A Survey on Python Libraries Used for Social Media Content Scraping

Conference Paper · September 2020

DOI: 10.1109/ICOSEC49089.2020.9215357

3 authors:

Thivaharan Sakthivadivel
PSG Institute of Technology and Applied Research
21 PUBLICATIONS **40** CITATIONS
SEE PROFILE

Srivatsun Gopalakrishnan
PSG College of Technology
19 PUBLICATIONS **130** CITATIONS
SEE PROFILE

S. Sarathambekai
PSG College of Technology
30 PUBLICATIONS **146** CITATIONS
SEE PROFILE

Proceedings of the International Conference on Smart Electronics and Communication (ICOSEC 2020)
DVD Part Number: CFP20V90-DVD; ISBN: 978-1-7281-5460-2

A Survey on Python Libraries Used for Social Media Content Scraping

Thivaharan. S
Asst. Prof (Sel. Grade)
CSE Department
PSG Institute of Technology and Applied Research
Coimbatore

Srivatsun. G
Associate Professor
ECE Department
PSG College of Technology
Coimbatore

Sarathambekai.S
Asst. Prof (Sel. Grade)
IT department
PSG College of Technology
Coimbatore

***Abstract* - Python has a rich set of libraries available for extracting the digital contents that are spread across the internet. Among the available libraries, the following three libraries are popularly deployed for the purpose: they are BeautifulSoup, LXml and RegEx. A statistical study carried out over the scattered available data set shows that RegEx is capable of delivering the answer on an average of 153.7 ms. Still, RegEx has the inherent drawback of having limited rule extraction when it comes for the web page with more inner tags. Because of this demerit RegEx is termed as capable of performing only moderately complex contexts. Nevertheless the other libraries BeautifulSoup and LXml are capable of extracting web content under critical environment yielding the response rate of 458.68 ms and 202.96 ms respectively. Also, these two libraries are based on the DOM model proving to be the scalable libraries. The modern content grading system [1] specifically developed for the regional languages available in social media are mostly influenced by the web scrappers. This survey justifies the overwhelming performance of RegEx under differing scenarios.**

Keywords: Hypertext Markup Language (HTML), Document Object Model (DOM), Digital content Scraping

I. INTRODUCTION

A recent study shows that the internet is constantly thronged with multifold of data. Among them, the social media platforms contribute to the majority of these data. Always it is a concern for the researchers and bloggers to differentiate from the "good" and "vulnerable" contents. To evade this, a technique called web scraping [2] is deployed to pull out the content of interest from the digital platform. Web scraping also takes in to account the data that are shared in real-time by means of chatting and live streaming [3]. In this article, it is carefully analyzed, how the digital content is pulled-out from the digital platforms. Python along with its rich set of utility libraries is considered for the survey in web scraping. The technique of Web scraping is derived from the following three models and is evolving over time: DOM (Document Object Model) [4], Java API Wrapper based Methods [5] and SVM based classifiers [6]. Java API based wrappers are the actual programs written from scratch for content segregation. DOM is the model, as well as the architecture consisting of the utility structures, sophisticated tags and the contributing attributes of the HyperText Markup Language (HTML). Support Vector Machine (SVM) based models have vastly available algorithms in the form of API for classification even in "n"-dimensional paradigms.

In this article, a detailed analysis is made by keeping the following as the metrics for evaluation: a minimal gap in classification, Minimum response time and less processing cycle consumption. In this article the purpose of web scraping is compared and correlated among the RegEx, BeautifulSoup [7] and LXml [8], where the later two are derived from the DOM architecture and are scalable. The distant vector metric machine learning approach [9] with the capability of n-dimensional space parameter classification for the above-mentioned metrics yields and assigns a maximal proportionate ratio to the single parameter classification considering the response time alone.

