

**FACULTY  
OF MATHEMATICS  
AND PHYSICS**  
Charles University

**MASTER THESIS**

Bc. Rastislav Kadleček

# **Converting HTML product data to Linked Data**

Department of Software Engineering

Supervisor of the master thesis: doc. Mgr. Martin Nečaský, Ph.D.

Study programme: Software Systems

Study branch: Software Engineering

Prague 2018

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date .....

signature of the author

I would like to thank my supervisor, doc. Mgr. Martin Nečaský, Ph.D., and my advisor, RNDr. Tomáš Knap, Ph.D., for their helpful and valuable advices and remarks that helped me with writing of this thesis.

I would also like to thank my family and my girlfriend for all their support during writing of this thesis.

Title: Converting HTML product data to Linked Data

Author: Bc. Rastislav Kadleček

Department: Department of Software Engineering

Supervisor: doc. Mgr. Martin Nečaský, Ph.D., Department of Software Engineering

Abstract: In order to make a step towards the idea of the Semantic Web it is necessary to research ways how to retrieve semantic information from documents published on the current Web 2.0. As an answer to growing amount of data published in a form of relational tables, the Odalic system, based on the extended TableMiner<sup>+</sup> Semantic Table Interpretation algorithm was introduced to provide a convenient way to semantize tabular data using knowledge base disambiguation process. The goal of this thesis is to propose an extended algorithm for the Odalic system, which would allow the system to gather semantic information for tabular data describing products from e-shops, which have very limited presence in the knowledge bases. This should be achieved by using a machine learning technique called classification. This thesis consists of several parts - obtaining and preprocessing of the product data from e-shops, evaluation of several classification algorithms in order to select the best-performing one, description of design and implementation of the extended Odalic algorithm, description of its integration into the Odalic system, evaluation of the improved algorithm using the obtained product data and semantization of the product data using the new Odalic algorithm. In the end, the results are concluded and possible directions for the future works are suggested.

Keywords: linked data, knowledge bases, data quality

# Contents

<b>Introduction</b>	<b>4</b>
<b>1 Web 2.0 vs the Semantic Web</b>	<b>6</b>
1.1 Web 2.0 . . . . .	6
1.1.1 Web 2.0 - Technologies . . . . .	6
1.1.2 Problems of Web 2.0 . . . . .	9
1.2 Web 3.0: The Semantic Web and Linked Data . . . . .	10
1.2.1 Linked Data . . . . .	10
1.2.2 Web 3.0 Technologies . . . . .	12
1.3 Web 2.0 Semantization - Annotations . . . . .	17
1.3.1 Microformats . . . . .	17
1.3.2 RDF in Attributes (RDFa) . . . . .	18
1.3.3 The OpenGraph Protocol . . . . .	19
1.3.4 JSON-LD . . . . .	19
<b>2 Semantic Table Interpretation &amp; Related Work</b>	<b>21</b>
2.1 Semantic Table Interpretation . . . . .	21
2.2 Existing Approaches . . . . .	22
<b>3 TableMiner<sup>+</sup> and Odalic</b>	<b>25</b>
3.1 TableMiner <sup>+</sup> Algorithm . . . . .	25
3.1.1 Algorithm Flow . . . . .	26
3.2 Odalic Project and Odalic Core . . . . .	28
3.2.1 Odalic Project Components . . . . .	29
<b>4 Background of This Thesis</b>	<b>31</b>
4.1 Motivation . . . . .	31
4.2 Goal . . . . .	32
<b>5 Obtaining Product Data</b>	<b>34</b>
5.1 Downloading HTML Product Details . . . . .	34
5.1.1 Scraped HTML Documents . . . . .	35
5.2 HTML to CSV Conversion . . . . .	35
5.3 Converted CSV files . . . . .	36
<b>6 Evaluating Machine Learning Classifier Algorithms</b>	<b>40</b>
6.1 Machine Learning & Classification . . . . .	40
6.2 Evaluated Classification Algorithms . . . . .	41
6.3 Classifier Evaluation . . . . .	44
6.3.1 The ML-Experiments Framework . . . . .	46
6.3.2 Experiments . . . . .	48
6.3.3 Experiments Conclusion . . . . .	52

<b>7</b>	<b>The New Odalic Algorithm</b>	<b>54</b>
7.1	Algorithm Description . . . . .	54
7.1.1	The ML PreClassification Phase . . . . .	54
7.1.2	Subject Column Detection . . . . .	62
7.1.3	Column Classification & Entity Disambiguation . . . . .	63
7.1.4	Relation Discovery . . . . .	64
7.1.5	Lookup of Predicate Between Subject and Object Entities	66
7.1.6	Adjustment of the thesis sub-goals . . . . .	67
7.2	Odalic Integration . . . . .	67
7.3	Evaluation . . . . .	68
7.3.1	Original Odalic Algorithm Results . . . . .	70
7.3.2	New Odalic Algorithm Results . . . . .	71
7.4	Semantization of Obtained Product Data . . . . .	73
	<b>Conclusion &amp; Future Work</b>	<b>76</b>
7.5	Future Work . . . . .	77
	<b>Bibliography</b>	<b>78</b>
	<b>List of Figures</b>	<b>82</b>
	<b>List of Tables</b>	<b>83</b>
	<b>List of Abbreviations</b>	<b>85</b>
<b>A</b>	<b>Attachment 1: CD Contents</b>	<b>86</b>
<b>B</b>	<b>Attachment 2: HTML to CSV Conversion Application</b>	<b>88</b>
B.1	Getting the application . . . . .	88
B.2	Compiling application JAR package . . . . .	88
B.3	Launching the application . . . . .	88
B.4	Application Configuration . . . . .	88
B.5	Job Configuration . . . . .	89
B.6	Adding Support For a New Website Type . . . . .	89
B.6.1	Parsers Package . . . . .	90
B.6.2	WebsiteType Enumeration . . . . .	90
<b>C</b>	<b>Attachment 3: Feature Experiments</b>	<b>91</b>
C.1	Experiment 1 . . . . .	91
C.2	Experiment 2 . . . . .	91
C.3	Experiment 3 . . . . .	92
C.4	Experiment 4 . . . . .	92
C.5	Experiment 5 . . . . .	93
C.6	Experiment 6 . . . . .	93
C.7	Experiment 7 . . . . .	93
C.8	Experiment 8 . . . . .	94
C.9	Experiment 9 . . . . .	94
C.10	Experiment 10 . . . . .	95
C.11	Experiment 11 . . . . .	95
C.12	Experiment 12 . . . . .	96

C.13 Experiment 13 . . . . .	96
C.14 Experiment 14 . . . . .	97
C.15 Experiment 15 . . . . .	98
<b>D Attachment 4: Algorithm Configurations</b>	<b>101</b>
D.1 Decision Tree . . . . .	101
D.2 Decision Stump . . . . .	102
D.3 Decision List . . . . .	102
D.4 Decision Table . . . . .	102
D.5 Naive Bayes . . . . .	104
D.6 Random Forest . . . . .	104
D.7 Support Vector Machine . . . . .	105
D.8 Multilayer Perceptron . . . . .	106
D.9 Best Algorithm Configuration Comparison . . . . .	106
<b>E Attachment 5: Proposed Algorithm Evaluation</b>	<b>108</b>
E.1 Task 1 . . . . .	108
E.2 Task 2 . . . . .	109
E.3 Task 3 . . . . .	109
E.4 Task 4 . . . . .	110
E.5 Task 5 . . . . .	110
E.6 Task 6 . . . . .	111

# Introduction

According to the statistics published by Live Internet Stats [2018], there has been a total of *1.7 billion* websites (unique host names) available on the Internet in the end of the year 2017. This represents a growth of almost *700 million* of websites compared to the year 2016. Last available statistics published by StatisticBrain [2015] state that, in the end of the year 2014, the Google<sup>1</sup> search engine indexed *8 trillions* of unique web pages. Even though many of those web pages contain invalid, outdated, uninteresting or duplicate information, the amount of web pages containing useful or interesting data is nevertheless still huge. It would be certainly useful for many reasons if this huge amount of published data could be easily automatically processed by machines and their algorithms. However, most of these web pages are published in a “human-friendly” fashion, which makes machine processing of data they contain difficult, or even impossible. To make data published on web pages more “machine-friendly”, it would be necessary to provide some context, also called semantic information, about the data in a “machine-understandable” format. Even though there exist some techniques that allow publishers of existing websites to add semantic information about published data easily by adding additional markup<sup>2</sup>, publishers of websites present on the current Web are adapting those techniques very slowly. Because of this slow adaptation, some third-party efforts emerged, which aim to *semantize* — add semantic information to — the data published on the Web using methods which are independent on the website publishers. However, since the amount of data published on the Web is so huge, it is impossible to add this semantic information to the data manually. Therefore it is necessary to find a way, how to make the semantization process more automatic.

