Collecting textual data from WEB pages using R

Availability and implementation: The inspector software, the demonstration scripts (in R), and the datasets are available at https://github.com/bsLab/webscraping.

1.1 Introduction: Textual data in computational social science

Today much social interaction and interpersonal communication takes place on the Internet. This produces large amounts of both textual data published on the Internet and behavioral marks people leave when surfing the web. The social media gained extremely in importance in recent years and represents meanwhile a rich and indispensable data source for social research. This the more, the more such data is available only on the Internet. This raises the question of how to collect such data appropriately. In answering this question, the chapter focuses on collecting textual data for later content analysis.

1.1.1 Content analysis

Next to the more common structured forms of data obtained in surveys or official statistics, the scientific analysis of textual data, referred to as content analysis, is dating back to the 1960s (Krippendorf 2018). Social scientists take interest in the exploration of textual data because of its unique conveyance of human thoughts and behaviour and its high density of information. Prior to the rise of databases and the World Wide Web, the process of analysing written human communication relied heavily on the time-consuming collection and processing of the textual data itself. The scope of social science research incorporating content analysis was therefore often limited regarding the quantity of collected data. A consequence of the increased availability of digital technologies has been a shift in how human interactions and communication are documented and as such are progressively available to researchers (Gentzkow et. al. 2019). Data sources on the internet are as diverse as their communicational context, ranging from searchable databases designed specifically to convey information to their users to structured contexts like message boards and also to single websites of blogs of companies or private individuals. Textual data nowadays is used by social scientists for conducting both qualitative and quantitative content analysis and can be expected to further gain in importance. Current fields of application for political scientists for example are the use textual data obtained from official sources as well as social media to analyse political discourses, campaings and voter behaviour (Stier et. al. 2018, Yang/Kim 2017). Sociologists follow changes in public opinions (Flores 2017), (…)

1.2 Digital collection of Textual Data

The section provides a short sketch of basic ways of collecting text data (via API, scraping, crawling) and discusses their relative merits/drawbacks as well as legal and monetary aspects.

1.2.1 WEB Data

The World Wide Web is an enormous source of information consisting of services (using the Hyper Text Transfer Protocol HTTP) providing HTML documents. The HTML content format is designed for presenting information to humans, not computers. Therefore, automated information extraction from the WEB (aka., web scraping) have to address the gap between HTML structure and textual structure related to humans speech semantics.

The source of WEB-based data is related to WEB content and WEB services that can be categorised in:

- Static and dynamic content of WEB pages (HTML);
- Dialogue and discussion data bases (social blogs, chats);
- Digital Communication;
- Search engine data bases;
- Personalised data (statistics of user access of WEB pages and services);
- Video media (containing oral and written text) and audio data (containing oral text);

- Any meta information of media (WEB pages, Browser cookies, video, audio media).

1.2.2 Feature Selection in WEB Data

Feature selection in WEB data is the general process to extract relevant data from WEB data sources. If these WEB data sources are publicly visible WEB pages, then WEB scraping can be applied to get data from text documents with layout, discussed below (first class data). If data is hidden in data bases, e.g., message data bases of social media messanger services like Facebook or Twitter (secondary class data), then often a dedicated WEB Application Programming Interface (API) can be used. The general process and data flow in WEB data mining is shown in Fig. 1.

Features of a WEB page can be classified in:

- The server domain (address) hosting the WEB page;
- Textual content;
- Meta data;
- Link references (to other WEB pages);
- Layout of the WEB page;
- Placement of external and secondary content (e.g., advertisement);
- User and personal data (e.g., author information not contained in meta data of WEB page);
- Statistical data;
- Numerical data (e.g., contained in tables).

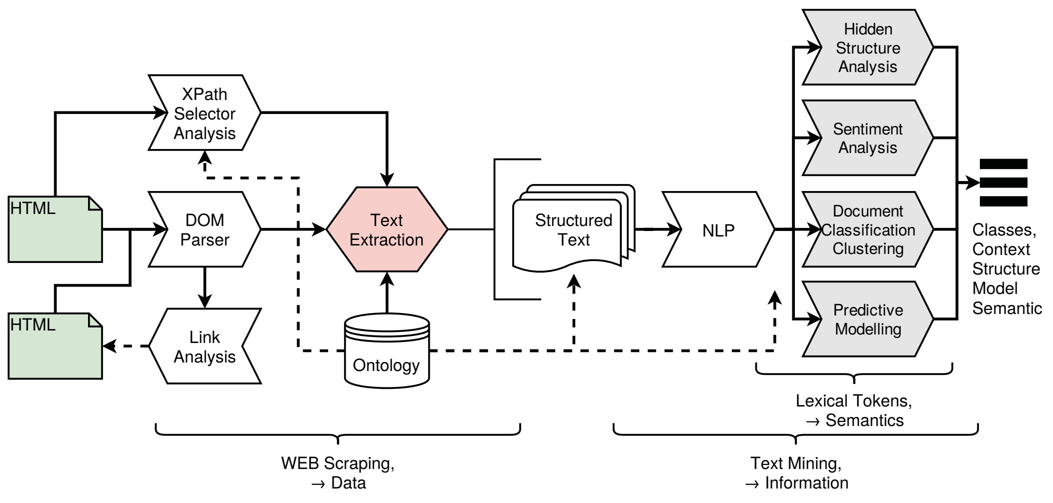

HTML
HTML
XPath Selector Analysis
DOM Parser
Link Analysis
Text Extraction
Ontology
Structured Text
NLP
Hidden Structure Analysis
Sentiment Analysis
Document Classification Clustering
Predictive Modelling
Classes, Context Structure Model Semantic
Lexical Tokens, → Semantics
WEB Scraping, → Data
Text Mining, → Information

Fig. 1. Principle functional and data flow diagram of the entire WEB mining task

1.2.3 *WEB Data Application Programming Interfaces (WEB API)*

A common way to collect WEB sourced data is a dedicated WEB API that is provided by the service hosters, e.g., a social media messanger service. WEB API enable filtered access to the underlying data base content, in contrast to WEB scraping, that get access to the meta-level of the data with a visual representation. WEB APIs pose a advantages and disadvantages compared with the WEB scraping process addressed in this chapter primarily:

- WEB API causes often monetary costs, and the amount to be paid determines the data filter (e.g., maximal number of messages that can be accessed, data quality, data variables);
- WEP API can change without any notice and anytime, mostly changing the filter function;
- WEB API can be biased for commercial reasons (i.e., a source of error in sentiment analysis);
- WEB API provides directly structured data based on an ontology or schema, getting structured data from WEB scraping is a challenge and

require user interaction;

- WEB API can be accessed by any client software, WEB scraping can require a WEB browser and user interaction.

