Bauhaus-Universität Weimar
Faculty of Media
Degree Programme Computer Science and Media

Content Extraction from Webpages Using Machine Learning

Master's Thesis

Hamza Yunis
Born Dec. 3, 1988 in Damascus, Syria

Matriculation Number 115233

1. Referee: Prof. Dr. Benno Stein
2. Referee: Dr. Andreas Jakoby

Submission date: December 16, 2016

Declaration

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

Weimar, December 16, 2016

. . . .
Hamza Yunis

Abstract

The content extraction problem has been a subject of study ever since the expansion of the World Wide Web. Its goal is to separate the main content of a webpage, such as the text of a news story, from the noisy content, such as advertisements and navigation links.

Most content extraction approaches operate at a block level; that is, the webpage is segmented into blocks and then each of these blocks is determined to be part of the main content or the noisy content of the webpage.

In this thesis, we try to apply content extraction at a deeper level, namely to HTML elements. During the course of the thesis, we investigate the notion of main content more closely, create a dataset of webpages whose elements have been manually labeled as either part of the main content or the noisy content, and apply machine learning to this dataset in order to induce rules for separating the main content and the noisy content. Finally, these induced rules are evaluated using a different dataset of manually labeled webpages.

Contents

Chapter 1

Introduction

1.1 Motivation

The webpages[1] (also referred to web documents) that constitute the World Wide Web are sources of very diverse categories of information. These include news, reference materials, forum discussions, and commercial product descriptions, just to name a few. Each category of information can in turn have various media formats, such as textual, graphical, or video. This vast amount of information is used by ordinary web users throughout the world, as well as by automated crawlers that traverse the Web for various purposes, such as web mining or web indexing.

In most cases, however, a single webpage consists of distinct "parts," which will be referred to in this thesis as the **contents** of the webpage. Only one type of content, which will be referred to as the **main content** of the webpage, is what makes the webpage a useful source of information. Other contents include advertisements, navigation buttons, page settings, and legal notices; these contents will be collectively referred to as the **noisy content** of the webpage. The process of identifying the main content of a webpage is called **main content extraction**, or more briefly **content extraction**.[2]

For most webpages, a human user can intuitively and quickly identify the main content. However, from an HTML markup perspective or from a DOM perspective, the main and the noisy contents are closely intermingled; therefore, separating them presents a significant challenge for automated information extractors. Due to the fact that webpages can have countless different formats (at both the structure layer and the style layer), there are no universal rules for accurately separating the main content and the noisy content.

The goal of this thesis is to induce new rules for content extraction using

[1] The solid spelling *webpage* will be used in this work instead of *web page*.

[2] This is the commonly used term in literature.

supervised machine learning algorithms based on a sample of webpages with manually labeled contents; that is, the contents of these webpages have been identified as main or noisy by a human annotator. In addition, the content extraction performance under these rules should be evaluated.

Machine learning has previously been used for content extraction [Louvan, 2009], [Zhou and Mashuq, 2014] and other similar tasks, such as spam email detection [Guzella and Caminhas, 2009] and Wikipedia vandalism detection [Smets et al., 2008]; these tasks are similar to content extraction in the sense that a human user can relatively easily identify a spam email or a vandalistic Wikipedia edit, but these are not straightforward tasks for computer programs.

The approach that is used in this work relies on a combination of ideas that have been used in earlier works. In addition, newly-introduced ideas are utilized, in particular, the inspecting of the *context* of a specific webpage element, as discussed in Section 3.8.2.

1.2 Importance of Content Extraction

Identifying the main content of a webpage is useful for various applications. One such an application is web mining, which is the application of data mining to the World Wide Web. In general, data mining attempts to extract useful information from a large data set. In web mining, the data set consists of webpages. Therefore, it is imperative, when carrying out web mining, to separate the main content from the noisy content of webpages, so that the latter is discarded and not used in the mining process.

Another application where content extraction is important is web search engines. Web search engines use crawlers to traverse the Web and copy the content of each document they visit into a *document data store* [Croft et al., 2010]. When processing a user query, the web search engine uses a *ranking algorithm* that identifies the relevance of each document in the document data store to the given query. For this purpose, the ranking of a document should depend only on its main content. For example, Figure 1.1 shows the upper part of a news article webpage from `http://reuters.com`. The webpage has a section called *Tending Stories*, which contains links to trending stories at the time the webpage was accessed. The textual content of these links should *not* be considered when ranking the webpage because they are not related to the subject of the webpage.

Content extraction can be useful not only for automated crawlers, but also for human users. For instance, content extraction can be used to set the focus on the main content when rendering webpages on small-screen devices, such as mobile phones and PDA's, so that the user does not have to scroll and search

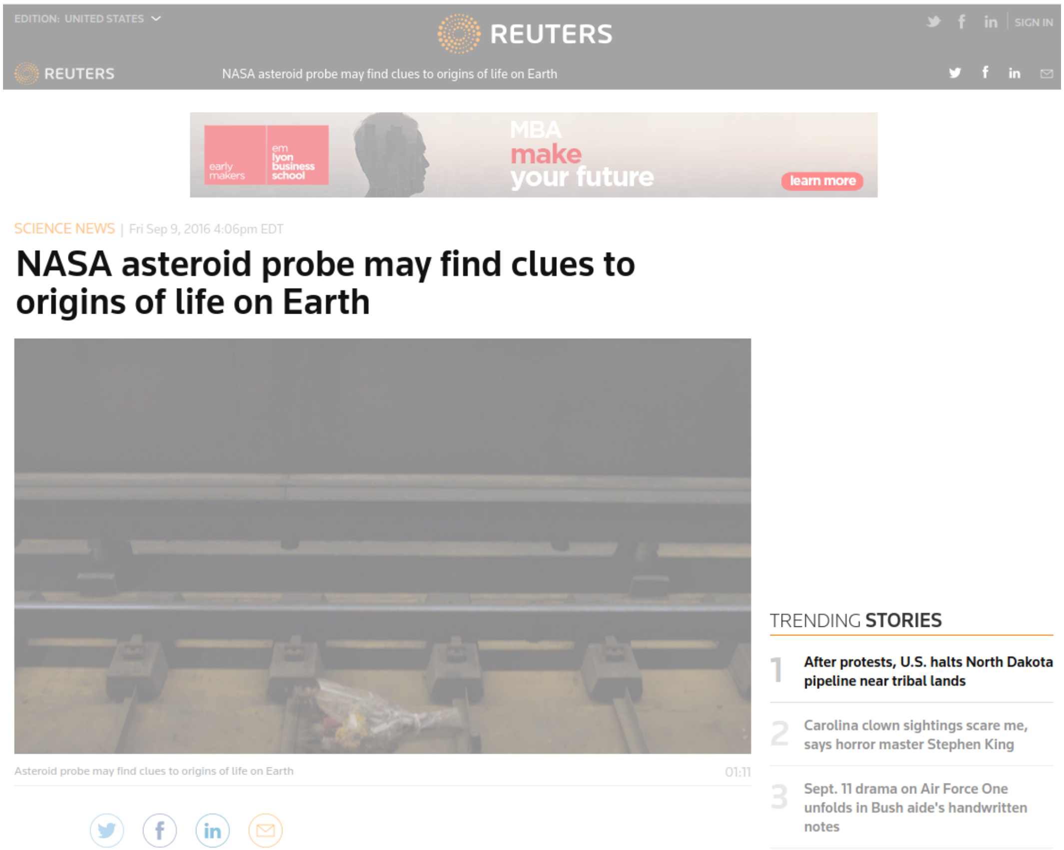


Figure 1.1: A screenshot of the upper part of a news story from http://reuters.com. Obviously, the story is not about the U.S. decision to halt the construction of the North Dakota pipeline (listed in the *Trending Stories* section). Thus, a search query like “North Dakota pipeline” should not lead to this webpage.

for the main content. Additionally, content extraction is especially important for visually-impaired or unsighted users, where the main content has to be visually emphasized or synthetically read aloud.

1.3 Thesis Organization

The remainder of this work is organized as follows:

- Chapter 2 provides a survey of previously developed approaches for content extraction.
- Chapter 3 provides a deeper inspection of the concept of content extraction, a formulation of content extraction as a classification problem that

can be handled by machine learning, a description of the manual annotation process of webpages, and a description of the features to be used in the learning process.

- Chapter 4 provides a description of the learning process and the evaluation scores of the content extractor that the learning process has produced.
- Chapter 5 provides a recapitulation of this work, along with list of potential improvements to the applied approach.

Chapter 2

Related Work

Since the expansion of the World Wide Web, numerous methods for content extraction have been proposed, many of of which were developed in the context of one of the applications of content extraction, rather than when treating the problem of content extraction itself [Gottron, 2009]. Many of these methods rely on heuristics, which can be applied to

- the HTML source of the webpage; or
- the DOM tree of the webpage; or
- the visual rendering of the webpage.

Sections 2.1, 2.2, and 2.3 give an example of each of these types of methods, respectively. In addition, Sections and 2.4 and 2.5 give an overview of wrappers and template recognition.

2.1 Body Text Extraction

Body Text Extraction (BTE) was introduced and described by Finn et al. [2001] as a method for identifying the main textual content of a webpage, which they refer to as the *main body of text*. The BTE algorithm is based on the observation that the main body of text of a webpage consists primarily of text and very little markup.

BTE starts by assigning all tokens in the HTML source of the webpage into one of two categories: HTML tag tokens and word tokens. Consequently, the webpage is viewed as a sequence $\{B_i\}$ of bits, with $B_i = 1$ when the ith token is a tag, and $B_i = 0$ when the ith token is a word. This sequence can be represented by the *document slope curve*, as shown in Figure 2.1. A point (x, y) that lies on the curve basically tells us: In the first x tokens of the webpage, there are y tag tokens. Therefore, segments that have low slope, usually referred to as plateaus, correspond to portions in the webpage source that have a small number HTML tags inside them.

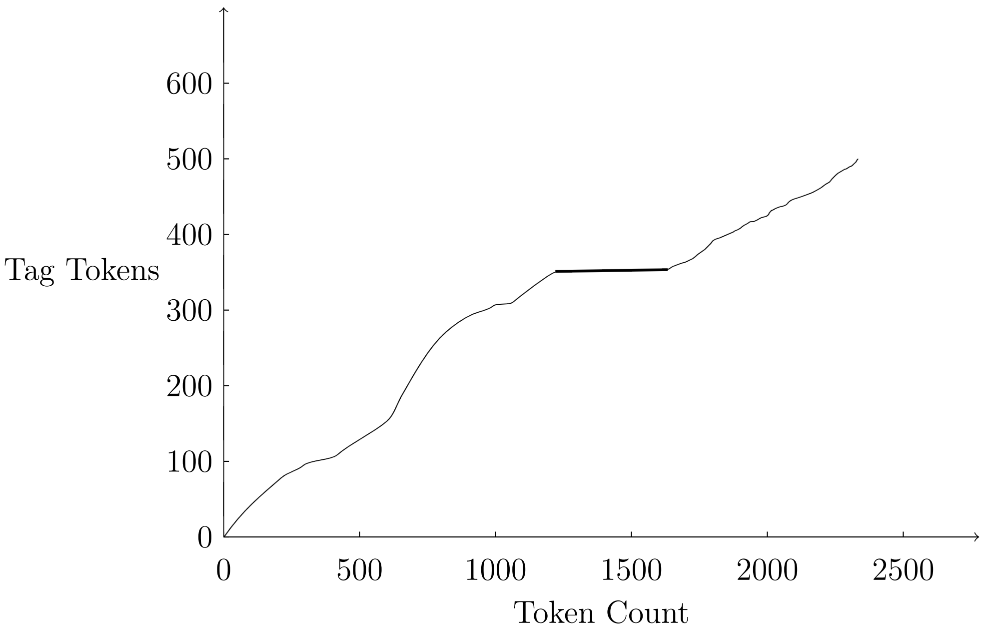


Figure 2.1: An example of the *document slope curve*. The area where the curve plateaus (drawn in bold) contains few or no HTML tags, so it should correspond to the main body of text.

BTE attempts to find a segment on the document slope curve that has a very low slope. Additionally, this segment should not not be too short; that is, it should correspond to a sufficiently long block of text. In other words, BTE tries find two indices i and j such that the number of tag tokens before i and after j is maximized, while the number of word tokens between i and j is also maximized. Formally, we search for two values i and j that maximize following function:

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

where N is the total number of tokens in the document.

The main drawback of BTE is that it makes the implicit assumption that the main body of text is connected; that is, there are no blocks of noisy content inside of it. Pinto et al. [2002] improved this method, so that it searches for multiple plateaus on the document slope curve, rather than just one.

2.2 DOM-Based Content Extraction

The Document Object Model (DOM) is a is a language-neutral programming interface to HTML documents [Stenback et al., 2003]. Thus, it provides a layer of abstraction over the raw HTML source of the webpage. DOM represents HTML documents using a tree strcuture, as shown in Figure 2.2.