Many of web pages also publish data in form of *tables* - either directly by publishing (e.g.) CSV<sup>3</sup> files with data, or by embedding a table into a web page itself.

A field of research called *Semantic Table Interpretation* focuses on development of methods, algorithms or whole systems for (semi-) automatic semantization of tabular data. An example of such algorithm is the *TableMiner*<sup>+</sup>, developed by *Ziqi Zhang*. This proof-of-concept algorithm was later extended into a complete table semantization system by students of *The Faculty of Mathematics and Physics at Charles University in Prague* in their software project called *Odalic*<sup>4</sup>.

The Odalic system, however, is not very effective in semantization of tables which contain data about products from e-shops. This is caused by the fact that the *TableMiner*<sup>+</sup> algorithm, used by Odalic, performs classification of data just by disambiguation of data entities using existing, already semantized, datasets (*Linked Data*) present in databases, also called *knowledge bases*. Since there are thousands and thousands of different products in existence, with new ones being introduced every day, datasets stored in the knowledge bases are not comprehensive enough and contain just a fraction of information related to product data.

---

<sup>1</sup><http://www.google.com>

<sup>2</sup>Described in chapter 1, section 1.3.

<sup>3</sup>CSV: Comma-Separated Values.

<sup>4</sup>More information about the Odalic system can be found in Chapter 3 of this thesis or in [Knap, 2017].



The goal of this thesis is to propose an improved version of a Semantic Table Interpretation algorithm for the Odalic system which would allow the algorithm to infer semantic information — such as *classifications of columns* of table to *entities*, or discovering *relations between table columns* — even on the product data. For this purpose, machine learning algorithms, namely algorithms related to a machine learning technique called classification should be used. Several classification algorithms should be evaluated in order to determine which one has the best results on the gathered product data. As a part of this thesis, the improved algorithm should also be integrated into the Odalic system and evaluated in comparison with the original Odalic algorithm. Finally, the algorithm should be used to help with semantization of obtained HTML product data. The goal of this thesis is in more detail described in the Chapter 4.

The thesis is organized as follows:

**Chapter 1** summarizes basic differences between the Web 2.0 and the Semantic Web, provides a brief summary of technologies used by these Web concepts and introduces the concept of Linked Data.

**Chapter 2** describes the motivation behind semantization of data represented in tables and challenges of this process. Semantic Table Interpretation research field is introduced and brief summary of already existing works on this topic is given.

**Chapter 3** introduces the *TableMiner<sup>+</sup>* algorithm and the *Odalic* system, built on top of it, in detail.

**Chapter 4** takes advantage of information provided in Chapters 1 – 3 in order to describe problems with semantization of data about products published by e-shops, using the Odalic Semantic Table Interpretation system and based on that, describes the goals of this thesis in detail.

**Chapter 5** describes process of obtaining HTML documents containing data about products from several chosen e-shops and some of the necessary pre-processing of this data in order to allow the data to be used in classifier evaluation process and also to allow the Odalic system to process the data.

**Chapter 6** provides a brief introduction into machine learning, especially into a set of machine learning algorithms called classifiers. As one of these classifier algorithms will be used to improve efficiency of the Odalic algorithm on product data, this chapter also provides experiments (and their evaluation) with several chosen classification algorithms. These experiments were performed in order to find out which one performs the best on the gathered input data.

**Chapter 7** explains in detail the proposed improved Odalic algorithm designed by the author of this thesis and also describes the process of integration of the algorithm into the Odalic system. This chapter also contains an evaluation of how the proposed algorithm improved the semantization of product data in comparison with the original Odalic algorithm. In the end, a process of semantization of the gathered product data using the new Odalic algorithm is described.

# 1. Web 2.0 vs the Semantic Web

The aim of this chapter is to introduce two different approaches towards the *Web* implementation – the older *Web 2.0* and the newer *Web 3.0 (the Semantic Web)*. This chapter does not only explain key concepts of both of these approaches, but also explains key differences between them. It also introduces terms “the Semantic Web”, “Linked Data” and explains their practical benefits. This chapter also aims to provide the reader with necessary theoretical information about Web technologies which are required to fully understand this thesis. The author tries to provide the information in a comprehensive way, however some additional knowledge about the Information Technologies field may be required from the reader. These additional information can be found in books or online resources.

## 1.1 Web 2.0

*Web 2.0* — also nicknamed as *people-centric Web* — is the version of the Web mostly used at present [Aghaei et al., 2012]. It was designed with needs of a human user in mind. Websites focus on the clarity of the *user interface* (UI) and want to provide the best possible *user experience*<sup>1</sup> (UX). Layouts of web pages and information structure is formatted in a way that allows simple orientation and information retrieval for human users.

Compared to the — so called — *Web 1.0*, which allowed users only to consume the content of static websites, the Web 2.0 offers also the possibility of human interaction [Aghaei et al., 2012]. Users are often allowed, or even encouraged, to add new content. For example, users can discuss with each other by adding new posts in discussion boards, write their own blog posts or product reviews, or share multi-media content on social networks. However, user interaction does not necessarily mean adding new content. User is also capable to use various controls<sup>2</sup> to send additional requests to a website controller, based on which a website can perform additional custom action, such as *returning more specific content* or *booking a plane ticket*. This feature allowed complex websites, such as e-shops, to be developed.

### 1.1.1 Web 2.0 - Technologies

As the Web evolved from the “read-only” to the “interactive” mode, it became apparent that websites can no longer consist of just a group of single static documents which are linked together. A typical Web 2.0 web page consists of two parts: so called *front-end* part which provides an user interface for the user and the server part which is responsible for returning relevant responses for the users requests, and is also called *back-end*.

The *back-end* side of the web pages are mostly created using programming

---

<sup>1</sup><https://www.usability.gov/what-and-why/user-experience.html>

<sup>2</sup>UI controls, such as hyperlinks, forms, buttons, sliders, etc.

languages such as *PHP*<sup>3</sup>, *Asp.net*<sup>4</sup> or *Java*<sup>5</sup>, however this master thesis does not cover them in more detail, as the reader does not necessarily need to understand these in order to understand this thesis. Additional information about these topics can be found in relevant books or Internet resources.

On the other hand, selected *front-end* technologies will be briefly introduced in following sections.

## (X)HTML

(X)HTML or (*Extensible*) *HyperText Markup Language* is — as its name suggests — a family of markup languages used for creating user interface components of websites. The fact that it is a *markup* — and not a *programming* — language means that it does not specify a behaviour of a website, nor the way how a website reacts to the user interaction<sup>6</sup>. Instead, it describes a basic *structure*, or *layout* of the *content* of a website. Following text will briefly introduce the most recent<sup>7</sup> version of HTML, the HTML 5.2 which was released as *W3C*<sup>8</sup> *recommendation* in December 2017 [W3C, 2017].

The structure of web pages in the HTML language is written using *HTML elements*, also called *tags*. Each element gives the web browser an information how to render the specific part of the web page. The element is written using *angle brackets* “<” and “>”, with element name stated between them: <**ElementName**>. Most of the elements do contain *closing* tag which has the element name prefixed with “/” character: </**ElementName**>. The content between opening and closing tag is called *element body*.

(X)HTML elements can contain *attributes*. Attributes can be optional for some elements and required for other elements. They are stated inside the element definition tag, after the element name between the angle brackets, and they can narrow down the meaning of the element.

The general structure of a HTML 5 document, as described by [W3C, 2017], can be seen in the Code 1.1.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Sample page</title>
  </head>
  <body>
    <h1>Sample page</h1>
    <p>This is a <a href="demo.html">simple</a> sample.</p>
    <!-- this is a comment -->
```

---

<sup>3</sup><http://php.net>

<sup>4</sup><http://www.asp.net>

<sup>5</sup><http://www.oracle.com/technetwork/java/javase/overview/index.html>

<sup>6</sup>The additional functionalities of the website, such as modal dialogs or event handling, can be implemented using additional *scripting language*, such as JavaScript.

<sup>7</sup>At the time of writing of this thesis.

<sup>8</sup>Consortium which looks after the development of the World Wide Web standards. The standard which is approved as “W3C recommendation” is considered as finalized and ready for implementation by the general public [W3C, 2005]

