Automated Web Scraping APIs

Daan Krijnen
Leiden University
naad.daan@gmail.com

Rinus Bot
Leiden University
RinusBot@gmail.com

Georgios Lampropoulos
Leiden University
g.lampropoulos1987@gmail.com

ABSTRACT

Web scraping is a very useful web technique to gather and structure different types of data from the internet. We analyse how different fields of webscraping developed since the beginning of the internet. It will become clear how the rise of scraping bots was driven by the publication of scraping specific library's and slowed down by rising tensions in the legal field and the alternative option of official API's. The operating principles of webscraping are explained in detail based on a division in to three categories: synctactic web scraping, semantic web scraping and computer vision webpage analysis. The strengths and weaknesses of two types of webscraping will be discussed: web scraping by coding and web scraping by creating an automated API with the help of a visual interface (Kimono and Import.io). The typical applications of data mining, research, marketing, company competion, personal tools and data combination will be discussed in more detail. At the end of the paper we describe the development of a quick application in order for people to get started with this technology.

Author Keywords

web scraping, APIs, css, semantic web, computer vision

INTRODUCTION

Purpose

To manage the information explosion which the dawn of the internet has brought upon us, it is necessary to handle knowledge (data) in an increasingly efficient way. The purpose of web scraping in the most general sense, is to order knowledge available through the internet, in structures that allow for more convenient ways of gaining understanding or practical advantage from this knowledge.

Web scraping is closely related to, but should not be confused with, web indexing (also known as web crawling or web spidering) which distinguishes itself from web scraping by the fact that the extracted data is more generic and directed towards the internet in general. For example, web indexing is employed by companies that offer web navigation services such as Google and Yahoo.

Context and History

Figure 1 will be used as a guidance for discussing the context and history of web scraping.

The big colored horizontal bars represent developments that happen over multiple years and do not have a clear starting point. The named events surrounded by circles represent certain milestones linked to a specific horizontal bar by color.

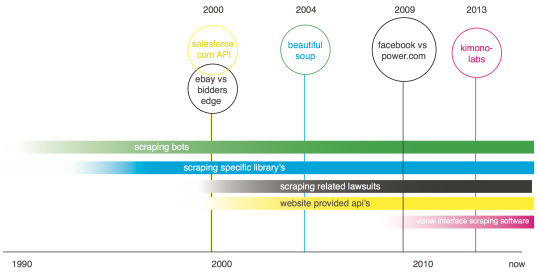

Figure 1: Timeline of webscraping history

scraping bots

A web scraping bot is basically just a piece of software that automatically scrapes certain targets at specific times. Such a program could in theory be written in any programming language. The rise of the scraping bots coincided with the rise of the Internet and, as shall described in coming paragraphs, has had a large influence on other developments within the webscraping history.

scraping specific libraries

The internet itself was a big factor in the flourishing of the open source community. This community has been responsible for the bulk of scraping specific libraries that have been made available over the years for a wide variety of programming languages. One specific library, named Beautiful Soup, is mentioned as a milestone in figure 1 Beautiful soup is a library designed for Python and released in the year 2004. It is widely considered as the most sophisticated and advanced library for the purpose of webscraping.

Scraping related lawsuits

The concept of webscraping has been the subjected to quite some lawsuits for two major reasons:

1. Web scraping often involves collecting/ordering material of which the deployment is part of the data sources core business

2. Automated web scraping can be taxing on the servers of the targeted websites.

Proportionally to the insight that data is the new gold, the first mentioned reason is increasingly functioning as the main motivator behind big lawsuits.

To fully comprehend this legal history it is important to first look at a non-internet-related lawsuit that was filed in America in 1991. The case is generally referred to as: Feist v. Rural [1]. In the case appealed, Feist (a telephone number aggregator) had copied information from Rurals telephone listings to include in its own, after Rural had refused to license the information. Feist was caught copying the numbers because Rural had included a small number of phony entries in its list to detect copying. The court ruled that information contained in Rurals phone directory was not copyrightable and therefore no infringement existed [2]. By this decision the court established that information alone, without a minimum amount of creativity cannot be protected by copyright.