II. WEB SCRAPING STAGES IN THE DOM ARCHITECTURE

The Modern DOM models even have the capability of producing the alternate trees based on the thesaurus or lexicon. The familiar DOM tags in mark up languages are <h1>, <div>, <Span> and the associated classes of the <Span> tag. A rubric developed for a web page is reusable in the sense it can be effectively applied for the subsequent web pages as well if they share the same semantics.

The relationship between HTML tags and DOM architecture based debuggers is shown in figure1 (Source: Mozilla Firefox 79.0 – 32 bit / Debugger console).

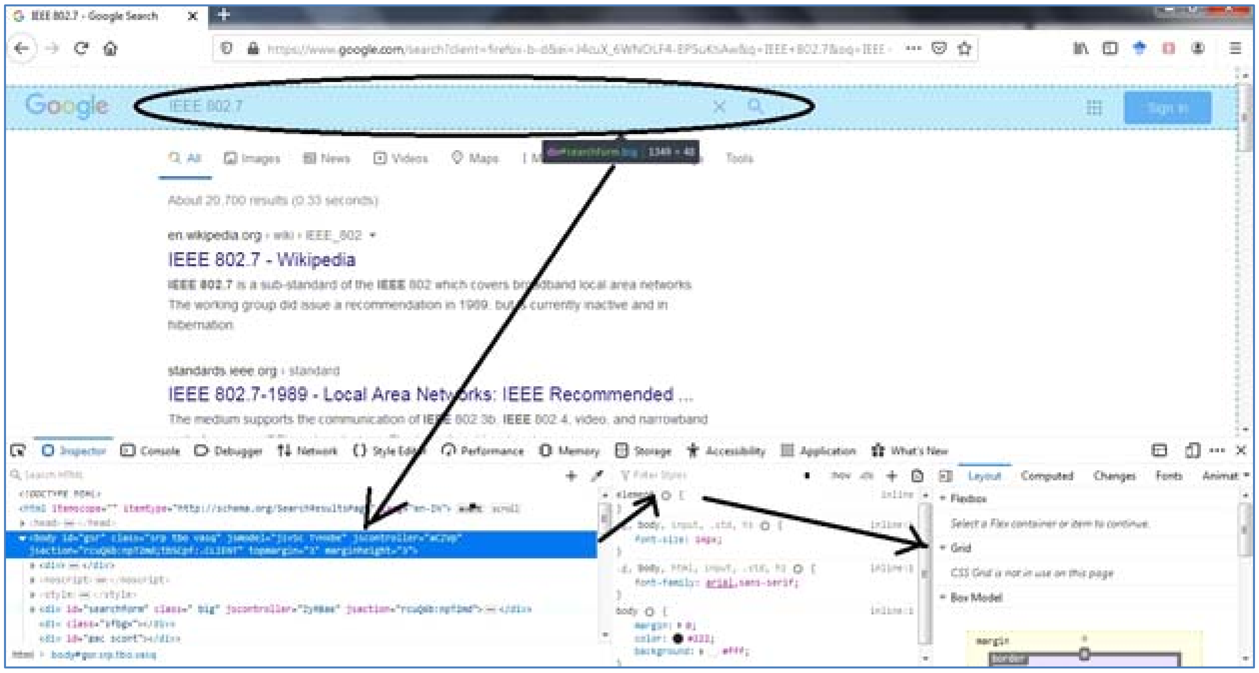

Fig 1: Web Browser with Debugger mode enabled (mapping DOM and Tag)

III. CONTENT EXTRACTION USING PYTHON LIBRARIES

In this article, the following three aspects of social media content extraction are taken in to account: Regular Expression, BeautifulSoup and LXml. Content extractions are executed on the web pages with the matching pattern. Once the extraction phase is over, the matching pattern along with the operational metadata are recorded in the dataset either as a new entry or as an updated entry in the existing dataset.

A. Regular Expression

Regular Expressions are (sometimes referred to as RegEx[10]) the pattern that needs to be searched in the digital content of interest. In python, a library called “re” has been used to perform the pattern matching task.

