Techniques for Web Data Extraction: Regular Expressions, XPath and RoadRunner

Brin Colnar (63190077), Mitja Kocjančič (63200482), Alen Kurtagić (63230480)

I. Introduction

In this report, we present our implementation of three advanced web data extraction techniques: regular expressions, XPath, and an automatic extraction algorithm inspired by RoadRunner. By implementing these methods in Python, our goal is to effectively extract structured data from various web pages. Each approach comes with unique strengths and challenges, and understanding their application will help you choose the best technique for your specific data extraction needs.

II. Pages

We were provided with pages from *Overstock.com* and *Rtvslo.si*. Additionally, we chose two specific pages, *BigBang.si* and *Craigslist.com*, as our primary test cases.

Figure 1 illustrates the Big Bang page, which includes details such as product type, product name, price, alternate pricing, and availability.

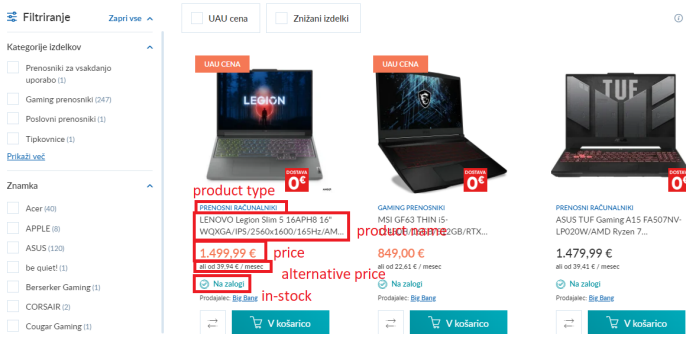

Figure 1. Page Big Bang.

Figure 2 displays the Craigslist page, where product data like price, name, and location are presented.

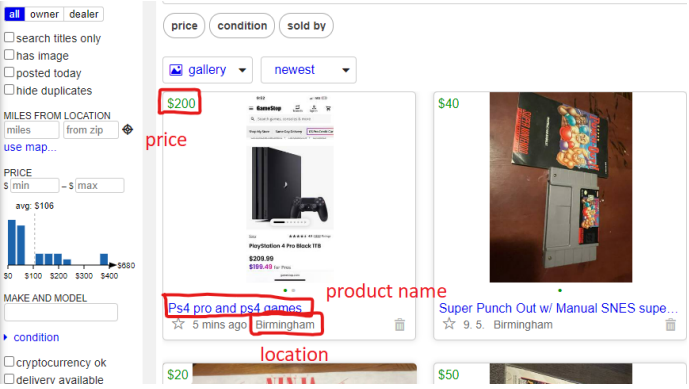

Figure 2. Page Craigslist.

Both sites offer products with essential data items such as names, prices, and availability, making them ideal examples for demonstrating the effectiveness of each data extraction method.

III. Regex Method

In this section, we demonstrate the use of regular expressions (regex) for data extraction across four different web pages. Each page required customized regex patterns to accurately identify and capture specific data items.

A. Overstock.com

The regex patterns for Overstock.com are detailed in Table I. These patterns target key data items such as titles, list prices, and savings.

Table I
Regex for Overstock.com.

Data item	Regex
Title	`<b>([\d]{1,2}-kt\.?.*?(?:\s*\([^)]+\))?)</b>`
ListPrice	`/<b>List Price:</b></td><td align="left" nowrap="nowrap"><s>([^<]+)</s></td>/`
Price	`<b>Price:</b></td><td align="left" nowrap="nowrap"><span class="bigred"><b>([^<]+)</b></span></td>`
Saving	`/<b>You Save:</b></td><td align="left" nowrap="nowrap"><span class="littleorange">(\$[\d,.]+) \(([\d]+%)\)</span></td>/`
Content	`</b></a> \s*<table>.*?<td valign="top"><span class="normal">([\s\S]*?<a href="[^"]+"><span class="tiny"><b>Click here to purchase \.</b></span></a>)'`

B. Rtvslo.si

The regex patterns for extracting data from the Rtvslo.si page are listed in Table II, focusing on elements like the author's name, publication date, and article content.

Table II
Regex for Rtvslo.si.

Author	`<div class="author-name">(.*?)</div>`
PublishTime	`<div class="publish-meta">\s*([^<]+)`
Title	`<h1[^>]*>(.*?)<\/h1>`
Subtitle	`<div class="subtitle">(.*?)<\/div>`
Lead	`<p class="lead">(.*?)<\/p>`
Content	`<div class="article-body">.*?(<p.*?>.*?<\/p>)+.*?<\/div>`

C. BigBang.si

The regex patterns for BigBang.si are provided in Table III, focusing on key data items like product category, name, and price.

