Structured Extraction of Terms and Conditions from German and English Online Shops

Tobias Schamel
Technical University of Munich
tobias.schamel@tum.de

Daniel Braun
University of Twente
d.braun@utwente.nl

Florian Matthes
Technical University of Munich
matthes@tum.de

Abstract

The automated analysis of Terms and Conditions has gained attention in recent years, mainly due to its relevance to consumer protection. Well-structured data sets are the base for every analysis. While content extraction, in general, is a well-researched field and many open source libraries are available, our evaluation shows, that existing solutions cannot extract Terms and Conditions in sufficient quality, mainly because of their special structure. In this paper, we present an approach to extract the content and hierarchy of Terms and Conditions from German and English online shops. Our evaluation shows, that the approach outperforms the current state of the art. A python implementation of the approach is made available under an open license.

1 Introduction

Terms and Conditions (T&Cs) of online shops are rarely read and even more rarely understood (Bakos et al., 2014), although we all still accept them. In recent years, the automated analysis of T&Cs has become an interesting field of research (Braun and Matthes, 2021; Lippi et al., 2019). A structured extraction of T&Cs from online shops is a necessary prerequisite for such further processing in an NLP pipeline.

Content extraction is a well-researched task and numerous open source libraries are available. Existing approaches are predominantly designed for or based on news articles and blog posts. In this paper, we will show that the existing approaches are not well-suited to deal with T&Cs, partially because of the strict hierarchical structures that can be found in such legal documents, which are not common in other types of content such as news articles.

In this paper, we present a domain-specific content extraction approach for T&Cs, that combines both, *Content Extraction* and *Hierarchy Extraction*, and an open source Python library that implements this approach. needs to combine both. Our evaluation shows that the library outperforms general-purpose content extraction approaches on T&Cs from German and English online shops. The library is available on GitHub: `https://github.com/sebischair/LowestCommonAncestorExtractor`.

2 Related Work

The existing work on content extraction has been mainly focused on news articles or generic content extraction, often based on the dataset of the cleaneval competition (Baroni et al., 2008). To the best of our knowledge, no work exists that specifically targets or is evaluated on T&Cs.

Gibson et al. (2007) described the process of content extraction as a sequence labeling problem on a document broken down into a sequence of blocks. Each block needs to be classified as either *Content* or *NotContent*. Kohlschütter et al. (2010) work with a more detailed four-class separation. Another approach is a Boundary Detection Method where a heuristic needs to determine a *Start*- and *End*-Block framing the whole content, i.e. everything between *Start*- and *End*-Block is considered *Content* whereas everything else is discarded as *NotContent*. According to Jiménez et al. (2018), the classification needs to take both HTML structure and the actual content into account. "Purely text-based or purely HTML-based approaches do not have perfect results."

Several approaches for classifying blocks as *Content* or *NotContent* can be found in the literature.

Kohlschütter et al. (2010) propose a classification inspecting the text on a functional level using a set of so-called shallow text features. Shallow text features are statistical calculations on block-level looking at domain and language independent features like link density, the average sentence length, the uppercase ratio, etc. Jiménez et al. (2018) introduce an improvement to this algorithm by also

Proceedings of The Fifth Workshop on e-Commerce and NLP (ECNLP 5), pages 181 - 190
May 26, 2022 ©2022 Association for Computational Linguistics

taking the HTML tree structure into account.

Pomikálek (2011) introduced a similar approach using a low amount of features to determine the likelihood of a block being *Content* or *NotContent*. Uncertainties are dealt with within the next step, which involves an analysis of the relative position of a block in the HTML tree including the classification of its neighbors. This step is based on the assumption that *Content* blocks are to be found near other *Content* blocks (and vice versa).

Pasternack and Roth (2009) tried to solve the task by finding a maximum subsequence in tokenized HTML documents, where each token is assigned a score determined by token-level classifiers. Different classifiers like simply assigning predefined scores to words and tags and more advanced classifiers which combined Naive Bayes classification with features of the surrounding tokens were investigated.

