# A MACHINE LEARNING APPROACH TO WEB-PAGE CONTENT EXTRACTION

Presented By

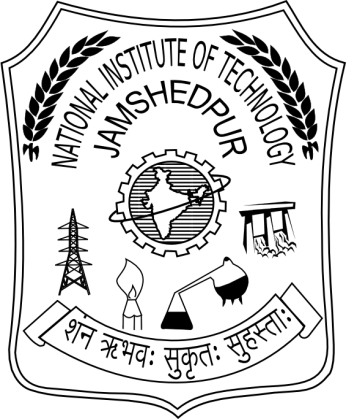
AWANISH KUMAR GUPTA RISHIKESH MAURYA SUMIT KUMAR TIWARY

272/11 260/11 131/11

B.Tech(H), VIIIth Semester B.Tech(H), VIIIth Semester B.Tech(H), VIIIth Semester Computer Science &Engg. Computer Science &Engg. Computer Science &Engg.

NIT Jamshedpur NIT Jamshedpur NIT Jamshedpur

awanishkumargupta28@rediffmail.com [rishi.nit.iim@gmail.com](mailto:rishi.nit.iim@gmail.com) sumitkrt@yahoo.in



**NATIONAL INSTITUTE OF TECHNOLOGY**

**JAMSHEDPUR, INDIA – 831014**

A REPORT

ON

**A Machine Learning Approach To Web-Page Content Extraction**

**PROJECT REPORT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT OF THE COURSE**

**MAJOR PROJECT (CS806)**

**Under Guidance: Project Guide – Miss Arpita Sarkar**

**Certificate**

This is to certify that the project report titled **A Machine Learning Approach To Web-Page Content Extraction** submitted by **Sumit Kumar Tiwary, Awanish Kumar Gupta** and **Rishikesh Maurya** in partial fulfillment of the requirements of course CS806 (Major Project) **,** as part of the degree of Bachelor of Technology in **Computer Science & Engineering** of **National Institute of Technology, Jamshedpur**, *session 2014–2015* is a record of confide work carried out under my supervision and has not been submitted anywhere else for any other purpose.

Mr. Rajiv Ranjan Suman Miss Arpita Sarkar

HOD Lecturer (Project Guide)

Deptt. Of Computer Science and Engg. Deptt. Of Computer Science and Engg.

**Acknowledgement**

We would like to take this opportunity to express our gratitude towards all those who have in various ways helped us in successful completion of the project.

We wish to take this opportunity to express our sincere gratitude and deep sense of respect to our beloved director, **Dr. Rambabu Kodali,** for making us available all the required assistance and for his support and inspiration to carry out this proposed topic in the Institute.

We express heartfelt thanks to our HOD **Mr. Rajiv Ranjan Suman** for providing us with all necessary infrastructure and thereby giving us freedom to carry out the proposed topic.

A deep sense of gratitude to **Miss Arpita Sarkar** for providing us the opportunity to work on this project and always providing us with a helping hand wherever things got stuck and confusing , in spite of being quite busy with her hectic schedules**.** We are grateful for her guidance, encouragement, understanding and insightful support. Her technical guidance expertise and immense help have largely contributed to the success of the proposed topic.

Finally we would like to thank our friends for their cooperation in completing this proposed topic.

**Abstract**

Web pages often contain clutter (such as unnecessary images and extraneous links) around the body of an article that distracts a user from actual content. Extraction of “useful and relevant” content from web pages has many applications, including cell phone and PDA browsing, speech rendering for the visually impaired, and text summarization. Most approaches to making content more readable involve changing font size or removing HTML and data components such as images, which takes away from a webpage’s inherent look and feel. Our key insight is to work with DOM trees, a W3C specified interface that allows programs to dynamically access document structure, rather than with raw HTML markup. Apart from the main content, a webpage usually also contains boilerplate elements such as navigation panels, advertisements and comments. These additional elements are typically not related to the actual content and can be treated as noise that needs to be removed properly to improve the user’s reading experience. This is a difficult problem as HTML is loose in semantics and flexible in structure. In this paper, we model the webpage content extraction problem as a classification problem and employ machine learning method to solve it. For each text block in the HTML document, we select a set of relevant features, based on which an SVM classifier is used to predict whether this text block is content or non-content.

