Bauhaus-Universität Weimar
Faculty of Media
Degree Programme Computer Science for Digital Media

Advancing and Benchmarking Large-Scale Content Extraction from the Web

Master's Thesis

Sanket Gupta
Born June 25, 1994 in Kawardha, India

Matriculation Number 121946

1. Referee: Prof. Dr. Benno Stein

Submission date: November 4, 2022

Declaration

Unless otherwise indicated in the text or references, this thesis is entirely the product of my own scholarly work.

Weimar, November 4, 2022

.....

Sanket Gupta

Abstract

Websites are essential information sources. A webpage contains main content, which is the most relevant part of the web page, such as news articles, sports events, informative posts, as well as boilerplate text such as header, footer, advertising, and copyright notice. Content Extraction is the process of locating and extracting a webpage's main content, distinguishing between the main content and the boilerplate. It is important to separate the main content from the boilerplate because it improves text analysis for unstructured and semi-structured data and lets us use automated systems to draw more accurate conclusions about our business problems, like displaying only main content on smaller devices, improving search engine results, minimising storage costs, lack of data mobility, and complex data management.

Significant research has gone into the development of content extraction algorithms. However, little research has been conducted on assessing the quality of content extraction algorithms to identify the most effective extractors for various web pages.

In the thesis, we examined various open-source content extractors, designed an ensemble-based content extractor, and benchmarked the extractors against a gold standard using text-based similarity metrics. We also discussed collecting labelled web pages from various sources, analyzing annotation errors in the gold standard, putting web pages into groups based on their complexity or using the K-means clustering algorithm, and benchmarking scores for these groups. We discovered that no single content extraction algorithm outperforms the rest. The best content extraction algorithm varies with the complexity of a web page.

Contents

1 Introduction **1**

1.1 Motivation 3

1.2 Goals & Approach 4

2 Background **6**

2.1 Related Work 6

2.1.1 Statistics-based Content Extraction 6

2.1.2 Machine Learning Based Content Extraction 9

2.1.3 Visual Based Content Extraction 10

2.1.4 Benchmarking of Extractors 11

2.2 Extractors 12

3 Creating and Analyzing Datasets for Benchmarking Extractors **17**

3.1 Collecting Annotated Web Pages for Benchmarking Extractors 17

3.2 Data Preparation 22

3.3 Classifying Web Pages used for Benchmarking to Determine Diversity . . 23

3.4 Catalogue and Quantify Gold Standard Manual Annotation Error 26

3.5 Grouping Web Pages by the Complexity 31

3.6 Grouping Web Pages using the K-means Clustering Algorithm 33

4 Surveying Text Similarity Metrics for Benchmarking Extractors **39**

5 Designing Ensemble Model Extractors **44**

5.1 Threshold based Ensemble Content Extractor 45

5.2 Machine Learning based Ensemble Content Extractor 46

6 Evaluation **49**
6.1 Benchmarking of Extractors 49
6.1.1 Benchmarking using the Source Dataset 49
6.1.2 Benchmarking using the Dataset Formed by Web Pages Complexity 53
6.1.3 Benchmarking using the Dataset Formed by K-Means Clustering Algorithm 56
6.2 Extractors Rank Correlation Among Text Similarity Metrics 59
6.3 Most Effective Score Distribution Across Web Pages for all the Benchmarked Extractors 63

7 Conclusion and Future Works **65**

Bibliography **67**

Acknowledgements

I am thankful to Dr. Johannes Kiesel, Mr. Janek Bevendorff and Mr. Nikolay Kolyada for guiding me throughout my thesis. They were consistently supportive and generous with their knowledge and skills, and their assistance and direction throughout the duration of the thesis were unparalleled. I would like to thank Prof. Dr. Benno Stein for sharing his knowledge of Machine Learning, which was of immense benefit to me as I worked on my thesis. Lastly, I would like to express my gratitude to my friends and family for their support.

Chapter 1

Introduction

The World Wide Web is a tremendous informational resource. The number of websites has expanded dramatically, from 17 million in 2000 to over 1 billion in 2018[1]. Approximately 63 percent of the population has access to the World Wide Web[2]. Scientists, educational institutions, and business intelligence, to mention a few, use the information available on the web for their purpose. Internet penetration is phenomenally high. Approximately 80% of the internet's information is unstructured and has enormous potential[3]. Unstructured data presents several difficulties, including data indexing and search results prone to inaccuracy, costly data storage, and complex data processing. As the volume of data is increasing at an alarming rate, it becomes impossible for humans to process information effectively and precisely on a large scale. Natural Language Processing lets users quickly extract meaningful information from a vast corpus of unstructured data. Before employing natural language processing techniques, eliminating possible boilerplate from unstructured data can be of immense value. The boilerplate is the sections of web pages that are of less interest to the reader, such as the header, footer, and advertisements, to name a few. Therefore, it is crucial to filter, organize, locate, and manage unstructured data on the Internet.

Each webpage is intended to convey specific information. A sports webpage, for instance, features sports-related events and includes advertisements, copyright, and a footer. A reader focuses primarily on the sports story and disregards the other less critical components. These additional sections provide meta information, such as contact or mailing

[1]https://siteefy.com/how-many-websites-are-there/
[2]https://www.axios.com/2021/12/03/world-internet-usage-un
[3]https://blog.bitext.com/the-bulk-of-data-wandering-on-the-net-is-unstructured-data

information, connections to related pages, etc.

The following are the primary components of a web page (Insa Cabrera et al. [2013]):

- **Main content**: The content is the most important to the web page's purpose.
- **Navigation / Header**: A navigation menu lists links to additional web pages, typically internal sites. They do not contribute to the main content but provide connections and descriptions of additional user-relevant material.
- **Advertisements**: As a method of monetizing websites, the author typically employs advertisements. These sections are typical of no relevance to the user, but they contain information based on the reader's recent search history.
- **Footer**: The footer is the portion at the bottom of the website that contains a copyright notice, social media connections, contact information, business information, and other related information.

Content Extraction is the process of identifying and extracting the webpage's primary textual content. There are two techniques for content extraction: manual and automated. Manual content extraction is when a reader visually examines a webpage, identifies significant text, and extracts it. The process of discovering and extracting relevant information using software or any other automated technique is called automated content extraction. In manual content extraction, for example, when a user visits a webpage, the user glances over the page and attempts to find the most interesting sections. Background information, layout interpretation, and saccades allow users to locate their desired content. However, a machine does not have such resources to determine the main content. Figure 1.1, a snapshot from a website, depicts the previously stated scenario. A reader will have no issue recognizing the webpage's main content; one can ignore the navigation menu at the top and the advertising on the right.

Most algorithms determine the primary content based on statistics such as stop word density, link density, HTML tags, length of text, etc., while some utilize machine learning. Automatic content extraction offers a more cost- and time-effective alternative to manual content extraction. As a result, benchmarking the extractors is required to assess their performance in various contexts.

Figure 1.1: A web browser snapshot of makeawebsitehub.com. There are various sections on the webpage, navigation at the top, the primary content on the left, and an advertisement and connections to relevant articles on the right. The reader can readily distinguish between the main content and the additional components.

1.1 Motivation

The exponential rise of the Internet has resulted in an abundance of semi-structured and unstructured data. Identifying relevant information from a web page can be used for a variety of applications, including but not limited to presenting relevant information on smaller devices, improving search engine results, and data mining. For consumers, businesses, and researchers to save time and money, reduce manual errors, enhance their data-driven operations, and avoid duplication of effort, it is crucial to extract the main content for processing information on a large scale.

There are research articles on improving the performance of web-mining and information retrieval from web pages, comparing results with standard extractors. However, little research has been conducted on benchmarking these extractors on a large scale, the scenarios in which different extractors function, the quality of labelled datasets used for benchmarking, and scoring metrics. Such an analysis can be significant when determining which extractor to use to extract the main content from a web page.

1.2 Goals & Approach

The objective of the thesis is to fill the research gap and provide a systematic method for benchmarking extractors. It's main contribution are:

- **Collection of 15 content extraction algorithms**: We analyzed generic content extraction algorithms that do not require domain-specific customization or are designed to extract specific text/tags from web pages. We accumulated 15 algorithms published in scientific journals and on GitHub, and the algorithms were coded in Python, Java, Go, and Scala.

- **Collection of web pages for benchmarking extractors**: Labelled web pages were gathered from multiple sources. The sources include the cleaneval dataset, readability, dragnet, and article-extraction-benchmark GitHub repositories. Sun et al. labelled around 700 web pages manually, and we contacted the author to allow access to their manually labelled dataset. Around 3100 archived webpages and their gold standard were collected from various sources.

- **Exploratory data analysis on a labelled dataset**: We conducted exploratory data analysis on the web pages and their gold standard. The error rate in the gold standard, the proportion of text recognized as the main content on a web page, K-Means cluster algorithm analysis to detect cluster formation, classifying web pages by categories, and estimating the complexity of a web page were all addressed. It is discussed in depth later in the thesis.

- **Largest empirical evaluation of content extractors**: The extractor's output on a web page was compared to its gold standard using text similarity score metrics. 15 extractors were evaluated using the score metrics 4-Gram, Levenshtein Edit Distance, Jaccard Index, RougeLSum, and Bag of Words.

- **Design of an ensemble extractor model**: We developed an ensemble model for content extraction and compared its performance on scoring criteria to competing algorithms.

The outcomes of the research are intriguing. We discovered that no single content extraction method outperforms the rest. The best content extraction algorithm varies with the complexity of the web page. In the coming chapters, we shall go into the implications of the findings.

Chapter 2

Background

This chapter reviews content extraction algorithms that are available as open source in the section 2.2. Section 2.1 addresses scientific research addressing statistical, machine learning, and visual-based techniques for content extraction. Section 2.1.4 reviews existing benchmark scores of content extraction algorithms using a text similarity measure published by Konstantin Lopukhin.

2.1 Related Work

HTML tags notify the developer and browser about the content of a web page. HTML offers a highly flexible way to express code; as a result, developers frequently abuse HTML tags while creating websites together with a large amount of unstructured data whose format varies from page to page. There is a discrepancy between how a browser renders a webpage and how the HTML is written. Because of the above obstacles, designing a content extraction algorithm that runs on all websites is difficult. Researchers have explored several methods for extracting the core material; some of the strategies are covered below.

2.1.1 Statistics-based Content Extraction

It includes utilizing rules to determine the main content of a webpage. Various leaf node characteristics are retrieved and compared to a static or dynamic threshold. The majority of existing content extraction methods are statistically based. Several of the characteristics utilized in this method include link density, text density, stop word density, nearby element characteristics, etc. This approach is utilized by algorithms such as trafilatura, readability,

body text extraction, and JustText.

Finn et al. describes Body Text Extraction as the process for identifying a webpage's primary content. Two types of tokens are considered to comprise a webpage: HTML tag tokens and text tokens. Thus, an HTML page may be represented as a sequence of bits B, where Bn = 0 denotes that the i^{th} token is a word and Bn = 1 indicates that it is a tag. The problem is currently viewed as an optimization issue. It needs i and j to be identified to maximize the number of tag tokens below i and above j and the amount of text tokens between i and j. Text is only gathered between i and j. In other words, the algorithm identifies the plateau in the slope curve. The optimization function is defined as:

$$T_{i,j} = \sum_{n=0}^{i-1} B_n + \sum_{n=i}^{j} (1 - B_n) + \sum_{n=j+1}^{N-1} B_n \tag{2.1}$$

Trafilautra's (Barbaresi [2021]) extraction algorithm is based on a cascade of rule-based filters and content heuristics. Content delimitation is performed by XPath expressions targeting common HTML elements, attributes, and irregularities of main content management systems. The selected nodes of the HTML tree are then processed, i.e. checked for relevance (notably by element type, length and link density) and simplified as to their HTML structure. Extraction is robust and modular, providing a trade-off between precision and recall in most settings. Main texts are returned, with optional preservation of structural elements (paragraphs, titles, lists, quotes, code, line breaks, in-line text formatting). Extraction of metadata is also included, such as descending frequency title, site name, author, category, categories and tags. For data extraction, the library acts like a wrapper around html date; a module specifically developed for this task.

JustText (Pomikálek [2011]) employs a straightforward segmentation method. By default, web browsers display the contents of certain HTML elements as blocks. The purpose of these tags is to divide the HTML page into textual sections. BLOCKQUOTE, CAPTION, CENTER, COL, COLGROUP, DD, DIV, DL, DT, FIELDSET, FORM, H1, H2, H3, H4, H5, H6, LEGEND, LI, OPTGROUP, OPTION, P, PRE, TABLE, TD, TEXTAREA, TFOOT, TH, THEAD, TR, UL are all block-level tags. A series of two or more BR tags can also be used to divide blocks.

Multiple observations may be made regarding these blocks:

- Short paragraphs including a link are nearly always boilerplate.
- Any block containing several links is nearly usually a boilerplate.
- Long blocks containing grammatical content are always practical, but all other long blocks are nearly always unproductive.

Both primary blocks and boilerplate blocks tend to form clusters, meaning that additional boilerplate blocks and vice versa surround a boilerplate block. Determining if a text is grammatical or not can be difficult, but a simple heuristic based on the number of stops words can be utilised. While a grammatical text will normally have a specific number of stop words, boilerplate information such as lists and enumerations contain few stop words. The algorithm's primary premise is that lengthy blocks and certain short blocks may be categorised with a high degree of certainty. The remaining short blocks can then be categorised based on their surroundings.

Weninger et al. describes Content Extraction via Text Ratio as a method to extract content text from diverse webpages by using the HTML document's tag ratios. Tag Ratio is the ratio of the count of HTML tags characters to the count of HTML tags per line. A tag ratio histogram is built between the line number and its tag ratio. The histogram is then passed through the Gaussian kernel. It is performed because without smoothing; many content lines will be lost. The content selection is then performed by threshold strategy. The strategy is to identify a threshold that distinguishes between content and non-content regions of tag ratios. Any tag ratio value more than or equal to the threshold should be classified as content, while any tag ratio value less than should be labelled as not content. The difficulty then becomes locating the optimal value for threshold.

Sun et al. [2011] describes Content Extraction by text density, a fast, accurate and general method for extracting content from diverse web pages, and using DOM (Document Object Model) node text density to preserve the original structure. The method makes use of standard text characteristics such as content and noise. It was discovered that web page noise is typically well structured and comprises fewer materials and shorter phrases. In contrast, the text is typically extensive and structured easily. Character number (number of all characters in its subtree) and Tag number (number of all tags in its subtree) are calculated for each node on a webpage. Text Density is then calculated for each node as

the ratio of Character Number to Tag Number. Text Density quantifies the amount of text in each web page node. It assigns high values to nodes that often include lengthy, simply-formatted text and low values to nodes that are heavily formatted and contain less, shorter content. The study determined that most of the page's noise comprises hyperlinks; hence, further statistical information such as LinkCharNumber (number of all hyperlink characters in its subtree) and LinkTagNumber (number of all hyperlink tags in its subtree) was calculated. In addition, Composite Text Density is generated for each node, and content is retrieved using a threshold technique. The objective of the threshold technique is to establish a threshold t that separates nodes into content and noise parts. Any node whose text density is more than or equal to t should be labelled as content, whereas any node whose text density is less than t will be labelled noise.

