Automatic Generic Web Information Extraction at Scale

An attempt to bring some structure to the web data mess

Mahmoud Aljabary

Automatic Generic Web Information Extraction at Scale

Master Thesis

Computer Science, Data Science and Technology
University of Twente. Enschede, The Netherlands

An attempt to bring some structure to the web data mess.

Author
Mahmoud Aljabary

Supervisors
Dr. Ir. Maurice van Keulen
Dr. Ir. Marten van Sinderen

March 2021

Abstract

The internet is growing at a rapid speed, as well as the need for extracting valuable information from the web. Web data is messy and disconnected, which poses a challenge for information extraction research. Current extraction methods are limited to a specific website schema, require manual work, and hard to scale. In this thesis, we propose a novel component-based design method to solve these challenges in a generic and automatic way. The global design consists of 1. a relevancy filter (binary classifier) to clean out irrelevant websites. 2. a feature extraction component to extract useful features from the relevant websites, including XPath. 3. an XPath-based clustering component to group similar web page elements into clusters based on Levenshtein distance. 4. a knowledge-based entity recognition component to link clusters with their corresponding entities. The last component remains for future work. Our experiments show huge potential in this generic approach to structure and extract web data at scale without the need for pre-defined website schemas. More work is needed in future experiments to link entities with their attributes and explore untapped candidate features.

Keywords: Web information extraction, Machine learning, Web data wrapping, Wrapper induction, Relevancy filter, Clustering, Entity recognition, XPath, DOM, HTML, JSON, Levenshtein distance, Design science.

Table of Contents

List of Tables

List of Figures

1

Introduction

The internet is full of web pages growing exponentially at a rapid rate. Web pages typically contain unstructured and semi-structured data in the form of natural language text, images, videos, tables, etc., that is valuable for nowadays business. This big data has led to open business opportunities and data-driven applications. Web data analytics and extraction can offer significant value to companies in many ways, including higher revenues, better customer satisfaction, and more efficient operations. Many industries have used web data as a competitive advantage tool for different needs, including business contact information collecting from public business directories, brand and competitor web present monitoring, market analysis, collecting job postings from job portals, product information from webshops, and online products and pricing comparison. As a result of the business interest and practical applications, technologies for processing and extracting value from the web data have become an important study area, so as the need for automated methods to collect and extract relevant information from the web, accurately and expeditiously.

Web Information Extraction is the process of extracting information from web pages data, which usually comes in a semi-structured format and converting that into structured information. This process includes collecting web page data by downloading the actual content of the page. A typical web page is formatted in an HTML document and other web technologies such as JavaScript and (Cascading Style Sheets) CSS. The HTML document is a hierarchy of HTML elements that are normally represented as DOM (Document Object Model).

Current web information extraction methods which aim at extracting information from the web pages are limited to particular website HTML document structure, and therefore hard to scale on different websites. Such methods are specific to a template definition or schema structure. Suppose someone is interested in extracting web information from 1 million websites, normally a developer has to hard

code scripts to extract data form each website individually, which means 1 million scripts and countless hours. The same issue is found in other solutions that aim at allowing non-technical people to annotate the data of interest visually through a user interface. This is clearly infeasible at scale.

This thesis discusses different approaches found from the literature related to Web Information Extraction and propose a novel component-based method based on the established methodology of (Wieringa, 2014). The global design includes several components; each component has a local goal and contributes to the global design. The main contribution is an automatic generic approach that can extract information from web pages from any website regardless of its structure. The method takes a collection of web pages, filter them based on a relevancy filter classifier. Candidate features (including XPath) are then extracted from the relevant websites. A clustering component groups similar HTML sections into clusters given the extracted features. Finally, clusters are linked to entities using a knowledge-based approach. The ideal output is a structured key-value pairs information in a JSON format. The output can then be used to fulfill the need for structured information in many applications and smart services.

The rest of the thesis is organized as follows. The research background and motivation are presented in the following section. Web Information Extraction methods found from literature and related work are in Section 3. Section 4 includes the proposed method, a design science research, including the research requirements, goals, and existing challenges. The system design components for the automatic generic method are discussed in detail in Section 5, including relevancy filter, feature extraction, clustering, and entity recognition component. Finally, section 6 provides conclusions and answers to the research questions with a discussion and future research directions.

2

Background and Motivation

As described in the previous section, the need for structured information is significantly increasing. Traditional information extraction methods are limited to specific website schemas, which require manual work and thus not scalable. To overcome these limitations, we must investigate alternative generic and automatic web extraction methods that can work well at the required scale. The rest of this section describes important background concepts.

2.1 Semi-structured vs Structured Data

Web data typically comes in a semi-structured format, which is a form of data that contains some semantic structure and include noticeable uniform patterns. However, the schema is not pre-defined and open among different websites. (Lin et al., 2018) describe three characteristics of semi-structured data: 1. The data has a certain structure, but this structure is mixed with the content. This is also the case in web pages, where the content data is mixed with the HTML document code. 2. The data may be formed from multiple elements and may be represented by different data types. In web pages, multiple sections may represent the same entity. 3. The data has no pre-defined model and include irregular structure. On the other hand, structured data obey tabular structured and has a strict data model that can work well with relational databases and other forms of data tables.

To overcome the problems of semi-structured data, methods and tools exist to cleanup and wrangle the data (Krishnan et al., 2019). This is often done through a user interface-driven tool, which takes time and requires manual labor. Another approach is to transform the data into the structured form using transformation methods based on natural language pattern matching (Califf & Mooney, 1999), efficient wrapper generation (Suzhi Zhang & Shi, 2009), and HTML and XML parsing using tree structure (Hu & Meng, 2004).

2.2 DOM (Document Object Model)

Document Object Model (DOM) is a tree-based representation of an HTML document structure that consists of nodes, also known as (aka) DOM Elements. Each element can have a key tag and possibly a value such as `p` to represent paragraph, `a` for links, and `h` for headings (De Mol et al., 2015). An element can have any number of child elements. Below is an example of the HTML DOM Tree.

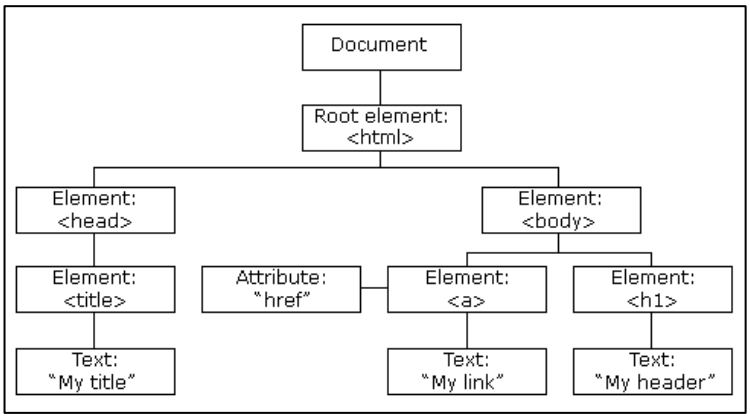

Figure 1: HTML DOM Tree, credits to w3schools.com

2.3 XPath (XML Path Language)

XPath can be used to identify one or more elements in XML or HTML documents. Although XPath supports both HTML and XML due to the fact that they are very similar and based on the Standard General Markup Language (SGML) (De Mol et al., 2015), in this study, the focus is on HTML web pages. XPath is also useful to navigate between elements and attributes in the HTML document and selecting particular nodes. Thus, a useful tool for extracting the desired information from web pages.

XPath consist of a set of steps separated by forward slashes. Each step can reference one or many node elements in the DOM. XPath also has (basic, set, and comparison) operators such as + addition, - subtraction, * multiplication, AND, and OR. Operators can be useful for selecting targeted elements from the DOM. Below is a simple example to illustrate how XPath can be used to select elements.

```
<html>
  <body>
    <h1>Very important title</h1>
    <p>First paragraph</p>
    <p>Second paragraph</p>
  </body>
</html>

//html/body/h1      --> Selects Very important title
//html/body/p[2]    --> Selects Second paragraph
//html/body/*       --> Selects all child elements
                        under the body tag
```

Figure 2: XPath examples on a simple HTML

2.4 OXPath

OXPath[1] from Oxford is an extension to XPath, a language designed for scalable web data extraction and automation (Fayzrakhmanov et al., 2018). It comes with improved features such as actions (e.g., clicking on elements and buttons, and form filling), Kleene star[2] (unbounded and bounded expressions) for iteration and navigation, and markers/wrappers for data extraction. The wrappers are responsible for extracting data from DOM trees of traversed web pages and appending the results to the output. The output of OXPath is a tree-like structure (Furche et al., 2013).

OXPath is suitable for data extraction at scale, since its memory requirement remains small regardless of the number of web pages crawled (Grasso et al., 2013). Furthermore, OXPath is designed to integrate with other technologies by providing an API. Although, the language has a standard API, XPath integrated libraries and tooling remain widely adopted.

[1] http://www.oxpath.org
[2] https://en.wikipedia.org/wiki/Kleene_star

2.5 JSON (JavaScript Object Notation)

JavaScript Object Notation (JSON) is an open standard format that can represent entities in an unordered enumeration of properties, consisting of key/value pairs (*JSON*, 2020) (Klettke et al., 2015). JSON supports basic data types such as number, string, and boolean. It can support more complex structures using multi-value nested objects and the use of arrays. Arrays are ordered lists of values separated by commas. An array value can include a JSON structure, resulting in an array of JSONs. Figure 3 shows an example of a JSON object, addresses attribute is an array of JSONs.


```
{
   "name":"Leanne Graham",
   "email":"Sincere@april.example",
   "addresses":[
      {
         "city":"Gwenborough",
         "geo":{
            "lat":"-37.3159",
            "lng":"81.1496"
         }
      },
      {
         "city":"Wisokyburgh",
         "geo":{
            "lat":"-43.9509",
            "lng":"-34.4618"
         }
      }
   ],
   "website":"hildegard.example"
}
```


Figure 3: Example of a JSON object

2.6 JSON-LD (JSON for Linked Data)

The linked data concept refers to a set of principles for sharing machine-readable structured and interlinked data on the web, a format type commonly used in Semantic Web (SW). SW is a concept to enable machines to manipulate interlinked web data meaningfully. In other words, the vision of SW is to have a universal global database of the web data that can be searched publicly and understood by machines (Berners-Lee & others, 1998).

JSON-LD is a method to encode linked data using the JSON format and add semantics to existing JSON objects (Sporny et al., 2014) (Lanthaler & Gütl, 2012). It was designed to empower existing tools and libraries that use traditional JSON and thus fully compatible by design. It comes with additional benefits such as pagination metadata, filters, and support to datatypes with values as dates and times. Furthermore, it provides a way in which a value in one JSON object can refer to another JSON object on a different site on the web. A useful feature to link nested object by reference instead of nesting the whole thing.

Figure 4 shows the former JSON example in a JSON-LD format. Notice that @context and @type were added to link the data semantics and @nest to reference a nested object.

```
{
   "@context":"http://schema.org/",
   "@type":"Person",
   "name":"Leanne Graham",
   "email":"Sincere@april.example",
   "addresses":"@nest",
   "website":"hildegrard.example",
   "addresses":[
      {
         "@type":"PostalAddress",
         "city":"Gwenborough",
         "geo":{
            "@type":"GeoCoordinates",
            "lat":"-37.3159",
            "lng":"81.1496"
         }
      },
      {
         "@type":"PostalAddress",
         "city":"Wisokyburgh",
         "geo":{
            "@type":"GeoCoordinates",
            "lat":"-43.9509",
            "lng":"-34.4618"
         }
      }
   ]
}
```

Figure 4: Example of a JSON-LD object

2.7 Google Rich Results

Google Rich Results, aka Google Structured Data or Schema Markup, is a standardized format to enable advanced search engine optimization (SEO) capabilities in Google Search. It is a way of describing a website to make it easier for search engines to understand and parse the site content to deliver enhanced experiences to its users.

Search engines including Google[3] and Yandex[4] have adopted JSON-LD to be the recommended format for describing search rich results.

[3] https://developers.google.com/search/docs/guides/intro-structured-data
[4] https://webmaster.yandex.com/tools/microtest

At this moment, Google supports the following features:

Article, Book, Breadcrumb, Carousel, Course, Critic review, Dataset, EmployerAggregateRating, Event, Local Business, Logo, Podcast, Product, Recipe, Review snippet, Software App, Subscription and paywalled content, Video, Fact Check, FAQ, Home Activities, How-to, Image License, JobPosting, Job Training, Speakable, Sitelinks Search box, Q&A, Movie, and Estimated salary.

Table 1: Google Rich Results features

Here is an example of a logo feature in JSON-LD code, and further details about the other features can be found in Appendix A. In this example, Google recognizes and requires at least the attributes logo and URL to display the content as a rich result. Additional image guidelines are required to be eligible for this feature, including image must be crawlable and indexable, and in a specific size.

```
<html>
  <head>
    <title>About Us</title>
    <script type="application/ld+json">
    {
      "@context": "https://schema.org",
      "@type": "Organization",
      "url": "http://www.example.com",
      "logo": "http://www.example.com/images/logo.png"
    }
    </script>
  </head>
  <body>
  </body>
</html>
```

Figure 5: Google Rich Results logo feature example

2.8 Robots.txt and No-index Meta Tag

It is a text file that tells search engines and crawlers which pages can be requested from the website and which not (*Introduction to Robots.Txt - Search Console Help*, 2020). This file allows crawlers to know and control how to crawl and index websites that are publicly accessible. It is a mechanism to enable traffic management to the site and avoid overloading it with crawling requests. To prevent crawlers from accessing the websites entirely, site owners can use the no index meta tag.

2.9 Structured Information Need and Applications

As mentioned in the previous sections, search engines such as Google prefer structured data to display rich results to users and to rank websites higher accordingly. In addition to that, a well-defined structured schema enables efficient data processing, data integration, data interoperability, and an improved way to query and analyze the content of the data (Sint et al., 2009). (Abiteboul et al., 2014) discuss several advantages of structured data, including optimizing query evaluation, improving data storage, and supporting strongly typed languages.

Due to the benefits of structured information and the problems of semi-structured data, transformation methods exist to overcome these limitations (Krishnan et al., 2019). Hence, the need for structured information is importantly increasing to enable many applications. Below a list of example applications in different industries:

- *Company data*: in our digital age, each company has a web presence. Companies have an interest in monitoring and getting insights on market dynamics, e.g., services and products being offered, monitor pricing changes of competitors. Data can provide answers to questions e.g., what is the average price point of x SaaS platforms? Which websites have updated their home page last week? What are the most common use cases published on x websites? etc.
- *Product data*: many webshops offer their products online, including descriptions, pricing, and other information. This data can be used, for example, to enrich existing product catalogues. Questions like: which products my existing catalogue lack, but other competitors have? What are similar products to product x? How have other companies described product x? What are the top recent reviews about product x? What are the top characteristics found on the home page of x company? How many companies use a large font on the home page? How many companies have a signup form on the home page? Does this increase the conversion rate?
- *Sport data*: information about players, local players across provinces, information about matches (e.g., location, time, results). Top-performing teams in specific locations in different sports.

3

Related Work: Existing Web Information Extraction Methods

(Laender et al., 2002) proposed a taxonomy comprised of six groups for existing web data extraction:

1. *Languages for wrapper development*: the idea is to employ specially designed languages using e.g., Python and Java, to assist users in developing and generating wrappers. A wrapper is a configuration to identify the data of interest among many other undesired data (Laender et al., 2002).
2. *HTML-aware*: methods that rely on the structural features of HTML web pages to turn documents into a parsing tree, and then perform the data extraction task.
3. *Wrapper induction*: approaches based on learning certain features such as web page formatting, in order to define the structure of the data and perform the extraction task (Varlamov & Turdakov, 2016).
4. *Ontology-based*: given a specific domain, an ontology is built relying directly on the data rather than the structure of the web page.
5. *Modelling-based*: aim at finding sections in the web pages that match a pre-defined model.
6. *NLP-based*: methods that leverage free natural text within the web pages to learn extraction rules.

Below we present a similar taxonomy that better fits this study and discuss each in more detail.

3.1 Specification-based Extraction Methods

This category includes traditional methods that require user interaction, user input, or manual work. Usually, researchers build browser-based plugins or rely on configuration-based files to specify how to extract information from web pages. In other words, the user must identify and provide the attributes of interest and the extraction rules for the extraction task manually, often per website or for a specific domain. A simple specification form would be providing a regular expression configuration to extract matching elements from web pages.

Such methods are not suitable for the requirements of this research, because they are limited specific website schema and require manual specification, although they might be useful for other use cases, for example, cases that aim at accessing private data. When accessing a website, the wrapper can benefit from browser-relevant cookies, authentication data, and user session data. In addition to that, since the plugin is installed locally in the browser and the extraction task takes place within this environment, rendering the web page content, including CSS styles and JavaScript, comes out of the box.

3.2 Element-specific Extraction

Element-specific methods are techniques that focus only on a specific element of the web page, such as WebTables (Cafarella et al., 2008) and TableMiner (Z. Zhang, 2009), which focus on identifying and extracting HTML tables with high quality data. (Shuo Zhang & Balog, 2020) (Hotho et al., 2016) present recent survey work including information extraction, web table extraction, retrieval, and augmentation. Others focus on extracting search results to build federated search engines (Trieschnigg et al., 2012), or extracting web assets such as images, audio, and videos to build media libraries.

The primary focus of this category is on extracting one specific element type (e.g., tables, forms, search results, images) from web pages rather than providing a generic extraction approach to any element.

3.3 Machine Learning-based Extraction

The studies in this category use machine learning techniques to extract web information from web pages. Such methods rely on the structure and presentation features of the data within a document to generate rules or patterns to perform the extraction task, usually, not fully automated but semi-automated and requiring training examples.

