FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Machine Learning Approach for Web Ranking Identification based on Visual Features

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Florian Eckerstorfer B.Sc.
Matrikelnummer 0725781

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Univ.-Prof. Mag. Dr. Reinhard Pichler
Mitwirkung: Dr.techn. Dipl.-Ing. Ruslan Fayzrakhmanov M.Sc.

Wien, 25. November 2017

Florian Eckerstorfer	Reinhard Pichler

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Machine Learning Approach for Web Ranking Identification based on Visual Features

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Florian Eckerstorfer B.Sc.
Registration Number 0725781

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Univ.-Prof. Mag. Dr. Reinhard Pichler
Assistance: Dr.techn. Dipl.-Ing. Ruslan Fayzrakhmanov M.Sc.

Vienna, 25th November, 2017

Florian Eckerstorfer	Reinhard Pichler

Erklärung zur Verfassung der Arbeit

Florian Eckerstorfer B.Sc.
Sobiesgkigasse 12/9, 1090 Vienna

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 25. November 2017

Florian Eckerstorfer

Danksagung

Zuerst möchte ich meinem Betreuer Univ.-Prof. Mag. Dr. Reinhard Pichler für seinen Rat und die Hilfe bei dieser Diplomarbeit danken. Außerdem möchte ich Ruslan Fayzrakhmanov für seine Beratung und Unterstützung, während ich diese Arbeit geschrieben habe, danken.

Ich möchte auch meinen Eltern, Heidi und Andreas, für ihre Unterstützung mein ganzes Leben lang danken und dafür, dass sie mich ermutigt haben, ein Diplomstudium zu beginnen. Zuletzt gilt mein Dank allen meinen Freunden, besonders Karin, Birgit, Franziska, Stefan und Andrea für ihre Unterstützung und ihre aufmunternden Worte.

Acknowledgements

First I would like to thank my supervisor Univ.-Prof. Mag. Dr. Reinhard Pichler for his advice and help for this thesis. Additionally I want to thank Ruslan Fayzrakhmanov for his advice, guidance and support while researching and writing.

Moreover I would also like to thank my parents, Heidi and Andreas, for their love and support my whole life and their encouragement to pursue a masters degree. Lastly I want to thank all my friends, but especially Karin, Birgit, Franzsika, Stefan and Andrea for their support and words of encouragement while writing this thesis.

Kurzfassung

In dieser Arbeit beschreiben wir Wres, ein System um Informationen in Web Rankings zu identifizieren und zu extrahieren. Web Rankings sind geordnete Listen von Webobjekten, die häufig auf Webseiten verwendet werden. Unser System benutzt visuelle Eigenschaften, um diese Objekte zu identifizieren, indem es deren gemeinsame Designsprache bedient. Daher können wir mit einer geringen Menge an Beispielen einen Klassifizierer trainieren und damit Daten von Webseiten mit einem anderen Quelltext extrahieren.

Wir präsentieren auch eine Erweiterung für Google Chrome, um Beispiele von Web Rankings zu annotieren. Die Wres Annotation Erweiterung haben wir mit einer großzögigen Open Source Lizenz veröffentlicht. Sie basiert auf W3C Standards. Durch die Annotierung direkt in einem kommerziellen Browser können wir unser Annotierungswerkzeug rasch an neue Versionen des Browsers anpassen und stets die gleiche Browser-Version wie ein Großteil der Benutzer_innen verwenden. Mit unserem Annotierungswerkzeug haben wir ein Trainingsset erstellt, das Beispiele von verschiedenen, häufig gebrauchten Layouts von Web Rankings enthält. Wir stellen außerdem ein Modell vor, um Web Rankings formal zu beschreiben.

Semi-supervised Machine Learning Techniken werden benutzt um Klassifizierer für die Web Ranking Identifizierung zu erstellen. Für das Training und die Ausführung des Klassifizierers setzen wir Weka ein. Unsere Arbeit enthält eine Beschreibung von supervised und semi-supervised Techniken. Zusätzlich beschreiben wir die folgenden Machine Learning Techniken im Detail: Decision Tree, Random Forests, PART Rule-learning algorithm und Support Vector Machines.

Wir werten unsere Klassifizierer aus und vergleichen die Resultate der verschiedenen Machine Learning Techniken. In unserer Auswertung zeigen wir, dass unser System am besten für Label funktioniert, die mehrere Male wiederholt werden und markante visuelle Kennzeichen besitzen. Mit dem Random Forest Algorithmus können wir eine Präzision von 0.9427 für wiederholende Label und eine Präzision von 0.7573 für nicht wiederholende Label erzielen. Zusätzlich vergleichen wir unser System mit zwei kommerziell verfügbaren, proprietären Web Extraction Systemen: Diffbot und Import.io. Die Analyse der Leistungen dieser beiden Systeme hat gezeigt, dass Diffbot keine geeignete Lösung ist um Web Rankings zu extrahieren, da es nur Daten von 2% der Webseiten in unserem Datenset Rankings extrahieren konnte. Import.io kann von 53% der Webseiten in unserem Datenset Web Rankings extrahieren; die Leistung ist hier abhängig von der Art der Web Rankings. Wir haben beobachtet, dass unser System mit 71% korrekt extrahierten Web Rankings

die Web Data Extraction Systeme Diffbot und Import.io übertrifft, indem wir uns auf Web Rankings spezialisieren.

Abstract

In this thesis, we introduce an approach to web ranking extraction, which is based on the analysis of visual features. Web rankings are ordered lists of web objects that are commonly used on websites. Our system uses visual features to identify these objects based on the fact that web rankings share a strong visual language. Therefore we can efficiently train a classifier with a small set of examples and extract data from websites with a different source code.

We also present an extension for Google Chrome to annotate examples of web rankings. The Wres Annotation Extension is available publicly under a permissive Open Source license and relies on W3C standards. Using our annotation tool we build a training set containing examples using commonly used web ranking layouts, namely table, list, simple list, grid and tiling layouts. We also present a model to formally describe web rankings.

Semi-supervised machine learning techniques are used to build classifiers for web ranking identification. We use Weka to train and run our classifiers throughout the thesis. Our thesis includes description of supervised and semi-supervised techniques. In our comparison of Machine Learning algorithms we include BayesNet, NaiveBayes, LibSVM, MultilayerPerceptron, SimpleLogistic, IBk, KStar, LWL, DecisionTable, JRip, OneR, PART, ZeroR, DecisionStump, HoeffdingTree, J48, LMT, RandomForest, RandomTree, and REPTree.

We evaluate our classifiers and compare the results of different machine learning techniques. In our evaluation we found that our system works best for labels that repeat multiple times and have a distinctive visual representation. With the Random Forest algorithm we could achieve a precision of 0.9427 for repeating labels and a precision of 0.7573 for non-repeating labels. Addtionally we compare our system to two commercial available, proprietary web extraction systems: Diffbot and Import.io. We analyze the performance of both these systems and find that Diffbot is not an appropriate solution to extract web rankings since it can only extract data from 2% of rankings in our dataset. Import.io can extract web rankings from 53% of web pages in a our dataset, with better performance for some types of web rankings and worse for others. We found that our system outperforms Diffbot and Import.io with 71% correctly extracted web rankings by specialising solely on web rankings.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

The thesis is dedicated to problems of identifying web rankings – ordered lists of items on web pages based on their visual appearance. There are many websites which provide ranked information as lists or tables, be it list of the best coffee shops in a town, the best movies of 2017, the best universities, etc. When a person searches for *best coffee shop in vienna* a typical search engine returns relevant pages which contain the keywords specified. The person then has to manually navigate through the result pages to find the coffee shop that most sites agree on to be the best. With the use of techniques presented in this thesis a search engine can identify, extract and index web rankings to provide the user with aggregated information according to its request. Following our example, the search engine thus can return an aggregated list of the best coffee shops in Vienna to the user, using a huge index of identified rankings, that wouldn't be possible with manual effort. Especially when considering the use of search engines on mobile or through non-visual search assistants (such as Siri[1], Cortana[2] or Google Now[3]) a single aggregating ranking is preferrable for the person as it reduces the time required to find the coffee shop they want to go. A user typically does not search for a website, but for information. The usability and value of a search engine can be increased if it can present the desired information directly instead of presenting the person a list of websites that contain the information. Amazon Echo[4], for example, does not even include a visual interface, a person interacts with the device only using natural language. Echo needs to return information, because it has no capabilities to direct the person to a web page. For example, if a person is asking Echo for the best coffee shop in Vienna it needs to return the name and address of a coffee shop instead of a website containing this information.

In this thesis we present an approach to identify and extract rankings from web pages using visual information. By using machine learning techniques we are able to

[1] https://www.apple.com/ios/siri/
[2] http://www.windowsphone.com/en-US/how-to/wp8/cortana/meet-cortana
[3] https://www.google.com/landing/now/
[4] http://www.amazon.com/oc/echo/

train classifiers with a relatively small data set to identify and extract information from previously unseen web pages. The analysis of visual information allows the classifier to leverage some aspects of a rendered web page, which human users naturally perceive and which help them to understand the web page content and identify relevant data on the web page layout.

1.1 Motivation

There is more data on the World Wide Web than ever before. According to Netcraft there have been nearly 900 million active sites in August 2015 [Net15] and additionally the amount of data on each individual site is increasing constantly. For example, the number of articles on the English Wikipedia has more than doubled from 2006 to 2008 and has doubled once again since 2009 [Wik15]. There was a brief period towards the end of the 2000s, in an era that was often called Web 2.0 were most new Internet startups offered access to their data using web services. After 2010 new web services rarely offer full access to all their data using an API and companies like Twitter restricted access to their data over time [Tso12] [TL14] [Del13]. Thus it becomes increasingly useful to find mechanisms to extract data from websites on a large scale with as little human interaction as possible.

Information extracted from web sites has dozens of application in different fields. Businesses use Business Intelligence and Competitive Intelligence to quickly react to changes in the market. [Got05] identifies five critical success factors:

- identify changes in the market early,
- anticipate competitor activities,
- recognize new and potential competitors,
- learn from errors and success stories of competitors, and
- validate and enhance own strategic goals, processes and products.

Web Data Extraction is an important means for accessing data sources in Business Intelligence systems, as this information is most likely not available in a structured data format (i.e., the source code)

A completly different application is in Web Accessibilty as presented in [Fay13]. Data extracted from websites can be used to improve the experience of blind and visual impaired people by providing additional context. Web Data Extraction techniques are also used during the development of websites. Software like Selenium[5] allows developers to record and reply navigation sequences and verify the existence of certain elements on the web page. Additional applications for web data extraction have been presented in a survey [FMFB14].

[5] `http://seleniumhq.org`

Web Rankings are particulary useful to extract since they provide, in addition to the actual content, context and order and therefore increase the value of the information. Because of this they are particulary popular on the web. Information without order is often less useful to users, since it is missing context. When a customer wants to buy a televison set and sees a random list of TVs it will likely overwhelm them. Ordering the list by some criteria, e.g., price, quality, or size provides context to the customers and allows them to better understand the data. The popularity of web rankings make them even more interesting to us, because by finding an effective way to extract web rankings we can provide a solution for a wide set of problems.

Many Web Data Extraction systems described in survey papers such as [CKGS06] or [FMFB14] use textual or structural information to extract information from web pages. However, the design language of related information often has more similarities than the structure or the textual content. General purpose search engines like Google[6], Bing[7] or DuckDuckGo[8] all display search results in a visually similar way. The forms to search for a flight on airline websites are also visually similar to each other, for example, the websites of British Airways[9], Lufthansa[10] or Vueling[11] use a very similar layout for their search forms. The same is true for many different types of web sites or at least certain elements (e.g., search form, result lists) on a web site.

In the mid 2000s website developers started to use a technology called Ajax [Gar05] to improve the user experience of websites and web applications. With Ajax the HTML file does not contain the full content of the web page and JavaScript is used to download content when it is, either explicitly or implicitly, requested by the user. For example, when the user visits a web page with a long list of images only the first 20 images are included in the HTML file. If the user scrolls down the page and reaches the end of the list the next 20 images are requested from the server. In recent years this has been brought to its extreme conclusion with Single Page Applications (SPA) [MP13]. A SPA consists of a single HTML file which only contains links to its JavaScript and CSS files. In order to extract content from this kind of web sites we need to download and render the applications JavaScript code in a web browser.

In this thesis, we demonstrate that using only a very small set of annotated web sites as examples we can extract similar data from a large amount of other web sites using visual information. This is achieved with the use of an effective classifier for web rankings due to a relatively small number of different ways to display ranked data on a website. Since we render HTML in a web browser and execute JavaScript we can also extract information from websites using Ajax and from Single Page Applications.

[6] https://www.google.com
[7] https://www.bing.com
[8] https://duckduckgo.com
[9] http://www.britishairways.com
[10] http://www.lufthansa.com/online/portal/lh/at/homepage
[11] https://www.vueling.com/es

1.2 State of the Art and Related Work

In this section, we give an overview of existing methods and techniques relevant to the problem addressed in this thesis.

One of the earliest systems which uses machine learning techniques in information extraction is WIEN [Kus00]. The system utilizes *inductive learning techniques* to automatically label training web pages and reduces human envolvement. However, one of the major short-comings of WIEN in real-world scenarios was that it could not handle missing values. Other systems, including Rapier (Robust Automated Production of Information Extraction Rules) [CM03] [CM99] and WHISK [Sod99] use rule-based techniques to build classifiers. Unlike the previsouly mentioned systems, SoftMealy [HD98] was specifically designed to extract data from the web. Using a generalization algorithm SoftMealy can generate a wrapper represented as *finite-state transducers* (FST) using a small amount of labelled training examples. An overview of web data extraction systems can be found in [FMFB14]. All of these systems are based on the analysis of the text or the structure of the content, not the visual representation of the content.

1.2.1 Information Extraction

IEPAD [CL01] does not involve human engagement and does not use content dependent heuristics. It automatically discovers extraction rules to discover repeated patterns in in the DOM tree of web pages by using *PAT trees* (Practical Algorithm to Retrieve Information Coded in Alphanumeric). IEPAD consists of three modules, an extraction rule generator, a pattern viewer, and an extractor module.

In [ZZWL13] Zhang et al. describe a system to extract top-k lists from the web. Instead of focusing on tables and lists they try to extract lists of content pages typically found in newspapers and blogs. Examples of the content they are interested in are the following:

- *20 Most Influential Scientists Alive Today*
- *Twelve Most Interesting Children's Books in USA*
- *10 Hollywood Classics You Shouldn't Miss*
- *.net Awards 2011: top 10 podcasts*

In contrast to extraction of table and list structures, their approach is aimed at identifying top-k lists from the text with the use of techniques from the field of Natural Language Processing. Top-k lists are extracted in three steps.

1. Recognize top-k pages based on the title by training a classifier on typical headlines of such web pages and by extracting information from the headline, such as k (number of items in the list), *concept*, *rankingcriterion* (best, worst, richest, and so on), *time*, and *location*.

2. Extract top-k list from each top-k page. Since content pages in general do not use semantic markup, such as `<ul>` or `<table>`, this process relies heavily on the information extracted from the title. For example, by knowing the length k and the concept of the list.

3. Evaluate the relevance of the list content by extracting meta information. For example, if the lists contains books this might evolve finding the author of each book.

1.2.2 Visual Features in Information Extraction

The VIPS (Vision-based Page Segmentation) [CYWM03] system is one of the earliest systems to rely on visual information for the wrapper generation process. VIPS simulates a person understanding of a web page by combining the DOM structure with visual cues to build a *vision-based content structure*. Visual cues used in the system are, for example, lines, blank areas, images, font sizes, or colors. Unlike other systems VIPS does not extract information from web pages, but rather focuses on understanding the structure of the content of a web page.

A system relying on visual information is presented in [KHF+13]. The authors use different machine learning techniques to extract information about forms on travel websites. Their web object identifcation works by first extracting features from web pages, then computing a distance matrix for the features, and then finally feeding the distance matrix into a classifier. Features are separated into four different categories, *interface features*, *spatial features*, *visual perception features*, and *textual features*. During the classification step they used three different machine learning techniques of different types, namely *Logistic Regression*, *M5PM*, and *SVM* (*Support Vector Machines*). Using their system the authors could achieve a classification rate of 77.07% to 80.31% depending on the category of travel site.

Zhao, Meng, Wu, Raghavan and Yu present *ViNTs* [ZMW+05] (Visual information aNd Tag structure based wrapper generator) to automatically generate wrappers for search engines. They extract *search result records* (SRRs) from *result pages* using both visual features (such as the shape of the blocks) and the tag structure. The approach is fully automatic and relies on analyzing the result pages of some sample queries during wrapper generation.

The approach proposed by Algur and Hiremath in [AH06] uses visual clue information to identify the main data region on a web page and then extracts data records from the region using visual information. By first identifying the data region the authors try to remove visual clutter, such as advertisments, navigation-panels or copyright notices from the web page before extracting information.

In [KA09] the authors acknowledge the fact that interfaces are designed for humans and that it is difficult for machines to understand an interface. Elements can be visually very close, but are far apart in HTML, which makes it hard for machines since they don't have a cognitive ability to identify segment boundries. The presented system uses *Hidden Markov Models* (HMM) and implicit knowledge from an interface designer. Using this

approach they build a single model to segment interfaces from differnet domains. HMM is a statistical machine learning technique, formally defined as a finite state automaton with stochastic state transitions and token emissions.

Visual features are used in PDF Analyser [Has09] to extract information from PDF files. Unlike existing systems, which use the bitmap representation of a PDF file their system extracts the textual and graphic context directly from the PDF document stream. They are using an adjacency graph representation and a best first clustering algorithm to segment the objects on a web page bottom up.

1.3 Problem Description

This master thesis approaches the problem of identifying rankings on web pages using visual information. Most websites are designed for humans, relying on visual information to detect and identify areas of interest on a website. By using techniques that might resemble how an user perceives a web page we build an algorithm that can identify objects with greater precision. However, in contrast to extraction methods that work soley on the level of HTML code and the DOM tree, we need to, in addition to downloading the HTML, render the HTML and CSS of the website and execute JavaScript. Rendering the web page in the browser enables analysis of a web page layout and visual cues which are available to the user.

We use visual information to feed a set of Machine Learning algorithms to build a classifier that can identify rankings even on new and unknown websites. Finding the best algorithm is another problem we face. We need to find the right balance between precision of the algorithm and the time required to achieve this precision. In finding the best classifier we also need to deal with a very unbalanced data set that is due to the nature of websites. On most websites the desired information is surrounded by additional content, such as navigation, advertisment, copyright information and so on. The number of labelled instances in the training set is therefore very small compared to the unlabelled instances and we need to find ways to produce precise results nevertheless.

Additionally humans experience websites through web browsers. For the fair evaluation we develop tools for annotating page's elements and computing features. These tools utilize web page models (DOM tree and CSS Object Model) rendered by state-of-the-art browsers. Commonly used tools in research often lack behind in adapting new features resulting in websites that can not be correctly processed. Therefore we need to find ways to integrate commercial web browsers into our research software.

1.4 Methodology

This master thesis follows the design science research cycle illustrated in Figure 1.1 and described by Hevner in [Hev07]. The cycle consists of three sub-cycles, the *relevance cycle*, *design cycle* and *rigor cycle*, which will be discussed below. The design science research cycle is not a rigor methodology, but provides a flexible framework for conducting research in an area that is often based on experimentation and iteration.

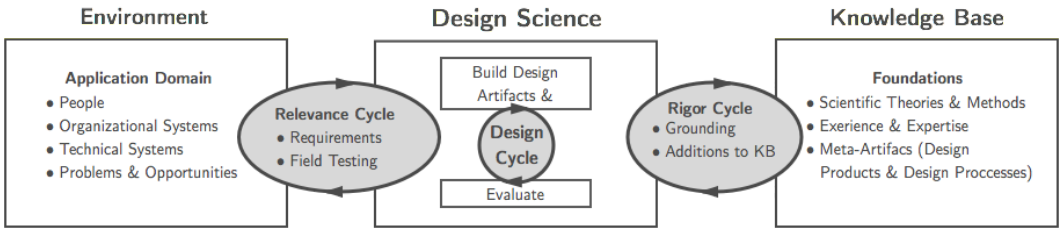


Figure 1.1: Design Science Research Cycle [Hev07]

1.4.1 Relevance Cycle

The relevance cycle provides the requirements and the acceptance criteria for the evaluation of the research. In [Hev07] states that research must ask itself: "Does the design artifact improve the environment and how can this improvement be measured?" Researchers refine the requirements and acceptance criteria iteratively as they know more about the problem.

1.4.2 Rigor Cycle

Past knowledge is provided to the research through the rigor cycle. The researcher needs to select and apply theories and methods appropriate for their research from a vast knowledge base of scientific theories and engineering methods.

1.4.3 Design Cycle

In the design cycle research is conducted in an iterative way were an artifcact is constructed, evaluated, and based on this feedback the design is refined.

1.5 Contribution

The system, Wrest, presented in this thesis, provides a workflow for identifying web rankings on websites. We present each step of the workflow, including feature extraction, feature computation, a detailled description of visual features used in the workflow as well as pre- and post-processing for the classification workflow. In addition we evaluate our system using a reasonably big dataset and a large amount of different classifiers. The evaluation process is clearly documented and the source code and the results are distributed together with this thesis. Therefore, the results are reproducable.

We present a specialised tool – an extension for Google Chrome – to annotate websites inside a commercial available web browser and to extract features from websites using a commercial rendering engine. A detailed description of the annotation workflow is given and the extraction workflow is also described in this thesis. In addition we describe visual, spatial, structural and textual features to identify web rankings and provide a detailed description of the feature computation workflow as well as the classification

workflow. We also evaluate our system and present the results, including raw data to reproduce our results.

Several parts of the source code have been made available as separate libaries. These libraries are not only useful for visual web data extraction, but have a broad range of capabilities.

1. *serialdom.js*[12] is a JavaScript libary to serialize the Document Object Model (DOM) of a web page in a simplified and non-recursive manner.
2. *seldom.js*[13] is a JavaScript libary that helps developers to build interfaces that allow users to select DOM elements on web pages.
3. *domlight.js*[14] is a JavaScript libary to visually highlight DOM elements on a web page.
4. *domcss.js*[15] is a JavaScript libarry to compute the position and CSS properties of a DOM element or all DOM elements on a web page.

All of these libraries have been released under the *MIT License*[16] and can be used in both educational and commercial environments.