In the following Section the WEB scraping basics are introduced.

1.2.4 WEB Scraping and Crawling

WEB scraping is widely known from search engine crawler robots and is the task to extract relevant data from WEB pages via generic HTTP requests with a automatically performed analysis of the HTML document mapping document elements on data structure.

1.3 HTML-Websites as a Data Source

The Hyper Text Markup Language (HTML) is a common and widely used standard for the representation of originally text content, structure, and layout, in recent years extended to the representation of multi-media content including audio and video (HTML5), too. HTML is a domain-specific sub-set of the generalised Extensible Markup Language (XML) model, and XML is a sub-set of the Standard Generalized Markup Language (SGML). XML is a standard for creating languages that meet XML criteria like structural organisations and strict conformity to schemas (a language ontology). HTML is defined by its own schema.

XML as well HTML consists of tagged elements (tag) that can be nested. Nested tag elements create a tree structure. A tag consists of a tag name, optional attributes, and content (if any). Content of a tag element can either be a list of other tag elements (children elements) or plain text (including so called entities, discussed later). XML as well as HMTL groups information in hierarchies. The elements in documents relate to each other in parent/child and sibling/sibling relationships. The element names and their structural relationships are defined in a schema.

The content of a tag element (children elements) are enclosed between a tag specifier and an anti-tag specifier shown below:

```
<tag attribute=value ..>
  ... children content ...
```

```
3: </tag>
```

Def. 1. XML/HTML tag element

The nesting of elements creates document trees composing the Document Object Model (DOM). Each node of the tree is a HTML tag element, children of a HTML element spawn sub-trees.

To understand how web scraping functions, a description of the basic composition of a website and its pages and how a reader can inspect this composition, using modern browsers and some helper utilities, is introduced. A website is commonly identified by an URL and organised in pages that are linked.

An HTML file is plain text with a specific character encoding (e.g., UTF8 or ASCII with national code pages associating characters to numerical code). HTML markup enables the definition of structure, layout, and style formats. HTML defines the positioning (layout) or WEG pages, their content, and theit visual styles. An HTML file consists of a header defining meta attributes, styles, and script code, discussed later. The body consists of a page structure tree defining the order, position, and content of visuals. Visual styling can be defined by style classes or by individual styling of page content.

Here is an example of different representations of a person name "John Doe" (with the first word as the surname, and the second word as the family name) using different formats and languages (CSV: Comma Separated Values, JSON: JavaScript Object Notation):

```
Text:   "John Doe"
CSV:    John, Doe
JSON:   { FirtName:'John', LastName:'Doe'}
XML:    <name><first>John</first><last>Doe</last></name>
HTML:   <html><head><title>Name</title></head><body>
        <p>John Doe</p>
        </body></html>
```

Ex. 1. Examples of data formats and representations

As well as the plain text representation of the person name the HTML format looses the semantical relationship (first and last name), too. This can be an important side effect and challenge in WEB data mining to extract structured content related to a more or less precisely defined ontology. An ontology defines semantical associations and structural relationships. The interpretation of XML or JSON content relies on ontologies giving tag names a meaning and context.

WEB browser and HTML support basically the following visuals (content boxes):

1. Text;
2. Images;
3. Audio- and Video media and players.

HTML supports the following structuring layouts for text visuals:

1. Section headings
2. Parapgraphs
3. Ordered lists (numbered)
4. Unordered lists (unnumbered)
5. Tables
6. Frames

The languages used to build web pages are basically HTML, CSS (cascading Style Sheets), and JavaScript, that are the technical targets of web scraping. Originally in the beginning of the WEB, a WEB page was organised as a paper with sections starting with a heading line and followed by content or sub-sections. For this purpose the heading tags exists (`h1`, `h2`, ..). Today, modern WEB sites are not organised in this one-dimensional layout style anymore and was superseded by a two- or three-dimensional dynamic layout.

1.3.1 From text content to structured data

Typically, text mining tasks are tasks that map plain text documents on structured and hierarchical feature records. There are structural and semantic features that have to be extracted from the raw text data. For example, a book is structured in chapters and sections. Chapters and sections can be considered as nested sub-documents.

Formally, the WEB data mining task *map* aims to map a text document T structured for visualisation and navigation and described by a document model schema $\mathbb{S}$ on a data record structure D related to information semantics that can be described by an ontology model $\mathbb{O}$:

$$\mathbf{map}(T) : T|\mathbb{S} \rightarrow D|\mathbb{O} \tag{1}$$

That means, the primary task to be solved is the derivation of a mapping relation function M (the model transformation function) that maps $\mathbb{S}$ on $\mathbb{O}$, commonly a domain and problem specific function that cannot be derived automatically.

$$M(T) : \mathbb{S} \rightarrow \mathbb{O} \tag{2}$$

The WEB data mining function consists of three chained functions:

1. A parser function P that structures the plain text document by arranging text elements in a abstract text tree structure (*ATT*) in compliance to the document schema $\mathbb{S}$ (e.g., HTML);
2. A compiler function C that transforms the abstract text tree *ATT* to an abstract data structure format, commonly a data tree *ADT*, in compliance to the ontology $\mathbb{O}$;
3. A data formatter function F that transforms the abstract data tree *ADT* in the desired output data format $\mathbb{Y}$, e.g. JSON

$$\mathbf{map}(T) = F_{\mathbb{Y}}(C_{\mathbb{O}}(P_{\mathbb{S}}(T))) \tag{3}$$

The data record structure D consists of a set of attributes a_i. i.e, D={a_i}, containing nested records and lists of elements. The document schema and the target data information ontology are typically lowly correlated.

In the following a short example shows the large gap between the document schema (here consisting of formatting elements like section headings or paragraphs) and the target data type (a bibliography data structure) as part of an ontology "Publication".

```
𝕊 := head, body
  head := styles
  styles := highlight | color | emphasis | footnote
  body := heading | paragraph | list | table
⇒
𝕆 := Novel | Science | News | Advertisment
  Science :=
  { title:text,
    affiliation:text,
    homepage: text,
    main-author: boolean,
    abstract:text,
    doi : text}
```

Ex. 2. Comparison of a document schema describing the elements of a text document (e.g., a HTML document) and a information ontology describing a publication data base with different classes of papers.

1.3.2 HTML Elements

There are a few HTML elements that are associated with language semantics summarized in the following table 1.