**Keywords.** DOM trees, content extraction, HTML documents

**Contents**

**CERTIFICATE……………………………………………….......................................................................1**

**ACKNOWLEDGEMENT……………………………………......................................................................2**

**ABSTRACT……………………………………………………....................................................................3**

**CHAPTER 1**

**ABOUT THE PROJECT…….................................................................................................................5**

**CHAPTER 2**

**RELATED WORKS................................................................................................................................6**

**CHAPTER 3**

**PROJECT DEVELOPMENT APPROACH.............................................................................................7**

**CHAPTER 4**

**OUR APPROACH………………………………………….........................................................................9**

**CHAPTER 5**

**IMPLEMENTATION.…..........................................................................................................................10**

**CHAPTER 6**

**FUTURE DIRECTIONS.….....................................................................................................................12**

**CHAPTER 7**

**CONCLUSION………………………………………………..…..................................................................13**

**REFERENCES…………...……………………………………..................................................................14**

Chapter 1

**About The Project**

Web pages are often cluttered with distracting features around the body of an article that distract the user from the actual content they’re interested in. These “features” may include pop-up ads, flashy banner advertisements, unnecessary images, or links scattered around the screen. Automatic extraction of useful and relevant content from web pages has many applications, ranging from enabling end users to accessing the web more easily over constrained devices like Personal Digital Assistants (PDAs) and cellular phones to providing better access to the Web for the disabled.

Most traditional approaches to removing clutter or making content more readable involve increasing font size, removing images, disabling JavaScript, etc., or a combination of these methods, all of which eliminate the webpage’s inherent look-and-feel. We propose a “Content Extraction” technique that can remove clutter without destroying webpage layout, making more of a page’s content viewable at once. These techniques should also work on web pages made up of multiple content bodies, even if they are separated by the distracting features or with them interspersed within the different sections of content.

Content extraction is particularly useful for the visually impaired and blind. A common practice for improving web page accessibility for the visually impaired is to increase font size and decrease screen resolution; however, this also increases the size of the clutter, reducing effectiveness.

Content extraction allows the algorithms to process only the extracted content as input as opposed to cluttered data coming directly from the web.

While many algorithms for content extraction already exist, it appears that few working implementations can be applied in a general manner. Our solution employs a series of techniques that address the aforementioned problems, and makes it easy to implement and experiment with additional algorithms.

In order to analyze a web page for content extraction, we pass web pages through an open source HTML parser, which creates a Document Object Model (DOM) tree. The Document Object Model [(www.w3.org/DO](http://www.w3.org/DOM)M) is a standard for creating and manipulating in-memory representations of HTML (and XML) content. By parsing a webpage's HTML into a DOM tree, we can not only extract information from large logical units but can also manipulate smaller units such as specific links within the structure of the DOM tree. In addition, DOM trees are highly transformable and can be easily used to reconstruct a complete webpage. Finally, increasing support for the Document Object Model makes our solution widely portable.

Chapter 2

**Related Works**

There is a large body of related work in content identification and information retrieval that attempts to solve similar problems using various other techniques. Many approaches have been suggested for formatting web pages to fit on the small screens of cellular phones and PDAs. For instance, the Opera browser uses the handheld CSS media type. One method is to transform a web page into a hierarchy of individual content units called Semantic Textual Units, or STUs. First, STUs are built by analyzing syntactic features of an HTML document, such as text contained within paragraph (<P>), table cell (<TD>), and frame component (<FRAME>) tags. These features are then arranged into a hierarchy based on the HTML formatting of each STU. STUs that contain HTML header tags (<H1>,<H2>, and <H3>) or bold text (<B>) are given a higher level in the hierarchy than plain text. This hierarchical structure is finally displayed on PDAs and cellular phones, but typically showing different content than the original work.

