200 response code is not received from a page, the proxy used to make that request is marked as dead and the page is requested again with another proxy.

Dataset

Number of tuples in table A (Amazon): **4753**
Number of tuples in table B (Barnes&Noble): **5469**

We have extracted **Books** entity from both the sources. The schema followed during extraction is as follows:

Attribute Name	Attribute Type	Description
title	String	Title of the book
authors	Text	Names of one or more authors of the book separated by a comma
book_format	Categorical	The format of the book such as Paperback, Hardcover, Audiobook, etc
cur_price	String	Current Price of the book after discounts and offers
old_price	String	Original Price of the book
isbn13	Integer	ISBN-13 identifier of the book

publisher	String	Publisher of the book
publication_date	Date	Date when the book was first published (mm/dd/yyyy)
pages	Integer	Number of pages in the book
product_dimensions	String	Dimensions of the book (width, depth, and height)

After extracting the data from both the web sources, 3 attributes: **isbn13**, **publication_date**, and **product_dimensions** did not follow the same representation format in Amazon and Barnes&Noble. So we cleaned the data from Amazon and converted it to the format followed by Barnes&Noble.

Open Source Tools

Selenium: We have used Selenium web driver to crawl and fetch entities from Amazon. Selenium WebDriver[2] accepts commands (sent in Selenese, or via a Client API) and sends them to a browser. This is implemented through a browser-specific browser driver, which sends commands to a browser and retrieves results. Most browser drivers actually launch and access a browser application (such as Firefox, Chrome, Safari).

Gecko driver[3]: This provides the HTTP API described by the WebDriver protocol to communicate with Gecko browsers, such as Firefox. It translates calls into the Firefox remote protocol by acting as a proxy between the local and remote ends.

Scrapy[1]: Scrapy is a Python Framework used to extract data from Barnes & Noble. It provides Spiders which are self-contained crawlers and can be given a set of instructions. It's mostly used for larger projects and provides multiple ways for reusing and maintaining code. It is also extremely fast as it uses asynchronous or non-blocking method for crawling a group of pages.

Scrapy-rotating-proxies[4]: This package is used as a Scrapy middleware to prevent it from getting blocked by the web source after making multiple requests from the same IP address in a short span of time. This allows Scrapy to use rotating proxies for subsequent web requests, check that they are alive and adjust crawling speed depending on the number of proxies that are alive.

US-proxy source[5]: We have used around 80 US proxies from this web source.

References

[1]. https://www.datacamp.com/community/tutorials/making-web-crawlers-scrapy-python
[2]. https://www.seleniumhq.org/projects/webdriver/
[3]. https://github.com/mozilla/geckodriver
[4]. https://pypi.org/project/scrapy-rotating-proxies/
[5]. https://www.us-proxy.org/