There is a number of synonyms to the process of "content extraction" (Gibson et al., 2007; Barbaresi, 2019) like *boilerplate detection/removal* (Kohlschütter et al., 2010), *template matching* (Sano et al., 2021) and *cleaning* (Lejeune and Zhu, 2018; Kilgarriff, 2007).

In addition to extracting the content, a (relatively small) number of approaches also try to extract the hierarchy of the content. According to Manabe and Tajima (2015), the nested hierarchy of an HTML document can contribute some information to hierarchy extraction but does not necessarily coincide with the actual hierarchical structure of an HTML document. The HTML tags originally meant to structure a document and indicate headings are often misused for SEO or not used at all.

Manabe and Tajima (2015) introduced a segmenting method extracting the hierarchical structure of HTML documents based on the differences in the visual styles in hierarchical headings. They defined a set of rules based on the way humans read hierarchically structured content. According to them, headings a characterized by more prominent visual styles (the same on one level of hierarchy) preceding the blocks they describe.

Sano et al. (2021) used a similar approach with only nine parameters. The parameters include the number of child nodes, text length of nodes and the styling of succeeding content.

1. headings have few child nodes
2. headings have a short text length
3. the width of headings is greater than their height
4. the size of a heading is smaller than the size of the following content block underneath it

3 Requirements

In order to derive requirements for the extraction of T&Cs, we compared existing extractors and our expected results. In addition, we manually inspected T&C pages to detect patterns and domain-specific characteristics. The test data was sampled from the data set by Braun and Matthes (2020) which is available under the CC BY-SA 3.0 license on GitHub[1]. We investigated the extraction results of the T&Cs pages of 30 German and 20 English online shops. 15% of the English sample had to be adjusted, as the URL from the data-set did not point to the actual T&Cs page.

3.1 Content Extraction

Through visual inspection of the T&Cs page and the DOM tree, we identified some prevalent patterns: While news articles are often interrupted by references to similar articles and advertisements, all cases examined in the sample of T&Cs pages displayed the relevant legal document without any interruptions on a rather simply structured page. For T&Cs of German online shops, the content is not always grouped within a single large paragraph but often divided into multiple paragraphs, e.g., *1. Allgemeine Geschäftsbedingungen* (German for general terms and conditions) and *2. Kundeninformation* (customer information), where both contain relevant information. Generally speaking, German T&Cs tend to be more structured than their counterpart in English online shops. In most of the cases, the relevant content shared a common style (font size and style) and could be found in the same depth of the DOM tree. This same style was used in the footer of the page in rare cases. Exceptions to these observations are headlines and differently styled withdrawal forms, which are contained in a number of T&C pages. These withdrawal forms are often comprised of underscores, blank spaces, and text. A small number of T&C pages had content that needed to be unfolded making and purely visual approach insufficient, as parts of the relevant content are not seen without further interaction with specific elements of the website.

[1] https://github.com/sebischair/TC-Detection-Corpus/

3.1.1 Existing Solutions

We compared the content extraction performance of three existing libraries on the data set. The extraction results were quantified by classifying the extraction quality for the following properties (quality sorted from *correct* to *most severe* error):

- content start: *correct*, *too early*, *too late*
- content end: *correct*, *too late*, *too early*
- main content: *correct*, *missing content* (links & addresses), *none-content* (whole paragraphs or sentences), *missing & none-content*

If the content start is detected earlier than it actually is, noise is added to the content, however, if the content start is detected later than it actually is, content is cut off and information lost. The latter is the more severe error. For the end of the content, the reverse is true.

The following three content extraction libraries were tested:

Boilerpipe We compared three different extractors implemented in *Boilerpipe*[2]:

ArticleExtractor The end of the main content was often identified too early, i.e., content towards the end of the pages was cut off. This was often caused by addresses and other contact information. Less frequently, *ArticleExtractor* also had problems identifying the start of the main content.

CanolaExtractor The *CanolaExtractor* was trained on the KrdWrd Canola corpus[3], which is created from random webpages across various domains (Stemle, 2009). It recurrently extracted cookie information or the footer of the web pages. The main content was usually detected but the extracted text was interrupted many times. This occurred for some short paragraphs, headings, addresses and withdrawal forms.