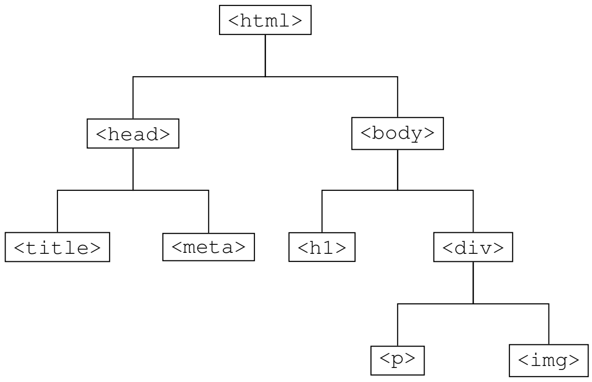


Figure 2.2: The DOM tree of a simple HTML document.

In contrast to BTE and other methods that deal directly with the HTML source of a webpage, Gupta et al. [2003] suggested an approach that relied on the DOM tree of the webpage. The DOM tree of a webpage gives better insight into the structure of webpages than their raw HTML source.

The algorithm begins by first transforming an HTML document into its DOM tree representation. Next, the DOM tree is traversed and two sets of filters are applied. The first set consists of simple filters that remove certain elements such as images, links, scripts, and styles.

The algorithm begins by first transforming an HTML document into its DOM tree representation. Next, the DOM tree is traversed and two sets of filters are applied. The first set consists of simple filters that remove elements such as images, links, scripts, and styles.

The second set consists of more complicated filters that remove advertisements, *link lists*, and tables that do not contain any "substantive information." These filters are based on various heuristics. For example, the values of `href` and `src` attributes are compared with a list of common advertisement servers. If an address is matched, the node that contained the link is removed from the DOM page.

After all filters have been applied to the DOM tree, the DOM tree can then be output in either HTML or plain text format. The plain text output removes all tags and retains only the text (which was identified as main content) of the webpage.

2.3 Vision-Based Content Extraction

Cai et al. [2003] introduced the Vision-Based Page Segmentation Algorithm (VIPS). It attempts to simulate a human user's approach for understanding the *content structure* of a webpage. A human user does not see the HTML markup or the DOM of webpage; rather, all she sees is the visual rendering of the page. VIPS therefore attempts to utilize the same spatial and visual cues that give hints to a human user about the content structure of the webpage.

VIPS is applied recursively to the DOM tree of the webpage. The first step in VIPS is *block extraction*. Starting from the root node down (initially the root node is the `<html>` element), each DOM node is inspected to check whether it represents a single visual block. If so, the block is added into a *block pool*. If the node contains multiple visual blocks, the children of that node are inspected in the same way until all blocks in the current (sub-)page are extracted and added to the block pool.

Whether a DOM node represents a single visual block or should be further divided depends on multiple considerations[1]. For example, if the background color of a DOM node is different from one of its children's background color, then this node should be divided. Another consideration is size: If the relative size of a DOM node compared to the current subpage is smaller than a specific threshold, then this node should *not* be divided.

For each block in the block pool, a *degree of coherence* DoC is a assigned. DoC corresponds to the level of "content consistency" within the block. Depending on the specific application of VIPS, a *permitted degree of coherence* PDoC is pre-defined in order to achieve a certain granularity of the content structure. Figure 2.3 displays the *layout structure* of a webpage with relatively low granularity. To achieve higher granularity, blocks `VB1_1` and `VB1_2` would have to be further divided.

The next step is *separator detection*, in which separators between blocks are detected and their *weights* are set depending on their visibility. The content structure (block hierarchy) for the current round is constructed based on these separators. For instance, in Figure 2.3 the blocks `VB3_1` and `VB3_2` (separated by white space) are children of the block `VB3`.

Next, each leaf node (block) in the current content structure is checked whether it satisfy the granularity requirement, which is DoC > PDoC. Every node that does not satisfy the granularity requirement is considered a subpage, and VIPS is applied to it recursively until we obtain a tree in which all leaf nodes satisfy the granularity requirement.

Remark. VIPS attempts to obtain the *content structure* of the webpage,

[1] The original paper lists 13 rules.

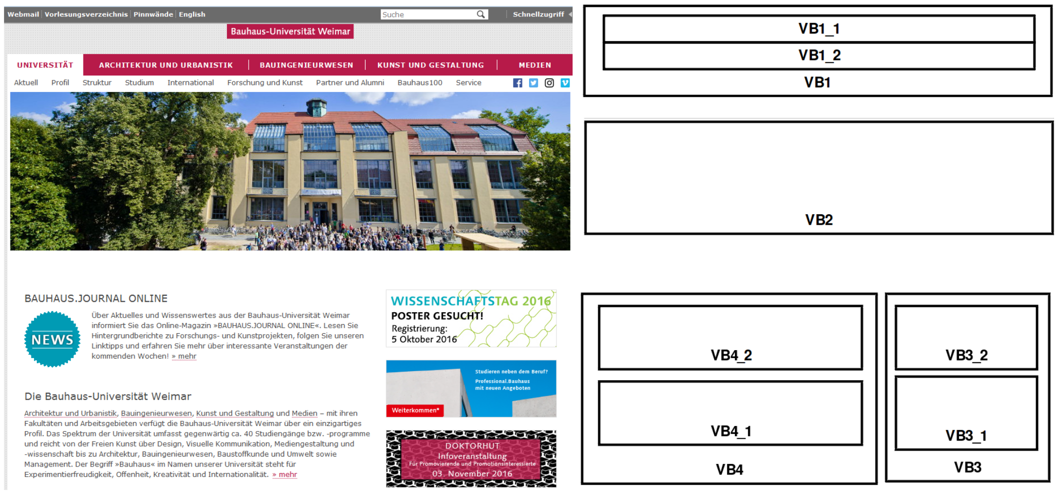


Figure 2.3: The *layout structure* of a webpage. Blocks VB1_1 and VB1_2 could be further divided into child blocks.

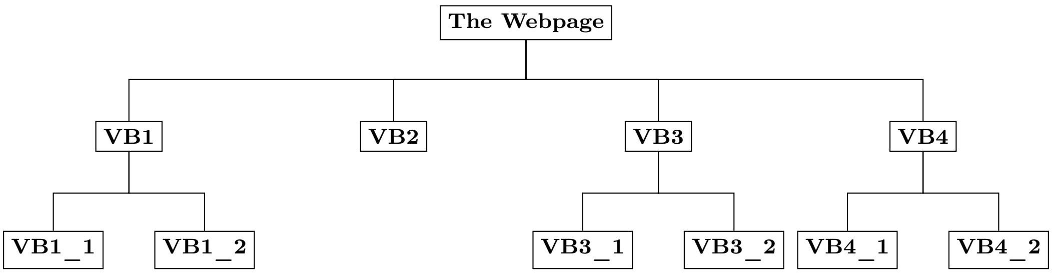


Figure 2.4: The vision-based *content structure* corresponding to the layout structure in Figure 2.3.

which is a hierarchical representation of webpage's semantic content. However, VIPS does not attempt to identify the main content; that is, strictly speaking, *VIPS does not perform content extraction*.

Liu et al. [2006] presented a technique that relied on VIPS in order to perform content extraction on *response pages*, which are defined by Liu et al. [2006] as webpages that contain data records that are retrieved from Web information sources.

First, VIPS is applied to the webpage and the content structure tree is extracted. The next step is identifying the block in the content structure that corresponds to the *data region*, the region where the data records are presented. In order to do so, two characteristics of the data regions are noted:

- Data regions are always centered horizontally.
- The size of the data region is usually large relative to the size of the

entire webpage.

Accordingly, all blocks in the content structure are checked if they satisfy these two conditions. The second characteristic is formalized as

$$\frac{(area)_{block}}{(area)_{webpage}} \geq T_0,$$

where the threshold T_0 is learned from a sample of response webpages. If multiple blocks satisfy the two conditions, the one at the lowest level in the content structure tree is chosen; that is, it is assumed the response page has a single data region.

After the data region has been discovered, the individual data records should be extracted. The process of extracting the data records relies on the following presumed characteristics of data record blocks:

- The data records are usually aligned flush left in the data region.
- All data records are adjoining.
- Adjoining data records do not overlap.
- Data records are very similar in their appearance.
- Data contents of the same type in different data records have similar presentations.
- All data records have mandatory contents and some may have optional contents.
- The presentation of contents in a data record follows a fixed order.

The process begins by filtering out visual blocks in the data region that are not part of any data record. Next boundaries between individual data records, which correspond to groups of visual blocks, are discovered based on the above listed characteristics.

2.4 Wrappers

A **wrapper** is a procedure (program) for extracting database records from a certain information source, in particular from a webpage. Many webpages include dynamically-generated contents that are obtained from a query to an internal database, for example webpages that describe product specifications. Wrappers attempt to restore this information to its relational form. There are three ways to construct wrappers [Liu, 2007]:

Manual coding Wrappers can be created by someone who is familiar with markup of the webpages that contain the data. For instance, a wrapper can be instructed to retrieve the content of certain table cells that contain the relevant data.

Wrapper induction Supervised machine learning is used to obtain the extraction rules. This requires a training set of webages with the manually-labeled relevant data in each webpage.

Automated data extracton Unsupervised machine learning is used instead of supervised learning to obtain the extraction rules. This obviates the need to manually label data in the webpages.

It should be noted that a specific wrapper is designed for a specific information source [Kushmerick et al., 1997].

The tasks of wrappers and content extraction overlap, but they are not identical [Gottron, 2009]. The difference lies in the data to be extracted. Wrappers search for structured or semi-structured data in a webpage, which is usually extracted and subsequently used as input to a relational database. In contrast, content extraction involves the identifying all the main content in a webpage, which usually consists of unstructured data.

2.5 Template Recognition

A **template** can be defined as a webpage layout with slots where *variable contents* can be inserted. For instance, product description pages on a certain e-commerce website usually have the same visual layout. Therefore, a template is designed for these webpages, with placeholders for contents that should be specific to each webpage, such as product name, images, and specifications. Other contents are repeated for many (or all) webpages that are based on the same template, such as banners and navigation menus. These are known as the *template-generated contents* [Gottron, 2009]; they are also referred to as *boilerplate*.

In **template recognition**, we attempt to extract the template structure of a set of webpages that are based on that template. This in turn facilitates identifying main content of the webpage, which usually corresponds to the components that occupy the variable content slots; that is, the webpage-specfic contents.

Lin and Ho [2002] introduced a system called InfoDiscoverer. The system attempts to separate the webpage-specific contents (which they refer to as the *informative contents*) from the template-generated contents (which they refer to as the *semantically redundant contents*).

A **webpage cluster** is defined by Lin and Ho [2002] as a set of webpages that are based on the same template. In order to recognize the template structure of the webpage cluster, we assume that we have a training set of webpages that belong to the same webpage cluster. Next, the *content blocks* of each webpage are extracted, which results in a content structure tree. Subsequently, the granularity of the content structure tree is refined.

The methodology by which InfoDiscoverer separates the informative content blocks from the redundant content blocks is based on the observation that the redundant content blocks have *features*[2] that are very frequent throughout the webpage cluster; this is implied by the fact that template-generated contents are frequently repeated. In the paper by Lin and Ho [2002], the features correspond to *meaningful keywords*, which are obtained after the the stop words are removed from each content block and Porter stemming [Porter, 1980] is applied to the remaining words.

The next step is calculating the entropy value of each feature in the webpage cluster. In the case of InfoDiscoverer, entropy corresponds to the *weight* distribution of the feature in the webpage cluster, where the weight w_{ij} of feature F_i in document D_j is the frequency (the total number of occurrences) of F_i in D_j. Features that are common throughout the webpage cluster should have high entropy.

In order to calculate the entropy of each feature, the features from all documents (webpages) in the webpage cluster are grouped in the *feature-document matrix* (F-D Matrix). A simple example of an F-D Matrix is demonstrated in Table 2.1.

Feature \ Document	D_1	D_2	D_3	D_4	D_5
F_1	14	9	8	5	12
F_2	0	18	2	4	6

Table 2.1: A simple feature-document matrix. The cell (i, j) displays the frequency of the feature F_i in the document D_j. In this example, F_1 has a relatively high entropy, whereas F_2 has a relatively low entropy.

The entropy of *each* feature will be calculated using Shannon's general formula [Shannon, 2001]:

$$H = -\sum_{j=1}^{n} \mathcal{P}(E_j) \log_2 \mathcal{P}(E_j), \tag{2.1}$$

[2]Not to be confused with features as defined in Section 3.4.

where $\mathcal{P}(E_j)$ is the probability of the event event E_j. In the case of InfoDiscoverer, $\mathcal{P}(E_j)$ is proportional to the weight of the feature under consideration in document D_j. Before the weights can be plugged into the Shannon's formula, they are normalized, so their values fall into inside interval $[1, 0]$:

$$H(F_i) = -\sum_{j=1}^{n} w_{ij} \log_2 w_{ij}. \tag{2.2}$$

In order to normalize the entropy values to the interval $[1, 0]$, we modify Equation 2.2 to:

$$H(F_i) = -\sum_{j=1}^{n} w_{ij} \log_d w_{ij}, \tag{2.3}$$

where d is the number of documents in the training set. Features with high entropy are frequently repeated in the webpage cluster, and therefore should be typical of template-generated blocks.

