Web-Scraping:

How Charlotte Is A Life Saver Again

Fayang Pan

November 19, 2013

Contents

List of Figures

Chapter 1

An Evaluation of Web-Scraping

Introduction

Web-scraping is a way to collect existing information from the Internet.(KEEP AN EYE ON THE OPENING SENTENCE) It is a nascent, powerful, but disputable subject in the arena of modern technology. Using tools to simulate viewing and downloading of a webpage through a browser, the 'spider' (a metaphoric name for the part that goes into each web structure) goes into the webpage and 'crawls' (a metaphor for copy-and-paste) desirable contents into a formatted data structure. Since this technology only requires access to the webpages, theoretically, anything that can be downloaded can be scraped off the Internet.

In the past summer, I worked with Pinnacle Solutions, Inc.[1], a company in Indianapolis, Indiana. My job was solely to develop web-scrapers. In this paper, I will provide an overview and discussion of web-scraping, followed by a description of my internship experience.

[1]For information, visit http://psiconsultants.com/

1.1 An Overview of Web-Scraping

In general, there are three basic ways to collect data: from primary sources, e.g., conducting surveys and studies; from secondary sources, e.g., databases like the U.S. Census; and from existing but uncompiled information, e.g., product reviews from *Amazon.com*. Web-scraping focuses on the last way: to harvest data from a structured webpage, e.g., HTML[2], to a structured format, e.g., *.json*[3] and *.csv*[4]. It provides an amalgamation of the deluge of data spread over websites.

1.2 Usage of Web-Scraping

As web-scraping does not generate new data or results, it seems useless. However, people from various backgrounds may find web-scraping very useful. Here are few examples:

If a student were to conduct a study on what lessons people learn from documentaries such as *Food, Inc.*, he/she could go to *imdb.com*, *amazon.com* and *rottentomatoes.com* to scrape all the viewers' reviews. As there are thousands of reviews for *Food, Inc.*, web-scraping will help the student save much time trying to copy and paste every review. From the integrated results, the students might do a word frequency count to guess what kind of impression the majority have.

If a company were to have launched a product, they could use web-scraping to monitor their product, their competitors' products, and their partners products. If Amazon wants to know how successful is the new Kindle Fire HD, they can web-scrape all the reviews about Kindle Fire HD, about Nexus 7, and about Kindle Fire original version. Through analysis, Amazon is able to collect customers' feedback promptly.

If a statistician were to monitor the price change of thousands of commodities, and if

[2]Abbreviation of HyperText Markup Language
[3]Abbreviation of JavaScript Object Notation
[4]Abbreviation of Comma-Separated Values

all these commodities are sold online, the official can use web-scraping to collect the prices. The scraper will just copy the prices from the product page of an online retailer website, and paste all of them in a spreadsheet. Shifts in prices may reflect inflation/deflation, and the data can be used in economic research.

If a data analyst from Apple were to study how the public responds whenever a new product is released, he/she could collect the review dates from review websites and plot a frequency graph. The number of reviews generated per day may reflect how receptive the customers are to a new product.

1.3 Advantages of Web-Scraping

1.3.1 Speed

Web-scraping allows people to copy and paste from webpages faster. For instance, to harvest 10,000 pieces of Amazon.com product reviews manually by copying and pasting every little piece of information into an excel spreadsheet, doing it manually would take a very long time and be prone to mistakes. Web-scraping technology, on the other hand, makes the process much faster. By specifying certain paths in the webpage structure, the spiders go into specific attributes and fields to crawl information. With the help of web-scraping, 10,000 reviews can be copied and pasted into a *.csv* file within 15 minutes.[5]

1.3.2 Convenience for Data Analysis

Web-scraping improves the efficiency of data analysis. More often than not, tables of data are more ready to be processed than plain text. As web-scraping grabs from a structured format, it is able to copy and paste all the information needed into another structured format.

[5]The result is based on running of my own scraper.

For instance, after a *.csv* file is generated by scraping data from Amazon product reviews, all cells in the *ratings* column have a range of 1 to 5 for number of stars, all cells in the *review body* column will contain texts of the review bodies. A well-structured database will then provide foundation for further data mining and other modeling techniques.