None of these concepts solve the problem of automatically extracting just the content, although they do provide simpler means in which the content can be found. These approaches perform limited analysis of web pages themselves and in some cases information is lost in the analysis process. By parsing a webpage into a DOM tree, we have found that one not only gets better results but has more control over the exact pieces of information that can be manipulated while extracting content.

Some of the existing algorithms are based on the observation that compared to the content fragments, the non-content fragments of an HTML document are usually highly formatted with more tags and also contain less text and shorter sentences. As a result, non-content fragments have a higher tag density while the content fragments have a higher text density. (Kullback-Leibler-divergence), among which the number of words and link density are proved to be the most relevant ones.

Alternatively, when a web page is divided into a tree whose nodes are visually grouped blocks, spatial and content features may also be used to detect the non-content nodes in this tree as in. This approach suffers from high computational cost to render a page in order to analyze it. There are also some other approaches like template detection algorithms which classify the identical parts found in all webpages as noisy components . Apparently, the application of these algorithm is limited to webpages from the same website, and thus it would be cumbersome to build models and templates for different websites or different versions of one website.

In this project, we adopted webpage context extraction as a classification problem on the block level. For this classification problem, we have selected three types for features: text features, relative position and id & class token feature and used SVM as classifier. The blocks classified as content would then be merged to construct ”clean” webpages,.

Chapter 3

**Project Development Approach**

We have used Iterative and Incremental Development model (IID) for our project development. This development approach is also referred to as Iterative Waterfall Development approach. Iterative and Incremental Development is a software development process developed in response to the more traditional waterfall model.

**Life Cycle:**



Figure 3.1: Iterative and Incremental Life Cycle

The basic idea behind iterative enhancement is to develop a software system incrementally, allowing the developer to take advantage of what was being learned during the development of earlier, incremental, deliverable versions of the system. Learning comes from both the development and use of the system, where possible. Key steps in the process were to start with a simple implementation of a subset of the software requirements and iteratively enhance the evolving sequence of versions until the full system is implemented.

At each iteration, the procedure itself consists of the Initialization step, the Iteration step, and the Project Control List. The initialization step creates a base version of the system. The goal for this initial implementation is to create a product to which the user can react. It should offer a sampling of the key aspects of the problem and provide a solution that is simple enough to understand and implement easily. To guide the iteration process, a project control list is created that contains a record of all tasks that need to be performed. It includes such items as new features to be implemented and areas of redesign of the existing solution. The control list is constantly being revised as a result of the analysis phase.

The iteration involves the redesign and implementation of a task from project control list, and the analysis of the current version of the system. The goal for the design and implementation of any iteration is to be simple, straightforward, and modular, supporting redesign at that stage or as a task added to the project control list. The code can, in some cases, represent the major source of documentation of the system. The analysis of an iteration is based upon user feedback, and the program analysis facilities available. It involves analysis of the structure, modularity, usability, reliability, efficiency, and achievement of goals. The project control list is modified in light of the analysis results.

Chapter 4

**Our Approach**

Our solution employs multiple extensible techniques that incorporate the advantages of the previous work on content extraction and attempts to avoid the common pitfalls like noisy results and slow performance.

In order to analyze a web page for content extraction, the page is first passed through an HTML parser that corrects HTML errors and then creates a DOM tree representation of the web page. (HTML on the Internet can be extremely malformed and most popular browsers like Internet Explorer and Mozilla are able to handle incorrect HTML by making the closest guess to what the HTML should be.) Once parsed, the resulting DOM document can be seamlessly shown as a webpage to the end-user by flattening the tree and producing back the HTML. This process accomplishes the steps of structural analysis and structural decomposition is analogous to those done by several other techniques. The DOM tree is hierarchically arranged and can be analyzed in sections or as a whole, providing a wide range of flexibility for our extraction algorithm, our content extractor navigates the DOM tree recursively, using a series of different filtering techniques to remove and adjust specific nodes and leave only the content behind.

