# Extracting HTML Web Content Using Paragraph Tags and Natural Language Processing

Bhavesh Kumar Bohara
*Department Of Information Technology*
*Indian Institute of Information Technology, Allahabad*
Prayagraj, India
mml2022013@iiita.ac.in

***Abstract*—This article investigates text extraction from HTML web pages, more specifically using paragraphs tag and NLP approaches. The suggested approach takes into account the structural information offered by paragraph tags as well as techniques for tokenization and part-of-speech tagging to identify significant information. Experimental results show the approach's effectiveness, and potential applications include web scraping and information retrieval.**

***Index Terms*—HTML, NLP, Event Extraction, Tokenization, Web scraping**

## I. Introduction

People all across the world use the internet as a necessary tool to acquire information, communicate with others, and conduct business. There is a growing need for efficient and effective techniques to extract relevant data due to the development of websites and the abundance of information available. However, because online pages come in a variety of designs and forms, it can be difficult to extract relevant information from them.

The essential events described in each paragraph of a news item, for instance, might be extracted using paragraph tags. This would make it easier for you to comprehend the article's substance and instantly identify its most crucial sections. Using NLP methods is one method of extracting text from web pages. NLP is a branch of artificial intelligence that studies how computers and human language interact. It makes it possible for computers to comprehend and examine spoken and written English. Researchers can more quickly and effectively extract pertinent data from web pages by utilising NLP techniques.

The suggested technique for extracting text from web pages in this case takes into account the structural information provided by HTML web page, paragraph tags. This method can be used to extract information from news articles and other web pages that are organised around paragraph tags. The method makes use of part-of-speech tagging and tokenization techniques to extract significant information from the gathered text. Higher F1 scores compared to other algorithms, such as the Boilerpipe method, in experimental findings demonstrated the usefulness of the suggested approach. Web scraping and information retrieval are two potential uses for this strategy that might have a big impact on fields including market research, corporate intelligence, and academics.

Also, this study shows how NLP techniques can be used to extract text from online pages. The suggested approach has the potential to completely change how we use the enormous amount of data that is available on the internet and shows promise for efficiently extracting useful information from web pages.

## II. Literacy Review

For most early attempts at content extraction, human intervention was necessary to identify the key components of a website, such as [1], [9], and [10]. While these techniques

To be precise, they were not easily scaleable to data collection in mass. Other early approaches used various NLP techniques [7] to discover linkages between web page zones or made use of HTML elements to distinguish different textual sections [14]. Kushmerick created a technique to just recognise and eliminate the advertising on a page [11]. The Document Object Model (DOM) is often used in methods to extract structured HTML data from websites [3, 4]. Programmes and scripts can access and edit the information and structure of documents dynamically thanks to DOM's "platform- and language-neutral interface." [13]. While Mantratzis et al. created an algorithm that recursively looks through a DOM tree to detect which HTML tags have a high density of hyperlinks in [20], Gongquing et al. used a DOM tree to help determine a text-to-tag path ratio in [2].

In the form of templates, layouts can be found on many different web pages. According to Bar-Yossef et al.'s findings in [22], when these templates are identified and the comparable information throughout several web pages is removed, what is left is the different content, which may be the main article itself. In [15], Chen et al. investigated a technique for finding the template by combining layout grouping with word indexing. Kao et al. created an algorithm that used a website's features, links, and content's entropy to help identify template portions in [16] and [23]. Additionally, Yi et al. (17) proposed a technique for categorising and grouping web content using a style tree to compare various website structures and identify the template being used. The quantity of words and link density of a website can be used to identify boilerplate (any section of a website that is not deemed core content), according to a method devised by Kohlschütter [30] called Boilerpipe.

A lot of research tends to expand on the findings of earlier studies. Gottron adapted the document slope curve algorithm from [18] and offered a comparison of several of the content extraction algorithms in use at the time in [5]. Within his test group, the adjusted document slope curve performed the best. In [18], Pinto et al. built on the Body Text Extraction work from [14] by identifying content vs. non-content pages using a document slope curve in an effort to determine whether or not a web page included text that was worthwhile being extracted. The algorithms ContentExtractor [21] and FeatureExtractor [22] were developed by Debnath et al. They examined the similarity of blocks on various web sites and categorised sections as content based on a user-defined desired feature set.

No algorithm has yet been able to extract all important text from webpages with 100Some algorithms might function properly on some webpages but not others. In terms of extracting website content, there is still a lot of work to be done.

## III. Methodology

There are multiple steps in the methods for extracting web information using paragraph tags and natural language processing:

### A. Identify the target website:

Determine the website from which you wish to extract content first. Any website that organises its content using HTML paragraph tags could be this.