LargestContentExtractor The *LargestContentExtractor* extracts a continuous piece of content in all cases. This connected part is not a single HTML node but a consecutive series of HTML nodes. By nature, the extraction performance for the center of the extracted content is excellent. However, the performance on the identification of the start and end of the main content is the worst observed in comparison with the other extractors.

JusText The performance of *JusText*[4] Pomikálek (2011) was similar to the results of the Boilerpipe CanolaExtractor. The main content was usually detected but the extracted text was interrupted, as *JusText* classified many headlines, addresses, and paragraphs containing links as boilerplate.

Trafilatura *Trafilatura*[5] performed best among the investigated solutions. However, *Trafilatura* ignored some of the paragraph headlines. In addition, there were problems with content that was not visible to a user without unfolding them in the browser manually.

6 of 20 sampled web pages from the English sample were not extractable due to malformed HTML. As small mistakes in the HTML structure are not uncommon and are usually fixed by browsers rendering them, this should not be the case. The proposed solution should be robust when encountering that type of problem. The detailed results can be found in Tables 2 and 3.

3.1.2 Derived Requirements

Based on the identified patterns in the data and the problems of the existing approaches, we derived the following reuqirements for a domain-specific content extraction approach for T&C:

1. extract (largest) continuous part of HTML document
2. extract the content sharing a common style and depth in HTML tree
3. extract the withdrawal form/information with different style and different depth
4. extract address information
5. always extract both of the sections *1. Allgemeine Geschäftsbedingungen* (general terms and conditions) and *2. Kundeninformation* (customer information) in German T&Cs
6. extract *hidden* content which needs to be unfolded in the browser (e.g. by clicking an expand button)
7. robust against malformed HTML

[2] https://code.google.com/p/boilerpipe
[3] https://krdwrd.org/
[4] https://code.google.com/archive/p/justext/
[5] https://github.com/adbar/trafilatura

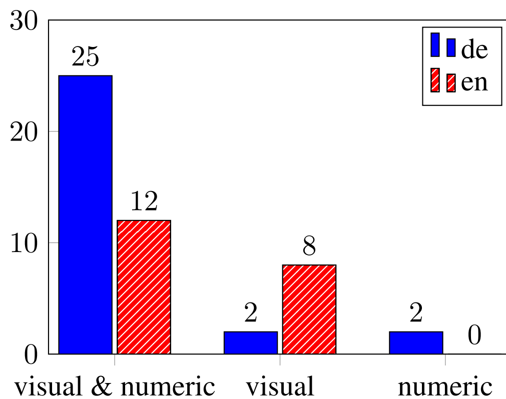

Figure 1: Occurrences of different hierarchy styles in the German and English sample

3.2 Hierarchy Extraction

Data on the visual and HTML-based representations of the hierarchical structure of web pages containing T&Cs were gathered in a manual review. The results found in the English sample and the German sample differed slightly. The following hierarchy representation classes were defined:

Visual page is structured using visual separators and different text styles

Numeric page is structured using a numeric scheme (Arabic, Latin/Roman, alphabetic, etc.)

Visual & Numeric a combination of both *Visual* and *Numeric* elements are used to structure the page

The analysis showed, that German shops were much more likely to use a combination of *Visual & Numeric* features to structure the content (see Figure 1).

Based on our observations, we formulated the following requirements:

1. extract subclauses grouped in their own paragraph forming HTML tag (block elements)
2. detect numeration patterns (alphabetic, Arabic, Latin/Roman, section sign, etc.)
3. extract styling (CSS) of titles to determine associated content
4. ignore enumerations for the table of contents

4 Approach and Implementation

This section covers the approach we developed based on the identified requirements and the design and implementation of the *StructuredLegalExtraction* library that implements the approach in Python.

4.1 Architecture

The extraction library consists of two main components described in Section 4.3 and Section 4.4 which serve to extract the content and the hierarchy of a given T&C page. In addition, there is an auxiliary component that serves to download the web page and one that transforms the HTML content into a DOM tree which is by far more useful for further processing. Another auxiliary component is used to generate the target structure relying on another auxiliary component to segment sentences.