After calculating the entropy of all the features in the training set, we can calculate the entropy of each content block. The entropy of a content block CB_i is defined by

$$H(CB_i) = \frac{1}{k} \sum_{j=1}^{k} H(F_j), \tag{2.4}$$

where k is the number of features in CB_i, and F_j is a feature of CB_i.

Based on the original observation behind InfoDiscoverer, that redundant content blocks have more high-frequency features compared to informative content blocks, redundant content blocks should have a higher entropy than informative blocks. Thus, each block is classified as redundant if its entropy is higher than a certain threshold H_0. The value of H_0 varies depending on the webpage cluster.

To find an optimal H_0 for a specific training set, Lin and Ho [2002] note that by increasing the value of H_0, the number of features that fall into the informative blocks will increase; this is because more blocks will be classified as informative. If the increase in H_0 does not add new features to the informative blocks, the boundary between the informative and the redundant blocks is assumed to have been reached. Accordingly, the following approach is suggested:

> *Starting from $H_0 = 0$, increment H_0 by 0.1 until the incrementation does not add any new features to the informative content blocks.*

During the experiments that were carried out by Lin and Ho [2002], the optimal value of H_0 ranged from 0.1 to 0.7.

2.6 Summary

This chapter presented a survey of the diverse approaches to content extraction. Section 2.1 introduced the Body Text Extraction algorithm, which operates directly on the HTML source of a webpage. Section 2.2 introduced an approach that operates on the DOM tree representation of a webpage. Section 2.3 introduced the Vision-Based Page Segmentation Algorithm, which operates on the visual rendering of the webpage. Section 2.4 gave an overview of wrappers. Finally. Section 2.5 gave an overview the task of template recognition and outlined the workflow of InfoDiscoverer, a system that performs template recognition.

Chapter 3

Methodology and Setup

This chapter begins by trying to define the concept of main content more accurately than stated in Chapter 1. Next, we formulate the task of content extraction as a classification problem, which is a common problem that is treated by machine learning. Sections 3.5, 3.6, and 3.7 describe the process of creating the training set and the test set to be used in the learning and the evaluation processes. Section 3.8 provides a description of the features that will be used in the learning process.

3.1 Defining the Main Content

In Chapter 1, the main content was introduced as "the part of a webpage that makes it a useful source of information," but this definition is rather vague. In this section, we attempt to formulate a more accurate definition. However, as we shall see, the concept of main content is highly subjective, and a precise formal definition cannot be easily given.

Throughout the course of his treatise on content extraction, Gottron [2009] implicitly gives three definitions of main content:

(3.1.1) The main content is what the webpage is supposed to communicate (according to its publisher)[1].

(3.1.2) The main content is what makes the webpage interesting to the user.

(3.1.3) The main content consists of the contents of a webpage that are unique to that webpage; that is, they cannot be found in other webpages.

Definitions (3.1.1) and (3.1.2) try to capture the point of view of the webpage publisher and that of the webpage user, respectively. They were motivated by

[1] Gottron [2009] did not explicitly specify according to whom.

webpages that feature a news story or an encyclopedia article. For example, a webpage about a certain news story *should communicate* information that are relevant to that story. Conversely, it is the information that are relevant to the news story that give Web users, in general, interest in that webpage. However, both of these definitions have complications.

Definition (3.1.1) has the problem of identifying the publisher. Although most websites (and consequently webpages) are owned by a single party, many webpages include contents that have been posted (published) by multiple parties, such as advertisements and comments. Therefore, according to their publishers, advertisements include information that the webpage is supposed to communicate. In fact, even if the webpage has a single publisher, it is not always clear what the publisher wants to communicate. For instance, some publishers *would like* the user to read the links to similar webpage on the website, so that the user might visit these webpages.

Definition (3.1.2) has the problem that different users might have different interests in the webpage. For instance, many users prefer to read only the article synopsis, skipping the article body, and many users are interested in the links to related articles that the webpage provides, which most content extraction algorithms classify as non-main.

Definition (3.1.3) is the most objective of the three definitions because it is based on concrete concrete facts, assuming we can identify identical (duplicated) contents. For simplicity, we will assume that contents are identical if and only if they are equal in their raw form. For instance, two paragraph are considered identical if they are literally equal, character for character. If two paragraph contain the same semantic information, but are formulated differently, they will not be considered identical.

Definition (3.1.3) has the problem that most information on the Web is duplicated; that is, included in multiple webpages[2]. Therefore, most webpages will have no main content at all according to Definition (3.1.3).

Definition (3.1.3) can be made more practical by restricting the comparable webpages to the same website or webpage cluster; in fact, this definition of main content roughly corresponds to the *informative content* term used during the discussion of the InfoDiscoverer system for template recognition (see Section 2.5). However, Definition (3.1.3) will still be inaccurate in some cases. For example, when two different webpages in the same website feature the same topic, they will have duplicated contents, such as images.

[2]The Wayback Machine, accessible under http://archive.com, stores archived versions of approximately 445 billion webpages as of Nov, 2015. [Forbes Magazine, 2015]

The Definition of Main Content Used in This Work

In this work, the main content of a webpage will be defined as consisting of all contents of the webpage that are not noisy. The rationale behind this definition is that the noisy content is easier to define than the main content, as will be clarified in Section 3.3.

3.2 Types of Webpages

Webpages can be divided into the following broad categories depending upon their purpose:

Directory webpages Contain links to other webpages, and include no *elaborate* information. The user visits these webpages in order to obtain the links to other webpages that include detailed information about a certain topic. The homepages of most websites fall into this category. Other examples of directory webpages include search result webpages and the main pages of specific website sections, such as News, Weather, Sports, and so on. It should be noted that directory webpages may include *non-detailed* information, such as synopses of news stories in the linked-to webpages.

Form webpages The main purpose of these webpages is to receive information from the user, rather than provide information to the user. Examples of form webpages include registration pages, settings pages, and email composition pages.

Article webpages Contain detailed information about a certain subject. The user visits these webpages primarily in order to access this information. This definition of article webpages encompasses not only webpages that include an article in the common sense, such as news article or an encyclopedia article, but also webpages that include detailed information of any kind, such as product specifications, statistical figures, forum discussions, and so forth.

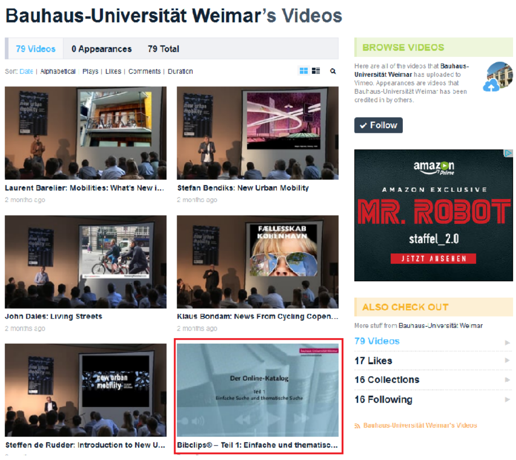
The line between these types of webpages can be blurry in some cases. For instance, article webpages may accept input from the user, such as commenting on a news story or posting an opinion in a forum discussion.

3.2.1 The Main Content in Different Types of Webpages

It should be pointed out that the notion of main content varies depending on the type of webpage that us being dealt with, in particular when considering the user's perspective. For instance, Figure 3.1a displays the webpage of a video from Bauhaus-Universität Weimar's channel on Vimeo. The webpage includes a section called "More from Bauhaus-Universität Weimar" that features other videos from the same channel, including, for example, a link to a video called *Bibclips®—Teil 1: Einfache und thematische Suche* (highlighted with a rectangle). This link would be considered noisy content because the user presumably visits this webpage in order to watch the video that the webpage itself features.

In comparison, Figure 3.1b displays a webpage that provides a list the videos posted by Bauhaus-Universität Weimar. In this case, obtaining the links to Bauhaus-Universität Weimar's videos is the reason for which the user visits this webpage. Therefore, the link to the same video (also highlighted with a rectangle) would be considered main content in this webpage.

(a) The webpage of a video titled *Klaus Bondam: News From Cycling Copenhagen – and Cycling Denmark* from Bauhaus-Universität Weimar's channel on Vimeo.



(b) The *Videos* webpage of Bauhaus-Universität Weimar's channel on Vimeo.

Figure 3.1: In (a), the highlighted video link is considered noisy content, while the same link is considered main content in (b).

Remark. In this thesis, content extraction will be restricted to article webpages.

3.3 The Non-Main Content

As mentioned earlier, the noisy content of a webpage consists of all of its contents that are not main. However, the noisy contents can be further subcategorized into distinct types. In the following list we attempt to provide an exhaustive categorization of all possible contents of a webpage that will not be considered main.

Advertisement This is the most obvious type of noisy content. Many webpages include paid advertisements of commercial products, which are sometimes related to the topic of the webpage (targeted marketing).

Navigation Most websites include a navigation menu (or bar). It consists of links to certain (usually important or frequently accessed) webpages on the website, such as the home page and the FAQs page.

Promoted webpages These include links to webpages other than the current webpage. The links may refer to

- webpages about the same topic as the current webpage or about a similar topic
- webpages that are currently trending; that is, frequently read, shared, or commented on.

The referred-to webpages can reside on the same website as the original webpage or on a different website

Legal information This category includes contents such as copyright notices and privacy notices.

Irrelevant information Some webpages include extra information, such as weather forecast or stock market indices, which may or may not be related to the topic of the webpage.

Sources and references Some webpages provide a list of sources of the information they contain or references for further reading.

Input elements These are the elements that receive input from the user of any kind, such as text boxes and check boxes. This category also includes elements that allow the user to perform any action, such as Like, Share, Print, and Send buttons. Although these elements may be *important* to the user, it was decided to treat them as non-main content because content extraction deals with webpages as information sources and does not deal with their interactive aspect.

This list is useful (in terms of content extraction) because each category can be easily and unambiguously identified by a human observer. For example, it is trivial to decide whether a certain content belongs to the advertisement category or not. In other words, there will be no disagreement between human observers about these categories.

Consequently, it was decided that in this work that every content that does not belong to one of the categories in the above list will be considered main content. This definition served as a guideline for annotating webpages during the preparation training set (see Section 3.7.3).

3.4 Using Machine Learning for Content Extraction

3.4.1 Content Extraction as a Classification Problem

The problem of content extraction can be regarded as a **classification problem**. In a classification problem, we attempt to assign a new **instance** (sometimes called an observation) to exactly one class (sometimes called a category). The set of possible classes is pre-defined and finite. The instances to be classified should have the same *type*, such as a person, a vehicle, a rasterized image, text document, and so on.

Instances of a certain type have **features**, which are individual measurable properties of the phenomenon that each instance abstracts [Bishop, 2006]. From a classification algorithm's perspective, an instance is fully described by the combination of its feature values. For example, in a certain classification problem, a car could be represented by its engine displacement, maximum speed, and brand name. These feature values are utilized by the classification algorithm when trying to classify an instance.

In the case of content extraction, an instance would be a webpage content, such as a single HTML element or a group of HTML elements, the pre-defined set of possible classes would be "main" and "noisy," and the features would be properties such as the length of the inner text, the number of certain words in the inner text, the visual position of the content inside the webpage, and so on.

3.4.2 Building Classifiers Using Machine Learning

Classification is a common problem that is treated by machine learning. In this work, supervised machine learning will be used. A **training set** that consists of instances *with known classes* is used by a supervised machine learning algorithm to induce rules for predicting the classes of future instances (whose classes are not known). These induced rules are then used to construct a **classifier**, which is itself a classification algorithm. In other words, the output of applying a machine learning algorithm to a training set is a classification algorithm.

After a classifier has been created, its performance is usually evaluated using a **test set**. A test set consists of instances with known classes (like a training set), but these instances were not used for training the classifier.

When training and evaluating a classifier that performs content extraction, the training set and the test set should consist of webpages whose contents have been manually classified (by a human user) as either "main" or "noisy."

3.5 Types of HTML Elements

As stated in Section 3.4, content extraction is a classification problem, where we classify arbitrary contents as either main or non-main. However, in most content extraction algorithms, the content to be classified is a set of one or more visually contiguous HTML elements, usually referred to as a **block**.