```
</body>
</html>
```

Code 1.1: General structure of a HTML 5 document. Source: [W3C, 2017].

The complete specification of the HTML 5.2 markup language and the comprehensive list of all elements, together with their description, can be found in W3C [2017].

In order to make parsing of HTML markup easier, a separate markup language — based on older HTML 4 — has been released as W3C recommendation in January 2000. This language is called *XHTML 1.0* and modifies HTML in a way so that its *XML<sup>9</sup> compliant* W3C [2002]. The development of XHTML language was, however, discontinued in 2009 in favor of HTML 5 W3C [2009].

## CSS And CSS Selectors

Previous sub-section introduced the family of (X)HTML languages which are used for markup of the structure of web pages. This (X)HTML structure is however very rough, formatted by styles default for each web browser and does not contain any graphics more complex than images. Typically, each element of a page is rendered on separate line. This can be sufficient for very simple pages, especially ones that contain only well structured text, however, more complex pages require more complex means to provide custom graphic styles.

This requirement led to the creation of CSS. CSS, *Cascading Style Sheets*, is defined by its specification W3C [2011] as “style sheet language that allows authors and users to attach style (e.g. fonts and spacing) to structured documents”. The possibilities of CSS are much more complex - it allows the creator of the document (web page) to redefine the styles of each element. That means, for example, changing the layout and background colors of table rows, styling the hypertext links<sup>10</sup> to look like buttons, or even changing the position of the elements within the document.

The most recent version of CSS is the *CSS 2*<sup>11</sup>. There already exists a newer version, the *CSS 3*, but it does not have an official W3C recommendation status yet W3C [2014a].

*CSS selectors* are identifiers which are used to determine which style should be used for which (X)HTML element.

Styles of each element can be defined by more than one, possibly even self-contradicting, rules. In order to determine which styles should be applied, a priority is assigned to each rule, based on the level of specificity of the selector. That means, the selector “**table tr td**” will have higher priority than selector “**tr td**” (hence the name *cascading* styles). Styles with highest priority are used by the web browser to render components of a web page.

Selected examples of <sup>12</sup> CSS selectors are described in the Table 1.1.

---

<sup>9</sup>Extensible Markup Language: <https://www.w3.org/XML/>

<sup>10</sup>Together with reaction to the change of link state (**link**, **visited**, **hover**, **active**)

<sup>11</sup>At the time of writing of this thesis

<sup>12</sup>Complete list of selectors can be found at <https://www.w3.org/TR/CSS2/selector.html>

An example of simple CSS selectors, together with style definitions, can be found in Code 1.2. CSS selectors are located in front of property blocks which are encapsulated in brackets.

```
body {
    font-family: Arial;
    font-size: 16px;
}
body div {
    background-color:red;
}
```

Code 1.2: Sample CSS rules.

### 1.1.2 Problems of Web 2.0

Introduction of the current Web in the section 1.1 may suggest that web pages at present time are very user friendly and easy to use. This is in fact true for many of them. However, is it easy to use enough? Following definition of Task 1 describes example of the common tasks performed by a visitor of a travel agency website.

**Definition 1** (Task 1). *The visitor of a specific travel agency website would like to book a summer holiday in the Dubrovnik region, Croatia. The holiday should take place between August 8, 2018 and August 21, 2018.*

The majority of todays Internet users would, without any doubts, manage to accomplish this task without bigger problems, using for example following steps:

1. Set the vacation search filter on the website of travel agency to search only *Croatia, Dubrovnik area* and given date range.
2. Read through the list of all available hotels and select the most suitable one, together with desired type of diet and way of transport.
3. Submit the reservation form.

Selector Pattern	Description
*	Matches any element.
E	Matches any E element (i.e., an element of type E).
E F	Matches any F element that is a descendant of an E element.
E > F	Matches any F element that is a child of an element E.
E[foo=warning]	Matches any E element whose "foo" attribute value is exactly equal to "warning".
E:hover	Matches E when the user hovers the mouse over this element.

Table 1.1: Selected CSS selectors. Source: <https://www.w3.org/TR/CSS2/selector.html>

Definition of Task 2 describes similar task, but a little bit more complex.

**Definition 2** (Task 2). *The Internet user would like to book a summer holiday, but in contrast with the user in the Task 1, he does not have the exact idea of the location where he would like to go, neither does he have the idea of specific date range. He however does have some criteria: The vacation should take place during the August 2018, be between 11 and 14 days long and he wants to book at least 3-star hotel, not more than 500 meters far from the sea in some South-European country. The vacation should also cost at most 800 EUR per person.*

Finding a vacation of dreams of the user from Task 2 would be a very time consuming task - it would require him to manually go through catalogs of many travel agencies and hotel offerings, with very limited help of the search filters on their web pages.

It would be logical to try to make this process automated. After all, if all data about all vacations and hotels were stored in single database, finding the optimal vacation would be as simple as running single SELECT SQL query<sup>13</sup>!

The problem is that in reality data is stored by various travel agencies on various servers using various formats and they can be accessed from outside world only using web presentations of these travel agencies. These web presentations of course do vary - each presentation has its own layout, shows the data in different formats and styles and even shows different subset of available information.

This variety of formats and data decentralization does make the automation of tasks similar to the task described in Task 2 very problematic, if not totally impossible.

## 1.2 Web 3.0: The Semantic Web and Linked Data

*The Semantic Web* — also known as *the Web of Linked Data* or *Web 3.0* [CambridgeSemantics] — is the response of the W3C consortium to address issues of similar type as the issue defined in Task 2 in Section 1.1.2. According to the W3C [2015b] standard, the main goal of Semantic Web is “to do more useful work and to develop systems that can support trusted interactions over the network”.

As the name *Web of Linked Data* suggests, the main building stone of Semantic Web is *data*, specifically *Linked Data*.

### 1.2.1 Linked Data

As already stated in previous sub-section, the basis of the Semantic Web are *data*. These data are structured representation of *objects* (more commonly called *resources*) within some domain and their properties. A *resource* can be almost anything - a *real world object* (such as *car*), an *abstract object* (such as *electronic document*), an abstract concept (such as *number*) and others. As more complex example, a resource can be a *description of a chemical element and its properties*, such as its *atomic number*, *half-life*, *boiling point*, and so on.

---

<sup>13</sup>SELECT query in the PostgreSQL database: <http://www.postgresql.org/docs/9.1/static/sql-select.html>

*Relations* between data, e.g. that the resource **Rudyard Kipling** *is an author* of the resource **The Jungle Book**, can be described as well.

Data that share some common properties or relations can be linked together to form a *collection*. This collection is called a *Dataset*<sup>14</sup>. Datasets that belong to different *data sources* can be further connected together to create larger and larger datasets. In order to allow data from different sources to connect, a common publication method needed to be developed. This publication method is called *Linked Data* and is defined by Bizer et al. [2009] as a “set of best practices for publishing and connecting structured data on the Web”. These best practices, often called *Linked Data principles*, guarantee that data published in compliance with these practices can be linked together with another data based on their *properties* or *relations* between them.

### Linked Data Principles

Author of the idea of the Semantic Web, *Tim Berners-Lee*, defined four basic principles that have to be respected if the data should be considered as Linked Data [Berners-Lee, 2006]. Those principles are:

1. “*Use URIs as names for things.*” Every resource in a dataset should have a unique name. This name should be unique not only within the given dataset, but it should be unique globally, so it is easy to identify resources in the case of connection with other datasets. To avoid possible issues caused by different naming conventions in different datasets, this principle suggests the common naming convention - usage of URI (later replaced by IRI)<sup>15</sup> identifiers. To achieve uniqueness of its identifiers, it is recommended for the publication authority to use a domain name it holds in URIs of its resources. This should guarantee that nobody else will use the same identifier.
2. “*Use HTTP*<sup>16</sup> *URIs so that people can look up those names.*” Previous principle suggests that an URI (IRI) identifier should be used for resource identification. In general, URI (IRI) does support different schemas (e.g. **ftp**, **file**, **mailto**, ...). This principle specifies that URI (IRI) resource identifiers must use the **http** (or **https**)<sup>17</sup> schema which allows to retrieve more information about the resource using HTTP<sup>18</sup> request to its URI (IRI). If an HTTP request is sent by human user, he should be redirected to a document describing the resource in “human-readable” way. If an HTTP request is sent by a machine, it should be redirected to “machine-readable” representation of the information about that resource.
3. “*When someone looks up an URI, provide useful information using the standards (RDF, SPARQL).*” This principle narrows down the previous principle in a way that user (human or machine), which issues HTTP request to the URI (IRI) of a resource in order to gather more information about it,

---

<sup>14</sup>One of the most known and largest datasets is DBpedia, <http://wiki.dbpedia.org/>

<sup>15</sup>See section 1.2.2.

<sup>16</sup>Hypertext Transfer Protocol

<sup>17</sup>Hypertext Transfer Protocol Secure

<sup>18</sup>Hypertext Transfer Protocol, <https://www.w3.org/Protocols/>

should get the most relevant information possible which should be represented using one of the well known standard formats (such as RDF), so the requestor can easily understand and process them.

4. “*Include links to other URIs, so that they can discover more things.*” The last principle requires the publisher to provide external links pointing to URIs (IRIs) of as much different resources as possible in the dataset. This ensures that the data from different datasets are properly connected, become Linked Data and form a Semantic Web.

## 1.2.2 Web 3.0 Technologies

This subsection will introduce the most important technologies used in the field of Linked Data and Semantic Web.

### URI and IRI

Berners-Lee et al. [2005] defines *Uniform Resource Identifier* as an identifier which provides simple and extensible way of identifying resources. Every resource that should be accessible for connections (e.g. web server endpoint) or every resource that can be described needs to be identified by its own URI.

The scope of URIs is global - in order to exactly distinguish between all of possible resources, the identifier of an objects needs to be globally unique. A situation when two different objects share the same URI is called a *collision*. Collisions are undesirable and may cause problems. On the other hand, a situation when one object is identified by more than one URIs is correct. Such URIs are called *URI aliases*.

Fundamentally, URI is a string of characters with fixed structure. There are many different types of URIs. Although each of the types (also called *schemas*), can have different structure, they all share the first part of URI - the *schema identifier* which determines what type of URI it is and what structure follows. As stated in the section 1.2.1 covering Linked Data principles, its required to use HTTP URI for the data published using Linked Data model. This section will therefore focus on this URI type.

**HTTP URI Syntax** As described by Fielding and Reschke [2014], the HTTP URI has strictly defined syntax which consists of multiple components:

$$http : // < authority > < path > [ ? < query > ] [ \# < fragment > ]$$

- *authority* - this URI component contains domain name (e.g. “**www.example.com**”) or IP address (e.g. “**192.168.1.1**”) of the target resource. The number of TCP port which should receive the request may be included as well. If a port number is present, it is separated from the domain name/IP address by colon (“:”) character (e.g. “**www.example.com:81**”). If a port number is not explicitly stated, port 80 is used by default for HTTP URIs. This URI component can also contain login credentials for the *HTTP Basic Authentication*. They are stated in the beginning of the component in form *username:password* and they are



separated by the “@” character from the rest of the component contents (e.g. “ **johndoe:mysecretpassword@www.example.com**”).

- *path* - absolute path to the desired resource within the target host.
- *query* - optional component, this component is prefixed by the question mark (“?”) character and allows to specify additional parameters in the form *param1=value1*. Multiple parameters are separated with the ampersand (“&”) character (*param1=value1&param2=value2*). These parameters are passed to the requested resource (e.g. website controller).
- *fragment* - optional component, prefixed by the hash (“#”) character. This part of URI is stripped from the URI and is not sent to the remote resource. It is useful for identification of relevant part of the document - whole document is returned in the HTTP response, but the clients browser displays only the relevant part according to the given fragment identifier.

For better understanding the Code 1.3 shows several examples of valid HTTP URIs.