1.6 Structure of the Master Thesis

This section describes the structure of the master thesis.

In this section we provide an introduction to this thesis. We state the problem in Section 1.3, describe the methodology used in Section 1.4 and the motivation for writing this thesis in Section 1.1. Additionally we give the current state of the art and related work in Section 1.2 and describe our contribution to the scientific community in Section 1.5.

We follow in Chapter 2 by taking a step back and giving an overview how machine learning and extracting data from the web works.

Chapter 3 illustrates how we approached the problem of identifying web rankings using visual information and describes the workflows for annotating websites, feature extraction, feature computation and classification. This chapter also contains a description of features we use in our system.

We illustrate how we evaluated our work and present the results of our evaluation in Chapter 4. Our evaluation consists of two parts. First we evaluate the performance of different machine learning algorithms and in Section 4.4 we compare Wres to Diffbot and Import.io, two commercial web ranking extraction systems.

In Chapter 5 we conclude the thesis by summarizing our results.

[12] https://github.com/florianeckerstorfer/serialdom.js
[13] https://github.com/florianeckerstorfer/seldom.js
[14] https://github.com/florianeckerstorfer/domlight.js
[15] https://github.com/florianeckerstorfer/domcss.js
[16] http://opensource.org/licenses/MIT

CHAPTER 2

State of the Art

In this chapter we are presenting the current state of the art on which we based our system. In Section 2.1 we present classification techniques. We give an overview on web data extraction in Section 2.2 and go into more detail on supervised information extraction systems in Section 2.2.1 and semi-supervised information extraction systems in Section 2.2.2.

2.1 Classification Techniques

Classification is a specific area of machine learning in which the goal is to assign a class k from a discrete set of possible classes to an instance of a data set [Mit97]. There exist other types of machine learning, such as *regression*, however, since our problem at hand, *web object identification*, does only involve classification we will not further discuss regression.

In Section 2.1.1 we start by introducing the concept of decision tree classifiers. We follow this with a description of the *Random Forest* learning technique in Section 2.1.2. Next we will introduce *Rule-based Learning Algorithms* in Section 2.1.3 and *Support Vector Machines* in Section 2.1.4.

2.1.1 Decision Tree

In this section, we discuss one machine learning technique, *Decision Trees*, to illustrate how such a learning algorithm works. Decision Tree has been choosen because compared to other techniques it can be easily explained and visualized.

The discussion will be presented using an example from [Liu07] and we will use the training data presented in Table 2.1. The training data is from a loan application dataset and the attributes A include the `Age` of the applicant (possible values are `young`, `middle`, and `old`), `Has_job` (either `true` or `false`), `Own_house` (`true` or `false`), and `Credit_rating` (can be `fair`, `good`, and `excellent`). The last attribute is the

ID	Age	Has_job	Own_house	Credit_rating	Class
1	young	false	false	fair	**No**
2	young	false	false	good	**No**
3	young	true	false	good	**Yes**
4	young	true	true	fair	**Yes**
5	young	false	false	fair	**No**
6	middle	false	false	fair	**No**
7	middle	false	false	good	**No**
8	middle	true	true	good	**Yes**
9	middle	false	true	excellent	**Yes**
10	middle	false	true	excellent	**Yes**
11	old	false	true	excellent	**Yes**
12	old	false	true	good	**Yes**
13	old	true	false	good	**Yes**
14	old	true	false	excellent	**Yes**
15	old	false	false	fair	**No**

Table 2.1: Training data from a loan application dataset

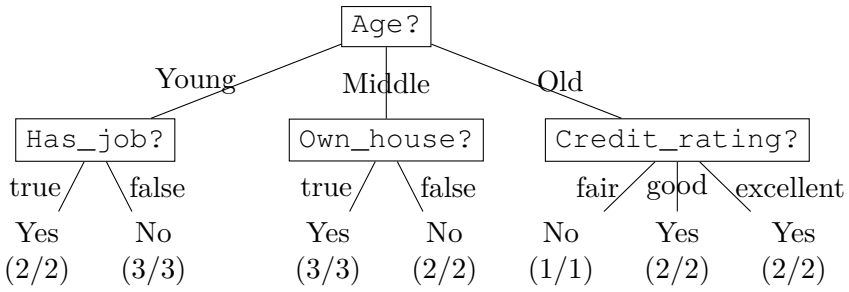


Figure 2.1: Decision tree based on the dataset in Table 2.1

class label and indicates if the loan application has been approved and can be either `Yes` or `No`.

In Figure 2.1 we see a possible decision tree for the training data from Table 2.1. The tree consists of two different types of nodes. Internal nodes are decision nodes that ask a question on a single attribute and depending on the answer the next node is selected. Leaf nodes present the possible values for the class label. When a leaf node is reached the value of that leaf node indicates the classifaction made by the model.

We can evaluate the decision tree by supplying a *test set* and walk through the tree to reach a leaf node. When we reach the leaf node we compare the value predicted by the leaf node and compare it with the class label in the test data. For example, a single test instance is given in Table 2.2 and if we use the attributes and the decision tree from Figure 2.1 we will reach a `No` leaf node, which is the same as in the test data. This means the decision tree correctly classified this instance.

ID	Age	Has_job	Own_house	Credit_rating	Class
16	young	false	false	good	**No**

Table 2.2: Training data from a loan application dataset

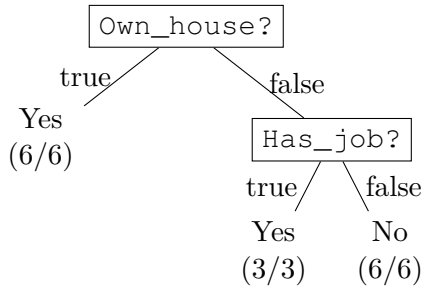


Figure 2.2: Decision tree based on the dataset in Table 2.1

ID	Age	Has_job	Own_house	Credit_rating	Class
17	old	false	false	excellent	?

Table 2.3: Instance that results in different classification using the trees in Figure 2.1 and Figure 2.2

If we use the instances from Table 2.1 the decision tree in Figure 2.1 would achieve a accuracy of 100%. However, this is not the only possible decision tree to achieve such an accuracy from the training data. Depending on the data, the algorithm, and the parameterization of the algorithm the tree will look differently. The machine learning package Weka alone provides eight different tree-based algorithms for classification[1]. In general, we want trees to be as small as possible. Small trees have several advantages, including that new instances can be classified faster, people can better understand them, and they have better accuracy. Accuracy is improved in small trees because with less nodes the tree is more general, that is, they can better handle previously unseen instances. Figure 2.2 shows a tree that provides the same classification results as the tree in Figure 2.1, but consists of 2 instead of 4 nodes. However, this is only true for the specific training data, it is easy to find an examples that results in a different classification. The instance in Table 2.3 predicts a class label `Yes` based on the tree in Figure 2.1 and `No` based on the tree in Figure 2.2.

Depending on the algorithm the width and depth of the tree can become quite big. As already mentioned bigger trees are less general, that is, these trees can not handle unseen test data. If a tree does not **generalize** very well it is called **overfitting**. If a classifier f_1 obtains an accuracy higher than classifier f_2 on a training dataset, but f_2 achieves a higher accuracy than classifier f_1 on a test dataset different from the training

[1] `http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/package-summary.html`

set we say f_1 **overfits** the data [Mit97]. Overfitting can be countered by pruning the tree, that is, removing branches or sub-trees and replacing them with leave nodes. The goal, of course, is to remove those branches and sub-trees that eliminate the most nodes and keeps the accuracy as high as possible. There are two ways to prune a tree: either pruning is done while building the tree (**pre-pruning**) or after the tree has been built (**post-pruning**). Most algorithms use post-pruning, because more information about the usefullness of branches is available after the tree has been fully built.

2.1.2 Random Forests

Random Forests is a classification and regression technique introduced in [Bre01] that grows an ensemble of trees which vote on the most popular class. Each tree is grown in isolation and the classifier is generated by averaging the predictions of the individual trees.

When constructing a random tree one has to make three choices [DMdF13]:

1. The method for splitting the leafs,
2. the type of predictor to use in each leaf, and
3. the method for injecting randomness into the tree.

Splitting the leafs requires to find possible candidates and a way to determine the quality of each candidate. One approach that is commonly used is to optimize a purity function over the leafs to maximise the information gain [JWHT13].

While there exist complex predictors (for example, described in [CSRK10]) most commonly a simple averaging over the training data is used for classification.

Randomness can be introduced using several different ways. One is to bootstrap each tree using slightly different data set. Another way is to randomize the dimensions when splitting the candidates at each leaf or choose a random combination of features. These can be either completley random or can be optimized using data in the leaf.

2.1.3 Rule-based Learning Algorithm

A rule-based learning algorithm learns a set of if-then rules that, in combination, classify new instances [Mit97]. There exist different ways to learn these if-then rules. One common solution is to convert a decision tree into a rule set by defining one rule for each leaf of the tree. Genetic algorithms encode rules in a bit string and genetic search operators can be used to explore the search space. The previous two approaches generated rules by converting another model into if-then rules, but rules can also directly learned. Rules that are directly learned are first-order rules that contain variables and are more expressive than propositional rules.

The following rules, taken from [Mit97], describe the concept of *Ancestor* and are recursive. It would be difficult to represent such a recursive concept using decision trees or other propositional rules.

```
IF Parent(x, y)                          THEN Ancestor(x,y)
IF Parent(x, z) ∧ Ancestor(z, y) THEN Ancestor(x,y)
```

One advantage of rule-based learning algorithms is that people can read and understand rules, that is, the knowledge is accessible to people. For example, domain experts can review, modify or extend rules generated by an algorithm.

PART Rule-learning Algorithm

In this section we describe PART, first presented in [FW98]. PART converts a decision tree into rules, but unlike other popular algorithms in this category, for example, C4.5 [Qui93] and RIPPER [Coh95] it does not require a global optimization step. We choose to describe PART in this section, because it was the best performing rule-based algorithm for identifying objects in our dataset (see Chapter 4).

The PART algorithm learns new rules by building a partial decision tree for a set of instances and transforms the leaf node with the largest coverage into a rule. The instances that are covered by the rule are then removed from the set and the algorithm continues with the remaining set of instances recursively. By building a new pruned tree for each rule the problem of over-prunning can be avoided. The decision trees used by PART are partial or pruned because they contain branches to undefined subtrees, tree building stops once a subtree is found that can not be simplified further.

The evaluation presented in [FW98] compared PART to C4.5, C5.0 and RIPPER and concludes that PART performs similarly to C4.5 and C5 and produces more accurate results than RIPPER. Compared to C4.5 and C5.0, however, rules generated by PART are simplier and thus easier to understand. Because no global optimization step is required the overall algorithm is simplier than those of other rule-based algorithms.

2.1.4 Support Vector Machines

Support Vector Machines (SVM) are a particular popular and effective learning technique first introduced in [CV95]. The idea of SVM is to map the input into a high dimensional feature space using a non-linear mapping. A high generalization is achieved by constructing a linear decision surface in the space using special properties.

In a support-vector network the classification problem can be seen as the problem of finding the optimal separation of all classes in an n-dimensional space. Figure 2.3 illustrates this for the 2 dimensional space. The best classification is achieved by maximising the margin of separation and finding the optimal hyperplane. Finding the optimal hyperplane that separates the classes well and generalizes well is a hard problem, since if every vector (or instance) is considered when searching for an optimal hyperplane the feature space would be billion dimensional. However, it was shown that one can construct the optimal hyperplane by only considering the support vectors (the filled circle and squares in Figure 2.3) since those determine the margin of separation. The hyperplane is expressed as a kernel function k, which is used to evaluate new instances on the training data. Because only some of the vectors are required to evaluate the kernel function, it is called a *sparse kernel function*.

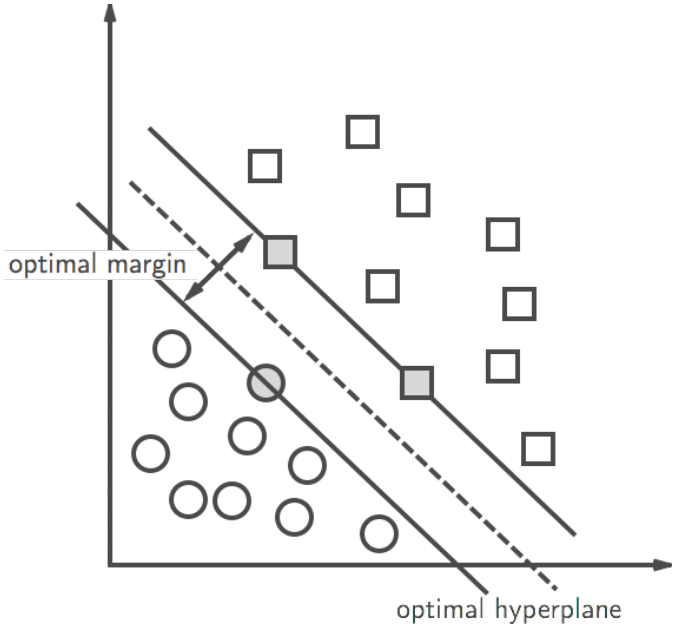


Figure 2.3: Support vectors separate two classes in a 2 dimensional space. Based on a figure in [CV95]

Standard support vector machines are two-class classifiers. In order to solve real world problems, which often requires solving problems with more than two classes there have been multiple proposals on how to adapt SVM [Bis07]. *One-versus-the-rest* creates K separate SVMs for each of the k possible classes in the model and trains the SVM against all other classes. One problem with this approach is combining the decisions of multiple classifiers can lead to an instance being assigned to multiple classes. Additionally the training sets for the individual SVMs are imbalanced and there no longer exists a symmetry between classes during training.

It is also possible to train all K SVM at the same time using a single objective function. However, this increases the complexity from $O(KN^2)$ to $O(K^2N^2)$.

While there are even more approaches for solving multi-class SVM the last approach we present is often called *one-versus-one*. For each possible pair of classes a SVM is trained and an instance is classifed by the number of votes from the individual SVMs.

SVM are, in addition to classification problems, also able to solve regression problems. The discussion on SVM for regression is, however, out of scope for this thesis.

Relevance vector machine (RVM) is another sparse kernel technique that is based on a Baysian formula and provides, in contrast to SVM, posterior probalisic outputs. In addition the solutions generated by RVM are typically much sparser than those of SVM [Bis07]. We did not test our system using RVM since the learning algorithm is currently not available in Weka.

2.2 Web Data Extraction

In a survey of web information extraction systems [CKGS06] Chang et al identified the automation degree as one possibility to classify these systems. They found four levels of automation degree:

Manually-constructed IE systems A developer uses a general programming language such as Perl or Ruby or a special-designed language to write the wrapper to extract information. A skilled programmer is required to write and maintain the wrapper and therefore this appraoch is only feasable if the number of required wrappers is small. OXPath [FGG+13] is an extension of XPath [CD99] that adds actions (e.g., clicks and filling out forms), iteration and markers for labelling elements to extract.

Supervised IE systems The systems provides a GUI for a user to label examples of the data to extract. However, the example has to be complete and exact for the system to be able to generate a wrapper. STALKER [MMK99] is a supervised system that extracts data by describing a page in a tree-like structure, in which leaves are the extracted attributes. A more detailed description is of supervised IE systems is provided in Section 2.2.1.

Semi-Supervised IE systems Similar to supervised IE systems, but they accept rough examples of the data to extract. Thresher [HK05] is a semi-supervised IE system tree edit distance of labelled examples to generate a wrapper. We describe semi-supervised IE systems in more detail in Section 2.2.2.

Un-Supervised IE systems The system can generate wrappers without labeled training examples and has no user interactions. RoadRunner [CMM01] automatically generates wrappers by comparing the similarities and differences of pages of a large web site. MDR-2 [ZL05] is another un-supervised IE system, however, it works on by finding similarities between records on a single web page.

2.2.1 Supervised Information Extraction Systems

Supervised systems can be explained in relation to human learning [Liu07]. While humans build new knowledge from past experiences, machine learning builds new knowledge from data collected in the past. The goal of a supervised system is to build a function called *classification model* or, simply, *classifier* that takes a dataset D and predicts a class label for each instance of the D. The classifier is built by providing a training set, where each instance consists of a set of attributes $A = \{A_1, A_2, \ldots, A_{|A|}\}$ ($|A|$ is the number of attributes) and a class attribute C, which can have a value out of a discrete set of values $\{C_1, C_2, \ldots, C_{|C|}\}$ ($|C|$ is the number of attributes, $C_{|C|} > 1$). The class attribute is often called class label or just label. Each instance of the dataset D is a past experience of the supervised system. The function can take a number of forms, for example, a decision tree, a set of rules, or a Bayesian model. After the classifier is generated using the training

set it is evaluated using a test set to evaluate the accuracy of the model. The test data needs to have class labels like the training set, but it should not be part of the training set. The accuracy of a classifier is defined by

$$\text{Accuracy} = \frac{\text{Number of correct classifications}}{\text{Total number of test cases}}$$

An instance is correctly classified if the class label predicted by the *classification model* is the same as the class label defined by a human trainer. One of the major problems of evaluating machine learning systems is that this kind of evaluation is only correct if the distribution of training examples and test examples (including future unseen examples) are identical. However, in practice this assumption is often violated. For example, in Section 3.1 we described five different types of *Web Rankings* and we included examples from all of these in our dataset (see Section 4.1). In a few years a new design trend introduces a new kind of ranking, which would be not considered by our system and this fault of our system would not be reflected in the evaluation. In addition, there could always be variations of the basic types of rankings that are used by a small number of web sites that are not included in the training and test examples and would not be classified correctly by our system.

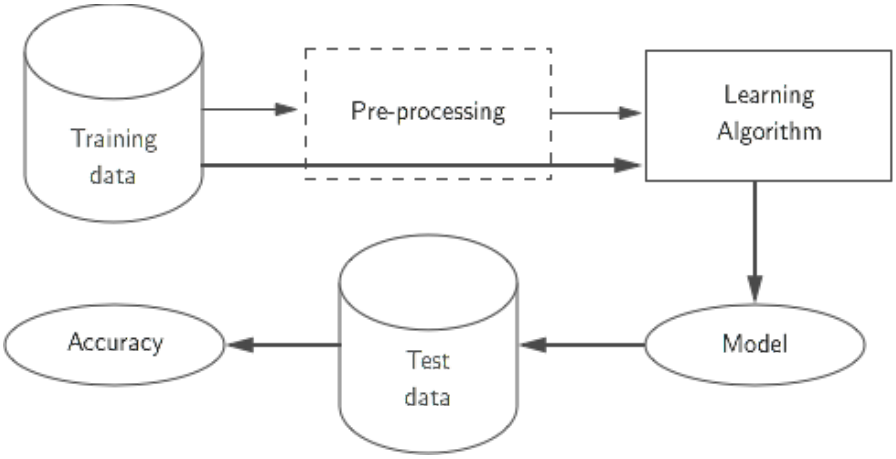


Figure 2.4: Illustration of the basic learning process in a supervised system. Based on a graphic in [Liu07]

The basic learning process is illustrated in Figure 2.4. First the model as to be generated by the learning algorithm through the supplied training data. In the second step the model is evaluated using the test data and the accuracy of the classifier is determined. If the accuracy is not satisfactory an optional *pre-processing step* can be introduced before the data is given to the learning algorithm. In most cases supervised learning is an iterative process and it is unlikely to achieve a high accurancy on first try. Instead different learning algorithms have to be tested and evaluated and many algorithms allow to tweak one or more parameters which can increase the accuracy of the model.

2.2.2 Semi-supervised Information Extraction Systems

Semi-supervised information extraction systems work similar to *supervised information extraction systems* (described in Section 2.2.1) but do not require complete examples to perform extraction. Because training examples only have to be partially labelled such systems are often also called *partially supervised learning systems*. By requiring to label only part of the examples semi-supervised learning systems reduce the amount of manual labour to create a training set. In the case of web data extraction this is extremely useful since web pages often consist of thousands of elements. Our test set (see Section 4.1), for example, contains web rankings with over 200 elements that would have to be labelled with `wr-item` and the same amount of elements for the `wr-item-title`, `wr-item-description`, `wr-item-thumbnail` labels. In practice it is unrealistic to require labelling that many elements and therefore semi-supervised systems have a clear advantage over supervised systems.

Our *Wres* system uses a semi-supervised approach to train classifiers, since training examples do not have to be complete. In our dataset we labelled a small number of positive examples and all others are labelled with `none`.

CHAPTER 3

Semi-supervised Web Data Extraction

In this chapter we explain our approach to extract web rankings. First, in Section 3.1 we define how a *Web Ranking* looks structurally and visually. In this section we also define the labels we use to annotate web rankings and the types of web ranking we extract. Next we discuss the *Annotation Workflow* in Section 3.4 and describe how annotating web rankings works in our system. In Section 3.6.2 we define how we extract raw features from web pages. The computation of the actual features is described in Section 3.6.2 and includes how we calculate relative feature and how we normalize certain features. Lastly we explain how the classifacation workflow looks like in Section 3.2.

3.1 Phenomenology of Web Rankings

The goal is to identify web rankings and the items of web rankings on web pages and extract the title of the ranking and the title, description, and thumbnail of each item in the ranking. These labels are related to each other, for example, each title of an item must belong to an item, the items must belong to an ranking, and so on. The following list shows all labels and Figure 3.1 highlights the relationships between the labels.

- **Web Ranking Region** `wr-region` Region that contains all elements that belong to the ranking, thus, the root of the web ranking
- **Web Ranking Title** `wr-title` *(optional)* Title of the web ranking
- **Web Ranking Item Region** `wr-item-region` Region that contains all items of the web ranking
- **Web Ranking Item** `wr-item` *(repeatable)* Single item of the web ranking

- **Web Ranking Item Title** `wr-item-title` *(optional)*
- **Web Ranking Item Description** `wr-item-description` *(optional)*
- **Web Ranking Item Thumbnail** `wr-item-thumbnail` *(optional)*
- **No label** `none` Every element on the web page that has none of the above lables.