2.1.2 Machine Learning Based Content Extraction

Web2text (Vogels et al. [2018]) uses machine learning based approach to extract main contents. Web2Text expects input from web pages to be formatted in (X)HTML. Using Jsoup, each document is processed into a Document Object Model (DOM) tree. This DOM tree is preprocessed by deleting empty nodes, nodes containing just whitespace, and nodes whose content cannot be extracted. Web2Text's content extraction algorithm is based on sequence labelling. A web page is treated as a sequence of blocks labelled main content or boilerplate. There are multiple ways to split a web page into blocks, the most popular currently used being lines in the HTML file or DOM leaves. Features are properties of a node that may indicate it being content or a boilerplate. Such features can be based on the node's text, CDOM structure or a combination thereof. It distinguishes between block features and edge features. Block features capture information on each block of text on a page. Edge features capture information on each pair of neighbouring text blocks. It then assigns unary potentials to each text block to be labelled and pairwise potentials to each pair of neighbouring text blocks. The unary potentials are the probabilities that the label li of a text block i is content or boilerplate. The pairwise potentials are the transition probabilities of the labels of a pair of neighbouring text blocks. The two sets of potentials are modelled using convolutional neural network (CNNs) with 5 layers, ReLU non-linearity between layers, filter sizes of (50, 50, 50, 10, 2) for the unary network and of (50, 50, 50, 10, 4) for the pairwise network. CNN receives a sequence of block features corresponding to the sequence of text blocks to be labelled and outputs unary potentials for each block. The pairwise CNN receives an edge class corresponding to edges to label

and output the pairwise potentials. Authors employ dropout regularization with a rate of 0.2 and L2 weight decay with a rate of 10^{-4} to ensure each layer produces a sequence the same size as its input sequence. A total of 128 features were extracted for each block.

2.1.3 Visual Based Content Extraction

In the VIPS algorithm introduced by Cai et al., the content structure of a page is deduced by combining the DOM structure and the visual cues. Each DOM node is checked to determine whether it forms a single block. Visual separators among these blocks are identified, and the weight of a separator is set based on the properties of its neighbouring blocks. The next phase aims to find all appropriate visual blocks contained in the current sub-tree. For each node representing a visual block, its DoC (Degree of coherence) value is set according to its intra-visual difference. DoC indicates the degree of consistency inside the block. This process is iterated until all appropriate nodes are found to represent the visual blocks in the web page. Separators are horizontal or vertical lines in a web page visually crossed with no blocks in the pool. From a visual perspective, separators are good indicators for discriminating different semantics within the page. A visual separator is represented by a 2-tuple: (Ps, Pe), where Ps is the start pixel, and Pe is the end pixel. The weight of a visual separator is assigned based on the distance between its neighbouring blocks. Rules are used to set a weight for each separator and include the following: If background colours are different on two sides of the separator, the weight will be increased. When the structures of the neighbouring blocks are very similar (e.g. both are text), the weight of the separator will be decreased. The construction process starts from the lowest weight, and the blocks beside these blocks are merged to form new virtual blocks. This process iterates till separators with maximum weights are met. After that, each leaf node is checked to see if it meets the granularity requirement. After that, it again goes to the Visual Block Extraction phase to further construct the sub-content structure within that node. If all the nodes meet the requirement, the iterative process is stopped, and the vision-based content structure for the whole page is obtained. The proposed VIPS algorithm takes advantage of visual cues to obtain the vision-based content structure of a web page. Since we trace down the DOM structure for visual block extraction, the algorithm is top-down. The page is partitioned based on visual separators and structured as a hierarchy closely related to how a user would browse the page.

Gottron explains in the article Content Code Blurring: A New Approach to Content

Extraction how to make use of visual characteristics. Typically, additional information is carefully structured and contains a few short sentences. The main text is lengthy and uniformly formatted. The tags correspond to the markup. The text instead contributes the content outside of the tags. The fundamental separation specified by code and content is utilized in two unique methods to create an appropriate document representation. The first method creates a new path for document representations in the context of content extraction by identifying each character, whether it is content or code. A document is therefore regarded as a string of code and content characters. The second method is based on a token sequence, similar to how Body Text Extraction operates. Each tag and word is associated with a token. Consequently, the entire document is represented as a series of tag and word tokens. The sequence is known as the content code vector (CCV). For each element in the CCV, calculate the ratio of content to code (CCR) in its neighbourhood to identify whether it is mainly surrounded by content or code. An area with high CCR values correlates to a region that contains the essential information. To determine the CCR, it computes a weighted and local average of the values in a neighbourhood with a defined symmetric range for each entry. If every element in this neighbourhood began with a value of 1, the neighbourhood average would likewise be 1; the same holds for neighbourhoods with a starting value of 0. In inhomogeneous areas, the average value will vary between 0 and 1 based on the surrounding factors.

2.1.4 Benchmarking of Extractors

Konstantin Lopukhin has published benchmarking of extractors for commercial services and open-source libraries newspaper3k, readability-lxml, dragnet, boilerpipe, html-text, trafilatura, go-readability, Readability.js, Go-DomDistiller, news-please, Goose3, inscriptis, html2text, jusText, BeautifulSoup.

The collection of data occurred in two phases. To get long-tail URLs, a random sample of one thousand domains from the one million most popular websites, according to Alexa, was selected. The second step was acquiring news stories from major websites. To do this, all URLs, excluding YouTube links, were concatenated, yielding 356 URLs from 189 domains (several domains had more than 10 pages).

The N-Gram text similarity metric was utilized for benchmarking. Precision quantifies how "clean" the output article body is or how effectively unnecessary content is omitted.

Recall quantifies how effectively the system preserves intended article body components. First, the N-grams methodology was chosen over the Bag of Words method because single words might appear in an article's favourable and undesirable portions. A set of words does not punish unnecessary repetitive text; thus, instead of a set, n-gram counts were also taken into account. On a total of 181 web pages, trafilatura, go readability, and go-domdistiller performed best on the N-gram based text similarity measure, according to the reports. This thesis is an expansion of their work by using additional text similarity measures, adding a larger corpus (3k+ webpages), and adding a few new extractors.

2.2 Extractors

The primary responsibility of the extractors is to eliminate all possible boilerplates, such as advertisements, a notice of copyright, header, footer, navigation etc., from the web page and deliver the main content. There are extractors accessible as open source or commercial software; however, we have chosen generic extractors that can function on any website with minimal configuration and can be deployed for benchmarking. Since most algorithms do not consider images or media part as the primary content, we only concentrated on text extraction. A lack of implementation details for the extractors approach was one of our difficulties when selecting extractors. The extractors were gathered after reading and examining research articles, which various authors utilized to compare their implementation to other extractors. We avoided extractors that require domain-specific modification or extractors that are designed to extract specific text/tags from online pages, as benchmarking such extractors on a collection of web pages from various domains would produce misleading results.

Listed below are the content extraction algorithms selected for benchmarking:

- **Boilerpipe** The Java-based boilerpipe library eliminates boilerplate and templates surrounding the primary content by detecting boilerplates using shallow text properties such as word length and sentence length. Kohlschütter et al. describes and implemented the library, which is distributed under the Apache License 2.0. However, we have utilized the Python wrapper for the open-source library licensed under the Apache License 2.0 (Kohlschütter et al. [2010]).
- **BeautifulSoup (bs4)** It is one of the most used Python libraries for scraping data from HTML or XML. The library is licensed under MIT. BeautifulSoup contains a

variety of simple approaches for extracting specific data from a webpage, including article tags, headings, paragraphs, and navigation. Using the library, however, we retrieve the whole HTML body tag text. Although the library lacks configurable options to minimize boilerplate and extract only the main content, it was selected for benchmarking to compare its performance to other extractors on web pages containing minimum boilerplate surrounding the main content (Richardson [2010]).

- **Body Text Extraction** A Python implementation of the algorithm under MIT-licensed was discovered (Vogels et al. [2018]). The implementation is based on Finn et al. BTE (Body Content Extraction) technique for extracting the primary text of a web page while eliminating surrounding irrelevant content. The Body Text Extraction approach is based on the notion that the primary content part of a webpage consists primarily of text and contains relatively little markup (Finn et al. [2001]).

- **Go Domdistiller** The algorithm is written in Go, a programming language licensed under MIT. The library is built on DOM Distiller implementation, which is also loosely based on Kohlschütter et al. [2010]. According to the documentation, the library is superior to DOM Distiller since DOM Distiller only includes render-level information. For example, things that are obscured from viewing are not considered content, nor are images that are too small considered lead images, etc[1]. The library pulls metadata from articles and is particularly suitable for processing news articles (Mobius and Fadlillah [2020]).

- **Lxml Cleaner** A Python module for removing HTML tags that are predefined from web pages. The library needs tuning. Internally, trafilatura python-based extractor uses Lxml Cleaner. The same tuning as the algorithm trafilatura was utilized. The module was chosen for benchmarking to determine whether the additional processes performed by trafilatura significantly improve content extraction.

- **Goose** is a Java, Scala, and Python application. Python implementation was chosen since it was simple to configure and deploy. The library is particularly intended for article-style webpages, from which it may extract not only the primary text but also, most likely, graphics that are part of the primary text. The licensing for this library is Apache license 2.0. Goose3 ranks HTML nodes depending on characteristics such as punctuation density, frequency of stop words, text length etc. Based on the node's

[1]https://github.com/markusmobius/go-domdistiller

score, the nodes are tagged as boilerplate and ultimately eliminated. The remaining nodes are the primary content nodes (GravityLabs [2010]).

- **Html2Text** A library written in Python that converts HTML pages into ASCII text. The downside of html2text is that it does not preserve the text components' spatial positioning. A few instances where the web page is difficult to extract may result in improper text alignment. The library is licensed under version 3.0 of the GNU General Public License (Swartz [2014]).

- **Html_Text** An enhancement to Beautifulsoup's Python-based library. The advantage of the hmlt_text library over the Beautifulsoup library is that extracted information does not include inline styles, javascript, or any other content that is not visible to the user. The library was selected for benchmarking purposes to compare its results with Beautifulsoup on simple web pages with less available boilerplate (TeamHG-Memex [2016]).

- **Inscriptis** After Beautifulsoup, it is also one of the most often utilized libraries. The script functions similarly to Beautifulsoup and offers few benefits, except it preserves the spatial positioning of text components, which HTML2Text does not (weblyzard [2016]).

- **JustText** The code for the library is written in C++, Java, Go, and Python. It is a generic removal script. There are 2 steps that the JustText script utilizes. In the first phase, three features are computed for each segment: token length, link count, and stop word count. The second phase (sensitive to context) modifies the categorization of short and near-good segments based on the classification of their neighbours. Good is assigned to a short segment if its adjacent segments are either Good or Near-good. Good is assigned to a Near-good node if at least one of its neighbours is Good (Gaël Lejeune [2020]).

- **Readability** The algorithm is primarily developed in javascript; however, go and python wrappers are available. The licensing for the library is Apache 2.0. The go-implemented script has been used for benchmarking. The conversion function accepts the document as input and outputs the article title, HTML string with processed article content, article length in characters, article description (or automatically extracted snippet), and author metadata (byline) (videoinu [2020]).

- **Resiliparse** is an extractor on rule-based filters and content heuristics. It applies a set of rules for removing page elements such as navigation blocks, sidebars, footers, some ads, and (as far as they are possible to detect without rendering the page) invisible elements (Bevendorff et al. [2018]).

- **Trafilatura** The GNU General Public License version 3.0-compliant Python library for main text discovery. Trafilatura employs readability, lxml and justtext as backup processes. The library applies a cascade of rule-based filters. Content delimitation is performed by XPath expressions targeting common HTML elements and attributes as well as peculiarities of major content management systems, first in a negative perspective with the exclusion of undesirable portions of the HTML code (e.g. div class="nav">) and then by focusing on the desired content. The same processes are carried out on comments if they are included in the extraction. The selected nodes of the HTML tree are then processed, i.e., relevancy is assessed (particularly by element type, text length, and link density), and their HTML structure is reduced (Barbaresi [2021]).

- **Web2Text** A boilerplate elimination approach based on a convolutional neural network. The algorithm consists of three phases. The initial phase is processing HTML into a DOM tree and extracting features from tree blocks. On top of these traits, the second phase includes training and predicting unary and pairwise probabilities. In the third stage, the blocks selected as major content in the second step are merged (Vogels et al. [2018]). The implementation is explained in further detail in section 2.1.2.

The extractors listed above can be classified into two categories: simple extractors and main content extractors. Simple extractors remove minimal or no boilerplate and retrieve the entire body tag text, whereas main content extractors remove all possible boilerplate and recover the main content. Table 2.1 provides a concise overview of extractors. Until otherwise specified, the word extractor will be used for both the simple and the main content extractors in subsequent chapters.

Table 2.1: Selected Content Extraction Algorithms for Benchmarking

Extractor	**Category**	**Author**	**Programming Language**
boilerpipe	Main	Kohlschütter et al.	java
body text extraction	Main	Finn et al.	python
go domdistiller	Main	Mobius and Fadlillah	go
lxml cleaner	Main	Behnel	c++, python
goose	Main	GravityLabs	go, java, python
justtext	Main	Pomikálek	c++, go, java, python
readability	Main	Mozilla	javascript, go, python
resiliparse	Main	Bevendorff et al.	cython
web2text	Main	Vogels et al.	scala, python
html2text	Simple	Swartz	python
html_text	Simple	TeamHG-Memex	python
inscriptis	Simple	weblyzard	python
beautifulsoup (bs4)	Simple	Richardson	python

Chapter 3

Creating and Analyzing Datasets for Benchmarking Extractors

This chapter describes the data collected from various sources for benchmarking and provides detailed data insights. Section 3.1 describes collecting data from various sources. Section 3.2 discusses parsing gold standards into a standard format to facilitate the processing in subsequent chapters. Section 3.3 discusses categorizing web pages into different categories to determine the diversity of web pages used for benchmarking. Section 3.4 describes analyzing human errors while defining the gold standards. Section 3.5 and section 3.6 discuss grouping web pages by complexity and using the K-means cluster algorithm respectively.

3.1 Collecting Annotated Web Pages for Benchmarking Extractors

Web pages and their gold standard are the fundamental unit for comparing the extractors mentioned in section 2.2 using the score metrics stated in chapter 4. The gold standard of a web page is the annotation (main content) that has been checked and corrected to evaluate content extraction algorithms. The significant challenge was collecting web pages and its gold standard from numerous domains for benchmarking the content extractors. The first step would be to collect web pages from multiple domains of varying content sizes. The second step would be to define its gold standard. To determine the gold standard, one strategy would be to manually analyze each web page and annotate its main content.

Before this phase, standards must be set on which HTML sections should be identified as the main text. The undertaking is difficult, time-consuming, and costly. The second approach would be to select the extractors with the highest performance based on research papers and available benchmarks, utilize the extractor's output as the gold standard, and then compare the other extractor's findings to the gold standard. Nevertheless, this would not be an accurate comparison of the extractors. When comparing the results in different contexts, the results would be skewed.

We followed the first strategy outlined above, but we searched online for labelled web pages that might be utilized for benchmarking. We investigated the GitHub repositories of the deployed extractors used in this study, analyzed research publications for the dataset used by the authors for benchmarking, and contacted 4 authors to access the manually labelled dataset. A total of 3114 web pages and their gold standard were collected from various domains, such as news, sports, electronics, fitness etc., from the five sources listed below.

- **Readability dataset** (Mozilla [2015]) The Go language implementation of readability.js had 115 labelled websites in the test module repository on Github. The markup was maintained in the gold standard. The gold standard format is depicted in Figure 3.1a.

- **Article Extraction Benchmark dataset** (Konstantin Lopukhin) It is a GitHub repository that assessed commercial and open source content extraction algorithms on 181 labelled webpages consisting of HTML files compressed with gzip and the gold standard encoded in JSON format with the fields articleBody containing the gold standard and URL containing the source of the webpage. Figure 3.1c depicts the format specified for the gold standard. The key is the URL's SHA-256 hash. Each HTML source filename corresponded to the key used in the gold standard to link the web page to the gold standard.

- **Content Extraction via Text Density dataset** (Sun et al. [2011]) Section 2.1 examines the research article. The authors of the research article have collected web pages from the domains Arcs Technica, BBC, Yahoo, and the New York Times, with 100 randomly selected web pages in each domain, as well as the Chaos dataset containing 200 randomly selected web pages from Google News and other blog platforms.