1.3.3 Availability

Web-scraping are being more widely available today. Popular tools for web-scraping include browser extensions like *firebug* and *iMacros*, programs like *Wget*, programming languages like *Perl*, *Hadoop* and *Java* which have libraries to support sending HTTP requests, and programming language extensions like *Scrapy* and *Beautiful Soup* for *Python*. It is becoming more feasible for anyone to learn web-scraping, and to harvest data beyond what traditional ways like web API requests[6] can offer.

1.4 Challengess of Web-Scraping

1.4.1 Legality(extensive overhaul needed)

Web-scraping raises legal and ethical issues. While it is free to download many webpages, usage of the data within might not be in compliance with the terms and conditions of the source. Currently, there is no law forbidding web-scraping, nor is there any way to ban downloading of webpages(footnote needed, but I cannot find any law passages anyway). Lawsuits were fought between companies like American Airlines against FareChase (http://www.fornova.net/documents/AAFareChase.pdf), but it is impossible to ban people from downloading webpages. Practically, people use free information with citations. Frankly, as activities beyond downloading the webpages are operated locally, it is very hard to track

[6]requesting information through interacting with website's database

what information the user really scraped from. Therefore, there is a grey area in law and practice that people who use web-scraping tools find time and space to develop their projects.

1.4.2 Intricate and Dynamic Web Structures

As more advanced web structures emerge, scraping becomes more difficult. AJAX, for instance, sends out XMLHTTPRequest only when the user performs certain maneuvers in the browser. Also, embedded JavaScript contents may not be downloaded. The advent of these dynamic ways of loading web contents brings new challenges to web-scraping. **CONSIDER THE IMPACT OF HTML 5**

1.4.3 Instability

Another underlying problem comes from the fact that web-scraping is based on webpage structure. If the website adds a few toolbars, the locations of HTML attributes will change, resulting in crash of the scraper. If the website has an unusual encoding for text, the scraper may crash, too. If the structure is dependent on web broswer, they content might be different as well. These potential problems manifest web-scraping's intrinsic dependence on web interface.

Figure 1.1: What Google Chrome displays versus what firefox displays with identical urls.

Chapter 2

My Internship

2.1 Overview of My Internship at Pinnacle Solutions, Inc.

Pinnacle Solutions, Inc.(PSI) is a data company which helps clients to analyze and interpret their databases. Founded in 1996 by Kalamazoo College alumnus Donald Penix, Jr.(DJ), the companyi,located in downtown Indianapolis, Indiana, currently has over 20 employees. PSI primarily uses *SAS* and *Futrix* [1] for processing data, and *Amazon Web Services* for some server management.

I was a summer intern at PSI in 2011. At that time, Mr. Penix was very interested in "sentiment analysis", a technology aims to discover and analyze sentiment reflected, for example, by comments, reviews, and blog posts. For instance, a comment like "I love this iPad!" indicates a positive sentiment, and one like "I am sick of the iPad!" suggests a negative sentiment. Sentiment analysis will consolidate related comments and analyze them using data mining and natural language processing. After research, we found it not feasible for PSI, as none of the employees currently have the needed expertise. So we decided to start

[1] For more information, see Glossary of Terms

from scratch, and data collection is the first step towards sentiment analysis.

2.2 How Web-Scraping Became the Best Option for PSI

Analysis of data starts from collection of them. In order to harvest usable data from the Internet, there are many ways other than web-scraping.**Probably ought to refer back to Section 1 where you introduced sources of data.**

First, one can extract necessary information through web APIs.**explanation of API needed!** Many websites offer APIs for developers to use to build applications. Twitter, for instance, published a new version of API months ago. Through an API, users can request information directly from the webites database, and the data will be clean and formatted. Twitter, for instance, offers exportable data in *.json* format.) However, to obtain more data, users may need to pay for them. Moreover, many websites may have limited information provided in their web APIs, or may not provide APIs at all. In that case, even though users can view the data from the browser, they cannot download it through APIs. Therefore, for a small company, it might be expensive and insufficient to rely on APIs for data harvesting.

Second, there are available third-party scraping/data harvesting services. There are applications like *Mozenda*[2], and highly customizable services such as *scrapinghub*[3], *GNIP*[4], and *Open Amplify*[5]. However, these services are not cheap to obtain. Moreover, in the long run, as PSI needs more information from the Internet, the dependency on these services will be ever increasing. Therefore, as a data company, it would be a wise choice if PSI could come up with a low cost, independent system to fetch information that is so close yet so far

[2]http://mozenda.com/
[3]http://scrapinghub.com/
[4]http://gnip.com/
[5]http://www.openamplify.com/

from them.