Wrapper induction is a subfield of wrapper generation in information extraction. Extraction rules are acquired from inductive learning examples then applied to new unseen websites (Muslea et al., 1999) (Goebel & Ceresna, 2009). In wrapper-based methods, learned rules are often applied to other pages within the same website, thus, require manual annotations for each new website. This makes them expensive and impractical at scale. Although this is generally the case, some researchers have proposed creative solutions to reduce the annotations workload, for example, using a bootstrapping approach for learning to use small seed annotations for training an initial classifier which then used to annotate the rest of the data (Jones et al., 1999). Also, there are situations where the necessary annotations can be derived from the website, given a sample structured data. Enough pages of the website can be automatically annotated this way before training the model (Jundt & Van Keulen, 2013).

(Yuchen Lin et al., 2020) define the task of web data extraction as a structured prediction task. They present a novel two-stage neural approach named *FreeDOM*, which combines both the text and markup information in the first stage and capture conditional dependencies between nodes in the second stage. The method is based on a neural network architecture and does not need to download all external files, including CSS style and JavaScript but relies only on the HTML content. However, it only focuses on detail pages that describe a single entity. Thus, might fail in real-world scenarios with multiple entities found on a single page.

Another direction of research is using end-to-end unsupervised deep learning methods. Html2Vec[5] and Web2Vec[6] (Feng et al., 2020) are proposed methods to encode web pages information based on multidimensional features in order to make the extraction task deep learning ready.

[5] https://github.com/dpritsos/html2vec
[6] https://github.com/Hanjingzhou/Web2vec

Figure 6 shows an example result of the Html2Vec on google.com home page, which encoded each of the DOM elements as machine learning ready vectors. Although this method has potential, it is still computationally expensive and may be challenging to use at scale.

```
python html2vec.py "https://google.com"


<Node: <Element html at 0x2068a63e640>>
[[ -402.47428608 -1547.25813076  2362.52118368 ... -2089.17908035
  -1683.24090653  3187.03134073]
 [-1976.48658743  -195.85040287  1080.3888098  ... -2957.07869986
  -1291.30960054  2712.65599157]
 [  182.01396129   226.8740429   -106.64038907 ...   247.96271209
   -304.35307753   -23.87794513]
 [ 3674.67073171  3674.67073171  3674.67073171 ...  3674.67073171
   3674.67073171  3674.67073171]
 [-2117.11113772 -1544.13482682  3215.64102592 ... -1995.88811237
    292.29043233  2382.17929111]]

<Node: <Element head at 0x206887be7c0>>
[[-3.37909572e+03 -7.53166519e+03  7.75814561e+03 ... -9.99011877e+03
  -9.98402913e+03  1.22860314e+04]
 [-1.42867226e+04 -5.91823399e+02 -1.48834708e+03 ... -1.57277051e+04
  -9.60222902e+03  1.11532172e+04]
 [ 1.29163054e+03  1.23525731e+03 -1.26250836e+03 ...  1.76596774e+03
  -2.02786761e+03  2.69001607e-01]
 [ 1.33844024e+04  1.33844024e+04  1.33844024e+04 ...  1.33844024e+04
   1.33844024e+04  1.33844024e+04]
 [-1.27583329e+04 -1.06106384e+04  1.59198359e+04 ... -8.35914209e+03
   2.08408578e+03  1.14322050e+04]]
```

Figure 6: Html2Vec example

3.4 Pattern Discovery-based Extraction

Pattern discovery techniques are methods that apply pattern discovery approaches to generate information extractors that can extract structured data from the web pages. Unlike most machine learning-based methods, which require manual annotated examples and training, pattern discovery methods do not rely on user labeled examples but on pattern discovery techniques including PAT-trees, multiple string alignment, and pattern matching algorithms. PAT-trees (aka Patricia trees) are effective at recognizing repeated patterns in semi-structured web pages.

(C.-H. Chang & Hsu, 1999) proposed a method to extract information blocks by converting the HTML web page into tokenized strings, which are then used to construct PAT-trees to detect patterns and locate the information. Similarly, (C. H. Chang et al., 2003) presented an unsupervised approach to generate extraction rules for web pages. The discovered patterns can be used on unseen web pages.

Although these methods require no labeled examples, they are quite limited when they encounter web pages that contain only one data record due to the lack of patterns. In addition to that, the number of patterns increases considerably when there are too many layouts/structures, and the discovered rules generalize poorly on other websites with different layouts (C. H. Chang et al., 2003). Finally, these methods do not include the process of automatically resolving the attribute names of the extracted data; but rely on the user to manually assign the attributes using a visual pattern viewer.

3.5 Vision-based Page Segmentation

Vision-based approach, known as VIPS Algorithm uses both the web page rendered source code and also analyzes the visual layout of the page (Cai et al., 2003). The visual layout input can be a screenshot image of the website. The algorithm generates a block tree that is structurally similar to the underlying DOM tree. Similarly, custom machine learning models can be trained on features by providing training samples, for instance, in the form of bounding boxes, to extract elements of interest from web pages.

(Liu et al., 2006) proposed a vision-based solution performs the extraction using only the visual information of the web pages. Given main visual features such as position, layout, appearance, and content of the web page.

Figure 37 in Appendix B outlines an example result which was done previously at M06 Company[7] using the vision-based approach, it shows the ability of clustering interesting elements of the web page. A clear downside of this approach is that embedded images on the page are included as part of the web page as well as the content inside the images. So, it is hard for the model to distinguish between which content belongs to the embedded images and which to the web page itself. Nevertheless, additional meta data can be provided to the model to overcome this challenge, for example, by excluding all the images from the page or covering them with solid colors.

[7] https://m06.company

4

Method: Design Science Research

According to the design science methodology (Wieringa, 2014), we treat design as well as research as a problem-solving process. In this section, we propose a complex design intended to improve on the web information extraction task and suggest research knowledge questions to return knowledge back to the design activity. The general goal of this design science research is to provide a *generic, automatic,* and *scalable* approach for extracting and structuring information from web pages.

The following subsections outline the problem statement and define requirements for the desired web extraction method in detail. The global method design is illustrated in Section 4.2. Challenges and research questions are discussed in Section 4.3. Our experiments are described in Section 4.4. Finally, Section 4.5 outlines metrics which are used to evaluate the experiments.

4.1 Problem Statement and Requirements

As mentioned earlier in Section 3, existing web information extraction methods are limited to specific website schema and hard at scale. Therefore, we seek alternative methods that can overcome these limitations. To be more precise, below, we define a set of requirements that have to be fulfilled for a web extraction system to reach the general goal:

- The system must be generic, work on any given website regardless of its schema and without pre-defined templates.
- The system must be fully automatic, require no user input nor interaction.
- The system must be scalable, adaptable, and can handle many different websites in millions.
- The system should be able to support different information needs.

- The system should support cross-language web pages, at least English and Dutch.
- The system does not enter any values in search bars or forms.
- The system does not consider websites that are locked behind a login page and only consider publicly accessible websites.

4.2 Global Design

Figure 9 outlines the global design of the system, which includes four steps. The subsections below provide a detailed description of each of the steps.

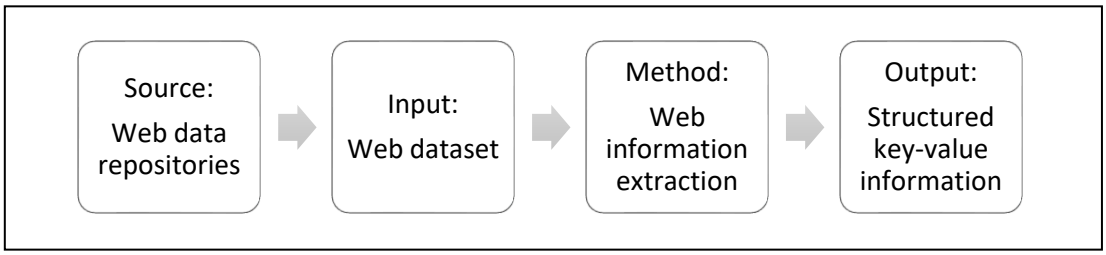

Figure 7: Global design

4.2.1 Input: Web Data

As mentioned in Section 4.1, the system must fulfill different information needs; this first step aims at obtaining web data (a set of web pages) that include information of interest to the user. Web data can be sourced from several different channels, such as the following:

- Directly from *public data repositories* such as CommonCrawl[8] a large corpus contains petabytes of web data collected over several years of web crawling, and ClueWeb[9] which contains about 733 million web pages.
- *Private data sources*, through partnering with data providers (e.g., dataprovider.com), a private database, or provided by a client for specific needs.
- Scraping the web and *own dataset collection*. Web crawlers can be coded to scrape the public web. This option should be used with care and respect to data privacy. Section 2.8

[8] https://commoncrawl.org
[9] https://lemurproject.org/clueweb12

discussed how site owners can use the no index meta tag to stop the crawlers from accessing their website.

After obtaining the web data of choice from some data *source*, the data might need preprocessing and transformation to form the *input* dataset. The input is a collection of unique links (URLs) to web pages with their raw HTML content (Figure 8).

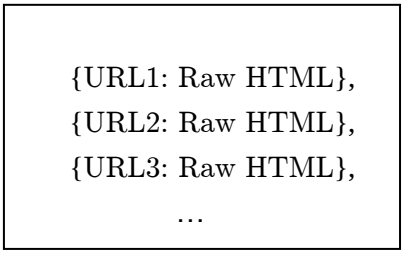

Figure 8: Web data input

4.2.2 Method: Web Information Extraction

This step represents a pipeline that contains four components, which form the generic web information extraction system (Figure 9).

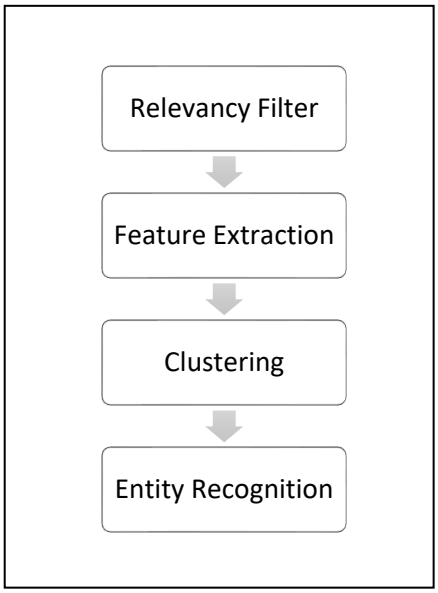

Figure 9: Generic web information extraction pipeline

Relevancy Filter to filter out irrelevant web pages to the user. Web data is messy as it contains many web pages in different shapes and forms; for example, a website may include different languages or may contain useless information to the user. Instead of putting the burden of filtering web pages on the user (which is time-consuming and not scalable), this component becomes essential as it can do the filtering automatically.

Collection of web pages ➔ Collection of relevant web pages

Figure 10: Relevancy filter input/output

Feature Extraction extracts candidate features given the input dataset, including XPaths for all elements per page and put them into a dataset. This process is typically followed by a feature selection process to select which features contribute the most to the output of interest.

Web page content ➔ List of features, including XPaths

Figure 11: Feature extraction input/output

Clustering to group similar HTML elements into groups based on the extracted features. The general assumption is that elements with similar features might belong to the same entity.

Features + HTML content ➔ List of clusters

Figure 12: Clustering input/output

Entity Recognition generates possible entity rankings for each cluster, given a knowledge base containing information about different entities. The knowledge base may consist of the entity rankings generated by the system and additional information about user-specific entities. This knowledge can be further enriched by public knowledge graphs such as Schema.org and Google Knowledge Graph. Entity recognition is also responsible for tagging each cluster with the corresponding entity.

Clusters + Entity knowledge base ➔ Key-value pairs output

Figure 13: Entity recognition input/output

4.2.3 Output: Structured Key-value Information

The output is a structured key-value pairs information in a JSON format (or JSON-LD ideally). This structured output can be used to fulfill the need for structured information and build smart applications (see Section 2.9).

While Section 2.5 discussed how JSON objects could be constructed, the ideal output is an object that can match an entity object, where the keys refer to the names of the entity object or attribute, and the values refer to the values for the same entity. Given a web page as an input, the expected output is a structured JSON object which contains a set of entity objects.

Although a web page might include information that cannot be classified as an entity, nevertheless, we consider elements with the same attributes as an entity (e.g., a web page might have a footer section with some navigation links or a header with menu items). The resulted information in these examples might not look relevant to someone, but very useful to others; this depends on the business use case, therefore from a generic design perspective, we leave all extracted information, and keep it up to the user to decide what is relevant. Another alternative approach would be to apply a post-filtering mechanism automatically given the use case (but this is outside the scope of this thesis).

4.3 Challenges and Research Questions

Within the proposed complex global design, some aspects are challenging and require research. In this thesis, we focus extensively on the first three pipeline components (proposed in Section 4.2.2). The fourth component Entity Recognition is discussed briefly in Section 5.4, but we leave extensive research on this component for future work. The remaining of this section outlines challenges and research questions (for each pipeline component) that need to be answered in order to return knowledge to the design science activity.

Relevancy Filter: web data is messy and disconnected, as it contains irrelevant web pages (e.g., a web page that has no structure, just a plain text or has unrelated content, given a business use case). Note that relevancy is subjective to a business use case or need; therefore, we must look for generic solutions.

***Q1**: How can irrelevant web pages be filtered out?*

Feature Extraction: web pages contain common patterns and features (e.g., elements have similar style, font-size, or color). These features, once identified, are helpful for the rest of the pipeline, for example, to cluster similar elements and recognize entities.

***Q2**: What candidate features can be extracted from web pages?*

XPath is a simple yet effective feature to locate elements in web pages; however, there could be many ways to construct XPaths to find elements effectively.

***Q3**: How can XPaths be constructed to effectively locate web page elements?*

Clustering: given the previous component features, the goal is to form clusters such that similar HTML elements group together. Many clustering methods exist, including spectral clustering, probabilistic relational models, graph partitioning, fuzzy clustering, and similarity-based clustering, but which one is best suited for the web information extraction task.

***Q4**: What existing clustering methods can be effectively used for clustering web page elements?*

***Q5**: What evaluation metrics can be used to measure the performance of clustering?*

Entity Recognition (for future research): given the previous components knowledge, the system should be able to suggest entities automatically for each cluster (e.g., based on the HTML structure, code, and content). For example, an element could be inside an HTML table, and directly above it a large-sized title. Can we then assume that the table column or row name correspond with the entity attribute and the large-sized title correspond to the entity name?

***FQ1**: How can candidate features be leveraged to identify entities?*

The knowledge of entities and entity features can be stored in a knowledge base (e.g., a price entity is a number, relatively short in length, often comes next to a currency sign and could include comma or a dot.). However, it could happen that the HTML structure, code,

and content do not include any information that can help the system in identifying entities. In this case, we could rely on additional information such as user-specific knowledge base and public knowledge graphs.

***FQ2**: What is an effective method for constructing a knowledge base to hold entity representations?*

The output is a structured key-value pairs JSON; however, it can be constructed in different ways (e.g., flatten or nested JSON objects)

***FQ3**: What is the best way to generate key-value structured information?*

4.4 Experimental Setup

Several experiments can be carried out to validate the proposed method. In this thesis, we focus on two experiments (the first two pipeline components, see Section 4.2.2). The subsections below explain the main two activities involved in the process of web information extraction, the data collection process to obtain a stream of web pages, the relevancy filter and clustering experiments setup. Finally, a use case which focuses on blocky websites is described to sketch out the need for structured information.

4.4.1 Extraction vs Recognition

The proposed method primarily consists of two main activities: *extraction* and *recognition*.

Extraction is the process that aims at locating and extracting candidate elements given a web page HTML content as an input. Clustering methods are applied to group similar candidate elements together. Additional filtering of unnecessary groups can be applied to filter out irrelevant groups. A group might result in an attributes group for a particular entity. Another clustering step is needed to enhance and link the group of entity attributes to an entity object.

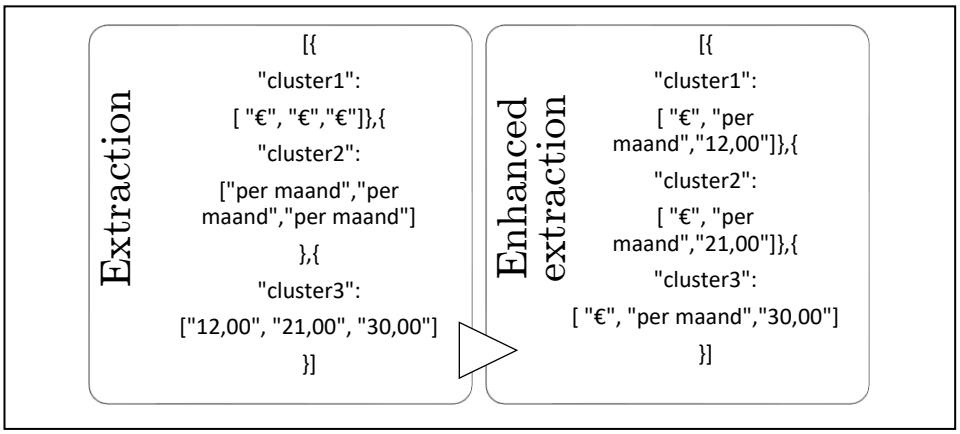

Figure 14: Extraction enhancement

Recognition is the process of identifying the entity class for each cluster/group to provide rich structured key-value pairs output. Recognition often comes after the extraction process. The figure below illustrates an example.