700 web pages were collected and stored in txt format using UTF-8 encoding for the gold standard. The gold standard format is depicted in 3.1d.

- **CleanEval dataset** (Baroni et al. [2008]) The CleanEval contest was held in 2007 to prepare online data for use as a corpus for linguistic and language technology research and development. The major assignment involved removing boilerplate from the website, such as the navigation menu, advertisements, footer, and copyright notice. The websites were gathered from diverse domains by submitting Google queries containing common search terms. The gold standard was recorded in txt format, and a total of 737 websites were tagged. Figure 3.1b depicts CleanEval's gold standard format. The tag before each text block identifies the leaf node tag containing the content. These tags were deleted during the data preparation process.

- **Dragnet dataset** (dragnet org [2012]) Dragnet is an open-source, C++ based content extraction technique. The gold standard for 1381 archived websites from the Dragnet dataset was maintained in text format in the project's GitHub repository.

No documentation was available describing the guidelines used to annotate the main content of the web pages obtained from different sources. It was learned that most webpages were in English, while few were written in other languages, and that the gold standard was stored using multiple encoding techniques, such as utf-8, utf-8 with bom, and ansi. While parsing the gold standard, the various encoding techniques caused considerable challenges. A summary of data sources is presented in Table 3.1.

Table 3.1: Labelled webpages data sources

Data Source	Data Size	Gold Standard Format
Readability	115	html
Article Extraction Benchmark	181	json
Content Extraction via Text Density Ratio	700	txt
CleanEval	737	txt
Dragnet	1381	txt

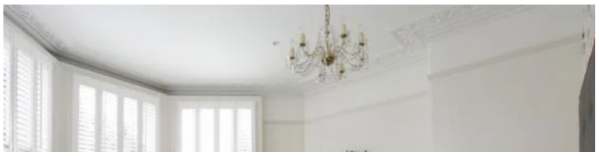
Most people go to hotels for the pleasure of sleeping in a giant bed with clean white sheets and waking up to fresh towels in the morning.

But those towels and sheets might not be as clean as they look, according to the hotel bosses that responded to an online thread about the things hotel owners don't want you to know.

Zeev Sharon and Michael Forrest Jones both run hotel start-ups in the US. Forrest Jones runs the start-up Beechmont Hotels Corporation, a hotel operating company that consults with hotel owners on how they can improve their business. Sharon is the CEO of Hotelied, a start-up that allows people to sign up for discounts at luxury hotels.

But even luxury hotels aren't always cleaned as often as they should be.

Here are some of the secrets that the receptionist will never tell you when you check in, according to answers posted on Quora.



(a) Readability (.html). The markup was maintained along with the main text

```
URL: http://artsedge.kennedy-center.org/irish/share/storytelling/workshop/reading/artsedge.html

   <h>Storytelling Online: Mythology Across Time and Borders

   <h>Online Writing Workshop

   <h>TEACHER READING

   <p>Clement, Lynne, "Revising & Editing," 1991, rev. 2000.
   A brief sheet that discusses the differences between revising and
   editing and gives teachers tips on dealing with groups of students
   when conducting these activities. It stresses the importance of
   revising student work for content and meaning, before giving any
   attention to the mechanics. Quick to read, but worthwhile.
   <p>Hansen, Jane, When Writers Read, Chapter 12; Heineman, 1985.
   This article gives teachers some information on the connections
   between reading and writing and how to capitalize on them for success
   in writing instruction. It focuses on skills instruction, reasons for
   it, how to implement it, and what to keep in mind as classes are
   conducted.

   <p>Jenkinson, Edward B., "Learning to Write/Writing to Learn," Phi Delta
   Kappan, 1988.
   This article will give teachers some background on the difference
   between the two activities mentioned in the title and the importance
   of both for their students. "For thousands of students, writing is
Ln 1, Col 1                                                    160%   Unix (LF)   UTF-8
```

(b) CleanEval (.txt). Highlighted text contains url and leaf node markup information


```
{"042bb7b5fedab6eac7db576522b89b93904c237d344bcbe14a6a5ab7f7335856": {
        "articleBody": "Gaming used to be so simple. We'd buy a game, sit down in front of a console or PC, grind our way to the end, then repeat.\n\nNow we spend money over and over on virtual perks. We play on phones and tablets while socializing with far-flung friends. And there is no "end," because developers keep updating their biggest hits with new maps, missions and characters.\n\nAll...",
"url": "https://www.wsj.com/articles/google-stadia-microsoft-xcloud-apple-arcade-so-many-ways-to-playand-pay-11574168580"
    }
```


(c) article-extraction-benchmark (.json). Highlighted text is the SHA-256 value of the url. The same key was used as the filename of the source web page.

```
FCC: Yup, we're going to stop "paid prioritization" on the 'Net
By Matthew Lasar | Last updated 8 days ago

The Federal Communications Commission is releasing the details of its new net neutralit

"Pay for Priority Unlikely to Satisfy 'No Unreasonable Discrimination' Rule," advises o

"A commercial arrangement between a broadband provider and a third party to directly or

Insofar as engaged

As we've reported, the FCC's new rules forbid Internet providers from blocking lawful c

Here's the text of the Commission's "no unreasonable discrimination" rule:

A person engaged in the provision of fixed broadband Internet access service, insofar a

What are "reasonable network management" practices? Here you go:
```

Ln 1, Col 1 | 200% | Windows (CRLF) | UTF-8 with BOM

(d) content extraction via text density (.txt). The encoding scheme is UTF-8 with BOM (highlighted in yellow)

```
NBC's Costas says he plans to honor Israelis
By DAVID BAUDER
AP Television Writer
Associated Press Sports
updated 4:36 p.m. ET July 23, 2012

NEW YORK (AP) - NBC Sports anchor Bob Costas says he plans his own on-air commemoration this week of

A bid to honor the athletes and coaches killed by Palestinian gunmen during the 1972 games with a mo

Costas, who called the International Olympic Committee's decision baffling, told the Hollywood Repor

"Many people find that denial more than puzzling but insensitive," Costas said. "Here's a minute of

Through a spokesman, Costas denied a request by The Associated Press to speak further about his plan

"Our production plans for Opening Ceremony are still being finalized and Bob is part of that plannin

IOC President Jacques Rogge offered a moment of silence Monday to the 11 Israelis during an Olympic

Two days earlier, Rogge said that the opening ceremony "is an atmosphere that is not fit to remember

Abraham Foxman, national direction of the Anti-Defamation League, said support from Costas would be
```

Ln 1, Col 1 | 170% | Unix (LF) | UTF-8

(e) dragnet (.txt).

Figure 3.1: Examples of gold standard files from data sources

"7bd033687f11eb016e8edf5a141c35f8047a8a85f72fa00691d6d74981fa8113": {
"articleBody": "'Ronaldo signing would be great for PSG' - PastoreSaint-Germain playmaker Javier Pastore has admitted that he would love to see Real Madrid attacker Cristiano Ronaldo sign for the club.\n\nThe Portuguese has been linked with a move to Carlo Ancelotti's side in recent weeks and the Argentina international believes the former Manchester United man would have an excellent effect on the squad.\n\nR̈onaldo is one of the world's best players, he scores almost every game,ḧe told L'Equipe.\n\nḦe knows how to make the difference on his own. If he signs for PSG, it would be great for the club, and for the players who are already here.\n\nS̈o I would say yes to him coming!\̈n\nRonaldo scored 46 goals in La Liga last season as Jose Mourinho's side was crowned champions.\n\nFollow Goal.com on Twitter to get the latest soccer news directly. Check out Goal.com's Facebook page; be part of the best soccer fan community in the world!\n\n\n\n\n,
"filename": "R493.html",
"source": "dragnet"
},

Figure 3.2: The JSON format to standardize the gold-standard format across various data sources. The key (highlighted in blue) is the SHA-256 of the webpage's source content. articleBody holds the primary text of its source webpage, source refers to the web page's source, and filename is the name of the web page's archived file.

3.2 Data Preparation

Several corpora sources adopted individual gold standard formats, such as JSON, TXT, and HTML, as indicated in section 3.1. The datasets employed by earlier authors varied widely in structure, size, and content. Due to many encoding standards, parsing the gold standard presented a significant challenge. The gold standard format was standardised for the thesis to examine the gold standard and evaluate the extractors.

The gold standard format was manually encoded to adjust the encoding systems with the utf-8 encoding scheme, and non-translatable characters were replaced with a question mark (?). Figure 3.2 depicts the JSON format standard (thesis standard format) to store the gold standards of the archived web pages. The Article Extraction Benchmark dataset (section 3.1) format was implemented, which utilized the URL SHA-256 hash as the key. However, since the URLs for each web page were unknown, the HTML content SHA-256 value was utilized as the key for the gold standard in the thesis standard format. This method lets us quickly link the web page with its gold standard. A source parameter that gives a description of the dataset and the filename of the web page provides additional information to look over. Text structure was not cleaned up; the same number of spaces,

new lines, and tabs as in the original were maintained.

When parsing the gold standard for various source datasets to JSON standard format, the following uncertainties occurred:

- In the CleanEval gold standard, multiple concatenated underscores distinguished one segment of the main text from another, using leaf node markup in front of the segment of the main text from which the text was extracted. Before it was saved in the standard JSON format, the use of underscores was cleaned up, and any HTML tags at the beginning were deleted. Figure 3.1b depicts the CleanEval gold standard format.
- The gold standard of the Readability dataset maintained markup with the main content. Only the text from the gold standard was extracted, excluding markups, and the output was saved in the standard JSON format. Figure 3.1a illustrates the gold standard format for Readability.

After establishing the gold standard for all the data sources in the defined JSON format, each extractor's output for all the web pages in the dataset was saved in the same standard JSON format. Each extractor accepts a web page in text format as input and outputs the primary content in text format (string). This made it easier and faster to compare the gold standard and extractor output using text similarity measures.

3.3 Classifying Web Pages used for Benchmarking to Determine Diversity

Web classification refers to categorizing a webpage's main content into pre-defined categories. The data sources contained minimal or no information regarding the genre of the web pages. Classifying web pages is required to establish if the dataset is sufficiently diverse, as generic content extractors that do not require domain-specific modification are being assessed.

There are two methods for classifying webpages into predefined categories. However, because the first strategy was ineffective, the second approach was implemented. The first approach to classifying is manually analyzing and categorizing each webpage into predefined categories. An individual could not categorize web pages in a time-efficient manner,

hence the approach was discarded. The second approach is Google's cloud-based natural language processing, the content classification API, that examines documents and delivers a list of categories together with their confidence score (Figure 3.3a). The API uses NLP techniques to categorize web pages. Google Cloud has developed more than 600 categories, including subcategories, for text classification. The API will deliver the most relevant category. For example, Travel/Tourist Destination is returned if the content is classified as both Tourist and Travel/ and Travel/Tourist Destination. The generic category for the study was chosen and disregarded subcategory classification. For example, if the API classifies the text as Travel/Tourist Destination, the web page was categorized as Travel/. The gold standard was utilized to classify webpages. If a gold standard was divided into numerous categories, the category with the highest confidence score was selected. Due to an API limitation, the text in the gold standard was reduced to the first 1000 characters.

The web pages in the dataset were divided into 28 categories, with the news category containing the most web pages, followed by Arts Entertainment with 298 web pages (Table 3.2). The metrics are not 100 percent correct because confidence intervals as low as 60% were considered when classifying webpages. It would have been impractical for an individual to test the classification accuracy manually, but the statistics provide a rough sense of the diversity of the web pages in the dataset, which would be used for benchmarking in the coming chapters.

Android users biggest data hogs of them all By Jacqui Cheng Last updated 22 days ago Move over iPhoneophiles: Android phone users are the biggest data hogs of all according to a new report. Network management firm Arieso conducted a series of studies in order to find out how much data current smartphone users are downloading and how often while the iPhone 4 is definitely near the top of the rankings, Android is king pig. Arieso used the old iPhone 3G as a reference point to create its comparison benchmarks. In general, current smartphone users have shown an increase of 130 percent in the number of times they make some sort of data request, along with an increase in uplink and downlink traffic of 130 and 40 percent respectively. When looking specifically at the iPhone 4, the firm saw 44 percent more data calls and 41 percent more data being downloaded. The device was the leading data gobbler compared to the iPhone 3G until Arieso looked at Android devices.

(a) Gold Standard text

{
"Internet Telecom/Mobile &Wireless/Mobile Phones": 0.670000016689301,
"Computers & Electronics/Consumer Electronics": 0.620000004768372
}

(b) Google Api classify text result for 3.3a

Figure 3.3: Google Cloud Classify Text Api to categorize webpage's main content

Table 3.2: Web page classification collected from various sources.

Category	Count	Percent
News	745	23.924
Arts & Entertainment	298	9.569
Computers & Electronics	249	7.996
Business & Industrial	195	6.262
Sports	171	5.491
Science	164	5.266
Law & Government	159	5.105
People & Society	157	5.041
Sensitive Subjects	109	3.500
Health	105	3.371
Internet & Telecom	95	3.050
Jobs & Education	85	2.729
Food & Drink	73	2.344
Hobbies & Leisure	66	2.119
Games	65	2.087
Online Communities	54	1.734
Books & Literature	47	1.509
Finance	45	1.445
Autos & Vehicles	43	1.380
Reference	38	1.220
Travel	35	1.123
Shopping	33	1.059
Beauty & Fitness	30	0.963
Home & Garden	28	0.899
Real Estate	19	0.610
Pets & Animals	4	0.128
Adult	2	0.064
Total	3114	

3.4 Catalogue and Quantify Gold Standard Manual Annotation Error

Human annotators produced the gold standard by manually copying and pasting web page sections that were recognised as the main text; hence they are susceptible to human errors. There was no documentation available for the error analysis and gold standard quality. To ensure that benchmarking is not influenced, it is vital to ensure that the quality of the

gold standard is maintained.
Common human errors that had occured while producing gold standards include:

- **Issue 1: Text duplication** (Figure 3.4) One of the frequent errors in the gold standard is duplicated text. For instance, a leaf node section of the main text is copied more than once in the gold standard.

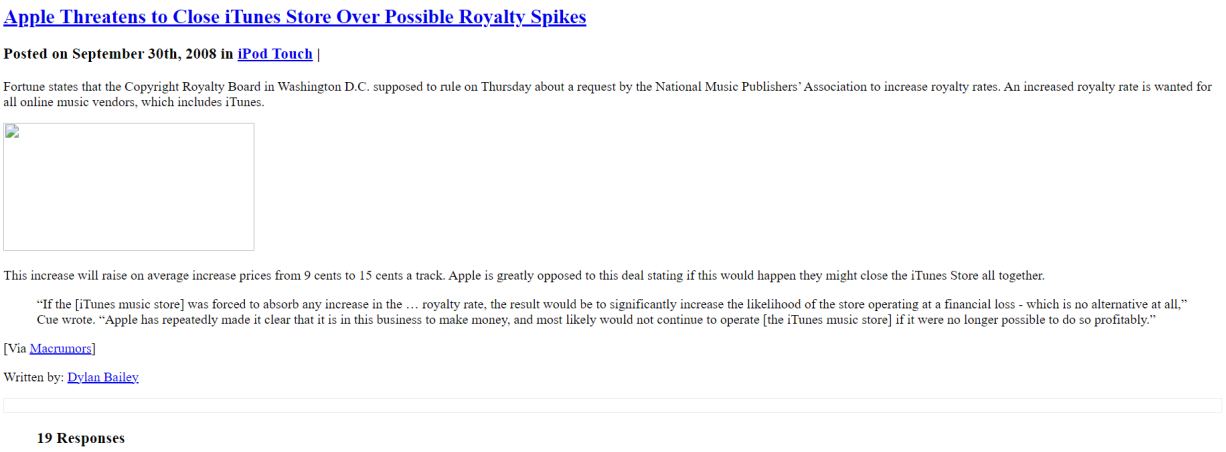
- **Issue 2: Concatenation of words without spaces between two html elements** This common error is when words are joined together without using spaces. For instance, the HTML element Text1 can be represented as Text1 or Text 1. At times, annotators concatenated the element text without a space; at other times, they inserted a space. The issue cannot be considered a mistake, but the absence of conventional guidelines for such situations has led to uncertainty, which might affect benchmarking scores.