Before discussing further details about this thesis' training set, a clarification should be made about the types of HTML elements. For purpose of content extraction, we will make the following categorization, which encompasses most HTML elements:[3]

Sectioning elements Contain other HTML elements (child elements), rather than data directly. Their goal is to organize the child elements in a certain way or to indicate that they are semantically related. Examples of sectioning elements include `<table>`, `<ol>`, and `<div>`.

Content elements Contain data directly (as a child node) and define its *structural type*, for instance whether the marked-up data is represents a paragraph of a list item. They may additionally include other elements. Examples of content elements include `<li>`, `<p>`, and `<img>`.

Inline semantic elements Define a semantic meaning for an arbitrary piece of text [Mozilla Developer Network, 2016]. Inline semantic elements are normally children of textual content elements. Examples of inline semantic elements include `<span>`, `<cite>`, and `<q>`.

It should be noted that some HTML elements could belong to more than one category, depending upon their role. For instance, the element `<div>` can be used as both a sectioning element and a content element.

The classifier that we attempt be construct over the course of this thesis will be used to classify only content elements as either belonging to the main

[3]Exceptions include scripting elements and webpage metadata elements. However, this categorization will be sufficient for the purpose of this thesis.

content or the noisy content. Other types of elements will be classified *indirectly*. For instance, if we wish to classify a list as main or noisy, we first classify each of its items. If all of the items are main, so is the list. It is possible that some, but not all, items are main, in which case the list is partially main—in fact, this is an advantage of our approach, which enables us to perform fine-grained content extraction in cases when sectioning elements contain both main and noisy contents. On the other hand, if we have a <q> element (an inline semantic element) that is the child of a <p> element (a content element), we first classify the <p> element, and the same classification applies to the <q> element.

It should be noted that the data inside a single content element will be considered atomic in terms of classification, for example a <p> element is either entirely main or entirely non-main.

3.6 The Dataset Format

This section discusses the formats of two previously created datasets of webpages with manually labeled contents that are publicly accessible (the CleanEval dataset and the L3S-GN1 dataset) before it discusses the format of the dataset that will be used in this thesis.

The CleanEval Dataset

As part off CleanEval competition for cleaning webpages in 2007, a gold standard of annotated webpages was created [Baroni et al., 2008]. The dataset contains 681 webpages and is restricted to textual contents; that is, only textual contents is classified as either main or noisy. The output of annotating an HTML document by a human user is a text document with all noisy contents removed and simple markup added. The markup indicates the original tag of the text.

For example, for the following HTML segment:

```
<a href="http://www.environment-agency.wales.gov.uk/">
<img src="/common/images/toolbar/banner_index_home_off.gif"
    width=47
height=24 hspace=0 vspace=0 border=0 alt="Home" align="left"
name="banner_index_home"></a>
<a href="http://www.environment-agency.gov.uk/news/?lang=_e">
<img src="/common/images/toolbar/banner_index_news_off.gif"
    width=40
height=24 hspace=0 vspace=0 border=0 alt="News" align="left"
name="banner_index_news"></a>
```

```
<h3><font face="Arial"><a name="eutro"></a>Eutrophication</font
    ></h3>
<p><font face="Arial" size="2">Concentrations in Welsh rivers
    of the main
plant nutrients (phosphate and nitrate) are generally much
    lower than those
found in the midlands and south-east England.</font></p>
```

the result of the annotation was:

```
<h>Eutrophication

<p>Concentrations in Welsh rivers of the main plant nutrients (
    phosphate
and nitrate) are generally much lower than those found in the
    midlands
and south-east England.
```

The L3S-GN1 Dataset

The L3S-GN1 dataset consists of 621 webpages and was created during the process of preparing Kohlschütter et al. [2010]. Like the CleanEval dataset, the L3S-GN1 dataset is restricted to textual contents. The result of labeling an HTML document is the same HTML document, with the textual main contents being enclosed by `<span>` elements. The class of each `<span>` element can be `x-nc-sel1` through `x-nc-sel5`, which encode the text as headline, full text, supplemental, related content[4], and user comments, respectively. Unselected text (not enclosed by the described `<span>` elements) is regarded as noisy content.

All the webpages in the L3S-GN1 are in the English language, and an examination of the webpages revealed that they all belong to the category of article webpages.

For example, for the following HTML segment:

```
<p>
Along the way, Fearnley-Whittingstall cooks some really nice
    food to prove that free-range chicks are best (though a chef
    -prepared risotto would surely taste good regardless of
    where the chicken came from); persuades a local tool company
    &#8217;s canteen to &#8216;do a Jamie Oliver&#8217;, that is
```

[4] These include links to other webpages. In the L3S-GN1 dataset, they are considered main content.

```
   , dump catering cuisine and cook &#8216;real&#8217; food
   instead; and finally, as is common to most TV production
   today, he makes some Axminster locals cry about their
   lifestyle choices (with weeping children for extra moral
   pressure!) when they visit his factory-farmed bird shed.
</p>
<p>
Happily, one of the Axminster locals, a generously proportioned
    single mum called Hayley, rather impressively refuses to
   cry or get upset on cue for the cameras. The reality of
   chicken farming is exactly what she imagined it might be
   like, she says. She&#8217;d probably prefer to eat the free-
   range stuff, but she&#8217;s just fine with intensive
   farming as it means she can afford to eat chicken and feed
   her family. She clearly hadn&#8217;t read the script.
</p>
```

the result of the annotation was:

```
<p>
<span class="x-nc-sel2">Along the way, Fearnley-Whittingstall
   cooks some really nice food to prove that free-range chicks
   are best (though a chef-prepared risotto would surely taste
   good regardless of where the chicken came from); persuades a
    local tool company?s canteen to ?do a Jamie Oliver?, that
   is, dump catering cuisine and cook ?real? food instead; and
   finally, as is common to most TV production today, he makes
   some Axminster locals cry about their lifestyle choices (
   with weeping children for extra moral pressure!) when they
   visit his factory-farmed bird shed.</span>
</p>
<p>
<span class="x-nc-sel2">Happily, one of the Axminster locals, a
    generously proportioned single mum called Hayley, rather
   impressively refuses to cry or get upset on cue for the
   cameras. The reality of chicken farming is exactly what she
   imagined it might be like, she says. She?d probably prefer
   to eat the free-range stuff, but she?s just fine with
   intensive farming as it means she can afford to eat chicken
   and feed her family. She clearly hadn?t read the script.</
   span>
</p>
```

The Format of the Dataset in This Thesis

In the annotation process used for this thesis, the output of annotating an HTML document is the same HTML document with some descendants of the <body> having the class CEML___MAIN__CONTENT[5] (in addition to their original classes); this class designate these elements as main contents. Elements that do not have this class are considered noisy contents.

For example, for the following HTML segment (many tags and attributes were removed in order to improve readability):

```
<h1 class="header">Control </h1>
<table cellpadding="3" cellspacing="0" border="0" width="100%">
<tbody><tr>
<td valign="top" class="content" width="25%"><div class="
    wobjectSQLReport" id="wobjectId98">
<a name="98"></a><table class="toplinks">
<tbody><tr>
<td>
<a href="#54">Brush-B-Gone / Roundup</a><p>
<a href="#55">General Discussion</a></p><p>
<a href="#56">Household mixtures to kill poison ivy plants</a
    ></p><p>
</p></td>
</tr>
</tbody></table></div>
</td>
<td valign="top" class="content" width="75%"><div class="
    wobjectItem"
id="wobjectId52" style="background-color: rgba(255, 0, 0,
    0.0980392);">
<a name="52" style="background-color: rgba(255, 0, 0,
    0.0980392);"></a>
Some suggestions on controlling poison ivy, oak and sumac
    plants.
If you're lucky you may be able to fully remove the plants -
I've only been able to get them under control.
</div>
```

the result of the annotation was:

```
<h1 class="header CEML___MAIN__CONTENT">Control </h1>
<table cellpadding="3" cellspacing="0" border="0" width="100%">
<tbody><tr>
```

[5]CEML stands for "**C**ontent **E**xtraction Using **M**achine **L**earning"

```
<td valign="top" class="content" width="25%"><div class="
    wobjectSQLReport" id="wobjectId98">
<a name="98"></a><table class="toplinks">
<tbody><tr>
<td>
<a href="#54">Brush-B-Gone / Roundup</a><p>
<a href="#55">General Discussion</a></p><p>
<a href="#56">Household mixtures to kill poison ivy plants</a
    ></p><p>
</p></td>
</tr>
</tbody></table></div>
</td>
<td valign="top" class="content" width="75%"><div class="
    wobjectItem CEML___MAIN__CONTENT"
id="wobjectId52" style="background-color: rgba(255, 0, 0,
    0.0980392);">
<a name="52" class=" CEML___MAIN__CONTENT" style="background-
    color: rgba(255, 0, 0, 0.0980392);"></a>
Some suggestions on controlling poison ivy, oak and sumac
    plants.
If you're lucky you may be able to fully remove the plants -
I've only been able to get them under control.
</div>
```

The details of the annotation process will be discussed in Section 3.7.2. It should also be noted that the annotated document is not used directly by the machine learning algorithm, rather it is transformed to CSV format before being used for training, as discussed in Section 3.7.2.

3.7 Creating the Dataset

Before a classifier can be trained, a set of webpages with manually labeled contents (by a human user) should be available. This section describes the process of creating the training set and the test set that were used in this thesis.

3.7.1 The Training and Test Sets Used in This Work

The training set that was used to train the classifier in this work consisted of 30 manually-selected article webpages, each webpage from a different website. The webpages were manually selected, so that diverse genres were represented in the training set. These genres included news articles, encyclopedia articles, product description webpages, forum discussions, and video webpages.

Figure 3.2: The manual annotation of a webpage. After injecting the JavaScript code into the page, the user can draw labeling rectangles on the webpage using the mouse. An element should lie *entirely* inside at least one labeling rectangle to be labled as main. Otherwise, it will be labeled as noisy.

The test set included 30 webpages with URLs that were randomly chosen from the L3S-GN1 dataset. Many webpages were no longer available, in which case they were replaced by webpages from the same website if the website itself was still operating. Additionally, 10 webpages from the same websites that were used in the training set were used in the test set. The webpages in the test set were manually annotated in the same way as the those in the training set.

3.7.2 Annotating the HTML Documents

In order to facilitate manual annotation, a JavaScript program was developed and injected into each webpage that had to be annotated. The JavaScript program allows the user to draw *labeling rectangles* on the webpage in order to identify the main content. HTML elements whose visual rendering lies *entirely* inside at least one of the labeling rectangles are labeled as main. All other HTML elements are labeled noisy, as shown in Figure 3.2.

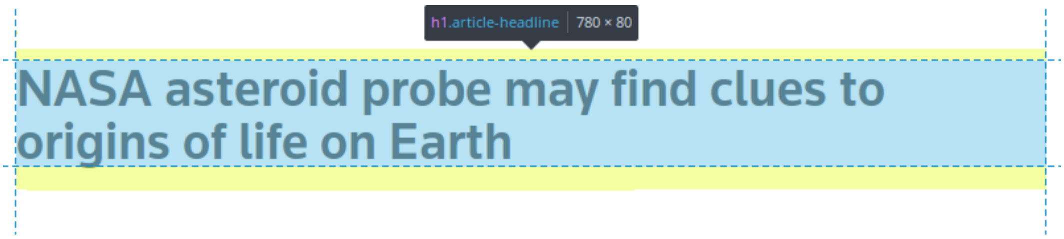


Figure 3.3: An inspection of the header element from a news story from http://reuters.com. In order to label the header as main, the user must draw a labeling rectangle that surrounds the entire bounding rectangle of the <h1> element. However, as can be seen, the bounding rectangle, which invisible to the user during annotation, is significantly wider than the visible text.

A technical complication arises as a result of this labeling approach. The JavaScript program relies on the `Element.getBoundingClientRect()` function to check whether a certain HTML element is located inside a certain labeling rectangle. However the bounding rectangle returned from `getBoundingClientRect()` extends well beyond the “visible" portion of the element. Figure 3.3 gives a demonstration of this issue.

In order to overcome this complication, the JavaScript program allows the user (at any point during the annotation process) to press C, upon which the program identifies all the elements whose bounding rectangle currently lies inside at least one of the labeling rectangles and changes their background color to light red. This allows the user to notice the elements that are not contained in a labeling rectangle (although they should be), so that she can draw a larger rectangle around these elements, as demonstrated in Figure 3.4.

When the user has finished annotating the webpage, she presses D, upon which the JavaScript program identifies all the elements that lie entirely inside at least one labeling rectangle and adds the class `CEML___MAIN__CONTENT` to these elements. Subsequently, feature extraction is performed, during which the features of *every* HTML element[6] that descends from <body> are calculated, including its label (classification), and a CSV file containing feature values of each element is created and saved. The specific features that are calculated will be discussed in Section 3.8.

The result of annotating a single webpage is a single CSV file. The final step in creating the training set is concatenating the individual CSV files that resulted from the annotation of each webpage, as demonstrated in Figure 3.5. The test set was created in the same way, using a different set of webpages.

