Machine Learning Report

Qiaotong Huang
College of Engineering
Northeastern University
Toronto, ON
huang.qiaot@northeastern.edu

1 Introduction

I tried to use selenium to crawl Amazon, the largest shopping website in the world. The following is the detailed process.

1.1 Problem Definition

Data Extraction: The primary objective of an Amazon crawler is to extract data from Amazon's website. This data can include product information (e.g., title, price, description, reviews), seller information, product images, and more.

Data Structuring: The extracted data needs to be structured and organized in a usable format (e.g., JSON, CSV) for further analysis, comparison, or integration with other systems.

Robustness and Scalability: The crawler needs to be robust enough to handle various scenarios, such as changes in website layout, rate limiting, and large volumes of data. It should also be scalable to handle increased data extraction demands.

1.2 Motivation

Market Analysis: Companies may use Amazon crawlers to gather data for market analysis, including pricing trends, product popularity, and competitor analysis.

Price Monitoring: Retailers and sellers may use Amazon crawlers to monitor prices of their products as well as competitors' products.

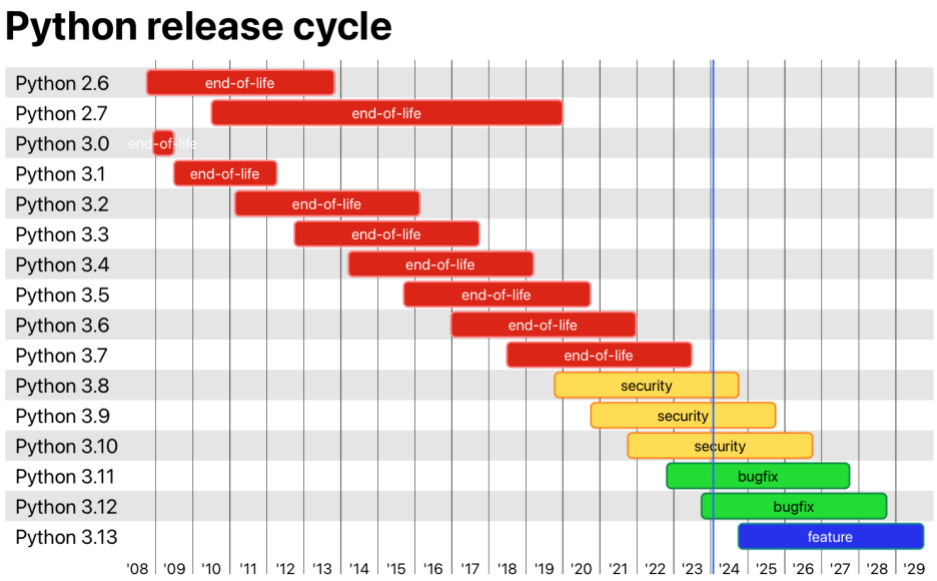
Product Research: Consumers and businesses may use Amazon crawlers to research products, read reviews, and compare prices before making purchasing decisions.

2 Methodology

The main crawler tool used is selenium.

2.1 Environment Setup

The picture below shows the status of the python versions. You can see that the versions before 3.7 is already in the end-of-life status. So version 3.8 is used here.



Other libraries used include:

selenium==4.13.0

prettytable==3.9.0

bs4==4.12.2

Can be found in the **requirements.txt** file.

2.2 Tools

In order to make the table more beautiful, the prettytable library is used.

3 Solution

First do a single page analysis, then find the next page link, and then let the whole process connect together.

3.1 Extract Page Information

First crawl a single page and analyze the elements. Find out the attribute common to product items is 'class': 'template=SEARCH_RESULTS'. In order to be more accurate, regular matching is used here, but later I found that the target can be accurately found using only the class attribute.

```
ProductListElement = soup.find_all('div', {'cel_widget_id': re.compile('MAIN-SEARCH_RESULTS-\d+'),'class': 'template=SEARCH_RESULTS'})
for item in ProductListElement:
    # get the product information
```

The results returned in this way are exactly the exhibits displayed on the page.

Then I analyzed a single product project. Here I mainly extracted eight main pieces of information: "Title", "Product URL", "Price", "Solder", "Review", "Review Number", "Image URL", "Crawl Time".

See the code for the specific process. The main problem is how to quickly find the appropriate attributes. The referenced address href sometimes contains the basic web page(https://www.amazon.com), and sometimes it does not. You need to make a judgment.