This case clearly relates to the debate of the legality of web scraping. As we go through the 90s with this court ruling weighing in on the discussion, the rise of the internet is not yet resulting in filing any major cases against web scraping activities.

In the year 2000 this changes when the first big court case concerning web scraping arrived in the form of: Ebay v. Bidders Edge (BE). BE was an aggregator of auction lists and included a substantial amount of auctions that originated from Ebay, on its site. Ebay wanted BE to stop doing so, allegedly because the vast amount of queries that BE requested where a burden on Ebays network. When technical measures to prevent the web scraping from happening failed to be effective, Ebay suid BE.

The court judged in favour of Ebay with the following reasoning: Even though BEs use of ebays bandwidth was rather small (1.5%), it did potentially harm Ebay since other companies might follow BEs example. This described harm falls under the legal definition of trespassing to chattels. Interestingly later that year Ebay launched an API on November 20th 2000.

Through the 00s there have been multiple similar cases with differing outcomes. In case the lawsuit was in favour of the prosecutor it was based on the application of the trespassing of chattels as described earlier

2009 brought a change to that legal tendency with the case: Facebook v. Power.com. Power.com offered a website that enabled its users to aggregate data about themselves that is otherwise spread across various social networking sites and services. Facebook sued power.com with the accusation that it had copied copyrighted material. Facebook does not own the copyright of its users profile data (which was what power.com effectively used in its service), but argued that it owned the copyright to the websites framework surrounding the users data. According to facebook, the power.com scraper operated in a manner that it required to copy the entire Web page in order to extract the users data and thus made use of copyrighted material.

Facebook won this court case with the described reasoning. The shocking thing about this is that the process of web scraping itself was basically criminalized by this court ruling. [3]

Website provided APIs

Web APIs started to appear in the early 00s, the first one being the salesforce.com API. They are of great relevance when discussing the topic of webscraping, since they offer developers an alternative way of gathering data provided from websites. Web APIs were partly developed for this specific reason [4]. Not only do they provide an alternative way to gathering information on the web, they also provide easier and more consistent access to this information.

Visual interfaced scraping software

In the early 10s the first scraping software with visual interfaces were developed. The launch op Kimino Labs in late 2013 is visualised as a milestone in figure 1 since it was the first scraping software with a visual interface that was so easily available and usable. Though it is only just released and still in beta it is already heavily used (15.000 users as of 03-04-2014) [5]. It is interesting to see what this simplification of webscraping will bring us in the future, and how these platforms might compete with official APIs. What is certain is that this new development democratizes the know-how for the business of web scraping in an unprecedented way, which is a good thing.

OPERATING PRINCIPLES

To understand the concept of webscraping, including the visual interfaced web services, it is important to comprehend the technological operating principles of the technology. Web scraping is done by using specific methods on what data to gather and aggregate. In order to achieve this, a well understanding of programming, web technologies such as HTML, and the structure of data on the web (e.g., the Document Object Model (DOM)) is required. This required knowledge and understanding is reduced by providing a web scraping API.

Automated web scraping can be categorized in 3 main techniques which are widely used by web scraping software[6].

- Syntactic Web Scraping
- Semantic Web Scraping
- Computer vision web-page analyzing

Syntactic Web Scraping

Syntactic web scraping extracts information from the website structure by parsing HTML, CSS and other typical web languages. To do so several techniques are used:

- Content Style Sheet selectors: Content StyleSheets define the visual properties of HTML elements. These visual properties are mapped to elements through the use of CSS