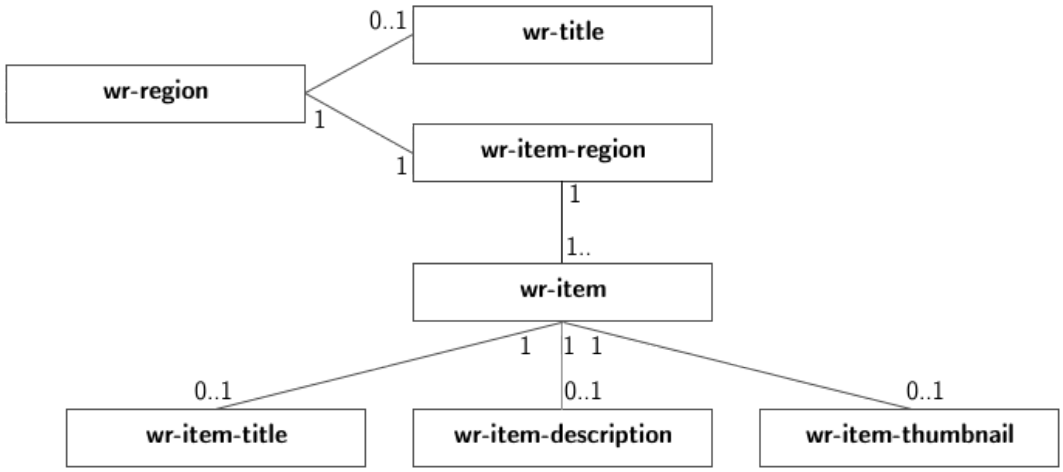


Figure 3.1: Relations of the labels to each other

We can see that `wr-region` is the root, that is, the element that describes the entire region of the ranking. The title is labelled with `wr-title` and is optional. `wr-item-region` encompasses only the items of the ranking and should, in contrast to `wr-region` not include the title. If the ranking does not contain a title, `wr-region` and `wr-item-region` may describe the same element, in which case `wr-region` should be choosen. The `wr-item-region` must contain at least one `wr-item` element, which describes a single item of the ranking. An item can contain elements labelled with `wr-item-title`, `wr-item-description`, and `wr-item-thumbnail`. It has to have at least one of these elements associated, and is not allowed to contain multiple elements of the same label.

There exists a great variety of visual styles to present a ranking on a website. We need to take these different types of rankings into account when tackling identification of web rankings. Figure 3.2 illustrates the different type of rankings we found. In the figure elements identified as *title* are colored yellow, *thumbnails* in light green and *description* in turquoise.

Table rankings (see Figure 3.2a) are in a tabular format and it is important to highlight that these can, but do not have to use the `<table>` HTML tag or the CSS display type `table`. The *list* ranking type (see Figure 3.2b) includes all rankings that display items below each other. This type defines the visual styles very loosely. Many list rankings in the dataset show a lot of additional information that is not extracted

(a) Table (b) List (c) Simple List

(d) Grid (e) Tiling

Figure 3.2: Types of web rankings

by Wres. A list of restaurants, for example, could show opening times, a rating, or a small map of the location in each list item. In contrast the *simple list* ranking type (see Figure 3.2c) does not include such complex items. Rankings of this type typically show no additional information and most likely do not include all available labels. Simple lists often only contain the title of the item, but no description and thumbnail. The same applies for the *grid* (see Figure 3.2d) and *tiling* (see Figure 3.2e) ranking types. However, in contrast to *simple-list* where most of the time only the title is displayed, grid and tiling often show an image. Sometimes the title of the item is shown inside or below the image. The difference between grid and tiling is that the items in grid rankings are roughly the same size and are shown in a very structured way with clearly visible rows and columns, while items in tiling rankings have different dimensions and no clearly distinguishable rows and columns.

Figure 3.3 shows examples of the different types we identified. In Figure 3.3a shows a tiling web ranking on 500px[1], which does only contain a thumbnail image, but no other title or description. A grid web ranking on Art of the Title[2] containing a title and thumbnail for each item is shown in Figure 3.3b. IMDB Top 250[3], see Figure 3.3c, uses a table web ranking and displays a small thumbnail and a title. Figure 3.3d shows a simple list on Last.fm[4] which includes a small thumbnail and the title. The last example in

[1] `https://500px.com/popular` (last accessed March 24, 2016)

[2] `http://www.artofthetitle.com/titles/view/grid/sort/released/` (last accessed Marh 24, 2016)

[3] `http://www.imdb.com/chart/top` (last accessed March 24, 2016)

[4] `http://www.last.fm/user/feredir/library/tracks?from=2015-01-01&rangetype=year` (last accessed March 24, 2016)

Figure 3.3e shows a list web ranking on Rotten Tomatoes[5] that contains in addition to a thumbnail image and the title a description and some information that are not identified by Wres.

3.2 Overall Approach

We propose the following workflow for classifying web rankings on a web page, which is illustrated in Figure 3.4:

1. **Feature Extraction:** Extract computed CSS styles from the web page as described in Section 3.6.2 and save the result in our JSON format.

2. **Feature Computation:** Compute features based on the raw features and store the result in ARFF format. This workflow is described in detail in Section 3.6.2.

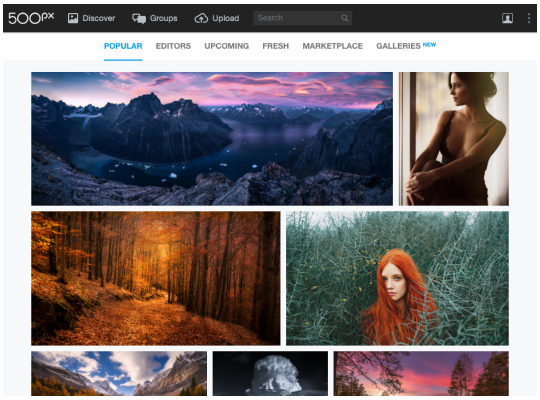
3. **Classification:** Read the ARFF file and pass the instances to a classifier built using the Weka library.

4. **Element Mapping:** Since the ARFF file used by Weka is flat, the elements have to be mapped back into the DOM or the JSON-representation of the DOM.

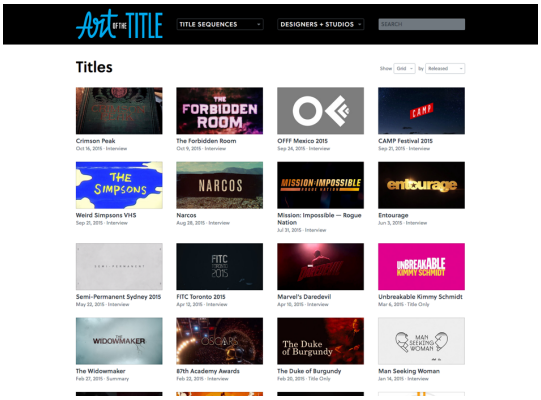
The first two steps have been described in detail in Section 3.6.2 and Section 3.6.2, thus we want to focus on the last two steps. Depending on the scenario there are multiple ways of building a classifier. If speed is important it is likely that only one classification algorithm is invoked and the result is used. However, if accuracy is more important than speed we can assemble multiple classifiers and combine the results for a higher accuracy. One approach to combine the results from multiple classifiers is described in [KHF+13]. Depending on the information available even more optimizations are possible. For example, if the type of web ranking is known prior to classification a classifier that has been trained only on rankings of this type can be used.

The last step in the classification process is identifying the classified elements on the web page. As mentioned the classifier operates on a flat file structure and we have map the result back into the original structure of the web page. If the classification was triggered from a browser, for example, through a bookmarklet or an extension the labels have to be applied to the actual DOM elements on the web page in the browser. There are use cases where we need to map the results onto the JSON representation of the DOM. For example, when we are interested in the text content of the identified elements but we do not need the visual representation of the web page it is faster to extract the data from the JSON instead of parsing and rendering the DOM tree. In both cases, mapping to the DOM and the JSON representation of the DOM, we can use the URL of the website and the *index* of the element in the DOM tree to perform the mapping.

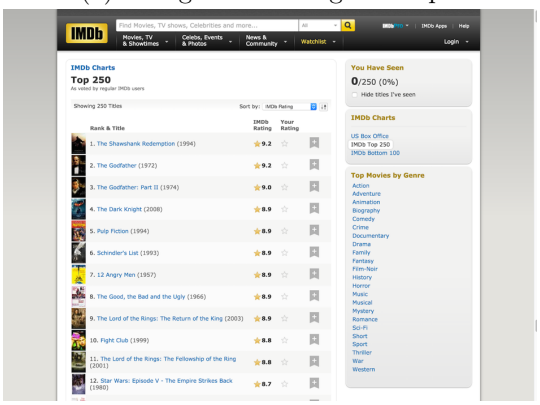
[5] `http://editorial.rottentomatoes.com/article/awards-leaderboard-2015/` (last accessed March 24, 2016)



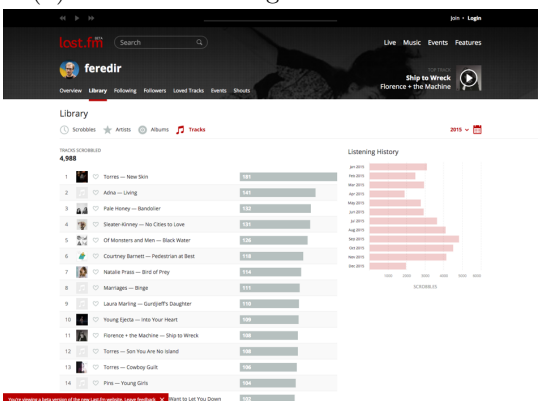
(a) Tiling web ranking on 500px



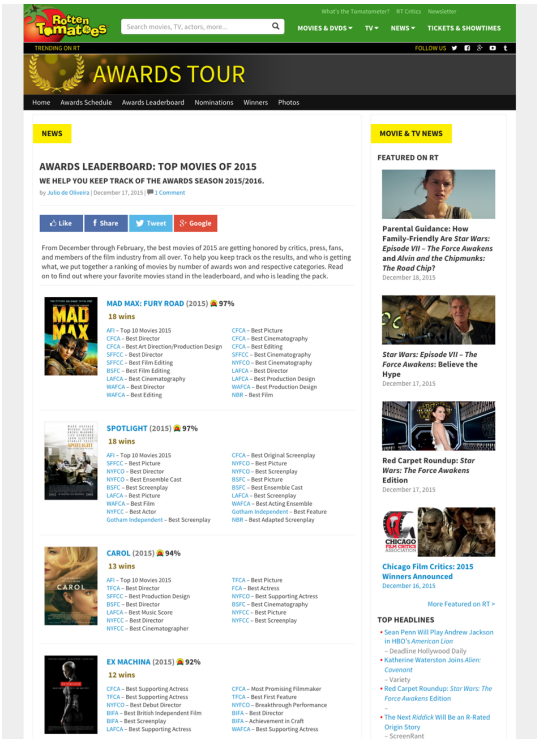
(b) Grid web ranking on Art of the Title



(c) Table web ranking on IMDB



(d) Simple list web ranking on Last.fm



(e) List web ranking on Rotten Tomatoes

Figure 3.3: Examples of web ranking types

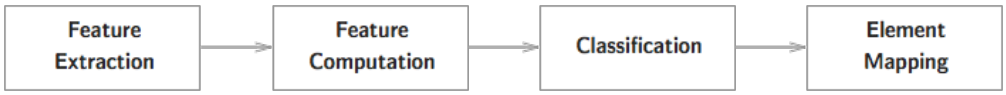


Figure 3.4: Illustration of the classification workflow

BayesNet
Naives Bayes

(a) Bayes-based algorithms

IBk
KStar
LWL

(b) Lazy algorithms

DecisionTable
JRip
OneR
PART
ZeroR

(c) Rule-based algorithms

LibSVN (Support Vector Machines)
MultilayerPerceptron
SimpleLogistic

(d) Function-based algorithms

DecisionStump
HoeffdingTree
LMT
RandomForest
RandomTree
REPTree

(e) Tree-based algorithms

Figure 3.5: Classification algorithms evaluated in our thesis

3.3 Classification Workflow

We leverage the classification algorithms provided by Weka and the graphical user interface provided by Weka[6] to load our dataset and run the classification. We run the algorithms manually and save the result buffer as text file, which can be parsed by a small utitlity to generate a Excel file that gives an overview of the results. The dataset and evaluation is described in detail in Chapter 4. In total we evaluate our system using 20 different algorithms available in Weka. All algoritms are listed in Table 3.5 arranged by the type of algorithms. First, Table 3.5a list algorithms based on Bayes' theorem. Table 3.5d contains algorithms based on functions, such as Support Vector Machines (LibSVM), Table 3.5b lists algorithms using lazy learning techniques and Table 3.5c lists algorithms that generate rules to build the classifier. Lastly Table 3.5e includes classification algorithms that are based on tree structures.

In the tool we developed for this thesis we automated most of the pre- and post-processing required. For example, our tool for feature computation (see Section 3.6.2) is able to output data in the ARFF format [Wek] required for Weka. During this step we also balance our dataset, which we describe in Section 4.1.3 in detail.

[6] http://www.cs.waikato.ac.nz/ml/weka/downloading.html

Label	Count
none	1828
wr-region	43
wr-title	43
wr-item-region	43
wr-item	538
wr-item-title	523
wr-item-thumbnail	305
wr-item-description	333

Table 3.1: Labels and the number of occurences in the balanced dataset

3.3.1 Balanced dataset

In our system we have to build a classifier from an unbalanced dataset. Since the data we want to extract is embedded in web pages intended for human viewers the page contains more negative than positive instances. Web pages contain header and footer elements, navigation, advertisment, comments and so on in addition to the web ranking we want to extract. In our dataset (see Section 4.1) only 3% of instances are positive, that is, instances related to the ranking.

In theory it would be desireable to have a completely balanced dataset, that is, a perfect distribution of labels accros instances, this is not possible in our situation. Due to nature of web rankings there exist multiple instances of `wr-item` for every instance of `wr-item-region`. Therefore we cannot achieve a truly balanced dataset, but we can create a dataset that is more balanced by randomly removing instances labelled `none`. In our balanced dataset 50% of instances are labelled `none` and the other 50% contain positive labels. We perform this balancing during pre-processing by counting the number of elements with positive labels and then selecting the same number of elements labelled with `none` from the remaining elements.

Due to the random selection of `none` elements the dataset changes every time it is generated. However, the distribution of labels remains constant since the number of `none` labels always matches the number of positive labels. Table 4.3 shows the number of labels in the balanced dataset.

3.4 Annotation Workflow

The annotation of data to build a training dataset is an essential part in the creation of a supervised machine learning system since the quality of the training set defines the quality of the classifier. Annotating the training data is a time consuming process for the human user and when designing the annotation workflow one has to balance the quality of the data and the amount required to obtain the data to achieve the best possible result in a reasonable time.

One possibility to annotate the source code of a website is for the user to read through the HTML source code in a text editor and add labels to the elements they want to annotate. However, this is a error prone and time consuming process and a better way is to provide the user with a graphical interface to select and label elements. This requires the software to render the HTML, CSS and JavaScript of the web page. The WPPS framework [Fay12] was developed as framework for web data extraction and web page understanding. The ObjIdent system [KHF+13] was developed using WPPS to extract data from the transportation domain. WPPS is also used in [Fay13] to help visually impaired people browse the web. The framework integrates Mozilla Firefox[7] with XUL Runner 1.9.2[8] into a Java application. Another approach, that is better suited to keep up with the rapid pace of browser development[9] is to integrate the annotation software into the browser by developing it as an extension. An extension is more robust in terms of new versions of the browser and are cross-platform compatible. Chrome extensions run on every operating system that is supported by the desktop version of Chrome without any additional work for the developer.

3.4.1 Annotation Extension for Chrome

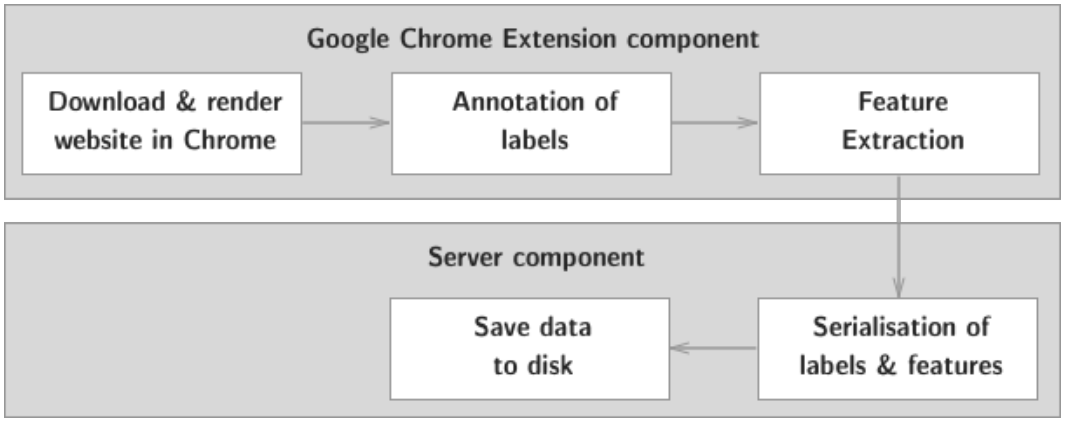


Figure 3.6: Visualization of the annotation workflow with the Chrome Extension and server component

Wres Annotation is developed as an extension for Google Chrome[10] and written entirely in JavaScript. Since the footprint of the annotation software is a lot smaller than the browser is it easier to update the code to match the latest version of the browser than by integrating each new version of the browser into a Java application. Since extensions

[7] https://www.mozilla.org/firefox

[8] https://developer.mozilla.org/en-US/docs/Mozilla/Projects/XULRunner

[9] In a couple of years many versions of XUL Runner have been released. For example, from 1.9.2 in January 2010 to 43.0 in December 2015.

[10] https://www.google.com/chrome/browser/desktop/index.html

are used by customers, browser vendors have a bigger interest in keeping the APIs stable and thus making maintaining an extension easier.

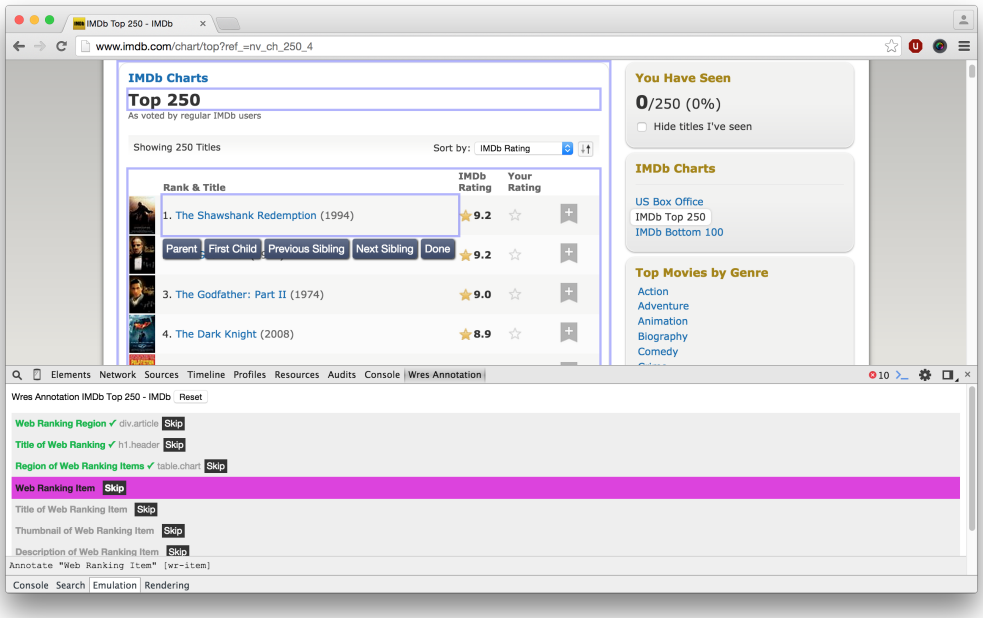


Figure 3.7: Screenshot of Wres Annotater while selecting an element

In addition to the Chrome extension Wres Annotater contains a small server written in JavaScript and run through Node.js[11] to circumvent the disability of Chrome to save files to disk. After starting the server the user can annotate web pages by opening *Developer Tools* in Chrome and selecting the *Wres Annotation* tab. At first the extension calculates the CSS properties of each element on the web page since these can be influenced by the visual guidelines inserted by the extension into the web page. The user can start annotating by selecting a label and the corresponding element. After selecting an element the user can refine their selection using a toolbar. Figure 3.7 shows the Wres Annotater extension and the process of refining a selection. After the user selected all elements on the web page they select the type of the ranking (see Figure 3.8) and press the *Save* button to send the source code of the web page to the server component for saving. The steps of the annotation workflow are shown in Figure 3.6.

The extension saves the web pages in the same format as the feature extraction tools, which allows us to reuse all processing tools. The format is described in Section 3.6.2.

[11] `https://nodejs.org`

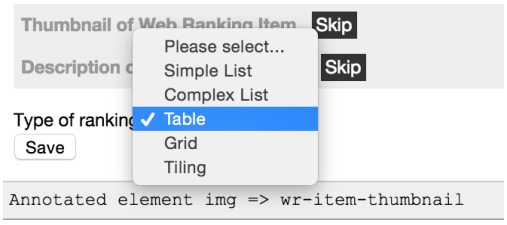


Figure 3.8: Screenshot of Wres Annotater while selecting the type of a ranking

3.4.2 Selecting and Highlighting DOM Elements

As mentioned in the previous section our annotation tool is an extension for Chrome and thus written entirely in JavaScript. To increase the modularity of our code we decided to split some functionality of the extension into libraries of their own and release them under a permissive open source license. These libraries can be used to develop annotation tools for Chrome or other browsers and could potentially be useful in projects completley unrelated to web data extraction.

All these libraries developed for this thesis are licensed under the permisse *MIT license* and can therefore be used in private, educational, and commercial projects.

Domlight.js[12] is a library to visually highlight DOM elements in the browser. It takes care of computing the correct position of an element on the web page and to overlay some kind of visual indication on top of it. The styles of that highlight can be modified by the user of the library. In addition the library deals with handling links and click events in Javascript, thus, it allows user interact with elements that are highlighted.

Seldom.js[13] handles the user interactions when selecting DOM elements on a web page. Internally it uses *Domlight.js* for highlighting the selected elements. The library deals with links and click events and allow users to select and highlight links and other clickable elements without triggering the default browser action. It also provides users with a toolbar to refine the selection. Included are tools to select the parent, siblings, and child elements of the currently selected element. This is useful when an element overlays another element completely, for example, the cells of a table often hide the row element that is their parent.