HTML Tag	**Semantic**
`<title>`	The title of the page (header section)
`<meta>`	The meta tag defines variables (header section), e.g., keywords, character set
`<h1>`	Heading and start of a main section
`<h2>`, `<h3>`, ..	Deeper nested document sections
`<dl><dt><dd>`	A definition list assigning a descriptive text to headline or topic (keyword)
`<table><th><tr><td>`	A structured table organising text in rows and columns. If there is a separate header row the columns can be assigned to attribute names creating a record table.
`<a href=URL>`	The anchor tag provides (named) links to other documents or parts of the current page
`<link href=URL>`	External content is referenced and included in the current document

HTML Tag	Semantic
`<iframe src=URL>`	External content is referenced and included in the current document in a separate and encapsulated frame

Tab. 1. HTML tag elements with structural or semantic associations

Further tag elements like lists (`<ul><li>`, `<ol><li>`) provide only weak semantic and structural correlation of the WEB page with the extracted structured data.

Beside the tag element type, a tag element can be attributed by formatting styles (commonly not relevant for text mining) and style classes. Style classes can be used as a semantic mapper, although the style class name (not its associated style formats) is only a opportunistic indicator for document structure and element semantics. Style class to semantic or structure mapping have to be constructed for each new WEB site and in the worst case for every WEB page to be scanned.

Content can be hidden and only displayed on specific events (e.g., by click events performed by the user).

1.3.3 Feature Selection: Selecting Content

The main challenge of extracting structured data from HTML documents is the selection of relevant elements and their mapping on feature types and hierarchies. Tree structured documents are typically parsed and analysed using paths of nodes along DOM node elements until nodes containing the relevant data are found.

Two different main data classes have to be distinguished:

1. Table and numerical data (organised in rows and columns with a mostly regular structure);
2. Text data (organised in sections).

Typically content is extracted by using descriptive methods, i.e., by providing patterns to select parts of the document. Common descriptive methods with pattern matching are regular expressions and Xpath expressions explained in the following sub-section. Dynamic content (e.g., a sub-range of lists with varying number of elements) is difficult to handle with patterns and pattern matching. State-based and problem-specific parsers that search desired content iteratively can be a better choice to extract data from WEB pages. Pattern matching can mostly only applied to context-free problems (i.e., the absolute position of elements is not relevant). Context-dependent search requires state-based parser programmed for each specific search problem.

Regular Expressions

Regular expressions are patterns used to match parts of text (e.g., sentences, keywords, numbers ..) associated with character combinations in strings. Regular expression can solve a broad range of text pattern matching and text extraction processes. Regular expressions are specially encoded text strings.

A regular expression consists of terminals (i.e., characters, character ranges, decimal numbers) that can only match one character or a static text string, or literals describing placeholders for dynamic content, shown in Table 2.

Expression	**Description**
`.`	Matches any character
`\s`	Matches a space or tabulator character
`\d`	Matches a digit character
?ε	Matches the expression one or zero times
[α-β]	Matches a character in the range α to β
[ˆα,..]	Matches a character without the ones following
ε*	Matches an expression ε zero, one, or more times
ε+	Matches an expression ε one, two, or more times
(ε)	Defines a capturing group that can be referenced
ε|ε	Alternation expression

Tab. 2. Regular expression syntax elements

Regular expression cannot be used primarily for content extraction from HTML pages. But they can be used in a second step to extract or find relevant textual information from previously extracted text (that can contain a lot of non relevant information). Regular expressions must be used in conjunction with text function, e.g., a match or replace function, shown in principle in Ex. 3

```
text="The brown fox hunts white eggs"
text.match(/brown/)? ⇒ true
text.match(/white/)? ⇒ true
text.match(/brown[ ]+([ˆ ]+)/)? ⇒ ["fox"]
text.replace(/fox|egg/g,"animal")
  ⇒ "the brown animal hunts white animals"
```

Ex. 3. Regular expressions applied to a sentence

XPath Expressions

XPath is a language for matching paths and patterns in tree-structured documents, i.e., XML or HTML documents. The XPath patterns address structure as well as data values. An XPath query uses a search pattern to return a list of matching nodes. An XPath is just a string descriptor composed of expressions, shown in Table 3.

Expression	**Description**
nodename	Selects all nodes with the name "nodename"
`/`	Selects from the root node
`//`	Selects nodes in the document from the current node that match the selection no matter where they are
`.`	Selects the current node
	Selects the parent of the current node
`@`	Selects attributes
`*`	Any matching element
[*n*]	Selects n-the element of a list
`last()`	Selects last element of a list

Tab. 3. XPath syntax elements

Referencing content of a HTML WEB page with XPath selectors is a non-trivial task. HTML tag elements can be referenced by:

1. Their (unique) identity (*id*) attribute (one one hand producing the best matching, but sequences or lists of elements require exact details of all single element identity names);
2. Their style classes (if there is any meaningful relation between style names and content structure);
3. Their relative or absolute position in element lists;
4. Their relative or absolute position in the DOM tree.

In the following Ex. 4 some typical XPath expressions are shown. They will be used in the next section dealing with text extraction of a simple WEB page.

```
//*[contains(@class,"author")]
//span
//*[contains(@class,"blog") and
    (((count(preceding-sibling::*) + 1) = 2) and
    parent::*)]
  //*[contains(@class,"date")] |
  //*[contains(@class,"author")]
  //*[(((count(preceding-sibling::*) + 1) = 1) and parent::*)]
  //*[contains(@class,"date")]
```

Ex. 4. Examples of XPath expressions

Mapping the DOM content structure on the target data structure is a challenge. HTML element classes describe commonly formatting styles, not semantics and data structure. An extensive commonly hand crafted analysis of the WEB pages is mandatory.

1.3.4 Automatised Extraction and Machine Learning

The derivation of suitable selection patterns, e.g., using XPath expressions, can be a time-consuming and error prone task, shown in the practical Sec. 1.4. After deriving a suitable set of pattern expressions, these pattern expressions must be associated with data mapping functions that map the raw extracted content to data structure. This task typically requires extensive scripting and programming. Finally, XPath expressions are context- and state-less and pose some limitations.

In [GRA13], the extraction pattern and the data structure mapping are merged by an advanced but minimalstic wrapping language OXPath. OXPath extends XPath expressions with more advanced conditional expressions reducing under- and over-fitting of content extraction. OXPath handles patterns and actions on patterns together. Finally, OXPath enable multi-page navigation. Most WEB pages are split over multiple pages.