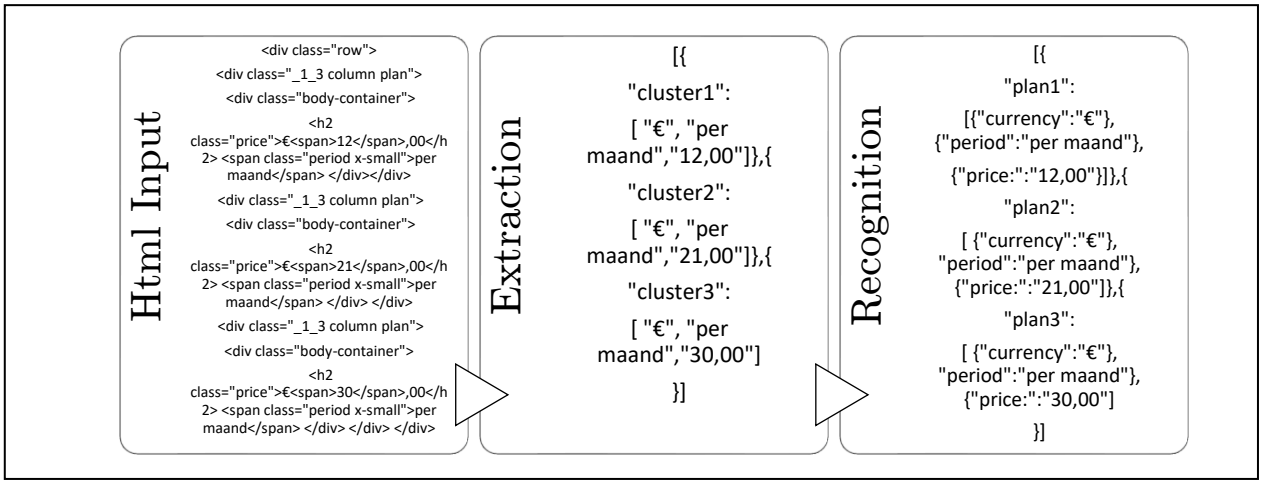

Figure 15: Method extraction vs recognition

4.4.2 Dataset Collection

To validate the proposed method and system design we need a stream of web pages as an input (see Section 4.2.1). This section outlines the process of web data collection at scale.

For this research, a publicly available data is collected from the Dutch Chamber of Commerce. The choice obtaining own dataset gives the research more flexibility and control. The data contains information about registered businesses in the Netherlands such as the official business name, website URL, and location address. For this research, only the website URL is needed.

Figure 16 illustrates the dataset collection process. The process starts with the collected companies' data. The records that are not reachable or do not include website URL are ignored; since the URL is the only required field needed for the rest of the pipeline. Furthermore, the homepage HTML content for each active website is collected and passed to the relevancy filter. The purpose of this filter is to classify and filter out irrelevant websites. Section 5.1 discusses in more details how this relevancy filter works.

Given only the relevant websites, external and internal links are extracted. External links are the URLs that link to other websites, and therefore treated as new websites. Internal links are all the nested links found under the same domain name. Finally, a limited sample (e.g., 10 web pages per website) is taken of the internal links, and the HTML content is collected and appended into the final dataset.

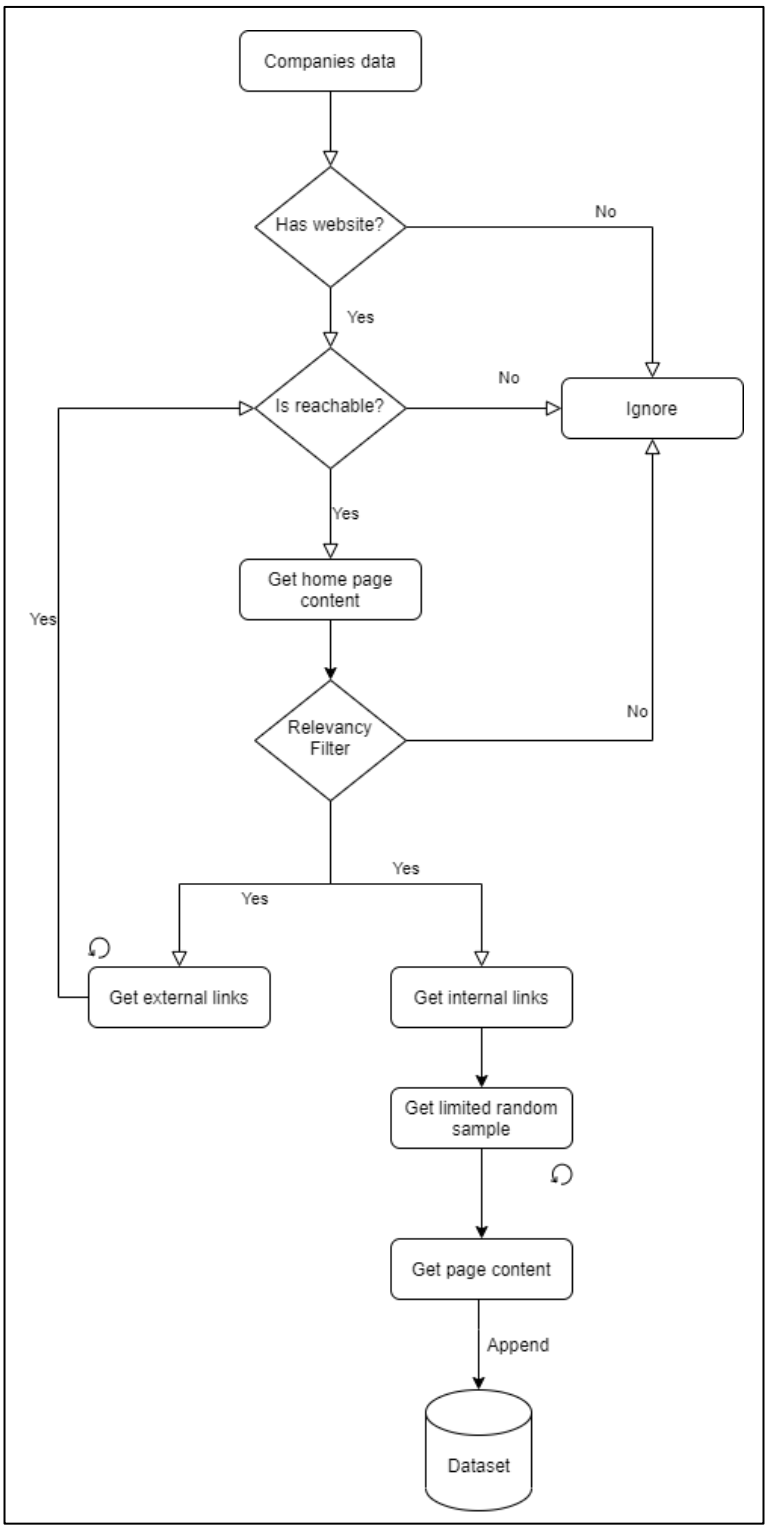

Figure 16: Dataset collection process

4.4.3 Experiments

This section briefly outlines the two experiments (further explanation is provided in the system design components sections 5.1 and 5.3)

As mentioned earlier, the goal is to extract and classify information from web pages and convert it into a structured form. The input is a web page, and output is a JSON object. As an example, two web pages with the desired structured output are listed in Appendix C.

Table 2 outlines the two experiments for the relevancy filter and clustering components. Note that the clustering is evaluated manually on a small sample proportion. The evaluation metrics are explained in detail in Section 4.5.

	Relevancy Filter	**Clustering**
Input	Collection of web pages	XPath features
Output	Collection of relevant web pages	List of clusters
Dataset	Three datasets: English (240 web pages) and Dutch (260 web pages, and a larger one containing 437 web pages)	Three web pages containing 1177 XPaths
Model	Custom binary classification	XPath-based similarity distance clustering
Evaluation metrics	Recall, F1, and Area under the curve	Purity and Percentage purity

Table 2: Experimental setup

4.4.4 Use case, Need of Structured Information

Section 2.9 discussed the need for structured information and some applications. Section 4.1 discussed the system requirements. In the research, we focus on the following use case based on the requirements, in particular on websites with blocky structure and exclude other irrelevant and private websites that are locked by the login screen.

A website is considered blocky when it has repetitive blocks like pattern. This pattern is often found in the following website categories (See Appendix D for figure examples):

- *Business informative websites*: sites that include only information and present media content, for example, self-starters, product offering business, non-profit, marketing-oriented websites, and other informative landing pages.
- *Ecommerce websites/webshops*: sites that offer and sell products online, both digital and physical. For example, B2C webshops that sell primarily products online and delivery directly to homes. B2B webshops that sell technical products to other companies and factories.
- *Pricing packages included websites*: such websites can be seen as ecommerce websites in its nature; however, the key difference is that webshops often include many products, but pricing packages included websites don't. They do include few-tiered pricing packages, for instance, as seen in freemium business models (e.g., free, basic, premium), SaaS offering platforms, service offerings, self-starters, and businesses that sell educational content and courses.

4.5 Evaluation Metrics

The method is evaluated based on commonly used metrics among researchers in web information extraction: precision, recall, F1, and area under the curve for the relevancy filter component and purity and percentage purity for the clustering component.

Usually, the following characteristics are needed to evaluate information extraction systems:

- A single score measure is needed to reflect on how well the system is performing.
- Extraction systems are often evaluated using a *relevance judgment*, in which the relevancy of one record does not affect other records.
- Relevance judgment is a *binary classification* choose, which means a record is either relevant or not.
- An ideal system should be able to classify relevant records and filter out irrelevant records.

Practically speaking, it is often hard and time-consuming to judge the full dataset; therefore, the evaluation is instead done on a smaller sample. Thus, the final measures are calculated from the sample proportion.

The next subsections outline in more details each of the evaluation metrics, but first, we define the following symbols which will be used to describe the metrics:

Symbol	Description
record	A data record referring to a website web page
r	The total number of relevant records correctly classified
n	The total number of records
R	The total number of relevant records
TP	True positive is where the model *correctly* classifies a *relevant* record
TN	True negative is where the model *correctly* classifies an *irrelevant* record
FP	False positive is where the model *incorrectly* classifies a *relevant* record
FN	False negative is where the model *incorrectly* classifies an *irrelevant* record

Table 3: Evaluation metrics symbols

4.5.1 Precision

Precision also known as a positive predictive value, which is defined as the percentage of results (predictions) made that are classified correctly as relevant by the model (TP) out of all the checked records (TP + FP). Precision is represented by the following formula:

$$Precision = \frac{TP}{TP+FP} \quad \text{OR} \quad Precision = \frac{r}{n}$$

4.5.2 Recall

Recall is the percentage of relevant records correctly classified by the system (TP) out of all relevant records ($TP + FN$). This percentage is also known as hit rate, coverage, or sensitivity and can be represented by the formula:

$$Recall = \frac{TP}{TP + FN} \text{ OR } Recall = \frac{r}{R}$$

4.5.3 F1

F measure (aka F1 score) is considered a sort of accuracy measure. It combines both precision (P) and recall (R) in a single score measure; therefore, it gives an overall estimate of the system performance. F measure can be calculated by the following formula:

$$F1\ Score = \frac{2\ x\ precision\ x\ recall}{precision\ +\ recall}$$

4.5.4 AUC (Area Under the Curve)

Area Under the Curve is a measure that describes the ability of a classifier to discriminate between two classes (Ling et al., 2003). When the AUC value is equal to one, that means the classifier can classify a record with a 100% confidence that it belongs to one class over the other.

4.5.5 Purity

The previous metrics are good to evaluate classification tasks, however other metrics are found to better measure the quality of clustering tasks. Clustering is the task of grouping similar objects in the same cluster. The quality of clustering can be measured based on the notion

of cluster *purity*. Purity is a simple and popular evaluation measure that describes how pure is one cluster with respect to a particular class (Amigó et al., 2009).

Purity can be computed by the following: For each cluster, the number of elements from the most dominant class is counted, and the sum over all clusters is calculated and divided by the total number of elements (Y. Zhao & Karypis, 2001).

$$purity(\Omega, C) = \frac{1}{N} \sum_{k} \max_{j} \left| \omega_k \bigcap c_j \right|$$

where $\Omega = \{\omega 1, \omega 2, \ldots, \omega K\}$ is the set of clusters and $C = \{c1, c2, \ldots, cJ\}$ is the set of classes. ω_k and c_j are the subsets of elements from the total set of elements (Manning et al., 2008).

Purity can be viewed as the weighted precision of all clusters (Huang, 2011), and the greater the value of purity indicates good clustering (Sripada & Rao, 2011). A perfect clustering has a purity score of 1 and is achieved by placing each of the elements in its own cluster.

Percentage purity is a similar metric that can better describe what proportion of the total elements within clusters is relevant. It can be calculated by the following formula:

$$Percentage\ Purity = \frac{Number\ of\ relevant\ elements}{Total\ number\ of\ all\ elements}\ x\ 100\%$$

5

System Design and Components

Section 4.2 presented the method global design. In this section, we research in more detail each of the pipeline components.

5.1 Relevancy Filter

The web is messy as it comes in many different shapes and forms and might contains irrelevant content. Therefore, this relevancy filter component is introduced to filter out irrelevant websites. The filtering challenge is seen as a binary classification task, in which a custom classifier is trained and evaluated on manual annotations for this purpose.

To train the custom classifier a library named Spacy[10] was used. Spacy is a free and open-source python library for advanced natural language processing tasks, including text classification. Spacy v3.0 just came out recently which introduced many new features and improvements, including end-to-end workflows[11] and pipelines. Although we chose Spacy for this component due to its simplicity, any other text classification library should reproduce similar results.

Two custom classifiers were trained on English dataset and another on Dutch dataset. This decision was made because both of these languages include different keywords and training a single multilingual model would be expensive, time-consuming, larger and less accurate than a focused model. However, note that this component can be retrained or extended on different data to narrow down the websites or include other languages based on the user use case and needs.

The next subsections present the models, annotation process, training the custom classifier, and evaluation metrics. The last subsection discusses and presents ideas on how to improve this component further.

[10] https://spacy.io
[11] https://spacy.io/usage/projects

5.1.1 Annotations

In order to train a custom binary classifier, manual annotations were needed, namely relevant and irrelevant examples.

A widget called pigeon[12] was used as an annotation interface due to its simplicity and compatibility with the rest of the pipeline. Figure 17 shows an example of the interface, which includes the following three options for each website: *accept* to consider the website as relevant, *reject* to count the website as irrelevant and *ignore* to disregard the record in case of any doubt. The interface includes additional helpful information, including the website to be annotated is shown, and the number of remaining and annotated examples so far.

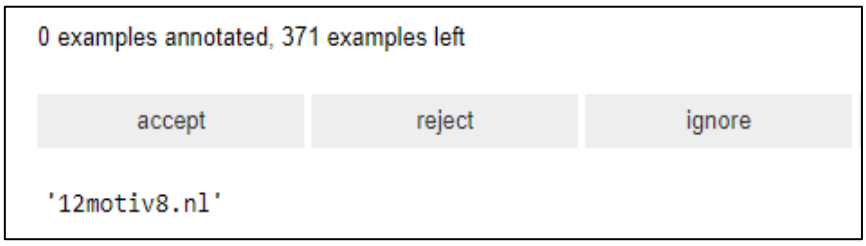

Figure 17: Annotation interface using Jupyter notebook pigeon widget

The table below outlines the criteria that were set and followed during the annotation process for both languages. See Appendix E for figures of several example annotation cases.

Action	Criteria
accept	- Websites with a blocky structure. As mentioned earlier (in Section 4.4.4), a website is considered blocky when it has repetitive blocks like pattern. For example, a page with a section that shows a list of products, brands, or team members etc.
reject	- Default web server pages - Default web hosting pages - Pages full of ads - Under construction pages - Under maintenance pages - Password-protected pages - Pages that include text content only without clear structure pattern - Pages that show no content but technical errors
ignore	- In case of unclear classification - In case of doubt

[12] https://github.com/agermanidis/pigeon

Table 4: Annotating criteria

Websites in both languages English and Dutch were annotated according to the mentioned criteria. Annotations are divided into two subdatasets training and evaluation, as shown in the table below.

Training data: a pre-labelled dataset which is given to the model with its label during the training process and is used to fit the model parameters and measure wights.

Evaluation data: an unseen dataset, in which labels are kept hidden in order to evaluate the trained model.

Dataset		**Accepted**	**Rejected**	**Total**
English				
E1	**Training**	40	100	140
	Evaluation	30	70	100
Dutch				
N1	**Training**	80	80	160
	Evaluation	50	50	100
N2	**Training**	80	135	215
	Evaluation	50	172	222

Table 5: Annotations dataset

5.1.2 Workflow

This workflow was inspired by a Spacy example project[13]. The aim is to train the custom relevancy filter classification models on the manual annotations we collected earlier (mentioned in the previous section 5.1.1).

The workflow includes the following steps:

1. *preprocess*: to convert the data to a binary format which can be understood by spacy. A blank Spacy instance is initialized for each of the target languages.
2. *train*: to train a text classification model. Two models for each language were trained, resulting in a total of four models. The English models are trained on the same dataset (E1) but using two different configurations (C1, C2). Similarly, the Dutch models on the same configurations (C1, C2) but trained on two different datasets (N1, N2).

[13] https://github.com/explosion/projects/tree/v3/tutorials/textcat_docs_issues

3. *evaluate*: to evaluate the model and output evaluation metrics.

5.1.3 Pre-processing

In this step, the input dataset is cleaned by removing the HTML tags and comments. The data is then tokenized using Tokenizer.v1[14] to convert text into words/tokens. The tokenizer batch size is set to 1000 records to process the input as a batch stream rather than one text at a time, which makes it faster and more efficient.

The tokenization step is followed by two processing components *Tok2Vec*[15] and *Text*Cat[16].

Tok2Vec basically converts tokens to vector embeddings which can be understood and processed by the machine. MultiHashEmbed.v1[17] embedding layer is used to embed features using hash embedding and build a mixed data representation. MaxoutWindowEncoder.v2[18] is used to encode the features using maxout units as activation functions. Figure 18 shows the tok2vec parameters we used to build the models. Here we include more details for some of the parameters. The *include_static_vectors* parameter is set to false because we do not have any pretrained vectors but aim to train the models from scratch on the annotated data. The *rows* parameter represents the number of rows for each embedding tables. This value is left the same as it was recommended by Spacy. The encoder *depth* represents the number of layers. Increasing the layers from 4 (in C1) to 5 (in C2) has improved the model performance.