3.4.3 Computing Styles and Serializing the DOM

In addition to the libraries mentioned in Subsection 3.4.2 we also created two additional libraries to compute the styles and position of DOM elements and to serialize the DOM tree. These two libraries are used in our annotation tool as well as in the tools to extract features from new and unknown web site (see Section 3.6.2).

DomCSS.js[14] encapsulates the process of computing styles and position of elements on a web page in a simple module. Accessing the style and position of an element is not

[12] https://github.com/florianeckerstorfer/domlight.js
[13] https://github.com/florianeckerstorfer/seldom.js
[14] https://github.com/florianeckerstorfer/domcss.js

as straightforward as it seems since we have to use the computed CSS styles calculated by the browser, not the CSS styles set by the developer [Fla11]. By using the computed styles we leave the calculation of relative attributes to the browser. In addition *DomCSS.js* can attach the computed CSS styles and position to the object representing the element in DOM tree.

SerialDOM.js[15] is a library to serialize the DOM tree into a nested JSON object. The library removes recursion from the DOM which would otherwise prohibit serialization into a string format. We need to convert the DOM structure into a string in order to save the data to disk. However, since *SerialDOM.js* keeps the elements nested no information on the structure of the DOM is lost and the tree can be built up after unserializing it during feature computation (see Section 3.6.2).

3.5 Features of Web Rankings

Features introduced in this chapter are based in large part on the features described in [FHK13], but many have been extended (for example, to reflect new element types defined in HTML5) or modified to better fit the problem described in this thesis.

Features are either inherent, that is, can be computed using only properties of the element, relative to the full web page or relative to the elements context. The context is an area surrounding the element and is defined by the coordinates x_1, x_2, y_1 and y_2. We need to consider the fact that CSS allows negative values and values that exceed the page width or height in many rules that affect the positioning and size of elements [CSS11]. These coordinates are computed using the following formula.

$$x_1 = \max(x_c - (w_{cx}/2), 0) \tag{3.1}$$
$$y_1 = \max(y_c - (h_{cx}/2), 0) \tag{3.2}$$
$$x_2 = \min(x_c + (w_{cx}/2), w_p) \tag{3.3}$$
$$y_2 = \min(y_c + (h_{cx}/2), h_p) \tag{3.4}$$

where x_c and y_c are coordinates of the center of the context, which is the center of the element; w_{cx} and h_{cx} are the width and height of the context and w_p and h_p are the width and height of the web page. We use min and max to force the context to be fully visible on the web page. x_c and y_c are computed using

$$x_c = x'_1 + (x'_2 - x'_1)/2 \tag{3.5}$$
$$y_c = y'_1 + (y'_2 - y'_1)/2 \tag{3.6}$$

where x' and y' are the coordinates of the element in pixels. The width and height of the context w_{cx} and h_{cx} are computed using the formula

[15] `https://github.com/florianeckerstorfer/serialdom.js`

$$w_{cx} = \max(w_p * 1.4, w_p + 500) \tag{3.7}$$
$$h_{cx} = \max(h_p * 2, h_p + 500) \tag{3.8}$$

where w_p and h_p are the width and the height of the element in pixels. The constants to calculate the width and height of the context are taken from [FHK13]. Context is defined as those objects that are in a P^{-1} relation [RCC92] with the selected object.

3.5.1 General and Interface Features

Tag Name represents the name of the HTML tag of the element. It can be any of the elements defined in the HTML5 specification [HBF+14] by the W3C, elements defined in the SVG specification [DDG+11] as well as custom-defined elements [Gla14].

Object Type is a more specific feature than *Tag Name*. The possible values of this feature are clearly defined and are as follows:

`HtmlButton`, `HtmlCheckbox`, `HtmlFileUpload`, `HtmlImage`, `HtmlPasswordInput`, `HtmlRadioButton`, `HtmlSelect`, `HtmlTextArea`, `HtmlTextInput`, `HtmlColorInput`, `HtmlDateInput`, `HtmlDateTimeInput`, `HtmlEmailInput`, `HtmlMonthInput`, `HtmlNumberInput`, `HtmlRangeInput`, `HtmlSearchInput`, `HtmlTelephoneNumberInput`, `HtmlUrlInput`, `HtmlWeekInput`, `HtmlTable`, `HtmlTableRow`, `HtmlTableCell`, `HtmlUnorderedList`, `HtmlOrderedList`, `HtmlListItem` and `HtmlText`.

The *Object Type* corresponds to the HTML tag and for `<input>` fields on the `type` attribute.

Editable elements can be used by the user to enter information. They are elements with the tag `<input>`, `<select>` or `<textarea>` with the attribute `disabled` not present as well any other element with the attribute `contentEditable`. Input elements must have a type attribute with one of the following values:

`checkbox`, `color`, `date`, `datetime`, `datetime-local`, `email`, `file`, `month`, `number`, `password`, `radio`, `range`, `search`, `tel`, `text`, `time`, `url`, or `week`.

The type of the *Editable* feature is boolean.

Selection is boolean feature that is `true` if the element is a selected checkbox or radio element or a select element with a selected option.

3.5.2 Visual Perception Features

Foreground Color represents the text color of used in the element in HSV color space. In practice each of the three components of HSV, Hue, Saturation, and Value are separate features.

Background Color reflects the background color of the element in HSV color space. Just like with *Foreground Color* each component of HSV is represented as a separate feature.

Emphasis is a feature that incorporates the styles defined by the CSS attributes `font-weight`, `font-style` and `text-decoration`. Emphasis is calculated using the formula $(w_n + s_n + d_n)/3$.

$$w_n = \begin{cases} 1 - (400 - w)/300 & \text{if } \texttt{font-weight} <= 400, \\ (w - 400)/300 + 1 & \text{otherwise;} \end{cases}$$

where `font-weight` is the font weight as defined in the CSS in numeric form and can be $100, 200, \ldots, 900$. `normal` corresponds with the numeric value 400 and `bold` with 700.

$$s_n = \begin{cases} 1 & \text{if } \texttt{font-size} = \textit{normal}, \\ 2 & \text{otherwise;} \end{cases}$$

$$d_n = \begin{cases} 1 & \text{if } \texttt{text-decoration} = \textit{none}, \\ 2 & \text{otherwise;} \end{cases}$$

A problem with the font weight is that not every font contains all available weights. This also depends on the concrete font file installed on the users computer. For example, one user could have installed a verison of *Helvetica Neue* with six weights, another user might have a version with only three different weights. However, we assume that by defining a specific weight the creator of the web page intended to be displayed in that weight and we ignore the fact that it might not be displayed in this way on the users computer.

Font Size is the size of the font in pixels as numeric value (that is, without the unit).

3.5.3 Spatial Features

Relative Area is the area of the element $w * h$, where w and h are the width and height of the element in pixels in relation to the area of the full web page.

Aspect Ratio is computed by

$$\text{ratio} = \begin{cases} w/h & \text{if } h > 0, \\ 0 & \text{otherwise;} \end{cases}$$

where w and h are the width and height of the element in pixels.

Context Vertial Alignment Count is the number of elements vertically aligned with the element in its context.

Context Horizontal Alignment Count is the number of elements horizontally aligned with the element in its context.

Context Alignment Count is the number of elements vertically or horizontally aligned with the element in its context and computed by ContextVerticalAlignmentCount + ContextHorizontalAlignmentCount.

Context Vertial Horiztonal Alignment Ratio is the ratio of vertically and horizontally aligned elements in its context and defined by (ContextVerticalAlignmentCount+ 1)/(ContextHorizontalAlignmentCount + 1)

Pixel Character Ratio is the ratio of the elements area and the elements text. The text includes the text of the element as well as the text contained in all child elements. It is computed by $(w * h)/l$ if l, the text length, is > 0. w and h are the width and height of the element in pixels.

Relative Width is the width of the element in relation to the width of the web page: w_e/w_p where w_e is the width of the element and w_p is the width of the web page.

Relative Height is the height of the element in relation to the height of the web page: h_e/h_p where h_e is the width of the element and h_p is the height of the web page.

Relative X Position is the left-most position of the element in relation to the width of the web page: x_1/w_p where x is the x-coordinate of the element and w_p is the width of the web page.

Relative Y Position is the top-most position of the element in relation to the height of the web page: y_1/h_p where y is the y-coordinate of the element and h_p is the height of the web page.

Grid Location is the location of the element in a 3×3 grid on the web page. The grid is created by dividing the web page into a grid with 9 cells, each $w_p/3$ pixels in width and $h_p/3$ pixels in height (w_p and h_p are the width and height of the web page). After creating the grid the location is computed using the following algorithm

```
calculateGridLocation(grid, position)
    location = 0
    for i = 0...n
        if areOverlapping(grid.i, position)
            location |= (1<<i)
        endif
    endfor
    return location
end
```

where `grid` contains the coordinates of each of the 9 cells and `position` the coordinates of the element on the web page and `areOverlapping` returns `true` if the element and the cell are overlapping.

Vertical Alignment Count is the number of elements vertially aligned with the element on the web page.

Horizontal Alignment Count is the number of elements horizontally aligned with the element on the web page.

Alignment Count is the number of elements vertically or horizontally aligned with the element on the web page and computed using VerticalAlignmentCount + HorizontalAlignmentCount

Vertical Horizontal Alignment Ratio is the ratio of vertically and horizontally aligned elements with the element on the web page and computed using (VerticalAlignmentCount+ 1)/(HorizontalAlignmentCount + 1)

3.5.4 Structural Features

Number of Children represents the number of children the element has. The number is calculated recursivly and contains both element nodes and text nodes.

3.5.5 Textual Features

Token Count reflects the number of tokens in the text contained in the element. This includes the text of the element as well as the text of its child elements. Tokens are split using the regular expression

```
(\s| |\x{0020}|\x{00A0}|\x{200B}|\x{3000})
```

Line Count is the number of lines in the element and approximated by the formula $\lfloor h/l \rfloor$, where h is the height of the element in pixels and l is the line height of the element in pixels. If the CSS property `line-height` is defined as *normal* the font size in pixels is used instead. The line count of an element is 0 if either the line height is 0 or the element contains no text. Text contained in child elements does not cound towards the text in the element.

Character Count is the number of characters in the element. Every character including spaces and punctuation characters are counted, but only text that is directly in the element. Text of child elements is not counted.

Word Count is the number of words in the element. Only words that are directly in the element are counted, not words that are in child elements.

Minimum Word Length is minimum word length of words directly contained in the element.

Maximum Word Length is the maximum word length of words directly contained in the element.

Average Word Length is the average word length of words directly contained in the element.

3.6 Feature Computation Workflow

In the earlier stages of web development, web pages were written by humans in HTML, which was downloaded and rendered by a web browser. Nowadays the HTML of many sites is dynamically generated by an application on the server, that is, HTML is still the core technology of the web, but it is no longer written directly by humans. However, it is possible to use JavaScript and the `<canvas>` element [Joh15] to present a web page to the user without using HTML. While possible, this technique is rarely seldom used and mostly only in web-based applications, like Flipboard[16]. In modern web development most websites separate the styling and the markup into different files. An HTML web page can include one or more Cascading Style Sheets (CSS) [CSS15a] that determine the styling of the web page. Unlike in the early years of the standard, the W3C does release new versions of individual modules and creates a snapshots of stable modules at certain points in time. We use the CSS Snapshot 2015 [CSS15b] as a point of reference throughout this thesis. In addition JavaScript can be included in the web page that can change the markup, content and styling of the web page dynamically while a user is viewing and interacting with the web page. In addition developers also have the possibility to define styling using relative values. For example, the width of an element can be defined in percent and the browser computes the actual pixel value while rendering the web page. These computed styles are obviously called *Computed Styles*.

3.6.1 Computed CSS Attributes Serialization

The computating styles are extracted from a web page using a library we have developed called DomCSS[17]. This libraries is able to retrieve the computed styles either of a specific element or all elements on the web page. However, currently not all available styles are retrieved, mostly because not everything is useful in web data extraction. We ignore styles that deal, for example, with animation. The full list of styles extracted from each web page is in Table 3.2, which also includes the unit of the extracted value. In addition to the computed styles the library also calculates the actual position of an element on the web page and returns it in the form of coordinates.

Style	**Unit**	**Style**	**Unit**
background-color	`rgba`	letter-spacing	`px`
Continued on next page			

[16] `https://flipboard.com`
[17] `https://github.com/florianeckerstorfer/domcss.js`

Table 3.2 – continued from previous page

Style	Unit	Style	Unit
background-image	string	line-break	string
background-origin	string	line-height	px, string
background-position	string	list-style-image	string
background-repeat	string	list-style-position	string
background-size	string	list-style-type	string
border-bottom-color	rgba	margin-bottom	px
border-bottom-left-radius	px	margin-left	px
border-bottom-right-radius	px	margin-right	px
border-bottom-style	string	margin-top	px
border-bottom-width	px	max-height	px
border-collapse	string	max-width	px
border-image-outset	px	min-height	px
border-image-repeat	string	min-width	px
border-image-slice	percent	object-fit	string
border-image-source	string	object-position	percent percent
border-image-width	px	opacity	float
border-left-color	rgba	orphans	
border-left-style	string	outline-color	rgba
border-left-width	px	outline-offset	px
border-right-color	rgba	outline-style	string
border-right-style	string	outline-width	px
border-right-width	px	overflow-wrap	string
border-spacing	px px	overflow-x	string
border-top-color	rgba	overflow-y	string
border-top-left-radius	px	padding-bottom	px
border-top-right-radius	px	padding-left	px
border-top-style	string	padding-right	px
border-top-width	px	padding-top	px
bottom	px	position	string
box-shadow	string	resize	string
box-sizing	string	right	px
caption-side	string	table-layout	string
clear	string	text-align	string
color	rgba	text-align-last	string
cursor	string	text-decoration	
direction	string	text-decoration-color	string
display	string	text-decoration-line	string
float	string	text-decoration-style	string
font-family	string	text-indent	px

Continued on next page

Table 3.2 – continued from previous page

Style	Unit	Style	Unit
font-feature-settings	string	text-orientation	string
font-kearning	string	text-overflow	string
font-language-override	string	text-rendering	string
font-size	px	text-shadow	string
font-size-adjust	string	text-transform	string
font-stretch	string	text-underline-position	string
font-style	string	top	px
font-synthesis	string	vertical-align	string
font-variant	string	visibility	string
font-variant-alternates	string	white-space	string
font-variant-caps	string	widows	iint
font-variant-east-asian	string	width	px
font-variant-ligatures	string	word-break	string
font-variant-numeric	string	word-spacing	px
font-variant-position	string	word-wrap	string
font-weight	string	writing-mode	string
height	px	z-index	int
left	px		

Table 3.2: Computed styles extract from web pages

For these reasons it is important to use the rendered web page instead of the HTML code as a basis for the extraction of features. This is especially important since we use visual features for the classification. We decided to extract features directly from the browser because it gives us access to the version of the web page that matches the version the user is seeing best. Using software like Selenium[18] we can automate opening the browser, accessing web pages, generating features and extracting the results in a relatively fast way.

In addition we use PhantomJS[19], a headless browser based on WebKit, to programmable navigate websites and extract data. A headless browser is a browser that offers the full features of a web browser but has no graphical user interface.

Features are computed for the <body> element and all of its child elements. In practice this works by using the code shown in Figure 3.9, which leaves finding all elements to the browser. The features are than attached to the DOM node directly and are only later retrieved when the DOM is saved. The advantage of this technique is that it does not involve recursion, since we do not need to traverse the DOM tree using the parent-child relation of nodes.

We need to store the web page in a way that converses the structure of the rendered HTML and the computed styles to achieve the best possible classification. While rendering

[18] http://docs.seleniumhq.org
[19] http://phantomjs.org

```
var elements = document.body.querySelectorAll('*');
```

Figure 3.9: Code used to find all elements on a web page to compute features for

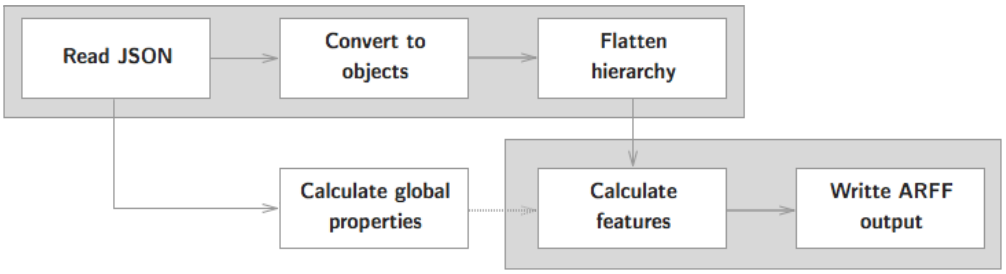


Figure 3.10: Steps executed when computing features

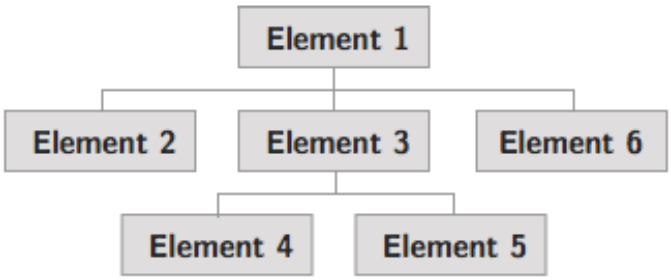
the web page the browser generates a DOM tree which satisfies both requirements, however this structure is redundant (elements link to their children and children link to their parents). Instead we create a simplified version of the DOM tree containing only necessary values and no redundancy and serialize them using JSON[20]. In addition to information stored already in the DOM we can also easily store the computed styles and computed position in our JSON format. The JSON format is in principle the same that is used in the annotation workflow and depicted in Figure 4.1 in Section 4.1. However, one difference is that values manually defined during annotation, such as web ranking type and the `wres-label` attribute, are missing in the files generated during feature extraction.

3.6.2 Feature Computation

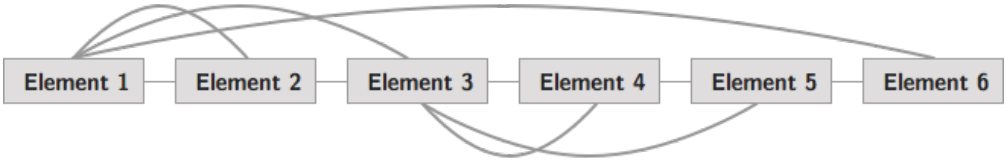
Features are calculated using a separated tool that reads a directory of JSON files, which can be generated either by the annotation tool (see Section 3.4) or the feature extraction tool (see Section 3.6.2), and outputs a list of instances in the ARFF format [Wek]. However, by using the data processing framework Plum[21] it is easily possible to write to another output format such as CSV or Microsoft Excel. ARFF was choosen as the default file format since it is native to Weka (see Section 3.2 for how Weka is used in the classifcation) and in constrast to CSV supports defining the type of an attribute.

The workflow is split into two parts and can be seen in Figure 3.10. In the first part a JSON file, which represents a single web page, is read and converted into objects. Parent-child relationships are stored in these objects as references as shown in Figure 3.11a. The object hierachy is flattened into a list. A list of objects resembles the ARFF output format more closely than a tree structure. However, because each element is an object with references to its parent and its child elements the context can be used during feature computation. The second step computes the individual features by iterating

[20] `http://json.org`
[21] `https://github.com/plumphp/plum`



(a) Tree-based object orientation generated from the JSON



(b) Flattened list representation with parent-child relations in tact

Figure 3.11: Different representation of the web page during feature computation

through the list of elements and passing each element to a set of *Feature Computers*. For each feature described in Section 3.5 a *Feature Computer* exists. Some of the features rely on global properties, for example, the web page width or the number of elements vertically aligned. To improve performance these properties are computed during the first step when every element is iterated and are stored in an object that is accessible to all elements during feature computation. Therefore it is not required to calculate these properties for each element seperatley. As described above Plum is used for managing the workflow and writting the result into a file. In addition to the computed features the output also contains the web page URL, title and the index of the element on the web page. These attributes allows us to map an element back into the tree after classification.

CHAPTER 4

Evaluation

In this chapter, we evaluate the Wres approach to web ranking extraction based on the analysis of visual features. In our machine learning settings, an instance is a DOM node [vKGM+15] of an HTML document. In particular, our main focus is Element nodes. We handle only `Element` nodes in our system. During pre-processing (see Section 4.1.2) we take text content from `Text` nodes and store them in their parent `Element` node. Other nodes, such as `Comment` or `Document` are ignored by our system as they provide no additional information. The dataset we present in 4.1 contains 124376 `Element` nodes as instances from 100 web pages.

Evaluation of classification results must take the inbalance of the dataset into account. Of 124376 instances in the dataset, 119684 instances (96%) have the *none* label. Inbalanced datasets need to be evaluated carefully, a high classification rate is reported, when instead every instance is classified as *none*. Therefore we created a balanced dataset with 9384 instances in addition. The dataset used in the evaluation of Wres is described in Section 4.1. In Section 4.2 we detail how we compare the classification algorithms we use in Wres and in Section 4.3 we present the evaluation results. We will describe which classification algorithm perform best for different web rankings and which labels can be extracted with high precision. Finally we present a comparison of our system to Diffbot and Import.io, two commercial available, general purpose, web extraction systems in Section 4.4.

4.1 Dataset

In this section we discuss the dataset used to evaluate Wres. The discussion can be split into two parts, the raw dataset, that is, the annotated web pages and the dataset after pre-processing, which is used as input for the machine learning algorithms. The raw dataset is discussed in Section 4.1.1, the pre-processed dataset is discussed in Section 4.1.2 and a balanced version of the dataset is discussed in Section 4.1.3.

```
{
    "content": {
        "nodeType": 1,
        "attributes": {...},
        "children": [...],
        "clientHeight": 1067,
        "clientWidth": 1249,
        "scrollHeight": 1083,
        "scrollTop": 509,
        "scrollWidth": 1265,
        "tagName": "BODY",
        "computedPosition": {...}
        "computedCSSProperties": {...}
    },
    "title": "Top Ten Longest Rivers in the World List -
        Fun Science Facts for Kids",
    "url": "http://www.sciencekids.co.nz/sciencefacts/topten/
        longestrivers.html",
    "rankingType": "table",
    "date": 1432213909127
}
```

Figure 4.1: Part of the JSON file representing a single annotated web page

4.1.1 Raw dataset

The raw dataset is based in part on the dataset of the WPPS project [Fay12]. However, the dataset has been extended and the screenshots and snapshots of the web pages have been updated. Pages in the dataset are not stored in HTML format, but rather as JSON. JSON offers advantages compared to HTML regarding the storage of additional information, such as the computed CSS properties and the computed position. Additionally JSON is a simple, human readable format that can be parsed easily. In contrast to flat-file formats (like CSV) nesting of objects is possible in JSON, which makes the format ideally for storing a simplified version of HTML. In our dataset the content of each web page is stored in a single JSON file combined with some metadata about the web page, such as the title and the URL. The snippet of JSON that shown in Figure 4.1 shows the root element, but hides the sub-objects of `attributes`, `children`, `computedPosition`, and `computedCSSProperties`. Additional data gathered during the annotation process is also stored in the `attributes` array. For example, the label of an element is stored in the attribute `wres-label`. The dataset has been released as open source and the full content of the JSON files can be inspected at `https://git.florian.ec/florian/wres-dataset`.