```
# Domain name only:  
http://www.example.com  
  
# Domain name and path:  
http://www.example.com/users/johndoe  
  
# Domain name including TCP port, path and query components:  
http://www.example.com:81/users/johndoe/files?showHiddenFiles=false  
  
# Domain name with login credentials, path, query and fragment  
# components:  
http://john:password@www.example.com/photos?limit=5&offset=5#photo-1
```

Code 1.3: Valid HTTP URI examples.

**Internationalized Resource Identifier (IRI)** There is one large drawback of URIs - they can contain only characters from the ASCII charset [Duerst and Suignard, 2005]. Because of this, it has only limited support for representation of names and characters of different national alphabets and character sets. Cyrillic characters, for example, need to be “transliterated” into latin before they can be used in URI. In recent years, however, new technologies that allow the usage of different national character sets on the Web were introduced. One of them is the *Internationalized Resource Identifier (IRI)* which extends URI and adds the support of national character sets.

## Resource Description Framework (RDF)

*Resource Description Framework* is a framework developed in order to allow people to create the machine-understandable description (model) of the real world [W3C, 2014c]. Data is modeled by listing (*RDF*) *statements* (also called (*RDF*) *triples*). Each triple describes a single property of the resource being modeled, or relation between two resources. A triple consists of following three components:

- *Subject* - resource which is being described. The resource can be identified by its URI (IRI), or the resource can be anonymous<sup>19</sup>.
- *Predicate* - a *property of the subject* or *relation between the subject and object* which is described by the triple. Predicate has to be represented by URI (IRI). Blank predicates are not allowed.
- *Object* - an object can be one of following:
  - If a triple describes a *property* of subject resource, the object determines the value of this property. The type of such object is a literal (e.g. a number or a string of characters). The data type of the property value can be optionally specified. If the value is a string of characters, the language of the text can also be specified.
  - If the triple describes a *relation* between 2 resources, the object determines the second resource which shares this relationship with subject. This resource can be either represented by its IRI, or it can possibly be a blank node.

A set of triples is internally represented as directed graph, in which every triple represents one edge that is connecting two vertices. This edge is named by the IRI of predicate, starts in the vertex representing the subject and ends in the vertex that represents the object of the statement.

As can be seen from the description of the components, each RDF triple can be interpreted as a “*sentence*”.

The following Definition 3 provides an example of a “human friendly” description of a cell phone. The RDF graph representing this example can be found in Figure 1.1. RDF triples describing the example will be shown in next section, after selected RDF notations are introduced.

**Definition 3** (Example Cell Phone Description). *Samsung Galaxy S6 edge+ is a smartphone made by the South Korean company Samsung. It has 8-core processor Samsung Exynos 7420 Octa Core, 4 GB of RAM and 5.7-inch display.*

## RDF Triple Notations

There are several different standardized ways of notation of RDF triples. The most common ones are *RDF/XML*<sup>20</sup>, *N-Triples*<sup>21</sup>, *Turtle*<sup>22</sup> will be introduced in this section.

The Code 1.4 shows the representation of information contained in the Definition 3 serialized using the N-Triples notation, where the RDF *subject*, *predicate*, *object* triples are clearly visible:

---

<sup>19</sup>Anonymous resource is a resource that has no URI (IRI) assigned. Its commonly called as **Blank node**.

<sup>20</sup><https://www.w3.org/TR/rdf-syntax-grammar/>

<sup>21</sup><https://www.w3.org/TR/n-triples/>

<sup>22</sup><https://www.w3.org/TR/turtle/>

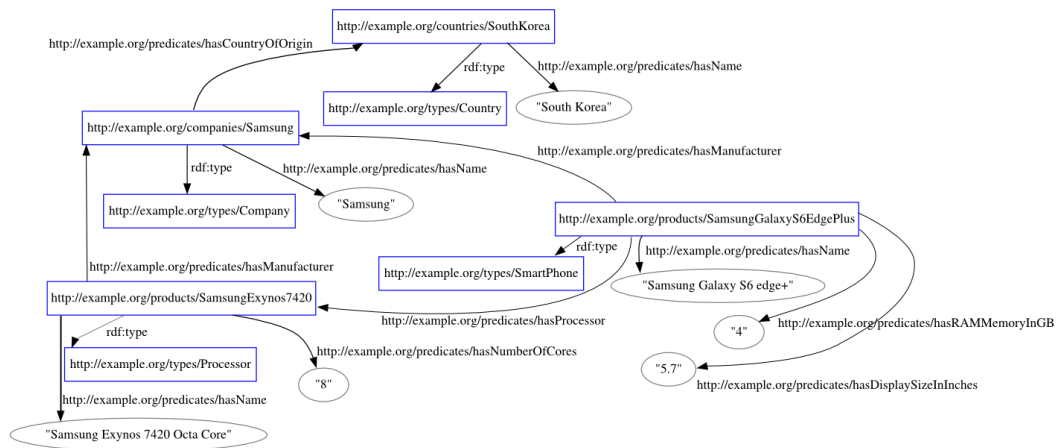


Figure 1.1: RDF graph representation of the information from Example 3.

```

<http://example.org/countries/SouthKorea>
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
    <http://example.org/types/Country> .

<http://example.org/countries/SouthKorea>
  <http://example.org/predicates/hasName>
    "South Korea"@en .

<http://example.org/companies/Samsung>
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
    <http://example.org/types/Company> .

<http://example.org/companies/Samsung>
  <http://example.org/predicates/hasName>
    "Samsung"@en .

<http://example.org/companies/Samsung>
  <http://example.org/predicates/hasCountryOfOrigin>
    <http://example.org/countries/SouthKorea> .

<http://example.org/products/SamsungGalaxyS6EdgePlus>
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
    <http://example.org/types/SmartPhone> .

<http://example.org/products/SamsungGalaxyS6EdgePlus>
  <http://example.org/predicates/hasName>
    "Samsung Galaxy S6 edge+"@en .

<http://example.org/products/SamsungGalaxyS6EdgePlus>
  <http://example.org/predicates/hasProcessor>
    <http://example.org/products/SamsungExynos7420> .

<http://example.org/products/SamsungGalaxyS6EdgePlus>
  <http://example.org/predicates/hasRAMMemoryInGB>

```

```

"4"^^<http://www.w3.org/2001/XMLSchema#int> .