At first, the content is downloaded using the *Downloader* component. The downloaded content is processed by the *DOMParser* to achieve the desired DOM tree holding no more information than those actually needed during later processing steps. The *ContentExtractor* component is responsible for detecting and extracting the main content of the downloaded page based on the parsed DOM tree. The *HierarchyExtractor* component uses the main content node of the DOM tree to build its hierarchy tree based on visual information (CSS attached to DOM nodes) and numerical patterns in the text. In the last step, the hierarchy tree needs to be transformed to the target format by the *TargetFormat* component, which uses the *SentenceSegmenter* component to tokenize and segment the content.

4.2 Additional Technologies

Some generic tasks can be solved by using existing solutions. The requirements and selected libraries are presented in this section.

4.2.1 Web Page Download

In an effort to determine the main content of a web page, the entire content must first be downloaded. The following requirements have been identified:

- download full HTML file representing the web page
- extract CSS style information on the content
- full XML Path Language (XPath) support to navigate the dom tree

Selenium[6] is a well-known library usually used for website testing. By this, it allows full interaction with the website using X-Path by controlling a browser. This also allows us to use the

[6] https://github.com/SeleniumHQ/selenium/

browser's HTML error correction when downloading the HTML. However, the `find_by_xpath (path)` method does not support text elements. Text can be extracted by accessing an element's `.text` attribute. Unfortunately, text segments (complete text - direct and indirect children - under the current node) cannot be mapped to the individual nodes so that one could track down which text has which style. This will require another library.

An outstanding feature of selenium is its capability of extracting CSS style information. The high number of visual represented hierarchies (see Figure 1) makes this a crucial feature.

4.2.2 HTML Parser

As Selenium does not support text nodes in its X-Path functionality, there is a need for a dedicated parser with the following requirements:

- full XPath support to navigate the DOM tree
- XPath generation from DOM tree nodes

lxml[7] an XML-parser offering dedicated HTML-parsing functionality. Due to its full XPath support, it also allows extracting all child nodes including the text nodes of a DOM-node using the XPath `child::node()`. The library also allows generating XPath for elements relative to a given parent element which makes it possible to link it to other libraries capable of XPath for style extraction (see Section 4.2.1).

4.2.3 Sentence Segmentation and Tokenising

The content of the document needs to be segmented into sentences which themselves need to be tokenized for the target format. This requires reliable segmenting and tokenizing with respect to text elements like references to laws or address information common in the domain of T&Cs.

According to Braun (2021), *SoMaJo*[8] performed best in the domain of T&Cs .

LangID[9] is used, as *SoMaJo* requires knowledge on the language of the text to segment and tokenize. LangID's multinomial naive Bayes model is trained to determine a text's language among 97 languages including German and English (Lui et al., 2021).

4.3 Content Extraction

As described in Section 3, the main content of a T&Cs page usually shares a common style. The main content makes up for the largest visible content block most of the time. Therefore, the extraction approach can be built upon this knowledge.

In a first step, we determine the MCS (Most Common Style) by traversing the DOM tree nodes while collecting a mapping of styles to number of characters. Different approaches to determine and approximate the style of a given node were investigated.

Naïve Style The first naïve approach is combining the HTML tag and all the attributes (incl. CSS classes and ids). However, this is just an approximation, as CSS style information is passed to child nodes of the parents holding them. This approach is rather efficient but less accurate.

Naïve Style and Short Text Exclusion As navigation bars and headlines consist of nodes holding only one to two words, it can be dangerous to include them when determining the MCS, as nested navigation elements hold large amounts of characters. Excluding short text nodes (< 4 words) can solve that issue

Rendered Style Using *Selenium* allows the retrieval of the rendered style information.

It turned out, that the *Naïve Style and Short Text Exclusion*'s approximation is almost as accurate as the *Rendered Style* approach. Using the *Naïve Style and Short Text Exclusion* approximation, we can limit the time-intensive rendered style extraction through *Selenium* to the actual main content for hierarchy extraction (see Section 4.4). The number of nodes for which the rendered style needs to be extracted can be reduced by approximately 69% in the German sample and by approximately 71% in the English sample.