The dataset contains 99 web pages from five different types of rankings. Table 4.1

Ranking Type	Number of web pages
Table	22
List	22
Simple List	18
Grid	20
Tiling	18

Table 4.1: Ranking types and the number of web pages of each ranking type in the dataset.

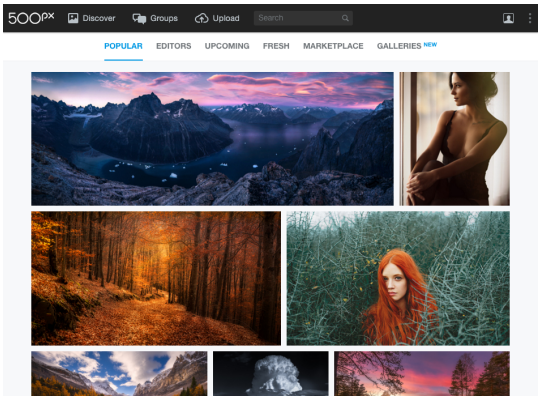
shows the ranking types and the number of web pages in the dataset for each type. Summed over all web pages the dataset contains 450509 nodes and on average each web page contains 4551 nodes. 528 is the smallest number of nodes on a single web page, 35786 is the largest number of nodes. Of these nodes 4717 are labelled, the other 445792 are not labelled. These unlabelled nodes will be labelled with `none` in the pre-processing step. On average 48 nodes per web page have a label. It is important to note that nodes do not directly result in instances for the machine learning algorithms. The format of the instances after the pre-processing are discussed in Section 4.1.2. The representation corresponds roughly to the DOM. For example, text is represented in its own node, just like it is in the DOM. However, attributes are simplied and instead of creating an object for each attribute, all attributes of an element are stored as properties of the element.

Table A.1 shows the title, URL, and ranking type of every web page in the dataset as well as the date the web page was added to the dataset. Figure 4.2 shows screenshots of one web page of each ranking type in the dataset.

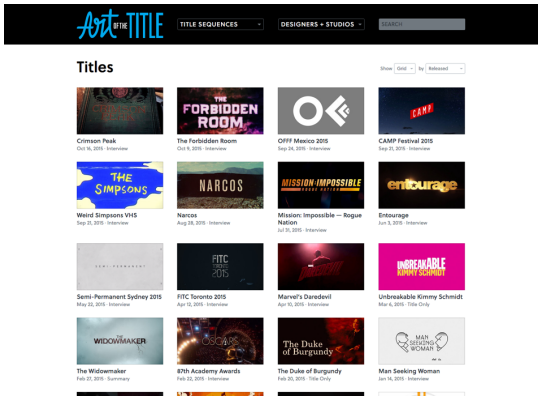
Before discussing the pre-processed dataset we need to point out the limitations of our dataset. When looking at real web pages we often have not one correct solution for labelling an element, but multiple correct solutions. A correct solution means that the label is applied to a DOM element that contains the seeked text. For example, consider the HTML code in Figure 4.3 that might be found in a ranking and describes the title of a ranking item. Any of the elements `<header>`, `<h1>`, `<div>`, and `<span>` represents the title and could be labelled with *wr-item-title*. However, during training only one of these elements can be labelled and during evluation the instance only counts as correctly classified if exactly the labelled element is chosen. This decision was made because it would have increased the time required to annotate a single web page and would involve a very close inspection of the source code of the web page to not miss any elements that could be labelled.

4.1.2 Pre-processed dataset

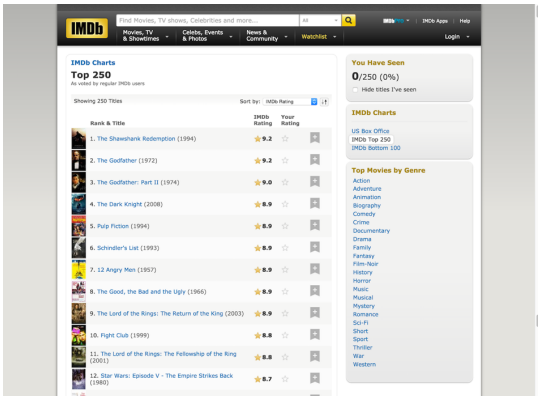
During the pre-processing step the raw dataset (described in Section 4.1.1) is read and the JSON files parsed. The output of the pre-processing tool is a ARFF file, compatible with Weka and we can analyze the dataset using Weka. In addition to the data used in the classification the dataset also contains three attributes required for identification of



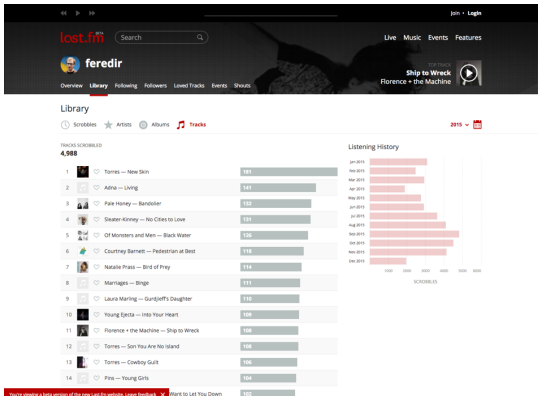
(a) Tiling web ranking on 500px



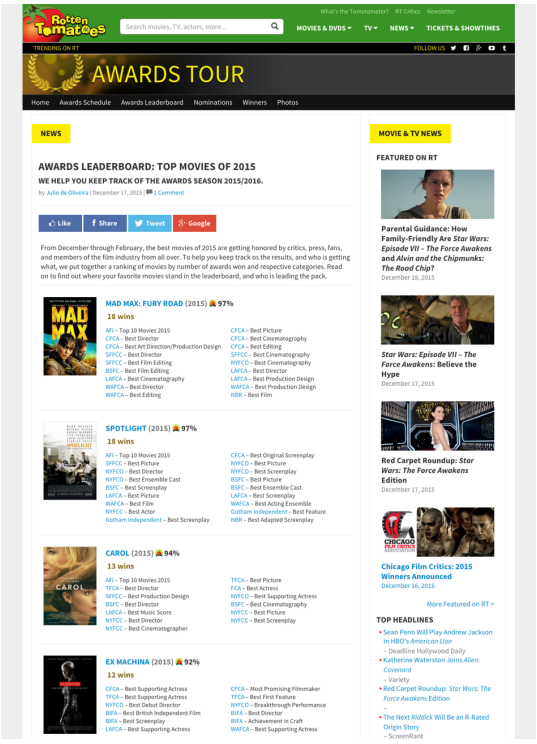
(b) Grid web ranking on Art of the Title



(c) Table web ranking on IMDB



(d) Simple list web ranking on Last.fm



(e) List web ranking on Rotten Tomatoes

Figure 4.2: Screenshots of some web pages from the dataset

```
<header class="item-title">
    <h1>
        <div class="title">
            <span>Title of Ranking Item</span>
        </div>
    </h1>
</header>
```

Figure 4.3: HTML code describing the title of a ranking item

Label	Count
none	119684
wr-region	90
wr-title	93
wr-item-region	92
wr-item	1545
wr-item-title	1204
wr-item-thumbnail	1029
wr-item-description	639

Table 4.2: Labels and the number of occurences in the dataset

elements. The columns `PageTitle`, `PageUrl`, and `Index` allow us to match an element from the pre-processed dataset with an element in the raw dataset. These attributes should be ignored during the classification process.

While pre-processing the dataset we only keep element and text nodes [Moz15] and eliminating nodes which do not contain any significant text. The significant difference in nodes in the web pages of the raw dataset and the number of instances in the final dataset is explained by the large number of irrelevant or empty nodes.

The labels found in the dataset and the associated counts are shown in Table 4.2. It is quite clear that the dataset is highly unbalanced, in fact, 96.23% of all elements are labelled with `none`. We need to take this into account when we design the evaluation methods (Section 4.2) and evaluate the results (Section 4.3).

4.1.3 Balanced dataset

As mentioned in the introduction of this chapter, our dataset is highly unbalanced: of 119684 instances roughly 96% are labelled `none`, the other 4% are all other labels combined. In theory it would be desireable to have a completely balanced dataset, that is, a perfect distribution of labels accros instances, this is not possible in our situation. Due to nature of web rankings there exist multiple instances of `wr-item` for every instance of `wr-item-region`. Therefore we cannot achieve a truly balanced dataset, but we can create a dataset that is more balanced by randomly removing instances labelled as

Label	Count
none	1828
wr-region	43
wr-title	43
wr-item-region	43
wr-item	538
wr-item-title	523
wr-item-thumbnail	305
wr-item-description	333

Table 4.3: Labels and the number of occurences in the balanced dataset

none. In our balanced dataset 50% of instances are labelled none and the other 50% contain positive labels. We perform this balancing during pre-processing (see Section 4.1.2) by counting the number of elements with positive labels and then selecting the same number of elements labelled with none from the remaining elements.

Due to the random selection of none elements the dataset changes every time it is generated. However, the distribution of labels remains constant since the number of none labels always matches the number of positive labels. Table 4.3 shows the number of labels in the balanced dataset.

4.2 Evaluation Methods

In order to verify our approach we evaluate it using the dataset described in Section 4.1 and because different machine learning algorithms are performing differently on different datasets we evaluated the dataset using all possible datasets in Weka. We did exclude some of the classifiers available in Weka, because it took too long to execute them (more than 12 hours). The other classifiers, the ones we evaluated, are listed in Table 4.4 and include some of the metrics we recorded. We are running each classifier for five different balanced datasets since the instances labelled with none are choosen at random the results can vary. The results presented in this section are averaged over the results for the five datasets. Time taken is recorded by Weka in seconds and describes the time it took to build the model, not the time to evaluate the dataset. The percentage of correctly classified instances is also calculated by Weka. In addition the weighted precision (labelled as *Precision (w)* in Table 4.4) as calculated by Weka and the unweighted precision are included in the table. Some of the classifiers produce nonsense results, for example, *ZeroR* and *DecisionStump* classify every instance in the dataset with the label none. These show up with 50% correctly classified instances in Table 4.4, but of course can not be considered as a serious result. In our evaluation we found that several classifiers produce useable results and *RandomForest* produces the best result. Using this data we can now dive into more detail and analyze the results of some of the more promising classifiers in Section 4.3.

Classifier	Time (s)	Corr. Classified	Precision (w)	Precision
BayesNet	0.556	80.82%	0.8392	0.6598
NaiveBayes	0.092	31.43%	0.619	0.3675
LibSVM	259.846	75.32%	0.7828	0.6676
MultilayerPerceptron	515.812	69.90%	0.6966	0.6178
SimpleLogistic	44.974	71.34%	0.7138	0.7262
IBk	0.004	89.45%	0.8956	0.7535
KStar	0.000	90.04%	0.9032	0.7546
LWL	0.000	55.63%	0.4072	0.2040
DecisionTable	4.736	83.70%	0.8534	0.8044
JRip	22.926	89.73%	0.899	0.8032
OneR	0.098	69.63%	0.684	0.5278
PART	8.360	92.07%	0.9206	0.8202
ZeroR	0.002	50%	0.25	0.0625
DecisionStump	0.118	50%	0.25	0.0625
HoeffdingTree	1.482	65.17%	0.6606	0.6351
J48	0.758	91.69%	0.9176	0.8250
LMT	191.332	89.94%	0.9012	0.7876
RandomForest	3.750	95.03%	0.9502	0.8761
RandomTree	0.066	90.68%	0.9068	0.7725
REPTree	0.370	89.92%	0.9008	0.8161

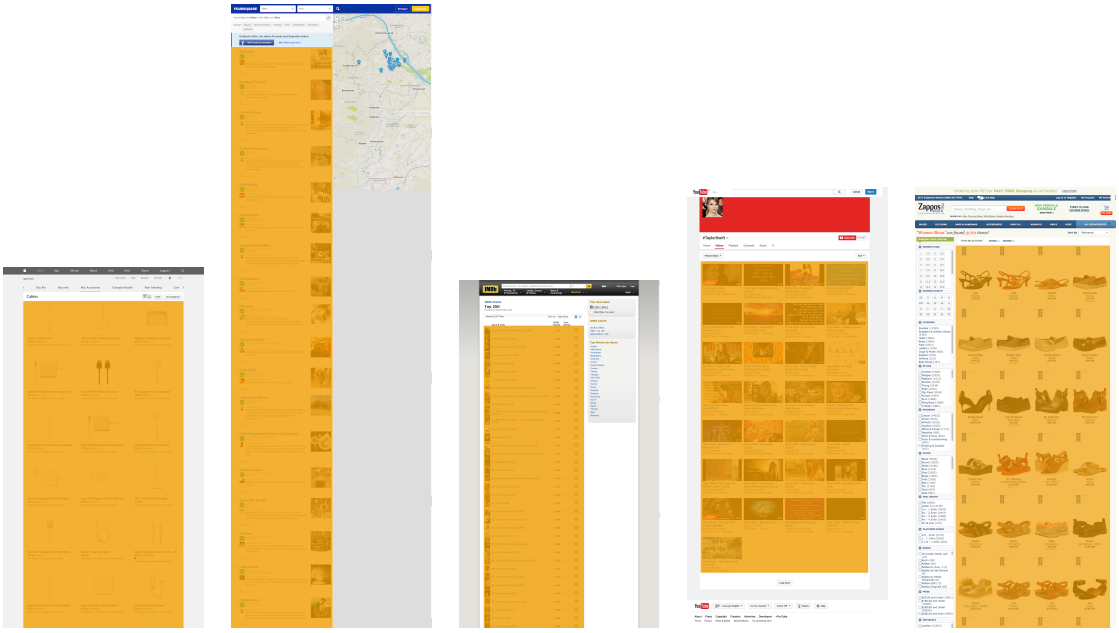
Table 4.4: Results of the evaluation of the classifiers available in Weka using 10-fold cross-validation

4.2.1 Cross-Validation

We use *10-fold cross-validation* [Liu07] to evaluate each ML algorithm. This method splits the data set into n (in our case 10) equal-size disjoint subsets and defines these sets as test sets. The remaining $n - 1$ subsets are used to train the classifier. Classifiers are trained for each of the n subsets. Therefore the data set is evaluated n times with a different subset each time. When all n subsets are evaluated the results are averaged and produce the final estimated accuracy of the learning algorithm.

4.3 Evaluation Results

Let us first analyse the results of the Random Forest [Bre01] classifier. As mentioned above, the average, unweighted precision of the classifier is 0.8761, the number of correctly classified instances is reported by Weka as 95.03% and the weighted precision is 0.9502. We will ignore the elements labelled with `none` in the following discussion, because we are not interested in them and the label was introduced to have a non-sparse dataset. When we look at the precision and the number of true positive instances we can see two

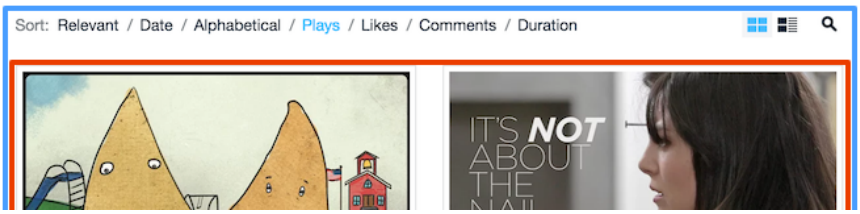


(a) Apple Store (b) Foursquare (c) IMDB (d) YouTube (e) Zappos

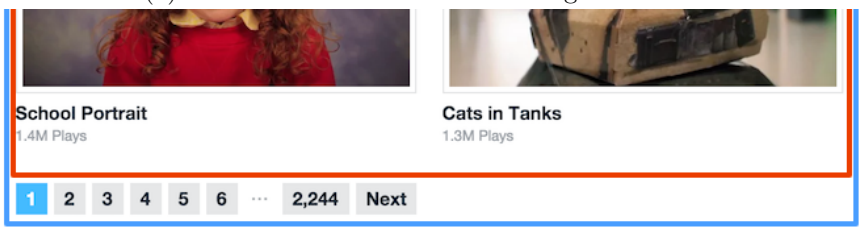
Figure 4.4: The web ranking region is visually easly to distinguish, but confuses the classifier

different groups of labels resulting in very different results. The non-repeating labels `wr-region`, `wr-title`, and `wr-item-region` have a considerable lower number of correctly classified instances than the repeated labels `wr-item`, `wr-item-title`, `wr-item-description`, and `wr-item-thumbnail`. This pattern can be seen nicely in the Random Forest classifier, but also in many of the other classifiers, such as *PART* or *REPTree*. The difference in weighted and unweighted precision is due this. One reason is that we have a lot more training examples of the repeated elements, because we tried to annotate as many items as possible (at least 10) per web page. Not every item contains all of the sub-labels, but in general there are a lot more examples than of the non-repeating labels. For example, there exist 1545 instances labelled with `wr-item`, but only 90 instances labelled with `wr-region`. This problem could possibly be solved by increasing the size of the training set. However, increasing the number of web pages in the training dataset would not only increase the number of non-repeating labels, but also the number of similar unlabled elements.

The other factor is that the non-repeating labels are much more generic than the repeating labels. Visually `wr-region` and `wr-item-region` are areas that take up much of the non-empty space of the web page and are easily distinguishable for the human eye. However, because of how web sites are built there are many possible elements in a web page that contain the region. This has already been illustrated in Figure 4.3 in Section 4.1.1. In addition to multiple elements discribing exactly the content, rankings



(a) Filter elements above the ranking on Vimeo



(b) Pagination below the ranking on Vimeo

Figure 4.5: Rankings often have additional elements above and below the ranking

	Repeatable labels			Non-repeatable labels		
Classifier	**Precision**	**Recall**	**F-Measure**	**Precision**	**Recall**	**F-Measure**
J48	0.8997	0.9463	0.9223	0.6857	0.6195	0.6471
LibSVN	0.9134	0.5960	0.7090	0.3667	0.0202	0.0377
PART	0.9092	0.9425	0.9256	0.6608	0.5790	0.6134
RandomForest	0.9427	0.9748	0.9579	0.7573	0.6297	0.6839
REPTree	0.8740	0.9318	0.9021	0.6998	0.6333	0.6597
Grand	0.9078	0.8783	0.8834	0.6341	0.4963	0.5284

Table 4.5: Our algorithm produces different results for repeating and non-repeating labels for almost all learning algorithms.

often have additional control elements or information boxes above or below the ranking. This is illustrated in Figure 4.5, where the ranking is highlighted in red and the additional control elements in blue, and results in elements that are visually very similar to the seeked region. This leads to a classifier that incorrectly classifies many of these elements, because there are more of the surrounding elements labelled `none` than actual labelled instances. The problem could be potentially solved by labelling all elements that could represent a region in the annotation process.

As mentioned above there is a huge difference in metrics like precicion, recall and F-measure between those repeating and non-repeating labels. Table 4.5 compares the precision, recall F-measure for a few selected algorithms. The measures are weighted and do not include instances labelled with `none`. We can see that our system can identify repeating labels in most cases with a high precision, however, this is not the case for non-repeating labels.

We continue the evaluation by looking at the confusion matrix of some of the classifiers that produce good results. Starting with *Random Forests* we can see that the algorithm classified only one instance of `wr-item-description` with the wrong positive label. All other confusion happened with the `none` label. By looking at the confusion matrix generated by Weka we see the effect we described above for non-repeating labels is also affecting repeating labels, but on a smaller scale. Since we should have enough positive examples of repeating labels the problem is most likely due to the fact that designers and developers often wrap text in multiple elements with the same visual properties. For the *J48graft* classifier we can see the decision tree built by the algorithm and therefore gain insight which attributes are used to classify an instance. Many of those attributes are based on the size and position of an element (`RelativeWidth`, `RelativeYPosition` or `GridLocation` among others) which leads us to the conclusion that there has been confusion with elements that are very similar and in fact represent the same content.

In Chapter 2.1 we described Support Vector Machines (SVM) as a popular learning algorithm [Bis07], which often produces very good results. However, this is not the case for our dataset, as shown in Table 4.4. For the evaluation we used LibSVM [CL11], which is available in Weka[1] by default. The first thing to notice is the execution time of 259.846 seconds, which is considerable longer than any other algorithm. When comparing the weighted precision computed by Weka, 0.7828, and with the unweighted precision, 0.6676, we see a significant difference. This is due to the fact that LibSVM was unable to classify the non-repeating labels such as `wr-region` or `wr-title`. The precision of LibSVM for non-repeating labels is only 0.3667 while the precision for repeating is 0.9134, which is one of the best we evaluated. While we have seen a significant difference in precision for repeating and non-repeating labels for most learning algorithms, other classifiers were at least able to classify some of the non-repeating labels. Most likely this is due to the fact that there exists only a very small set of positive examples of these labels and, in addition, on most web pages there exist many elements that are visually very similar to the annotated element.

We should also note about the SVM classifier that it does not confuse labels with each other, for example, `wr-item-title` is not confused with `wr-item-description`. In our test set only one instance of `wr-item-description` was classified as `wr-item`. Confusion is most likely to happen between a labelled element and `none`. However, we have to note that this is true for most of the learning algorithms that perform well in our evaluation and not unique to LibSVM.

4.4 Comparison to other systems

In this section, we compare Wres with state-of-the-art commercial systems such as Diffbot[2] and Import.io[3]. We present Diffbot in Section 4.4.1 and reveal that it is

[1] `http://weka.sourceforge.net/doc.packages/LibSVM/weka/classifiers/functions/LibSVM.html`

[2] `https://www.diffbot.com`

[3] `https://www.import.io`

ineffectiveness in web ranking extraction. In Section 4.4.2 we discuss Import.io and evaluate the results generated by this system. Finally, in Section 4.4.3 we present how our system compares to these commercial systems and in which cases our solution has advantages and disadvantages.