The following is the code to perform the extraction in the <h2> tag along with its class name. The digital data can be pulled out from the markup language either by applying it over the whole document or the partly available content. The partly available content is called as the privileged content.

```
def pattern_match(str):
        return str.replace(“ “,”
        “).replace(“<h2>””,”.”).replace(“?”)+”</” +
        separate_element(str)+”/>”
```

```
def separate_element(input):
        return element[(element.find(‘<’) +
        (element.find(‘’))
```

In the above code, two functions namely “separate_element()” and “pattern_match()” are used for web scraping. Each of the functions accepts a parameter. The separate_element() function accepts an input from the user normally a patter to be parsed in the digital content. This parameter is then passed to another function pattern_match(), which uses the method replace() to find out the pattern. Once the pattern is found out the result is then forwarded to the code written below for grouping the records of the findings.

```
def group(DOC, str):
        answer = re.search(pattern_match(str),
        DOC, re.DOTALL))

        if answer:
                return answer.group(1)
        else:
                return ‘’ # a null string
```

B. BeautifulSoup

BeautifulSoup is an open-source library provided along with python. It has the default architecture binding capability with that of DOM. This library operates in the following stages to perform web scraping.

The first stage collects the list of repositories [11] using the property “user-repositories-list”. This property focuses on the div components and its location is marked along with its binding and spread in the entire document of interest.

In the second stage, the relevant information of each of the DOM components scrapped in the first stage is taken for investigation. “Relevance information” [12] in turn is a list that is getting populated dynamically. The list segregates the sections as it appears in the relevance DOM model.

In the third stage, the location-specific information of the repository and the associated links are stored. The associated links are referred to in the mark-up language using the <href> tag.

The fourth stage prepares a documented “repository description” [13], thereby makes the further pattern matching easy and is reusable.

The fifth stage encodes the scrapped digital contents with a specific language. The following code demonstrates all these stages concisely.

```
def scrap_all(DOC, str, BS):
        ANS = BeautifulSoup(DOC, BS)
                Soup.find_all(res.append(res.decod
        e_all(style=”xhtml”)))
        For item in ANS:
        Tag = tag + ANS.append(format(“Xhtml”))
        Return Tag
```

```
from bs4 import BeautifulSoup

def Init_BS(DOC, Parser):
        soup = BeautifulSoup(page.content,
        'html.parser')
        for item in list(soup.children)
                Tag = Tag + list(soup.children)[2]
        return Tag
```

The above has a function “scrap_all()”. This function accepts three parameters namely document to be parsed, the pattern to be looked after and the type of RegEx extension. The first line in the function initiates the document extraction using the BeautifulSoup() constructor using the default parameters.

After initialization, a search_all() method is called alongside of the soup property. The answers are stored in a default buffer unless and until specified explicitly. From the buffer, iteratively the tag component is getting updated. Finally, the updated tag component is returned.

C. Lxml

Lxml is a stand-alone content scraping library which looks the document of interest as a chunk. Chunking is the mechanism used for linking the stranded associations in the digital content. Chunking is a novel approach used for identifying the weakly linked and strongly linked components. This makes the extraction process simple when the document is of highly inward in nature. Inward document normally contains multiple recursive links making the other parsers to stammer a lot in the extraction process. Xpath[14] is a content extraction tree, which is found to be useful in the Lxml parsing.

To perform the scraping process, “etree” module should be imported from the Lxml package. StringIO is another module to be imported for the process. The first step is to initialize the Xpath process. Once Xpath succeeds the actual extraction is started. The following depicts the Lxml way of scraping.

```
def Init_Xpath(str, tagname):
        base = etree.fromstring(pattern(“tagname”))
        tag = ‘’
        for item in base.keys():
                tag = tag + base.get(‘att’)
        return tag
```

The above code accepts two arguments and populates the base dictionary though the etree module and the method fromstring(). Iterative the pattern is instated upon the tag and the same is returned after the exhaustion of the loop.

```
from Lxml import etree

def scrap_all(DOC, str):
        base = etree.parse(io.stringio(DOC), LX)
        temp = Xpath(str)
        for item in temp:
                lst1.append(etree.tostring(item,
                elem_code=’unicode’))
        return lst1
```

The above imports the etree module. The function scrap_all() generates the parse tree and the same is searched using the Xpath. The results are stored in the list called lst.