After the MCS is identified, the tree is traversed to identify a node covering at least 85% (this threshold is variable) of the characters of the MCS. Once this node is identified, all descendants are classified as *Content*. This decision is justified by the findings in Section 3: T&Cs are usually continuous texts often structured into a container (e.g. a `<div>`) by content management systems.

The selection of the right (domain-specific) threshold is crucial for the success of this extraction algorithm. A threshold too low can result in only a part of the main content being extracted; a threshold too high can make it impossible to find a node covering the given amount of MCS characters

[7] https://github.com/lxml/lxml/
[8] https://github.com/tsproisl/SoMaJo/
[9] https://github.com/saffsd/langid.py

and thereby triggering a fallback solution described later. After investigating the MCS (*Naïve Style and Short Text Exclusion*) coverage of the nodes holding the main content and the next largest child in the German and English samples from Section 3, the lower bound of the interval of possible threshold values is found to be limited to approximately 83.65% by the English sample (lowest MCS coverage of the next largest MCS coverage in a main content's child node). The upper bound of this interval is defined by the lowest coverage of the main content node above 83.65%, which can also be found in the English sample with a value of approximately 91.01%. Therefore, the threshold is set to 85%.

In some rare cases, the algorithm will not find a node covering at least 85% of the MCS, as the webpage does not use a dedicated container to hold the main content. Instead, individual containers for each of the paragraphs are placed as direct descendants of the `<body>` node. By identifying the longest subsequence containing the MCS, one can most likely extract the main content while excluding boilerplate content like the navigation bar and footer. This fallback solution provides much worse results than the actual content extraction algorithm.

We call the extraction algorithm *LowestCommonAncestorExtractor*

4.4 Hierarchy Extraction

The hierarchy extraction is based on the idea presented by Manabe and Tajima (2015). The visual style of the text is used to identify headings and their associated text blocks. In some cases, T&Cs are also structured using enumerations or a combination of visual style and enumerations (see Figure 1). Thus, this information needs to be considered, too. For the actual hierarchy extraction information from both, the enumerations and the visual styles, need to be taken into account in a rule-based approach in order to produce accurate results. Information from the DOM tree does not provide reliable information on the hierarchy. Given the data from Figure 1, the hierarchy extraction algorithm will focus on visual features and use numeric patterns for verification and adjustments.

In a first step, the DOM (sub-)tree identified as the main content is converted to a list of content blocks where each block has its own style. A block is a sequence of characters ending with a forced newline. This forced newline could be a `<br>` tag, the start or end of a paragraph (`<p>`), or any other element with `block` as the standard level for the `display` property. The style of a block corresponds to the style that the majority of the characters in it are part of (excluding the anchor tag `<a>`).

The style attached to a block is determined by extracting the rendered style retrieved through *Selenium*. The style is defined by font-decorations, font-weight, font-size, font-family and font-color.

For each of the blocks, possible enumeration patterns are identified and attached to the blocks by using the following regular expression:

```
\s[\(§]?(([IVXLivxl]{1,7})|
([0-9]{1,2})|[a-zA-Z])
([\.\-,:](([IVXLivxl]{1,7})|
([0-9]{1,2})|[a-zA-Z]))*[\-:\.)]?
\s
```

Arabic enumeration Arabic numbers are the most common enumeration used in the domain of T&C structuring. They can easily be extracted from a string and transformed into an integer representation.

Roman enumeration Since in a few cases Roman numerals were also used for numbering, these must be converted into Arabic numerals. In all investigated cases Roman numerals were not bigger than 20. By limiting the allowed Roman enumeration characters to *I*, *V*, *X* and *L*, the risk of mixing up alphabetic and roman enumeration can be reduced, as *I* would only be used in alphabetic enumerations larger than 8.

Alphabetic enumeration As alphabetic enumeration is only applicable for single characters, there is no need for extensive conversion. Letters are mapped to their position in the alphabet.

Lists `<li>` elements are automatically separated into blocks, however, enumerations rendered by the browsers when using `<ol>` tags are not part of the textual content of the blocks. Thus, the information about list enumerations is attached to the block as regular enumeration information.