There are two sets of filters that we have implemented. The first set of filters simply ignores tags or specific attributes within tags. With these filters, images, links, scripts, styles, and many other elements can be quickly removed from the web page. However, the second set of filters is more complex and algorithmic, providing a higher level of content extraction. This set, which can be extended, currently consists of the advertisement remover, the link list remover, the removed link retainer and the empty table remover. We have found that numerous tables that are either completely empty or have several empty cells take up large swaths of space remain on the webpage. The empty table remover removes tables that are empty of any “substantive” information. This does not require much prior knowledge of HTML since the syntax of the markup language is simple and matches words from the English language closely, e.g., table, form, etc. The table remover checks a table for substance after it has been parsed through the filter. If a table has either no substance or less than some user defined threshold, it is removed from the tree. This algorithm effectively removes any tables left over from previous filters that contain small amounts of unimportant information. This filter is typically run towards the end to maximize its benefit.

After the DOM tree is completely parsed, the list of removed links is added to the bottom of the page. In this way, any important navigational links that were previously removed remain accessible, and since the parser had parsed them initially as separate units, each menu or navigational link is kept intact and they can all be viewed without any loss of original setup or style.

After the entire page is parsed and modified appropriately, it can be output in either HTML or as plain text . The plain text output removes all the tags and retains only the text of the site, while eliminating most white space.Our algorithm doesn’t technically find the content but instead eliminates likely non-content. In this manner, we can still process and return results for sites that don’t have an explicit “main body”.

It, however, does have some limitations:

1) It cannot filter non-HTML content like Flash.

2) Dynamically generated pages often aren't filtered so nicely.

Chapter 5

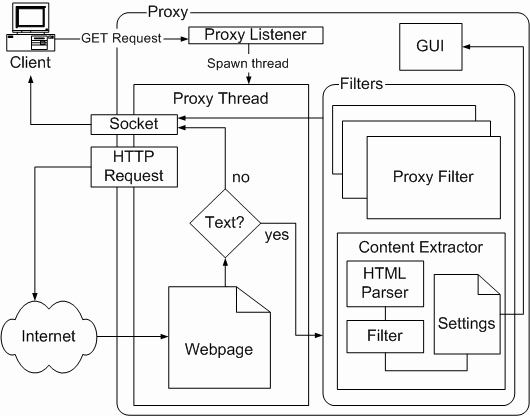
**Implementation**

5.1.1. *Overview*

The current implementation is in Java for cross-platform support, and has been successfully tested .The Content Extraction framework itself has a complexity of O(N), where N is the number of nodes in the DOM tree after the HTML page is parsed. During tests, the algorithm performs quickly and efficiently because of the corrections automatically applied while the page is parsed into a DOM tree. However, sites that are extremely link heavy produce bad results; when the link to text ratio approaches 100%, we experienced anomalous behavior. Depending on the type and complexity of the web page, the content extraction suite can produce a wide variety of output. The algorithm performs well on pages with large blocks of text such as news articles and mid-size to long informational passages. Most navigational bars and extraneous elements of web pages such as advertisements and side panels are removed or reduced in size. When printed out in text format, most of the resulting text is directly related to the content of the page, making it possible to use summarization and keyword extraction algorithms efficiently and accurately.

The initial implementation is designed for simplicity in order to test and design content extraction algorithms. Additionally, we should mention that there isn’t any sort of preservation of objects that may be lost after the HTML is passed through our parser, except links can be retained as explained above.

5.1.2. *Implementation details*



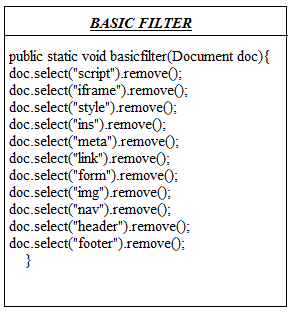
**Figure 5.1**

The life cycle of the process that gets a page to the client’s browser through the proxy from a very high level is - the client passes a request for the webpage to the proxy which opens a socket, fetches the original content of the page, and parses the page to create a DOM tree representation. It is then passed through the different filters . The edited DOM tree is then either flattened into the HTML form, to be sent back to the client’s browser, or stripped of all HTML tags and only the text content is sent to the client for rendering. An architectural diagram is shown in Figure 5.1.