Beside the content extraction, validation is required to ensure a high data quality of the extracted content (avoiding under- and over-mapping). In [THO12], a complete WEB scraping framework is introduced decomposing the WEB scraping process in selection, validation, and reinduction functions with a high degree of reusability.

But all presented approaches require a significant amount of user and expert interaction. Automated data extraction from WEB content can utilize Machine Learning to extract structure from flat and noisy content. Using such methodology the WEB page content is extracted as a more or less contiguous block of text (linear text), e.g., discussed by [ZHO14]. An either supervised or unsupervised trained model *M* performs a block clustering of the linear text with a final classification and selection of the text blocks with respect to the target data structure.

1.3.5 Dynamic versa Static Content

Originally in the beginning of the WEB the pages were static text files stored on a file server. Each WEB request got a exact copy of the stored files. In the last decades dynamic creation of HTML pages by the server raises significantly, and today most WEB pages are created dynamically by using SQL data bases storing the content (data) and PHP frameworks creating the HTML document format and styling. This content is called server-based content.

By introducing the JavaScript programming language that can be embedded in any HTML document and executed by the browser another class of documents were created: Client-based dynamic content. JavaScript was primarily used to create interactive documents with dynamic layouts and visibility of content. JavaScript code can modify the DOM and therefore can load and present data from remote sources at viewing time. This capability makes it more difficult to get and extract desired content from a WEB page. Moreover, only requesting the WEB page is not sufficient. The code must be executed prior to content extraction.

Finally, the dynamically code-created content of the WEB page can depend on:

1. The WEB browser identification (the so called user-agent identification, e.g., 'Mozilla/5.0 (X11; SunOS i86pc; rv:45.0) Gecko/20100101 Firefox/45.0');
2. The location (Internet network address or geo-spatial location);
3. Cookies stored by the code of the WEB page (or by code of another WEB page from the same domain);
4. User (log-in) credentials;
5. User interactions (multi-modal: Mouse, Keyboard, ..).

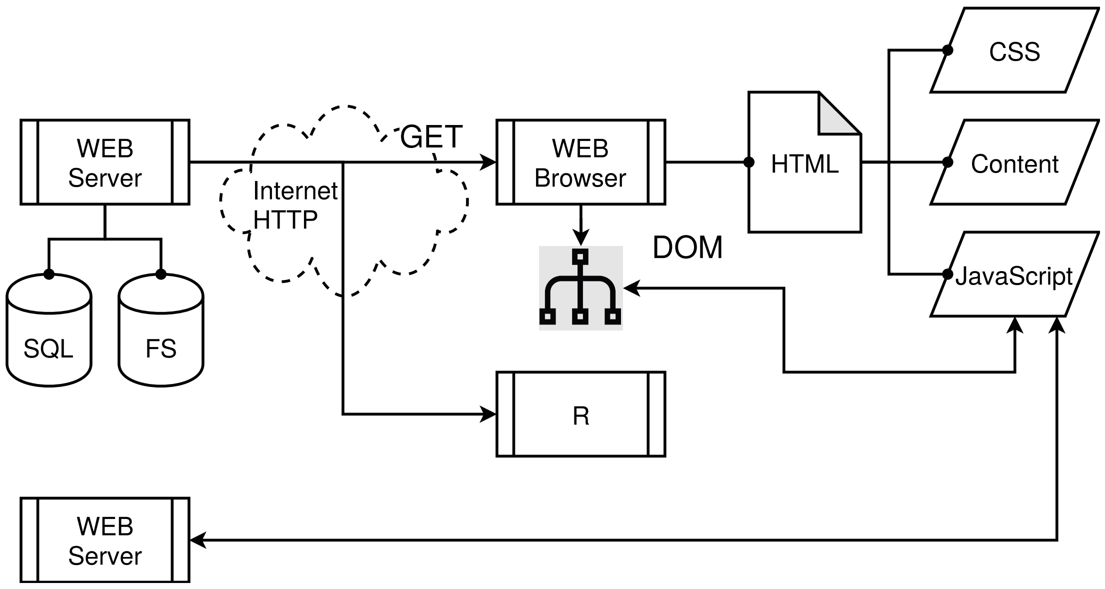


Fig. 2. A WEB server provides HTML files. An HTML file consists of style defintions, document content, and JavaScript code. The WEB server can create the HTML sources from a static file (system) or dynamically by using a data base and HTML templates (typically composed by PHP code). On client-side, JavaScript can modify the document content and can load external content or data, too, e.g., from a different WEB server.

1.4 Using R for data collection: How to parse web pages with a simple example

In this section two practical examples of WEB data mining is demonstrated using the widely used statistical and data analysis software *R*. This examples show the programmatical implementation of the three mining functions introduced formally in Sec. 1.3.1. The first example shows the extraction of table data (primarily numerical data), and the second example performs extraction of structured text content.

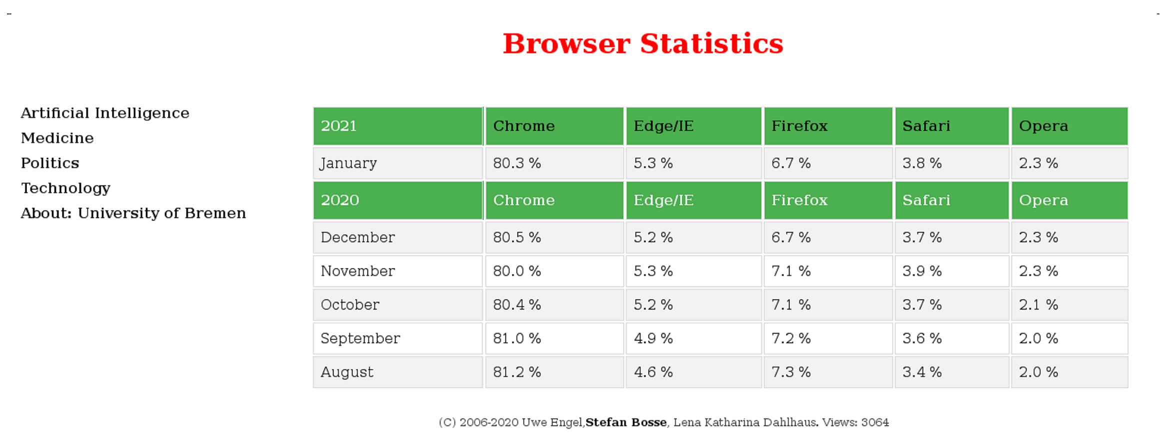
1.4.1 Demonstration

1.4.2 The visual appearance

The following Fig. 3 shows an example page containing table structured data, and Fig. 4 shows an example page containing structured text in form of news snippet blocks arranged in a linear list. Both demonstration pages embed the relevant content in a navigation frame, typical for modern WEB site layouts.