In this evaluation we use different setting. With Diffbot and Import.io we cannot compare instances effectively as web pages are rendered on their servers. Therefore we compare the systems on a page-by-page basis, that is, we will record if a system can extract data in full, in part, not at all or if an error occurred during the loading of the page or the extraction process. We consider a page fully extracted if $P_t \geq 0.9 \wedge P_f \leq 0.1$ for each label, where P_t is the rate of *true positives* and P_f is the rate of *false positives*. Since not all labels are present on every page we require this condition only for labels that are present on the page. In addition we present the same comparision separately for each type of ranking.

We use a different dataset for this comparison because at the time of the evaluation some pages of the original dataset (found in Appendix A) were no longer available or had significantly changed. The dataset we used for evaluating Wres, Diffbot and Import.io on a per-page basis can be found in Appendix B.

4.4.1 Diffbot

Diffbot is a commercial and closes-source system that uses computer vision identify and extract data from web pages. Their product offers types of API and a custom API where the user extract additional fields not detected by the default Diffbot APIs. To create a custom API the user has to add fields and specify the CSS selector identifying the field on the page. In contrast to Import.io (see Section 4.4.2) this process is unguided and requires technical knowledge of HTML and CSS. The automatic APIs are described by Diffbot in the following way [Dif17]

Analyze API Determines the page-type for any given URL (and routes it to the appropriate extraction API, where applicable)

Article API For structuring news articles, blog posts and other text-heavy pages

Discussion API For discussion forums and message board threads

Image API For extracting the primary images from pages

Product API For structuring e-commerce product pages

Video API For extracting metadata from video pages

In addition to extracting data Diffbot can also analyse the data to add semantic meaning to the data, for example, identifying people or places in the content. We did not evaluate this feature for this thesis, since it is not relevant to our problem domain.

Based on the description of the available APIs we can already see that Diffbot does not explicitely support extracting web rankings from web pages. Our dataset (see Appendix

Ranking Type	Pages	Errors	Fully	Partly	None
All	100	5	2	0	93
Grid	24	0	0	0	24
List	29	2	0	0	27
Simple Llist	10	2	0	0	8
Table	25	1	0	0	24
Tiling	12	0	2	0	10

Table 4.6: Number of pages were data could be fully, partly or not extracted using Diffbot

B) contains pages that are rankings of products and pages with ranked lists of images and we would expect that Diffbot performs well for these sites. Diffbot is designed to be used using their API and their exist SDK for multiple programming languages. For our analysis we use the JavaScript SDK provided by Diffbot[4] to extract rankings from all 100 pages in our dataset.

Table 4.6 shows in detail the results of our evaluation. Their system could only extract data from two pages, both were lists of images arranged in a *Tiling* ranking. We can conclude this section by stating that Diffbot in its current form is not a suitable system to extract web rankings from web pages.

4.4.2 Import.io

Import.io is a commercial and proprietary web extraction system that can automatically extract repeating data from websites. The system is designed for people without a lot of technical knowledge, it automatically detects features (which Import.io calls columns to resemble spreadsheets) and gives users the option to refine extractors by selecting elements from a rendered web page. Everything can be done in a web browser and there is no need for users to install a browser extension. The extracted data is returned by Import.io is in a spreadsheet format, that is, columns represents features and rows represent instances. When not manually changing the extractor the features are automatically detected and named by their system, for example, for the page IMDb Top 250[5] in our dataset the following features are detected:

- `url`
- `Col 2`
- `Col 2_alt`
- `Col 2_link`
- `Ranktitle 1`

[4] `https://github.com/diffbot/diffbot-js-client`
[5] `http://www.imdb.com/chart/top?ref_=nv_mv_250_6`

- `Ranktitle 2`
- `Ranktitle 2_link`
- `Ranktitle 3`
- `Imdbrating`

For this specific page the feature `Col 2` corresponds to the label `wr-item-thumbnail` from our system presented in Section 3.1 and `Ranktitle 1` and `Ranktitle 2` combined corresponds to `wr-item-title`. This specific page does not contain an element corresponding to `wr-item-description`, but we could equate the row in the output with our `wr-item` feature. Import.io does not support extracting data outside of the repeating elements, therefore we are not able to find matching elements for the `wr-region`, `wr-title` and `wr-item-region` labels from our system.

Import.io allows users to schedule weekly, daily or hourly runs of an extractor and the option to send the results to the user as an email. In addition Import.io has an API that allows executing an extractor and retrieving results as a CSV or JSON file. However, there is currently no possibility to create an extractor using the API, this can be only done by using the graphical user interface Import.io provides.

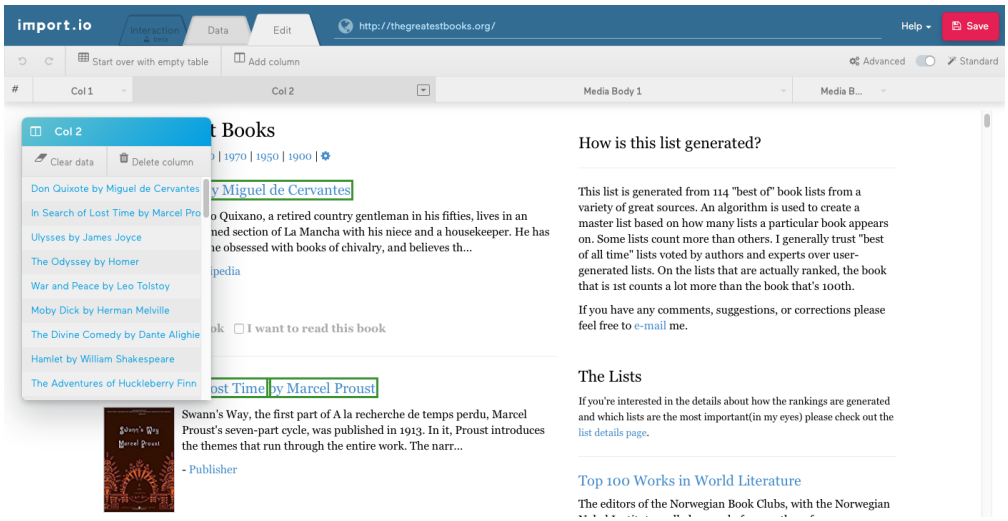


Figure 4.6: Import.io allows users to manually change extractors

As already mentioned it is possible to manually change the extractor by selecting elements on the page, defining which columns should be extracted and renaming the columns. Their system also allows to define XPath [CD99] queries to extract elements from the page and to extract link URLs and image source URLs in addition to the elements text content. To help the automatic extractor it is possible to add multiple URLs for a web page, their system then can extract paginated content. Import.ios interface to manually change an extractor is shown in Figure 4.6 for the website `http://thegreatestbooks.org`.

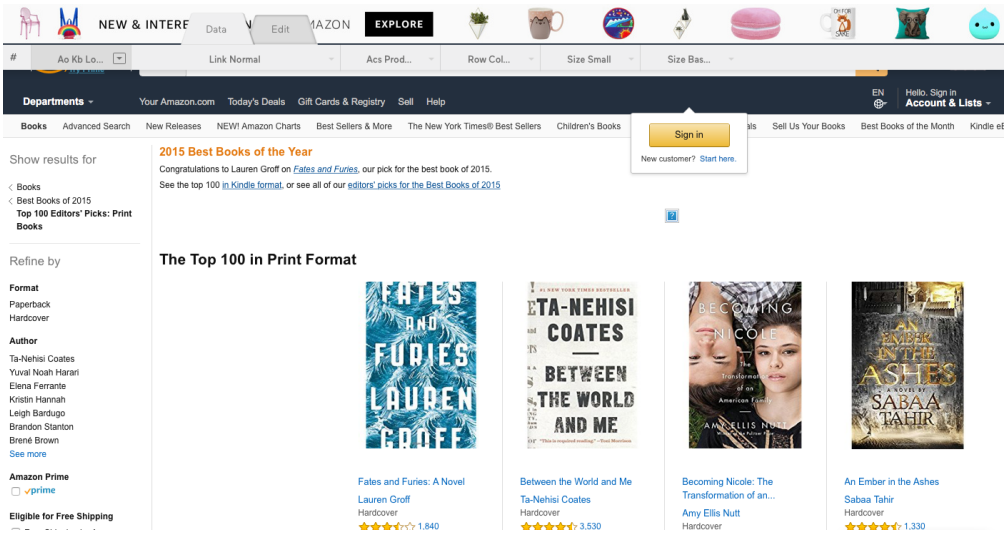


Figure 4.7: Import.io interface does not work for a page on Amazon

Ranking Type	Pages	Errors	Fully	Partly	None	Initial None
All	100	4	53	10	33	19
Grid	24	1	18	1	4	2
List	29	0	13	7	9	6
Simple List	10	0	2	0	8	6
Table	25	2	12	1	10	4
Tiling	12	1	8	1	2	1

Table 4.7: Number of pages were data could be fully, partly or not extracted using Import.io

When evaluating Import.io[6] we found that the interface to manually change the extractor was broken for 53% of pages in our dataset. Figure 4.7 shows a page on Amazon[7] with parts of the Import.io interface missing. In the screenshot the *Data* tab is selected, that is, Import.io should show a preview of the data it is going to extract, but the table is not visible. When selecting the *Edit* tab the tools used to refine selection are also not visible. In the top right corner of the page should be a *Save* button, which is also missing in Figure 4.7. We were only able to save the extractors by manually changing the source code of the page using the developer tools in the Safari browser. By doing this we were able to create extractors for the 53 pages in our dataset that were not working in Import.io.

To create these results we did not use the tools to manually change an extractor provided by Import.io. In this thesis we are interested in

[6] On September 9, 2017

[7] `https://www.amazon.com/b?node=13128468011`

In Table 4.7 we present the results of our evaluation of Import.io. In addition to showing the number of pages where we could fully, partly or not extract data we also show the number of pages where Import.io could not find anything to extract. If the initial extraction does not find any data it is not possible to save the extractor in Import.io. We also note in the table pages that caused an error while creating an extractor. For example, while the page `https://www.infoplease.com/world/world-geography/worlds-14-highest-mountain-peaks-above-8000-meters` loads fine in a normal web browser, when it is loaded in Import.io the browser crashes and needs to be restarted. For the pages from the domain `usnews.com` we needed multiple attempts to create an extractor, because we encountered an *Access Forbidden* error the first few times. Since Import.io is closed source we cannot be sure but we assume that `usnews.com` is blocking Import.io to access its site and Import.io tries to circumvent the block which does not work every time.

When we look at the numbers of fully and partly extracted data we can see that Import.io works best for pages using a *Grid* ranking. 75% of grids could be extracted fully and *4.17%* could be extracted partly. It also works quite well for pages with a *Tiling* ranking by extracting 66.67% fully and 8.33% partly. Import.io does not work very well for *Simple List* rankings, from 80% of the pages in our dataset no data could be extracted. Rankings of the other two types can be extracted a little bit less than half of the time, with 44.83% for *Grid* rankings and 48% for *Table* rankings. Overall Import.io was able to extract data from 53% of pages fully and from 10% partly. No correct data could be extracted from 33% of pages and at the time of our evaluation we were not able to create an extractor for 4% of pages.

4.4.3 Comparison to our system

In Section 4.4.2 we described that works in the browser, giving users the ability to create extractors by loading the page in the main browser window and overlaying toolbars and buttons to modify and save the extractor. We have also shown that approach is prone to conflicts between the page and the toolbars and buttons and that at the time of our evaluation it was not possible to annotate and save 53% of pages using Import.io. Wres uses an extension for Google Chrome (described in Section 3.4.1 to annotate pages that renders toolbars mostly inside Chromes developer tools. When necessary we use the capabilities of the Chrome extension APIs to inject code directly into the page. By using a browser extension we were able to annotate 100% of pages correctly.

To evaluate the dataset (see Appendix B we build a classifier using Weka and Support Vector Machines with the balanced dataset. We then evaluate the built model for each page in the dataset and map the result back to the original DOM node to extract the text content. In detail this process looks like this:

1. Create classifier in Weka with the Support Vector Machines algorithm on the balanced dataset (see Section 4.1.3

2. Save model built by the classifier

Ranking Type	Pages	Errors	Fully	Partly	None
All	100	3	71	22	4
Grid	24	1	14	7	2
List	29	2	23	4	0
Simple List	10	0	9	1	0
Table	25	0	20	5	0
Tiling	12	0	5	5	2

Table 4.8: Number of pages were data could be fully, partly or not extracted using Wres

3. Split full dataset to get one file per page
4. Load model in Weka and re-evaluate it using the split-dataset for each page
5. Map each instance in the predictions generated by Weka to the original DOM node using the page URL and index
6. Compare if text content (or image source for `wr-item-thumbnail`) is the same as in the dataset

As already mentioned at the begining of Section 4.4 we allow small errors when couting a page as fully extracted by Wres.

Table 4.8 shows the number of pages where we could extract data in full, in part or not at all using our Wres system. Overall we could extract the full data from 71% of pages and in part from 22% of pages. Only on 4% of pages we were not able to extract any data at all. This result is very good compared to 53% of pages where the full data and 10% of pages where part of the dat can be extracted using Import.io. Wres works best for pages with *Simple List* rankings, where we could extract the full data from 90% of pages and part of the data from 10% of pages. Our system works also better than Import.io for *Table* rankings (80% fully extracted, 20% partly extracted) and *List* rankings (79.31% fully extracted, 13.79% partly extracted). However, our system could extract from less pages containing *Grid* and *Tiling* than Import.io: 58.33% of *Grid* and 41.67% of *Tiling* rankings where extracted in full.

We conclude this section by providing an overview directly comparing the results from Diffbot, Import.io and Wres. Table 4.9 shows the number of page where we could fully and partly extract web rankings from for all three systems. First, we conclude that Diffbot is not a suitable system to extract web rankings, since it was only able to extract rankings from 2 pages. Based on the marketing material and documentation on their website their main target is to extract data from articles and product detail pages and not web rankings that contain data about multiple entities.

Import.io performs a lot better and can fully extract rankings from 53% of pages and partyl from 10% of pages. Our system Wres outperforms both systems and is able to extract rankings in full from 71% and in part from 22% of pages in our dataset. However, for *Grid* and *Tiling* rankings Import.io is able to correctly extract data from

	Diffbot		Import.io		Wres	
Ranking	**Fully**	**Partly**	**Fully**	**Partly**	**Fully**	**Partly**
All	2	0	53	10	71	22
Grid	0	0	18	1	14	7
List	0	0	13	7	23	4
Simple List	0	0	2	0	9	1
Table	0	0	12	1	20	5
Tiling	2	0	8	1	5	5

Table 4.9: Comparing number of fully and partly extracted web rankings using Diffbot, Import.io and Wres

more pages than our system, while our system achieves better results for *Simple List*, *List* and *Table* rankings. The Import.io product is more flexibel by also providing tools to manually modify an extractor using viusal tools, allowing to achieve a higher accuracy by increasing the amount of human labour. In general Import.io always requires users to manually create an extractor by providing the URL, previewing the results and selecting a column it uses to match data on repeating runs.

Another advantage is that Wres uses a Chrome extension for the annotation process it is more reliable than Import.io. Import.io loads the page in its web-based app and overlays its toolbars and buttons. In our evaluation we found that this works only for 47% of pages correctly. For the other 53% of pages it was not possible to manually modify the extractor to change the elements being extracted from the page.

Lastly we note that Import.io requires manual annotation of extracted features (called columns in Import.io) for each web page. Wres extracts the labels defined in Section 3.1 for each page and is able to extract the content of the full item in addition to the title, description and thumbnail of the item. It can also extract the area of the ranking and the title of the ranking.

CHAPTER 5

Conclusion

This thesis introduces a system to identify web rankings on new and unknown web sites. When evaluating our system in Chapter 4 we found that it works well for repeating labels, that is, labels that exist multiple times on each web ranking, but fails to deliver a high precision for non-repeating labels, that is, labels that exist only a single time on each ranking. In Section 5.1 we will present some ideas on how to improve the accuracy of our system with a focus on non-repeating labels.

In Section 4.4 we compared our system to two commercially available web data extraction systems. The first system, Diffbot, could only extract 2% of web pages correctly and we can conclude that the system was not designed to extract web rankings. We found that our system also outperforms the second system, Import.io. Wres fully extracts rankings from 71% of pages, while Import.io extracts the correct content from 53%. Import.io achieves better results for rankings of the types `grid` and `tiling`, while our system has better results for `list`, `simple-list` and `table`. In addition Wres has several advantages when extracting web rankings to systems that are general purpose extraction systems. With our system, data is correctly classified with the labels defined in Section3.1, while other systems require a mapping to semantic labels. Import.io requires users to create an extractor for each web page, making the approach not fully automatic. In summary we can say that Wres performs well when compared to commercial available general purpose exraction systems.

In addition we also produced an easy to use tool for users to annotate web rankings directly in the browser. Our annotation tool can be adapted without a lot of effort to be used to annotate any kind of object on a web page directly in the browser. The way we built the tool allows us to reuse a lot of the code used to annotate the training set also in the feature extraction tool for new and unseen web pages. By publishing our code using an open source license these tools can be used by others. We wrote our software in a very modular fashion which allowed us to extract certain parts and publish them as separate modules under a very permissable open source license. These libraries include:

domlight.js is a library to highlight DOM elements on a web page using JavaScript. It can be used provide a visual guide to users to indicate that a element has been selected. The style of the highlight can be changed using CSS properties.

seldom.js builds upon *domlight.js* and allows users to select an element in the DOM using a mouse or trackpad. In addition to being able to select an element by clicking on it the library provides a set of tools for the user that allows a refinement of the selection. Thus an user is able to change the selection to the parent, children or sibling elements of the currently selected element.

domcss.js accesses the styles and position of an element computed by the browser and attached them to the element. By attaching styles and position to the DOM tree the information is available to other libraries or modules without specifically communicating with the *domcss.js* libary.

serialdom.js provides developers a way to serialize the DOM of a website in a non-recursive way. By removing recursion from the DOM the library is able to convert the DOM into JSON. It provides an interface to either access the JSON object or the stringified version of the JSON object.

All these libraries mentioned above are available on Github and on NPM, a package manager for Node.js under the very permissive MIT license.

5.1 Future Work

While our results are promising there exist many areas in which our system can be improved to produce more accurate results. One area of improvement is the classification of non-repeating labels. As the evaluation in Chapter 4 has shown our system can only classify a small percentage of these labels correctly. We identified one problem might be the number of positive examples compared to the number of examples for non-repeating elements and negative examples (labelled with `none`). However, this can not be easily solved, since increasing the number of positive examples for non-repeating elements would mean increasing the number of training web pages, which in turn would also increase the number of elements with repeating and `none` labels. Increasing the training set would also mean more work, because the web sites in the training set have to be manually labelled by a person.

Another reason we identified was that often elements are wrapped in other elements with similar visual appearances. Figure 5.1 illustrates this by highlighting three different DOM elements that could be labelled with `wr-item-region`. The first element is a wrapper element that includes the ranking item region and a filter bar (see Figure 5.1), and the other two elements both encompass the same spatial area. For the latter two elements the size and position based attributes will be the same, but even for the first attribute the difference for attributes such as `RelativeHeight`, `RelativeYPosition` or `AspectRatio` will only show a minimal difference.

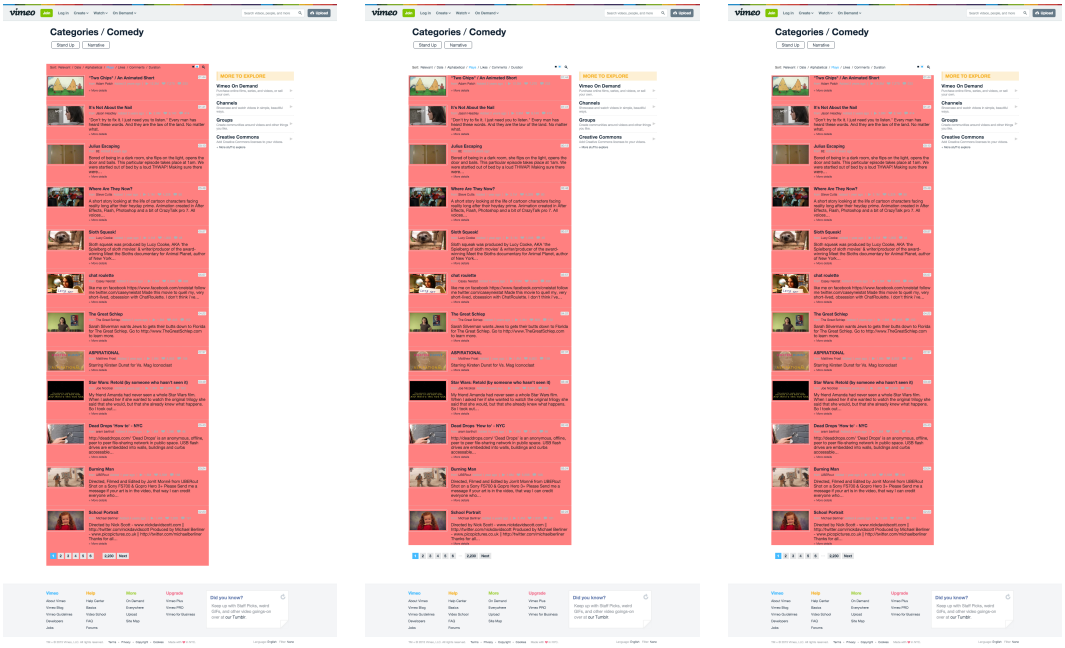


Figure 5.1: On Vimeo three visual very similar elements could identify wr-item-region.

We see three possible solutions for this problem. The first solution would be to annotate all possible elements for a label during the annotation process. In the above example this would mean labelling all three elements with `wr-item-region`. This solution is easy to implement but would require significant work by the user to correctly identify and annotate all possible elements for a label. There also could be edge cases such as two elements exactly overlapping that forbid the annotation using a visual annotation tool (such as the one we presented in Section 3.4). Instead the user would have to look through the source code to find and select all possible elements for a label. While theoretically possible we consider this approach inpractical for real-world scenarios.