[6]These include non-content elements, which will be discarded during the learning process.

Figure 3.4: When the user presses C, the JavaScript program identifies all elements whose bounding rectangle lies completely inside at least one labeling rectangle, and changes their background color to light red. In this example, the bounding rectangle of the story header does not lie entirely inside the drawn labeling rectangle, so the user has to draw a larger labeling rectangle in order for the header to be labeled as main content.

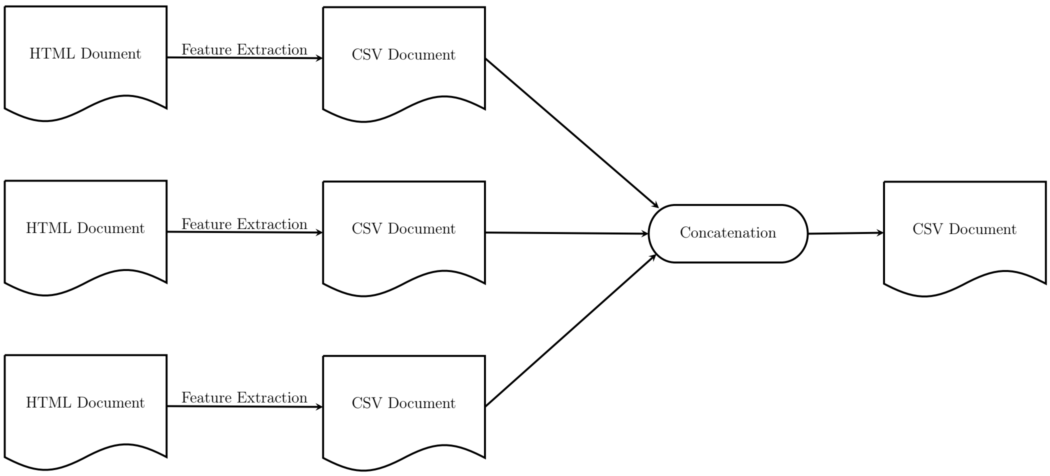


Figure 3.5: The workflow of creating the training set. The CSV file on the extreme right will be the training set. The workflow of creating the test set is identical.

3.7.3 Annotation Guidelines

Although identifying the main content is considered a relatively easy task for human users, disagreements about whether some portions of a webpage belong to the main content can arise between human users.

As mentioned in Section 3.1, there is no plain definition of main content that users can strictly follow when performing annotation. However, in Section 3.3, a list containing a categorization of noisy contents was given. Each category in the list is trivially and unambiguously identifiable by a human user. Therefore, it was decided that when performing annotation, if a content does *not* belong to one category in List 3.3, this content will be considered main.

Another guiding rule is that elements that contain information (data) that is not repeated in other webpages should be labeled as main content. For example, Figure 3.6 demonstrates the annotation of the Youtube comment action bar, located at the bottom of every comment on a Youtube video. The "Reply," "Thumbs Up," and "Thumbs Down" buttons promote the user to take actions, rather than provide her with information, so they are not labeled as main content. In contrast, the number of likes that the comment has received represents a piece of information that may be interesting to the user, and this information cannot be found in other webpages, so the number of likes is labeled as main content.

This example demonstrates the "deep" level at which the annotation process was performed, and the the high degree of granularity of the resulting training set and test set. Constructing a classifier that can accurately perform content extraction at the same degree of granularity represents a significant challenge.

Figure 3.6: The annotation of the Youtube comment action bar. The only piece of information found in the action bar is the number of likes that the comment has received, so it is the only part that is labeled as main content.

In fact, most content extraction methods would classify the entire action bar in Figure 3.6 (as noisy content), rather than its individual sub-blocks. However, the goal of the annotation process was to produce a fine-grained **gold standard**. The performance of the classifier that would be constructed based on that gold standard is a separate issue.

It should be noted that there are some cases in which annotation with such high granularity is not possible. For example, Figure 3.7 displays a header from a Wikipedia article. The title of the header ("Overview") should be labeled as main content, and the clickable text (`[Edit]`) should be labeled as noisy content. However, this is not possible because the clickable text (delimited by a `<span>` element) lies entirely inside the header element `<h2>`. Therefore, it is not possible label the text "Overview" as main content without also labeling the clickable text `[Edit]` as such.

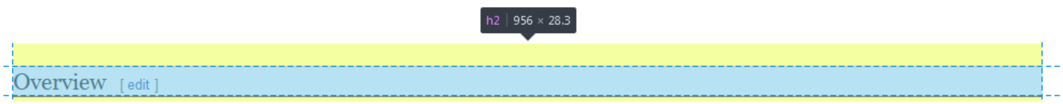


Figure 3.7: An inspection of a header element that contains the title of a section in a Wikipedia article. It is not possible to label the title ("Overview") as main content without also including the clickable text (`[Edit]`).

3.7.4 The Language Dependence of Our Approach

During the course of webpage annotation, it was imperative that the annotator understood the textual contents of the webpage, so that she could decide which contents are main and which are noisy. In fact, some text blocks had to be carefully inspected in order to decide whether they were main or noisy.

In addition, some of the features that our learning process uses are language dependent, as will be discussed in Section 3.8. Therefore, all the webpages in our dataset are in the English language. However, the approach used in this thesis can be easily extended to other languages.

3.8 Feature Engineering

Most content extraction algorithms (including those based on machine learning) classify entire webpage blocks as either main or noisy. A webpage block usually corresponds to a `<div>` element, including its descendants. Blocks vary widely in size, and can in many cases include mixed contents (both main and noisy).

In contrast, as mentioned in Section 3.5, the classifier to be constructed in this thesis will operate on individual content elements, which contain indivisible pieces of data (from a structural point of view).

Determining whether the data that a content element contains is main or noisy depends on several *factors*, many of which are not inherent in the content element. The most obvious of these *external* factors is the location of the element on the webpage. For example, main content is usually displayed in the middle of the webpage; therefore, an element that is displayed in the middle of the webpage is more likely to belong to the main content than an *identical* element that is displayed at one side of the webpage.

Other significant external factors include the description of ascendant HTML elements. For example, a paragraph element (`<p>`) that descends from a `<div>` element that has the class `"cookies"` is probably a statement regarding the website's use of cookies.

The first step in building the content classifier is translating these factors (both internal and external) to features.

3.8.1 Features Used in Other Works

Song et al. [2004] used supervised machine learning to produce a classifier to identify the *the level of importance* of webpage segments (blocks). The used features include:

Spatial features The dimensions and coordinates of the segments.

Content features These included, among others, the number of images inside the segment, the number of links inside the segment, and the length of the inner text of the segment.

Louvan [2009] applied supervised machine learning to segment-based content extraction. The used features include:

stopWordRatio The ratio of stop words that is contained in all of the text nodes of a particular DOM node.

domHeight The maximum depth that can be reached from a particular DOM node to a certain leaf node.

headerAround Whether there are here are any header elements near a particular DOM node.

Zhou and Mashuq [2014] applied unsupervised machine learning (clustering) to text blocks, which they describe as HTML block elements that contain texts. The used features include:

Text length The number of non-whitespace characters.

Tag path The path it takes to navigate the DOM tree from its root to the text block, for example `"body>div>p"`. Each different tag path was uniquely treated and this feature was vectorized.

CSS properties These include color, font-size, font-style, line-height, and so on.

3.8.2 The Raw Features

The injected JavaScript program (described in Section 3.7.2) calculates a set of features for *every* element in the webpage that descends from `<body>`. These features include every factor that we thought could play a role in the classification of an element.

However, many of these raw features are textual and cannot be used directly by machine learning algorithms. Therefore, they are processed later to produce other features that are boolean, nominal, or numerical.

For instance, the inner text of certain types of HTML element (such as `<p>` and `<li>`) was extracted by the JavaScript program. Subsequently, various features could be extracted from the inner text, such text length and the ratio of stop words inside the text. It is these later extracted features that are used by the machine learning algorithm to train the classifier.

The following list describes the raw features that are extracted by the injected JavaScript code for each HTML element. The *page-related features* and the *contextual features* represent the "external factors" that affect the element's role in the webpage. The *inherent features* represent inherent properties of the HTML element that are not related to other elements in the webpage.

Remark. For boolean features, `"true"` values were encoded by `"1"` and `"false"` values were encoded by `"0"`. This remark holds for both the raw features and the derived features, discussed in Section 3.8.4.

Page-Related Features

These features will be identical for elements of the same webpage.

url The URL of the webpage

title The title of the webpage.

meta_description The `content` of the description `<meta>` element of the webpage (if present). This features, along with `title`, give give hints about the topic of the webpage.

doc_dom_depth The maximum depth of the DOM tree of the webpage.

Contextual Features

These features are related to the "context" that surrounds the HTML element in the webpage, both from a visual point of view or a structural point of view. They also include properties of "surrounding" elements, which give hints about the role of the HTML element under consideration.

ancestors_names A comma-separated list of the tag names of the ancestor elements until, but not including, `<body>`. A simple example: `DIV, DIV, OL`.

ancestors_ids A comma-separated list of the id's of the ancestor elements until, but not including, `<body>`. When an element does not have an id, the value `CEML__NO__ID` is used.

ancestors_classes A comma-separated list of the classes of the ancestor elements until, but not including, `<body>`. The classes of a single element are separated by whitespace. When an element does not have classes, the values `CEML__NO__CLASSES` is used.

siblings_names A comma-separated list of the tag names of the siblings of the element.

siblings_ids A comma-separated list of the id's of the sibling elements.

siblings_classes A comma-separated list of the classes of the sibling elements.

nearest_header The inner text of the nearest header element *in the DOM tree* (not necessarily visually) that has a lower DOM depth than the current element, if such a header element exists. This is supposedly the

Figure 3.8: Part of the language setting section in the homepage of `http://bbc.com`. Each of the displayed options (`<li>` elements) has its `nearest_header` feature value equal to `"More Languages"`.

header that describes the HTML element under consideration. The value of this feature can give a hint about the role of the element. For instance, if the value includes the string `"languages"`, the element is probably located in the language setting section, as shown in Figure 3.8.

normalized_top The normalized vertical coordinate of the upper edge of the bounding rectangle relative to the webpage's upper-left corner.

normalized_bottom The normalized vertical coordinate of the bottom edge of the bounding rectangle relative to the webpage's upper-left corner.

is_middle `"1"` if the element client bounding of the element intersects with the bisector of the webpage, otherwise `"0"`.

is_leftmost `"1"` if the leftmost edge of the bounding rectangle of the element touches the leftmost edge of the webpage *and* `is_middle="0"`, otherwise `"0"`.

is_rightmost `"1"` if the rightmost edge of the bounding rectangle of the element touches the rightmost edge of the webpage *and* `is_middle="0"`, otherwise `"0"`. Figure 3.9 demonstrates the horizontal positioning features.

num_siblings The number of sibling elements of the element under consideration.

distance_to_root The depth of the node that corresponds to the element under consideration in the webpage DOM.

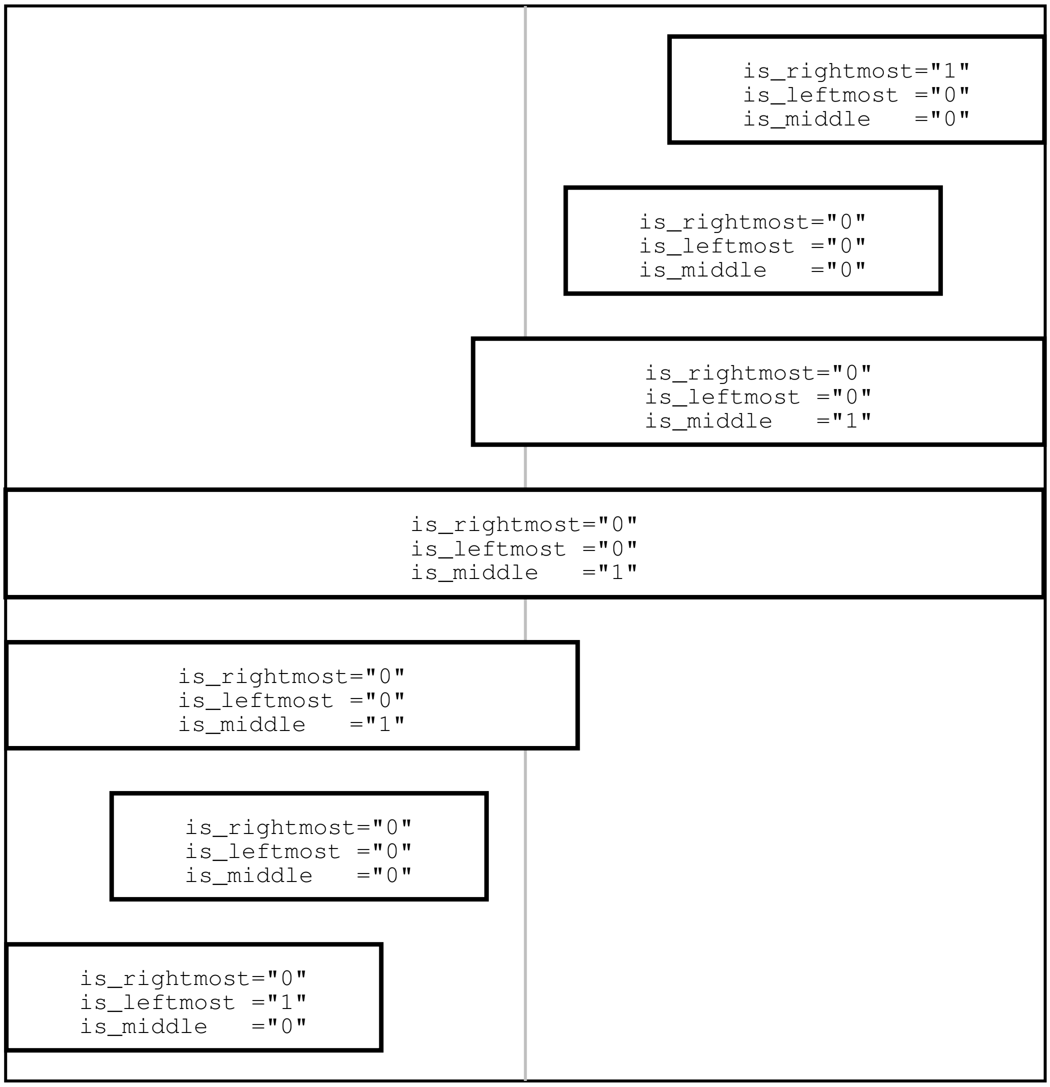


Figure 3.9: A demonstration of the horizontal positioning features.