The basis DOM structure of the first page is shown in Ex. 5. A challenge is the mixed layout of the table with header and data rows and two stacked sub-tables (for year 2021 and 2020).

The second demonstration page shows a list of news blog entries. Each news blog snippet contains a date, an author name, a headline, a keyword list, and a shortened preview of the news text. The preview and the news text can be switched in place by user interaction (clicking on the preview text field). The keyword list opens only by user interaction, too (hovering over the keyword field).



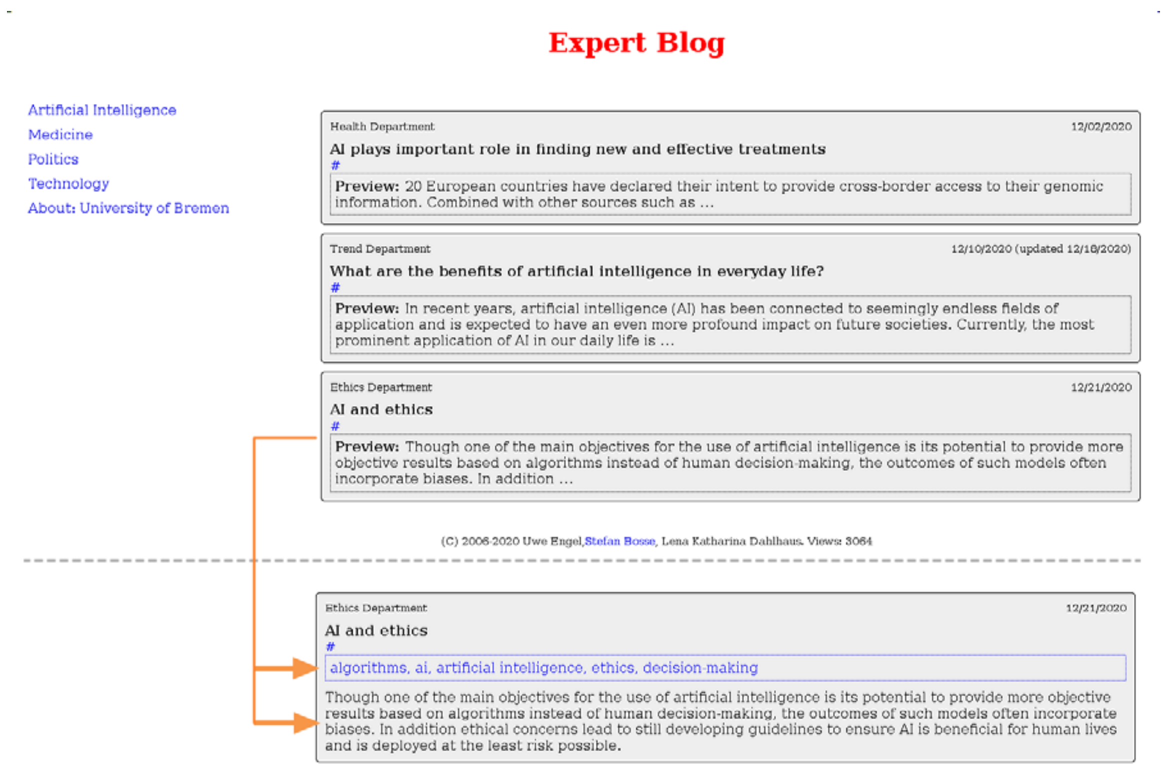
Browser Statistics

Artificial Intelligence
Medicine
Politics
Technology
About: University of Bremen

2021	Chrome	Edge/IE	Firefox	Safari	Opera
January	80.3 %	5.3 %	6.7 %	3.8 %	2.3 %
2020	Chrome	Edge/IE	Firefox	Safari	Opera
December	80.5 %	5.2 %	6.7 %	3.7 %	2.3 %
November	80.0 %	5.3 %	7.1 %	3.9 %	2.3 %
October	80.4 %	5.2 %	7.1 %	3.7 %	2.1 %
September	81.0 %	4.9 %	7.2 %	3.6 %	2.0 %
August	81.2 %	4.6 %	7.3 %	3.4 %	2.0 %

(C) 2006-2020 Uwe Engel,**Stefan Bosse**, Lena Katharina Dahlhaus. Views: 3064

Fig. 3. Table Data Mining Demo 1: The visually and rendered appearance of the example WEB page shown by a typical WEB browser



Expert Blog

Artificial Intelligence
Medicine
Politics
Technology
About: University of Bremen

Health Department 12/02/2020
AI plays important role in finding new and effective treatments
#
Preview: 20 European countries have declared their intent to provide cross-border access to their genomic information. Combined with other sources such as …

Trend Department 12/10/2020 (updated 12/16/2020)
What are the benefits of artificial intelligence in everyday life?
#
Preview: In recent years, artificial intelligence (AI) has been connected to seemingly endless fields of application and is expected to have an even more profound impact on future societies. Currently, the most prominent application of AI in our daily life is …

Ethics Department 12/21/2020
AI and ethics
#
Preview: Though one of the main objectives for the use of artificial intelligence is its potential to provide more objective results based on algorithms instead of human decision-making, the outcomes of such models often incorporate biases. In addition …

(C) 2006-2020 Uwe Engel,Stefan Bosse, Lena Katharina Dahlhaus. Views: 3064

Ethics Department 12/21/2020
AI and ethics
#
algorithms, ai, artificial intelligence, ethics, decision-making

Though one of the main objectives for the use of artificial intelligence is its potential to provide more objective results based on algorithms instead of human decision-making, the outcomes of such models often incorporate biases. In addition ethical concerns lead to still developing guidelines to ensure AI is beneficial for human lives and is deployed at the least risk possible.