Third, as they are a certified *SAS* reseller, they could use the software *SAS Sentiment Analysis Studio*. However, this application is not only expensive for clients to buy, but also dependent on the web API.

After many thorough discussion sessions, a free software capable of fetching data from the Internet was determined to be desirable, and web-scraping seemed most promising. Many web-scraping resources are free, so the cost of doing web-scraping is low. As many programming languages and their extensions are open sources, many people contribute to the library and trouble-shooting, so the technical support is very active.

After some research, I chose Scrapy, a Python web-scraping framework, for the job.

2.3 Scrapy, a Python Framework

To quote from its website, "Scrapy is a fast high-level screen scraping and web crawling framework, used to crawl websites and extract structured data from their pages. It can be used for a wide range of purposes, from data mining to monitoring and automated testing."[6]

Scrapy depends not only on Python, but also few other libraries. Scrapy uses Twisted[7], an event-driven networking engine, and Zope[8], a web application server framework. The installation guide can be found on Scrapy's website[9].

There are few basic ideas or classes in the Scrapy framework.

First, spider. *Spider.py* is responsible for going into the webpages and crawling data from them. At the very least, the user needs to specify the domain, the starting url, and the location in the webpage from which to crawl. If the user wants to scrape from multiple tables or multiple pages, or even conditionally scrape from certain pages, he/she can define

[6]http://scrapy.org/
[7]http://twistedmatrix.com/trac/
[8]http://zope2.zope.org/about-zope-2/what-is-zope-2
[9]http://doc.scrapy.org/en/latest/intro/install.html

those in *Spider.py*

Second, items. *Items.py* is a class specifing the column names in the result. In the end, the result would be a table consisting of all the scraped data, and items defines what are they. Items in the same class will be in the same table.

Third, settings. *Settings.py* includes information such as what information will be written into the log file, limit on frequency of requests sent, and other customizable features. In the event of a user login is needed, the user can pre-fill that in the login to avoid access denial.

Fourth, pipeline. *Pipelines.py* decides how the scraped data are processed. By default, all data will be dumped into a single file. However, if the user wants to add a filter to get rid of some data, pipeline handles that. If the user wants to split the results from one spider into two separate files, he/she can do that through coding in *pipelines.py*.

Scrapy has more advanced features, which would fall outside of the scope of this paper. In the next chapter, I would like to offer a high level approach of web-scraping using Scrapy.

Chapter 3

Scrapy Explained in Greater Detail

3.1 Test How Much You Can See

Web-scraping depends on whatever the browser can "see", and there are times when the browser sees less than a person does. To check, one can use the following line of code in the command-line (cmd.exe in Windows, Terminal in Mac OS/Linux):

```
scrapy shell http://amazon.com
```

After Scrapy finish analyzing the web address, it goes into its prompt and asks for further instructions. Here, the user can key in

```
view(response)
```

A browser window opens and shows you what the website is like in Scrapy's perspective.

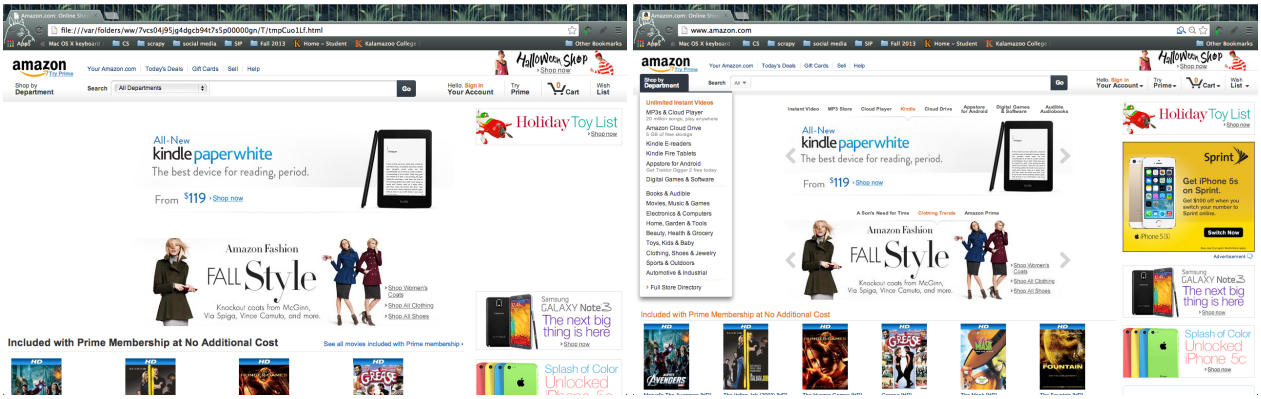

Figure 3.1: What scrapy is able to download locally versus what the user can see.