<http://example.org/products/SamsungGalaxyS6EdgePlus>
  <http://example.org/predicates/hasDisplaySizeInInches>
    "5.7"^^<http://www.w3.org/2001/XMLSchema#decimal> .

<http://example.org/products/SamsungGalaxyS6EdgePlus>
  <http://example.org/predicates/hasManufacturer>
    <http://example.org/companies/Samsung> .

<http://example.org/products/SamsungExynos7420>
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
    <http://example.org/types/Processor> .

<http://example.org/products/SamsungExynos7420>
  <http://example.org/predicates/hasName>
    "Samsung Exynos 7420 Octa Core"@en .

<http://example.org/products/SamsungExynos7420>
  <http://example.org/predicates/hasNumberOfCores>
    "8"^^<http://www.w3.org/2001/XMLSchema#int> .

<http://example.org/products/SamsungExynos7420>
  <http://example.org/predicates/hasManufacturer>
    <http://example.org/companies/Samsung> .

```

Code 1.4: Information contained in Definition 3 serialized using N-Triples notation.

## Ontologies

In order for data to be correctly connected between multiple datasets and to allow data to be processed by anyone, its necessary to find a way how to capture the correct semantics of the descriptions in a universal way.

An answer for this requirement is an *ontology*. An ontology, in the context of the Semantic Web, expresses a formal and generally recognized specification of concepts and relations, used for description and presentation of specific domain [W3C, 2015a]. That means, an ontology for a domain defines *a dictionary of properties* which can be used to describe the objects (resources) of given domain, their properties and relations with other objects.

Ontology consists of *classes*, which describe resources, and of *predicates* which describe the properties of resources and/or relations between resources.

The RDF framework can be used to map the information about resources to exactly defined ontological properties - and to assign well defined semantic informations to the data.

It is possible to use various ontologies when describing a single resource. That means, a new ontologies do not have to duplicate the properties of already existing ontologies which makes the whole system more transparent.

There are two main languages used for ontology description: **RDF Schema**

(RDFS)<sup>23</sup> and **Web Ontology Language(OWL)**<sup>24</sup>.

Some of the most commonly used ontologies are *vCard*<sup>25</sup>, *Friend of a Friend*<sup>26</sup>, *GoodRelations*<sup>27</sup> or *DBpedia Ontology*<sup>28</sup>.

## 1.3 Web 2.0 Semantization - Annotations

The *Web 2.0 semantization* is a process of converting “human-readable” information represented in the Web 2.0 websites to the “machine-readable” representation (RDF) that can be then joined into the Linked Data datasets and became a part of the Semantic Web.

Several new techniques appeared to help the publishers of web pages to make their websites more semantical by adding additional markup, called *annotations* to already existing web pages. They are adopted by many of the modern web sites. This section will briefly describe the most common annotation techniques.

### 1.3.1 Microformats

Microformats represent one of the first approaches to attach machine readable information inside of conventional Web sites. Current version<sup>29</sup> is the *Microformats 2*. As the website of the project presents Microformats Wiki [2015], Microformats are “the simplest way to markup structured information in HTML”. Microformats are basically a set of predefined classes and their properties which can be used to describe various information - information about people, contact information, information about articles, events, products and so on. The names of classes and properties are prefixed according to their meaning, e.g. class names are prefixed with “**h-\***” (e.g. “**h-card**”), the names of textual properties with “**p-\***” (e.g. “**p-name**”), URL properties by “**u-\***” (e.g. “**u-photo**”) and similar. The names that represent Microformats are included into the **class** attribute of (X)HTML elements. An example HTML snippet which uses microformats, can be found in Code 1.5.

```
<div class="h-event">
  <a class="p-name u-url" href="http://indiewebcamp.com/2012">
    IndieWebCamp 2012
  </a> from <time class="dt-start">2012-06-30</time>
  to <time class="dt-end">2012-07-01</time> at
  <span class="p-location h-card">
    <a class="p-name p-org u-url" href="http://geoloqi.com/">
      Geoloqi
    </a>,
    <span class="p-street-address">920 SW 3rd Ave. Suite 400</span>,
    <span class="p-locality">Portland</span>,</div>
```

<sup>23</sup><https://www.w3.org/TR/rdf-schema/>

<sup>24</sup><https://www.w3.org/2001/sw/wiki/OWL>

<sup>25</sup><https://www.w3.org/TR/vcard-rdf/>

<sup>26</sup><http://xmlns.com/foaf/spec/>

<sup>27</sup><http://www.heppnetz.de/ontologies/goodrelations/v1.html>

<sup>28</sup><http://wiki.dbpedia.org/services-resources/ontology>

<sup>29</sup>At the time of writing of this thesis.

```
<abbr class="p-region" title="Oregon">OR</abbr>
</span>
</div>
```

Code 1.5: An example of IndieWebCamp 2012 event description using Microformats. Source: Microformats Wiki [2015].

### 1.3.2 RDF in Attributes (RDFa)

*RDFa* is a technique which allows creators of the web sites to include the structured, RDF formatted [Herman et al., 2015], machine readable data in their pages together with the human readable content. As the name suggests, this machine readable information is located inside of the various attributes of (X)HTML elements. RDFa uses some of the already existing attributes (such as **href**, **src**), but also introduces its own new attributes (such as **resource**, **property**, **vocab**). In the **property** attribute one can provide the IRI of the predicate whose value is described in the content of the element body. If there exists **href** or **src** attribute in the element, the value of one of this attributes is preferred. The **vocab** attribute can be used to set the IRI prefix of the default ontology. The attribute **property**, which refers to the property of default ontology, does not need to include the IRI of the ontology afterwards. Single Web page can describe multiple resources. In this case, the IRI of currently described resource can be stated in the **resource** attribute. All properties described in the succeeding elements of element with the **resource** attribute will then be assigned to this resource. IRI of the RDF type of the resource can be specified using the **typeof** predicate. *RDFa 1.0* version could be used only in the XHTML-based documents, versions *RDFa 1.1* and later supports the HTML5-based documents as well. The Code 1.6 shows an example HTML snippet which uses RDFa annotations.

```
<body vocab="http://purl.org/dc/terms/">
  ...
  <div resource="/alice/posts/trouble_with_bob">
    <h2 property="title">The trouble with Bob</h2>
    <p>Date: <span property="created">2011-09-10</span></p>
    <h3 property="creator">Alice</h3>
    ...
  </div>
  ...
  <div resource="/alice/posts/jos_barbecue">
    <h2 property="title">Jo's Barbecue</h2>
    <p>Date: <span property="created">2011-09-14</span></p>
    <h3 property="creator">Eve</h3>
    ...
  </div>
  ...
</body>
```

Code 1.6: An example of RDFa usage. Source: Herman et al. [2015].

### 1.3.3 The OpenGraph Protocol

The *OpenGraph protocol* was created by engineers working for the *Facebook*<sup>30</sup> social network to achieve easier sharing of information contained within the web-pages shared across the social network. Currently, it offers wide-spread way to publish structured information, typically about articles on web pages or information about multimedia content. All information published using the OpenGraph protocol are connected into one large “social” graph [The Open Graph protocol, 2015]. OpenGraph protocol uses the methods of RDFa, semantic information are recorded inside of `<meta>` (X)HTML elements in the document header. An example of usage of OpenGraph annotations inside HTML document can be seen in Code 1.7.

```
<html prefix="og: http://ogp.me/ns#">
  <head>
    <title>The Rock (1996)</title>
    <meta property="og:title" content="The Rock" />
    <meta property="og:type" content="video.movie" />
    <meta property="og:url"
      content="http://www.imdb.com/title/tt0117500/" />
    <meta property="og:image"
      content="http://ia.media-imdb.com/images/rock.jpg" />
    ...
  </head>
  ...
</html>
```

Code 1.7: An example of OpenGraph description about the “The Rock” movie.  
Source: The Open Graph protocol [2015].

### 1.3.4 JSON-LD

According to its specification in W3C [2014b], JSON-LD is a W3C recommendation, which provides a way to serialize Linked Data using lightweight JSON format<sup>31</sup>. This allows the creators of web pages to embed a “machine-readable” copy of the data in the HTML document next to the “human-readable” data. Since the JSON-LD notation is actually a JSON, it can be processed using all existing JSON manipulation libraries. The semantic information is stored in special JSON attributes, called *syntax tokens* (e.g. “@id”). An example of JSON-LD data description can be seen in Code 1.8. The most recent version of the JSON-LD specification is <sup>32</sup> JSON-LD 1.0.

```
{
  "@context": "https://json-ld.org/contexts/person.jsonld",
  "@id": "http://dbpedia.org/resource/John_Lennon",
  "name": "John Lennon",
  "born": "1940-10-09",
```

<sup>30</sup>Facebook: <https://www.facebook.com>

<sup>31</sup><https://www.json.org/>

<sup>32</sup>At the time of writing of this thesis.