Fig. 4. Text Mining Demo 2: (Top) The visually and rendered appearance of the example WEB page shown by a typical WEB browser (Bottom) Expanded hidden content

1.4.3 The Document Object Model (DOM)

The basic structure of the demonstration DOM is shown in Ex. 5 (showing irrelevant content) and 6 and 7 (showing relevant content containing the target data to be extracted).

```
html
├─head
├──style
├─body
├──script
├──h1#text
├──div(flex-container)
├───div(column-left)
├────ul
├─────li
├──────a(href="demo02.html")#text
├─────li
├──────a(href="demo03.html")#text
...
```

Ex. 5. Start of DOM tree of both demonstration pages (up to here no relevant content)

```
...
├───div(column-right)
├────table
├─────tbody
├──────tr
├───────th#text
├───────th#text
...
├──────tr
├───────td#text
├───────td#text
...
├──────tr
├───────th#text
├───────th#text
...
├──────tr
├───────td#text
├───────td#text
```

Ex. 6. First content relevant part of DOM tree of the demonstration page 1 containing a data table

```
...
├────div(column-right)
├─────div(blog)
├───────span(author)#text
├───────span(headline)#text
├───────span(tags-button)#text
├───────span(tags)#text
├───────span(id=preview1)#text
├───────span(id=text1)#text
├─────div(blog)
├───────span(author)#text
├───────span(headline)#text
├───────span(tags-button)#text
├───────span(tags)#text
├───────span(id=preview2)#text
├───────span(id=text2)#text
...
```

Ex. 7. First content relevant part of DOM tree of the demonstration page 2 containing a news list

The target data must be extracted from HTML *div* and *span* tag elements. The relevant data structure mapping can be achieved here by unique style classes (e.g., `author`, `headline`). This is commonly not the case and ambiguous data content types must be resolved either by HTML tag positions (relative and absolute), by an automatic posterior text analysis, or by hand.

1.4.4 R Preparation

The following code snippet Code 1 shows a basic *R* set-up loading required libraries and finally parsing the DOM tree of a WEB page. The libraries must be previously installed. The central library is *rvest*. The installation of the *rvest* packages via the `install.packages("rvest")` command has a lot of dependencies that requires attention and in particular the installation of additional system packages (e.g., development libraries with support for SSL, XML, HTTP to access and process HTML pages).

The HTML pages can be loaded via remote HTTP communication or from files stored previously in the local file system (e.g., by using a generic

WEB browser). Accessing remote WEB pages via R and HTTP is performed without providing user credentials and a user-agent. This anonymous request can be rejected by the remote WEB server.

```
# General-purpose data wrangling
library(tidyverse)
# Parsing of HTML/XML files
library(rvest)
# String manipulation
library(stringr)
# Verbose regular expressions
library(rebus)
# Eases DateTime manipulation
library(lubridate)
# JSON formatting
library(jsonlite)
```

Code 1. R set-up for WEB scraping

1.4.5 R Data Extraction

```
# Load and parse DOM of HTML file
html <- read_html('demo01.html')
rows <- html %>% html_nodes(xpath='//tr')
tables <- c()
table  <- c()

for (row in rows) {
  th <- row %>% html_nodes('th')
  td <- row %>% html_nodes('td')
  if (!(th %>% rlang::is_empty())) {
    # Start of a new table
    if (!( table %>% is_empty)) {
      tables <- append(tables,list(table))
    }
    table <- c();
    table <- append(table,list(th %>% html_text()));
  }
  if (!(td %>% rlang::is_empty())) {
    # Append row to current table
    table <- append(table,list(td %>% html_text()));
  }
}
```

```
# Append last table to tables list
if (!( table %>% is_empty)) {
  tables <- append(tables,list(table))
}
# Format tables in JSON format
json <- toJSON(tables)
```

Code 2. Demo 1, Table extraction: Parse the HTML document and extract content finally coded in JSON format

```
# Load and parse DOM of HTML file
html <- read_html('demo02.html')
# Selecting headline nodes
headlines <-
  html %>% html_nodes(xpath='//*[contains(@class,"headline")]')
       %>% html_text()
# Selecting authors/source nodes
sources <-
  html %>% html_nodes(xpath='//*[contains(@class,"author")]')
       %>% html_text()
# Selecting date nodes
dates <-
  html %>% html_nodes(xpath='//*[contains(@class,"date")]')
       %>% html_text()
# Create a compound data structure
data <- c();
for (i in 1:length(headline)) {
  data <- append(data, list(list(headline=headlines[i],
                             source=sources[i], date=dates[i])))
}
# Format mining results in JSON format
json <- toJSON(data);
```

Code 3. Demo 2, Text extraction: Parse the HTML document and extract content finally coded in JSON format

The following tasks are to be performed:

1. Analyse the WEB page for relevant content and create XPath selectors by visual inspection by using a WEB browser to extract the content from the WEB page;
2. Parse and analyse WEB page for external links, store all WEB links in a list;

3. Demo 1: Extract all tables as a list (array);
4. Demo 2: Extract all news blog snippets as a record list (array);
   - Each record should contain information about the author, date of publication, the headline, and the preview text;
5. Demo 2: Analyse linked WEB content contained in the news snippet;
6. Create a content tree with all the referenced sub documents (both demonstrations).
7. Create a JSON data object from the extracted content.

Program code 2 and 3 produce a JSON object with the following type signature:

```
type table = table []
type table = (number|string) [][]
type news = { headline: string [],
              source: string [],
              date: string []} []
```

The date attribute of each entry has to be further processed to create a uniform date format. Some date entries contain auxilliary text like "updated on".

The output from demo #1 is:

```
[[["2021","Chrome","Edge/IE","Firefox","Safari","Opera"],
  ["January","80.3 %","5.3 %","6.7 %","3.8 %","2.3 %"]],
 [["2020","Chrome","Edge/IE","Firefox","Safari","Opera"],
  ["December","80.5 %","5.2 %","6.7 %","3.7 %","2.3 %"],
  ["November","80.0 %","5.3 %","7.1 %","3.9 %","2.3 %"],
  ["October","80.4 %","5.2 %","7.1 %","3.7 %","2.1 %"],
  ["September","81.0 %","4.9 %","7.2 %","3.6 %","2.0 %"],
  ["August","81.2 %","4.6 %","7.3 %","3.4 %","2.0 %"]]]
```

The output from demo #2 is:

```
[{"headline":["Solidarity with Members of Boğaziçi University"],
  "source":["Health Department"],
  "date":["12/02/2020"]},
 {"headline":["Arctic Climate Change: From Greenhouse to Icehouse "],
  "source":["Trend Department"],
  "date":["12/10/2020 (updated 12/18/2020)"]},
 {"headline":["Workshop: Cognitive Architectures for Robots"],
  "source":["Ethics Department"],
  "date":["12/21/2020"]}]
```

More details using *R* and *rvest* performing WEB scraping and data mining can be found in [MUN15].