In more detail, in order to analyze a web page for content extraction, the page is passed through an HTML parser that creates a Document Object Model tree. The algorithm begins by starting at the root node of the DOM tree (the <HTML> tag), and proceeds by parsing through its children using a recursive depth first search function . The function initializes a Boolean variable to true to allowto check the children. The currently selected node is then passed through a filter method analyzes and modifies the node . At any time variable can be set to false, which allows the individual filter to prevent specific subtrees from being filtered. That is, certain filters can elect to produce the final result at a given node and not allow any other filters to edit the content after that.

The filtering method begins by examining the node it is passed to see if it is a “text node” (data) or an “element node” (HTML tag). Element nodes are examined and modified in a series of passes. First, any filters that edit an element node but do not delete it are applied.

The second phase in examining element nodes is to apply all filters that delete nodes from the DOM tree. We have used Jsoup for this purpose. Jsoup is a java html parser. It is a java library that is used to parse HTML document. Jsoup provides api to extract and manipulate data from URL or HTML file. It uses DOM, CSS and Jquery-like methods for extracting and manipulating file. The org.jsoup.Jsoup class provides methods to connect, clean and parse the HTML document.



Chapter 6

**Future Directions**

Currently we do not do any form of learning of a user’s browsing habits. It may be possible to implement artificially intelligent heuristic algorithms, such as Bayesian learning or Markov Model creation, as a browser plug-in that reads metadata from the client about how to change the settings. Such a browser plug-in might provide an interface for the user to rate pages. With the addition of trainable filtering, it could adapt to a particular user's or group’s preferences. Even basic control from the browser, without any AI, would enhance usability.

Finally, one of our main goals was to expose a simple API for programmers to extend, so that current and future natural language processing and information retrieval algorithms can easily be added. This would allow users to truly be able to customize the content they would like to view on visited web pages.

Chapter 7

**Conclusion**

Many web pages contain excessive clutter around the bodies of one or more articles, the actual content of the page. Although much research has been done on content extraction, and there are many special-case solutions to remove advertising (particularly pop-ups) or reformat for small screens, it is still a relatively new field where few general purpose tools are available so most researchers must construct their content extractors from scratch. Our approach, working with the Document Object Model tree as opposed to raw HTML markup, enables us to apply in tandem an extensible collection of Content Extraction filters, and potentially other kinds of filters such as format translators and NLP summarizers. The heuristic filters that we have developed to date, though simple, are quite effective.

**References**

1. <http://stackoverflow.com/questions/19930737/jsoup-efficient-way-to-remove-html-elements-and-childrens>
2. <http://stackoverflow.com/questions/10111511/how-i-can-traverse-the-html-tree-using-jsoup>
3. <http://docs.oracle.com/javase/7/docs/api/org/w3c/dom/Element.html>
4. <http://stackoverflow.com/questions/3103962/converting-html-string-into-dom-elements>
5. <http://www.javatpoint.com/jsoup-tutorial>
6. <https://docs.oracle.com/javase/7/docs/api/org/w3c/dom/Node.html>
7. <http://docs.oracle.com/javase/7/docs/api/javax/swing/text/Document.html#getDefaultRootElement()>
8. <http://jsoup.org/cookbook/extracting-data/dom-navigation>
9. <http://www.w3schools.com/jsref/prop_node_childnodes.asp>
10. <http://jsoup.org/apidocs/org/jsoup/select/Elements.html>
11. <http://www.java2s.com/Code/Java/XML/PrintTreenode.htm>
12. <https://docs.oracle.com/javase/7/docs/api/org/w3c/dom/package-summary.html>
13. <http://en.wikipedia.org/wiki/Document_Object_Model>
14. Yao Zuo - A Machine Learning Approach To Web-page Content Extraction, Stanford University
15. Zhou Mashuq – Web Content Extraction Through Machine Learning
16. Automating Content Extraction of HTML Documents , Suhit Gupta , Peter Grimm and Justin Starenn , Columbia University
17. Java :The Complete Reference, [Schildt](http://www.google.co.in/search?tbo=p&tbm=bks&q=inauthor:%22Schildt%22) Tata McGraw-Hill Education