Web Information Extraction and Retrieval Programming Assignment 2

Martin Arsovski, Maja Nikoloska, Emil Batakliev

April 2021

1 Introduction

In this report we present our second assignment for the subject Web Information Extraction and Retrieval. The goal of this programming assignment was to implement three different approaches to structured network data extraction:

- Using regular expressions
- Using XPath
- Using a Webstemmer-like implementation

We had to extract data from four web pages (two Overstock sites and two Rtvslo.si sites), which were already given in the instructions and two pages of our choice.

2 Selected pages

For additional pages on which we performed the data extraction, we have selected two pages from the "Nepremicnine" site. We extract the following data on each data item: district, title, image URL, year of construction, description, size, price and agency. All the data is presented in the image below (Figure 1).

Figure 1: Data for each data item

3 Regular expressions

There is a library in Python that allows the use of regular expressions called "Regular expression" or "RegEx" denoted (and imported) as "re". In order to extract the data from the sites we use the following regular expressions:

1. For the site Overstock:

Title:

```
"PROD_ID[^>]*><b>([^<]*)</b>"
```

List price:

```
"List Price:</b></td><td align=\"left\" nowrap=\"nowrap\"><s>([^<]*)</s>"
```

Price:

```
"Price:</b></td><td align=\"left\" nowrap=\"nowrap\"><span
class=\"bigred\"><b>([^<]*)</b>"
```

Saving and saving percent:

```
"You Save:</b></td><td align=\"left\" nowrap=\"nowrap\"><span
class=\"littleorange\">([^<]*) ([^<]*)</span>"
```

Content:

```
"span class=\"normal\">([^<]*)<br>"
```

2. For the site Rtvslo.si:

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

3. For the site Nepremicnine:

```
Price: "<span class=\"cena\">([^<]*)<"
Agency: "<span class=\"agencija\">([^<]*)</span>"
Size: "<span class=\"velikost\" lang=\"sl\">([^<]*)</span>"
District: "<span class=\"title\">([^<]*)</span>"
Description: "<div class=\"kratek\"
(itemprop|itemqrop)=\"description\">([^<]*)</div>"
Year of construction: "<span class=\"atribut leto\">Leto:
<strong>([^<]*)</strong></span>"
Title: "<span class=\"vrsta\">([^<]*)</span>"
Image url: "<img class=\"(lazyload|lazyloaded| lazyload|
lazyloaded| ls-is-cached lazyloaded)\" data-src=\"([^\"]*)\""
```

4 Xpath

The library in Python that allows the use of Xpath is "lxml". In order to extract data from our sites we use the following Xpath expressions:

1. For the site Overstock:
Title:

```
"//table[2]//tr[1]/td[5]/table//tr[2]/td/table//tr/td/table//
tr[@bgcolor]/td[2]/a//text()"
```

List price:

```
"//table[2]//tr[1]/td[5]/table//tr[2]/td/table//tr/td/table//tr
[@bgcolor]/td[2]/table//table//tr[1]/td[2]//text()"
```

Price:

```
"//table[2]//tr[1]/td[5]/table//tr[2]/td/table//tr/td/table//
tr[@bgcolor]/td[2]/table//table//tr[2]/td[2]//text()"
```

Saving and saving percent:

```
"//table[2]//tr[1]/td[5]/table//tr[2]/td/table//tr/td/table//tr
[@bgcolor]/td[2]/table//table//tr[3]/td[2]//text()"
```

Content:

```
"//table[2]//tr[1]/td[5]/table//tr[2]/td/table//tr/td/table//tr
[@bgcolor]/td[2]/table/tbody/tr/td[2]//text()"
```

2. For the site Rtvslo.si:

```
Author:  "//div[@class='author-name']/text()"
Published time and Published place:
"//div[@class='publish-meta']/text()"
Title: "//h1/text()"
Subtitle:  "//div[@class='subtitle']/text()"
Lead:  "//p[@class='lead']/text()"
Content:  "//article[@class='article']/p//text()"
```

3. For the site Nepremicnine:

```
Price: "//span[@class='cena']/text()
    | //span[@class='cena' and not(text())]"
Agency: "//span[@class='agencija']/text()"
Size: "//span[@class='velikost']/text()
    | //span[@class='velikost' and not(text())]"
District: "//span[@class='title']/text()"
Description: "//div[@class='kratek']/text()"
Year of construction: "//span[@class='atribut leto']/strong/text()"
Title: "//span[@class='vrsta']/text()"
```

```
Image url: "//img[@class='lazyload']/@data-src
    | //img[@class='lazyload']/@data-src
    | //img[@class='lazyloaded']/@data-src
    | //img[@class=' lazyloaded']/@data-src
    | //img[@class=' ls-is-cached lazyloaded']/@data-src"
```

5 Webstemmer-like implementation

For the Webstemmer-like implementation we have taken a different approach at gathering data from the first two implementations we mentioned earlier. Here, the program has to figure out which parts of the website are actually useful and later save them, rather than taking the exact same elements for each new web page on a particular site.

In the beginning we start by reading two similar websites. We extract all the elements within those pages and save the element tag names and the whole elements within two lists.

The next step is extracting the similar elements using a layout pattern. We have found that creating the layout pattern and extracting the elements simultaneously works faster and better, which is why we chose that approach. If there are any differences in the files, they are quickly resolved by using the "compare PageElements(page1Elemenets, page2Elemenets, i, j)" function.

Lastly, we have to compare the differences, or rather, the similarities between each element. We go through each element and use the "similar(a, b)" function, to check to see how much those elements differ. If the elements similarity is higher than half, we assume that we have stumbled upon repetitive content and we skip it. Once we have gathered all the different elements, the Webstemmer-like implementation reaches its end.

The final elements are stored within the two lists "diffElements1" and "diffElements2" for both pages respectively. In order to avoid confusion, we have decided to not print the elements. For testing purposes and a clearer understanding of what each two similar pages print, we suggest commenting two lines of the "implementationC(pages)" function, for clearer comparison between the two pages (instead of printing 6 pages).

6 Conclusion

In this programming assignment we saw that there are different approaches to data extraction. In our opinion, the most useful is automatic extraction, for which there are available algorithms and implementations depending on our needs. We believe this is the best type of algorithm because you can use it on any template. There are of course certain cases where we do not want that much information from pages, but rather some specific information which is where the other two implementations might be better for the task.