3.2 Test If the Data is Scrapable

While in theory, any data which can be seen can be downloaded, and any downloaded data can be scraped, in practice, it is not that ideal. Even if Scrapy can see the data the user needs, the difficulty of scraping them varies. For instance, in the event of embedded *JavaScript* code, getting the result of a function call will be more difficult.

After setting up a project with few commands[1], it is wise to test if certain data are scrapable. Before knowing that, one needs to specify where exactly the data are located. To establish a standard way of referring to a certain field in the webpage, XPath[2] is used.

To retrieve a certain XPath, one can use an built-in *"Inspect Element"* tool, or an extension such as *firebug*[3], available for *Firefox* and *Google Chrome*. Here is an example using *"Inspect Element"* in *Google Chrome* to extract an XPath from *http://www.nbcnews.com*:

After knowing a specific XPath, one can write related code to test if Scrapy can recognize the XPath and scrape the data.If one piece of data is scrapable, it is likely that the entire table which contains that piece of data can be scraped. If the table is scrapable, it is also

[1]For more information, visit http://doc.scrapy.org/en/latest/intro/tutorial.html. The website has a simple example on how to start a project

[2]Abbreviation of "XML Path Language".For more information, see http://www.w3.org/TR/xpath/

[3]Visit http://getfirebug.com/ for more information

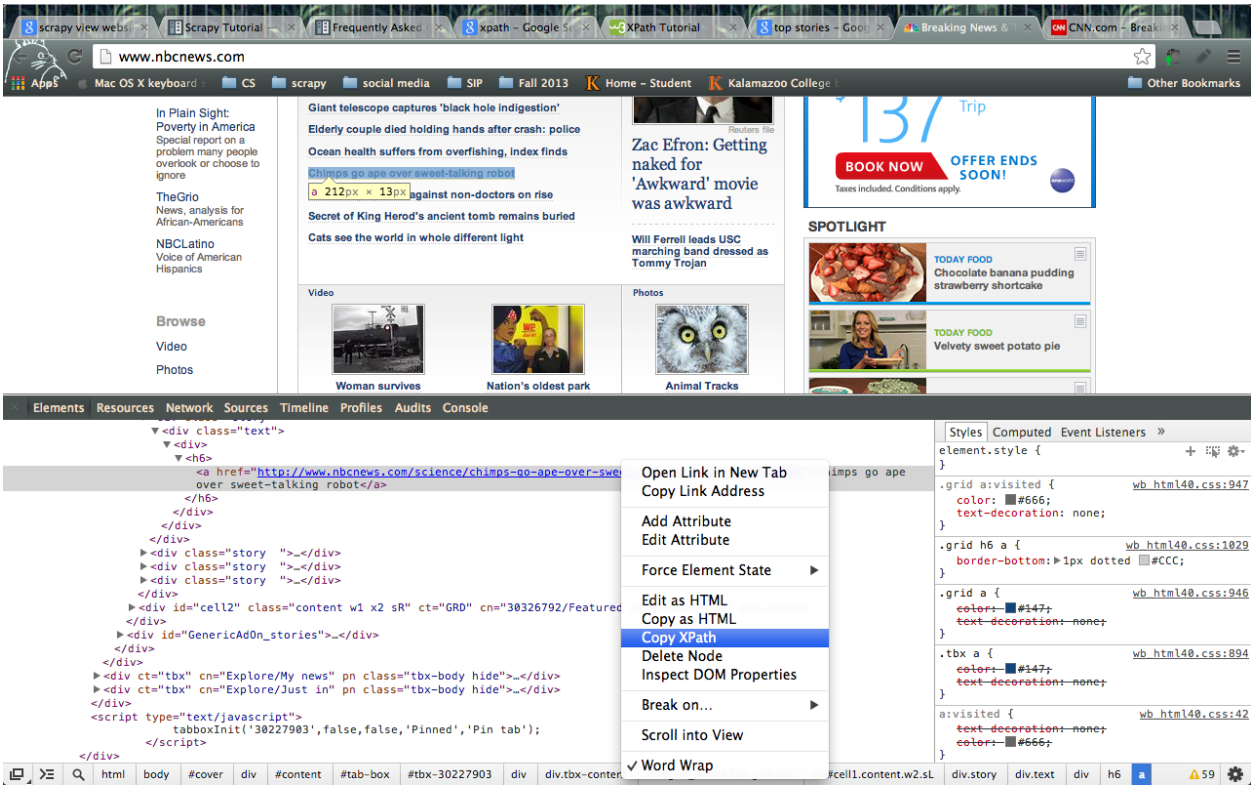

Figure 3.2: A screenshot of how to obtain an XPath from a webpage, and the XPath of the link is *//*[@id="cell1"]/div[8]/div/div/h6/a*.

likely that similar tables can be scraped, too.

3.3 Iterators

Web-scraping relies heavily on the structure of the webpage. If the webpage is very organized, it is possible for the user to use XPath to find all entries in a table, and send out a request to scrape the next page. In Scrapy, the request is in the form of a callback function. The spider will scrape the data yield the results from the page, and then excute the callback function. Here is a snippet of that concept:

```
class Spider(BaseSpider):
      def parse(self,response):
      ##some code to extract data from this current page
            yield reslts_from_this_page
            if next_page:
                  yield Request(url_of_the_next_page,
                     callback=self.parse)
```

In the code above, *BaseSpider* is one of the spider classes the user has to inherit to build a user-defined spider. In any spider class, a method like *parse(self,response)* is necessary to communicate with the computer and instruct which fields in the html to scrape from. The *parse* method yields a generator, and the code does not necessarily end there. In the event that there is a next page with the same html format, the parse method will be able to go into the new html webpage and collect the new data in the same way as it did for the previous page. Therefore, the method *parse* can also yield a *Request*. *Request* represents an HTTPRequest in many case. It sends a signal to the scraper and asks for a *Response*, which will be handed to the spider and the spider will scrape it.

3.4 Specification

After sketching and testing the basic functionalities, it is time to study what types of information the clients want, and what types of information are needed to make database management easier.