- **Issue 3: Incorrect linking of gold standard to its webpage** There were two web pages in the dragnet corpora affected by this problem. The web pages were wrongly archived, as the archived web pages indicated error:500, internal server error, yet its gold standard referred to other text. One of the potential causes of the issue might be that the annotators utilized the active URL for annotation rather than the archived web pages.

- **Issue 4: Web page lacks content, yet the gold standard contains it** There were rare instances where a portion of content from a gold standard was absent from the source webpage.

- **Issue 5: Adding alt attribute text of images to the gold standard** (Figure 3.5) The Alt attribute specifies a replacement text if the image cannot be displayed. On some web pages, media tags such as photos and alt attribute text were added to the gold standard. The majority of content extractors do not regard the alt property portion as the primary text; hence, this issue will result in biased benchmarking. Some of the gold standards with this issue were rectified manually, however it was not possible to address the issue in every instance.

Two different techniques were utilized to detect human mistakes in the gold standard: N-Gram technique and bag of words technique. However, N-Gram technique was discarded in favour of the bag of words technique.



Apple Threatens to Close iTunes Store Over Possible Royalty Spikes

Posted on September 30th, 2008 in iPod Touch |

Fortune states that the Copyright Royalty Board in Washington D.C. supposed to rule on Thursday about a request by the National Music Publishers' Association to increase royalty rates. An increased royalty rate is wanted for all online music vendors, which includes iTunes.

This increase will raise on average increase prices from 9 cents to 15 cents a track. Apple is greatly opposed to this deal stating if this would happen they might close the iTunes Store all together.

"If the [iTunes music store] was forced to absorb any increase in the … royalty rate, the result would be to significantly increase the likelihood of the store operating at a financial loss - which is no alternative at all," Cue wrote. "Apple has repeatedly made it clear that it is in this business to make money, and most likely would not continue to operate [the iTunes music store] if it were no longer possible to do so profitably."

[Via Macrumors]

Written by: Dylan Bailey

19 Responses

(a) A Webpage snapshot

Apple Threatens to Close iTunes Store Over Possible Royalty Spikes
Posted on September 30th, 2008
Fortune states that the Copyright Royalty Board in Washington D.C. supposed to rule on ....
This increase will raise on average increase prices from 9 cents to 15 cents a track. .....
"If the [iTunes music store] was forced to absorb any increase in the . . . ...

Apple Threatens to Close iTunes Store Over Possible Royalty Spikes
Posted on September 30th, 2008 in iPod Touch |
Fortune states that the Copyright Royalty Board in Washington D.C. supposed to rule on...
This increase will raise on average increase prices from 9 cents to 15 cents a track.

(b) Text duplication in gold standard

Figure 3.4: Example of Text duplication in gold standard: Issue 1

- **N-Gram technique** In this procedure, n-gram tokens for the gold standard were generated, and their occurrences were recorded; if any n-gram token count is greater than 1, the flag count is incremented by 1. If the number of flags exceeds the threshold k, the gold standard and its source web page were thoroughly examined. The value of n in n-gram was 10, whereas the threshold k was 15. Approximately 800 web pages were collected. Examining 800 web pages was time-consuming and impractical for the study's purposes. As a result, the strategy was discarded in favour of the bag of words approach, as discussed below.

- **Bag of words technique** A bag of words is a textual representation of the occur-

```
<span class="commentdate">
        <a href="#c68267497" title="Link to this comment">#1.7</a>
        - Tue Jul 24, 2012 1:38 PM EDT
</span>
...
...
<div class="c_author">
        <div class="normal">
                <a href="http://govhater.newsvine.com/">
                    <img class="comment_author_avatar"
                     alt="Comment author avatar" >
                </a>
                    GovHater
                </a>
        </div>
</div>
...
...
<p>Free market economy is what did it. When the companies
could freely compete, prices came down.
</p>
<p>Bwaaahahahahahaha! That's the funniest thing
I've heard in a long time...</p>
```

(a) Webpage IMG element containing the alt attribute

Tue Jul 24, 2012 1:38 PM EDT\nComment author avatarGovHater\n \nwillowbrook\n\nF̈ree market economy is what did it. When the companies could freely compete, prices came down.̈\n\nBwaaahahahahahaha! That's the funniest thing I've heard in a long time...

(b) The gold standard section of a webpage containing alt attribute text of IMG HTML tag

Figure 3.5: Example of adding alt attribute text of IMG tag to the gold standard: Issue 5

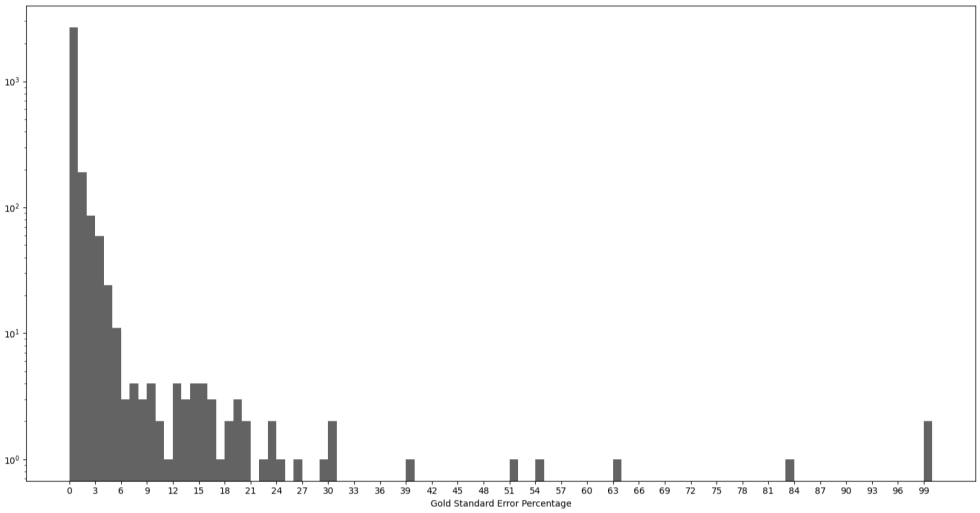


Figure 3.6: Gold standard error distribution according to the equation 3.2

rence of words in a document. The occurrence of each token in the gold standard to its occurrence in the source web page was compared. A token's frequency in the gold standard should be less than or equal to its frequency on the source web page. The inaccuracy for each web page was determined (Equation 3.1 3.2). Figure 3.6 depicts the distribution of errors.

$$B_{\mathrm{K}}(x) = \begin{cases} 1, & \text{if } F_K^G(\mathrm{x}) > \mathrm{F}_K^S(\mathrm{x}) \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

$$E(K) = \frac{\sum B_{\mathrm{K}}(x)}{M} \tag{3.2}$$

where:

K : Webpage
G : Gold standard
S : Source webpage
x : Distinct alphanumeric token in K
F_K^Y(x) : Frequency of token x in K in the representation Y
$E(K)$: Error of K
M : Total alphanumeric tokens in K

Analyses were conducted on web pages with error rates over 1%. There were two web pages with an error rate of 100%. These web pages were linked to issue 3 (incorrect linking of gold standard to its webpage) stated previously. Issue 1 (text duplication) , issue 4 (web page lacks content, yet gold standard contains it) , and issue 5 (adding alt attribute text of images to the gold standard) were linked to the web pages with an error ratio between 1% and 10%. However, numerous web pages with an error ratio of less than 1% were linked to the issue 2 (concatenation of words without spaces between two html elements). Some of the problems relating to the issue 4 were addressed manually, but it was difficult and impractical to fix all errors in a complete dataset. During the benchmarking of extractors, all web pages with an error rate of greater than 2% were discarded in order to limit evaluation errors to a minimal. 3100 web pages were narrowed down to 2804 web pages during this process.

3.5 Grouping Web Pages by the Complexity

One of the important findings by benchmarking extractors using the source dataset (section 6.1.1) is that no single extractor outperforms the others and choice of a content extractor depends on the properties of a webpage. Regarding the RougeLSum score metric, the web2text extractor performs best on the CleanEval dataset, whereas 14 extractors perform better than the web2text extractor on the Article Extraction Benchmark dataset. As the dataset varies, there is a drastic variation in rankings. Selecting a single extractor for a random webpage content extraction is tricky.

After thoroughly evaluating webpages from each source, the choice of webpages in the source dataset group varied. In the CleanEval dataset, in the gold standard, most webpages contained all of their text; however, in the Article Extraction Benchmark dataset, just a few sections of HTML were recognized as the main text (Table 3.3). This finding

concludes that the choice of extractor depends on the webpage. Therefore, webpages from various sources were gathered and categorized according to their complexity.

As a first stage, the website complexity is computed, which is the ratio of the number of tokens in the gold standard to the number of tokens in the source webpage for each webpage, where a token is an alphanumeric word (Equation 3.3). The ratio indicates the proportion of a webpage's text that comprises the main text. The ratio is inversely proportional to the complexity of a website; the smaller the ratio, the more complicated the web page. We define complexity ratio C as:

$$C = \frac{|T_g|}{|T_s|} \times 100 \tag{3.3}$$

where:

T_g : multiset of alphanumeric tokens in gold standard
T_s : multiset of alphanumeric tokens in source webpage

The webpages from all the sources were divided into three categories: complex, mid-complex, and easy. The quantile technique was chosen to divide the web pages into categories based on complexity. If a webpage ratio is below the 25^{th} percentile, it was categorized as complex; if it was above the 75^{th} percentile, it was defined as easy; and if it was between the 25^{th} and 75^{th} percentiles, it was classified as mid-complex (Equation 3.4, Table 3.3).

$$f(K) = \begin{cases} easy, & \text{if } C(K) \geq P_{75} \\ complex, & \text{if } C(K) \leq P_{25} \\ mid - complex, & \text{if } P_{25} < C(K) < P_{75} \end{cases} \tag{3.4}$$

where:

K : Webpage
$C(K)$: Complexity ratio of a web page as per Equation 3.3
$f(K)$: Complexity of a webpage (Equation 3.3)
P_{75} : Webpages complexity ratio 75^{th} percentile.
P_{25} : Webpages complexity ratio 25^{th} percentile.

Table 3.3: Equation 3.3 complexity of webpages percentile value.

DataSet Source	25^{th} percentile	50^{th} percentile	75^{th} percentile
Article Extraction Benchmark	33.534	48.838	69.398
Content Extraction via Text Density	53.742	64.190	80.212
CleanEval	79.019	91.708	97.213
Readability	43.116	62.205	83.637
Dragnet	28.704	46.528	67.046
All Sources Combined (Mean)	40.816	62.640	84.078

Table 3.4: The proportion of webpages from the source dataset assigned to each complexity group.

Dataset Group by Complexity	Source Dataset Group				
	Dragnet	Readability	Article Extraction Benchmark	Content Extraction via Text Density	CleanEval
Easy	7.98	46.08	7.18	20.31	67.97
Mid-Complex	50.82	33.91	55.80	71.92	26.99
Complex	41.18	20	37.01	7.77	5.04

For the easy complexity group, it was anticipated that the performance of simple extractors would be comparable to or superior to that of main content extractors. Table 6.4 indicates the expected outcome indicated in the previous statement. The evaluation results based on the complexity-based dataset group provide more detailed information on which extractor to choose; for example, if a webpage belongs to the easy group, simple extractors will be suitable, whereas, for complex webpages, main content extractors such as web2text, readability, and trafilatura are the best options.

3.6 Grouping Web Pages using the K-means Clustering Algorithm

Two terms must be discussed before discussing dataset grouping by the K-means cluster algorithm.

- **TSNE dimensionality reduction** Dimensionality reduction is a technique to embed higher-dimension data to lower dimensions. Tsne (T-distributed Stochastic Neighboring technique) is a non-linear dimensionality reduction technique in which similar data points are assigned with high probability, and non-similar data points are assigned with low probability. It minimizes the Kullback-Leibler divergence between the two distributions regarding points position[1].

- **K-mean clustering** The process of grouping together related objects is known as clustering. K-means clustering is a non-supervised machine learning technique that separates n observations into k-clusters, making it simple to discover groupings in unlabeled datasets. The technique begins with a specified k centroid (k clusters), and the centroid nearest to each observation is assigned. Each centroid is changed to decrease the distance between it and its representative set of observations (Madhulatha [2011]).

In section 6.1.2, it seemed more logical to evaluate extractors based on the complexity of the web page, as this offered information on which extractor to select for a web page. It is improbable, however, that a user would already have the main text before extraction, as this would render the creation of extractors pointless. There should be an alternative way to separate the collected, labelled web pages from various sources into distinct categories with identical properties without using the gold standard. This section discusses the k-means clustering strategy for separating web pages. Before k-means cluster analysis, the following steps were taken:

- Each webpage's features were retrieved. First, research was conducted on the most frequently used HTML tags on a webpage[2], and for each of the most frequently used tags, the ratio of the number of occurrences of a tag to the total number of HTML tags on the webpage was calculated (Equation: 3.5). A total of 14 features were gathered (Table 3.5).

$$\forall x, R_x = \frac{Y}{Z} \tag{3.5}$$

  where:

[1]https://towardsdatascience.com/what-why-and-how-of-t-sne-1f78d13e224d
[2]https://www.geeksforgeeks.org/most-commonly-used-tags-in-html/

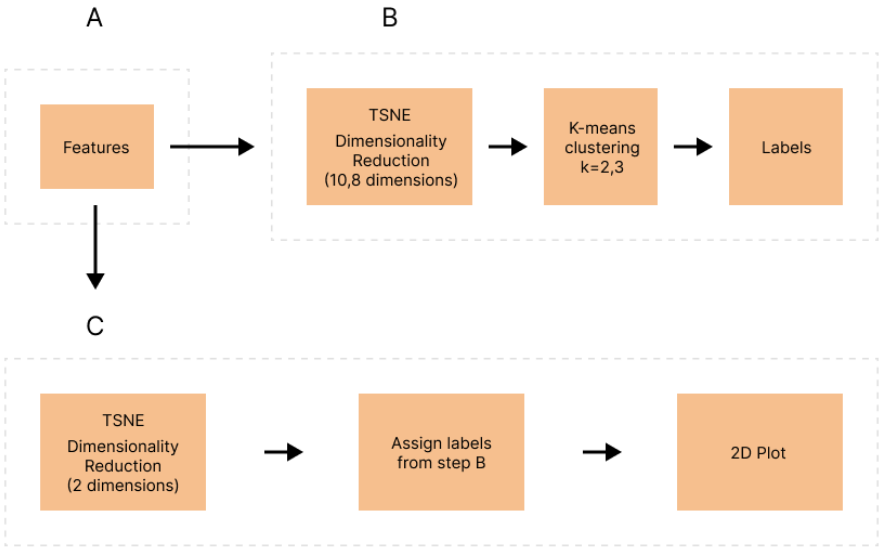


Figure 3.7: Steps performed on the web pages from all sources to group web pages by K-Means clustering.

x : most frequently used tags in a webpage (Table 3.5)
R_x : Ratio of x
Y : count of x in a web page
Z : count of all tags in a webpage

- The 14-featured dataset generated in the preceding step was subjected to Tsne dimensionality reduction. A 14-dimensional data collection was reduced to eight, ten, and two dimensions. K-means clustering algorithm was used to cluster web pages on 8- and 10-dimensional data (Step B ,Figure 3.7). K-means clustering was performed using two and three clusters.