IV. COMPARISON AND CORRELATION OF THE PYTHON LIBRARIES

All the above codes which are intended for the same purpose is executed in the presence of time.clock() method. The outcome is documented and is taken for the analysis. This method returns the floating point as the outcome. This is a win32 method based on QueryPerformanceCounter [15].

A database and a referral document are made available in the offline status. The document is chosen such that it consists of average to medium patterns. The result is summarized below for each of the scrapping libraries.

Quantitative characteristics of the Input workspace:
Total number of words: 15,026
Complexity of the document: Average, Best and worst case complexity

Table 1 shows the metric and their proportionate weights in the evaluation of the response time considering the document complexity ranging from worst to best case.

TABLE 1 – proportionate Metric & weights

Document Complexity /Metric	Type of parsers	Initializers	Scrapping method	Weights
Worstcase	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{2}$	0.05556
Average case	$\frac{1}{3}$	$\frac{1*1}{3*2}$	$\frac{1*1}{2*2}$	0.01389
Best case	$\frac{1}{3}$	$\frac{1*1*1}{3*2*2}$	$\frac{1*1*1}{2*2*2}$	0.00347

Table 2, below shows the response time taken by the individual models of discussion in worst case complexity scenario. Worst case scenario tests the response time with maximal erroneous pattern and maximally stretched inherently looping patterns as well as the mixture of different regional language words.

TABLE 2 – Worst case - RegEx vs BeautifulSoup vs Lxml

Qualitative / Quantitave	Method			Time (ms)		
Type of Parsers	RegEx	BeautifulSoup	Lxml	RegEx	BeautifulSoup	Lxml
Initializers	Pattern_match()	Init_BS()	Init_Xpath()	371.11	872.91	87.30
Scrapping method	Separate_element()	Scrap_all()	Scrap_all()	581.93	371.03	321.05
			Total time (ms)	953.04	1243.94	408.35

Table 3 below shows the response time taken by the individual models of discussion in average case complexity scenario. Average case scenario tests the response time with minimally occurring erroneous pattern and less number of stretched inherently looping patterns.

TABLE 3 – Average case - RegEx vs. BeautifulSoup vs. Lxml

Qualitative / Quantitave	Method			Time (ms)		
Type of Parsers	RegEx	BeautifulSoup	Lxml	RegEx	BeautifulSoup	Lxml
Initializers	Pattern_match()	Init_BS()	Init_Xpath()	165.71	807.01	56.71
Scrapping method	Separate_element()	Scrap_all()	Scrap_all()	233.03	512.72	163.2
			Total time (ms)	398.74	1319.73	219.91

Table 4, below shows the response time taken by the individual models of discussion in best case complexity scenario. Best case scenario tests the response time with minimally or no occurring erroneous pattern and less or no inherently looping patterns.

TABLE 4 – Best case - RegEx vs. BeautifulSoup vs. Lxml

Qualitative / Quantitave	Method			Time (ms)		
Type of Parsers	RegEx	BeautifulSoup	Lxml	RegEx	BeautifulSoup	Lxml
Initializers	Pattern_match()	Init_BS()	Init_Xpath()	56.17	321.01	50.13
Scrapping method	Separate_element()	Scrap_all()	Scrap_all()	97.53	137.67	152.83
			Total time (ms)	153.7	458.68	202.96

Table 5, below lists the True positive and true negative results of all the above execution with the previously calculated weights 0.055556, 0.013889, 0.003472 for the worst, average and best case complexity of the document respectively. True positive is calculated from the best case outcomes of all the parsers. True negative is calculated from the worst case outcomes of all the parsers.