```
"spouse": "http://dbpedia.org/resource/Cynthia_Lennon"  
}
```

Code 1.8: An example of JSON-LD description about John Lennon. Source:  
<https://json-ld.org/>.



## 2. Semantic Table Interpretation & Related Work

The Chapter 1 described the differences between the Web 2.0 and the Semantic Web and provided an overview of technologies and challenges related to those Web concepts. This chapter will introduce the background of *Semantic Table Interpretation* and will also provide an overview of selected related works and approaches aiming towards tabular data semantization.

### 2.1 Semantic Table Interpretation

According to [Zhang, 2014], one of challenges towards creation of the Semantic Web is semantization of data published in form of structured tables which tend to contain relational data. [Zhang, 2014] also states, that the number of published tables is rapidly growing, it is getting more and more important to find ways how to recover semantics of these tabular data in order to connect the data to the Linked Data cloud which would allow the data to be more easily processable by machines.

[Zhang, 2014] also introduces *Semantic Table Interpretation* as a field of research which is focused on the problem of retrieval of semantic information from the tabular data — the *Table Semantization*. Author describes following three main tasks which the *Semantic Table Interpretation* addresses:

- *Disambiguation of cells of the table*
- *Classification of table columns*
- *Identification of relations between table columns*

Before the actual explanation of particular tasks, it is necessary to state, that each column of given table can be of one of two types – either *NE-Column (Named Entity Column)* or *Literal Column*. These two column types are introduced in Definitions 4 and 5. All definitions are based on Zhang [2016].

**Definition 4** (NE-Column (Named Entity Column)). *A table column which references a semantic entity.*

**Definition 5** (Literal Column). *A table column which does not reference an entity, but contains the literal value (number, text string, etc.) instead.*

The Semantic Table Interpretation tasks are defined in Definitions 7, 6 and 8.

**Definition 6** (Entity Disambiguation). *A process of association of a single NE-cell of a table with a single canonical entity (resource).*

**Definition 7** (Column Classification). *A process of annotating the columns with determined best-fit concept (class) — for NE-columns — or — for literal columns — a single property of a concept.*

**Definition 8** (Relation Identification). *A process of identification of semantic relations (predicates) between the table columns.*

The result of the Semantic Table Interpretation process is tabular data enriched with semantic annotations which can be used to efficiently search the data, but can also be converted to RDF triples and then connected to the Linked Data cloud.

## 2.2 Existing Approaches

This section aims to introduce some of already existing relevant approaches towards solving of the *Semantic Table Interpretation* problem described in the beginning of this chapter. It shows that a huge effort has been given by various research groups in order to improve automatic table semantization over the last years.

Finin et al. [2010] in their approach use the help of *Wikitology*<sup>1</sup> in their Wikipedia<sup>2</sup>-based web tables semantization methods. Wikitology is an index of concepts (Wikipedia articles represent concepts) which is extended with the fields from different knowledge bases (such as *DBpedia*, *Freebase*<sup>3</sup>, *Yago*<sup>4</sup>). It contains for example fields: *Title*, *Redirects*, *First Sentence in Article*, *Wikipedia categories*, *Types (DBpedia, FreeBase, Yago types)*, and others. Wikitology accepts two types of queries - *text queries*<sup>5</sup> or *structured constraints*<sup>6</sup>. The method described by authors at first detects the concepts of the columns by performing queries with the column header and its values to the Wikitology index by multiple properties, such as *title*, *types*, *first sentence*, and so on. Each of returned concept candidates is scored, and the one with highest score is elected as the column concept.

Limaye et al. [2010] experiment with completely different method. This approach represents tables using a probabilistic graphical model. For querying for the entities and relations, authors use Yago knowledge base. This algorithm performs the cell disambiguation, column classification and relation enumeration tasks simultaneously which according the authors claim has “accuracy benefits compared to the local decisions”. Table components (called *variables*) are represented as *variable nodes* of the graph. The relation between variables and their candidate concepts are represented as *factor nodes* which can represent coupling of multiple variables. The inference is then determined as finding values of variables which will lead to a maximum probability to create joints.

Venetis et al. [2011] experimented with annotation of column types and binary relations between the pairs of columns in the tables on a large testing set consisting of about 12 million tables. Instead of using existing knowledgebases, the authors constructed 2 databases: the first one, called *isA*, contained tuples of (Class, Instance); second one, *relations table*, contained triples (a, Predicate,

---

<sup>1</sup><http://ebiquity.umbc.edu/project/html/id/83/Wikitology>

<sup>2</sup>Wikipedia, the free encyclopedia: [https://en.wikipedia.org/wiki/Main\\_Page](https://en.wikipedia.org/wiki/Main_Page)

<sup>3</sup><https://developers.google.com/freebase/>

<sup>4</sup><http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago>

<sup>5</sup>Text queries are for example words in a document.

<sup>6</sup>An example of structured constraint is e.g. *rdfs:type == yago:Person*

b). These databases were filled from the data scraped from the Web using lexicographical patterns. A *maximum likelihood* model is used to determine the best match of column concepts and relations based on the *co-occurrence* statistics that were gathered while filling-up the databases during the pattern extraction. To enumerate the relations between the concepts, the algorithm queries DBpedia to retrieve list of possible relations between two entities for each row (two entities in different columns of the same row of the table). Any found relation is considered as candidate relation between the two columns. The relation with the most occurrences is considered as the relation between concepts.

Wang et al. [2012] used similar approach, but instead of building their own database of facts, they use the *Probase*<sup>7</sup> database. The authors performed the experiments on a huge testing set of approximately *65 million* web tables which according to their research potentially contained *valuable information*. According to the authors, the majority of the tables contain one *entity* (subject) *column*, and other columns which describe the attributes of the entities. Their algorithm uses this knowledge and at first tries to identify the subject column and its Probase concept, and then uses Probase to identify the schema of the table.

Muñoz et al. [2013] focus on triplication of Web tables embedded in the bodies of *Wikipedia*. articles. They are interested in the table cells which contain *internal links*<sup>8</sup>. These URIs of these links are at first converted to the DBpedia resource URIs<sup>9</sup>. The relation enumeration is done similarly to Venetis et al. [2011] - by querying DBpedia for list of relations between entities. In Muñoz et al. [2014], authors further improved this process by using several machine learning approaches<sup>10</sup> to filter out the incorrect tripples and thus improve the algorithm precision.

Zwicklbauer et al. [2013] in their experiment used a simple algorithm which does not take into account any relation between columns, and does not even use the values of table column headers. For each column, the algorithm retrieves a list of candidate entities for specified number<sup>11</sup> table cell in given column. Each of this entity candidates yields a set of types<sup>12</sup>. The type with most occurrences is then used as the type of the whole column. Even though the dataset authors used for experiments was rather small, they proved that it is not necessary to disambiguate all of the cells in the column to achieve similar precision.

Zhang [2014] introduces the *TableMiner* algorithm which consists of 2 phases: *forward-learning* phase and *backward-learning* phase. In the forward-learning phase, the *Incremental inference with stopping (I-inf)* algorithm<sup>13</sup> is used which iteratively searches for disambiguation candidates for the table cell values. According to the score of found concept candidates, the concept with highest score is used as column classification. The *backward-learning* phase furtherly improves the classifications and disambiguations found by the forwards learning phase. The

---

<sup>7</sup><https://www.microsoft.com/en-us/research/project/probase/>

<sup>8</sup>Links leading to other Wikipedia pages.

<sup>9</sup>This is done by replacing the <http://en.wikipedia.org/wiki/> prefix with <http://dbpedia.org/resource/> prefix.

<sup>10</sup>Such as: Naive Bayes, Bagging Decision Trees, Random Forest, and others.

<sup>11</sup>Authors were observing the precision based of number of actually used cells in a column.

<sup>12</sup>Authors determined the types by looking at the *rdf:type* and *dcterms:subject* RDF properties of candidate entities.

<sup>13</sup>The algorithm is described in Chapter 3.