- The labels generated by K-means on eight- and ten-dimensional datasets were allocated to a two-dimensional dataset and analyzed for cluster formation (Step C , Figure 3.7). The cluster formation on variable parameters of K for K-means and the number of dimensions for TSNE is evident from Figure 3.8.

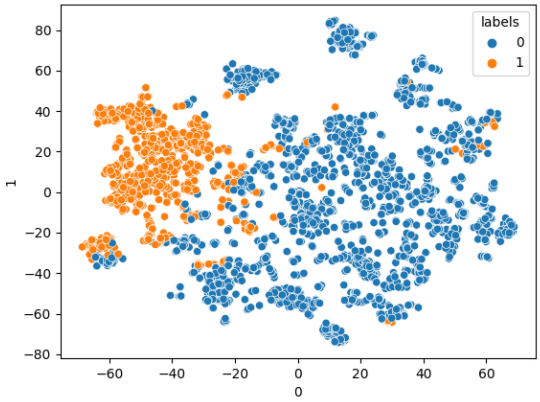
Table 3.6: The proportion of web pages in each complexity group assigned to each K-means cluster group (in percent).

DataSet Group by Complexity	**K-Means Cluster Group (TSNE reduction 8 dimensions, K-Means=3)**		
	label 0	**label 1**	**label 2**
Easy	11.98	20.47	67.55
Mid-Complex	62.48	19.94	17.58
Complex	70.29	18.54	11.17

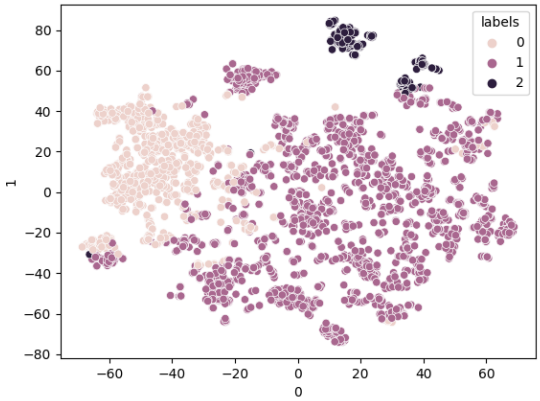
Figure 3.7 depicts a summary of the preceding phases. Two-dimensional data visualization reveals a clear clustering structure (Figure 3.8). There are a few outliers; however, as dimensions increase, they will probably move closer to their cluster position. The clusters, formed by adopting 8 dimensions, and 3 clusters were chosen for assessing extractors on the dataset grouping by K-Means clustering since the grouping in visualization (Figure 3.8) seemed more feasible. K = 3 was used for K-means to match the number of clusters with the number of groups created while classifying the dataset by a web page complexity (Section 3.5). According to Table 3.4, the two categories discussed in Sections 3.5 and 3.6 appear to be unrelated. Each K-means cluster has a reasonable number of pages from each category of the complexity dataset.

Table 3.5: Most frequently used HTML elements

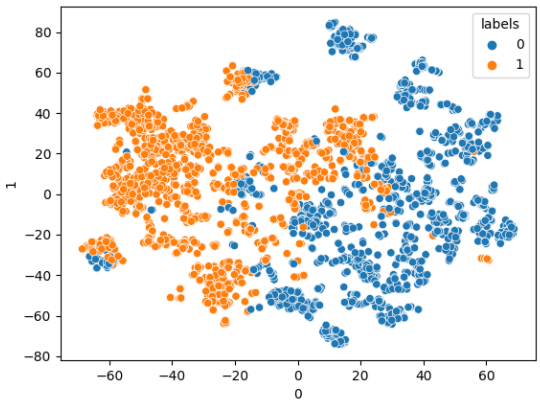
tag	**Description**
h1	Most commonly used to mark up a web page title
h2	Sub-level heading
h3	Sub-level heading
h4	Sub-level heading
h5	Sub-level heading
h6	Sub-level heading
p	Represents a paragraph
ul	Unordered list
a	Defines a hyperlink
div	Division or a Section
br	create a line or break
strong	Define text with bold characters
em	Define emphasized text



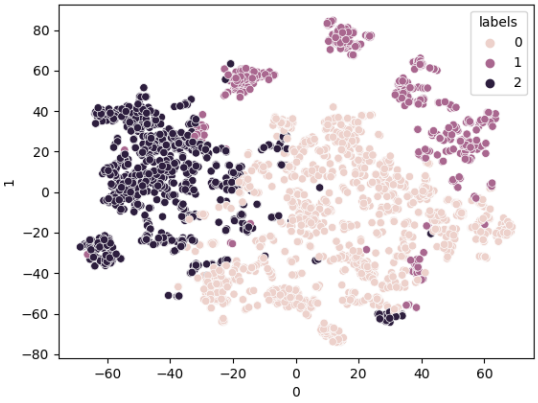
(a) No dimensionality reduction, K-means clusters=2



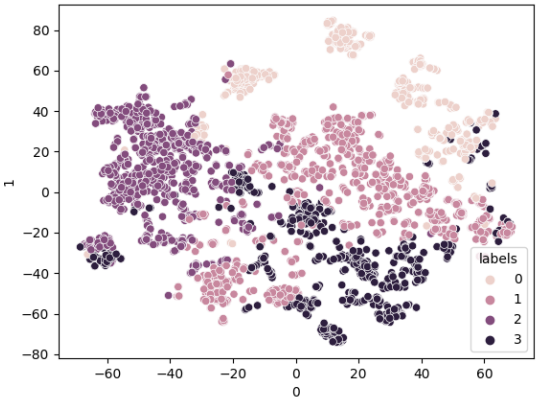
(b) No dimensionality reduction, K-means clusters=3



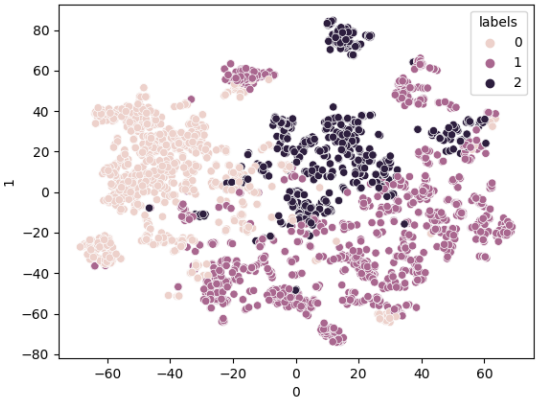
(c) TSNE 8 dimensions, K-means clusters=2



(d) TSNE 8 dimensions, K-means clusters=3



(e) TSNE 8 dimensions, K-means clusters=4



(f) TSNE 10 dimensions, K-means clusters=3

Figure 3.8: Cluster visualization using variable dimensions for TSNE dimensionality reduction and K-means clusters.

Chapter 4

Surveying Text Similarity Metrics for Benchmarking Extractors

Text similarity indicates the degree of lexical or semantic similarity between two words, texts, or sentences. Text similarity measures play an increasingly important role in text related research and applications in tasks such as information retrieval and text classification (Gomaa et al. [2013]). Text similarity metrics can also be used to evaluate the extractor's performance to compare the extractor's outcome to the gold standard.

Semantic similarity measures the likelihood that two text blocks have the same meaning. Typically, the measurement ranges from 0 to 1. 0 implies that two text segments are not semantically related, whereas 1 shows semantic similarity. Semantic similarity defines "is a" connection. For example, vehicle and automobile are semantic identical. Euclidean distance and cosine similarity are an example of semantic similarity measurements (Corley and Mihalcea [2005]).

The lexical similarity assessment determines the degree of similarity between two blocks of text based on the intersection of their respective word sets. Lexical similarity analyzes the similarity between two documents at the word level. A score of 1 indicates complete word overlap between two text blocks. Lexical similarity measurements consist of the Jaccard Index, Levenshtein edit distance, RougeLsum, Bag of Words, and N-grams. Taking note of the two phrases 1. A man drives an automobile 2. An automobile drives a man. As their word sets overlap, the phrases are lexically similar but not semantically (Gomaa et al. [2013]).

Because extractors scrape text sections from a webpage without modifying the text block corresponding to the main content, the lexical similarity is the best for benchmarking extractors.

- **Levensthein Edit Distance** measures the dissimilarity between two blocks of text[1]. Edit distance is the minimal number of tokens that must be inserted, deleted, or substituted to change one text block to another. Each operation has a unit cost, which is normalized by dividing by the maximum number of tokens present in one of the text blocks. The score then ranges from 0 to 1. A score of 0 indicates that two blocks of text are identical, while a score of 1 indicates that they are entirely distinct.

  Example:

  s1 = A black dog chases a cat
  s2 = A lion chases a zebra

  For the text blocks s1 and s2, Edit distance is:

  $$ED = \frac{\mathrm{O}}{\mathrm{M}} = \frac{3}{6} = 0.5$$

  where:

  O : Minimum number operations required to make s1 and s2 identical
  M : Maximum token count in s1 or s2
  ED : Levensthein Edit distance

  The least number of operations necessary to make the text blocks s1 and s2 identical is 3. The lion token can be substituted with a dog token, the zebra token with a cat token, and the black token can be eliminated.

- **Jaccard Index** compares the members of two sets to discover which are unique and which are duplicates[2]. The score ranges from 0 to 1, with 0 being entirely distinct and 1 being identical. It estimates the overlap between two token sets. It is the intersection of two token sets by the union of two token sets. The intersection of

[1] https://en.wikipedia.org/wiki/Levenshtein_distance
[2] https://www.statisticshowto.com/jaccard-index/

tokens refers to the tokens shared by two blocks of text, whereas the union of tokens refers to the collection of tokens shared by both blocks.

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{4.1}$$

Example:

s1 = A black dog chases a cat
s2 = A lion chases a zebra

For the text blocks s1 and s2, Jaccard Index is:

s1 tokens = t1 = {A, black, dog, chases, a, cat}
s2 tokens = t2 ={A, lion, chases, a, zebra}

$$J(s1, s2) = \frac{|t1 \cap t2|}{|t1 \cup t2|} = \frac{3}{8}$$

- **Bag of Words** is a natural language programming technique that defines a phrase by the set of tokens used to construct the text[3]. Bag of Words is used to classify documents using the frequency of words as characteristics (wikipedia [2007]). Bag of Words is also utilized as a measure of text similarity. Bag of word text similarity is determined by the precision (equation 4.2), recall (equation 4.3, and F1 (equation 4.4) scores, with each score ranging from 0 to 1. A token is an alphanumeric word.

  Let x and y be two blocks of text where x is the gold standard and y is the predicted text of an extractor.
  Let xt and yt be set of tokens for the x and y blocks, respectively.

$$\text{true positive} = |xt \cap yt|$$

$$\text{false positive} = |yt - xt|$$

$$\text{false negative} = |xt - yt|$$

$$\text{Precision} = \frac{\text{true positive}}{\text{true positive} + \text{false positive}} \tag{4.2}$$

[3]https://machinelearningmastery.com/gentle-introduction-bag-words-model/

$$\text{Recall} = \frac{\text{true positive}}{\text{true positive} + \text{false negative}} \tag{4.3}$$

$$\text{F1} = \frac{2 \times \text{Precison*Recall}}{\text{Precision} + \text{Recall}} \tag{4.4}$$

- **N-Gram** are continuous sequences of n tokens extracted from a text block. In contrast to bag of words text similarity, N-Gram identifies token ordering in a text block. The precision, recall, and F1 score measure N-Gram text similarity. For example the sentence "A dog chases a cat for fun", all possible 4-gram would be (A, dog, chases, a), (dog, chases, a, cat), (chases, a, cat, for), (a, cat, for, fun). For benchmarking extractors based on the n-gram similarity metric, the n-value was set to 4. The value of 4 was selected to compare the findings to an extractors benchmark previously available on a corpus of 181 web pages[4]. The implementation of the N-Gram similarity measure is identical to that of the Bag of Words, except that the token count is considered in N-Gram.

- **RougeLSum** is the combination of Rouge-N and LCS (longest common subsequence)[5]. LCS detects the longest matching token sequence between two text blocks that are not necessarily sequential but are in order. The benefit of LCS is that it does not need consecutive matches; instead, it looks for matches within a sequence that corresponds to the word order at the block level. Since it automatically contains the longest consecutive common n-grams, the approach does not require a fixed length for the n-grams. Precision, recall and F1-score for RougeLSum can be defined as:

$$Precision : \frac{|LCS(s1, s2)|}{|TS_2|}$$

$$Recall : \frac{|LCS(s1, s2)|}{|TS_1|}$$

where:

[4]Konstantin Lopukhin, Evaluating the quality of article body extraction for commercial services and open-source libraries

[5]https://medium.com/free-code-camp/what-is-rouge-and-how-it-works-for-evaluation-of-summaries-e059fb8ac840

$s1$	: Gold standard text block
$s2$	: Predicted text block
TS_1	: Mutliset of tokens in s1
TS_2	: Multiset of tokens in s2
$LCS(s1, s2)$	: Longest common subsequence between s1 and s2

Example:

s1 = The bus is on the highway
s2 = A Red bus is on the road

For the text blocks s1 and s2, RougeLSum is:

LCS(s1, s2) = bus is on the
$\text{TS}_1 = \{The, bus, is, on, the, highway\}$
$TS_2 = \{A, Read, bus, is, on, the, road\}$

$$Precision = \frac{|LCS(s1, s2)|}{|TS_2|} = \frac{4}{7}$$

$$Recall = \frac{|LCS(s1, s2)|}{|TS_1|} = \frac{4}{6}$$

There are a total of five lexical similarity measures addressed in this section. However, as will be seen in the following sections, evaluating extractors using the aforementioned metrics is correlated for some metrics.

Chapter 5

Designing Ensemble Model Extractors

Open-source content extractors were explored in Section 2.2. These main content extractors utilize statistical methods or machine learning. In contrast, a new extractor based on an ensemble model and using existing content extractors as a foundation is developed and explained. Ensemble Modeling is a technique that entails the execution of two or more related but otherwise distinct models, followed by the integration of the results into a single value to improve performance. Three types of ensemble model methods exist: stacking, boosting, and bagging[1].

- **Stacking** is the process of fitting multiple models to the same data and integrating the findings of all models using additional models. The procedure consists of two steps. In the first phase, each model output based on input data is determined, and in the second step, the results from the first step are compiled into a single outcome.
- **Bagging** is the aggregation of various variants of a model. Each model is trained independently before being integrated. The objective of Bagging is to minimize variance, and Bagging is also known as bootstrap aggregation.
- **Boosting** combines multiple weak classifiers into a single robust classifier. First, a model is produced using the training data; subsequently, a second model is constructed using the first model's errors as the basis for its corrections. The number of models is increased until the model has a minimal bias or until a predetermined maximum number of models has been added.

The purpose of the ensemble model design was to combine the decisions of all viable extractors into one. The stacking ensemble model is the best solution in this situation. As

[1] https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-for-ensemble-models/

Table 5.1: Selection of extractors for the y parameter in threshold based ensemble model content extractor.

y	extractors
5	resiliparse, body text extraction, trafilatura, readability, go domdistiller
8	resiliparse, body text extraction, trafilatura, readability, go domdistiller, boilerpipe, justext, goose3

previously noted, Layer 1 of the stacking model consists of fitting multiple extractors to the same data, while Layer 2 consists of an aggregator model that aggregates the decisions of all the models' outputs from Layer 1. This study evaluated two distinct stacking ensemble models: 1) Layer 2 threshold-based binary classifier 2) Layer 2 machine learning binary classifier.

5.1 Threshold based Ensemble Content Extractor

The ensemble stacking model was designed as follows (Figure 5.1):

- Determine the main content of a web page using each extractor in layer 1.
- Extract all leaf nodes from the web page. Count, for each leaf node, the layer 1 extractors whose output includes the leaf node's text as the main content.
- If the number of extractors exceeds or equals the threshold value, the text of the leaf node is included in the main content.