1.4.6 XPath Selection

XPath content filters have to be defined to extract parts of the HTML DOM tree that contain the desired data content. The XPath selectors provide automatic filtering of HTML element nodes. They are not able to extract the text content (except in some simple cases) and perform the mapping to the target data structure.

XPath selectors have to be created by visual inspection of the WEB page(s) under test. A naive way to specify an XPath selector is inspecting the DOM tree by using the DOM inspector tool contained in any modern WEB browser. But this can be a highly complex and time consuming process considering modern blown-up WEB pages created by client- and server-side code.

Another more efficient and reliable way is using a dedicated XPath inspector written in JavaScript and executed in the target WEB page context. By side loading injection and content framing it is possible to analyse arbitrary WEB pages by using a WEB browser and the inspector framework contained in a (locally stored) HTML wrapper document (named `frame.html`). The HTML wrapper document loads the target WEB page in a HTML frame and injects the inspector software.

Due to cross-origin access policy restrictions the target WEB page must be downloaded first (e.g., by using the WEB browser itself).

After the inspector was loaded in the target page context, HTML DOM elements can be selected visually by clicking on the elements. An appropriate HTML and XPath selector is created that can be used in *R* and *rvest*.

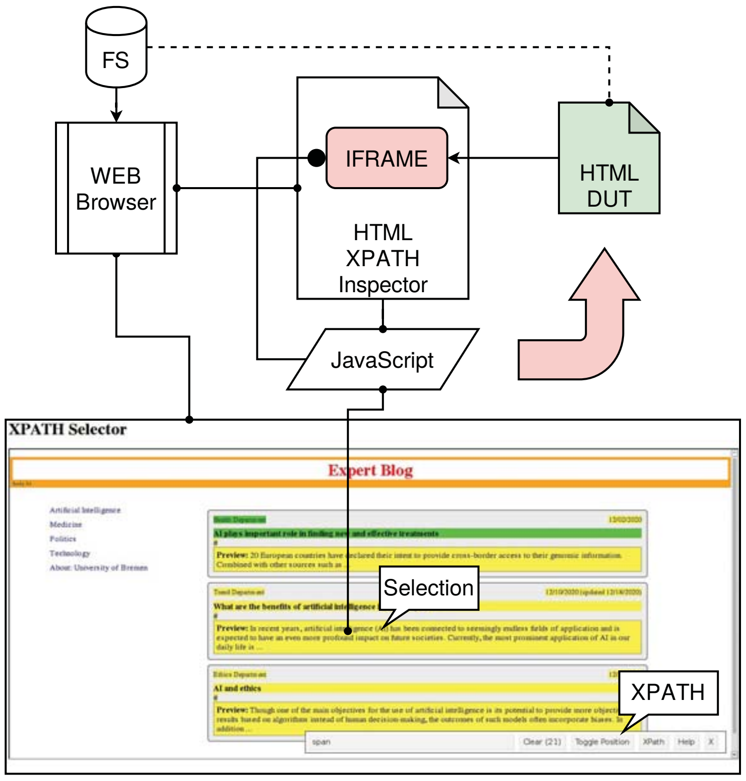


Fig. 5. XPath Inspector injecting diagnostics and highlighting code in the test page (DUT)

1.5 The Real World and Challenges

In the previous section WEB data mining was demonstrated with a simple static WEB page. Accessing data from today real-world WEB pages has to address different difficulties:

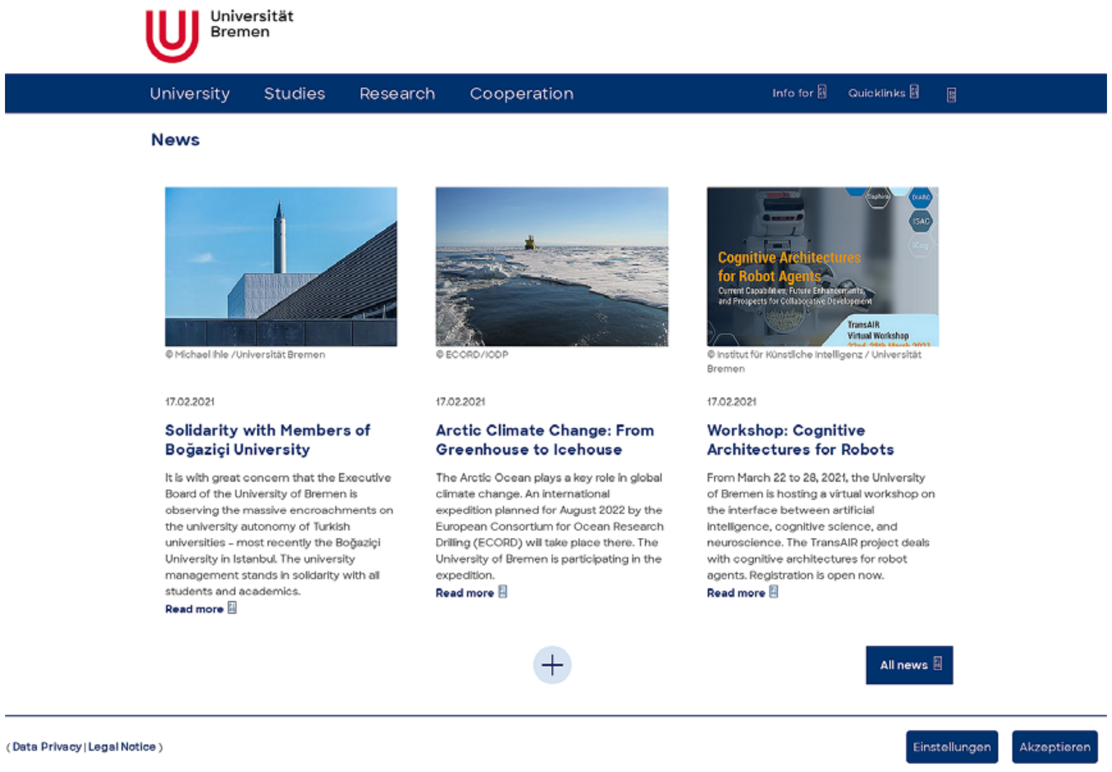
1. The WEB server delivers browser-specific content. A generic HTTP request, i.e., by *R*, provides no user-agent information, i.e., an identification of the WEB browser that requests the page. The result of the request is unpredictable and ranges from a default WEB page notification (WEB browser not supported) to the desired content. Furthermore, modern WEB sites support mobile devices with limited user interaction capabilities and targeting small screen size layouts. Without proper user-agent information the WEB server can return a

limited version of the WEB page with reduced content, too.