Inherent Features

These features represent properties of the HTML element, and are not related to its surroundings.

tag_name The tag name of the element.

element_id The id of the element (if any).

class_name The classes of the element (if any).

children_names The tag names of the direct children.

children_ids A comma-separated list of the id's of the direct child elements.

children_classes A comma-separated list of the classes of the direct child elements.

image_alt The alternate text of an image, if present (applicable only to `<img>` elements).

rect_size The size of the bounding client rectangle of the element.

num_child_elements The number of the direct child elements of the element under consideration.

dom_subtree_depth The maximum depth of the DOM subtree whose root is the element under consideration.

inner_text The inner text of the element. This feature is applicable to textual content elements.

child_text The text that is directly contained as a child node of the element. This feature is applicable to textual content elements.

3.8.3 Remarks About the Raw Features

- Section 4.3.1 clarifies which elements will be considered textual content elements.
- When features are not applicable, special values were used. For instance, for elements other than `<img>`, the value of the `image_url` feature is `CEML_NON_IMG_TAG`.
- When a features value is not present, a special value is used. For instance, if an `<img>` element does not have an `alt` attribute, the value `CEML_NO_ALT` is used.

- Many of these raw features were not utilized in training the content classifier in these thesis.

3.8.4 Derived Features

As mentioned earlier, raw text cannot be used directly by machine learning algorithms. Therefore, new features that could be used by machine learning algorithms were derived from the raw textual features. Additionally, more features were derived from other raw non-textual features.

During the course of this thesis, we tried using numerous different features; some of these features were useful, while others were not. The following list contains features that we found useful:

is_desc_a `"1"` if the element descends from an `<a>` element, otherwise `"0"`.

is_desc_X `"1"` if one of the element's ancestors has the class or id `X` (ignoring case), otherwise `"0"`. The values of `X` that were used include`"navigation"`, `"advertisement"`, `"comment"`, `"main"`, `"footer"`, `"wrapper"`, and `"aside"`.

inner_text_length The word count of `inner_text`.

child_text_length The word count of `child_text`.

contains_X `"1"` if the inner text contains the string `X` (ignoring case), otherwise `"0"`. The values of `X` that were used include `"rights reserved"`, `"like"`, and `"share"`.

is_sib_X `"1"` if the element a sibling `X` element, otherwise `"0"`. In particular, `is_sib__p` was very useful when classifying `<p>` elements. Other useful variations include `is_sib__a` and `is_sib__input`.

has_children `"1"` if the element has child elements, otherwise `"0"`.

is_uppermost_or_bottommost `"1"` if `normalized_bottom>0.97` or `normalized_top<0.03`, otherwise `"0"`. In other words, this feature specifies whether the element is very close to either the bottom or the top of the webpage.

is_on_side `"1"` if either `is_leftmost="1"` or `is_rightmost="1"`, otherwise `"0"`.

is_link `"1"` if the element has no child nodes other than a single `<a>` element, otherwise `"0"`.

is_thumbnail "1" if the element under consideration is the only child node of an <a> element, otherwise "0". This feature was used only when classifying <img> elements, although it is theoretically applicable to any type of element.

Remark about Class Name and ID Variations

Different webpages use different class names and IDs to denote navigation and advertisement sections. Common class names for navigation sections include "navbar", "nav-bar", "nav-main", "navigation-menu", and so on. Thus, the solution we used was to search for the string "nav" in the ancestor_classes and ancestor_id and set is_desc_nav="1" if the string was found.

The situation with "advertisement" is more complicated because the variations include "ad-box", "adblock", "advert-box", "img_ad", "ads-section", and so on. Thus, we created a collection of regular expressions that match the commonly used advertisement class names and IDs, listed in Table 3.1.

\bad-	-ad\b
\bad_	_ad\b
\badv-	-adv\b
\badv_	_adv\b
advert	\bads
adblock	adbox

Table 3.1: A list of regular expression patterns that are used when searching for class names or IDs that designate an advertisement section. If a class name or an ID matches one of these patterns, the respective element (along with its descendants) will be considered part of the advertisement section of the webpage.

3.9 Summary

In Section 3.1, we took a closer look at the concept of main content and outlined the complications that arise when we try to accurately define it. We then defined defined the main content as the non-noisy content of a webpage. Section 3.2 provided a categorization of webpages and narrowed down the content extraction process to article webpages. In Section 3.3, we attempted to define the noisy content. In Section 3.4, we formulated content extraction as a classification problem that will be treated using machine learning and identified the instances to be classified as HTML elements. Section 3.5 provided a categorization of HTML elements and narrowed down the content extraction process to content elements. Section 3.6 described the format of the dataset to be used in this thesis and compared it with the formats used in other datasets. Section 3.7 described the process of annotating the webpages and producing the training set and the test set. Section 3.8 discussed the raw extracted features from webpages, along with the derived features that could be used by a machine learning algorithm.

Chapter 4

Experiment and Evaluation

This chapter begins by giving a brief description of decision trees, which is the model that we will try to induce. Additionally, an overview of the rpart package, the software package used in this thesis, is given.

Section 4.2 discusses the metrics that are usually used to evaluate the performance of binary classifiers in general, including content extractors. Section 4.3 discusses the HTML elements that we will build models for. Finally, Sections and 4.4 and 4.5 list the performance scores of our induced text and image element classifiers, respectively.

4.1 Using Decision Trees as Predictive Models

A **predictive model** is a simplified, high-level representation of a classifier. The predictive model that we attempt to construct in this work will be a decision tree, which is considered to be one of the most popular approaches for representing classifiers [Rokach and Maimon, 2005].

A **decision tree** is a finite tree graph. Each internal node in a decision tree corresponds to a test that is applied to a single feature of the instance that we wish to classify. The branches that stem from the node represent all the possible outcomes of the test. Leaf nodes represent predicted classes.

An example of a decision tree is demonstrated in Figure 4.1. In this example, the value of a specific feature is checked at each internal node; the branches that stem from the node form a **partition** of all the possible values of the feature that the node corresponds to.

4.1.1 The rpart Package

The rpart package [Therneau et al., 2015] was used to construct the decision trees in this thesis. The rpart package generates decision trees using ideas

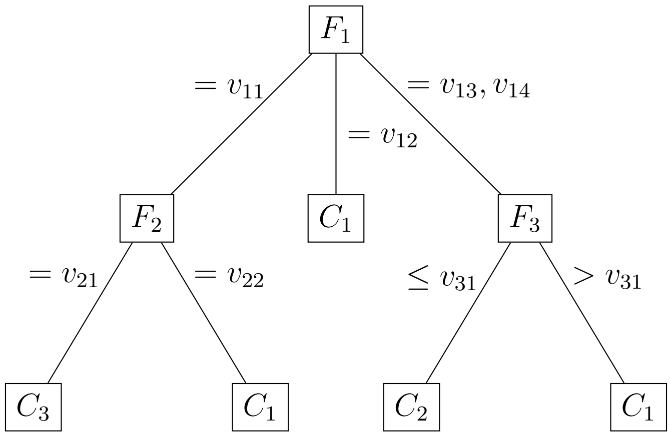


Figure 4.1: An example of a decision tree. The feature F_1 can have the values v_{11}, v_{12}, v_{13}, and v_{14}. If $F_1 = v_{12}$, then the class C_1 is predicted for the instance. If $F_1 = v_{11}$, then the value of F_2 is checked. If $F_2 = v_{21}$, then the class C_3 is predicted. If $F_2 = v_{22}$, then the class C_1 is predicted. If $F_1 = v_{13}$ or $F_1 = v_{14}$, then the value of F_3 is checked. If $F_3 \leq v_{31}$, then the class C_2 is predicted. If $F_3 > v_{31}$, then the class C_1 is predicted.

introduced by Breiman et al. [1984].

The decision trees that rpart constructs are classification and regression trees (CARTs) [Therneau et al., 1997]. A CART is a binary decision tree, in which each internal node corresponds to a boolean condition that is applied to one feature. The left branch that stems from the node represents the case that the boolean condition holds, while the right branch represents the case that the condition does not hold. Figure 4.2 demonstrates an example of a CART.

The rpart package employees *recursive partitioning* when building decision trees. The process of tree construction begins by finding the feature that *best splits* the training set into two subsets (based on the different values that the feature can take). Next the same process is repeated recursively with each new subset. The process stops when the size of the of the subsets reach a pre-defined minimum or until no more improvements can be made; that is, there is no splitting that can improve the current predictive model.

4.1.2 Splitting Criteria

When constructing trees, rpart tries to make the leaf nodes as "pure" as possible. Formally, the **impurity** of a node A is defined as

$$I(A) = \sum_{i=1}^{C} f(p_{iA}), \tag{4.1}$$

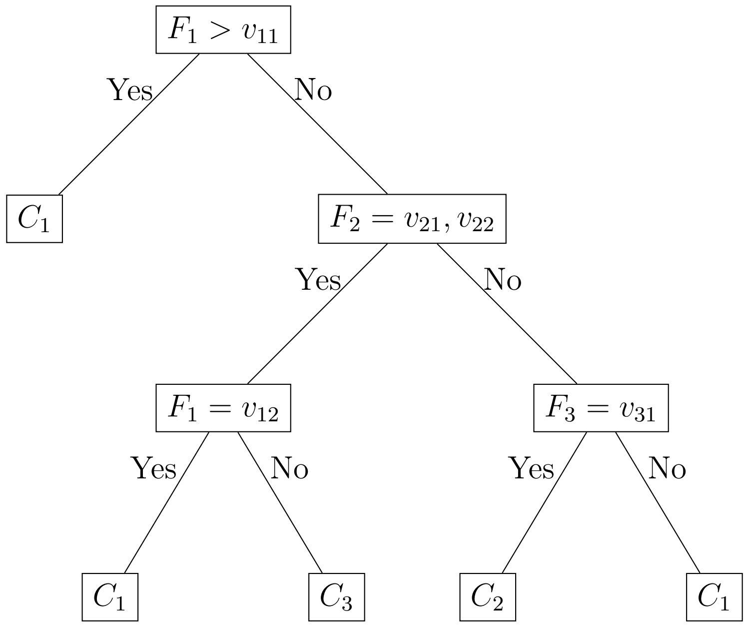


Figure 4.2: An example of a classification and regression tree. Each internal node represents a test that compares an exactly one feature against a single possible value (as in the case of the features F_1 and F_3) or a set of values (as in the case of the feature F_2). It should be noted that the same feature can appear multiple times on the same path from the root node to a leaf node, for example F_1 in this tree.

where C is the number of possible classes, p_{iA} is the proportion of instances in node A that belong to the class i, and f is some impurity measure.

The two candidates for f are the information index $f(p) = -p\log(p)$ and the Gini index $f(p) = p(1-p)$. According to Tan et al. [2006], the choice of impurity measure has little effect on the performance of decision tree induction algorithms because many impurity measures are consistent with each other. In this work, the information index was used.

When performing splitting, rpart tries to find the split with the maximum *impurity reduction*[1]. The impurity reduction that results from splitting a node A into two nodes A_L and A_R is given by

$$\Delta I = p(A)I(A) - p(A_L)I(A_L) - p(A_R)I(A_R), \tag{4.2}$$

where $p(A)$ is the number of instances in node A.