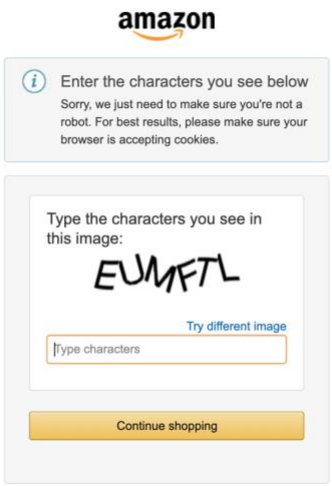
```
if product_url and product_url.get('href'):
    # if the url is relative, add the base url
    if product_url.get('href').startswith('/'):
        return BASE_URL+product_url.get('href').strip()
    return product_url.get('href').strip()
```

Here I put the code to extract the next page link.

```
# get next page url
def get_next_page_url(soup):
    # find the element that contains the next page url
    urlListElement = soup.find('a', {'class': 's-pagination-item s-pagination-next s-pagination-button s-pagination-separator'})
    if urlListElement and urlListElement.get('href'):
        # if the url is relative, add the base url
        if urlListElement.get('href').startswith('/'):
            next_url = BASE_URL+urlListElement.get('href')
        else:
            next_url = urlListElement.get('href')
        return next_url
    return None
```

3.2 Handle Errors

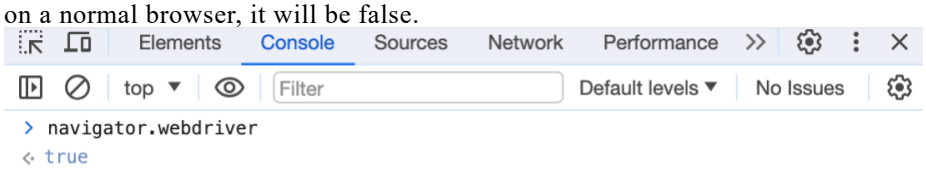
The first crawling attempt was not smooth, and it jumped directly to the page where a verification code was required.



Selenium is fairly easily detected, especially by all major anti-bot providers (Cloudflare, Akamai, etc).

Why?

1. Selenium, and most other major webdrivers set a browser variable (that websites can access) called navigator.webdriver to true. You can check this yourself by heading to your Google Chrome console and running console.log(navigator.webdriver). If you're on a normal browser, it will be false.



2. The User-Agent, typically all devices have what is called a "user agent", this refers to the device accessing the website. Selenium's User-Agent looks something like this:

Mozilla/5.0 (X11;Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) HeadlessChrome/59.0.3071.115 Safari/537.36. Did you catch that? HeadlessChrome is included, this is another route of detection.

So I added the User-Agent header to the parameters[1], and there was no problem for single page!

```
# avoid bot detection  - https://stackoverflow.com/questions/49565042/way-to-change-google-chrome-user-agent-in-selenium/49565254#49565254
options.add_argument("--user-agent=Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_2) AppleWebKit/537.17 (KHTML, like Gecko) Chrome/24.0.1309.0 Safa
```

3.3 Data Store

The data is stored in CSV format and stored separately according to the crawled date (subsequent improvements include classifying according to crawled keywords).

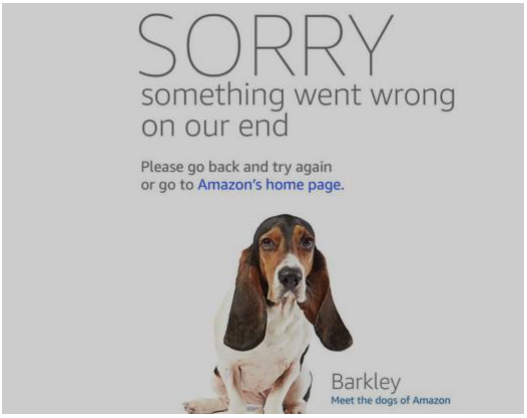
```python
def write_to_csv(data):
    # Create the output directory if it doesn't exist
    isExists = os.path.exists(current_dir+'/outputs')
    if not isExists:
        os.makedirs(current_dir+'/outputs')

    file_name = current_dir+"/outputs/{}-amazon-crawl.csv".format(datetime.today().strftime("%Y-%m-%d"))

    # Write the data to csv file
    with open(file_name, "w") as f:
        writer = csv.writer(f)
        for row in data:
            writer.writerow(row)

    return file_name
```

4 Challenges and outlook

In the whole process, page information extraction is the smoothest part, but the most difficult thing is that Amazon has a complete anti-crawler mechanism. I tried many methods to bypass the system monitoring.

4.1 Challenges

At the beginning, I planned to crawl all Kindle Store pages, which contained more than 100,000 items of product data. After starting as planned, nearly a hundred pages in, I encountered an error. It seems that Amazon has detected that I am using a crawler to control the browser.