TABLE 5 – Prediction Accuracy

Prediction accuracy / Type of parser	RegEx	BeautifulSoup	LXml
True Positive (TP)	8.53	6.37	0.704
True Negative (TN)	52.94	17.27	1.41

The figure 2 below shows the correlation chart in line mode between the prediction accuracy and the type of parsers. It is evident that, from the chart in both the True Positive and True negative mode of prediction accuracy the RegEx has the maximum turn-out.

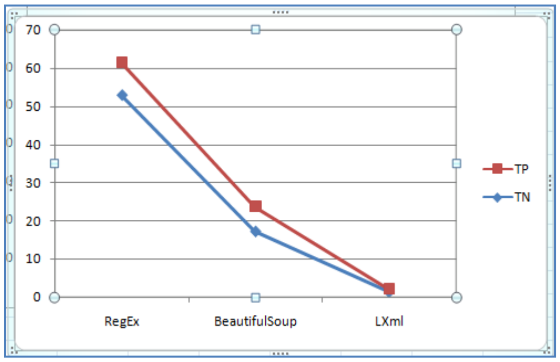

Fig 2 - correlation chart (prediction accuracy Vs. type of parsers

All the above details are plotted as line chart in the figure3 below. The line plot is associated and analyzed between the document complexity (Worst, Average and Best) and the response time in milliseconds.

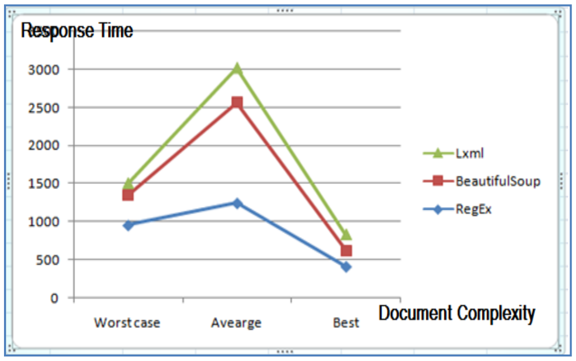

Fig 3: Comparison of RegEx vs. BeautifulSoup vs. LXml

From the above results it is evident that chosen a document with moderate matching patterns, the RegEx is extremely faster considering all the cases.

The line chart is obtained from Microsoft Excel. The blue line, firebrick line, yellow green line corresponds to the response times of RegEx, BeautifulSoup and LXml respectively.

CONCLUSION

In this study, a dataset consisting of 15,026 words is taken for the analysis. Among the various available IDEs, this study used the "PyCharm" stable release 2020.1.2 (Build: 201.7846.105) / 25 June 2020 developed by JetBrains under the license of Apache® Inc. The experiment is carried over all the three scrapping libraries in python namely RegEx, BeautifulSoup and Lxml. The document dataset is flexed from best case complex patterns to worst case complex patterns through average case complex patterns.

The pattern is inflected to ensure the durability of the scrappers in all circumstances. The pattern tuning is exhibited manually by way of changing the grammatical pattern of the sentences as well as the replacement of high degree, rarely used thesaurus words. The outcome of each of the cases is recorded through a python time lapse method called time.clock() under the same hardware platform without changing or tuning any of the application software.

From the tables, it is evident that RegEx is capable of delivering good throughput in pattern matching and scrapping. Further, this study has to be carried out to check the accuracy of these three scrappers along with other metrics like dynamism in the content, originality of the content.

REFERENCES

[1] Thivaharan S, Hariharan K, Christie Jerin Kumar, content grading system for Tamil based on indexed set weights using PC-Kimmo, International journal of engineering research and technology, 177-181, Vol 8, Issue-3, ISSN: 2278—0181, March – 2019.

[2] Eloisa Vargiu, Mirko Urrul, Exploiting web scraping in a collaborative filtering- based approach to web advertising, DOI: 10.5430/air.v2n1p44, Artificial Intelligence Research, Vol. 2, No.1, 2013.