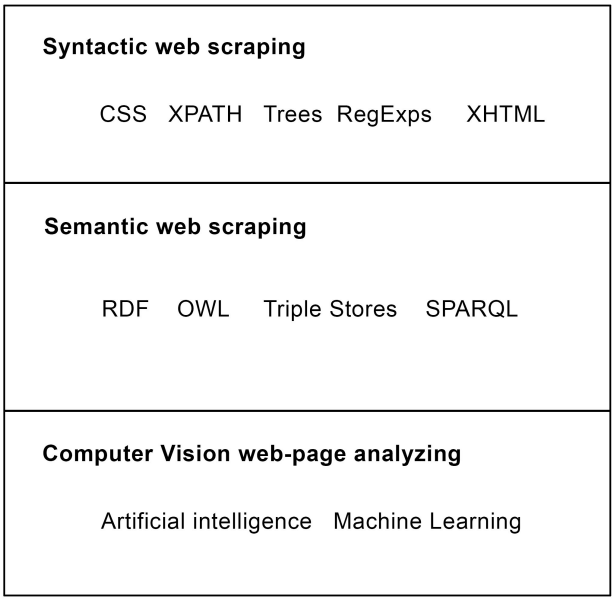

Figure 2: Types of web scraping techniques

selectors, defined through a specific language. Therefore, CSS is one technology that serves to select and extract data.

- XPath selectors. Similarly to CSS selectors, the XML Path Language is a different language for HTML node selection.

- URI patterns. URI patterns allow to select web resources according to a regular expression that is applied on the resources URI. While XPath or CSS selectors are able to select an element at document level, URI patterns allow selecting documents, i.e. resources representations, according to the resources URI.

- Visual selectors. Visual information can be used to select nodes. HTML nodes are rendered with a set of visual properties given by the used browser. It is common that human users prefer uniform web designs. Web designers thus make elements of a same kind to be rendered with similar visual properties to help identification. A visual selector is a condition that combines several visual properties of an element to identify the elements class.

Semantic Web Scraping

As the semantic web grew, ways and frameworks of handling the semantic data, were developed.

Consequently, the data extracted by the syntactic web scraping can be mapped to semantic web resources, for better representation and manipulation. To do this, several frameworks can be used, like the Resource Description Framework(RDF) and the Web Ontology Language (OWL). The failure of the semantic web, so far, makes this technique nonpreferable by the most applications.

Computer Vision Web page Analysis

Finally, machine learning and computer vision techniques can be used to identify and extract information from web pages by interpreting pages visually as a human being might and then associate these to css selectors [7]. One example of this is diffbot[8].

STRENGTHS AND WEAKNESSES

The previous sections mostly discussed webscraping by coding, however, from here on we are focusing more on the subject of automated APIs scraping with visual interface for webscraping. The two main webservices that provide this technology are Kimono Labs and Import.io. Both require the user to create an account to use the service. In this section we will discuss the strengths and weaknesses of these automated services versus coded, or manual, webscraping. Before we can juxtapose coding versus automated, we must distinguish the general strengths and weaknesses of the technology. As the uses of the technology range from project to project, as will become clear later on in this paper, there are most likely more points to be defined.

Scraping by Coding

First of all the strengths of web scraping in general. It offers you the chance to get any data you want in a structured way. This structure can be can be based on the syntactic, computer vision or semantic abstraction technologies discussed before. A great strength of web scraping is the fact that it enables the user to structure the data in the way it suits best for the underlying project. In other words, you are in full control of data that you do not control. It is this data that is another strength of scraping. It is based on the data that is build to be shown to the viewers of the website. This means that it has the highest priority of the web developer to keep this data up to date, and thus provides the scraper with high quality, up to date data. This point also explains why it is sometimes better to scrape data instead of using a provided public API. The core functionality of most web services lies not in maintaining an API, but in maintaining the html front end their (paying) users see. Linked to the points above is also the fact that there are no fixed rate limits on data queries, and usage is not logged in a way other than that of a normal web user. Some of the weaknesses of scraping web data are the following. First, an update to the layout of the website, or merely the renaming of certain elements in the CSS could lead to a malfunctioning scraper. Secondly you will need programming skills to write a scraper, and you need a server to run the scraper. Lastly there is never any documentation on how to scrape the website you want to scrape. Each case is different, and requires a custom built scraper.