A solution that would not require additional work by the user would be to remove duplicate elements from the simplied DOM representation (see Section 4.1.1). We define a duplicate element as an element that describes the same content and has the same spatial features as another element. This would only remove exact duplicates, but leave elements that are just similar in the dataset. One major problem with this solution is that there could be complications with other visual properties, for example, background with transparency or relative font sizes.

The third solution we propose is to not remove duplicate elements but automatically add the label to all elements that are exactly the same or even just very similar. Therefore if a user labels an element the system would find during pre-processing the data find all elements that cover the exact or nearly the exact same area and apply the label to these elements as well. We think this solution is preferable to the other two since it leaves the

original DOM tree intact and does not involve additional work by users.

Further improvements are likely possible by introducing post-processing to our system. We could introduce heuristics that do a sanity check on the results of the classifier, for example, by verifying that an element identfied as `wr-item-title` has an ancestor identified as `wr-item` and so on. Such a process could also decrease the number of correctly predicted instances when the percentage of true positive instances labelled with `wr-item-title` is higher than the number of true positive instances labelled with `wr-item`. However, such a measure could be effective if for the goal of the system it is more important to have true positives than false positives.

Many systems that use machine learning combine the results of different classifiers to improve to overall accuracy. For example, [KHF+13] first computes the relative frequency from each classifier output and then sums up those relative frequencies to build the combined results.

APPENDIX A

Dataset

Table A.1 shows the full dataset including the title, URL, ranking type of each web page and the date it was added to the dataset. More information about the dataset is given in Section 4.1.

Title	**URL**	**Ranking Type**	**Date**
Most Popular Photos on 500px Right Now	`https://500px.com/popular`	tiling	18/12/2015
Die besten Alpenpässe mit dem Rennrad - Sport & Freizeit - 7Lists	`http://www.7lists.de/list/die-besten-alpenpsse-mit-dem-rennrad`	simple-list	20/05/2015
Die Top Nvidia - Grafikkarten der 500er Serie - Technik - 7Lists	`http://www.7lists.de/list/die-top-nvidia-grafikkarten-der-500er-serie`	simple-list	20/05/2015
Amazon.com: Top 100 Editors' Picks: Print Books: Books	`http://www.amazon.com/b?node=13128468011`	grid	18/12/2015
iTunes - Browse the top paid apps on the App Store - Apple	`https://www.apple.com/itunes/charts/paid-apps/`	grid	20/12/2015
MBLWHOI Library : Free Books : Free Texts : Download & Streaming : Internet Archive	`https://archive.org/details/MBLWHOI?&sort=-downloads&page=4`	tiling	19/12/2015

Continued on next page

Table A.1 – continued from previous page

Title	URL	Ranking Type	Date
Titles - Art of the Title	http://www.artofthetitle.com/titles/view/grid/sort/released/	grid	19/12/2015
Titles - Art of the Title	http://www.artofthetitle.com/titles/view/list/sort/released/	grid	19/12/2015
Marvel Cinematic Universe Moviesat the Box Office - Box Office Mojo	http://www.boxofficemojo.com/franchises/chart/?view=main&id=avengers.htm&sort=opengross&order=DESC&p=.htm	table	21/05/2015
Box Office Prophets Awards Tracking Page	http://www.boxofficeprophets.com/awards/	list	19/12/2015
Top 100 Richest People In The World | Celebrity Net Worth	http://www.celebritynetworth.com/list/top-100-richest-people-in-the-world/	list	20/12/2015
Vergleich: Fernseher 32 bis 42 Zoll im Test - CHIP	http://www.chip.de/bestenlisten/Bestenliste-Fernseher-32-bis-42-Zoll--index/index/id/1023/	table	21/05/2015
The World Factbook	https://www.cia.gov/library/publications/the-world-factbook/rankorder/2233rank.html	table	21/05/2015
Top 10 Superheroines	http://www.comicvine.com/profile/ferdi47/lists/top-10-superheroines/13205/	list	20/12/2015
Fernseher günstig online kaufen	http://www.cyberport.at/fernseher	list	20/05/2015

Continued on next page

Table A.1 – continued from previous page

Title	URL	Ranking Type	Date
|Art| |Kanal_Name| Videos |Seite|	http://www.dailymotion.com/de/visited/channel/music/1	list	18/12/2015
Bilder und Videos suchen: lion	https://de.fotolia.com/search?k=lion&filters%5Bcontent_type%3Aphoto%5D=1&search-submit=Suchen	tiling	19/12/2015
Kaffee Wien	https://de.foursquare.com/explore?mode=url&near=Wien&q=kaffee	list	20/05/2015
Browse Art - DeviantArt	http://www.deviantart.com/browse/all/	tiling	19/12/2015
iPad, Tablets and eBook Readers | eBay	http://www.ebay.com/sch/iPads-Tablets-eBook-Readers-/171485/i.html	list	20/05/2015
Awards Leaderboard: Top Movies of 2015 « Movie & TV News and Interviews - Rotten Tomatoes	http://editorial.rottentomatoes.com/article/awards-leaderboard-2015/	list	18/12/2015
List of Marvel Cinematic Universe films - Wikipedia, the free encyclopedia	https://en.wikipedia.org/wiki/List_of_Marvel_Cinematic_Universe_films	table	21/05/2015
The Best Singles of All Time - everyHit.com	http://www.everyhit.com/bestever.html	table	21/05/2015
Top Movies of All Time, Top 100 Movies, Most Popular Movies List by FilmCrave	http://www.filmcrave.com/list_top_movie.php	list	19/12/2015
Flickr - Photo Sharing!	https://www.flickr.com/explore	tiling	21/05/2015
The World's Billionaires - Forbes	http://www.forbes.com/billionaires/	grid	20/12/2015
The World's Billionaires List - Forbes	http://www.forbes.com/billionaires/list/#version:static	table	21/05/2015

Continued on next page

Table A.1 – continued from previous page

Title	URL	Ranking Type	Date
2015 Driver Standings	http://www.formula1.com/content/fom-website/en/championship/results/2015-driver-standings.html	table	21/05/2015
Fotki: Share and Print Your Photos | Fotki.com, photo and video sharing made easy.	http://www.fotki.com/	tiling	20/12/2015
Skyline Stock Footage and Video. 5,573 skyline royalty free videos and movie clips available to download from over 20 stock motion brands.	http://www.fotosearch.com/video-footage/skyline.html	grid	19/12/2015
Funny Or Die - All Videos - Most Viewed This Week	http://www.funnyordie.com/browse/videos/all/all/most_viewed/this_week	grid	19/12/2015
Reviews and News Articles - GameRankings	http://www.gamerankings.com/browse.html	table	19/12/2015
Solid State Drives (SSDs) Preisvergleich | Geizhals Österreich	http://geizhals.at/?cat=hdssd	table	21/05/2015
Bestpix Bilder Und Fotos | Getty Images	http://www.gettyimages.at/fotos/bestpix?excludenudity=false&family=editorial&page=1&phrase=bestpix&sort=mostpopular	tiling	19/12/2015
GifBoom - Express yourself in motion, make animated GIFs with your iPhone or Android	http://gifboom.com/	tiling	19/12/2015

Continued on next page

Table A.1 – continued from previous page

Title	URL	Ranking Type	Date
Best Fiction 2015 - Goodreads Choice Awards	https://www.goodreads.com/choiceawards/best-fiction-books-2015	grid	18/12/2015
Best Archives and Preservation Programs | Top Library Information Science Schools | US News Best Graduate Schools	http://grad-schools.usnews.rankingsandreviews.com/best-graduate-schools/top-library-information-science-programs/library-preservation-rankings	simple-list	21/05/2015
Korean Box Office @ HanCinema :: The Korean Movie and Drama Database	http://www.hancinema.net/korean_boxoffice.php	table	19/12/2015
Video Games, Wikis, Cheats, Walkthroughs, Reviews, News & Videos - IGN	http://www.ign.com/	simple-list	19/12/2015
PC Game Reviews: Latest PC Games - IGN	http://www.ign.com/games/reviews/pc?sortBy=score&sortOrder=desc	list	20/05/2015
ImageShack - Discover	https://imageshack.com/discover	tiling	20/12/2015
IMDb Top 250 - IMDb	http://www.imdb.com/chart/top?ref_=nv_ch_250_4	table	15/05/2015
IMDb | Year in Review | Top 10 Movies of 2013	http://www.imdb.com/year-in-review/top-user-rated-movies-of-2013/	list	20/05/2015
Imgur: The most awesome images on the Internet	http://imgur.com/hot/viral	grid	18/12/2015
World's 14 Highest Mountain Peaks (above 8,000 meters)	http://www.infoplease.com/ipa/A0777280.html	table	21/05/2015

Continued on next page

Table A.1 – continued from previous page

Title	URL	Ranking Type	Date
Ten Countries with the Highest Population in the World	http://www.internetworldstats.com/stats8.htm	table	21/05/2015
Stock Photos & Illustrations - iStock	http://www.istockphoto.com/search/lightbox/18308815?facets=%7B%22pageNumber%22:1,%22perPage%22:100,%22abstractType%22:%5B%22photos%22,%22illustrations%22,%22video%22,%22audio%22%5D,%22order%22:%22bestMatch%22,%22lightboxID%22:%5B18308815%5D,%22additionalAudio%22:%22true%22,%22f%22:true%7D	grid	19/12/2015
Calendar | It's a Shape Christmas	http://itsashapechristmas.co.uk/	tiling	19/12/2015
Find and Browse Music on Last.fm - Last.fm	http://www.last.fm/music	simple-list	18/12/2015
feredir's Library - Users at Last.fm	http://www.last.fm/user/feredir/library/tracks?from=2015-01-01&rangetype=year	simple-list	19/12/2015
Browse Films - Letterboxd	http://letterboxd.com/films/popular/this/week/	grid	18/12/2015
Browse Films - Letterboxd	http://letterboxd.com/films/popular/this/week/size/large/	grid	18/12/2015
Best Selling PC Games list	http://www.listal.com/list/bestselling-pc-games	list	20/05/2015

Continued on next page

Table A.1 – continued from previous page

Title	URL	Ranking Type	Date
Best Movies - Metacritic	http://www.metacritic.com/browse/movies/score/metascore/90day	table	21/05/2015
Minnesota Cities by Population	http://www.minnesota-demographics.com/cities_by_population	simple-list	19/12/2015
Zustellung Wien 1030 - Essen online bestellen | Mjam	http://www.mjam.net/wien/1030/	list	20/05/2015
List of Best Superheroines	http://www.mytopdozen.com/Best_Superheroines.html	simple-list	20/12/2015
NYPL | Recommendations	http://www.nypl.org/browse/recommendations/staff-picks/	tiling	19/12/2015
OpenCritic - The top critics in gaming. All in one place.	http://opencritic.com/browse?page=0&sort=score&platforms=%5B%5D&genres=%5B%5D&date=Released	grid	19/12/2015
Panoramio - Photos of the World	http://www.panoramio.com/map/#lt=12.508973&ln=-86.702503&z=1&k=2&a=1&tab=1&pl=all	grid	19/12/2015
Best Pony - crowdranking	http://panzi.crowdranking.com/crowdrankings/t89g0--best-pony#?view=tile	tiling	21/05/2015
Last Year's Winners ÂăÂăPerformance Marketing Awards 2016 26 April 2016	https://www.performancemarketingawards.co.uk/2016/last-years-winners/	grid	18/12/2015
Mountains Pictures, Images & Photos | Photobucket	http://photobucket.com/images/mountains	grid	18/12/2015

Continued on next page

Table A.1 – continued from previous page

Title	URL	Ranking Type	Date
(41) Tattoos And Body Art on Pinterest | Irezumi tattoos, Lower back tattoos and Nose piercings	https://www.pinterest.com/explore/tattoos/	tiling	18/12/2015
The 100 Best Tracks of 2015 | Pitchfork	http://pitchfork.com/features/staff-lists/9765-the-100-best-tracks-of-2015/10/	list	18/12/2015
Best New Albums | Pitchfork	http://pitchfork.com/reviews/best/albums/	list	18/12/2015
Editor's Choice Photos	https://pixabay.com/en/editors_choice/	tiling	20/12/2015
Best College Football Coach of All Time | List of Greatest NCAA Head Coaches	http://www.ranker.com/crowdranked-list/best-college-football-coaches-of-all-time	simple-list	21/05/2015
The Best Characters in the Star Wars Universe	http://www.ranker.com/crowdranked-list/the-best-characters-in-the-star-wars-universe	list	20/12/2015
Top Jobs You Can Get Without a College Degree	http://www.ranker.com/list/great-jobs-that-don_t-require-a-college-degree/american-jobs	simple-list	21/05/2015
The Most Overrated Actors of All Time	http://www.ranker.com/list/most-overrated-actors-of-all-time/ranker-film	simple-list	21/05/2015
The Most Trusted & Trustworthy Celebrities	http://www.ranker.com/list/most-trustworthy-celebrities/celebrity-lists	simple-list	21/05/2015

Continued on next page

Table A.1 – continued from previous page

Title	URL	Ranking Type	Date
Top 100 Action & Adventure Movies - Rotten Tomatoes	http://www.rottentomatoes.com/top/bestofrt/top_100_action__adventure_movies/?category=1	table	21/05/2015
SATURN Shop - 50,1 Zoll bis 59 Zoll (149 cm) - LED TV - Unterhaltungselektronik	http://www.saturn.at/mcs/productlist/bis-47-Zoll-%28117-cm%29,90452,381451.html?langId=-15	list	20/05/2015
Top Ten Longest Rivers in the World List - Fun Science Facts for Kids	http://www.sciencekids.co.nz/sciencefacts/topten/longestrivers.html	table	21/05/2015
Animal & Wildlife Stock Photos : Shutterstock Stock Photography	http://www.shutterstock.com/cat-1-Animals-Wildlife.html	tiling	19/12/2015
Newest 'symfony2' Questions - Stack Overflow	http://stackoverflow.com/questions/tagged/symfony2	list	20/05/2015
Stereogum	http://www.stereogum.com/	simple-list	18/12/2015
Cables - Mac Accessories - Apple Store (U.S.)	http://store.apple.com/us/mac/mac-accessories/cables	grid	21/05/2015
The Numbers - Weekend Box Office Chart for December 11th, 2015	http://www.the-numbers.com/weekend-box-office-chart	table	19/12/2015
Best Songs of 2015 - Top Ten List - TheTopTens.com	http://www.thetoptens.com/2015-songs/	list	18/12/2015
Top Ten Star Wars Characters - TheTopTens.com	http://www.thetoptens.com/star-wars-characters/	list	20/12/2015
November 2014 | TOP500 Supercomputer Sites	http://www.top500.org/lists/2014/11/	table	21/05/2015
Trending | Tumblr	https://www.tumblr.com/explore/trending	tiling	19/12/2015

Continued on next page

Table A.1 – continued from previous page

Title	URL	Ranking Type	Date
Liberal Arts Colleges Where the Most Accepted Students Enroll - US News	http://www.usnews.com/education/best-colleges/articles/2015/01/21/liberal-arts-colleges-where-the-most-accepted-students-enroll	table	21/05/2015
Universities That Attract the Most International Students - US News	http://www.usnews.com/education/best-colleges/the-short-list-college/articles/2014/02/05/universities-that-attract-the-most-international-students	table	21/05/2015
10 Colleges Where In-State Students Pay the Most Tuition - US News	http://www.usnews.com/education/best-colleges/the-short-list-college/articles/2014/10/28/10-colleges-where-in-state-students-pay-the-most-tuition	table	21/05/2015
Best Charter School rankings | Top Charter Schools | US News	http://www.usnews.com/education/best-high-schools/national-rankings/charter-school-rankings	table	21/05/2015
Comedy Videos on Vimeo	https://vimeo.com/categories/comedy/videos/sort:plays/format:detail	list	20/05/2015
Comedy Videos on Vimeo	https://vimeo.com/categories/comedy/videos/sort:plays/format:thumbnail	grid	20/05/2015

Continued on next page

Table A.1 – continued from previous page

Title	URL	Ranking Type	Date
The Countries with the Lowest Death Rates - WorldAtlas.com	http://www.worldatlas.com/articles/the-countries-with-the-lowest-death-rates.html	simple-list	19/12/2015
City Populations, Largest Cities of the World - Worldatlas.com	http://www.worldatlas.com/citypops.htm	simple-list	19/12/2015
Europe Facts, Capital Cities, Currency, Flag, Language, Landforms, Land Statistics, Largest Cities, Population, Symbols	http://www.worldatlas.com/webimage/countrys/eufacts.htm	simple-list	19/12/2015
Europe Landforms and Land Statistics - Europe Landforms, Land Statistics	http://www.worldatlas.com/webimage/countrys/eulandst.htm	simple-list	19/12/2015
A Review of Online Time-Management Programs - WSJ	http://www.wsj.com/news/articles/SB10001424052748703410004575029402709924656	simple-list	21/05/2015
Food in Wien - Yelp	http://www.yelp.com/c/wien/food	tiling	20/12/2015
The Best 10 Food in Alsergrund, Wien, Austria	http://www.yelp.com/search?cflt=food&find_loc=Alsergrund%2C+Wien	list	20/12/2015
#TaylorSwift - YouTube	https://www.youtube.com/channel/UCPC0L1d253x-KuMNwa05TpA/videos	grid	20/05/2015
Women's Shoes | Zappos.com FREE Shipping	http://www.zappos.com/womens-shoes~1i7	grid	20/05/2015

Table A.1: Overview of the dataset used in Wres

APPENDIX B

Dataset for Per-Page Evaluation

Table A.1 shows the full dataset used to compare Wres to Import.io and Diffbot in Section 4.4.3 including the title, URL, ranking type of each web page and the date it was added to the dataset. We had to create another dataset when performing the evaluation of Import.io and Diffbot many of the pages in the orignal dataset (see Appendex A) where no longer available or had significantly changed since first creating the dataset.

Title	**URL**	**Ranking Type**	**Date**
Most Popular Photos / 500px	`https://500px.com/popular`	tiling	09/09/2017
Amazon.com: Top 100 Editors' Picks: Print Books: Books	`https://www.amazon.com/b?node=13128468011`	grid	09/09/2017
iTunes - Browse the top paid apps on the App Store - Apple	`https://www.apple.com/itunes/charts/paid-apps/`	grid	09/09/2017
iTunes - Browse the top music downloads - Apple	`https://www.apple.com/itunes/charts/songs/`	grid	09/09/2017
Power & Cables - Mac Accessories - Apple	`https://www.apple.com/shop/mac/mac-accessories/power-cables`	grid	09/09/2017
Best Books of 2016 : NPR	`http://apps.npr.org/best-books-2016/`	tiling	09/09/2017

Continued on next page

Table B.1 – continued from previous page

Title	URL	Ranking Type	Date
MBLWHOI Library : Free Books : Free Texts : Download & Streaming : Internet Archive	https://archive.org/details/MBLWHOI?&sort=-downloads&page=5	tiling	09/09/2017
Titles - Art of the Title	http://www.artofthetitle.com/titles/view/grid/sort/released/	grid	09/09/2017
Titles - Art of the Title	http://www.artofthetitle.com/titles/view/list/sort/released/	grid	09/09/2017
Top Billboard 200 Albums - Year-End 2016 | Billboard	http://www.billboard.com/charts/year-end/2016/top-billboard-200-albums	list	09/09/2017
Marvel Cinematic Universe Movies at the Box Office - Box Office Mojo	http://www.boxofficemojo.com/franchises/chart/?view=main&id=avengers.htm&sort=opengross&order=DESC&p=.htm	table	09/09/2017
Box Office Prophets Awards Tracking Page	http://www.boxofficeprophets.com/awards/	table	09/09/2017
Top 100 Richest People In The World | Celebrity Net Worth	https://www.celebritynetworth.com/list/top-100-richest-people-in-the-world/	list	09/09/2017
Vergleich: Fernseher 32 bis 43 Zoll im Test - CHIP	http://www.chip.de/bestenlisten/Bestenliste-Fernseher-32-bis-43-Zoll--index/index/id/1023/	table	09/09/2017

Continued on next page

Table B.1 – continued from previous page

Title	URL	Ranking Type	Date
The World Factbook - Central Intelligence Agency	https://www.cia.gov/library/publications/the-world-factbook/rankorder/2233rank.html	table	09/09/2017
Top 10 Superheroines	https://comicvine.gamespot.com/profile/ferdi47/lists/top-10-superheroines/13205/	list	09/09/2017
Fernseher kaufen | LCD/LED-TVs günstig online bei Cyberport	https://www.cyberport.at/tv-audio/fernseher/liste.html	list	09/09/2017
|Art| |Kanal_Name| Videos |Seite|	http://www.dailymotion.com/de/visited/channel/music/1	list	09/09/2017
Bilder und Videos suchen: lion	https://de.fotolia.com/search?k=lion&filters%5Bcontent_type%3Aphoto%5D=1&search-submit=Suchen	tiling	09/09/2017
Kaffee Wien	https://de.foursquare.com/explore?ll=48.259094%2C16.333893&mode=url&near=Wien&q=kaffee	list	09/09/2017
IGN Deutschland	http://de.ign.com/	list	09/09/2017
Browse What's Hot | DeviantArt	https://www.deviantart.com/whats-hot/?offset=0	tiling	09/09/2017
500 Greatest Albums | Rolling Stone	https://www.discogs.com/lists/500-Greatest-Albums-Rolling-Stone/140759	simple-list	09/09/2017
Tablets & eBook Readers | eBay	https://www.ebay.com/b/Tablets-eBook-Readers/171485/bn_320042	list	09/09/2017

Continued on next page

Table B.1 – continued from previous page

Title	URL	Ranking Type	Date
Awards Leaderboard: Top Movies of 2015 « Rotten Tomatoes ÃćâĆňâĂIJ Movie and TV News	http://editorial.rottentomatoes.com/article/awards-leaderboard-2015/	list	09/09/2017
US States: Area and Ranking - EnchantedLearning.com	http://www.enchantedlearning.com/usa/states/area.shtml	table	09/09/2017
The Best Singles of All Time - everyHit.com	http://www.everyhit.com/bestever.html	table	09/09/2017
Top Movies of All Time, Top 100 Movies, Most Popular Movies List by FilmCrave	http://www.filmcrave.com/list_top_movie.php	list	09/09/2017
Explore | Flickr	https://www.flickr.com/explore	tiling	09/09/2017
World Football / Soccer Clubs Ranking - FootballDatabase	http://footballdatabase.com/ranking/world/1	list	09/09/2017
The World's Billionaires List	https://www.forbes.com/billionaires/list/#version:static	table	09/09/2017
2017 FORMULA 1 PIRELLI BELGIAN GRAND PRIX - QUALIFYING	https://www.formula1.com/en/results.html/2017/races/970/belgium.html	table	09/09/2017
2017 Constructor Standings	https://www.formula1.com/en/results.html/2017/team/970/belgium/qualifying.html	table	09/09/2017
Fotki: Share and Print Your Photos | Fotki.com, photo and video sharing made easy.	http://www.fotki.com/	tiling	09/09/2017