The following assumptions by Manabe and Tajima (2015) are used as a basis for the visual hierarchy extraction:

1. headings appear at the beginning of the corresponding blocks

2. headings are given prominent visual styles
3. headings of the same level share the same visual style

A section's start is identified by determining its headline, i.e. a line with a different style than the MCS identified during content extraction. A headline is only allowed a maximum of 10 words. The section ends whenever the next line styled like the current section's headline occurs. The content in between these two lines forms the provisionally content of the upper headline. Each of the identified sections is then grouped into subsections using the same algorithm (see Appendix, Algorithm 1) until no more prominent style is visible in between two headlines. Assuming, that no section is interrupted by another section and later continued, content in between two headlines, respectively the last headline and the end of the main content, is assumed to be the content of the upper headline.

Enumerations extracted during the conversion to the block list are used to correct and validate the existing hierarchy which was extracted visual features. In the first step, the blocks assigned to each of the visually separated sections are examined for possible numerical hierarchies. List enumerations are treated in a special way, as we allow them to interrupt a section that is continued after the list element blocks. Given that the section can be divided into further subsections based on the enumeration patterns found in the blocks, these subsections are processed in the same way. After the initial enumeration-based segmentation within the nodes' content blocks, all headlines on the same level of the tree are checked for enumeration patterns. If there are different enumeration patterns on one level, the tree is modified in order to have consistent enumeration.

An enumeration pattern is only considered if there are at least two consecutive numberings of that pattern whose numerical values reflect a valid step. Invalid steps or enumeration patterns occurring only once are most likely to be detected due to an error in the enumeration detection and thus ignored.

4.5 Target Format

The tree-like results from content and hierarchy extraction are converted to JSON after being segmented and tokenized.

	too late	too early	correct
start	3	0	46
end	0	2	47

Table 1: Extraction performance for the LowestCommonAncestorExtractor on the test set. 49 out of 50 web pages in the test set could be processed.

5 Evaluation

The performance of the developed algorithms is evaluated in this section. Besides the sample used to derive the requirements, there is also another sample to evaluate the library to prelude overfitting.

5.1 Content Extraction

The content extraction algorithms were tested with a focus on the correct identification of the start and end of the content, as the center is always identified correctly given the functionality of the previously introduced *LowestCommonAncestorExtractor* with *NaïveStyle*, *ShortTextExclusion*, and a threshold of 85%. The results of the evaluation are shown in Table 1.

During the evaluation, the following three reasons were identified as the main drivers of extraction errors: (1) **threshold too high**: Whenever the threshold is too high for the page, the fallback algorithm is triggered; (2) **no container for main content**: Whenever the T&C page lacks a container wrapping the whole main content, the fallback algorithm is triggered; (3) use of different tags: The NaïveStyle approach cannot handle the usage of different tags rendering to the same actual style used for the main content. If one of the tags is held in its own container, as often is the case in lists, only this container is extracted.

However, the *LowestCommonAncestorExtractor* performed significantly better than the previously examined extractors, as shown in Tables 2 and 3.

5.2 Hierarchy Extraction

The hierarchy extraction algorithm was, similar to the content extraction, evaluated with the sample used to derive requirements and a test sample created for the purpose of evaluation. Errors arising from a failed content extraction are ignored in this section, as they provide no information on the quality of the algorithm applied during hierarchy extraction.

The algorithm showed good results (see Ap-

	too late	too early	correct	processing error
LowestCommonAncestorExtractor	**1**	3	**45**	**1**
Boilerpipe ArticleExtractor	16	4	24	6
Boilerpipe LargestContentExtractor	29	2	13	6
Boilerpipe CanolaExtractor	7	15	22	6
JusText	6	11	25	6
Trafilatura	5	**2**	37	6

Table 2: Performance of detecting the start of T&Cs (for “correct”, higher numbers are better, for all others, lower numbers are better).

	too late	too early	correct	processing error
LowestCommonAncestorExtractor	**3**	2	**44**	**1**
Boilerpipe ArticleExtractor	23	13	8	6
Boilerpipe LargestContentExtractor	32	**1**	11	6
Boilerpipe CanolaExtractor	**3**	27	14	6
JusText	8	15	19	6
Trafilatura	5	4	35	6