Scraping with Visual Interfaced Services

To compare the above strengths and weaknesses in comparison with automated APIs such as Kimono or Import.io, it is easiest to start with the weaknesses discussed above, and explain the strengths of automated web scrapers. To use Kimono or Import.io the user does not need any programming experience. The visual CSS element selection works quite well in both services and offers the user to create a complete working API without any programming knowledge. It also

enables users to create a web scraper without the need of a server, it all runs online. A big advantage is the fact that the resulting API uses standard API structures in multiple formats, so that the data can be shared easily with other developers. An additional service you get from Kimono and Import.io is a notification on failed updates of the API. Meaning you do not need to actively check if your application is still working. The fact that the user is provided with these services also means that you rely on these third party services to provide you with your data feed. It is very important to keep in mind that at this moment both of the services discussed, Kimono and Import.io are in beta status. This means that the usage, as well as the pricing (still free for now), can change in the future. Also there can be some rate limits, something a manual scraper does not encounter. Lastly it is important to realise that if you have the skills required, building your own scraper will always be more customizable than any of these services. If you for instance need to get calendar data from a website, it is usually easy to use Kimono or Import.io. It requires you to identify the elements that contain the date, time and description, and usually the service does the rest for you. If you need to scrape data that is not easily identified via CSS, for instance script generated textboxes that change based on the context of the location of your mouse position, it might be better to write your own scraper.

TYPICAL APPLICATIONS

As could be seen in the section above, there are many aspects that can be considered when using a web scraper. This can be extrapolated to the applications of scraping. Also, the vague legality of building a project based on scraped data makes it difficult to distinguish projects that are relying on scraping technologies. We can, however, give an overview of the most typical fields the technology is used in. With some of these fields we will give short examples of how such a project could look like. The fields we consider being typical for web scraping are:

- Data mining
- Research
- Marketing
- Company competition
- Personal tools
- Data combination

First, data mining, is a field that is very broad. One of the main sectors in which data mining is often used is in the Spam business. The usage of web scrapers to aggregate email addresses to distribute unwanted emails is a widely used, but hard to proof method.

Second is research, which could be research by any party. For instance a university doing research on the use of Twitter. In that case the quality of the data must be as good as it gets, which means not using the API. Writing a scraping bot for Twitter, or using any of the services mentioned above, will provide the research institute with the same data as the users they are investigating are using.

In marketing applications closely related to the first and second usage examples can be thought of. For example researching the development of your brand awareness campaign on multiple social media, and logging each and every activity of that for future usage.

Furthermore it can be used by companies to keep track of their competitions activity. For instance by monitoring the prices they ask for their products. This can be easily projected on the next point in the list, where you can imagine that individuals will make use of scraping technologies in order to keep track of the best deals online.

Lastly scraping can be used to structure similar data from different sources to combine them in a single source. For instance the combination of all the publications on crowdfunding websites could be joined together to keep track of the developments in that field.

Many of the possible applications are only limited by the imagination of the user/developer. In the next section we will exemplify the limits of imagination in respect to web scraping.

SURPRISING APPLICATIONS

Not mentioned above, the domain of journalism, or more in particular data journalism can be both as a typical application that produces many surprising applications. It is important to keep in mind that in any case the data is merely a source to build a story on top of. For instance the job applications on many websites were tracked and linked to a list of many companies, to find abstract from the resulting linked data if any of the companies was growing. Another example is using Import.io to combine the results of the Indian elections, to power real time data visualizations as the results came in. As India did provide a website where the data could be seen, a web scraper built in Import.io made it possible to combine this data and offer it in a structured way to create insights the government was not able to provide to journalists [9].

GETTING STARTED

This section will discuss briefly how to create a working API with one of the discussed services, namely Kimono. It will guide you through the selection of the data, the structuring of the data, the options in Kimono on calling the data, and how to create a simple visualization from the data. For this section we will create a small API that gathers the delay data from the major railway stations in the Netherlands.

Kimono interface

Kimono Labs offers a bookmarklet that starts the entire process of creating your first API. With the bookmarklet placed in reachable place, we head to the website that requires scraping. In this case the mobile website of the NS, `m.ns.nl` offers a simpler view that could provide us with easier selectable data. In the departure section we select Utrecht Cen-

(a) delay selection (b) add data type

(c) reject incorrect data (d) name your API

Figure 3: API creation process

```
{
  "delay": "+ 1 min",
  "destination": {
    "text": "Breukelen",
    "href": "http://m.ns.nl/actvertrektijden.action?from=BKL"
  }
```

Figure 4: Resulting JSON endpoint

traal, which leads us to `http://m.ns.nl/actvertrektijden.action?from=UT`. At this point the kimonify bookmarklet should be pressed, bringing us to the main Kimono interface, which they call the "Extractor View".