[3] Xiaofei Liao, Hai Jin, Yunhao Liu, Lionel M. Ni, and Dafu Deng, AnySee: Peer-to-Peer Live Streaming, publication in the Proceedings IEEE Infocom, Vol. 2, Issue No.2, Dec – 2011.

[4] Document Object Model (DOM) Level 1 Specification, 2nd Edition, Version 1, W3C Working Draft 29 September, W3C, 2000.

[5] Nithesh.V. Chawla et al, Wrapper-based computation and evaluation of Sampling methods for imbalanced datasets, ACM, 11https://doi.org/10.1145/1089827.1089830, Pages 24–33, August 2005

[6] Hui-LingChen et al, A support vector machine classifier with rough set-based feature selection for breast cancer diagnosis, Expert Systems with Applications, Elsevier, Volume 38, Issue 7, Pages 9014-9022, July 2011.

[7] ChunmeiZheng et al, A Study of Web Information Extraction Technology Based on Beautiful Soup, Journal of computers, Volume 10, Number 6, November 2015

[8] Stephan Richter, lxml - XML and HTML with Python, https://lxml.de/

[9] Kilian Q. Weinberger, Lawrence K. Saul, Distance Metric Learning for Large MarginNearest Neighbor Classification, Journal of Machine Learning Research, 207-244, 2009.

[10] Thomas, G.S., Thompson, R.C., Miyamoto, M.I. et al. The RegEx trial: a randomized, double-blind, placebo- and active-controlled pilot study combining regadenoson, a selective A2A adenosine agonist, with low-level exercise, in patients undergoing myocardial perfusion imaging. J. Nucl. Cardiol. 16, 63–72, https://doi.org/10.1007/s12350-008-9001-9, July 2011.

[11] Deborah L. McGuinness et al, Investigations into Trust for Collaborative Information Repositories: A Wikipedia Case Study, Proceedings of the Workshop on Models of Trust for the Web, May 21, 2006

[12] Buckley C., Salton G., Allan J. (1994) The Effect of Adding Relevance Information in a Relevance Feedback Environment. In: Croft B.W., van Rijsbergen C.J. (eds) SIGIR, Springer, London, 1994.

[13] Eong, D., In, P. H., Jarnjak, F., Kim, Y.-G., & Baik, D.-K., A message conversion system, XML-based metadata semantics description language and metadata repository. Journal of Information Science, 31(5), 394–406. https:// doi.org / 10.1177 / 0165551505055403, 2005.
[14] James Clark, XML Path Language (XPath), Version 1.0, W3C Recommendation 16 November 1999,

[15] QueryPerformanceCounter( ) function: https:// docs.microsoft.com / en-us /windows /win32 /api/ profileapi/nf-profileapi-queryperformancecounter

[16] Daniel Glez-Peña, Anália Lourenço, Hugo López-Fernández, Miguel Reboiro-Jato, Florentino Fdez-Riverola, Web scraping technologies in an API world, Briefings in Bioinformatics, Volume 15, Issue 5, pages 788–797, https://doi.org/10.1093/bib/bbt026, September 2014.

[17] Praveena, A., and S. Smys. "Ensuring data security in cloud based social networks." In 2017 International conference of Electronics, Communication and Aerospace Technology (ICECA), vol. 2, pp. 289-295. IEEE, 2017.

[18] Sathesh, A. (2019),"Enhanced Soft Computing Approaches For Intrusion Detection Schemes In Social Media Networks", Journal of Soft Computing Paradigm (JSCP), 1(02), 69-79.

[19] Sotiris Kotsiantis et al, Handling imbalanced datasets: A review, GESTS International Transactions on Computer Science and Engineering, Vol.30, 2006.

[20] Mohamed Bekkar et al, Evaluation Measures for Models Assessment over Imbalanced Data Sets, Journal of Information Engineering and Applications, ISSN 2224-5782 (print) ISSN 2225-0506 (online), Vol.3, No.10, 2013