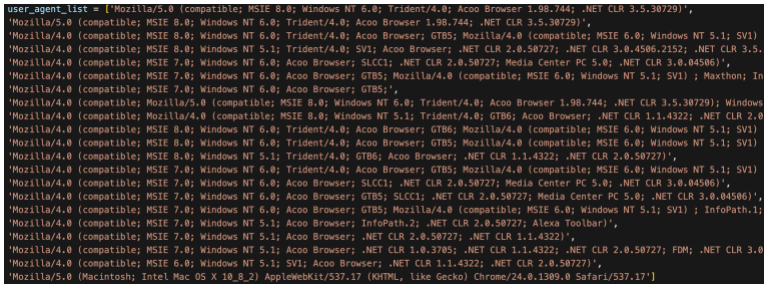


At first I thought I was not logged in, so Amazon redirected me to an error page. I wrote a login script and executed the crawler after the login script, but it didn't work.

```
driver = webdriver.Chrome(
# WEB_DRIVER_LOCATION,
    options=options)
driver.get(LOGIN_URL)

# login to Amazon
time.sleep(TIMEOUT)
login = driver.find_element(By.ID, 'ap_email')
login.send_keys(EMAIL)
login.send_keys(Keys.ENTER)

submit = driver.find_element(By.ID, 'ap_password')
submit.send_keys(PASSWORD)
submit.send_keys(Keys.ENTER)
```

Later I realized that every request header sent using selenium needs to add the login user token, otherwise the request will not automatically add user information after logging in. So I added cookies to the header parameters.

To fix this and make my behavior look more human-like, I added multiple user-agent headers for random selection and changed the sleep time to a random number. And change the crawled target page to Kindle's Today's Deals, so that the number of products will be controlled within 1,000.




```
# Timeout needed for Web page to render
time.sleep(random.randrange(1,TIMEOUT))
```

This time, no error page appeared, but the crawled results turned into garbled characters.

Title	Product URL	Price	Solder	Review	Review Number	Image URL	Crawl Time
H	h	$	J	4	1	h	2
a	t	1	e	.	2	t	0
n	t	.	a	3	,	t	2
g	p	9	n		7	p	4
	s	9	n	o	9	s	-
A	h	$	B	4	1	h	2
	t	1	a	.	,	t	0
W	t	.	r	4	4	t	2
o	p	9	b		9	p	4
m	s	9	a	o	8	s	-
T	h	$	M	4	1	h	2
h	t	0	a	.	2	t	0
e	t	.	n	5	,	t	2
	p	9	d		5	p	4
G	s	9	y	o	1	s	-
T	h	$	K	4	1	h	2
h	t	2	l	.	,	t	0
e	t	.	i	3	5	t	2
	p	9	p		8	p	4
C	s	9	h	o	7	s	-
W	h	$	B	4	5	h	2
h	t	0	a	.	6	t	0

...

I came up with another method. I first crawled the URLs that needed to be traversed, and