Table III
REGEX FOR BIGBANG.SI

Data item	Regex
Category	`<div .*?class="cp-category".*?>([^<]+)<\/a><\/div>`
Name	`<h2 .*?class="cp-title".*?>([^<]+)<\/a><\/h2>`
Price	`<div .*?class="cp-current-price".*?>([^<]+)<\/div>`
Alternative Price	ali od (+,2 €)
In stock	`<div\s+[^>]*class="available-qty-btn"[^>]*>\s*<span[^>]*>(?:<strong>)?(.*?)(?:<\/strong>)?<\/span>`

D. Craigslist.com

Table IV shows the regex patterns for extracting data from Craigslist.com, including product price, name, and location.

Table IV
REGEX FOR CRAIGLIST.COM

Data item	Regex
Name	`<a .*?class="cl-app-anchor text-only posting-title".*?>([^<]+)<\/span><\/a>`
Price	`<li.*?title="' + re.escape(product) + '".*?><span class="priceinfo">(.*?)<\/span>`
Location	`<li.*?title="' + re.escape(product) + '".*?><div class="meta">(.*?)<\/div>`

IV. XPATH METHOD

In this section, we outline the use of XPath expressions for data extraction across four web pages. Each page required specific XPath expressions to identify and retrieve relevant data items.

A. Overstock.com

Table V presents the XPath expressions used for extracting data from Overstock.com. The expressions are carefully constructed to identify elements like product titles, list prices, and discounts.

B. Rtvslo.si

The XPath expressions for the Rtvslo.si page are shown in Table VI. These patterns focus on key elements such as the author's name, publication date, and content.

Table V
XPATH FOR OVERSTOCK.COM

Data item	Regex
Title	`/html/body/table[2]/tbody/tr[1]/td[5]/table/tbody/tr[2]/td/table/tbody/tr/td/table/tbody/tr/td[2]/a/b/text()`
ListPrice	`/html/body/table[2]/tbody/tr[1]/td[5]/table/tbody/tr[2]/td/table/tbody/tr/td/table/tbody/tr/td[2]/table/tbody/tr/td[1]/table/tbody/tr[1]/td[2]/s/text()`
Price	`/html/body/table[2]/tbody/tr[1]/td[5]/table/tbody/tr[2]/td/table/tbody/tr/td/table/tbody/tr/td[2]/table/tbody/tr/td[1]/table/tbody/tr[2]/td[2]/span/b/text()`
Saving	`/html/body/table[2]/tbody/tr[1]/td[5]/table/tbody/tr[2]/td/table/tbody/tr/td/table/tbody/tr/td[2]/table/tbody/tr/td[1]/table/tbody/tr[3]/td[2]/span/text()`
Content	`/html/body/table[2]/tbody/tr[1]/td[5]/table/tbody/tr[2]/td/table/tbody/tr/td/table/tbody/tr/td[2]/table/tbody/tr/td[2]/span/text()`

Table VI
XPATH FOR RTVSLO.SI

Author	`//div[@class='article-meta']/div[@class='author']/div[@class='author-name']/text()`
PublishTime	`//div[@class='article-meta']/div[@class='publish-meta']/text()`
Title	`//header[@class='article-header']/h1/text()`
Subtitle	`//header[@class='article-header']/div[@class='subtitle']/text()`
Lead	`//header[@class='article-header']/p[@class='lead']/text()`
Content	`//div[@class='article-body']/article[@class='article']/p/text() | //div[@class='article-body']/article[@class='article']/p/strong/text()`

C. BigBang.si

Table VII contains the XPath expressions used for extracting data from BigBang.si, including product category, name, and price.

D. Craigslist.com

The XPath expressions for Craigslist.com are detailed in Table VIII, covering elements such as price, product name, and location. Because price was not constant a complex if statment was needed to be envolved as well as 2 regex calls, 1. would extraxt the product name and the second would extraxt the price based on that name and if no match on price based on name was found, the price would not be extraxted

Table VII
XPath for bigbang.si

Type	/html/body/div[1]/div/div[4] /div[2]/div/div[2]/div[3]/div[1] /article/div[2]/div[1] /div[2]/a/text()
Name	/html/body/div[1]/div /div[4]/div[2]/div/div[2]/div[3] /div[1]/article /div[2]/div[1] /h2/a/text() /div[@class='publish-meta']/text()
Price	/html/body/div[1]/div/div[4] /div[2]/div/div[2]/div[3] /div[1]/article /div[2]/div[2]/div[1]/text()
Alternative price	/html/body/div[1]/div/div[4] /div[2]/div/div[2]/div[3]/div[1] /article/div[2]/div[2] /div[2]/p/strong/text()
In stock	/html/body/div[1]/div /div[4]/div[2]/div /div[2]/div[3]/div[1] []{}/article/div[2] /div[3]/div[1]/span/text() | /html/body/div[1]/div/div[4] /div[2]/div/div[2]/div[3] /article/div[2]/div[1] /div[3]/div[1]/span/strong/text()