author continued his work on further improving of the TableMiner algorithm which resulted into creation of the *TableMiner<sup>+</sup>* algorithm described in Zhang [2016]. As this thesis is directly related to the algorithm, it is necessary to completely understand how the algorithm works. For this reason, the *TableMiner<sup>+</sup>* algorithm will be described in detail in the following Chapter 3.

## 3. TableMiner<sup>+</sup> and Odalic

The Chapter 2 introduced the motivations behind *table semantization* challenges and defined *Semantic Table Interpretation* as approach to solve this challenge. It also described existing works focused on semantization of tabular data. The chapter finished with introduction of *Ziqi Zhang* and his *TableMiner<sup>+</sup>* algorithm. This algorithm — and its extension called *Odalic* — will be introduced in more details in this chapter, since the rest of this thesis is directly related to these algorithms.

### 3.1 TableMiner<sup>+</sup> Algorithm

TableMiner<sup>+</sup><sup>1</sup> algorithm was originally developed by *Ziqi Zhang* while working at the *Department of Computer Science at University of Sheffield*.

In Zhang [2016], the author himself describes the algorithm as “a Semantic Table Interpretation method that annotates Web tables in a both effective and efficient way”.

The autor mentions some of the key advantages of the algorithm, compared to other Semantic Table Interpretation algorithms:

- *Taking advantage of out-table contexts*: Whereas the other semantic table interpretation algorithms are capable of deriving the semantic information only from the (so called) *in-table context* - table components, such as table heading row, as author says, the TableMiner<sup>+</sup> algorithm does use also the (so called) *out-table context*, such as the other textual information in the text of the HTML file outside of the scope of the table. For example, having a table containing a column called “genre”. Genre is pretty general term and can have a different meanings - e.g. movie genre, or music genre, etc. However, the algorithm is able to find multiple occurrences of the word “book” in the document, outside the scope of a table. From this information, the algorithm can infer that the column contains a collection of literary genres.
- *Efficiency*: Instead of processing of all of the items in the table, the algorithm processes only a limited amount of them. This allows it to cut down the number of requests to the knowledge bases. This approach does not only improve the overall performance, but can also reduce the financial costs of knowledgebase requests.
- *Completeness*: The algorithm tries to identify as much information as possible. This is achieved using the “start small, build complete” principle. The algorithm starts with a preliminary annotation-based model created from partial table data which is likely to error-proof. This model is then iteratively optimized by enforcing the relations between its components.

---

<sup>1</sup>Source codes of the TableMiner<sup>+</sup> algorithm can be found on GitHub: <https://github.com/ziqizhang/sti>.

As described in the first advantage in the list above, the input data for the algorithm do not have to be just plain tables (e.g. CSV), but can also be HTML files containing data in the table representation, together with other contextual information.

As an example, an IMDb<sup>2</sup> dataset is provided which consists of a collection of HTML files containing details about movies, one movie per file. Each of these files contains (among other things) the table of actors and their roles in given movie.

### 3.1.1 Algorithm Flow

This section provides a brief description of the individual steps in the TableMiner<sup>+</sup> algorithm flow, based on the algorithm description in Zhang [2016], to help with a better understanding of the process. The algorithm depends heavily on the usage of the **Incremental inference (I-Inf)** algorithm. This is an iterative algorithm which in each iteration processes the entries of the given dataset to find new, more accurate, key-value descriptions of the entries. The algorithm ends when convergence is found.

The detailed diagram of the algorithm flow can be seen in Figure 3.1. The individual algorithm steps are described below.

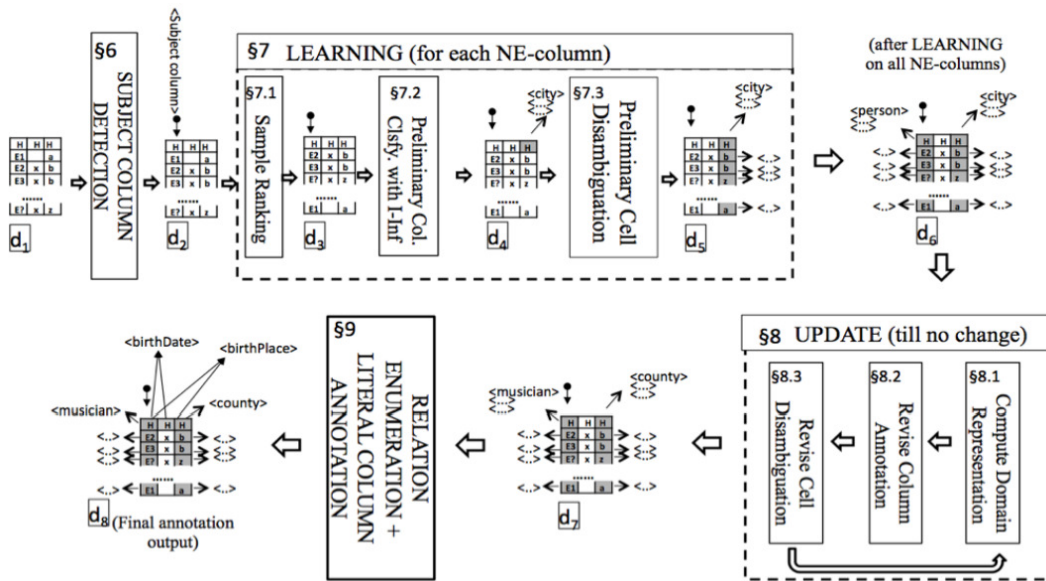


Figure 3.1: TableMiner<sup>+</sup> Algorithm Flow. Source: Zhang [2016]

The comprehensive description of both, the TableMiner<sup>+</sup> and the I-Inf algorithms can be found in Zhang [2016].

### Subject Column Detection

In this phase, the algorithm tries to detect a *subject column*. This phase consists of several steps:

<sup>2</sup>International Movie Database: <http://www.imdb.com/>.

1. *Detect the data type for each columns* - the algorithm assigns a data type to each of the table columns. The data type of a column is determined as most common data type from all of the column values.
2. *Ignore not important columns* - the columns that clearly are not subject columns (if a column header is a preposition word) are removed from the list of subject column candidates.
3. *Feature computation* - several features, such as *empty cell fraction* or *cells with unique content ratio* (a comprehensive list of these features together with their description can be found in *Table 2* of Zhang [2016]) are computed for every candidate column.
4. *Detection* - based on the features computed in previous step, a final *subject column score* is calculated for each candidate column. Candidate column with a highest score is marked as subject column.

### NE-Column Interpretation - Learning phase

In this phase, the algorithm processes independently each NE-column in the table, except the subject column chosen in previous step, in order to create a preliminary classification of columns and preliminary disambiguation of cells. Thus, following sequence of steps is taken for each column in the table:

- *Sample Ranking* - Each cell in the column receives a *preference score*, and rows containing these cells are reordered in descending order, based on this score. A hypothesis called *one-sense-per-discourse* is used. This hypothesis determines that if a column contains multiple cells with the same polysemous value, all of these cells are considered to represent the same entity. This allows the algorithm to “combine” information from all of these rows to create an enhanced feature representation of the entity. The number of all features in this enhanced representation determines the preference score of given entity.
- *Preliminary Column Classification* - In this step, the TableMiner<sup>+</sup> algorithm uses the *I-Inf* algorithm to find candidate entities for the cells in ordered table from previous step. For each of the cells, a set of concepts is retrieved from the knowledge base(s). If any of the concepts of the current cell is new, it is added to the list of concepts for the column. If the concept has already been defined for the column by any of the previous nodes, the score of the concept is updated. This process is repeated by the *I-Inf* algorithm until the convergence is reached. The concept with best score is then used to annotate given column.
- *Preliminary Cell Disambiguation* - For each column, the algorithm disambiguates cells by assigning the knowledge base entities that represent the instance of winning column concept. If there are more concepts assigned to the instance, the column concepts are further updated, and after this phase, the concept of the column can even change.

## NE-Column Interpretation - Update phase

In this phase, the algorithm takes the preliminary column classification and cell disambiguations of the NE-columns created in the learning phase, and performs further column and cell annotation updating based on the found relations not only between the column classifications and cell disambiguations, but also based on the inter-column relations. This process is repeated until the annotations reach their final state - and they don't update anymore.

## Relation Enumeration and Literal-Columns Annotation

This phase consists of two steps:

- *Relation Enumeration* - In this step, at first the relations between the subject column and every other columns are enumerated separately. A set of candidate relations between two columns are retrieved from the knowledge base for every row in the table and they are assigned a confidence score, and a winning candidate for each row is selected. A set of candidates for the entire relation is constructed as an union of all row-relation candidates. For each of the candidates, a relation context score is computed and the winning candidate is selected as the relation between the subject column and the given second column.
- *Literal-columns annotation* - As the literal columns are supposed to contain the attribute values, and are not NE-columns, they cannot be processed by the phases described in previous subsections. In the TableMiner<sup>+</sup> algorithm, the column header of the literal column is used as the annotation label of the values in the column cells.

## 3.2 Odalic Project and Odalic Core

Whereas the original TableMiner<sup>+</sup> algorithm implementation was indeed working properly, it is more of a proof of concept than a production system which would allow its users to comfortably process their data and display the outputs in user-friendly form.

Odalic project<sup>3</sup>, a software project developed by the students of *The Faculty of Mathematics and Physics at Charles University in Prague*, tries to address this disadvantage and aims to be a production-grade table semantization system. The project also introduces further enhancements to the main TableMiner<sup>+</sup> algorithm. Precisely, Knap [2016] defines the goals of the project as following:

*Remark* (Odalic Core (STI)). The Odalic project consists of several components. The main component which is built atop of the TableMiner<sup>+</sup> algorithm is referred to as **Odalic Core (STI)**<sup>4</sup>.