2. The access of a WEB page requires user authentication (access-restricted content).
3. The content is built dynamically by client-side JavaScript code
4. The content is blocked due the usage of cookie storage violating German/European privacy restrictions and require user interaction to allow exceptions. The content blocking is mostly limited to the visual representation and do not affect the DOM of the HTML page, but there is no guarantee that the desired data content is available before user interaction.
5. The data content structure is not related with unique HTML tag elements and styles (style classes), i.e., there is no bijective mapping function *map*: $T \Leftrightarrow D$. This is a typical problem in dynamically created content, e.g., by using data-base driven server frameworks like TYPO3.
6. The structural HTML elements like headings `h1`, `h2`, .. , are used in a misleading context with a high degree of ambiguity. The following real-world examples demonstrates this issue. Heading elements are often used to ease recognition of relevant highlighted content for search engine crawlers (like news headlines).

1.5.1 Complex Example WEB Page

To demonstrate the challenges of WEB scraping of complex dynamically server-created WEB pages, a final example with the widely used TYPO3 content management system (CMS) created WEB page from the University of Bremen is shown in Fig. 6 and analysed. This WEB page shows complex and nested content layout. In TYPO3, like any other CMS, content management is completely separated from content layout and styles. This leads to a significant decorrelation of HTML DOM structure and styles from content and data structure (loss of semantic relation).


Universität
Bremen

University Studies Research Cooperation Info for Quicklinks

News

© Michael Ihle /Universität Bremen

17.02.2021

Solidarity with Members of Boğaziçi University

It is with great concern that the Executive Board of the University of Bremen is observing the massive encroachments on the university autonomy of Turkish universities – most recently the Boğaziçi University in Istanbul. The university management stands in solidarity with all students and academics.

Read more

© ECORD/IODP

17.02.2021

Arctic Climate Change: From Greenhouse to Icehouse

The Arctic Ocean plays a key role in global climate change. An international expedition planned for August 2022 by the European Consortium for Ocean Research Drilling (ECORD) will take place there. The University of Bremen is participating in the expedition.

Read more

© Institut für Künstliche Intelligenz / Universität Bremen

17.02.2021

Workshop: Cognitive Architectures for Robots

From March 22 to 28, 2021, the University of Bremen is hosting a virtual workshop on the interface between artificial intelligence, cognitive science, and neuroscience. The TransAIR project deals with cognitive architectures for robot agents. Registration is open now.

Read more

Cognitive Architectures for Robot Agents
Current Capabilities, Future Enhancements, and Prospects for Collaborative Development
TransAIR
Virtual Workshop
All news
(Data Privacy | Legal Notice)
Einstellungen
Akzeptieren

Fig. 6. The real-world test page from the University of Bremen (home page) accessed at 22.2.2021 (shown is a part of the viewport of the page content with relevant news content snippets to be extracted)

The data mining task consists of the extraction of news entries with date, headline, and short text message fields. A first overview analysis of the WEB page shows that:

1. The date fields can be accessed with a `//time` XPath expression;
2. The news headlines can be accessed with a `//h3` XPath expression;
3. The news short text message can be accessed with `//p` XPath expression.

Applying these simple filters to the content extraction delivers 20 date entries, 35 headline entries, and 48 message entries. The filter is insufficient to extract the desired content (only 20 news messages, there are visible, the rest is hidden by code) correctly. The list of the entire news blocks (containing date, headline, and message child elements) can be extracted by using an more specific XPath `//*[contains(@class,"news-link")]` referencing division elements with the specific style class attribute. This list is finally divided in the sub-parts with additional XPath selectors applied to each block list.

This example shows the difficulties to extract the data from complex WEB pages. typically, this process cannot be automatised and requires a lot of hand-crafted work.

1.5.2 Data Quality

The previous demonstration showed different query results depending on the accuracy of the query patterns. This uncertainty can reduce the data quality with respect to completeness and wrongly extracted and classified data.

Further issues can influence the data quality and the data extraction process, that must be addressed by the WEB scraping software:

1. Blocking of content based on the Internet address, requiring proxy server support;
2. Blocking of content due to human-bot tests (CAPTCHA), requiring support for CAPTCHA based scraping and proxy supports (CAPTCHA appearance can relate to Internet address, too);
3. Point-and-click user interfaces for identifying content is error prone and can result in under- or over-fitted content matching, requiring validation tools;
4. Irregular text formats, e.g., different date and time formats mixed with auxiliary text; reduces text-data mapping quality;
5. Different views of a page based on WEB browser (user-agent) identification and personalisation of WEB page can invalidate extraction patterns or change the data. Most WEB services provide a mobile version with a different layout and reduced content. Practically, unknown or older Browsers are considered as mobile software, obfuscating the data mining process.

To summarize, the data quality of data derived from WEB scraping technologies can vary significantly and requires validation and quality evaluation by expert interaction.

1.6 References

[FLO17] Flores, R. D. (2017). Do anti-immigrant laws shape public sentiment? A study of Arizona's SB 1070 using Twitter data. American Journal of Sociology, 123(2), 333-384.

[GEN19] Gentzkow, M., Kelly, B., & Taddy, M. (2019). Text as data. Journal of Economic Literature, 57(3), 535-74.

[GRA13] G. Grasso, T. Furche, and C. Schallhart, *Effective Web Scraping with OXPath*, in WWW 2013 Companion, May 13-17, 2013, Rio de Janeiro, Brazil., 2013.

[KRI18] Krippendorff, K. (2018). Content analysis: An introduction to its methodology. Sage publications.

[STI18] Stier, S., Bleier, A., Lietz, H., & Strohmaier, M. (2018). Election campaigning on social media: Politicians, audiences, and the mediation of political communication on Facebook and Twitter. Political communication, 35(1), 50-74.

[THO12] J. G. Thomsen, E. Ernst, C. Brabrand, and M. Schwartzbach, *WebSelF : A eb Scraping Framework*, in ICWE 2012, LNCS 7387, 2012, pp. 347-361.

[YAN17] Yang, J., & Kim, Y. M. (2017). Equalization or normalization? Voter–candidate engagement on Twitter in the 2010 US midterm elections. Journal of Information Technology & Politics, 14(3), 232-247

[ZHO14] Zhou, Ziyan, and Muntasir Mashuq. *Web content extraction through machine learning*. Standford University (2014): 1-5.

1.7 Further Readings

[MUN15] S. Munzert, C. Rubba, P. Meißner, and D. Nyhuis, Automated Data Collection with R Web Scraping and Text Mining A Practical Guide to. Wiley, 2015.