The clients usually ask for specific data that are useful to their research. For instance, an author may want to read all of his books' reviews. To collect book reviews from websites such as *amazon.com* and *goodreads.com*, we need different spiders for different websites, but potentially generate tables with identical column names so that the author can view everything in one sheet. That datasheet may contain book name, edition number, publisher, review website, user name of the reviewer, date of review, ratings, review bodies, and other information as the column names.

On the developer's end, however, the structure will be very different. To track every piece of review, the developer may need a column with the url from which that particular review was pulled. To establish an iterator, the developer may need to track the book identification number on the website to generate the correct url to scrape from. To deal with different encodings and web structures of different websites, the developer needs to customize every spider to suit its respective website.

(More examples are on the way)

3.5 Produce a sample output

After studying what and how the spider should scrape from webpages, we are ready to produce a sample output. The sample output is necessary for three reasons.

First, formatting and encoding. It is not unusual when the ideal is far from reality. Although XPath offers a specific location of all scrapable information, it is highly possible that the formatting is not desirable. For instance, due to the fact that most websites are

written in utf-8, but Python's csv module defaults encoding to ascii, a paragraph often has a $\backslash u$ in front of it.(code snippets and examples on their way).

Second, communication with clients and manager. Only when the sample data sheet is out, can the results of scraping be evaluted. There are limitations to scraping, so it is essential to make clients and managers aware of those. For instance, the time of producing a sample output influences the frequency of running the scraper. In the event that the target website restricts IP adress from visiting it too fast, the time taken to scrape data will definitely be longer than expected.

After harvesting data from online sources, there are various ways to enjoy the fruit. This chapter will illustrate some of possible ways to use the data.

Third, testing. As web-scraping is based on web structure and HTTP requests, it is unstable by nature. Although the code may work perfectly in theory, it may encounter problems in practice. These problem may be relevant to Internet connection, broadband width, geographic locatin, configuration of the computer, and other factors which may influence the scraping process. As the spider sends out requests and receive responses at a fast pace, it might encounter concurrency issues, causing some data to be skipped over or scraped repeatedly. All these issues are instrumental to the performance of a certain spider and Scrapy project, so the developer needs to read the sample output in great detail and make amendments on the code.

3.6 Customization

After the sample output seems qualified, the developer may consider other functionalities which may enhance the utility of a particular scraper.