In the above architecture, the parameters are the threshold value in layer 2 and the selection and number of content extractors in layer 1. The ensemble stacking model for the mentioned design was designated em_x_y, where x is the threshold value in layer 2 and y is the number of extractors in layer 1. Various combinations of x and y were selected to assess the extractor's performance (Table 5.1, Section 6.1).

5.2 Machine Learning based Ensemble Content Extractor

In the prior ensemble model-based extractor, the threshold value determined whether leaf node text should be included in the main text. It can, however, be substituted by a machine learning classifier, such as a logistic regression classifier or a random forest classifier. Using the machine learning classifier necessitates classifier training, which involves training and test data.
The ensemble stacking model was designed as follows (Figure 5.1)

- Determine the main content of a web page using each extractor in layer 1 when given a webpage.
- Extract all leaf nodes from the web page.
- For each leaf node, determine for each extractor in layer 1 whether the leaf node text is included in the main text. If the leaf node is part of the main text, set the content extractor feature to 1; otherwise, set it to 0.
- The features acquired in the previous step serve as input for the machine learning classifier at layer 2. If the prediction of the ml classifier is positive, the leaf node text is appended to the final main content text; otherwise, the leaf node is omitted.

em_y is the name of the ensemble stacking model for the above architecture, where y is the number of content extractors in layer 1. The chosen value of y was 13. In layer 1, thirteen extractors were selected, as described in section 2.2, except the web2text extractor, whose binary outputs served as features for the machine learning classifier in layer 2.

Before developing the ensemble model described above, it is necessary to train the ml classifier and evaluate its accuracy in identifying leaf nodes as the main content text or boilerplate. The dataset discussed in section 3.1 was divided into training and test sets for this purpose. 70% of the web pages were utilized for training the classifier, whereas 30% were used as the validation set. Each web page's leaf nodes were extracted, and each leaf node's text was used to train and test the ml classifier. The total number of features was thirteen, the number of extractors in layer 1. Each extractor's binary output served as the feature for layer 2.

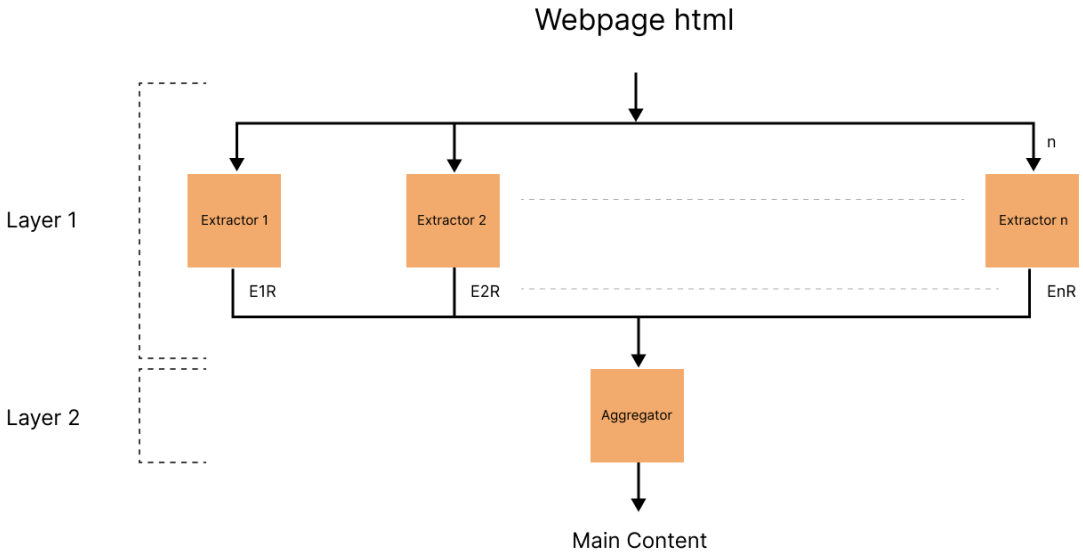
Table 5.2: ML based ensemble model content extractor results on training and validation data.

ML Classifier	Training Data			Validation Data	
	10 Fold Cross-Validation F1 Score		AUC Score	F1 Score	AUC Score
	Mean	Std deviation			
Logistic Regression	0.859	0.0013	0.926	0.820	0.924
Random Forest Classifier	0.872	0.00152	0.933	0.8355	0.925

Each feature had a binary value of 0 or 1, where 0 indicates that the leaf node text is not a subset of the extractor's main text block, and 1 indicates that it is a subset. Each page's gold standard was utilized to assess if the leaf node content is a subset of the gold standard. A total of 465K data points were collected, of which 233810 were classified negatively and 231801 were classed positively. The dataset was balanced before the classifier was trained.

Random Forest Classifier and Logistic Regression were trained to identify which classifier exhibited superior performance. The 10-Cross validation process with scoring metric F1 was utilized. Table 5.2 provides an overview of the results. The AUC represents the level of separation, and it describes whether or not the model can differentiate between positive and negative classes[2]. A model with an Auc score of 0.5 is equal to a random classifier and is incapable of class distinction. The greater the AUC score, the greater the model's ability to discriminate across classes. Table 5.2 demonstrates that Random Forest Classifier is the superior classifier for layer 2 due to its higher average F1 score for 10 fold cross-validation and higher F1 score on the validation set, as well as its higher AUC score. Due to its superior performance on training and validation data, the Random Forest Classifier was chosen over the Logistic Regression classifier in the layer 2 ml classifier.

[2]https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5



(a) Stacking Ensemble Model Extractor Design Layer Overview

- **Step A**
  - Initialize main_text = ""
  - Initialize threshold Θ (for binary threshold classifier)
  - Extract all leaf nodes from a webpage and loop over each leaf node (i)
- **Step B**
  - Threshold based ensemble model content extractor
    - Count (C), $E_N R$ with leaf node text is included in its main content.
    - check condition: $C >= \Theta$, if true proceed to Step C.
  - Machine Learning based ensemble model content extractor
    - Extract features of leaf node text.
    - Feed features into the ml classifier and predict if leaf node is a subset of the main content. If the prediction is positive, proceed to step C.
- **Step C**
  - main_text = main_text + i

(b) Layer 2 Aggregator Steps in 5.1a

Figure 5.1: Stacking Ensemble Model Extractor Design

Chapter 6

Evaluation

To effectively extract the main content of a webpage using one of the extractors (Section 2.2), it is essential to comprehend the performance of extractors in different data contexts. This part describes the benchmarking of extractors utilizing text similarity metrics in detail. The benchmarking of extractors is first addressed in several data contexts, followed by an insightful analysis of the benchmarks. This review aims to provide extensive insights and a conclusion on the extractors for extracting a webpage's main content.

6.1 Benchmarking of Extractors

A benchmark is a process of executing a program or group of programs to evaluate the relative performance of an object, often by subjecting it to a series of standard tests[1]. A benchmark is a measurement used to compare the performance of comparable entities. Extractor benchmarking allows an objective view of how well one extractor performs relative to others. 15 extractors were evaluated on various dataset groups discussed in Chapter 3.

6.1.1 Benchmarking using the Source Dataset

The section 3.1 discusses the source dataset group. The benchmarking of extractors on source dataset groups is illustrated in Table 6.1 and Table 6.2. The following are observations on benchmark scores:

- In terms of each particular statistic, no single extractor outperforms the others in the dataset group. For instance: em_3_5 ranks top in the article-extraction-benchmark

[1]https://en.wikipedia.org/wiki/Benchmark_(*computing*)

dataset, but web2text performs best on F1 score metrics (RougeLSum text similarity measure) in the readability dataset.

- In the article extraction benchmark dataset, web2text ranks last for the Levenshtein edit distance measure; however, it ranks second in the ClenEval dataset. The trafilatura extractor produces a comparable outcome.

A question raised by the benchmarks is: Why does the ranking vary so drastically when the dataset changes? One of the explanations may be attributable to the selection of web pages within each source dataset group. By analyzing the CleanEval dataset, it was discovered that virtually all of the web page's text was tagged as the main content, but in the article extraction benchmark dataset, just a few webpage sections were recognized as the main content (Table 3.3). Therefore, benchmarking extractors on the source dataset group yields no useful information regarding each extractor's performance. Based on these observations, web pages were then categorized according to their complexity (Section 3.5).

Table 6.1: Benchmarking of Extractors using Edit Distance Metric on Source Dataset. The Body Text Extraction (bte) performs best in the CETD dataset with the lowest Edit Distance score of 0.018, whereas in the RDB dataset, it performs average with a score of 0.048. The ranking of the extractors changes as the dataset changes, which indicates that benchmarking on the source dataset yields no helpful information.

Metric	**Edit Distance (Mean)**				
Source Dataset	**CETD**	**RDB**	**AEB**	**DG**	**CE**
Extractor					
boilerpipe	0.031	0.057	0.036	0.050	0.037
beautifulsoup	0.055	0.062	0.081	0.088	0.026
bte	**0.018**	0.048	0.039	0.047	0.022
em_13	0.053	0.054	0.063	0.077	0.030
em_2_5	0.018	0.044	0.026	0.045	0.021
em_3_5	0.022	0.044	0.018	0.045	0.022
em_4_5	0.028	0.053	0.015	0.050	0.032
em_4_8	0.023	0.046	0.021	0.044	0.021
em_6_8	0.030	0.058	0.020	0.050	0.031
go_domdistiller	0.024	0.041	0.022	0.048	0.024
goose3	0.033	0.064	0.026	0.053	0.042
html_text	0.055	0.062	0.080	0.087	0.026
html2text	0.052	0.058	0.072	0.080	0.025
inscriptis	0.037	0.043	0.057	0.061	**0.010**
justext	0.033	0.096	0.057	0.052	0.023
lxmlCleaner	0.052	0.050	0.057	0.078	0.028
readability	0.020	0.049	0.020	0.043	0.024
resiliparse	0.024	0.038	0.017	0.058	0.025
trafilatura	0.025	0.037	**0.015**	**0.041**	0.033
web2text	0.021	**0.030**	0.170	0.044	0.015
xpath_text	0.052	0.034	0.081	0.075	0.030

CETD: Content Extraction via Text Density.
RDB: Readability
AEB: Article Extraction Benchmark
DG: Dragnet
CE: CleanEval

Table 6.2: Benchmarking of Extractors using RougeLSum Metric on Source Dataset. The em_2_5 content extractor model performs best in the CETD dataset with an F1 score of 0.937, whereas in the RDB dataset, it performs average with a score of 0.906. The top performing content extractors across different source datasets considering RougeLSum F1 score are em_2_5, em_3_5, trafilatura and web2text. Web2text ranks highest in two of the source datasets RDB and CE.

Metric	F1 Score					Precision					Recall				
Source Dataset	**CETD**	**RDB**	**AEB**	**DG**	**CE**	**CETD**	**RDB**	**AEB**	**DG**	**CE**	**CETD**	**RDB**	**AEB**	**DG**	**CE**
Extractor															
boilerpipe	0.873	0.828	0.848	0.787	0.819	**0.993**	0.907	0.856	0.897	**0.950**	0.818	0.811	0.884	0.765	0.789
beautifulsoup	0.787	0.770	0.627	0.606	0.893	0.671	0.686	0.493	0.476	0.838	0.998	**0.999**	**1.000**	**0.996**	**0.999**
bte	0.936	0.836	0.820	0.813	0.899	0.942	0.837	0.769	0.778	0.903	0.947	0.885	0.933	0.914	0.934
em_13	0.805	0.829	0.733	0.671	0.879	0.702	0.763	0.619	0.553	0.850	0.982	0.981	0.989	0.983	0.953
em_2_5	**0.937**	0.907	0.906	0.831	0.910	0.980	0.934	0.868	0.883	0.917	0.906	0.915	0.974	0.849	0.933
em_3_5	0.921	0.895	**0.950**	0.826	0.903	0.990	0.948	0.956	0.934	0.929	0.875	0.881	0.956	0.800	0.912
em_4_5	0.895	0.863	0.950	0.784	0.847	0.991	0.964	0.956	0.938	0.926	0.835	0.826	0.956	0.738	0.826
em_4_8	0.915	0.896	0.928	0.823	0.904	0.992	0.957	0.909	0.930	0.935	0.865	0.879	0.969	0.798	0.908
em_6_8	0.886	0.798	0.919	0.788	0.847	0.992	0.903	**0.956**	**0.938**	0.939	0.823	0.755	0.909	0.736	0.820
go_domdistiller	0.911	0.929	0.914	0.800	0.877	0.989	0.950	0.889	0.899	0.904	0.862	0.923	0.962	0.781	0.887
goose3	0.876	0.762	0.877	0.775	0.796	0.984	0.866	0.931	0.928	0.882	0.811	0.721	0.859	0.721	0.777
html_text	0.787	0.770	0.627	0.606	0.893	0.671	0.686	0.493	0.476	0.838	0.998	**0.999**	**1.000**	**0.996**	**0.999**
html2text	0.786	0.773	0.633	0.604	0.885	0.669	0.696	0.499	0.476	0.835	0.998	0.991	**1.000**	0.989	0.987
inscriptis	0.792	0.788	0.642	0.621	0.890	0.678	0.712	0.510	0.492	0.839	0.997	0.996	**1.000**	0.995	0.991
justext	0.868	0.536	0.726	0.789	0.892	0.963	0.572	0.732	0.842	0.928	0.814	0.538	0.764	0.781	0.880
lxmlCleaner	0.803	0.844	0.750	0.662	0.871	0.693	0.779	0.641	0.542	0.836	0.996	0.994	0.989	0.991	0.965
readability	0.926	0.869	0.914	0.820	0.881	0.992	0.943	0.929	0.935	0.930	0.883	0.860	0.936	0.786	0.888
resiliparse	0.906	0.926	0.938	0.739	0.891	0.949	0.917	0.902	0.822	0.873	0.889	0.951	0.988	0.753	0.951
trafilatura	0.909	0.935	0.940	**0.843**	0.841	0.981	0.948	0.930	0.915	0.878	0.865	0.944	0.971	0.835	0.842
web2text	0.926	**0.966**	0.815	0.830	**0.932**	0.924	**0.981**	0.727	0.775	0.930	0.943	0.955	0.988	0.944	0.954
xpath_text	0.641	0.600	0.339	0.441	0.825	0.516	0.528	0.233	0.322	0.750	**0.999**	**0.999**	**1.000**	**0.996**	0.995

CETD: Content Extraction via Text Density.
RDB: Readability
AEB: Article Extraction Benchmark
DG: Dragnet
CE: CleanEval

6.1.2 Benchmarking using the Dataset Formed by Web Pages Complexity

It was noticed in the section 6.1.1 that benchmarking of extractors on respective source datasets does not produce meaningful information. Therefore, a new dataset group was created based on the complexity of the web pages (section 3.5). Tables 6.3 and 6.4 offers a quick overview of the benchmark scores. For complex webpages, em_3_5 scores highest, but simple extractors perform better than main content extractors for the easy dataset group.

The benchmark score makes it easy to select an extractor based on a web page's complexity, which builds credibility in the extracted main content of a webpage through the judicious selection of an extractor. The score indicates that selecting a simple extractor over others is sometimes preferable to prevent loss of main content yet minimise the noisy content.

Table 6.3: Benchmarking of extractors using Edit Distance Metric on Complexity Dataset. The em_2_5 content extractor performs best in the complex dataset with the lowest edit distance score of 0.040, whereas in the easy dataset, it performs average with a score of 0.028. The benchmark shows that the lxml cleaner used by trafilatura internally for removing generic boilerplates performs much better on the easy dataset. The result shows that a simple extractor should be preferred for the easy dataset.