[14] https://spacy.io/api/tokenizer
[15] https://spacy.io/api/tok2vec
[16] https://spacy.io/api/textcategorizer
[17] https://spacy.io/api/architectures#MultiHashEmbed
[18] https://spacy.io/api/architectures#MaxoutWindowEncoder

```
[components.tok2vec.model.embed]
@architectures = "spacy.MultiHashEmbed.v1"
width = ${components.tok2vec.model.encode.width}
rows = [2000,1000,1000,1000]
attrs = ["NORM","PREFIX","SUFFIX","SHAPE"]
include_static_vectors = false

[components.tok2vec.model.encode]
@architectures = "spacy.MaxoutWindowEncoder.v2"
width = 128
depth = 4 in C1 and 5 in C2
window_size = 1
maxout_pieces = 3
```

Figure 18: Tok2Vec processing component

TextCat is a text categorizer component that is used for single label (binary) text classification. The categorizer predicts classes over a whole record (web page). A record can have exactly one true label (either relevant or irrelevant). The categorizer threshold is set to 0.5 to cutoff the result and consider a prediction. TextCatEnsemble.v1[19] model instance is used to predict scores for each class. No additional pretrained vectors were used but only the feature encodings from the previous step. Therefore, the *pretrained_vectors* parameters were set to null. Figure 19 outlines the parameters we tuned to optimize the model performance. The *width* value refers to the output dimension of the feature encoding step. The *ngram_size* determines the maximum length of the ngrams in the model. Increasing the width from 64 (in C1) to 128 in (C2) in addition to the ngram size from 3 (in C1) to 5 in (C2) have accelerated the performance. This way larger number of text phrases and co-occurring words are captured as candidate matches, leading to better classification.

[19] https://spacy.io/api/architectures#TextCatEnsemble

```
[components.textcat.model]
@architectures = "spacy.TextCatEnsemble.v1"
exclusive_classes = false
pretrained_vectors = null
width = 64 in C1 and 128 in C2
conv_depth  5
embed_size = 5000
window_size = 1
ngram_size = 3 in C1 and 5 in C2
```

Figure 19: TextCat processing component

To sum up, all the four models had similar configuration parameters but with some tuning to optimize performance (As mentioned earlier in this section). *Table 6* presents the parameters we changed.

Parameter	C1	C2
tok2vec.model.depth	4	5
textcat.model.width	64	128
textcat.model.ngram_size	3	5

Table 6: Processing components parameter optimization

5.1.4 Training

This section outlines settings and controls that are used for the training process. Figure 20 outlines the exact parameters we used. Most of the parameters are set to the default value recommended by Spacy.

Spacy's training documentation[20] is rich and has explained all the parameters. Nevertheless, here we discuss most important ones. The *dropout* parameter is the probability of training a given node in a layer. Training batcher *batch_by_words.v1*[21] is used to split the text into a set of small batches of words. The *batcher.size* represents the target number of words per batch. *Adam.v1*[22] replacement optimization algorithm is used as a training optimizer. Optimizer learning rate warmup linear over the first 250 steps and 20000 total steps. This warmup setting is necessary in order to use a low learning rate than base learning rate for the initial few steps and prevent model overfitting.

[20] https://spacy.io/api/data-formats#config-training
[21] https://spacy.io/api/top-level#batch_by_words
[22] https://thinc.ai/docs/api-optimizers#adam

```
[training]
dropout = 0.1
accumulate_gradient = 1
patience = 1600
max_epochs = 0
max_steps = 20000
eval_frequency = 200
frozen_components = []
before_to_disk = null

[training.batcher]
@batchers = "spacy.batch_by_words.v1"
discard_oversize = false
tolerance = 0.2
get_length = null

[training.batcher.size]
@schedules = "compounding.v1"
start = 100
stop = 1000
compound = 1.001
t = 0.0

[training.optimizer]
@optimizers = "Adam.v1"
beta1 = 0.9
beta2 = 0.999
L2_is_weight_decay = true
L2 = 0.01
grad_clip = 1.0
use_averages = false
eps = 0.00000001

[training.optimizer.learn_rate]
@schedules = "warmup_linear.v1"
warmup_steps = 250
total_steps = 20000
initial_rate = 0.00005
```

Figure 20: Training parameters

5.1.5 Evaluation Metrics and Experimental Results

Out of the commonly used evaluation metrics (previously mentioned in Section 4.5) for extraction and classification tasks, we select and calculate Recall, F1 and AUC. These metrics make more sense for this task because they represent what we want to measure: the total relevant websites correctly classified by the models.

The table below outlines the evaluation metrics for the two models. The English models were assessed on the same dataset while the second Dutch model was evaluated on a different (slightly larger) dataset. As shown, the second models are the winners which we use further to classify relevant websites to use in the rest of pipeline.

Model configuration	**Dataset**	**Recall**	**F1**	**AUC**
English				
C1	E1	66.00	79.52	**0.52**
C2	E1	**97.00**	**98.48**	0.32
Dutch				
C1	N1	76.00	86.36	**0.66**
C2	N2	**97.75**	**98.86**	0.46

Table 7: Relevancy filter evaluation metrics

5.1.6 Discussion and Future Work

The custom trained models perform very well to classify relevant websites from others. However, they support only two languages English and Dutch. Additional different models are required in case other languages are needed. Two approaches to tackle this multilingual challenge:

- Training a custom model for each additional language.
- Training one larger language-agnostic multilingual model.

Another challenge linked to training larger models is the need for manual annotations which is an expensive and time-consuming task. A solution direction to this issue is to use what is known as *Bootstrapping approach* for text learning (Jones et al., 1999) (McCallum & Nigam, 1999). Bootstrapping annotations is the idea to use a small set of seed annotations to train an initial classifier. Then applying it to predict labels for a large collection of unlabelled data and incorporating some of the then new labels to train a final large classifier.

Another purpose and motivation to training own custom models is the adaptability to specific use cases. Someone could be interested only in webshops for example. In this case, annotations for webshops are marked as relevant and other types of websites are ignored. Thus, the advantage of this component that it can be easily adapted to other domains and use cases.

5.2 Feature Extraction

This step aims at extracting features from the relevant websites that were filtered and selected in the previous component. The extracted features could be useful for the following components, for example, to identify and group similar elements together in the clustering component and recognize named entities in the entity tagging/recognition step.

This section outlines candidate features, a list of possible features that could be extracted from web pages. In addition to that, it discusses the feature selection process and XPath extraction.

5.2.1 Candidate Features

This section presents a list of candidate features that can be extracted from three different web page representations: 1. visual page presentation is what people see visually and can be extracted using computer vision-based algorithms (see Section 3.5). 2. The actual source code of the web page (e.g., HTML, JavaScript, and CSS). 3. DOM tree representation which is parsed from the HTML source code (see Section 2.2).

All these features could be studied and leveraged to enhance the overall system performance, especially in the clustering and entity recognition components.

Table 8 outlines a list of candidate features.

Feature class	Candidate feature
Style	Font size, type, color, weight (e.g., bold, light, thin), style (e.g., normal, oblique, italic, underline), and background color
Dimensions and location	Offset/margin/padding width, height, top, right, and left. Border width and height, scroll height and width, z-index
Element information	Element attributes, classes, values, tag type (e.g., heading <h1>…<h6>, <a> link, <img> image etc.)
DOM information	Tag name, children, inner text, bounds/bounding client rectangle, XPath, parent XPath
Meta data	Title, description, type, site name, site page name, keywords
Other calculated values	Length of the text, average and standard deviation, average number of nodes, text density, number of attributes

Table 8: List of candidate features

The literature has also proposed similar features, for example, (Vogels et al., 2018) distinguish between two types of features, namely *block features* which capture information on each block of text in the web page and *edge features* which capture information on each pair of neighboring text blocks. Table 9 outlines a list of features proposed by (Vogels et al., 2018). These features study binary attributes (e.g., has duplicate, has parent, and has grandparent), the content structure of elements (e.g., what an element contains, what does it end with) and

compute additional unique features (e.g., the number of stopwords, number of words with capital letters).

Feature name
Has duplicate: is there another element with the same text?
Relative position of the element in the source code
Has parent
Has grandparent
Contains form element
Contains punctuation
Contains URL
Contains email
Ends with punctuation
Ends with question mark
Has a commonly used word (stopword)
Number of stopwords
Average word length
Number of words with capital letters
Has multiple sentences

Table 9: List of features proposed in Web2Text (Vogels et al., 2018)

5.2.2 Feature Selection

The previous section 5.2 listed many candidate features; however, in this thesis, we focus the feature selection process for the clustering component to one simple but informative feature, namely XPath. Nevertheless, we describe the feature extraction component generically, as it would enable future experiments to leverage the multiple features.

In general, the aim of the feature selection step is to select useful attributes and eliminate non-informative and redundant ones. In addition to that, to enable faster training time of machine learning algorithms, reduce model complexity and help in model interpretability. Below we outline several feature selection approaches found in the literature (Chandrashekar & Sahin, 2014; Khalid et al., 2014; Miao & Niu, 2016):

- *Filter methods*: typically, generic and does not incorporate a specific machine learning algorithm. The relevancy of each individual feature is analyzed and evaluated as a single factor.
- *Wrapper methods*: evaluate on a specific machine learning algorithm in mind to find optimal features. Wrapper

methods use combinations of features to find the best performing combination that determines the desired output.

- *Embedded methods*: filter out features during the model building and training process. The model takes care of the feature selection process since it has this function built in.

Feature selection methods aim at selecting relevant features and filtering out irrelevant ones without changing them. Another approach is to use dimensionality reduction methods to reduce the number of features by transforming them to lower dimension (Raymer et al., 2000; Sorzano et al., 2014). Most popular dimensionality reduction methods are: Principal Component Analysis (PCA) (Tipping & Bishop, 1999; Wold et al., 1987), Linear Discriminant Analysis (LDA) (Ioffe, 2006; Izenman, 2013), and Canonical Correlation Analysis (CCA) (Thompson, 1984, 2005).

5.3 Clustering

This clustering component aims at grouping similar HTML elements into clusters. Clustering is typically categorized as an unsupervised learning task of dividing and grouping a set of elements into several clusters such that elements in the same group are more likely similar to other elements in the same groups than those in other groups. In this thesis, the general assumption is that HTML elements with similar features will group under the same cluster; a cluster which most likely represents a named entity or entity attribute.

This section first presents several clustering methods found in the literature followed by a discussion on XPath feature, the main selected feature to perform the clustering. Levenshtein edit distance, which measures the difference between two strings, in terms of edit distance score is described and evaluated. Finally, future work and discussion on the clustering component is presented in 5.3.6.

5.3.1 Existing Methods

This section outlines a list of existing clustering methods which could be suited for grouping similar web page elements into clusters. Since the selected feature of the clustering component is XPath, which is a computed text-value reference of each element in the web page, we compiled the list focusing on text similarity techniques (see Table 10). The list was collected from different survey papers and scientific sources; below, references are provided.

Text similarity methods can be categorized into the four similarity approaches, String-based (includes Character-based and Term-based), Corpus-based, Knowledge-based, and Hybrid-based (Gomaa et al., 2013; Vijaymeena & Kavitha, 2016).

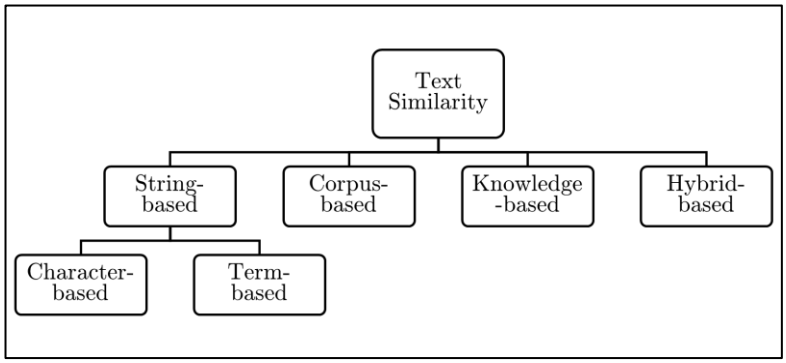

Figure 21: Text similarity categories

- *String-based* measures operations between strings and represents similarity in terms of distance scores. Usually, a small distance score represents a high similarity between strings and vice versa. String-based similarity includes character-based in which edit distances is measured between characters and term-based on words and phrases of text.
- *Corpus-based* measures semantic similarity between words based on information gained from large corpora/dataset.
- *Knowledge-based* determines similarity between words using information obtained from semantic knowledge bases (e.g., WordNet[23] a lexical database for English).
- *Hybrid-based* uses multiple similarity measures, basically, a mix of the former methods.

Category	Method	Reference
Edit-based	Levenshtein distance	(Levenshtein, 1966)
	Damerau-Levenshtein distance	(C. Zhao & Sahni, 2019)
	Hamming distance	(Bookstein et al., 2002)
	Jaro–Winkler distance	(Winkler, 1990)
	Needleman-Wunsch distance	(Needleman & Wunsch, 1970)
	Smith-Waterman algorithm	(Smith et al., 1981)
Token-based	Cosine similarity	(Li & Han, 2013; Sidorov et al., 2014)
	Sørensen–Dice coefficient	(Dice, 1945)
	Tversky index	(Tversky, 1977)
	Jaccard coefficient, aka Jaccard index	(Jaccard, 1912; Kosub, 2019)
	Szymkiewicz–Simpson coefficient, aka Overlap coefficient	(Vijaymeena & Kavitha, 2016)
	Tanimoto distance	(Lipkus, 1999)

[23] https://wordnet.princeton.edu

	Bag distance	(Bartolini et al., 2002)
	Monge-Elkan	(Monge & Elkan, 1997)
	Token sort ratio, aka fuzzy matching score	(Rao et al., 2018)
Other	K-means	(MacQueen & others, 1967)
	K-NN (K-nearest neighbor)	(Fix, 1985)
	EM (Expectation-maximization)	(Moon, 1996)
	GMM (Gaussian mixture model)	(Reynolds, 1993)
	Brown clustering	(Brown et al., 1992)

Table 10: Clustering methods

5.3.2 XPath Feature

The web page consists of many elements, this step aims at identifying each of the elements by an XPath locator. Normally, an element can be located by multiple different ways, for example by *element ID*, *CSS selector*, *text of the element*, *element class, or raw Full XPath*[24]. The XPath provide us a simple, effective, and unified way to locate all elements from the web page. Therefore, it was selected as the main feature to perform the clustering task.

The general assumption is a similarity-based clustering approach, in which elements that have similar XPaths are likely to cluster under the same group. In order to better illustrate this concept, consider the following example. Figure 22 shows a section of a pricing page (from Figure 39), where one XPath locates all the prices (the yellow boxes).

[24] A raw Full XPath traverses the hierarchy from the root element of the web page to the target element.

Figure 22: Prices located using XPath

As mentioned above, the price can be located in different ways, for example, by the element class name **(//div/h2[@class='price'])[*]**, or by the Full XPath. In order to provide a generic solution that works on any website regardless of how it was built (e.g., if elements include class names or not), we intentionally use the Full XPath for locating elements in the web page.

/html/body/div[2]/main/section[1]/section[1]/div[2]/div/div/div[**1**]/div/h2
/html/body/div[2]/main/section[1]/section[1]/div[2]/div/div/div[**2**]/div/h2
/html/body/div[2]/main/section[1]/section[1]/div[2]/div/div/div[**3**]/div/h2

Figure 23: Prices located by the Full XPath

Figure 23 outlines XPaths of the three prices. As shown above, the difference between XPaths is only one character (marked in bold). Hence, the potential of using character-based string similarity as a clustering method is encouraging. The next section outlines Levenshtein distance, a metric for measuring the difference between two strings.

5.3.3 Levenshtein Distance

Levenshtein distance is a character-based (aka *edit distance*) metric for measuring the difference between two strings. It is defined as the number of the smallest edit operations required to transform one string into the other (Levenshtein, 1966). A possible edit operation is a single character edit, including *insertion*, *deletion*, and *substitution*.

Levenshtein distance between two strings a, b (of length $|a|$ and $|b|$ respectively) is calculated by the following formula, $lev(a, b)$:

$$\operatorname{lev}(a,b) = \begin{cases} |a| & \text{if } |b| = 0, \\ |b| & \text{if } |a| = 0, \\ \operatorname{lev}(\operatorname{tail}(a), \operatorname{tail}(b)) & \text{if } a[0] = b[0] \\ 1 + \min \begin{cases} \operatorname{lev}(\operatorname{tail}(a), b) \\ \operatorname{lev}(a, \operatorname{tail}(b)) \\ \operatorname{lev}(\operatorname{tail}(a), \operatorname{tail}(b)) \end{cases} & \text{otherwise.} \end{cases}$$

Figure 24: Levenshtein distance formula, credits to wikipedia.org[25]

Where:

- $Lev(a, b)$ is the smallest number of edits to change a to b.
- The $tail$ function of some string x is a string of all characters except the first character of x.
- $Lev(tail(a), b)$ corresponds to deletion (from a to b).
- $Lev(a, tail(b))$ corresponds to insertion.
- $Lev(tail(a), tail(b))$ corresponds to substitution (replacement of a and b).

The next example illustrates the calculation of the Levenshtein edit distance between two strings *kitten* and *sitting*.

kitten → **s**itten (substitution of **s** for **k**)
sitt**e**n → sitt**i**n (substitution of **i** for **e**)
sittin → sittin**g** (insertion of **g** at the end)

Figure 25: Levenshtein distance example calculation

[25] https://en.wikipedia.org/wiki/Levenshtein_distance

The transformation had three edits to change the word kitten to sitting, and there is no other way to achieve the same with fewer changes than three edits, hance $lev(kitten, sitting) = 3$.

5.3.4 XPath-based Clustering

As mentioned previously (see Section 5.3.2), the goal is to cluster similar XPaths into groups using the Levenshtein edit distance metric (*Lev distance*). Levenshtein edit can measure the distance between two strings only, therefore, we must look at XPath locators that best suit the clustering task.