[1] impurity reduction is known as *information gain* when the information index is used as the impurity measure.

4.2 Evaluating the Performance of a Binary Classifier

When evaluating the performance of a binary classifier, we are interested in the number of errors that this classifier makes when applied to a specific test set, as well as the types of these errors.

4.2.1 Errors in Binary Classification

When carrying out binary classification, there are two types of errors that may occur:

Type I error Occurs when an instance is classified as positive when it is actually negative. Such an instance is said to be a false positive. In the case of content extraction, a type I error occurs when a noisy content is classified as main.

Type II error Occurs when an instance is classified as negative when it is actually positive. Such an instance is said to be a false negative. In the case of content extraction, a type II error occurs when a main content is classified as noisy.

Deciding which type of error is more grievous than the other depends on the specific application of content extraction.

4.2.2 Evaluation Metrics

In order to assess the performance of applying a given binary classifier C on a specific test set S, we first define the following subsets:

- S_p is the set of positive instances in S
- S_n is the set of negative instances in S
- C_p is the set of instances in S that were classified as positive by C. In content extraction, members of C_p are said to have been *retrieved* by C.
- C_n is the set of instances in S that were classified as negative by C.

Then the following metrics are defined as follows[2]:

$$\begin{aligned}
&\textbf{\textit{tp}} \text{ (number of true positives)} && = |C_p \cap S_p| && (4.3)\\
&\textbf{\textit{tn}} \text{ (number of true negatives)} && = |C_n \cap S_n| && (4.4)\\
&\textbf{\textit{fp}} \text{ (number of false positives)} && = |C_p \cap S_n| && (4.5)\\
&\textbf{\textit{fn}} \text{ (number of false negatives)} && = |C_n \cap S_p| && (4.6)\\
&\textbf{precision} && = \frac{|C_p \cap S_p|}{|C_p|} && (4.7)\\
&\textbf{recall} && = \frac{|C_p \cap S_p|}{|S_p|} && (4.8)\\
&\boldsymbol{F_\beta} && = (1+\beta^2)\cdot\frac{\text{precision}\cdot\text{recall}}{(\beta^2\cdot\text{precision})+\text{recall}} && (4.9)
\end{aligned}$$

In the context of content extraction, precision is the ratio of the actual main content that was retrieved by the classifier to the entire content that was retrieved, while recall is the is the ratio of the actual main content that was retrieved by the classifier to the entire actual main content in the test set.

The F_β metric a weight average of the precision and the recall, where β is a variable parameter. A higher value of β attaches more importance to the recall [Rijsbergen, 1979]. Usually the value $\beta = 1$ is chosen, which gives the precision and the recall the same importance, and the resultant metric is called the F_1 metric. The F_1 metric will be used in this thesis.

In addition, a confusion matrix is usually constructed to provide a summary of the performance of a binary classifier. The general form of a confusion matrix is illustrated in Table 4.1.

Actual Class \ Predicted Class	`"False"`	`"True"`
`"False"`	tn	fp
`"True"`	fn	tp

Table 4.1: The general form of a confusion matrix for a binary classifier.

4.2.3 A Clarification About the Evaluation Values

The sets defined in Section 4.2.2 consist of instances. As mentioned in Section 3.5, the classifier that we attempt to construct in this thesis operates on HTML content elements, such as paragraphs and headers. Thus, the *unit of*

[2] The notation $|A|$ denotes the number of elements in a set A.

measurement for the derived evaluation metrics is an HTML element. However, the size of text inside a single textual content element varies widely, so the values of these evaluation metrics may not convey the performance of a content classifier in terms of text size (measured in word number).

Nonetheless, it should be noted that the elements that contain a short inner text usually include important information, such as an author name or a section header. Therefore, it was decided in this thesis to calculate the evaluation metrics' values for the induced classifier when applied to textual content elements twice:

1. once using the classified HTML elements as units; and
2. once using individual words in each HTML element as units.

These values will be referred to as element based and text based, respectively.

Given a confusion matrix with HTML elements as units, the text based values can be easily obtained by concatenating the inner text of the HTML elements in each cell (given that the elements themselves are available).

4.3 Using Different Decision Trees for Different Element Types

As mentioned in Section 3.5, content extraction in this thesis will be applied only to content elements. During the learning process, it was discovered (unsurprisingly) that the classifier (as represented by the decision tree) varies significantly depending on the element type. For example, the decision tree of `<p>` elements is completely different from that of `<td>` elements.

Accordingly, it was decided to divide the training set into multiple training sets based on element type. Subsequently, a separate classifier for each training set was constructed using the rpart package. During the evaluation phase, the test set was divided in the same way as the training set. The learned classifiers were tested separately, each classifier on its respective test set.

Finally, the total evaluation scores for textual content extraction were computed. This was achieved as follows: Given the different values $tp_1, tp_2, \ldots, tp_k$ as defined in Equation 4.3 for the different classifiers, the total tp value was computed by summing these values: $tp = tp_1 + tp_2 + \cdots + tp_k$. The same procedure is repeated to obtain the total values for tn, fp, and fn. Finally, the total evaluation scores, as defined in Equations 4.7, 4.8, and 4.9, were computed.

4.3.1 The Elements to be Classified

This section lists the content elements for which we will attempt to develop a predictive model. All of the elements listed in this section, except for the `<img>` element, are textual content elements.

Paragraph Elements

The `<p>` elements are primarily used to mark up a text paragraph. In most webpages, long blocks of text consist of multiple `<p>` elements.

<div> Elements

The `<div>` elements are usually used as containers for organizing the content in a webpage (sectioning elements). However, there are cases where `<div>` elements are used as content elements. In this work, we regard a `<div>` element as a sectioning element, and not use it for classification, if *either* of the following conditions hold:

- The `<div>` element includes a content element as a descendant.
- The depth of the DOM subtree that descends from the `<div>` element is greater than 2.

These conditions are checked by the injected JavaScript code during the feature extraction stage (discussed in Section 3.7).

Cell Elements

Cell elements consist of `<th>` elements (table headers) and `<td>` elements. Cell elements are the building blocks of a table, represented by a `<table>` element. `<table>` elements are used to represent tabular data; however, they are often used for layout organization, in which case the cell elements should be regarded as sectioning elements. Such elements are filtered out in the same way as the sectioning `<div>` elements.

List Item Elements

A list item is represented by an `<li>` element. They form the building blocks of ordered lists (`<ol>` elements), unordered lists (`<ul>` elements), and menus (`<menu>` elements).

Header Elements

Headers are represented in decreasing importance by the `<H1>`, `<H2>`, `<H3>`, `<H4>`, `<H5>`, and `<H6>` elements. It should be noted that headers are sometimes represented differently using the `<div>` element, for example:

```
<div class="widget-header">Trending</div>
```

Such cases are handled under the `<div>` tags.

Caption Elements

The `<figcaption>` element marks up the caption for the data that is illustrated by a `<figure>` element. The data itself may be textual or pictorial.

Preformatted Text Elements

The `<pre>` element is used to mark up text with special formatting, usually computer code.

Image Elements

Images are represented by the `<img>` element. Its `src` attribute provides the URL of the described image.

4.3.2 Filtering Out Certain Elements

Before applying the learning algorithms to the elements in the training set, certain elements were filtered out. The same elements were filtered out of the test set before carrying out the evaluation. These elements are:

- All textual content elements with `num_words=0`. These correspond to elements that do not contain inner text or contain only whitespace.
- All content elements elements with `rect_size=0`. These correspond to elements that were not visible to the user during the annotation.

4.3.3 Manually Classifying Certain Elements

Before applying the learning algorithms, certain elements were filtered out of the training set and were *regarded* as noisy. These elements are:

- All <p> elements with `is_desc_a="1"`, `is_desc_nav="1"`, `is_desc_ad="1"`, and `is_link="1"`.

- All <div> elements with is_desc_a="1", is_desc_nav="1", is_desc_ad="1", and is_link="1".
- All <li> elements with is_desc_ad="1".
- All header elements with is_desc_a="1", is_desc_nav="1", is_desc_ad="1", and is_link="1".
- All <th> and <td> elements with is_desc_nav="1".

The reason for this procedure is that an examination of the training set revealed that these elements are almost always noisy content. Our original intention was to leave these elements in the training set and let the learning algorithm produce a decision tree that classifies these elements as noisy. However, a complication occurred due to the relatively small proportion of these elements. For example, Figure 4.3 shows a comparison between impurity reduction when splitting using the is_sib_p feature and the is_desc_nav feature of <p> elements. Both of these splits result in one highly pure node, namely is_desc_nav="1" and is_sib_p="1". The node is_desc_nav="1" is more pure than the node is_sib_p="1", but contains far fewer elements. Therefore, in accordance with Equation 4.2, the split using the is_sib_p feature produces a higher impurity reduction.

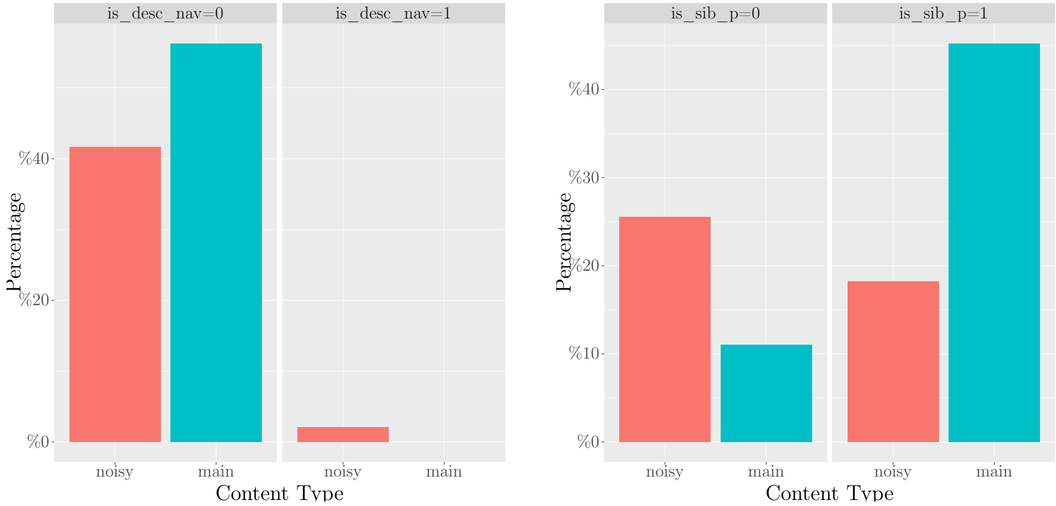
4.4 Evaluation Scores for Text Elements

A classifier was built for each type of textual content element and the performance scores for these classifiers were combined, as described in Section 4.3. The combined results were:

The element-based evaluation results:

$$\begin{aligned} &\text{precision} && = 0.828 \\ &\text{recall} && = 0.786 \\ &F_1 && = 0.806 \end{aligned}$$

Actual Class \ Predicted Class	"Noisy"	"Main"
"Noisy"	4625	211
"Main"	277	1018



(a) Content type distribution of <p> elements grouped by the values of the `is_desc_nav` feature.

(b) Content type distribution of <p> elements grouped by the values of the `is_sib_p` feature.

Figure 4.3: A comparison between the splits that the `is_desc_nav` and `is_sib_p` features produce. The node `is_desc_nav="1"` has only one class, so it has maximum purity, but it has a very small size. Thus, splting using the `is_sib_p` feature is perfered by the tree construction algorithm.

The text-based evaluation results:

$$\begin{aligned} \text{precision} &= 0.893 \\ \text{recall} &= 0.851 \\ F_1 &= 0.871 \end{aligned}$$

Actual Class \ Predicted Class	`"Noisy"`	`"Main"`
`"Noisy"`	496921	19618
`"Main"`	28654	163908

Remarks about the results:

Both the evaluation scores and the confusion matrices show that our induced classifiers performs better on the text-level than on the element-level. This is because elements that contain short text are generally harder to classify than those that contain long text.

The confusion matrices show that that most of the content in the test set is noisy, and that our induced classifiers were able to filter out most of the noisy content, as demonstrated by the high number of true negatives. However, the number of true negatives has no effect on the evaluation scores.

4.5 Evaluation Scores for Image Elements

The `<img>` elements were subdivided into two groups: One group contains what we consider small and medium sized images (≤ 40000px), and the other group contains what we consider large images (> 40000px).

Performance scores for small and medium sized images:

$$\begin{aligned} \text{precision} &= 0.833 \\ \text{recall} &= 0.205 \\ F_1 &= 0.328 \end{aligned}$$

Actual Class \ Predicted Class	"Noisy"	"Main"
"Noisy"	900	5
"Main"	97	25

Performance scores for large images:

$$\begin{aligned} \text{precision} &= 0.828 \\ \text{recall} &= 0.743 \\ F_1 &= 0.783 \end{aligned}$$

Actual Class \ Predicted Class	"Noisy"	"Main"
"Noisy"	122	6
"Main"	10	29

Remarks about the results:

Similar to the textual content elements, most of the image elements were noisy, and our induced classifiers were able to filter out most of the noisy content. The results also show that our induced classifiers perform better with large images than with small and medium sized images.