Metric	**Edit Distance (Mean)**		
Complexity Dataset	**Complex**	**Mid-Complex**	**Easy**
Extractor			
boilerpipe	0.051	0.034	0.049
beautifulsoup	0.123	0.063	0.013
bte	0.070	0.027	0.015
em_13	0.111	0.056	0.016
em_2_5	0.040	0.030	0.028
em_3_5	0.047	0.028	0.026
em_4_5	0.048	0.034	0.041
em_4_8	0.042	0.029	0.032
em_6_8	0.048	0.034	0.042
go_domdistiller	0.048	0.030	0.034
goose3	0.049	0.040	0.050
html_text	0.122	0.063	0.013
html2text	0.108	0.059	0.014
inscriptis	0.079	0.043	0.008
justext	0.074	0.033	0.032
lxmlCleaner	0.109	0.056	0.014
readability	0.040	0.029	0.031
resiliparse	0.057	0.037	0.030
trafilatura	0.042	0.030	0.032
web2text	0.078	0.033	0.015
xpath_text	0.086	0.065	0.019

Table 6.4: Benchmarking of extractors using RougeLSum Metric (Mean) on Complexity Dataset. The web2text content extractor performs best in the mid-complex dataset with an F1 score of 0.921, whereas in the complex dataset, it performs below average with a score of 0.709. The top performing content extractors are em_3_5, web2text and beautifulsoup. Selecting a simple content extractor for a webpage belonging to the easy dataset category should be preferred.

Metric	**F1 Score (Mean)**			**Precision (Mean)**			**Recall (Mean)**		
Complexity Dataset	**Complex**	**Mid-Complex**	**Easy**	**Complex**	**Mid-Complex**	**Easy**	**Complex**	**Mid-Complex**	**Easy**
Extractor									
boilerpipe	0.769	0.866	0.773	0.797	0.964	0.985	0.797	0.829	0.710
beautifulsoup	0.382	0.759	**0.961**	0.246	0.623	0.932	0.999	0.998	0.995
bte	0.687	0.905	0.941	0.641	0.886	0.956	0.862	0.950	0.939
em_13	0.477	0.797	0.951	0.336	0.682	0.947	0.983	0.981	0.962
em_2_5	0.815	0.899	0.904	0.785	0.941	0.984	0.911	0.890	0.870
em_3_5	**0.841**	0.892	0.872	0.873	0.965	0.985	0.854	0.863	0.834
em_4_5	0.789	0.866	0.822	0.881	0.968	0.981	0.766	0.824	0.767
em_4_8	0.830	0.892	0.866	0.857	0.967	0.988	0.850	0.863	0.824
em_6_8	0.781	0.865	0.812	0.873	0.971	0.985	0.751	0.817	0.753
go_domdistiller	0.797	0.883	0.849	0.820	0.955	0.957	0.824	0.856	0.819
goose3	0.778	0.840	0.774	0.854	0.957	0.944	0.750	0.789	0.718
html_text	0.382	0.759	0.961	0.246	0.623	0.932	0.999	0.998	0.995
html2text	0.386	0.754	0.958	0.250	0.621	0.930	0.999	0.988	0.990
inscriptis	0.402	0.767	0.961	0.264	0.635	0.934	0.998	0.995	0.992
justext	0.659	0.870	0.857	0.676	0.931	0.939	0.702	0.840	0.813
lxmlCleaner	0.473	0.791	0.949	0.335	0.673	0.934	0.992	0.988	0.977
readability	0.828	0.883	0.862	0.878	0.963	0.980	0.829	0.852	0.832
resiliparse	0.748	0.846	0.873	0.737	0.891	0.959	0.828	0.854	0.857
trafilatura	0.827	0.889	0.863	0.845	0.950	0.950	0.861	0.867	0.829
web2text	0.709	**0.921**	0.957	0.622	0.894	0.975	0.925	0.963	0.944
xpath_text	0.227	0.566	0.910	0.135	0.421	0.858	0.999	0.996	0.995

6.1.3 Benchmarking using the Dataset Formed by K-Means Clustering Algorithm

It was made apparent in section 6.1.2 that the choice of extractor serves an essential function for a webpage. However, grouping webpages by complexity was based on its gold standard. In an ideal scenario, just the webpage is accessible, not its gold standard; this is the issue that extractors are attempting to address. Grouping webpages using K-means cluster analysis was conducted for this purpose (Section 3.6). The overview of extractors' performance for each cluster is presented in Table 6.5 and 6.6. Webpages were categorized into three clusters to correspond with the number of complexity groups. When a new web page's main content is to be extracted, the first step would be to determine the cluster it belongs to and then uses the extractor with the highest performance in that cluster.

Table 6.5: Benchmarking of extractors using Edit Distance Metric on K-Means Cluster Dataset. The em_3_5 and readability content extractors perform best in the label 0 dataset with the lowest edit distance score of 0.035, whereas in the label 2 dataset, simple extractor inscriptis ranks highest with a score of 0.019.

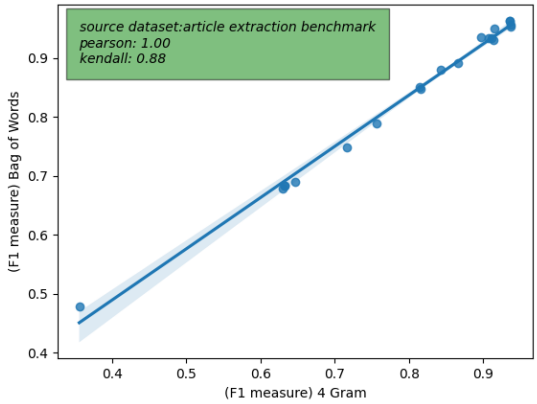
Metric	**Edit Distance (Mean)**		
K-Means Cluster Dataset	**Label 0**	**Label 1**	**Label 2**
Extractor			
boilerpipe	0.041	0.043	0.045
bs4	0.083	0.065	0.036
bte	0.040	0.035	0.025
em_13	0.074	0.059	0.035
em_2_5	0.037	0.029	0.028
em_3_5	0.035	0.029	0.029
em_4_5	0.042	0.036	0.037
em_4_8	0.036	0.032	0.029
em_6_8	0.041	0.039	0.039
go_domdistiller	0.038	0.036	0.031
goose3	0.044	0.041	0.049
html_text	0.082	0.064	0.036
html2text	0.076	0.060	0.034
inscriptis	0.056	0.045	0.019
justext	0.047	0.046	0.035
lxmlCleaner	0.073	0.058	0.034
readability	0.035	0.030	0.029
resiliparse	0.048	0.032	0.031
trafilatura	0.036	0.031	0.032
web2text	0.048	0.046	0.022
xpath_text	0.072	0.055	0.037

Table 6.6: Benchmarking of extractors using RougeLSum Metric on K-Means Cluster Dataset. The top performing extractor in label 0, label 1 and label 2 are em_3_5, em_2_5 and web2text respectively. One of the observations from the benchmark is that the main content extractors perform better than the simple extractor in all the dataset categories.

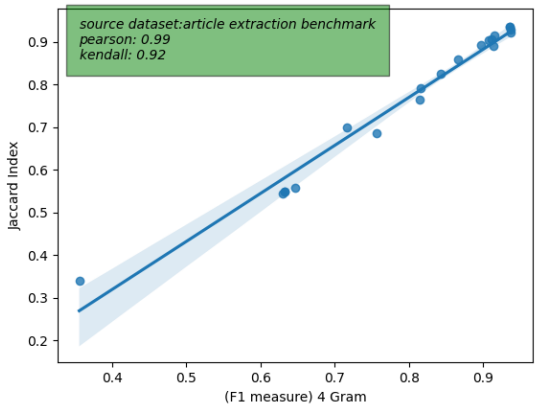
Metric	**F1 Score (Mean)**			**Precision (Mean)**			**Recall (Mean)**		
K-Means Dataset	**Label 0**	**Label 1**	**Label 2**	**Label 0**	**Label 1**	**Label 2**	**Label 0**	**Label 1**	**Label 2**
Extractor									
boilerpipe	0.833	0.813	0.796	0.910	0.948	0.945	0.814	0.780	0.759
bs4	0.636	0.726	0.853	0.504	0.612	0.787	0.998	0.999	0.994
bte	0.841	0.860	0.893	0.812	0.857	0.887	0.924	0.917	0.935
em_13	0.692	0.766	0.863	0.573	0.664	0.822	0.981	0.986	0.962
em_2_5	0.865	0.894	0.895	0.900	0.925	0.927	0.880	0.898	0.904
em_3_5	0.868	0.885	0.878	0.942	0.960	0.947	0.844	0.858	0.868
em_4_5	0.829	0.847	0.839	0.946	0.960	0.948	0.788	0.802	0.805
em_4_8	0.863	0.874	0.881	0.935	0.956	0.953	0.843	0.847	0.866
em_6_8	0.832	0.832	0.828	0.943	0.961	0.954	0.785	0.781	0.786
go_domdistiller	0.850	0.852	0.861	0.917	0.938	0.918	0.832	0.827	0.858
goose3	0.817	0.826	0.778	0.930	0.956	0.905	0.767	0.771	0.745
html_text	0.636	0.726	0.853	0.504	0.612	0.787	0.998	0.999	0.994
html2text	0.634	0.726	0.848	0.503	0.613	0.787	0.992	0.999	0.984
inscriptis	0.650	0.728	0.857	0.520	0.617	0.794	0.997	0.997	0.990
justext	0.803	0.806	0.839	0.852	0.896	0.882	0.797	0.771	0.821
lxmlCleaner	0.688	0.764	0.856	0.567	0.659	0.808	0.992	0.994	0.971
readability	0.858	0.874	0.868	0.946	0.952	0.942	0.826	0.849	0.863
resiliparse	0.788	0.870	0.872	0.842	0.915	0.889	0.803	0.875	0.911
trafilatura	0.865	0.881	0.860	0.926	0.951	0.901	0.855	0.853	0.859
web2text	0.856	0.866	0.924	0.808	0.847	0.915	0.953	0.932	0.954
xpath_text	0.446	0.597	0.767	0.323	0.484	0.688	0.999	1.000	0.992

6.2 Extractors Rank Correlation Among Text Similarity Metrics

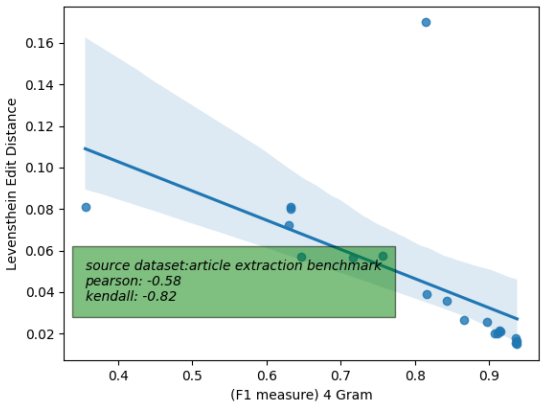
Five text-based similarity metrics were explored in Chapter 4; however, only the RougeLSum and Edit Distance metrics were reported in section 6.1 for benchmarking. For each dataset group, it was determined that RougeLSum, Jaccard Index, Bag of Words, and 4-Gram have a high correlation with one another (Figure 6.1d, 6.1g, 6.1h, 6.1b, 6.1a). Figure 6.1 only depicts the rank correlation plot for the article extraction benchmark dataset, but similar trends can be observed in other dataset groups. An extractor with a high ranking in RougeLSum will almost certainly have a relative ranking in the other text similarity measures listed above. If the ranking in one of the metrics mentioned above is known, it is possible to forecast the extractor's performance on any other measure. There appears to be a low-rank correlation between the text similarity measures mentioned above and the Levenshtein edit distance (Figure 6.1c, 6.1e, 6.1i, 6.1f). Because of the above reasoning, the RougeLSum and Levenshtein Edit Distance findings were recorded in section 6.1.



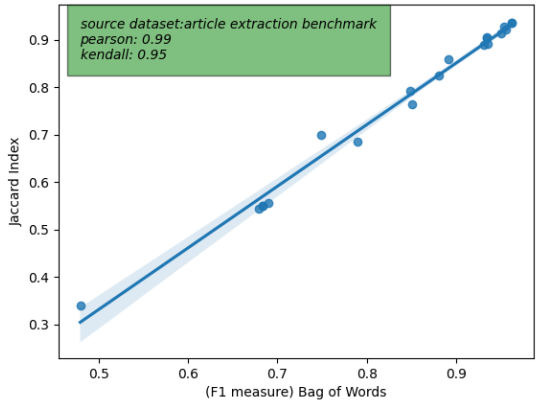
(a) 4 Gram vs Bag of Words



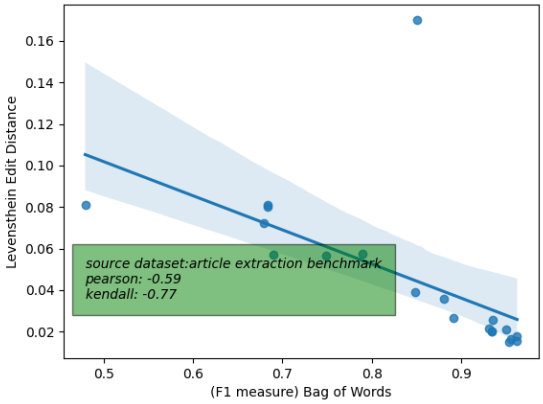
(b) 4 Gram vs Jaccard Index



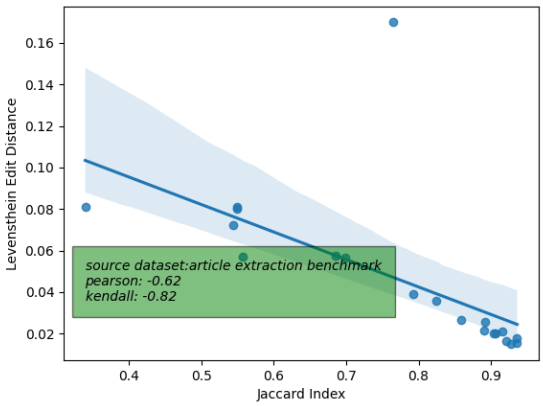
(c) 4 Gram vs Levensthein Edit Distance



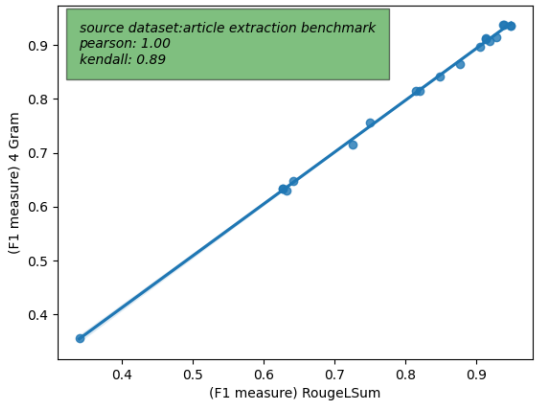
(d) Bag of Words vs Jaccard Index



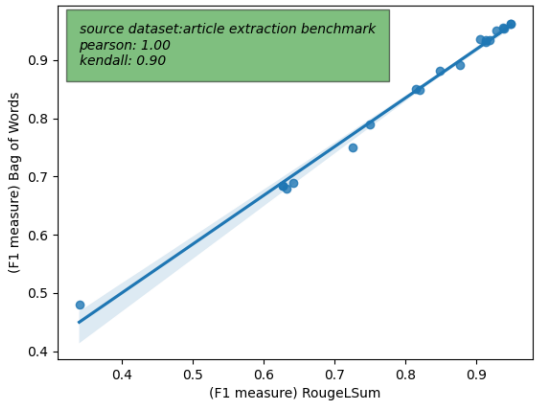
(e) Bag of Words vs Levensthein Edit Distance



