CS 839 Project Stage 2: Crawling and Extracting Structured Data from Web Pages - Extracting Laptops from Online Retailers

Aishwarya Ganesan
ag@cs.wisc.edu

David Liang
david.liang@wisc.edu

Viswesh Periyasamy
vperiyasamy@wisc.edu

Web Data Sources

1. **Amazon:** The first web data source we chose was the online retailer Amazon (**www.amazon.com**). Although it is notoriously tricky to scrape, Amazon offers a wide variety of products and detailed information so for the purpose of building a data set on electronic products, it was a natural choice.

2. **Walmart:** The second web data source we chose the Walmart's online store (**www.walmart.com**). We chose Walmart for a similar reason as Amazon - it too offers a wide selection of electronic products and more so than some other competitors such as BestBuy or Target.

Extraction of Structured Data

1. **Amazon:** To extract data from Amazon, we started from a basic Amazon web page crawler someone had created which only was used to find product page links (code base can be found here: https://github.com/hartleybrody/public-amazon-crawler). However, we needed to change it to fit our purposes and also add much more functionality to scrape actual product pages (the changes we made can be found here: https://github.com/aishwaryaganesan/public-amazon-crawler/).

   We used some scraping libraries which allowed us to directly parse the HTML of the web pages and extract the needed information. By visually inspecting the HTML source code of pages, we could label which elements were of importance to us and hard-code our scraper to find and extract these elements. We started by heading to a section page which listed several products and employing our scraper to identify unique products along with the embedded links to that product's page. We then dumped each product URL to a file and cross-checked for duplicate URLs which we might have come across while traversing many pages of product listings.

   Once we had this list of several thousand product URLs, we modified our scraper to look at each product page and parse information from the tables residing in the product description. Although not every product had an identical table format, many were fairly consistent and we were able to consolidate them all by manually finding similar column names and unifying the data. Once we had amassed tuples for every product page in our list of URLs, we converted the final data into a CSV-formatted file.

2. **Walmart:** Extracting structured data from Walmart followed a very similar format to Amazon with a few differences. By modifying our Amazon crawler, we were able to generalize to extract from Walmart as well. We used a similar method of inspecting the desired HTML elements and coding them into our scraper, but we had to modify it because some of the content was generated dynamically which was not initially recognized by the scraper. After that, we again collected a list of product URLs by going to a category page and extracting the embedded URLs from each product thumbnail and discarded duplicate URLs.

   Using the list of product URLs, we changed our scraper to parse the tabular information from each product page again. Unfortunately, the tables were not standardized across product pages so we made it more generalized and also came across a lot of missing values for attributes. To try and consolidate this, we again manually found similar column names across Amazon and Walmart and grouped them together as one single attribute. Finally, we converted the tuples to a CSV-formatted file.

Entity of Choice

Our entity of choice across both web sources was **laptops**. We chose laptops because they are abundant in both websites, product features are relatively standard, and various retailers often offer the same products. Each table represents all of the product tuples we were able to extract from the respective web source, and the attribute columns represent our consolidated versions that matched similar but not identical product features. A full list of the attributes can be found in the README of the data directory (https://github.com/dliangsta/cs839-project/tree/master/stage-2/crawlers/data).

- **Amazon:** 3,068 tuples
- **Walmart:** 4,847 tuples

Open Source Tools

To write our data extraction programs, we used a number of open-software packages available for Python:

- **BeautifulSoup (bs4)** - Beautiful Soup is a Python package for parsing HTML and XML documents. It creates a parse tree for parsed pages that can be used to extract data from HTML, which was very useful for our scraping.
- **dryscrape** - dryscrape is a lightweight web scraping library for Python. It uses a headless Webkit instance to evaluate Javascript on the visited pages, which was essential for scraping the dynamically generated content from Walmart.
- **eventlet** - Eventlet is a concurrent networking library for Python that allows non-blocking I/O and is used in conjunction with the standard urllib2 package for opening URLs.

- **HTMLParser** - HTMLParser is another library which we used to scrape HTML content which was returned from web requests. This was used more heavily for the Amazon web crawler than the Walmart web crawler.