Table 11 presents XPath-based candidate features that can be derived from the XPath feature. Let us first define the following concepts:

- **Target**: the target element that we wish to cluster.
- **Root**: the root element, top element in the DOM tree; it is typically the HTML tag and can be located by /∗ [1].
- **Next**: the following sibling of the target element.
- **Previous:** the preceding sibling of the target element.
- **Parent:** the parent of the target element.

Note that Levenshtein edit score is calculated on the XPaths of these elements.

Key	Description
D1	$Lev(Root, Target)$
D2	$Lev(Parent, Target)$
D3	$Lev(Next, Target)$
D4	$Lev(Previous, Target)$
D5	Depth of the target element by counting its ancestors (which include the parent, the parent of the parent, until the top end of the DOM tree). The depth can be calculated using the following: **count(/html/ancestor-or-self::*)**, which equals to 1.
D6	Weighted Levenshtein distance to specify different weights for edit operations. In a default Levenshtein distance setting, all edit operations have a cost of one. It could be interesting to give different weights to some characters (e.g., give high weights to numbers since they refer to the position of an element in the XPath).

Table 11: XPath-based candidate features

Figure 26 illustrates some of the XPath-based candidate distances for the target element (the price on the right tire). In this example, the shortest distance is found between the target element T and its parent element P. Note that knowing the shortest distance by itself is not helpful at this stage, however such information might be useful in the rest of the pipeline. For the clustering task, it is useful to know that similar elements will have the same or nearby distances, hence they will cluster in the same group. For example, in the figure below, the prices 12, 21, and 30 have the same Levenshtein distance (Figure 27); therefore, they will group under the same cluster.

Figure 26: Levenshtein distances illustration

T: /html/body/div[2]/main/section[1]/section[1]/div[2]/div/div/div[3]/div**/h2**
P: /html/body/div[2]/main/section[1]/section[1]/div[2]/div/div/div[3]/div

$$Lev(Parent, Target) = 3$$

Figure 27: Example of XPaths distance between the target element and its parent

In this clustering component, given a web page input, we start by extracting XPaths of elements with a filter using the following expression: **//body//*[not(self::script or self::br)]**. The goal of the filter is to reduce unnecessary XPaths that do not link to actual informative elements, visible to the user. The following are cleaned out:

- Elements which are located outside the <body> tag (e.g., elements in the <head> tag, styles, and fonts).
- <script> JavaScript code, even the ones inside the body.
-
 empty line breaking tags, this tag does not include information.
- Empty elements which have no content.
- Hidden elements which are present in the DOM but not visible on the web page to the user. These elements are identified by searching for the term *hidden* in the element attributes or class values.
- Element wrappers/containers (e.g., tags) which group a set of child elements to define a division or section in the Html document. These containers often do not include any information (e.g., text content) but used for styling and structuring. Childless containers may contain unique text or additional text to what is in the child elements; in these cases, containers are kept (not removed).

Note that these elements are filtered out locally for the purpose of this component, and such meta information should be kept because it could be useful for the rest of the pipeline (e.g., for entity recognition).

The removal of element containers is applied to reduce the number of irrelevant XPaths prior to the clustering. The idea is simply to match short and long XPaths. For example, **/html/body/div[1]/div** is a container that is matched with **/html/body/div[1]/div/img** an XPath of an image element. Note that elements which contain additional unique text are not removed (e.g., /html/body/div[1]/div/p).

Figure 28 presents an example of 8 input XPaths were reduced to 5.

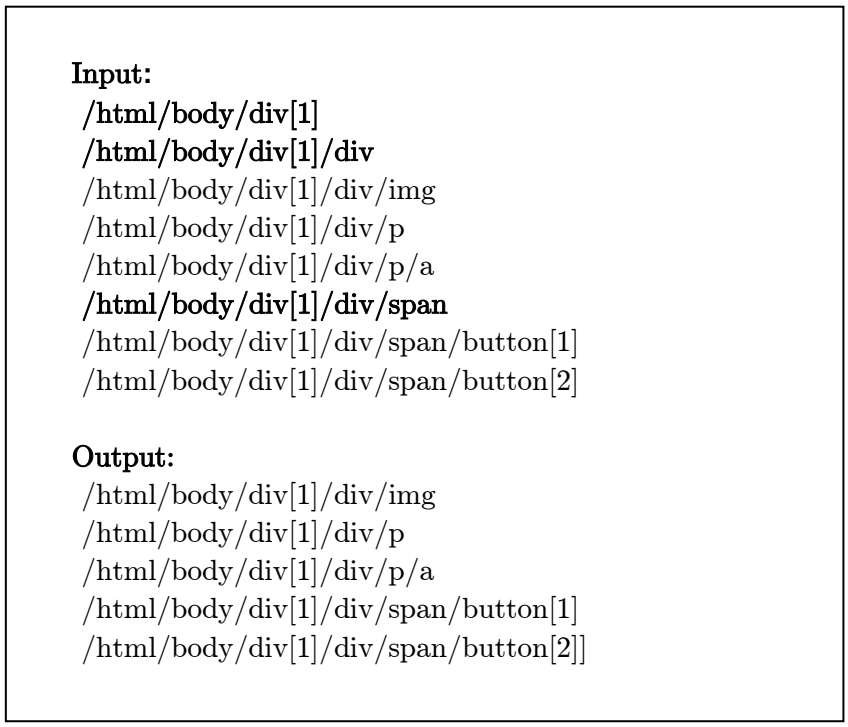

Figure 28: Example of element containers reduction

First, consider this example of three web pages from different websites (WP1[26], WP2[27], WP3[28]) which explores the effectiveness of the cleaning process and possible clustering patterns. Table 12 shows the number of XPaths before and after the cleaning process, which achieves a reduction average of 55.66% of the total XPaths.

Web page	Before	After	Reduction %
WP1	265	148	44.15%
WP2	1818	879	51.65%
WP3	521	150	71.20%

Table 12: Number of XPaths before and after the cleaning process

[26] https://www.moneybird.nl/prijzen
[27] https://www.imes.be/gereedschap-handling-en-bevestigingen/handgereedschappen/lampen-en-verlengkabels.html
[28] https://www.st-group.com/our-brands

Figure 29 and Figure 30 visualize the values of candidate features extracted from WP1, which shows a clear clustering pattern; as shown horizontally, similar values can form a possible cluster.

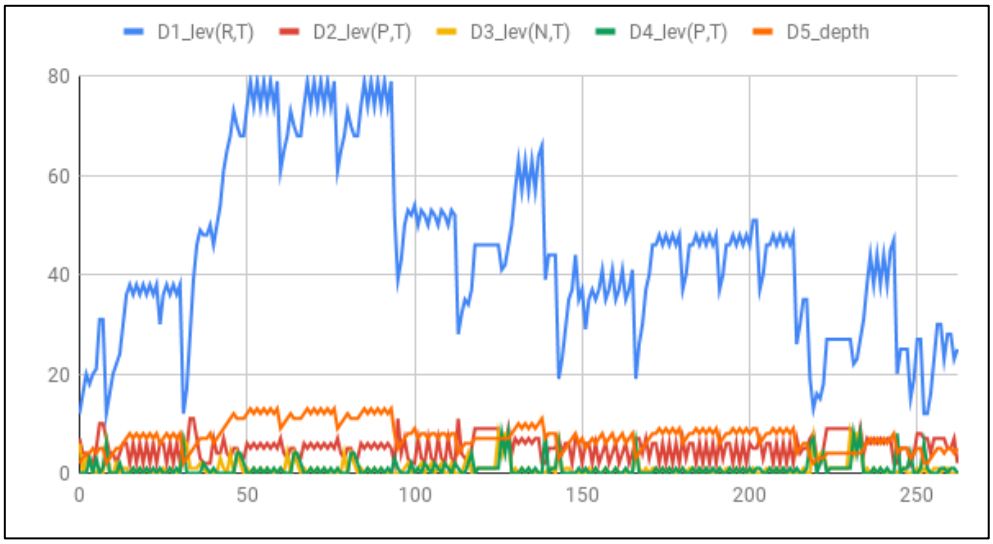

Figure 29: Line chart of XPath-based Levenshtein distances

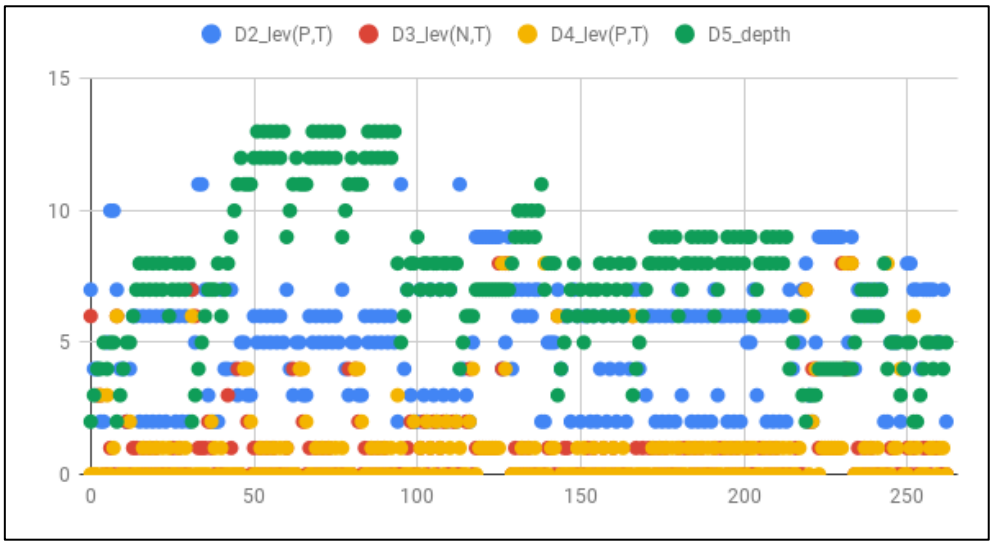

Figure 30: Scatter chart of XPath-based Levenshtein distances

The general idea to achieve clustering is to do sorting and grouping by a Levenshtein distance value (other approaches are possible; see Section 5.3.6). In order to validate which of the XPath-based candidate features fit the clustering best, we explore further the same web pages and present the results below. Note that the Weighted Levenshtein distance (D6) remains unexplored for future work, which requires an additional variable for each distance.

	Number of clusters		
Candidate feature	**WP1**	**WP2**	**WP3**
D1	26	53	5
D2	8	8	5
D3	8	8	5
D4	7	7	5
D5	12	15	5

Table 13: Number of clusters by XPath candidate feature

The next section presents evaluation and experiment results performed on D2 (the Levenshtein distance between the target element and its parent). This distance was selected in particular because it outputs the best clusters in terms of the number of clusters (average size as seen in Table 13) and their quality. The quality was evaluated visually by looking at the output clusters of each distance across the three web pages, using a visual evaluation interface (More in the next section). Figure 31 shows an example of a brands section that was clustered given D1 and D2. The left side (D1) had two clusters (C16 and C17) of the same thing, but the correct expected result should have been one cluster. D2 clusters are correct, (C2) refers to the brand boxes (e.g., entity object of name and image) and (C3) represents the entity attribute (e.g., brand name).

Figure 31: A section taken from WP3 shows the difference between D1 (left side) and D2

5.3.5 Evaluation and Experimental Results

Evaluating the quality of a clustering result is an important yet can be a difficult challenge because it is subjective task. It depends on the task specifics; for example, in this section the goal is evaluate how well is the proposed clustering component performs on the web pages (e.g., whether similar elements are grouped correctly or not). In order to evaluate this, several measures can be considered to capture performance at different levels: across various websites, different web pages within a website, across clusters within a web page or all pages to capture relevancy of clusters, and finally, across XPaths, to capture relevancy of locating target elements, and the XPaths relation with respect to the cluster.

The *purity* of clusters is considered as a metric (see Section 4.5.5) to evaluate the clustering of similar XPaths. It fits well this clustering component and the generic approach followed in the global design. A good clustering should include pure clusters and have less impure clusters.

In order to evaluate the XPath-based clusters and compute the purity metric, a simple visual evaluation interface was developed. The interface generates a color scheme automatically, containing the color, cluster id, and number of XPaths in the cluster (Figure 32). The interface then maps the different but unique colors (and cluster id tags) to each cluster, making it easy for the human eye to interpret and judge the resulted output. A bounding box is drawn on each cluster. Finally, the interface presents an image of the web page visually, including all the annotated information.

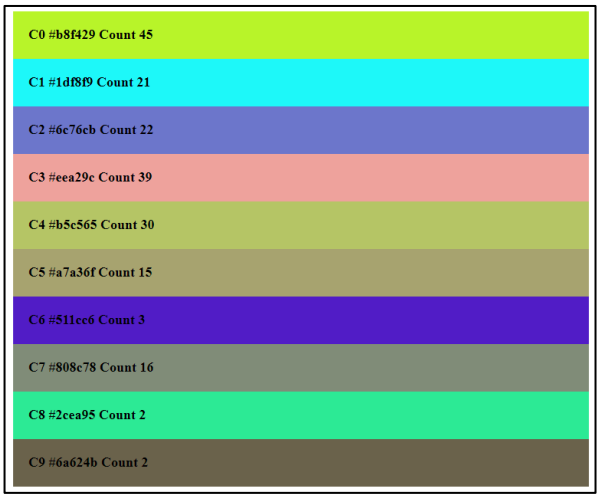

Figure 32: Clustering color scheme generated by the evaluation interface

Figure 33 shows an output example of a questions (FAQ) section. As shown, two main clusters were annotated (C0 and C1). C0 refers to the question attribute and C1 to the answer attribute; these attributes can relate to a FAQ entity object.

Figure 33: XPath-based clustering performed on FAQ section (WP1)

The previous section introduced a removal mechanism to reduce the number of irrelevant XPaths (e.g., element containers) prior to the clustering. Although progress has been made in cleaning up more than half of the XPaths, some XPaths passed through, which can lead to misevaluating the quality of the clustering. The purity metric is calculated based on the number of XPaths in the majority class. High purity is easy to achieve when irrelevant XPaths are left uncleaned, because they might represent the majority class in most of the cases.

To overcome this, we introduce the concept of XPath relevancy within a cluster. A relevant XPath is the one that locates an element that has similar elements within the cluster; single elements are considered irrelevant. Elements are considered relevant if they can represent a candidate entity attribute/class (e.g., price, product name). A candidate attribute is a possible output representation (e.g., the price attribute in WP1 can be represented as 21, 21,00, or €21,00); a follow up step should predict the best attribute given the candidates.

Furthermore, the concept of percentage purity which describes what proportion of the total XPaths is relevant; it can be calculated by dividing the total number of relevant XPaths by the total of all XPaths, and then multiplying this number by 100.

Figure 34 illustrates an annotated example of a pricing packages section used for clustering evaluation. The green bounding boxes represent the elements were located by the XPaths under one cluster. The red markings are annotated manually by a human judge. **/I/** is a

single irrelevant element which can be ignored. Although **A1**, **A2**, and **A3** are clustered under the same cluster, they represent different candidate attributes (e.g., A1 a price, A2 a period, A3 part of a feature name). The majority class in this example is 15 (A3).

Figure 34: pricing packages cluster annotated by candidate entity attribute (WP1)

Table 14 outlines the evaluation results for the three randomly selected web pages. The size of a cluster is determined by the number of XPaths it has. Note that clusters which include only one XPath were ignored prior to the clustering.

Web page	**WP1**	**WP2**	**WP3**
# of clusters	8	8	5
# of XPaths	148	879	150
Largest size	40	276	51
Smallest size	2	13	11
Mean size	18.5	109.9	30
Median size	16.5	80.5	25
# of irrelevant XPaths	58	572	44
# of relevant XPaths	90	307	106
# of relevant candidate entity attributes/types	15	29	14
% purity	60.81%	34.93%	70.67%
Purity	64.44%	43.32%	76.42%

Table 14: XPath-based clustering evaluation

The purity is computed by counting the number of XPaths of the majority class (in each cluster) and diving by the number of relevant XPaths. The average of percentage purity is **45.47%** and purity is **61.39%**.

The next section provides a detailed discussion on the results and outlines possible directions to improve the results in future work.

5.3.6 Discussion and Future Work

The clustering results presented in the previous section show that this research direction is promising and has enormous potential. Using a simple feature (XPath), the system was able to group similar elements groups correctly more than sixty percent of the time. The clustering could be enhanced further by leveraging the multiple features (see Section 5.2.1); for example, by applying k-means clustering on additional features besides the XPath (or other methods, see Section 5.3.1). In this case, additional work is needed to encode the various features into machine-learning ready features (e.g., element location as numerical feature, color, and tag type as categorical, and convert text content into vectors). In addition to that, to normalize large numbers and independent features of data (e.g., using z-scaling to ensure that feature distributions have a zero-mean and standard deviation of one) (Bejarano et al., 2011; Kumar & others, 2014). Although, K-means may capture better clusters, it comes with some disadvantages: the number of clusters (K) must be known beforehand and chosen manually; this could be challenging since the number of clusters varies between different web pages, as it depends on the web page structure and content.

Various features can be extracted from the structure of web pages (e.g., DOM tree, location of elements, neighboring elements); in addition to that, the text content of web page elements can be a useful attribute for clustering elements. Methods to measure the content similarity such as paragraph and word similarity (e.g., Word2Vec and Doc2Vec (Le & Mikolov, 2014)) can be applied on each element to group elements with similar content together.

The clustering experiment was done using simple sorting and grouping by the XPath-based distances. However, more complex approaches can be performed to optimize the process of forming clusters; for example, using *Jenks natural breaks* to normalize and group close numbers together (Jiang, 2013). The method minimizes the variation within each similarly distances range, resulting in fewer clusters.