The next step is selecting the data. As we are interested in creating an API with delay data, we click on one of the delays times, for instance

`+6`. Kimono will then automatically offer you similar data based on the CSS selectors that were used to create this selection, select the other "+ [n] min" texts as seen in figure 3a. Discard any bad data and select the correct data.

In Kimono it is possible to add different data types. So including the destination of the delayed train is an option as seen in fig 3b and 3c. After the correct data is selected the API is ready. After clicking "done" Kimono will ask you for a name, and the update frequency, which is in our case "on Demand" as seen in figure 3d. This results in a fully functional API for Utrecht Central Station delays, that can be called on demand.

Resulting End Points

In your Kimono Account the endpoints of your new API can be tested in the browser. A Json response would look like the one in figure 4.

URL Parameters

An interesting option in Kimono is the fact that it detects wether the URL has any customizable parameters that lead to similar pages that might have interesting data. To call our API we normally call `http://www.kimonolabs.com/api/{API_ID}?apikey={YOUR_API_KEY}`, which results in a response as seen before. However, adding the URL parameter `from=ASD` to the URL will lead to a response that is from the Amsterdam Central Station departure times.

visualization

Using OpenFrameWorks we created a small visualization that loops through the different stations using the URL Parameters with a html GET request. The returned JSON is then parsed and saved into an array and further processed to compute the mean delay times of each station(figure 5).

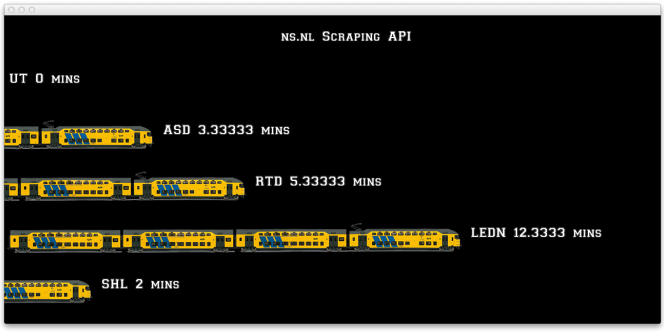

Figure 5: Screenshot of our developed application

FINAL THOUGHTS

Being able to scrape the web is very empowering and it is quite interesting to see how the use of this web technology will develop in the years to come. The democratization of web web scraping through interfaces like Kimono Labs and Import.io are very exciting and positive developments. The legal pressures could endanger the practice of web scraping, but raising public awareness around this issue might positively influence the debate.

REFERENCES

1. Wikipedia Feist v. Rural. `https://en.wikipedia.org/wiki/Feist_v._Rural`.
2. `http://www.law.cornell.edu/copyright/cases/499_US_340.htm`.
3. Rami Essaid. Is web scraping legal? `http://www.distilnetworks.com/is-web-scraping-illegal-depends-on-what-the-meaning-of-the-`
4. The History Of API's. `http://apievangelist.com/2012/12/20/history-of-apis/`.
5. Kyle Vanhemert. This Simple Data-Scraping Tool Could Change How Apps Are Made. `http://www.wired.com/2014/03/kimono/`.
6. Carlos A. Iglesias Mercedes Garijo Jose Ignacio Fernandez-Villamor, Jacobo Blasco-Garcia. A Semantic Scraping Model for Web Resources, Applying Linked Data to Web Page Screen Scraping.
7. Muntasir Mashuq MichelZiyan Zhou. Web Content Extraction Through Machine Learning.
8. Diffbot: Extract content from standard page types: articles/blog posts, front pages, image and product pages. `http://www.diffbot.com/`.
9. Alex Gimson. This Just In: A Data Journalism Webinar with Bea Schofield. `http://blog.import.io/post/this-just-in-a-data-journalism-webinar-with-bea-schofield`.