### B. Retrieve and Parse the HTML code:

Once you have located the target website, you must use a web scraper or other similar tool to extract the HTML code from the page. All of the website's content, including paragraph-tagged content, will be contained in this HTML code. The HTML code must then be parsed in order to retrieve only the material that is formatted with paragraph tags. Several libraries and programmes, including BeautifulSoup and lxml, can be used to accomplish this.

### C. Pre-process the extracted content:

Once you have extracted the content from the paragraph tags, you need to pre-process it to remove any unwanted characters, symbols, or HTML tags that may be present.

### D. Apply natural language processing techniques:

With the pre-processed content in hand, you can now apply natural language processing techniques to extract insights and information from the text. This could involve tasks such as sentiment analysis, topic modeling, or named entity recognition.

### E. Analyze and visualize the results:

In order to obtain understanding of the website's content, you may finally analyse and visualise the outcomes of the natural language processing algorithms. Visualisations like word clouds, topic maps, or sentiment analysis charts might be created in this process.

## IV. Related Method

The extraction of HTML web content using paragraph tags and natural language processing includes numerous processes, and the code for each step is dependent on the programming language and tools used. Here's a high-level overview of the stages involved in extracting online content with Python and several common libraries:

### A. Import the required libraries:

We start by importing the necessary Python libraries that we will be using in this code example. We import the requests library to send a GET request to the website, the BeautifulSoup library to parse the HTML content, and the NLTK library to perform NLP tasks.

### B. Specify the URL of the website to scrape :

The URL of the website to be scraped is then specified. We are scraping the Wikipedia page on natural language processing in this case.

### C. Send a GET request to the website and store the response in a variable:

We submit a GET request to the website and save the response in a variable using the requests library.

### D. Use BeautifulSoup to parse the HTML content of the page and extract all the paragraph tags:

BeautifulSoup is used to parse the HTML content of the page and extract all paragraph elements. The outcome is saved in a variable called paragraphs.

### E. Loop through each paragraph and extract the text content:

Using the 'text' attribute, we loop through each paragraph tag and extract the text content. We concatenate all of the text content into a single string and save it in the variable 'text'.

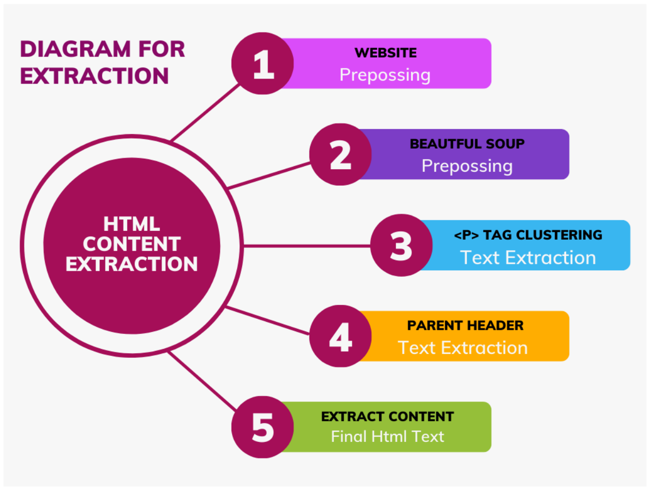

Fig. 1. Flow-diagram of the presented ParEx method

### F. Tokenize the text content:

Using $NLTK's'word_tokenize'$ function, we tokenize the text content into individual words. Then we remove all stop words from the token list.

### G. Perform named entity recognition on the cleaned tokens using NLTK:

We execute named entity recognition (NER) on the cleaned tokens by first assigning part-of-speech tags to each token with the $NLTK's'pos_tag'$ function, and then doing NER on the tagged tokens with the $'ne_chunk'$ function.

### H. Print the named entities:

We loop through the NER tags, checking if each entity has a label and if the label is 'NE' (named entity). If the object has a 'NE' label, we output it to the console.

### I. Extracting Named Entities of Organization Type from Paragraphs

Extracting named entities of organization type from paragraphs involves using Natural Language Processing (NLP) techniques to identify and extract specific types of entities from text. In this case, the code uses spaCy to identify named entities of organization type from paragraphs extracted using BeautifulSoup.

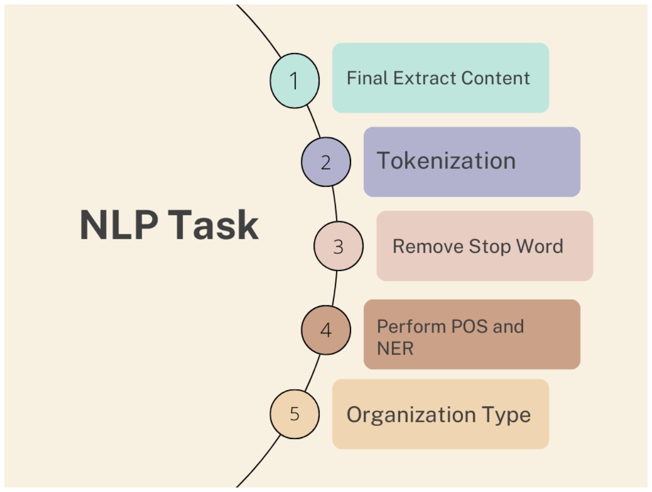

Fig. 2. Flow-diagram of NLP Task

## V. Result

I used Python tools such as requests, BeautifulSoup, nltk, and spacy to extract information from the Wikipedia page on natural language processing. The code first downloads the page's HTML content, then uses BeautifulSoup to extract the text content of each paragraph.