The clustering component was evaluated using a visual evaluation interface on a small sample of web pages due to the manual work involved. At present, the visual evaluation interface generates colors randomly; this can be improved further by displaying distinct colors and colors that do not look alike to the human-eye. Furthermore, in

the future, more research should be done to investigate automated clustering evaluation methods that fit the web information extraction task; once found, evaluation on a larger sample size becomes easy.

5.4 Global System Discussion

This section lists the contributions of this study and reflects on the results of each component. Furthermore, it outlines a multi-clustering nested challenge and provides ideas for the entity recognition component.

As previously mentioned, the system's goal is to extract and structure information from web pages generically; given web page as input, the system produces a structured output in JSON format.

The main contribution of this study is the proposed pipeline system design, consisting of the relevancy filter to filter out irrelevant web pages, feature extraction to study and extract candidate features from web pages, clustering component to group similar web page elements together, and finally, entity recognition to identify and link extracted clusters to entities. In addition to the global design, we have discussed each component in detail and proposed a list of candidate features which can be extracted from web pages (e.g., style-based, element and DOM information, and location-based features).

Furthermore, we have implemented and evaluated two components: relevancy filter and clustering. The relevancy filter performed well on classifying irrelevant web pages; however, it supports only English and Dutch. Training a larger language-agnostic multilingual model would improve the system further. In the clustering component, we provided a list of existing clustering methods focusing on techniques suited for web information extraction and text similarity. Moreover, we experimented with XPath-based Levenshtein distance features to form clusters of similar web page elements. Although the XPath-based clustering results are promising and show potential in this generic research approach, they can be improved further by leveraging the multiple web page features besides the XPath.

The similarity-based clustering performed well on clustering entity attributes; however, linking these attributes to an entity object remains challenging. We define this challenge as a multi-clustering nested task. Figure 35 illustrates an ideal clustering example of a pricing packages section. In a similarity-based clustering, the system will output price (C0) and period (C1) as entity attributes; however, to produce a fully

structured JSON, more work is needed. The attributes should be flattened and grouped vertically to form entity objects (C2, the packages), and since the objects relate to the same entity type, they must be grouped under one parent cluster (C3). In this example, C0 and C1 have no feature similarity (only the location of elements). Hence, we must look beyond the similarity-based clustering to link the entity attributes with their objects. A possible solution is to look at the DOM tree and investigate the parent-child relation; child elements represent the entity attributes, and parent element corresponds to the entity object.

Figure 35: Pricing packages multi-clustering example

Finally, the entity recognition component solves the last task to tackle the generic web information extraction. The evaluation of component remains for future work, but here we provide general ideas about the component and outline future directions. Given the clustering component's output clusters, the system assigns named entity attributes and entity object names. For this component to work correctly, the system must have a knowledge base containing information about possible entities and how they can be described.

The meta information of all steps carried out in the previous components can help the recognition process; therefore, it should be kept and passed on to this step. Given an entity knowledge base, components meta information and additional entity descriptions from open graphs (e.g., schema.org), the system should suggest possible entities automatically. In addition to that, the user may provide the system with extra entity definitions of interest. The entity definitions can be provided in a knowledge base as a set of rules or training examples (to train a machine learning model). The main challenge is to identify what features describe each entity. Consider this example of a pricing page that includes at least a *price* entity. The knowledge base should define what the price entity looks like; for example, a price is a number next to a currency sign, and the currency is a euro or dollar symbol. This way the system can resolve the entity attribute

name. The system may recognize a name for the entity object (e.g., object names are often located in the parent section and have a large bold font). This simple idea can generalize on many examples, including web tables. Figure 36 illustrates a web table example; when we apply the ideas presented above. The entity attributes will be the *name* and *color* because they are in bold font, centered in the middle, separated by lines, and use headline style capitalization.

Name	Color
Fred	Orange
Bob	Green
Lindy	Yellow

Figure 36: Simple web table example

In the future, we could study and list common entities (with features for each entity) that occur in web pages; by taking a random sample of web pages and annotating interesting entities. Finally, an easy way to model and integrate user specific entities in the system is needed.

6

Conclusions

This thesis reviewed different approaches on how to extract and structure information from web pages and proposed a component-based system design. The goal is to extract and recognize web information and convert it into a structured form (JSON format) in a generic way. The resulted structured information is beneficial and increasingly needed to support the business demands of building smart applications and system integrations (Section 2.9).

In general, some literature was found on the topic of web information extraction; however, no system or method could be found that can fulfill all the desired requirements for a generic and automatic extraction (Section 4.1). Extracting structured information from web data is challenging as it contains many web pages in different shapes and forms. Current methods are limited and fail to scale across different websites; some methods tackle parts of the challenge such as WebTables and TableMiner for HTML tables; others are limited to a pre-defined schema, require manual user input, visual-driven interaction, or training examples.

In the following subsections, we summarize the global system design and provide conclusions to each pipeline component. Furthermore, answer the research questions we defined at the beginning in Section 4.3 and outline future work suggestions.

6.1 Conclusions

This thesis proposed a novel component-based web information extraction system based on the well-established design science methodology (Wieringa, 2014). Given a collection of web pages as input, the system can extract and cluster web information generically.

The global design consists of four pipeline components; we studied literature and listed existing methods for each component.

Relevancy filter aims to clean out irrelevant websites based on custom binary classification models for two languages (English and Dutch). We trained and evaluated four models with different configurations on companies' dataset obtained from the Dutch Chamber of Commerce. The winning models did well on classifying irrelevant web pages (98.86% F1).

Feature extraction component aims at extracting helpful candidate features from the relevant websites, including XPath (Section 5.2.1). We studied and extracted the candidate features from three different web page representations: the visual presentation (what people see visually via browsers), source code, and DOM tree. The XPath feature was selected because of its simplicity in locating web page elements.

Clustering aims at grouping similar web page elements into clusters based on XPath-based Levenshtein distance. We listed existing clustering methods and focused on text similarity approaches, which are better suited for this research (Section 5.3.1). We extracted several features from the XPath-based Levenshtein distance and focused on the distance between target element and its parent to perform the clustering. Furthermore, a rule-based cleaning process was introduced to filter out unnecessary XPaths. Finally, the clustering was manually evaluated using a visual evaluation interface on three web pages (average purity of 61.39%).

Entity recognition is the last component in the pipeline; its goal is to link clusters with their corresponding entities. Although this component has remained for future work, we presented a knowledge-based approach to recognizing entities and proposed future research questions (see Section 5.4 and Section 4.3).

6.2 Research Questions

This thesis involves several research questions, as explained in Section 4.3. Below we discuss each of them based on the system global design and findings of this research.

***Q1**: How can irrelevant web pages be filtered out?*

In general, the relevancy of web pages is subjective and depends on the business use case. We solve this challenge by annotating a dataset of web pages and training a custom binary classifier. This same process can be adapted to different use cases (see Section 5.1).

***Q2**: What candidate features can be extracted from web pages?*

Web pages contain feature-rich semi-structured information. In Section 5.2.1, we studied and proposed a list of candidate features (style, dimensions and location, element information, DOM information, meta data, and other calculated features). These features could be leveraged to enhance the overall system performance.

***Q3**: How can XPaths be constructed to effectively locate web page elements?*

There are several ways to construct XPaths; we preferred the Full XPath, which traverses the DOM tree hierarchy from the root element of the web page (/html/) to the target element. This way, we get a standard and unified way to locate elements from web pages, making it easy to perform clustering experiments.

***Q4**: What existing clustering methods can be effectively used for clustering web page elements?*

Several existing methods were found from the literature, which have potential in clustering similar web page elements. We focused on character-based text similarity methods, including the Levenshtein distance, mainly because it fits well with the XPath feature. (see the complete list in Section 5.3.1).

***Q5**: What evaluation metrics can be used to measure the performance of clustering?*

Clustering evaluation is challenging and specific to the task. For this thesis, we adopted the purity and percentage purity to evaluate the quality of clusters. Purity measures the number of pure clusters among all clusters, while percentage purity estimates the percentage of relevant clusters.

6.3 Future work

Future research includes further optimization to the global system design to achieve better performance. Carrying out additional exploratory studies to the candidate features might be beneficial to uncover unseen patterns among web pages.

Training a large language-agnostic multilingual model can help the relevancy filter component in supporting additional languages. Hence, the ability to filter out web pages that include unseen languages correctly.

As for the clustering component to leverage and apply multiple features not only the XPath-based features. In the current implementation, all Levenshtein features had the same weights in the similarity distance calculation; it could be useful to apply a weighted distance by specifying different weights for each edit operations; for example, by giving larger weights to numbers than characters, since numbers refer to the element position in the XPath. Moreover, further research for automated clustering evaluation methods can come in handy to evaluate the same clustering experiment but on a larger sample size. Alternatively, improvements to the visual evaluation interface are needed to carry out the evaluation manually faster (e.g., print out a distinct color for each cluster). Finally, more research is needed for methods to group entity attributes with their objects.

Lastly, the entity recognition component needs more extensive study. The knowledge-based approach has good potential in resolving named entities. In this direction, a study of entities must be done on a sample of web pages or public knowledge bases (e.g., schema.org) to identify common entities and list their features. The following knowledge questions are listed for future research.

FQ1: How can candidate features be leveraged to identify entities?

FQ2: What is an effective method for constructing a knowledge base to hold entity representations?

FQ3: What is the best way to generate key-value structured information?

References

Abiteboul, S., Buneman, P., & Suciu, D. (2014). *Data on the Web: From Relations to Semistructured Data and XML*. https://books.google.com/books?hl=en&lr=&id=hYnUJY7FlBEC&oi=fnd&pg=PR5&dq=Abiteboul,+S.,+Buneman,+P.,+Suciu,+D.:+Data+on+the+Web:+from+relations+to+semistructured+data+and+XML.+Morgan+Kaufmann+Publishers+Inc.+San+Francisco,+CA,+USA+(1999)&ots=HB28AgMhqn&sig=b1gQCedplEMFzfLH9ZRyLh78vLI

Amigó, E., Gonzalo, J., Artiles, J., & Verdejo, F. (2009). A comparison of extrinsic clustering evaluation metrics based on formal constraints. *Information Retrieval*, *12*(4), 461–486.

Bartolini, I., Ciaccia, P., & Patella, M. (2002). String matching with metric trees using an approximate distance. *International Symposium on String Processing and Information Retrieval*, 271–283.

Bejarano, J., Bose, K., Brannan, T., Thomas, A., Adragni, K., Neerchal, N. K., & Ostrouchov, G. (2011). Sampling within k-means algorithm to cluster large datasets. *UMBC Student Collection*.

Berners-Lee, T., & others. (1998). *Semantic web road map*.

Bookstein, A., Kulyukin, V. A., & Raita, T. (2002). Generalized hamming distance. *Information Retrieval*, *5*(4), 353–375.

Brown, P. F., Della Pietra, V. J., Desouza, P. V, Lai, J. C., & Mercer, R. L. (1992). Class-based n-gram models of natural language. *Computational Linguistics*, *18*(4), 467–480.

Cafarella, M. J., Halevy, A., Wang, D. Z., Wu, E., & Zhang, Y. (2008). WebTables: Exploring the power of tables on the web. *Proceedings of the VLDB Endowment*, *1*(1), 538–549. https://doi.org/10.14778/1453856.1453916

Cai, D., Yu, S., Wen, J.-R., & Ma, W.-Y. (2003). *VIPS: a Vision-based Page Segmentation Algorithm VIPS: a Vision-based Page Segmentation Algorithm VIPS: a Vision-based Page Segmentation Algorithm*. https://www.microsoft.com/en-us/research/publication/vips-a-vision-based-page-segmentation-algorithm/

Califf, M. E., & Mooney, R. J. (1999). Relational Learning of Pattern-Match Rules for Information Extraction. In *aaai.org*. www.aaai.org

Chandrashekar, G., & Sahin, F. (2014). A survey on feature selection methods. *Computers & Electrical Engineering*, *40*(1), 16–28.

Chang, C.-H., & Hsu, C.-N. (1999). Automatic Extraction of Information Blocks Using PAT Trees. *Academia.Edu*. https://www.academia.edu/download/51171402/autoextract.pdf

Chang, C. H., Hsu, C. N., & Lui, S. C. (2003). Automatic information extraction from semi-structured Web pages by pattern discovery. *Decision Support Systems*, *35*(1), 129–147. https://doi.org/10.1016/S0167-9236(02)00100-8

De Mol, R., Bronselaer, A., Nielandt, J., & De Tré, G. (2015). Data driven XPath generation. *Advances in Intelligent Systems and Computing*, *322*, 569–580. https://doi.org/10.1007/978-3-319-11313-5˙50

Dice, L. R. (1945). Measures of the amount of ecologic association between species. *Ecology*, *26*(3), 297–302.

Fayzrakhmanov, R. R., Michels, C., & Neumann, M. (2018). Introduction to OXPath. *ArXiv Preprint ArXiv:1806.10899*.

Feng, J., Zou, L., Ye, O., & Han, J. (2020). Web2Vec: Phishing Webpage Detection Method Based on Multidimensional Features Driven by Deep Learning. *IEEE Access*, *8*, 221214–221224.

Fix, E. (1985). *Discriminatory analysis: nonparametric discrimination, consistency properties* (Vol. 1). USAF school of Aviation Medicine.

Furche, T., Gottlob, G., Grasso, G., Schallhart, C., & Sellers, A. (2013). OXPath: A language for scalable data extraction, automation, and crawling on the deep web. *The VLDB Journal*, *22*(1), 47–72.

Goebel, M., & Ceresna, M. (2009). Wrapper Induction. In *Encyclopedia of Database Systems* (pp. 3560–3565). Springer US. https://doi.org/10.1007/978-0-387-39940-9˙1160

Gomaa, W. H., Fahmy, A. A., & others. (2013). A survey of text similarity approaches. *International Journal of Computer Applications*, *68*(13), 13–18.

Grasso, G., Furche, T., & Schallhart, C. (2013). Effective web scraping with oxpath. *Proceedings of the 22nd International Conference on World Wide Web*, 23–26.

Hotho, A., Ba Nguyen, D., Cheatham, M., Martinez-Rodriguez, J. L., Hogan, A., & Lopez-Arevalo, I. (2016). Information Extraction meets the Semantic Web: A Survey; 2 Anonymous Reviewers Open review(s). In *content.iospress.com*. http://prefix.cc.

Hu, D. D., & Meng, X. F. (2004). Automatically extracting Web data using tree structure. *Jisuanji Yanjiu Yu Fazhan/Computer Research and Development*, *41*(10), 1607–1613.

Huang, L. (2011). *Concept-based text clustering*. University of Waikato.

Introduction to robots.txt - Search Console Help. (2020). https://support.google.com/webmasters/answer/6062608?hl=en

Ioffe, S. (2006). Probabilistic linear discriminant analysis. *European Conference on Computer Vision*, 531–542.

Izenman, A. J. (2013). Linear discriminant analysis. In *Modern multivariate statistical techniques* (pp. 237–280). Springer.

Jaccard, P. (1912). The distribution of the flora in the alpine zone. 1. *New Phytologist*, *11*(2), 37–50.

Jiang, B. (2013). Head/tail breaks: A new classification scheme for data with a heavy-tailed distribution. *The Professional Geographer*, *65*(3), 482–494.

Jones, R., McCallum, A., Nigam, K., & Riloff, E. (1999). Bootstrapping for text learning tasks. *IJCAI-99 Workshop on*

Text Mining: Foundations, Techniques and Applications, *1*(7).
JSON. (2020). https://www.json.org/json-en.html
Jundt, O., & Van Keulen, M. (2013). Sample-based xpath ranking for web information extraction. *8th Conference of the European Society for Fuzzy Logic and Technology (EUSFLAT-13)*.
Khalid, S., Khalil, T., & Nasreen, S. (2014). A survey of feature selection and feature extraction techniques in machine learning. *2014 Science and Information Conference*, 372–378.
Klettke, M., Störl, U., & Scherzinger, S. (2015). Schema Extraction and Structural Outlier Detection for JSON-based NoSQL Data Stores. *Undefined*.
Kosub, S. (2019). A note on the triangle inequality for the Jaccard distance. *Pattern Recognition Letters*, *120*, 36–38.
Krishnan, R., Parashar, A., & Sarkar, S. (2019). *Automatic generation of structured data from semi-structured data*. Google Patents.
Kumar, S., & others. (2014). Efficient K-Mean Clustering Algorithm for Large Datasets using Data Mining Standard Score Normalization. *Int. J. Recent Innov. Trends Comput. Commun.*, *2*(10), 3161–3166.
Laender, A. H. F., Ribeiro-Neto, B. A., Da Silva, A. S., & Teixeira, J. S. (2002). A brief survey of Web data extraction tools. In *SIGMOD Record* (Vol. 31, Issue 2, pp. 84–93). https://doi.org/10.1145/565117.565137
Lanthaler, M., & Gütl, C. (2012). On using JSON-LD to create evolvable RESTful services. *Proceedings of the Third International Workshop on RESTful Design*, 25–32.
Le, Q., & Mikolov, T. (2014). Distributed representations of sentences and documents. *International Conference on Machine Learning*, 1188–1196.
Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, *10*(8), 707–710.
Li, B., & Han, L. (2013). Distance weighted cosine similarity measure for text classification. *International Conference on Intelligent Data Engineering and Automated Learning*, 611–618.
Lin, Y., Jun, Z., Hongyan, M., Zhongwei, Z., & Zhanfang, F. (2018). A method of extracting the semi-structured data implication rules. *Procedia Computer Science*, *131*, 706–716. https://doi.org/10.1016/j.procs.2018.04.315
Ling, C. X., Huang, J., & Zhang, H. (2003). AUC: a better measure than accuracy in comparing learning algorithms. *Conference of the Canadian Society for Computational Studies of Intelligence*, 329–341.
Lipkus, A. H. (1999). A proof of the triangle inequality for the Tanimoto distance. *Journal of Mathematical Chemistry*, *26*(1), 263–265.
Liu, W., Meng, X., & Meng, W. (2006). Vision-based Web Data Records Extraction. *WebDB*.
MacQueen, J., & others. (1967). Some methods for classification and analysis of multivariate observations. *Proceedings of the Fifth*

Berkeley Symposium on Mathematical Statistics and Probability, *1*(14), 281–297.