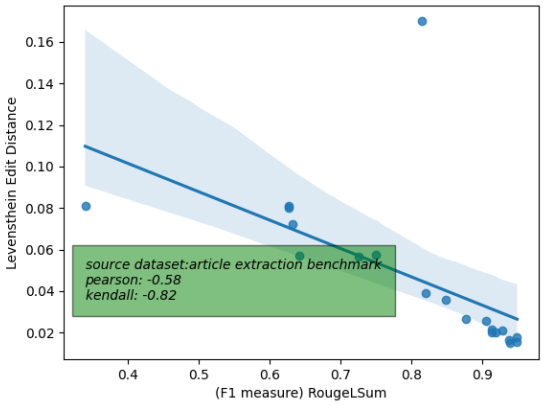
(f) Jaccard Index vs Levensthein Edit Distance



(g) RougeLSum vs 4 Gram



(h) RougeLSum vs Bag of Words

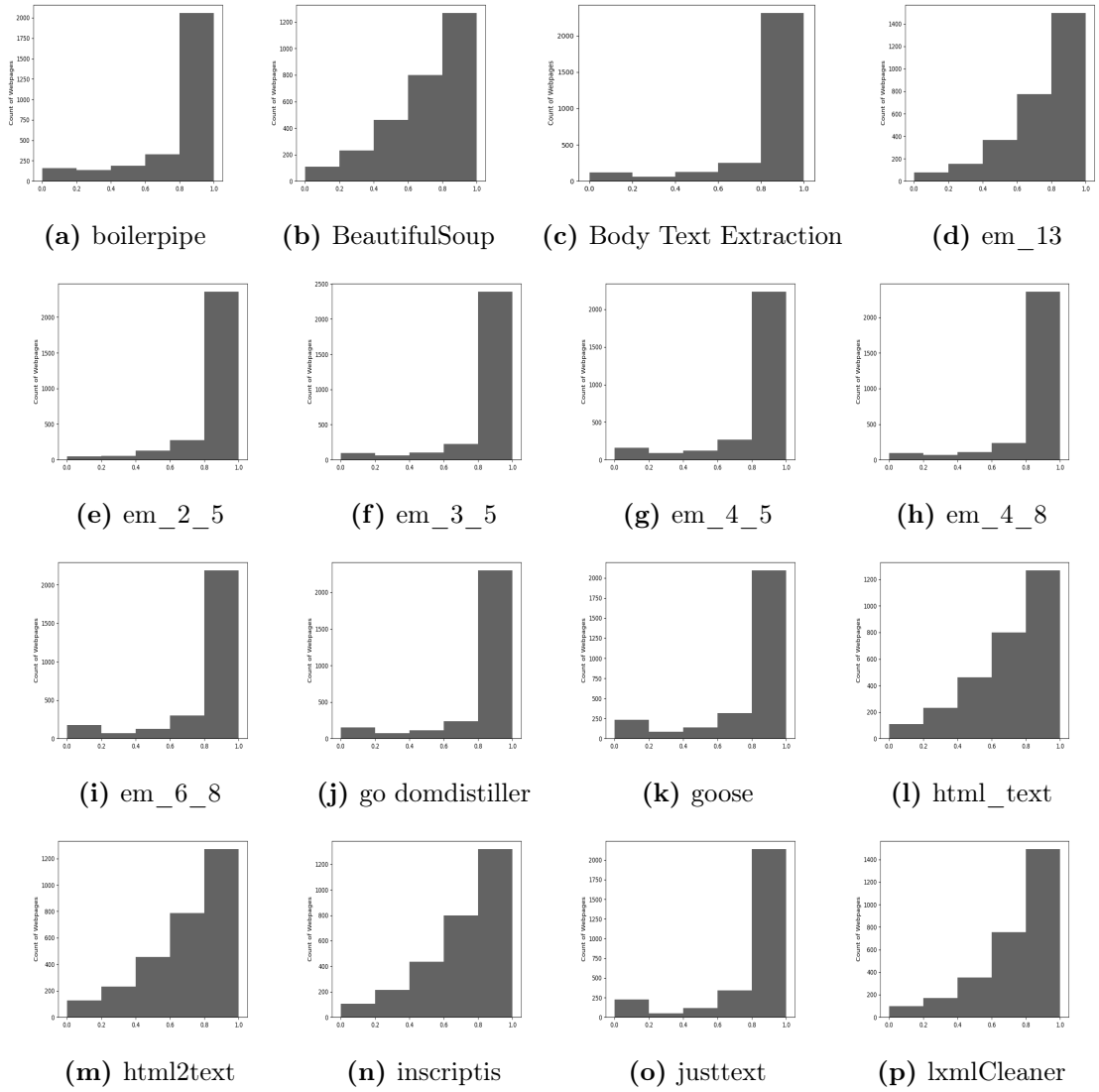


(i) RougeLSum vs Levensthein Edit Distance

Figure 6.1: Rank correlation plot of extractors among text similarity measures for Article Extraction Benchmark dataset. RougeLSum and Levenshtein edit distance benchmark scores were documented due to their low-rank correlation.

6.3 Most Effective Score Distribution Across Web Pages for all the Benchmarked Extractors

Figure 6.2 depicts the RougeLSum F1 measure distribution for evaluating extractors' performance across all web pages. The majority of webpages have an F1 score between 0.8 and 1 for the main content extractors like trafilatura(6.2s), web2text(6.2t), and body text extraction(6.2c). In contrast, the F1 measure distribution has a significant number of web pages in each bin for the simple extractors such as html2text(6.2m), xpath text(6.2u), beautifulsoup(6.2b), inscriptis(6.2n) and em_13 (a machine learning based ensemble model, section 5).

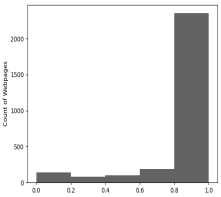


(a) boilerpipe **(b)** BeautifulSoup **(c)** Body Text Extraction **(d)** em_13

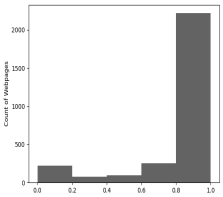
(e) em_2_5 **(f)** em_3_5 **(g)** em_4_5 **(h)** em_4_8

(i) em_6_8 **(j)** go domdistiller **(k)** goose **(l)** html_text

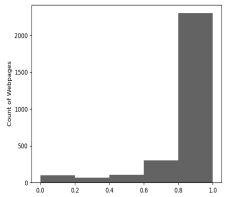
(m) html2text **(n)** inscriptis **(o)** justtext **(p)** lxmlCleaner



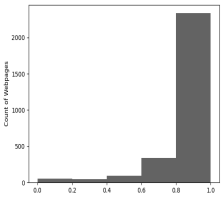
(q) readability



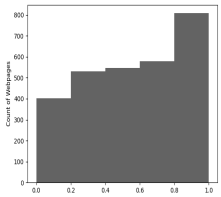
(r) resiliparse



(s) trafilatura



(t) web2text



(u) xpath_text

Figure 6.2: Distribution plot of RougeLSum F1 measure for each content extractor. X-axis: F1 measures between 0 and 1 with a bin size of 0.2, Y-axis: count of web pages. The majority of web pages have an F1 score between 0.8 and 1 for the main content extractors in contrast to the simple extractors, which have many web pages in each bin.

Chapter 7

Conclusion and Future Works

In the thesis, we have covered the definition of content extraction and its implementation. We studied the widely used extractors' benchmark scores by extending prior content extraction and benchmarking research. As shown, one extractor does not outperform the others, and selecting a suitable extractor for the webpage to extract its primary content is crucial. We have also observed that the dataset has a binding effect on the conclusion. The thesis began with the Introduction (Chapter 1), which emphasized the prevalence of unstructured data on the web and the need to extract the main content and reduce boilerplate. We reviewed the HTML boilerplate elements and the benefits of automated and manual content extraction.

In Chapter 2, we examined the open-source extractors that can be utilized for research purposes and their requirements. In addition, we analyzed the existing research publications on content extraction, the authors' content extraction model technique, and the dataset information.

In Chapter 3, we explored the sources of labelled datasets and the format of the gold standard. We first determined the diversity of the web pages in the corpus. We discovered that web pages were classified into 28 groups, indicating that the dataset was sufficiently diversified. We assessed dataset error, enhanced the dataset's quality, and catalogued the forms of error in the gold standard. We also discussed categorizing web pages into groups based on their complexity or using the K-means clustering algorithm. The objective was to group webpages with similar features; as we discovered in subsequent chapters, simple extractors work better than main content extractors on a few dataset groups.

Chapter 4 discussed semantic and lexical text similarity metrics and elaborated on the many lexical similarity measures, including the Jaccard index, RougeLSum, N-Gram, Bag

of Words, and Levenshtein edit distance. In a subsequent chapter, it was discovered that there appears to be a high-rank correlation between text similarity measure scores. For this reason, only the RougeLSum and the Levenshtein edit distance scores were recorded.

In Chapter 5, we explored the design and implementation of two content extractors based on the stacking ensemble model, with layer 2 serving as a threshold-based binary classifier and a machine learning-based binary classifier. We evaluated Logistic Regression and Random Forest Classifier in a machine-learning-based classifier to identify whether a leaf node contains boilerplate text or main content. By assessing the benchmarking results, it was discovered that a threshold-based binary classifier appears to perform better than one based on machine learning.

In Chapter 6, we compared the outcomes of benchmarking extractors on several dataset groupings. We determined that a single extractor is inadequate for all web pages. On several web pages, it was discovered that the simple extractors outperform the main content extractors based on machine learning and statistics. The selection of an extractor relies on the webpage. Complex extractors should not always be selected since doing so might result in losing a few main content sections.

In the study, we researched unexplored areas of content extractor benchmarking and developed an ensemble-based content extractor. Nevertheless, there is an opportunity for improvement as several aspects remain unanswered.

Extractor based on machine learning was addressed in section 5.2; however, its performance on the benchmarking tests was average. The extractor seems to perform poorly in complex dataset group as well. The model may be improved by incorporating additional features and experimenting with other classifiers as an aggregator in layer 2, such as the support vector machine, decision trees, etc.

In Section 3.3, we determined that the dataset was partitioned into 28 distinct categories, such as news, fitness, and arts entertainment, but no benchmarking was conducted on each category to establish the optimal extractors for each category. For news-related webpages, for instance, does the trafilatura extractor perform better than readability?

We covered K-means clustering and complexity-based grouping of webpages in Section 3.5 and 3.6. However, benchmarking scores (Section 6.1) using the complexity group was more intuitive than the K-means clustering group. As complexity-based grouping of web pages is based on the gold standard, which appears to be non-ideal in real-world scenarios, the future steps could be to group webpages similar to complexity-based without using the gold standard, either through machine learning or clustering-based techniques.

Bibliography

Adrien Barbaresi. Trafilatura: A web scraping library and command-line tool for text discovery and extraction. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*, pages 122–131, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-demo.15. URL https://aclanthology.org/2021.acl-demo.15.

Marco Baroni, Francis Chantree, Adam Kilgarriff, and Serge Sharoff. Cleaneval: a competition for cleaning web pages. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco, May 2008. European Language Resources Association (ELRA). URL http://www.lrec-conf.org/proceedings/lrec2008/pdf/162_paper.pdf.

Stefan Behnel. Lxml, 2011. URL https://github.com/lxml/lxml.

Janek Bevendorff, Benno Stein, Matthias Hagen, and Martin Potthast. Elastic ChatNoir: Search Engine for the ClueWeb and the Common Crawl. In Leif Azzopardi, Allan Hanbury, Gabriella Pasi, and Benjamin Piwowarski, editors, *Advances in Information Retrieval. 40th European Conference on IR Research (ECIR 2018)*, Lecture Notes in Computer Science, Berlin Heidelberg New York, March 2018. Springer.

Deng Cai, Shipeng Yu, Ji-Rong Wen, and Wei-Ying Ma. Extracting content structure for web pages based on visual representation. In Xiaofang Zhou, Maria E. Orlowska, and Yanchun Zhang, editors, *Web Technologies and Applications*, pages 406–417, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-36901-1.

Courtney D Corley and Rada Mihalcea. Measuring the semantic similarity of texts. In *Proceedings of the ACL workshop on empirical modeling of semantic equivalence and entailment*, pages 13–18, 2005.

dragnet org. dragnet-org/dragnet, 2012. URL `https://github.com/dragnet-org/dragnet`.

Aidan Finn, Nicholas Kushmerick, and Barry Smyth. Fact or fiction: Content classification for digital libraries. In *DELOS*, 2001.

Lichao Zhu Gaël Lejeune. A new proposal for evaluating web page cleaning tools. Marrakech, Morocco, May 2020. Computación y sistemas, Instituto Politécnico Nacional IPN Centro de Investigación en Computación, 2018, ff10.13053/CyS-22-4-3062f. URL `https://hal.archives-ouvertes.fr/hal-02467732/document`.

Wael H Gomaa, Aly A Fahmy, et al. A survey of text similarity approaches. *international journal of Computer Applications*, 68(13):13–18, 2013.

Thomas Gottron. Content code blurring: A new approach to content extraction. *2008 19th International Conference on Database and Expert Systems Applications*, 2008. doi: 10.1109/dexa.2008.43.

GravityLabs. goose, 2010. URL `https://github.com/GravityLabs/goose`.

David Insa Cabrera, Josep Francesc Silva Galiana, and Salvador Tamarit. Using the words/leafs ratio in the dom tree for content extraction. *Journal of Logic and Algebraic Programming*, 82(8):311–325, 2013.

Christian Kohlschütter, Peter Fankhauser, and Wolfgang Nejdl. Boilerplate detection using shallow text features. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, WSDM '10, page 441–450, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605588896. doi: 10.1145/1718487.1718542. URL `https://doi.org/10.1145/1718487.1718542`.

scrapinghub Konstantin Lopukhin. Evaluating quality of article body extraction for commercial services and open-source libraries. URL `https://github.com/scrapinghub/article-extraction-benchmark#more-details`.

Tagaram Soni Madhulatha. Comparison between k-means and k-medoids clustering algorithms. In David C. Wyld, Michal Wozniak, Nabendu Chaki, Natarajan Meghanathan, and Dhinaharan Nagamalai, editors, *Advances in Computing and Information Technology*, pages 472–481, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-22555-0.

Markus Mobius and Radhi Fadlillah. markusmobius/go-domdistille, 2020. URL `https://github.com/markusmobius/go-domdistiller.git`.

Mozilla. mozilla/readability, 2015. URL `https://github.com/mozilla/readability`.

Jan Pomikálek. *Removing boilerplate and duplicate content from web corpora*. PhD thesis, Masaryk university, Faculty of informatics, Brno, Czech Republic, 2011.

Leonard Richardson. Beautifulsoup, 2010. URL `https://code.launchpad.net/beautifulsoup/`.

Fei Sun, Dandan Song, and Lejian Liao. Dom based content extraction via text density. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, page 245–254, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450307574. doi: 10.1145/2009916.2009952. URL `https://doi.org/10.1145/2009916.2009952`.

Aaron Swartz. html2text, 2014. URL `https://github.com/soundasleep/html2text`.

TeamHG-Memex. Html to text, 2016. URL `https://github.com/TeamHG-Memex/html-text`.

videoinu. How does firefox's reader view work?, 2020. URL `https://videoinu.com/blog/firefox-reader-view-heuristics/`.

Thijs Vogels, Octavian-Eugen Ganea, and Carsten Eickhoff. Web2text: Deep structured boilerplate removal. *CoRR*, abs/1801.02607, 2018. URL `http://arxiv.org/abs/1801.02607`.

weblyzard. inscriptis – html to text conversion library, command line client and web service, 2016. URL `https://github.com/weblyzard/inscriptis`.

Tim Weninger, William H. Hsu, and Jiawei Han. Cetr: Content extraction via tag ratios. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, page 971–980, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605587998. doi: 10.1145/1772690.1772789. URL `https://doi.org/10.1145/1772690.1772789`.

wikipedia. Bag-of-words model, 2007. URL `https://en.wikipedia.org/wiki/Bag-of-words_model`.