then I used the URL list as a parameter to crawl a single page, because at least the single page I tested was still normal.

```
urlList = ["https://www.amazon.com/s?k=Kindle+Store&i=digital-text&crid=35BK3MAMZOGFM&sprefix=%2Cdigital-text%2C79&ref=nb_sb_noss",
'https://www.amazon.com/s?k=Kindle+Store&i=digital-text&rh=n%3A133140011%2Cp_n_deal_type%3A23566064011&dc&page=2&qid=1706498884&rnid=1
'https://www.amazon.com/s?k=Kindle+Store&i=digital-text&rh=n%3A133140011%2Cp_n_deal_type%3A23566064011&dc&page=2&qid=1706498884&rnid=1
'https://www.amazon.com/s?k=Kindle+Store&i=digital-text&rh=n%3A133140011%2Cp_n_deal_type%3A23566064011&dc&page=3&qid=1706498890&rnid=1
'https://www.amazon.com/s?k=Kindle+Store&i=digital-text&rh=n%3A133140011%2Cp_n_deal_type%3A23566064011&dc&page=4&qid=1706498899&rnid=1
'https://www.amazon.com/s?k=Kindle+Store&i=digital-text&rh=n%3A133140011%2Cp_n_deal_type%3A23566064011&dc&page=5&qid=1706498910&rnid=1
'https://www.amazon.com/s?k=Kindle+Store&i=digital-text&rh=n%3A133140011%2Cp_n_deal_type%3A23566064011&dc&page=6&qid=1706498920&rnid=1
'https://www.amazon.com/s?k=Kindle+Store&i=digital-text&rh=n%3A133140011%2Cp_n_deal_type%3A23566064011&dc&page=7&qid=1706498930&rnid=1
'https://www.amazon.com/s?k=Kindle+Store&i=digital-text&rh=n%3A133140011%2Cp_n_deal_type%3A23566064011&dc&page=8&qid=1706498938&rnid=1
'https://www.amazon.com/s?k=Kindle+Store&i=digital-text&rh=n%3A133140011%2Cp_n_deal_type%3A23566064011&dc&page=9&qid=1706498947&rnid=1
'https://www.amazon.com/s?k=Kindle+Store&i=digital-text&rh=n%3A133140011%2Cp_n_deal_type%3A23566064011&dc&page=10&qid=1706498956&rnid=
'https://www.amazon.com/s?k=Kindle+Store&i=digital-text&rh=n%3A133140011%2Cp_n_deal_type%3A23566064011&dc&page=11&qid=1706498965&rnid=
'https://www.amazon.com/s?k=Kindle+Store&i=digital-text&rh=n%3A133140011%2Cp_n_deal_type%3A23566064011&dc&page=12&qid=1706498973&rnid=
'https://www.amazon.com/s?k=Kindle+Store&i=digital-text&rh=n%3A133140011%2Cp_n_deal_type%3A23566064011&dc&page=13&qid=1706498984&rnid=
'https://www.amazon.com/s?k=Kindle+Store&i=digital-text&rh=n%3A133140011%2Cp_n_deal_type%3A23566064011&dc&page=14&qid=1706498994&rnid=
'https://www.amazon.com/s?k=Kindle+Store&i=digital-text&rh=n%3A133140011%2Cp_n_deal_type%3A23566064011&dc&page=15&qid=1706499004&rnid=
'https://www.amazon.com/s?k=Kindle+Store&i=digital-text&rh=n%3A133140011%2Cp_n_deal_type%3A23566064011&dc&page=16&qid=1706499014&rnid=
'https://www.amazon.com/s?k=Kindle+Store&i=digital-text&rh=n%3A133140011%2Cp_n_deal_type%3A23566064011&dc&page=17&qid=1706499024&rnid=
'https://www.amazon.com/s?k=Kindle+Store&i=digital-text&rh=n%3A133140011%2Cp_n_deal_type%3A23566064011&dc&page=18&qid=1706499033&rnid=
'https://www.amazon.com/s?k=Kindle+Store&i=digital-text&rh=n%3A133140011%2Cp_n_deal_type%3A23566064011&dc&page=19&qid=1706499042&rnid=
'https://www.amazon.com/s?k=Kindle+Store&i=digital-text&rh=n%3A133140011%2Cp_n_deal_type%3A23566064011&dc&page=20&qid=1706499053&rnid=
'https://www.amazon.com/s?k=Kindle+Store&i=digital-text&rh=n%3A133140011%2Cp_n_deal_type%3A23566064011&dc&page=21&qid=1706499062&rnid=
'https://www.amazon.com/s?k=Kindle+Store&i=digital-text&rh=n%3A133140011%2Cp_n_deal_type%3A23566064011&dc&page=22&qid=1706499072&rnid=
'https://www.amazon.com/s?k=Kindle+Store&i=digital-text&rh=n%3A133140011%2Cp_n_deal_type%3A23566064011&dc&page=23&qid=1706499080&rnid=
'https://www.amazon.com/s?k=Kindle+Store&i=digital-text&rh=n%3A133140011%2Cp_n_deal_type%3A23566064011&dc&page=24&qid=1706499091&rnid=
```

This method turned out to be successful, but unfortunately I can't guarantee that it works every time. The disadvantage of this method is that it takes twice as long because you have to go through all the pages twice.

If you continue to encounter similar problems, it means that your IP and cookies have been banned. You can usually try to change the cookies. Of course, if you have money, it is more convenient to purchase an IP proxy.

So far, I have successfully crawled all items from the Amazon Kindle Store's Today’s Deals. But the whole process still leaves a lot to think about for me.

4.2 Outlook

1. The category I am currently crawling is Kindle Store, and I plan to try Best Sales next time, because it is obvious that this has more business opportunities.

2. If I can find sponsorship, I plan to use a paid IP proxy interface, so that I can crawl data more efficiently without having to worry about IP blocking.

3. The next step is to connect to the progress database. Because the crawled data is mainly structured data, it is very convenient to store the crawled data directly in the database.

References

[1] stackoverflow, Way to change Google Chrome user agent in Selenium.