Manning, C. D., Raghavan, P., & Schütze, H. (2008). Flat clustering. In *Introduction to Information Retrieval* (pp. 321–345). Cambridge University Press. https://doi.org/10.1017/CBO9780511809071.017

McCallum, A., & Nigam, K. (1999). Text classification by bootstrapping with keywords, EM and shrinkage. *Unsupervised Learning in Natural Language Processing*.

Miao, J., & Niu, L. (2016). A survey on feature selection. *Procedia Computer Science*, *91*, 919–926.

Monge, A., & Elkan, C. (1997). *An efficient domain-independent algorithm for detecting approximately duplicate database records*.

Moon, T. K. (1996). The expectation-maximization algorithm. *IEEE Signal Processing Magazine*, *13*(6), 47–60.

Muslea, I., Minton, S., & Knoblock, C. (1999). Hierarchical approach to wrapper induction. *Proceedings of the International Conference on Autonomous Agents*, 190–197. https://doi.org/10.1145/301136.301191

Needleman, S. B., & Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, *48*(3), 443–453.

Rao, G. A., Srinivas, G., VenkataRao, K., & Prasad Reddy, P. (2018). A partial ratio and ratio based fuzzy-wuzzy procedure for characteristic mining of mathematical formulas from documents. *IJSC—ICTACT J Soft Comput*, *8*(4), 1728–1732.

Raymer, M. L., Punch, W. F., Goodman, E. D., Kuhn, L. A., & Jain, A. K. (2000). Dimensionality reduction using genetic algorithms. *IEEE Transactions on Evolutionary Computation*, *4*(2), 164–171.

Reynolds, D. A. (1993). *A Gaussian mixture modeling approach to text-independent speaker identification.*

Sidorov, G., Gelbukh, A., Gómez-Adorno, H., & Pinto, D. (2014). Soft similarity and soft cosine measure: Similarity of features in vector space model. *Computación y Sistemas*, *18*(3), 491–504.

Sint, R., Schaffert, S., Stroka, S., & Ferstl, R. (2009). Combining unstructured, fully structured and semi-structured information in semantic wikis. *CEUR Workshop Proceedings*, *464*, 73–87.

Smith, T. F., Waterman, M. S., & others. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology*, *147*(1), 195–197.

Sorzano, C. O. S., Vargas, J., & Montano, A. P. (2014). A survey of dimensionality reduction techniques. *ArXiv Preprint ArXiv:1403.2877*.

Sporny, M., Longley, D., Kellogg, G., Lanthaler, M., & Lindström, N. (2014). JSON-LD 1.0. *W3C Recommendation*, *16*, 41.

Sripada, S. C., & Rao, M. S. (2011). Comparison of purity and entropy of k-means clustering and fuzzy c means clustering. *Indian Journal of Computer Science and Engineering*, *2*(3), 343–346.

Thompson, B. (1984). *Canonical correlation analysis: Uses and interpretation* (Issue 47). Sage.
Thompson, B. (2005). Canonical correlation analysis. *Encyclopedia of Statistics in Behavioral Science*.
Tipping, M. E., & Bishop, C. M. (1999). Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *61*(3), 611–622.
Trieschnigg, D., Tjin-Kam-Jet, K., & Hiemstra, D. (2012). Ranking XPaths for extracting search result records. *CTIT Technical Report*. https://research.utwente.nl/files/5092970/techrep.2012.trieschnig g.pdf
Tversky, A. (1977). Features of similarity. *Psychological Review*, *84*(4), 327.
Varlamov, M. I., & Turdakov, D. Y. (2016). A survey of methods for the extraction of information from Web resources. *Programming and Computer Software*, *42*(5), 279–291.
Vijaymeena, M. K., & Kavitha, K. (2016). A survey on similarity measures in text mining. *Machine Learning and Applications: An International Journal*, *3*(2), 19–28.
Vogels, T., Ganea, O.-E., & Eickhoff, C. (2018). Web2text: Deep structured boilerplate removal. *European Conference on Information Retrieval*, 167–179.
Wieringa, R. (2014). *Design science methodology for information systems and software engineering*. https://books.google.com/books?hl=en&lr=&id=xLKLBQAAQ BAJ&oi=fnd&pg=PR5&dq=Design+Science+Methodology+for +Information+Systems+and+Software+Engineering&ots=bWBr cvYnGv&sig=bvXgVKSqzaZH1BjcPUQMXXgZg1o
Winkler, W. E. (1990). *String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage.*
Wold, S., Esbensen, K., & Geladi, P. (1987). Principal component analysis. *Chemometrics and Intelligent Laboratory Systems*, *2*(1–3), 37–52.
Yuchen Lin, B., Sheng, Y., Vo, N., & Tata, S. (2020). FreeDOM: A Transferable Neural Architecture for Structured Information Extraction on Web Documents. *Dl.Acm.Org*, 2020. https://doi.org/10.1145/3394486.3403153
Zhang, Shuo, & Balog, K. (2020). *Web Table Extraction, Retrieval and Augmentation: A Survey*. http://arxiv.org/abs/2002.00207
Zhang, Suzhi, & Shi, P. (2009). An efficient wrapper for Web data extraction and its application. *Proceedings of 2009 4th International Conference on Computer Science and Education, ICCSE 2009*, 1245–1250. https://doi.org/10.1109/ICCSE.2009.5228403
Zhang, Z. (2009). Start small, build complete: Effective and efficient semantic table interpretation using tableminer. In *semantic-web-journal.org* (Vol. 1). http://semantic-web-journal.org/system/files/swj668.pdf
Zhao, C., & Sahni, S. (2019). String correction using the Damerau-

Levenshtein distance. *BMC Bioinformatics*, *20*(11), 1–28.
Zhao, Y., & Karypis, G. (2001). *Criterion functions for document clustering: Experiments and analysis*.

Appendices

A. Google Rich Results Features

Feature	Description
Article	A blog article displayed with a highlighted headline and larger images.
Book	Actions to enable users to buy books directly.
Breadcrumb	Navigation to move users to a specific content position
Carousel	Rich results that display in a gallery view
Course	Display course title, provider, and a short description for educational courses
Critic review	A snippet review from a larger article about a book, movie, or local business.
Dataset	Dataset files that appear in Google Dataset Search
EmployerAggregateRating	An evaluation review of a hiring process displayed from many users to enhance a job search experience.
Event	Shows a list of events such as festivals and concerts that start at a particular time and place in order to help people attend and small business to increase their sales.
Local Business	Show information about a local business including opening hours, ratings, google maps directions, and actions to book appointments or order products directly.
Logo	Shows an organization logo in search results and on the side as part of google knowledge panel.
Podcast	Shows podcasts in a playable link in Google Search and Google Podcasts app.

Product	Information about a product, including price, stocks availability, and review ratings.
Recipe	Shows what are the recipe ingredients, cooking time and temperature, and calories.
Review snippet	A short description, review or rating from a review site about a book, recipe, product, software app, and local business.
Software App	Information about a software app such as rating, reviews and a link to download the app.
Subscription and paywalled content	Show a paywalled content on a site.
Video	Shows information about a video or live-stream content with the option to enable users to play it right away.
Other features, including: Fact Check, FAQ, Home Activities, How-to, Image License, JobPosting, Job Training, Speakable, Sitelinks Search box, Q&A, Movie, and Estimated salary.	

Table 15: Google Rich Results features[29] explained

[29] https://developers.google.com/search/docs/guides/search-gallery

B. Vision-based Page Segmentation

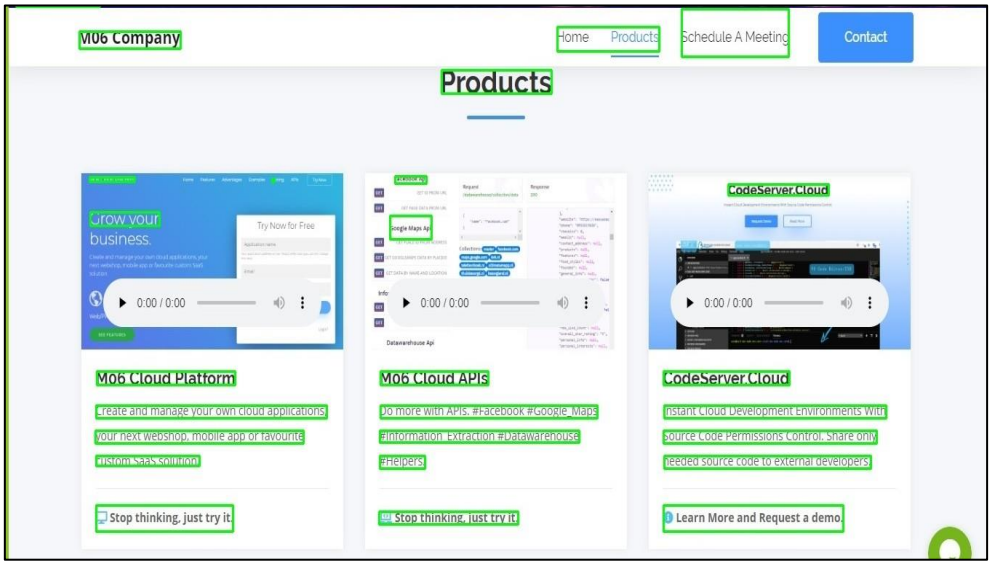

Figure 37: An example of web data extraction using a vision-based technique

C. Web Pages Input and Output

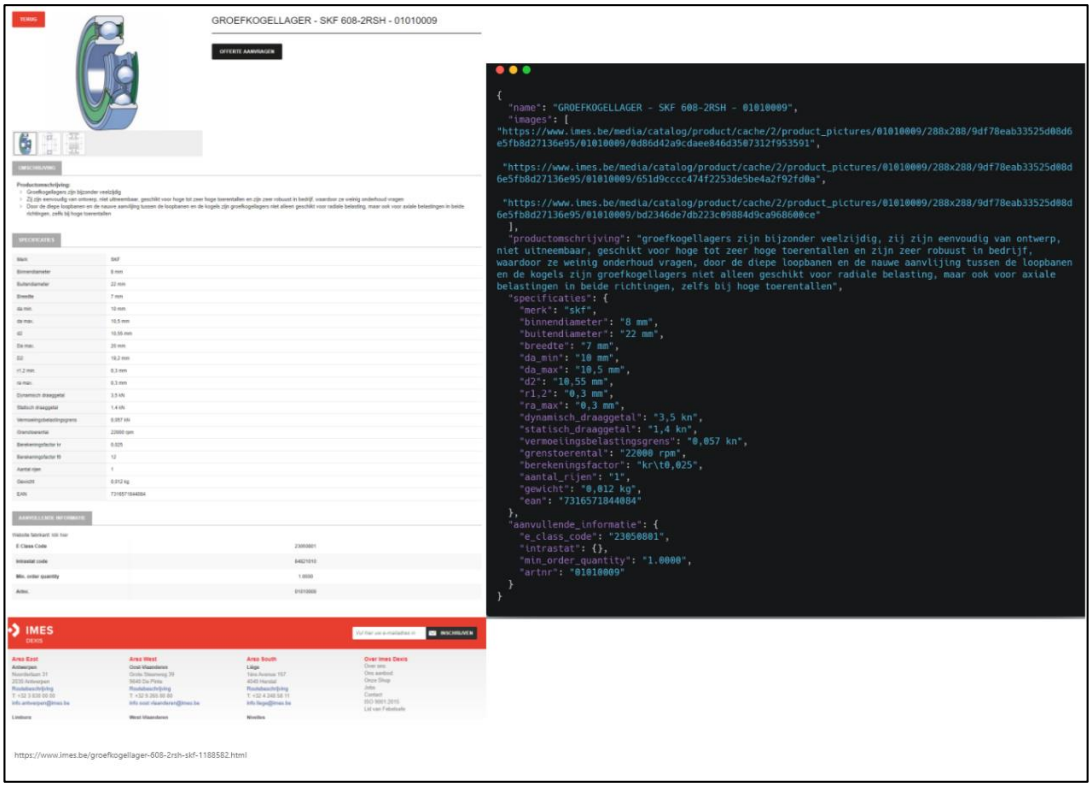

Figure 38: Example of webshop product detail page and it's structured output

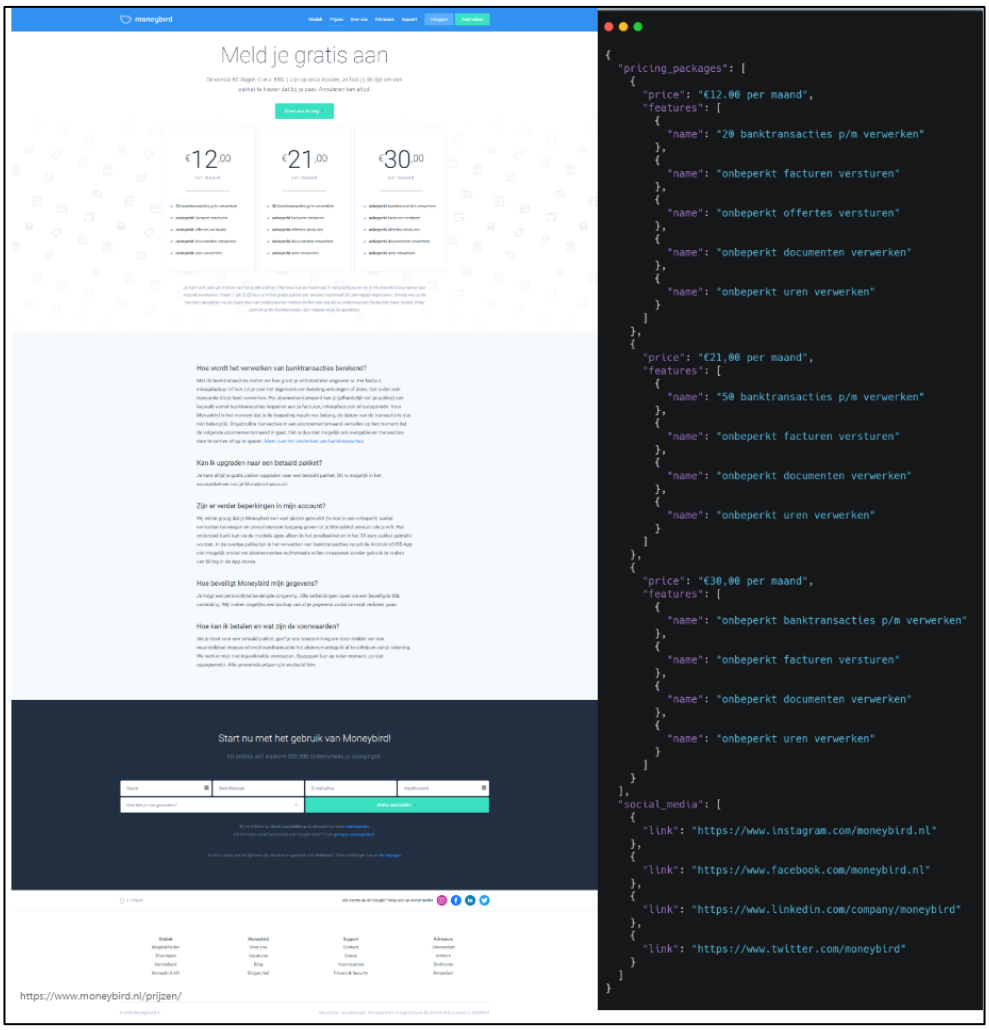

Figure 39: Example of pricing packages web page and it's structured output

D. Website Categories

Business informative website

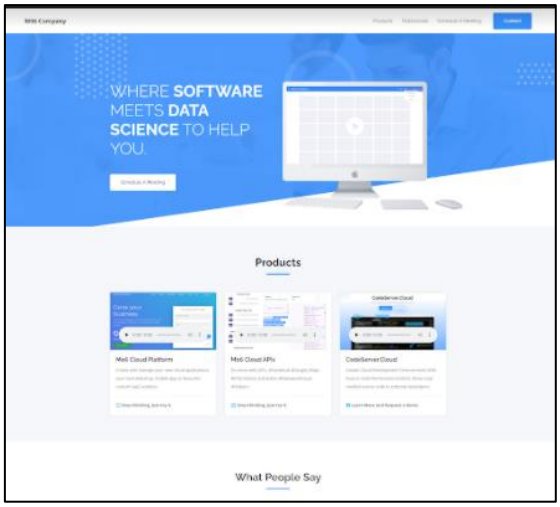

Figure 40: Business informative websites: present information and media content

Ecommerce websites

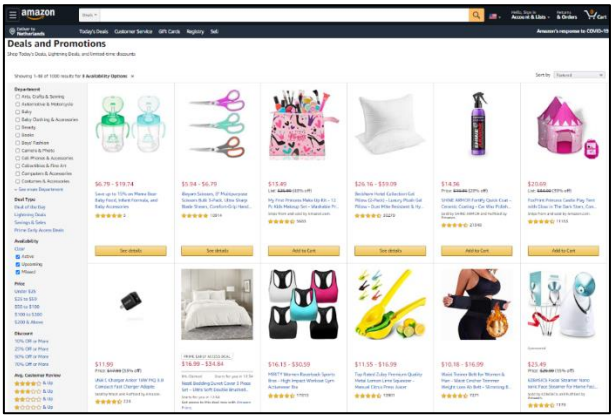

Figure 41: Ecommerce websites/webshops: sell many products online

Pricing packages included websites

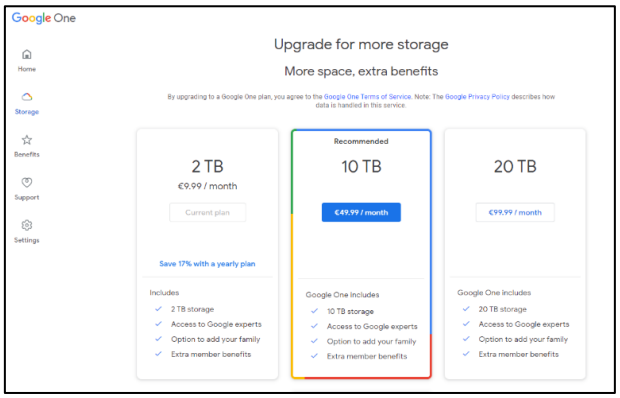

Figure 42: Pricing packages included websites: include few-tiered pricing packages