4.6 Summary

In Section 4.1, we discussed the decision tree model, which we used in this thesis, and took a closer look at the algorithm that is used by the rpart package to construct decision trees. Section 4.2 discussed the evaluation metrics that we used for evaluating the performance of our induced content classifiers and made the distinction between element-based and text-based values. Section 4.3 discussed the HTML elements that we classified as either main content or noisy content, and discussed the manual classification that we performed on certain elements. Section 4.4 listed the element-based and text-based performance results for our induced classifiers of textual content elements. Section 4.5 listed the performance results of our induced image classifiers.

Chapter 5

Conclusion and Potentional Future Work

This thesis provided a treatment of the problem of content extraction. We introduced the approach of element-based classification, which in turn facilitates high-granularity content extraction. During the course of the thesis, an annotation method was developed that facilitates the labeling of the main content of webpages based on the visual rendering of the webpages. A gold standard, which is easily expandable, was created using this method.

In Chapter 1, we discussed the importance and uses of content extraction. In Chapter 2, we explored a survey of the diverse approaches of performing content extraction. In Chapter 3, we investigated the possible definitions of the main content of a webpage and discussed the complications that can arise with each definition. Then we defined the main content as the non-noisy content because we thought the noisy content could be less ambiguously defined than the main content. Next, we outlined the process of manually annotating webpages and transforming the annotated webpages into a dataset that can be used by a machine learning software, were each instance in the dataset corresponds to an HTML element in a webpage. At the end of Chapter 3, we gave an overview of the features that we utilized during the process of learning. In Chapter 4, we discussed the general form of the predictive model that we attempted to induce in this thesis (namely decision trees), and we also took a look at the inner workings of rpart, which is the software package that we used for generating the predictive model. Next, we defined multiple metrics that are used for evaluating the performance of a binary classifier, and then we listed the values that we obtained for these metrics when we applied a classifier that is based on our induced predictive model to a test set.

5.1 Future Work

The approach we followed in this thesis could be improved in many ways:

Using a larger training set The training set that was used in this thesis consisted of only 30 pages. Using larger training sets generally results in higher-performance classifiers.

Using other machine learning software In this these, the rpart package was used. Other machine learning software could produce a superior classifier.

Using different derived features In this thesis, numerous features were derived from the raw extracted features and used in the learning process, as discussed in Section 3.8. However, there countless other features that could be derived.

Using different raw features This would require modifying the injected JavaScript program.

Utilizing headers This is discussed in Appendix A.

Appendix A

Utilizing Headers in Content Extraction

Simulating the way the user perceives the webpage is an effective method of carrying out content extraction. As discussed in Section 2.3, the Vision-Based Page Segmentation Algorithm attempts to simulate the way the user perceives the visual cues in a webpages.

Another kind of cues that give important hints to the user are the semantic cues, in particular the headers of webpage sections. For instance, when the user reads the header "See Also," she understands that the respective webpage section contains links to other webpages. Table A.1 contains a list of headers that are commonly displayed on top of webpage sections that consist entirely of noisy content. These headers were manually extracted from numerous websites.

Table A.1: Commonly used headers that designate noisy content sections in a webpage. Any webpage section that has one of these headers can be immediately filtered out as noisy content.

Advertisement	Also In Entertainment News	Also Read
Around the Web	Cookie Control	Editor's Choice
Elsewhere on [Website Name]	External Links	Featured Sections
From Around the Web	Further Reading	Just In
Latest News	More News	More from [Website Name]
More from the Author	More to Explore	Most E-Mailed
Most Popular	Most Popular Stories	Most Viewed Today
News From Your Area	Next In Entertainment News	On Our Radar
Paid Content	Paid Partner Content	Partner Content
Recent News	Recent Posts	References
Related	Related Content	Related Coverage
Related Links	Related to This Story	Recommended
See Also	Share This Article	Share This Story

Sign Up	Sponsor	Sponsored Content
Sponsored Links	Sponsored Posts	Sponsored Topics
Sponsored Stories	Sport Headlines	Subscribe
Subscribe and Follow	Take a Look	The Best of [Website Name]
Top News	Top Stories	Trending Articles
Trending Today	Trending on [Website Name]	What's Hot
You May Also Like	You May Like	You Might Like

Glossary

article webpage A webpage that includes a substantial amount of information about a specific topic in the form of textual content.

binary classification problem A classification problem where the set of classes consists of “true” and “false”.

binary classifier A classifier whose output is either “true” or “false”.

block A visually contiguous portion of a webpage.

boolean feature A feature that can assume either of the values “true” or “false”. It usually indicates the presence or absence of a property in the phenomenon that is abstracted by an instance.

classification and regression tree A binary decision tree, in which each internal node corresponds to a boolean condition that is applied to one feature, and the branches correspond to whether the condition is satisfied or not.

classification problem The problem of assigning an instance to an element of a pre-defined set of classes.

classifier A function from a set of instances of a certain type to a finite set of classes. This function must be computable by a machine.

confusion matrix A table layout for evaluating the performance of a classifier on a test set. The rows represented the actual classifications of the instances in the test set, while the columns represent the predicted classifications.

content Any arbitrary part of a webpage.

content element An HTML element that enclose a piece of data, identifying its purpose.

content extraction Another term for main content extraction.

decision tree A predictive model of a classifier in the form of a tree graph. Each internal node in the tree represent a test that is a applied to a feature value. Each branch that stems from the node represents a possible outcome of the test. Leaf nodes represent predicted classes.

directory webpage A webpage whose purpose is to provide links to other webpages.

element-based metric value A classifier evaluation metric applied to a content classifier where the length of content is measured in the number of HTML elements.

false negative A positive instance that has been falsely classified as negative by a binary classifier.

false positive A negative instance that has been falsely classified as positive by a binary classifier.

feature An individual measurable property of a phenomenon being observed.

form webpage A webpage whose purpose is to receive input from the user.

gold standard A term that is used to refer to either the training set or the test set.

granularity (In the context of content extraction) the level at which webpage contents are divided and classified as main or noisy.

inline semantic element An HTML element that gives a semantic meaning to an arbitrary piece of text.

instance A specific observable phenomenon of any type, such as a person, a rasterized image, or a piece of text. An instance is specified by the values of its features.

machine learning Either supervised machine learning or unsupervised machine learning.

main content Roughly speaking, the part of a webpage that makes it useful.

main content extraction The process of identifying the main content in a webpage.

negative instance An instance whose actual class is "false" in a binary classification problem.

noisy content Any type of content in a webpage other than the main content.

nominal feature A feature that can assume one value in a finite set of permissible values. The permissible values have no meaningful order.

numerical feature A feature that can assume a numerical value.

partition A partition of a set A is a set of disjoint non-empty setsets $P = \{A_1, A_2, ...\}$ such that

$$\bigcup_{A_i \in P} A_i = A.$$

.

positive instance An instance whose actual class is “true” in a binary classification problem.

precision The ratio of the positive instances to all the instances that were classified as positive by a binary classifier.

predictive model A high-level abstraction of a classifier.

recall The ratio of the instances that were classified as positive by a binary classifier to all the positive instances in a test set.

sectioning element An HTML element that includes other HTML elements in order to organize them in a certain way or to designate them as semantically related.

supervised machine learning The procedure of inducing a predictive function (a binary classifier in the case of content extraction) from a training set.

template A webpage layout that contains slots where arbitrary contents can be inserted.

template recognition The task of analyzing a set of webpages that are based on the same template in order to discover the template structure.

test set A set of instances whose classes have been identified (usually manually by a human user). The test set is used to assess the performance of a classifier.

text-based metric value A classifier evaluation metric applied to a content classifier where the length of content is measured in the number of words.

training set A set of instances whose classes have been identified (usually manually by a human user). The training set is used by a supervised machine learning algorithm to train a classifier to automatically classify instances with unknown classes.

type I error The classification of a negative instance instance as positive in a binary classification problem.

type II error The classification of a positive instance instance as negative in a binary classification problem.

unsupervised machine learning Similar to supervised machine learning, except that the classes of the instances in the dataset do not have to be identified in advance.

webpage cluster A set of webpages that are based on the same template.

wrapper A program or procedure for extracting information from webpages.

Bibliography

Marco Baroni, Francis Chantree, Adam Kilgarriff, and Serge Sharoff. Cleaneval: a competition for cleaning web pages. In *LREC*, 2008. 3.6

Christopher Bishop. *Pattern recognition and machine learning*. Springer, 2006. 3.4.1

Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984. 4.1.1

Deng Cai, Shipeng Yu, Ji-Rong Wen, and Wei-Ying Ma. Vips: A vision-based page segmentation algorithm. Technical report, Microsoft technical report, MSR-TR-2003-79, 2003. 2.3

W. Bruce. Croft, Donald Metzler, and Trevor Strohman. *Search Engines: Information Retrieval in Practice*. Addison-Wesley, 2010. 1.2

Aidan Finn, Nicholas Kushmerick, and Barry Smyth. Fact or fiction: Content classification for digital libraries, 2001. 2.1

Forbes Magazine. How much of the internet does the wayback machine really archive, 2015. URL `http://www.forbes.com/sites/kalevleetaru/2015/11/16/how-much-of-the-internet-does-the-wayback-machine-really-archive`. Accessed: 2016-10-22. 2

Thomas Gottron. *Content Extraction: Identifying the Main Content in HTML Documents*. PhD thesis, Johannes Gutenberg-Universität in Mainz, 2009. 2, 2.4, 2.5, 3.1, 1

Suhit Gupta, Gail Ka iser, David Neistadt, and Peter Grimm. Dom-based content extraction of html documents. In *Proceedings of the 12th international conference on World Wide Web*, pages 207–214. ACM, 2003. 2.2

Thiago S Guzella and Walmir M Caminhas. A review of machine learning approaches to spam filtering. *Expert Systems with Applications*, 36(7):10206–10222, 2009. 1.1

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*, pages 441–450. ACM, 2010. 3.6

Nicholas Kushmerick, Daniel S. Weld, and Robert Doorenbos. Wrapper induction for information extraction. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1997. 2.4

Shian-Hua Lin and Jan-Ming Ho. Discovering informative content blocks from web documents. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 588–593. ACM, 2002. 2.5, 2.5

Bing Liu. *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data*. Springer Science & Business Media, 2007. 2.4

Wei Liu, Xiaofeng Meng, and Weiyi Meng. Vision-based web data records extraction. In *Proc. 9th International Workshop on the Web and Databases*, pages 20–25, 2006. 2.3

Samuel Louvan. Extracting the main content from web documents, 2009. 1.1, 3.8.1

Mozilla Developer Network. Html element reference, 2016. URL `https://developer.mozilla.org/en/docs/Web/HTML/Element#Inline_text_semantics`. Accessed: 2016-10-22. 3.5

David Pinto, Michael Branstein, Ryan Coleman, W. Bruce Croft, Matthew King, Wei Li, and Xing Wei. Quasm: A system for question answering using semi-structured data. In *Proceedings of the 2Nd ACM/IEEE-CS Joint Conference on Digital Libraries*, 2002. 2.1

Martin F Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980. 2.5

C. J. Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, Newton, MA, USA, 2nd edition, 1979. ISBN 0408709294. 4.2.2

Lior Rokach and Oded Maimon. Top-down induction of decision trees classifiers-a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 35(4):476–487, 2005. 4.1

Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001. 2.5

Koen Smets, Bart Goethals, and Brigitte Verdonk. Automatic vandalism detection in wikipedia: Towards a machine learning approach. In *AAAI workshop on Wikipedia and artificial intelligence: An Evolving Synergy*, pages 43–48, 2008. 1.1

Ruihua Song, Haifeng Liu, Ji-Rong Wen, and Wei-Ying Ma. Learning block importance models for web pages. In *Proceedings of the 13th international conference on World Wide Web*, pages 203–211. ACM, 2004. 3.8.1

Johnny Stenback, Philippe Le Hégaret, and Arnaud Le Hors. Document object model (dom) level 2 html specification. *W3C Recommendation*, 2003. 2.2

Pang-Ning Tan et al. *Introduction to data mining*. Pearson Education India, 2006. 4.1.2

Terry Therneau, Beth Atkinson, Brian Ripley, and Maintainer Brian Ripley. Package 'rpart', 2015. 4.1.1

Terry M Therneau, Elizabeth J Atkinson, et al. An introduction to recursive partitioning using the rpart routines, 1997. 4.1.1

Ziyan Zhou and Muntasir Mashuq. Web content extraction through machine learning, 2014. 1.1, 3.8.1