Table 3: Performance of detecting the end of T&Cs (for “correct”, higher numbers are better, for all others, lower numbers are better).

pendix A.2). In most cases, small extraction errors can be found in the hierarchy. However, their impact on the overall result can be described as minor. A precise analysis of the sources of errors showed the following reasons for erroneous hierarchy extraction:

1. Use of bold text: Some pages used bold text elements to highlight whole sections or just some blocks. This can screw up the whole result as the bold blocks might be identified as headlines.
2. Wrong enumeration: A surprisingly large amount of T&C pages contain errors in their enumerations. As the algorithm requires a strict sequence of numerations, this can lead to problems in the hierarchy extraction.
3. Violation of the assumption "Sections are not interrupted": The algorithms assume that there is no more content of a section after one of its subsections.
4. Use of tables: Whenever tables occurred on a page (often in the context of shipping costs), the algorithm separated each cell into its own block resulting in a large number of blocks with different styles. A high frequency of numbers occurring in the table worsened the results as the vast amount of detected enumeration patterns triggered further adjustments to the table.
5. Failed style extraction: In order to link the custom DOM tree structure to the Selenium tree, each DOM node is attached with its full XPath. As some pages render elements after a short time span, they may not be included in the parsed DOM tree. At the time the algorithm starts style extraction, new elements can render and tackle the validity of the XPath attached to the DOM node making the extracting of visual features impossible.

6 Conclusion

We introduced a new content extraction algorithm that performs better than existing solutions in its specific domain of T&C web pages. Since the algorithm is based on some domain-specific assumptions, it is unclear how successful it would operate on a generic web corpus. Further research in the field could answer this question. Initial small tests looked promising under the assumption that the content of the page is not interrupted. The style extraction, which is currently based on *Selenium* can be considered the performance bottleneck, as retrieving certain CSS properties takes a rather long time. One should look for a more efficient solution to extract the rendered style. In addition, several content extraction threads can op-

erate in parallel. The general functionality of the rule-based approach to hierarchy extraction could be demonstrated. The general idea of Manabe and Tajima (2015) was extended by an enumeration detection due to the frequent usage of enumerations to structure T&C pages. It is much more difficult to achieve similar success rates in hierarchy extraction as with content extraction, due to the many irregularities in visual representation. This is probably due to the fact that the operators of different online shops often want to highlight very different elements from the contract text visually. In cases where such outliers do not occur in the visual representation, hierarchy extraction yields good results.

Acknowledgements

The project was supported by funds of the Federal Ministry for the Environment, Nature Conservation, Nuclear Safety and Consumer Protection (BMUV) based on a decision of the Parliament of the Federal Republic of Germany via the Federal Office for Agriculture and Food (BLE) under the innovation support programme.

References

Yannis Bakos, Florencia Marotta-Wurgler, and David Trossen. 2014. Does anyone read the fine print? consumer attention to standard-form contracts. *The Journal of Legal Studies*, 43:1–35.

Adrien Barbaresi. 2019. Generic web content extraction with open-source software. In *Proceedings of the 15th Conference on Natural Language Processing, KONVENS 2019, Erlangen, Germany, October 9-11, 2019*.

Marco Baroni, Francis Chantree, Adam Kilgarriff, and Serge Sharoff. 2008. Cleaneval: A competition for cleaning web pages.

Daniel Braun. 2021. *Automatic Semantic Analysis, Legal Assessment, and Summarization of Standard Form Contracts*. Ph.D. thesis, Technical University of Munich.

Daniel Braun and Florian Matthes. 2020. Automatic detection of terms and conditions in german and english online shops. In *16th International Conference on Web Information Systems and Technologies, WEBIST 2020*. SciTePress.

Daniel Braun and Florian Matthes. 2021. NLP for consumer protection: Battling illegal clauses in German terms and conditions in online shopping. In *Proceedings of the 1st Workshop on NLP for Positive Impact*, pages 93–99, Online. Association for Computational Linguistics.