Table VIII
XPath for Craiglist.com

Name	/html/body/div[1]/main/div[1] /div[5]/ol/li/div/a/span/text() /div[@class='author'] /div[@class='author-name']/text()
Price	/html/body/div[1]/main/div[1] /div[5]/ol/li["+str(i)+"] /div/span/text()
Location	/html/body/div[1]/main/div[1] /div[5]/ol/li["+str(i)+"] /div/div[2]/text()

V. Roadrunner

As for automatic web extraction, we decided to go with Roadrunner algorithm [1].

We haven't fully implemented all the features like finding optionals. We had to heavily simplify and restructure the webpages for the algorithm to work as the default web pages were far too complex and the algorithm failed to finish.

The algorithm works as shown in pseudocode below. Two pointers are simultaneously moved through 2 pages of the same type. Both pages were tokenized, such that any given token is either a string or a tag.

In the case that tokens match they are simply added to the wrapper. In the case of a mismatch there are several possibilities.

The simple case is when both tokens are strings and we identified a data field. Here token $PCDATA$ is added to wrapper.

If both tokens are tags, we try to identify iterator pattern, or square as its called in the original Roadrunner paper. We investigate the two possibilities of square either being the sample page or in the wrapper page. Terminal terminal tag is found by moving the cursor backwards by 1 position. This is then used as a basis to determine if this page contains square.

After finding the page, we generate a regex for identifying square by performing a backwards matching search. In case this search succeeds, we add square regex to wrapper.

Missing from our implementation is a search for optionals. We attempted to implement this , but it worked only on very simple examples.

Please see appendix for results.

Algorithm 1: Roadrunner pseudocode

Result: Generate a generalized wrapper from two tokenized web pages

```
Initialize: wrapper from first page, sample from second page
Set wrapper_pointer to 0
Set sample_pointer to 0
Initialize wrapper_ as an empty string
while have content to process do
    if both tokens are tags then
        Print "Tag mismatch!"
        Determine last token of square as iterator_terminal_tag
        if iterator_terminal_tag matches cur_wrapper_token then
            Try matching a square in the wrapper starting at wrapper_pointer
            if matched then
                Find and append the square regex pattern with repetition to wrapper_
            else
                Mark element as optionally repeating and handle complexity
            end
        else if iterator_terminal_tag matches cur_sample_token then
            Try matching a square in the sample starting at sample_pointer
            if matched then
                Find and append the square regex pattern as optional to wrapper_
            else
                Mark element as optionally repeating and handle complexity
            end
        end
    else if one token is a tag and the other is text then
        Handle mixed content mismatch cases
    end
    else
        Append 'PCDATA' to wrapper_
    end
    Increment wrapper_pointer
    Increment sample_pointer
    Print the current state of wrapper_
end
```

Appendix

```
<body>
    <article>
        <div>
            <figure><a><img></a></figure>
        </div>
        <div>
            <div>#PCDATA</div>
            <div><a>#PCDATA</a></div>
            <h2><a>#PCDATA</a></h2>
        </div>
        <div>
            <div>#PCDATA</div>
            <p>ali od<strong>#PCDATA</strong>/ mesec</p>
        </div>
        <div>
            <div><strong>Na zalogi</strong></div>
        </div>
        <div><span>Prodajalec:<a>Big Bang</a></span></div>
    </article>
</body>
```

Figure 3. Wrapper for simplified BigBang page

```
<body>
    <ul>
        (<li>
            <p>#PCDATA</p>
            <p>#PCDATA</p>
            <p>#PCDATA</p>
            <p>#PCDATA</p>
        </li>)+
    </ul>
</body>
```

Figure 4. Wrapper for simplified Overstock page

```
<body>
    <article>
        <header>
            <h1>#PCDATA</h1>
            <h2>#PCDATA</h2>
            <p><strong>Author:</strong> Miha Merljak</p>
            <p><strong>Published:</strong> 28. december 2018 ob
        </header>
        <section>
            <p>#PCDATA</p>
        </section>
        <section>
            <p>#PCDATA</p>
            <p>#PCDATA</p>
            <p>#PCDATA</p>
            <p>#PCDATA</p>
        </section>
        <footer>
            <p><strong>Source:</strong> RTVSLO.si</p>
        </footer>
    </article>
</body>
```

Figure 5. Wrapper for Simplified RTVSLO

References

[1] V. Crescenzi, G. Mecca, and P. Merialdo, "Roadrunner: Towards automatic data extraction from large web sites," in *Proceedings of the 27th International Conference on Very Large Data Bases*, ser. VLDB '01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, p. 109–118.