Continued on next page

Table B.1 – continued from previous page

Title	URL	Ranking Type	Date
Skyline Stock Footage and Videos. 5,488 skyline royalty free video and movie clips available to search from over 20 buyout footage publishers.	http://www.fotosearch.com/video-footage/skyline.html	grid	09/09/2017
All Videos: Most Viewed This Week | Funny Or Die	http://www.funnyordie.com/browse/videos/all/all/most_viewed/this_week	grid	09/09/2017
Reviews and News Articles - GameRankings	http://www.gamerankings.com/browse.html	list	09/09/2017
Solid State Drives (SSD) Preisvergleich | Geizhals ÃČâĂŞsterreich	https://geizhals.at/?cat=hdssd	list	09/09/2017
Bestpix Bilder und Fotos | Getty Images	http://www.gettyimages.at/fotos/bestpix?excludenudity=false&family=editorial&page=1&phrase=bestpix&sort=mostpopular	tiling	09/09/2017
Best Fiction 2015 ÃćâĆňâĂİ Goodreads Choice Awards	https://www.goodreads.com/choiceawards/best-fiction-books-2015	grid	09/09/2017
Korean Box Office @ HanCinema :: The Korean Movie and Drama Database	http://www.hancinema.net/korean_boxoffice.php	table	09/09/2017
PC Game Reviews - IGN	http://www.ign.com/reviews/games/pc?sortBy=score&sortOrder=desc	list	09/09/2017
IMDb Top 250 - IMDb	http://www.imdb.com/chart/top?ref_=nv_ch_250_4	table	09/09/2017

Continued on next page

Table B.1 – continued from previous page

Title	URL	Ranking Type	Date
IMDb | Year in Review | Top 10 Movies of 2013	http://www.imdb.com/year-in-review/top-user-rated-movies-of-2013/	list	09/09/2017
Imgur: The most awesome images on the Internet	http://imgur.com/hot/viral	grid	09/09/2017
World's 14 Highest Mountain Peaks (above 8,000 meters)	https://www.infoplease.com/world/world-geography/worlds-14-highest-mountain-peaks-above-8000-meters	table	09/09/2017
Ten Countries with the Highest Population in the World	http://www.internetworldstats.com/stats8.htm	table	09/09/2017
Vast and expansive views Pinnwand	http://www.istockphoto.com/at/collaboration/boards/xmZhLPHOJEOCV8YNd4AeBw	grid	09/09/2017
Best Laptops of 2017 - Windows Laptops, Chromebooks and Macs	https://www.laptopmag.com/best-laptops	grid	09/09/2017
Find and Browse Music on Last.fm ÃćâĆňâĂİ Last.fm	https://www.last.fm/music	simple-list	09/09/2017
feredirÃćâĆňâĎćs Library ÃćâĆňâĂİ Users at Last.fm	https://www.last.fm/user/feredir/library/tracks?from=2015-01-01&rangetype=year	table	09/09/2017
ÃćâĆňÅ¡Browse Films ÃćâĆňÂć Letterboxd	https://letterboxd.com/films/popular/this/week/	grid	09/09/2017
ÃćâĆňÅ¡Browse Films ÃćâĆňÂć Letterboxd	https://letterboxd.com/films/popular/this/week/size/large/	grid	09/09/2017
Best Selling PC Games list	http://www.listal.com/list/bestselling-pc-games	list	09/09/2017

Continued on next page

Table B.1 – continued from previous page

Title	URL	Ranking Type	Date
Lonely Planet's 100 Best Cities in the World - How many have you visited?	http://www.listchallenges.com/lonely-planets-100-best-cities-in-the-world	grid	09/09/2017
Best Movies - Metacritic	http://www.metacritic.com/browse/movies/score/metascore/90day	simple-list	09/09/2017
Minnesota Cities by Population	https://www.minnesota-demographics.com/cities_by_population	simple-list	09/09/2017
Lieferservice Wien 1030 || online Essen bestellen mit MJAM	https://www.mjam.net/lieferservice/wien/1030/	list	09/09/2017
Die besten Filme | moviepilot.de	http://www.moviepilot.de/filme/beste	list	09/09/2017
Nomad List - Best Cities to Live and Work Remotely for Digital Nomads	https://nomadlist.com/#view=list	table	09/09/2017
Best Pony - crowdranking	http://panzi.crowdranking.com/crowdrankings/t89g0--best-pony#?view=tile	grid	09/09/2017
Last Year's Winners - ÂăPerformance Marketing Awards 2016	https://performancemarketingawards.co.uk/2016/last-years-winners/	grid	09/09/2017
mountains Pictures, Images & Photos | Photobucket	http://photobucket.com/images/mountains	grid	09/09/2017
Die besten 25+ Tattoos Ideen auf Pinterest | Tatoo, Tatuajes und Fischschuppen tattoo	https://www.pinterest.at/explore/tattoos/	tiling	09/09/2017
Best New Albums | Pitchfork	http://pitchfork.com/reviews/best/albums/	grid	09/09/2017
Editor's Choice - Photos	https://pixabay.com/en/editors_choice/	tiling	09/09/2017

Continued on next page

Table B.1 – continued from previous page

Title	URL	Ranking Type	Date
The Best Books of 2016 So Far - PowellÃćâĆňâĎćs Midyear Roundup - Powell's Books	http://www.powells.com/best-books-of-2016-midyear-roundup	grid	09/09/2017
Best College Football Coach of All Time | List of Greatest NCAA Head Coaches	http://www.ranker.com/crowdranked-list/best-college-football-coaches-of-all-time	list	09/09/2017
Best Game of Thrones Characters | Ranking All GoT Characters	http://www.ranker.com/crowdranked-list/best-game-of-thrones-characters	simple-list	09/09/2017
The Best Characters in the Star Wars Universe	http://www.ranker.com/crowdranked-list/the-best-characters-in-the-star-wars-universe	list	09/09/2017
Top Jobs You Can Get Without a College Degree	http://www.ranker.com/list/great-jobs-that-don_t-require-a-college-degree/american-jobs	simple-list	09/09/2017
The Most Overrated Actors of All Time	http://www.ranker.com/list/most-overrated-actors-of-all-time/ranker-film	list	09/09/2017
The Most Trusted & Trustworthy Celebrities	http://www.ranker.com/list/most-trustworthy-celebrities/celebrity-lists	list	09/09/2017
Top 100 Action & Adventure Movies - Rotten Tomatoes	https://www.rottentomatoes.com/top/bestofrt/top_100_action__adventure_movies/?category=1	table	09/09/2017

Continued on next page

Table B.1 – continued from previous page

Title	URL	Ranking Type	Date
Top Ten Longest Rivers in the World List - Fun Science Facts for Kids	http://www.sciencekids.co.nz/sciencefacts/topten/longestrivers.html	table	09/09/2017
Animals/wildlife Stock Images, Royalty-Free Images & Vectors | Shutterstock	https://www.shutterstock.com/category/animals-wildlife	grid	09/09/2017
100 Tallest Completed Buildings in the World - The Skyscraper Center	https://www.skyscrapercenter.com/buildings	table	09/09/2017
Bestenlisten - sport.orf.at	http://sport.orf.at/fussball/competitions/comp/119/best/	simple-list	09/09/2017
FuÃČÅÿball Bundesliga - Torschützen | Karten | Sperren - Bundesliga - Fuball - sportschau.de	http://www.sportschau.de/fussball/bundesliga/tore/index.html	table	09/09/2017
Newest 'symfony' Questions - Stack Overflow	https://stackoverflow.com/questions/tagged/symfony	list	09/09/2017
Stereogum	http://www.stereogum.com/	list	09/09/2017
The Numbers - Weekend Box Office Chart for August 18th, 2017	http://www.the-numbers.com/weekend-box-office-chart	table	09/09/2017
The Greatest Books: The Best Books - 1 to 50	http://thegreatestbooks.org/	list	09/09/2017
Top 100 Richest Actors | The Richest	http://www.therichest.com/top-lists/top-100-richest-actors/	table	09/09/2017
Best Songs of 2015 - Top Ten List - TheTopTensÃĆÂő	https://www.thetoptens.com/2015-songs/	list	09/09/2017
Top Ten Star Wars Characters - TheTopTensÃĆÂő	https://www.thetoptens.com/star-wars-characters/	list	09/09/2017

Continued on next page

Table B.1 – continued from previous page

Title	URL	Ranking Type	Date
November 2014 | TOP500 Supercomputer Sites	https://www.top500.org/lists/2014/11/	table	09/09/2017
Trending | Tumblr	https://www.tumblr.com/explore/trending	tiling	09/09/2017
Best Archives and Preservation Programs Ranked in 2017	https://www.usnews.com/best-graduate-schools/top-library-information-science-programs/library-preservation-rankings	simple-list	09/09/2017
10 Colleges With the Highest Tuition for In-State Students | The Short List: Colleges | US News	https://www.usnews.com/education/best-colleges/the-short-list-college/articles/2016-05-03/10-colleges-with-the-highest-tuition-for-in-state-students	table	09/09/2017
Best Charter High Schools in America | US News	https://www.usnews.com/education/best-high-schools/national-rankings/charter-school-rankings	list	09/09/2017
Watch, rent, and buy comedy videos online | Vimeo	https://vimeo.com/categories/comedy/videos/sort:plays/format:detail	grid	09/09/2017
Lowest Death Rates Worldwide - WorldAtlas.com	http://www.worldatlas.com/articles/the-countries-with-the-lowest-death-rates.html	table	09/09/2017
City Populations, Largest Cities of the World - Worldatlas.com	http://www.worldatlas.com/citypops.htm	table	09/09/2017

Continued on next page

Table B.1 – continued from previous page

Title	URL	Ranking Type	Date
Europe Facts, Capital Cities, Currency, Flag, Language, Landforms, Land Statistics, Largest Cities, Population, Symbols	http://www.worldatlas.com/webimage/countrys/eufacts.htm	simple-list	09/09/2017
Europe Landforms and Land Statistics - Europe Landforms, Land Statistics	http://www.worldatlas.com/webimage/countrys/eulandst.htm	simple-list	09/09/2017
Food in Wien - Yelp	https://www.yelp.com/c/wien/food	tiling	09/09/2017
The Best 10 Food within 5 miles of Alsergrund, Vienna, Wien, Austria - Last Updated August 2017 - Yelp	https://www.yelp.com/search?cflt=food&find_loc=Alsergrund%2C	+Wien list	09/09/2017
Taylor Swift - Topic - YouTube fi grid	09/09/2017		
Shoes, Women | Shipped Free at Zappos	http://www.zappos.com/women-shoes/CK_XAcABAeICAgEY.zso?s=goliveRecentSalesStyle/desc/	grid	09/09/2017

Table B.1: Datasset used to compare Wres to Import.io and Diffbot

Bibliography

[AH06] Siddu P. Algur and P. S. Hiremath. Extraction of flat and nested data records from web pages. In *Data Mining and Analytics 2006, Proceedings of the Fifth Australasian Data Mining Conference (AusDM2006), Sydney, NSW, Australia, 29-30 November, 2006, Proceedings*, pages 163–168, 2006.

[Bis07] Christopher M. Bishop. *Pattern recognition and machine learning, 5th Edition*. Information science and statistics. Springer, 2007.

[Bre01] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[CD99] James Clark and Steve DeRose. XML Path Language (XPath) version 1.0 W3C recommendation. Technical report, World Wide Web Consortium, November 1999.

[CKGS06] Chia-Hui Chang, Mohammed Kayed, Moheb R. Girgis, and Khaled F. Shaalan. A survey of web information extraction systems. *IEEE Trans. Knowl. Data Eng.*, 18(10):1411–1428, 2006.

[CL01] Chia-Hui Chang and Shao-Chen Lui. IEPAD: information extraction based on pattern discovery. In *Proceedings of the Tenth International World Wide Web Conference, WWW 10, Hong Kong, China, May 1-5, 2001*, pages 681–688, 2001.

[CL11] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM TIST*, 2(3):27:1–27:27, 2011.

[CM99] Mary Elaine Califf and Raymond J. Mooney. Relational learning of pattern-match rules for information extraction. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence, July 18-22, 1999, Orlando, Florida, USA.*, pages 328–334, 1999.

[CM03] Mary Elaine Califf and Raymond J. Mooney. Bottom-up relational learning of pattern matching rules for information extraction. *Journal of Machine Learning Research*, 4:177–210, 2003.

[CMM01] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy*, pages 109–118, 2001.

[Coh95] William W. Cohen. Fast effective rule induction. In *Machine Learning, Proceedings of the Twelfth International Conference on Machine Learning, Tahoe City, California, USA, July 9-12, 1995*, pages 115–123, 1995.

[CSRK10] Antonio Criminisi, Jamie Shotton, Duncan P. Robertson, and Ender Konukoglu. Regression forests for efficient anatomy detection and localization in CT studies. In *Medical Computer Vision. Recognition Techniques and Applications in Medical Imaging - International MICCAI Workshop, MCV 2010, Beijing, China, September 20, 2010, Revised Selected Papers*, pages 106–117. 2010.

[CSS11] CSS Working Group. Lengths in the Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) specification. `https://www.w3.org/TR/CSS2/syndata.html#value-def-length`, 2011. Last accessed on 2016-02-18.

[CSS15a] CSS Working Group. Cascading Style Sheets. `http://www.w3.org/Style/CSS/`, 2015. Last accessed 2016-02-19.

[CSS15b] CSS Working Group. CSS snapshot 2015. `https://www.w3.org/TR/css-2015/`, 2015. Last accessed 2016-02-18.

[CV95] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[CYWM03] Deng Cai, Shipeng Yu, Ji-Rong Wen, and Wei-Ying Ma. Extracting content structure for web pages based on visual representation. In *Web Technologies and Applications, 5th Asian-Pacific Web Conference, APWeb 2003, Xian, China, April 23-25, 2002, Proceedings*, pages 406–417, 2003.

[DDG+11] Erik Dahlström, Patrick Dengler, Anthony Grasso, Chris Lilley, Cameron McCormack, Doug Schepers, Jonathan Watt, Jon Ferraiolo, Jun Fujisawa, and Dean Jackson. Scalable Vector Graphics (SVG) 1.1 (second edition). `http://www.w3.org/TR/SVG11/`, August 2011. Last accessed 2016-02-19.

[Del13] AJ Dellinger. Twitter third party clients continue to shut down and its API is just getting more restrictive. `http://www.digitaltrends.com/mobile/firstworldproblems-twitter-api-and-third-party-problem/`, March 2013. Last accessed on 2016-02-19.

[Dif17] Diffbot. Diffbot help and documentation. `https://www.diffbot.com/dev/docs/`, 2017. Last accessed on 2017-09-11.

[DMdF13] Misha Denil, David Matheson, and Nando de Freitas. Narrowing the gap: Random forests in theory and in practice. *CoRR*, abs/1310.1415, 2013.

[Fay12] Ruslan Rustamovich Fayzrakhmanov. Wpps: A novel and comprehensive framework for web page understanding and information extraction. pages 19–26, 2012.

[Fay13] Ruslan Rustamovich Fayzrakhmanov. Web Accessibility for the Blind Through Visual Representation Analysis, 2013.

[FGG+13] Tim Furche, Georg Gottlob, Giovanni Grasso, Christian Schallhart, and Andrew Jon Sellers. Oxpath: A language for scalable data extraction, automation, and crawling on the deep web. *VLDB J.*, 22(1):47–72, 2013.

[FHK13] Ruslan R. Fayzrakhmanov, Christoph Herzog, and Iraklis Kordomatis. Web objects identification for web automation: objects and their features. Technical report DBAI-TR-2013-80, Institute of Information Systems, TU Vienna, Vienna, 2013.

[Fla11] David Flanagan. *JavaScript - The Definitive Guide: Activate Your Web Pages: Covers ECMAScript 5 and HTML5 (6. ed.)*. O'Reilly, 2011.

[FMFB14] Emilio Ferrara, Pasquale De Meo, Giacomo Fiumara, and Robert Baumgartner. Web data extraction, applications and techniques: A survey. *Knowl.-Based Syst.*, 70:301–323, 2014.

[FW98] Eibe Frank and Ian H. Witten. Generating accurate rule sets without global optimization. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML 1998), Madison, Wisconsin, USA, July 24-27, 1998*, pages 144–151, 1998.

[Gar05] Jesse James Garrett. Ajax: A new approach to web applications. `http://adaptivepath.org/ideas/ajax-new-approach-web-applications/`, February 2005.

[Gla14] Dimitri Glazkov. Custom elements. `http://www.w3.org/TR/custom-elements/`, December 2014. Last accessed on 2016-02-19.

[Got05] Georg Gottlob. Web data extraction for business intelligence: The lixto approach. In *Datenbanksysteme in Business, Technologie und Web, 11. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme" (DBIS), Karlsruhe, 2.-4. März 2005*, pages 30–47, 2005.

[Has09] Tamir Hassan. Object-level document analysis of PDF files. In *Proceedings of the 2009 ACM Symposium on Document Engineering, Munich, Germany, September 16-18, 2009*, pages 47–55, 2009.

[HBF+14] Ian Hickson, Robin Berjon, Steve Faulkner, Travis Leithead, Erika Doyle Navara, Edward O'Connor, and Silvia Pfeiffer. HTML5 a vocabulary and associated APIs for HTML and XHTML. http://www.w3.org/TR/html5/, October 2014. Last accessed on 2016-02-19.

[HD98] Chun-Nan Hsu and Ming-Tzung Dung. Generating finite-state transducers for semi-structured data extraction from the web. *Inf. Syst.*, 23(8):521–538, 1998.

[Hev07] Alan R Hevner. A three cycle view of design science research. *Scandinavian journal of information systems*, 19(2):4, 2007.

[HK05] Andrew W. Hogue and David R. Karger. Thresher: automating the unwrapping of semantic content from the world wide web. In *Proceedings of the 14th international conference on World Wide Web, WWW 2005, Chiba, Japan, May 10-14, 2005*, pages 86–95, 2005.

[Joh15] Michael Johnston. 60fps on the mobile web. http://engineering.flipboard.com/2015/02/mobile-web/, 2015. Last accessed on 2016-02-19.

[JWHT13] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[KA09] Ritu Khare and Yuan An. An empirical study on using hidden markov model for search interface segmentation. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management, CIKM 2009, Hong Kong, China, November 2-6, 2009*, pages 17–26, 2009.

[KHF+13] Iraklis Kordomatis, Christoph Herzog, Ruslan Rustamovich Fayzrakhmanov, Bernhard Krüpl-Sypien, Wolfgang Holzinger, and Robert Baumgartner. Web object identification for web automation and meta-search. In *3rd International Conference on Web Intelligence, Mining and Semantics, WIMS '13, Madrid, Spain, June 12-14, 2013*, page 13, 2013.

[Kus00] Nicholas Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artif. Intell.*, 118(1-2):15–68, 2000.

[Liu07] Bing Liu. *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data*. Data-Centric Systems and Applications. Springer, 2007.

[Mit97] Tom M. Mitchell. *Machine learning, International Edition*. McGraw-Hill Series in Computer Science. McGraw-Hill, 1997.

[MMK99] Ion Muslea, Steven Minton, and Craig A. Knoblock. A hierarchical approach to wrapper induction. In *Agents*, pages 190–197, 1999.

[Moz15] Mozilla Developer Network. Node type constants. https://developer.mozilla.org/en-US/docs/Web/API/Node/nodeType#Node_type_constants, August 2015. Last accessed on 2016-04-10.

[MP13] Michael Mikowski and Josh Powell. Single Page Web Applications: Javascript end-to-end. 2013.

[Net15] Netcraft. August 2015 web server survey. http://news.netcraft.com/archives/2015/08/13/august-2015-web-server-survey.html, August 2015. Last accessed 2016-02-19.

[Qui93] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[RCC92] David A. Randell, Zhan Cui, and Anthony G. Cohn. A spatial logic based on regions and connection. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92). Cambridge, MA, October 25-29, 1992.*, pages 165–176, 1992.

[Sod99] Stephen Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1-3):233–272, 1999.

[TL14] Dan Thorp-Lancaster. Twitter's new search API closed to third-party apps. http://www.imore.com/twitter-reportedly-closes-new-search-api-third-party-apps, November 2014. Last accessed on 2016-02-19.

[Tso12] Alexia Tsotsis. No API for you: Twitter shuts off Find Friends feature for Instagram. http://techcrunch.com/2012/07/26/no-api-for-you-twitter-shuts-off-find-friends-feature-for-instagram/, July 2012. Last accessed on 2016-02-19.

[vKGM+15] Anne van Kesteren, Aryeh Gregor, Ms2ger, Alex Russell, and Robin Berjon. W3C DOM4. https://www.w3.org/TR/dom, 11 2015. Last accessed on 2017-09-24.

[Wek] Weka Wiki. ARFF. http://weka.wikispaces.com/ARFF+%28book+version%29. Last accessed 2016-02-19.

[Wik15] Wikipedia. Size of wikipedia. https://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia, September 2015. Last accessed on 2016-02-19.

[ZL05] Yanhong Zhai and Bing Liu. Web data extraction based on partial tree alignment. In *Proceedings of the 14th international conference on World Wide Web, WWW 2005, Chiba, Japan, May 10-14, 2005*, pages 76–85, 2005.

[ZMW+05] Hongkun Zhao, Weiyi Meng, Zonghuan Wu, Vijay V. Raghavan, and Clement T. Yu. Fully automatic wrapper generation for search engines. In *Proceedings of the 14th international conference on World Wide Web, WWW 2005, Chiba, Japan, May 10-14, 2005*, pages 66–75, 2005.

[ZZWL13] Zhixian Zhang, Kenny Q. Zhu, Haixun Wang, and Hongsong Li. Automatic extraction of top-k lists from the web. In *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, pages 1057–1068, 2013.