Following that, the code employs spaCy to detect named entities of the organisation type and publishes information about those entities, as well as the original text content and the POS and NER tags. The code specifically prints the following information:

- Text: The original text content of each paragraph that contains named entities of organization type.
- Organizations: The named entities of organization type that were identified in each paragraph.
- Named Entities: The named entities that were identified in each paragraph, regardless of their type.
- POS tags: The part-of-speech tags for the cleaned tokens in each paragraph.
- NER tags: The named entity recognition tags for the cleaned tokens in each paragraph.

Overall, the work shows how to utilise Python modules to extract information from a webpage and then execute basic natural language processing tasks on the collected text. The findings can be utilised to obtain insight into the types of organisations referenced in the Wikipedia article on natural language processing, as well as the most often cited named entities across the page.

## VI. Conclusion

This code demonstrates how to extract and analyse text information from a Wikipedia article on Natural Language Processing using popular Python modules like as requests, BeautifulSoup, nltk, and spacy. The script downloads the page's HTML content, extracts the text content of each paragraph, and employs spaCy to find named entities of organisation type. On the cleaned tokens in each paragraph, the script also does part-of-speech tagging and named entity recognition. The script's output can be used to get insights about the types of organisations referenced in the Wikipedia page on Natural Language Processing, as well as the most commonly cited named entities across the page. Additional information, such as the frequency of each named entity or the co-occurrence of named entities with specific keywords, can also be included in the output.

## References

[1] Carey, Howard and Manic, Milos. (2016). HTML web content extraction using paragraph tags. 1099-1105. 10.1109/ISIE.2016.7745047.
[2] T. Weninger, W.H. Hsu, "Text Extraction from the Web via Text-to-Tag Ratio," in Database and Expert Systems Application, pp.23-28, Sept. 2008.
[3] T. Weninger, W.H. Hsu, J. Han, "CETR: content extraction via tag ratios," in Proc. Intl. conf. on World wide web, pp. 971-980, April 2010.
[4] T. Gottron, "Evaluating content extraction on HTML documents," in Proc. Intl. conf. on Internet Technologies and Apps, pp. 123-132. 2007.
[5] D. Song, F. Sun, L. Liao, "A hybrid approach for content extraction with text density and visual importance of DOM nodes,"in Knowledge and Information Systems, vol. 42, no. 1, pp. 75-96, 2015.
[6] A.F.R. Rahman, H. Alam, R. Hartono, "Content extraction from html documents," in Intl. Workshop on Web Document Analysis, pp. 1-4, 2001.

## VII. Summary

- Choose the website or web page that you wish to extract data from.
- Use web scraping applications like BeautifulSoup or Scrapy to extract the target web page's HTML code.
- To extract the text content included within the HTML paragraph elements, use HTML parsing techniques.
- To clean up the retrieved text data, use text pre-processing techniques like stemming, stop-word removal, and tokenization.
- Utilise NLP methods like named entity identification, sentiment analysis, and part-of-speech tagging to analyse the cleaned text data.
- Put the extracted data in a structured format, such a database or spreadsheet, for additional analysis and visualisation.
- Validate and confirm the accuracy of the extracted data by contrasting it with the original source and, if necessary, cross-referencing with additional sources.
- To ensure the precision and applicability of the extracted data, continuously monitor and update the extraction process.

### *A. Code Summary*

The code performs the following tasks:

- Imports necessary libraries including requests, $BeautifulSoup, nltk, spacy, stopwords,$ and $wordtokenize.$
- Downloads various nltk resources including $stopwords, punkt, words,$ $averagedperceptrontagger,$ and $maxentnechunker.$
- Loads the $encorewebsm$ model from the spacy library for named entity recognition.
- Scrapes the content of a Wikipedia page on natural language processing using requests and BeautifulSoup.
- Extracts all paragraphs on the Wikipedia page using BeautifulSoup.
- Concatenates the text from all paragraphs into a single string.
- Removes stop words from the text using nltk's set of stopwords.
- Tokenizes the cleaned text into individual words.
- Tags the parts of speech for each token using nltk's pos tag function.
- Performs named entity recognition using nltk's ne chunk function and prints the named entities identified in the text.
- For each paragraph on the Wikipedia page, extracts named entities of organization type using spacy's named entity recognition model and prints the text of the paragraph and the identified organizations.