E. Relevancy Filter Annotation Examples

Good Examples

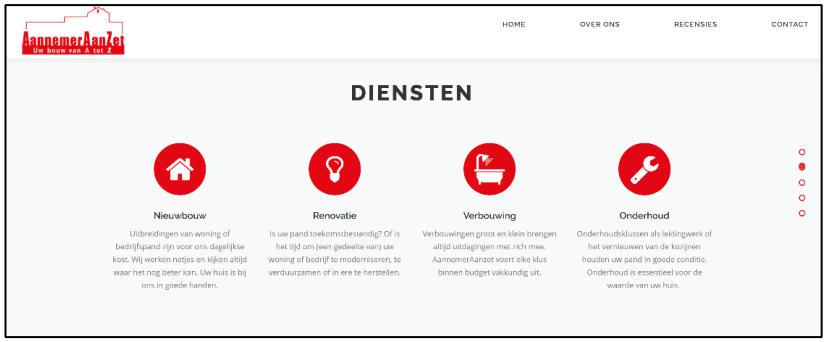

Figure 43: Accepted example 1 (aannemeraanzet.nl)

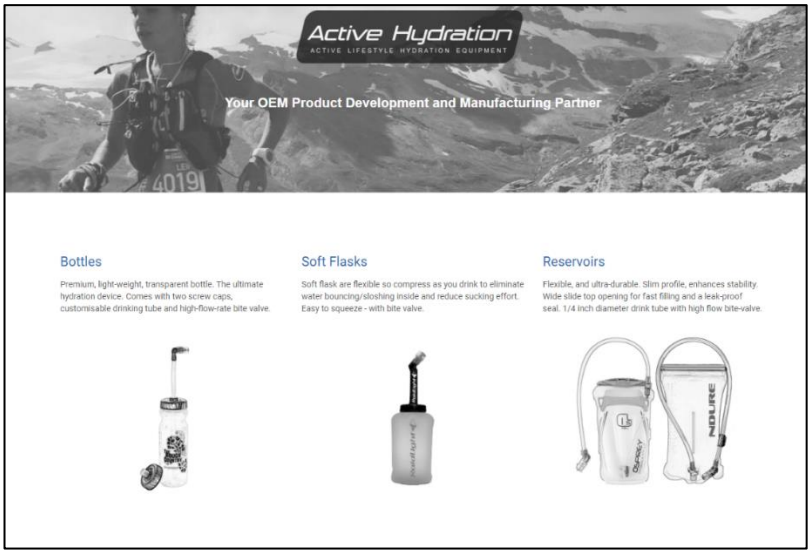

Figure 44: Accepted example 2 (active-hydration.com)

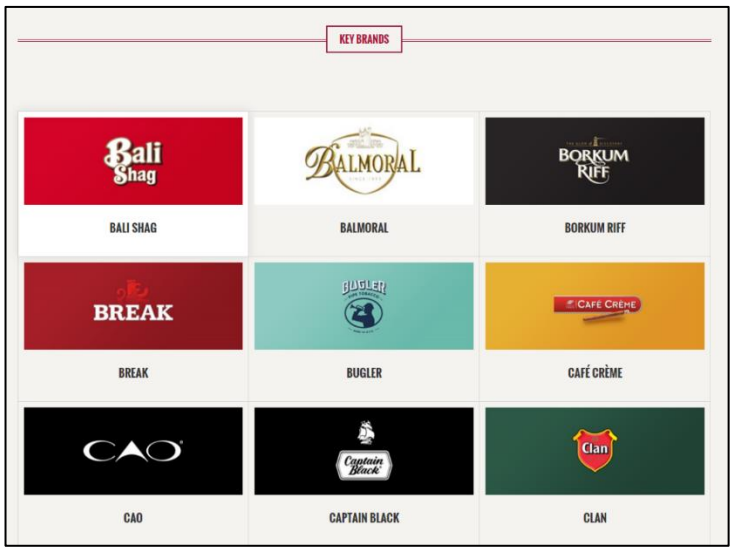

Figure 45: Accepted example 3 (agiocigars.com)

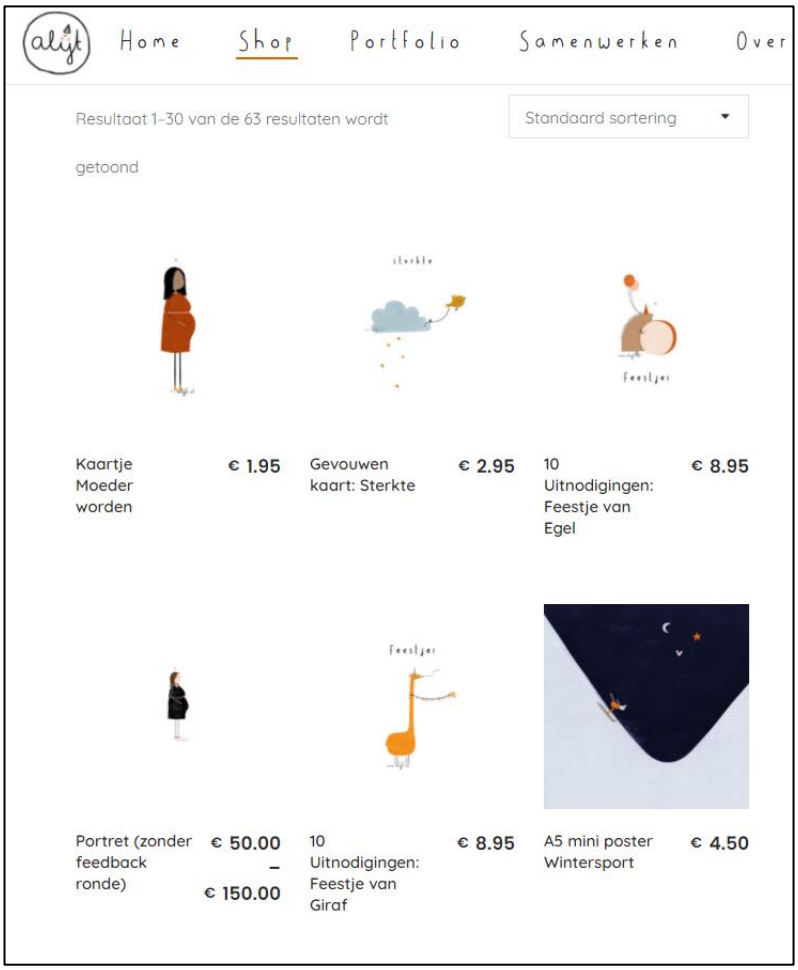

Figure 46: Accepted example (alijt.nl)

Bad Examples

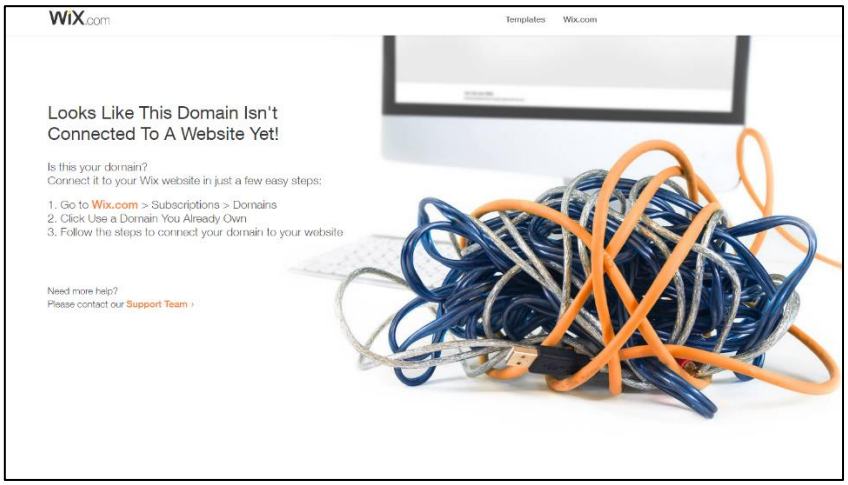

Figure 47: Rejected example 1 (12motiv8.nl)

Figure 48: Rejected example 2 (180gradenom.com)

Figure 49: Rejected example 3 (1816media.nl)

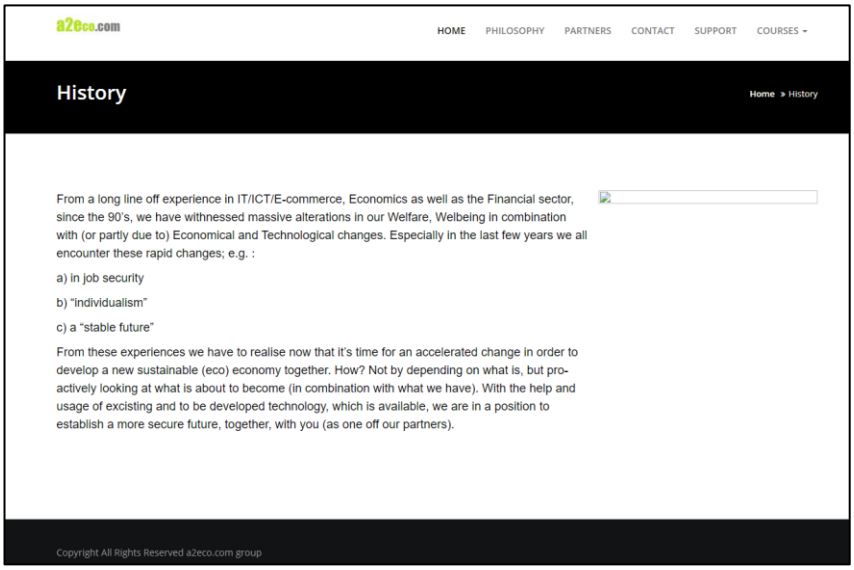

Figure 50: Rejected example 4 (a2eco.com)

Figure 51: Rejected example 5 (aceauctions.eu)

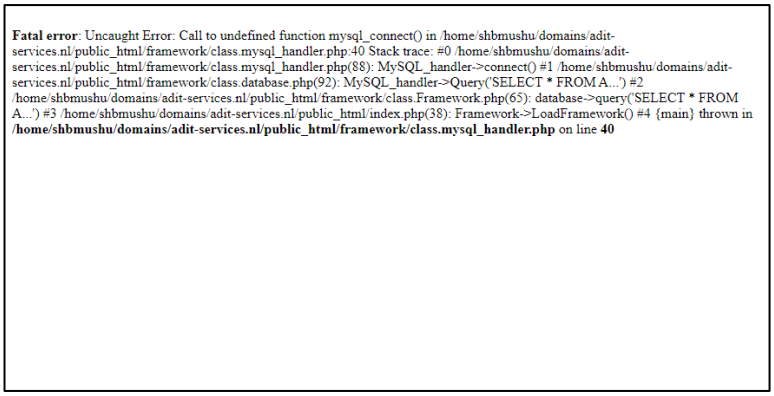

Figure 52: Rejected example 6 (adit-services.nl)

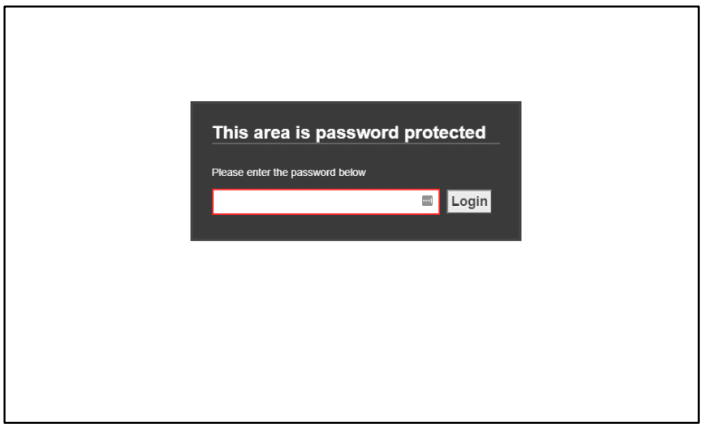

Figure 53: Rejected example 7 (admarsol.com)

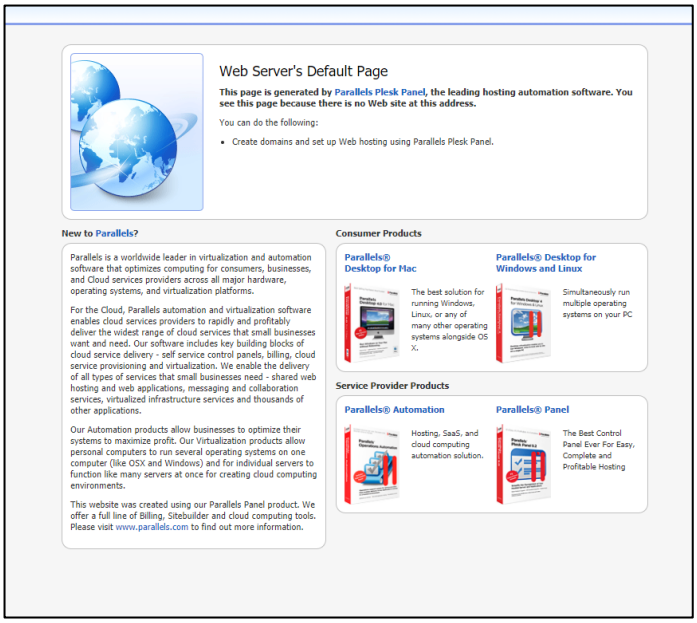

Figure 54: Rejected example 8 (adrenalinemedia.net)

F. XPath-based clustering examples

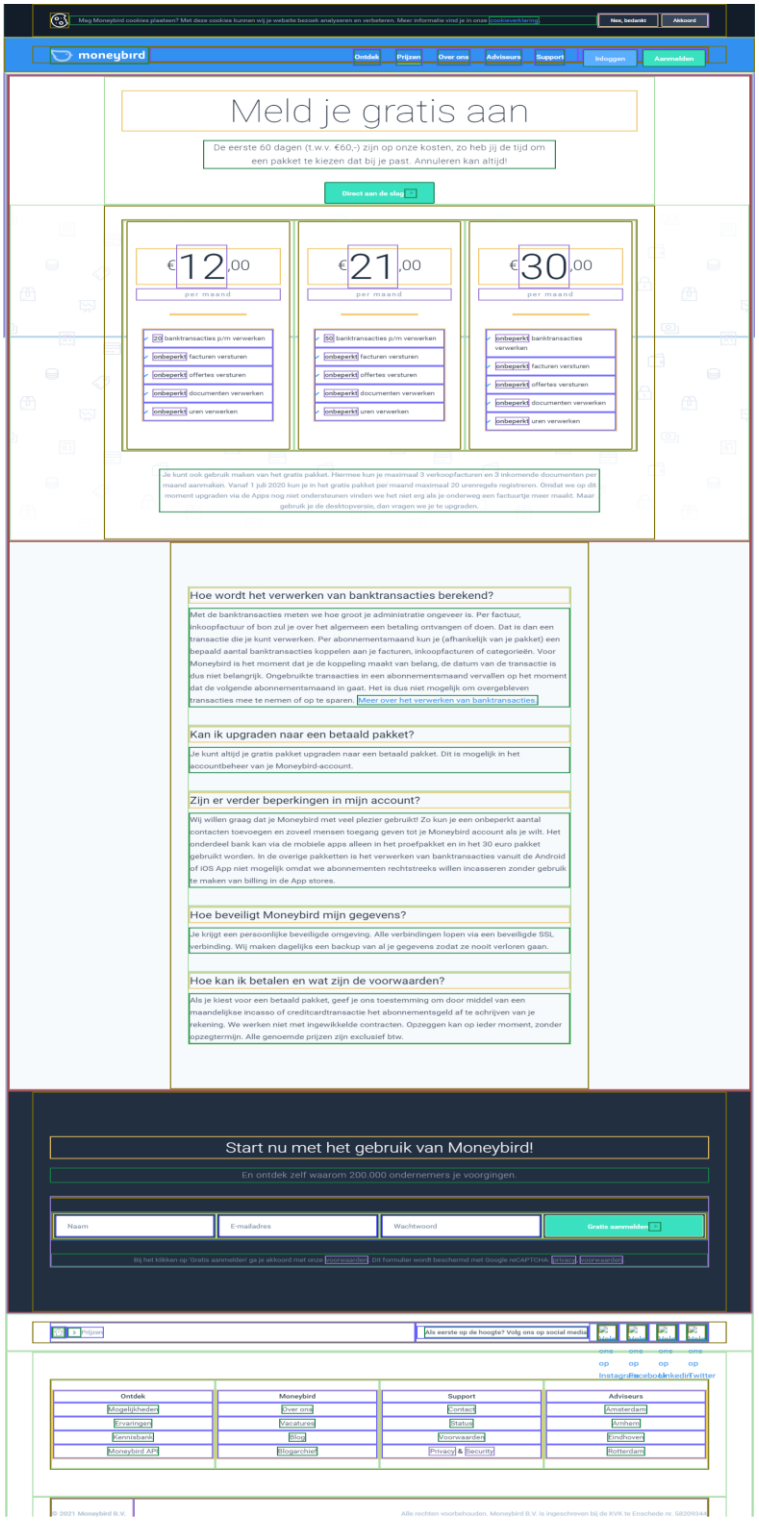

Figure 55: XPath-based clustering performed on pricing packages page

Figure 56: XPath-based clustering performed on products page

Figure 57: XPath-based clustering performed on brands page