John Gibson, Ben Wellner, and Susan Lubar. 2007. Adaptive web-page content identification. pages 105–112.

Francisco Viveros Jiménez, Miguel A. Sánchez-Pérez, Helena Gómez-Adorno, J. Posadas-Durán, G. Sidorov, and Alexander Gelbukh. 2018. Improving the boilerpipe algorithm for boilerplate removal in news articles using html tree structure. *Computación y Sistemas*, 22.

Adam Kilgarriff. 2007. Last words: Googleology is bad science. *Computational Linguistics*, 33(1):147–151.

Christian Kohlschütter, Peter Fankhauser, and Wolfgang Nejdl. 2010. Boilerplate detection using shallow text features. pages 441–450.

Gaël Lejeune and Lichao Zhu. 2018. A new proposal for evaluating web page cleaning tools. *Computación y Sistemas*, 22.

Marco Lippi, Przemysław Pałka, Giuseppe Contissa, Francesca Lagioia, Hans-Wolfgang Micklitz, Giovanni Sartor, and Paolo Torroni. 2019. Claudette: an automated detector of potentially unfair clauses in online terms of service. *Artificial Intelligence and Law*, 27(2):117–139.

Marco Lui, Timothy Baldwin, and Nicta Vrl. 2021. Cross-domain feature selection for language identification.

Tomohiro Manabe and Keishi Tajima. 2015. Extracting logical hierarchical structure of html documents based on headings. *Proceedings of the VLDB Endowment*, 8:1606–1617.

Jeff Pasternack and Dan Roth. 2009. Extracting article text from the web with maximum subsequence segmentation. pages 971–980.

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

Hiroyuki Sano, Shun Shiramatsu, Tadachika Ozono, and Toramatsu Shintani. 2021. A web page segmentation method based on page layouts and title blocks.

Egon Stemle. 2009. The krdwrd annotation framework – gathering training data for sweeping web pages: the canola corpus.

A Appendix

A.1 Hierarchy Extraction Algorithm

Algorithm 1: Extract hierarchy based on headlines (recursive).

```
Input: List of blocks (blockList)
Result: Children of a Node
/* Determine headline style of
   current level and gather all
   headlines on this current level.
   */
headlineStyle ← getNextHeadlineStyle(blockList);
headlineList ← [];
for block in blockList do
    if block.style = headlineStyle then
        headlineList.append(block);
    end
end
/* Create children list for current
   node by adding the blocks
   associated to the current node
   and by extracting the lower
   level nodes.                    */
children ← [];
children.append(blockList[0 : headlineList[0].index];
for headline in headlineList do
    cChildren ←
      extractHierarchy(blockList[(headline.index +
      1) : headline.next.index];
    children.append(Node(headline, cChildren));
end
return children;
```

A.2 Hierarchy Extraction

The deviations of the hierarchy extraction algorithm from the expected results are determined by assigning the following scores to the extracted nodes:

- 0: each section with correct parent, correct content, and correct title
- 0.4: wrong parent
- 0.5: wrong content
- 0.1: wrong title

As different T&C pages contain different amounts of sections, the score is divided by the total amount of sections identified by the algorithm. An error score of 0 accounts for a perfect extraction.

The meaning of the brackets used in Tables 4 and 5 is the following:

$$x \in [a; b) \mid x \geq a \land x < b$$

Error Score	German	English
0	12	5
(0; 0.05]	12	4
(0.05; 0.1]	1	1
(0.1; 0.15]	2	2
(0.15; 0.2]	1	0
(0.2; 0.3]	1	0
(0.3; 0.5]	1	1
(0.5; 1]	0	0
Failed	0	6

Table 4: Distribution of error scores for the hierarchy extraction of the German and English requirements sample.

Error Score	German	English
0	8	9
(0; 0.05]	11	3
(0.05; 0.1]	1	2
(0.1; 0.15]	1	1
(0.15; 0.2]	1	2
(0.2; 0.3]	1	2
(0.3; 0.5]	5	1
(0.5; 1]	0	0
Failed	2	0

Table 5: Distribution of error scores for the hierarchy extraction of the German and English test sample.