For instance, to efficiently scrape updated reviews from a website, one can build a last page tracker. If a product has 50 pages of reviews on day 1, one can view the reviews by

ascending date order so that the latest review appears on the last page. After the scraper scrapes the 50 pages on day 1, it store the last page as 50, so that when there are 60 pages on day 2, the scraper can start scraping from page 50 to 60, without re-scraping the first 49 pages.

For another instance, one can make the scraper more interactive with the user. One can use Pipeline to save the output data into a user specified directory. For some scraping job like product reviews, it is also possible to ask for an identifier of the product, and the scraping process will start from an auto-generated url.

Chapter 4

Future Developement of the Project

4.1 Natural Lanuage Toolkit

After harvesting data from online sources, there are various ways to enjoy the fruit. This chapter will illustrate some of possible ways to use the data.

NLTK, the abbreviation for "Natural Language Toolkit", is "a leading platform for building Python programs to work with human language data."[1] This toolkit has many functions, and one of them is tokenization. Tokenizatio is the process of breaking up a paragraph into sentences, a sentence into phrases, a phrase to words, or even a word to its components. These subsets are called tokens, and more study can be done through analyzing these tokens.

For instance, if a product manager would like to monitor the sentiment of customers on the product, he/she could use web-scraping to collect review texts from online rating websites. Although those texts are available, the feedback from the reviews are not quantitative. With NLTK, however, one can tokenize each review into sentences, and sentences into phrases. For example, one can assign a score of "1" if the review says "I hate this prod-

[1] quote from http://nltk.org

uct!", and a score of "10" if the review reads "I love this product!". There are databases such as WordNet available for download as a package for NLTK, and there are algorithms for assigning scores to words.

However, there are few challenges in the process of detecting sentiment.

First, the ambiguity of language. Same word may have different meanings in different context. For instance, in the review of a product, "I am sick of it" and "This product is sick!" have opposite sentiments, but they both use the word "sick" to express the sentiment. As the internet language becomes increasingly versatile, with slangs, emoticons, puns, and other forms of language in the scraped text, the difficulty of guessing the right sentiment of an expression increases drastically.

Second, the complexity of language. Short statements such as "This product is great!" or "I hate it!" are generally easier for sentiment detection. As the statement becomes longer into a complex sentence, a paragraph or even multiple paragraphs, the evaluation of a certain review becomes more complicated, too. For instance, for a review such as "Though this product may not be the worst in the world, please choose other products if possible.", it seems obvious to human that the sentence contains negative sentiment against the product. To the computer, however, when it sees "not" followed by "worst", it may assign a very positive sentiment to this review.

4.2 Data Mining

Web-scraping has the potential of collecting data from multiple facets. Whena review website displays reviews on its webpage, information such as date, ratings, review texts and other relevant details will be available as well.

While it seems feasible to operate data mining under many collectable variables, the actual practice will not be easy. There are two main limitations.

First, limitation of parsing. The default format of scraped data is string, but more often than not, numbers are preferred in data mining. The process of parsing numbers from their string representations may incur errors.

Second, limitation of accessible data. Websites are not likely to display some data, such as demographic data, to the public. To investigate possible correlations, insufficient data will be available for scraping.

4.3 Graphic User Interface

For future development of web-scraping, it is possible to build an user interface from the code.

To scrape information related to a particular item, the user can choose a website to scrape from, and input necessary fields such as the unique identifier of that item in the website. Then, the user can choose from a checklist of what areas to scrape from, and the number of results shown. Then, after clicking a button, the Python script at the backend will use Scrapy to retrieve the input information, create starting and ending urls, and scrape selected results from relevant pages.

However, there are few limitations of establishing an interface.

First, validity of user input. As the backend Python script retrieves information from the user interface, invalid input may cause the program to stop functioning.

Second, constantly changing web interfaces. As web-scraping depends heavily on web structures, in the event of a change in web interfaces, the spider will probably fail to crawl information. Therefore, both the programmer and the user need to update frequently. The stability of the software will be very fragile.

Appendices

Glossary of Terms

Ajax, JavaScript

Amazon Product Reviews

API

Business Intelligence

Crawl

.csv, .json, .xml

data harvesting

data mining

encoding, decoding

HTTPRequest

NLP and NLTK

SAS, Futrix

sentiment analysis

spider

scraper

web-scraping