- *Improve the classification, disambiguation and relation discovery parts of TableMiner<sup>+</sup> algorithm.*

---

<sup>3</sup>Source codes of the Odalic project can be found on GitHub: <https://github.com/odalic>.

<sup>4</sup>Semantic Table Interpretation



- *Implement user interface for interaction with Odalic Core*
- *Provide API to allow applications to use Odalic Core algorithm easily*
- *Provide executor for the automated tabular data to RDF conversions*
- *Provide support for additional knowledge bases*

Knap [2017] furtherly describes that the Odalic system consists of a server application which provides an API server and User Interface (*Odalic UI*). This user interface allows the users to provide their data (in form of CSV files), processes the data and displays the output in user-friendly user interface, together with export in several formats, such as *RDF (Turtle notation)* or *CSV on the Web*<sup>5</sup>.

Apart from the server application, the Odalic project also introduces several improvements to the original *TableMiner<sup>+</sup>* algorithm [Knap, 2017]:

- Replaced deprecated Freebase<sup>6</sup> knowledge base with DBpedia<sup>7</sup>.
- Added support for general SPARQL-based knowledge bases.
- Support for querying multiple knowledge-bases for a single job.
- Eliminating obviously wrong results using data-type constraining.

Apart from that, the Odalic system allows the user to provide a *manual* feedback in order to help the algorithm with column classification and/or cell disambiguation. This provides an (quite) simple way of semantization of massive amounts of data.

### 3.2.1 Odalic Project Components

The Odalic project consists of three main components:

- *Odalic-STI - Semantic Table Interpretation* server component which provides an API for task management. Also provides runtime for task processing which extends the original *TableMiner<sup>+</sup>* implementation.
- *Odalic-UI* - Provides Web-based user interface for the server component which allows users to create, configure and submit tasks and review their results. The UI can be also extended with a customized *LodLive*<sup>8</sup> component which allows it to browse the known knowledge bases.
- *Odalic-UV-Plugin* - Odalic plugin for RDF ETL<sup>9</sup> tool *UnifiedViews*<sup>10</sup>.

<sup>5</sup><https://www.w3.org/TR/tabular-data-primer/>

<sup>6</sup><https://developers.google.com/freebase/>

<sup>7</sup><http://wiki.dbpedia.org/>

<sup>8</sup><http://lodlive.it>

<sup>9</sup>Extract, Transform, Load

<sup>10</sup><https://github.com/UnifiedViews>

*Odalic-STI* is a java application, compiled to a *war* archive which needs to be launched by an external application server, such as *Apache Tomcat*<sup>11</sup>.

*Odalic-UI* is a *JavaScript (AngularJs framework*<sup>12</sup>) web application which needs a HTTP server to serve its files to the end-user. It can run either on the same host together with the *Odalic-STI* component (and being served by the same application server), or can be running on completely different host. The preview of results of *column classification* and *entity disambiguation* tasks in *Odalic UI* can be seen in Figure 3.2.

More details about the project, including installation and usage manual can be found in *ODALIC [2017]*.

The screenshot shows the 'Column classifications and disambiguations of the values in cells' screen in the Odalic UI. The interface includes a navigation bar at the top with 'Tasks', 'Input files', and 'Knowledge bases' menus, and a 'My account' link. Below the navigation, there are tabs for 'Classifications and disambiguations', 'Subject columns', and 'Relations'. The main content area displays a table with the following columns: 'Name', 'Ort', and 'Bezirk'. The table lists several rows, each representing a location and its classification into ontology-based categories. For example, 'Actor' is classified as 'Location (dbpedia-owl:Location)', 'Spatial Thing (geo:SpatialThing)', and 'Concept (skos:Concept)'. The 'Bezirk' column shows the corresponding district or region, such as 'Bezirk (dbpedia-owl:Location)' and 'Spatial Thing (geo:SpatialThing)'. At the bottom of the table, there are 'Reexecute', 'Save', and 'Cancel' buttons. Below the table, there is an 'Export' section with buttons for 'Extended CSV', 'Annotations', 'RDF (turtle)', and 'RDF (JSON-LD)'. The footer of the page indicates 'Odalic v1.1'.

Figure 3.2: Odalic UI - Feedback screen.

<sup>11</sup><http://tomcat.apache.org/>

<sup>12</sup><https://angularjs.org/>

# 4. Background of This Thesis

Chapters 1 - 3 described differences between concepts of the “human-centric” Web 2.0 and the “machine-centric” Semantic Web, motivation behind the task of semantization of tabular data, and introduced some of already existing approaches, with a deeper focus to the *TableMiner<sup>+</sup>* algorithm and the *Odalic* system. This chapter will continue by explaining of the motivation behind this thesis, outlining the goal of this thesis, and will introduce the approach taken by the author of this thesis in order to reach this goal.

## 4.1 Motivation

Nowadays, a very popular and convenient form of buying stuff is shopping on the Internet using e-shops and various Internet marketplaces. Each larger e-shop and especially a marketplace, usually sells thousands of various products. For each of these products, e-shops and marketplaces do have a separate web page, a *product detail* page. This page contains a description of a given product, photos of the product, price, and more or less detailed information about the specification of given product. An example of such product page can be seen on the Figure 4.1.

A section of the product detail page which contain product specifications (an example can be found in the bottom right part of Figure 4.1) usually contains a table of listed product properties and values of those properties.

These product specifications are certainly very useful for the customer, while comparing multiple products in order to decide which one he should buy. However, if a website of an e-shop does not support any semantic annotations<sup>1</sup>, it is not possible to easily semantize the contained data and connect them into the Linked Data cloud to allow them to be processed by machines.

With the knowledge from Chapters 1 - 3 a straightforward approach towards semantization of such product data would be to *scrape* (download) product detail pages from a website of e-shop, extract important information, such as *title*, *price* and *specification* of the products into a table(s) (e.g. CSV file(s)) and semantize these files using the *Odalic* system.

There is however a problem with this approach. The semantization algorithm of the *Odalic* system, while performing its semantic table interpretation tasks, such as *classification* of table columns and *disambiguation* of table cell, depends heavily on the data already present in selected knowledge bases. Since there are thousands and thousand of different products, many of which also have relations to other products and entities (e.g. key property of a cell phone is the type of its CPU<sup>2</sup> which can be also sold as a product), and new products are being introduced every day, there just is not enough information about all of the products in the knowledge bases, and thus *Odalic* would not be able to infer much semantic information in order to correctly find semantic information for the data.

---

<sup>1</sup>Introduced in Chapter 1, section 1.3.

<sup>2</sup>Central Processing Unit

## 4.2 Goal

The main goal of this thesis is to propose an improved (extended) version of a Semantic Table Interpretation algorithm for the Odalic system which would allow the system to semi-automatically infer semantic information on product data obtained from e-shops. The general guidelines of improvements which this algorithm should take into consideration are:

- Detection of predicates (attributes) of a product.
- Detection of classes of products based on the detected predicates.
- Disambiguation of values of attributes against generic Linked Open Data knowledge bases.
- Detection of predicates for the rest of attributes. If the algorithm detects that that certain product is a laptop, then try to match rest of the predicates according to properties which are usually assigned to laptops.
- Output the converted data as Linked Data.

For the detection of *classes* and *predicates* of individual columns, which represent properties and features of a product, the algorithm should use machine learning technique which will allow the algorithm to detect these classes and predicates based on the similarity<sup>3</sup> with example values in the training dataset.

An additional goal of this thesis is to obtain data about products from HTML pages of several e-shops which then will be used to train the algorithm and will be semantized by the algorithm in order to perform an evaluation of the algorithm.

The thesis focuses on the following types of products: *cell phones*, *tablets*, *laptops*, *desktop computers*.

---

<sup>3</sup>Value feature-wise similarity. The topic of machine learning and value features is discussed in the Chapter 6.

DEBENHAMS PLUS
Back to Debenhams Homepage | Order Tracking | Cookie Information

My Shopping Bag  
0 items

---

Laptops, Tablets & PCs | Mobile Phones | TV & Home Cinema | Large Kitchen Appliances | Dehumidifiers & Air Con | CCTV & Security | Gadgets |
SALE

---

**FREE DELIVERY**  
Click for postcode exclusions

**NEXT DAY DELIVERY AVAILABLE!**

**PAY WITH DEBENHAMS GIFTCARD**

---

Home » Phones And PDAs » Smartphones » ASUS » 90AK0021-M00660

[Compare](#) | [Add to wishlist](#) | [View Printable Version](#)

## Asus Zenfone AR Black 5.7" 128GB 4G Dual SIM Unlocked & SIM Free - 90AK0021-M00660

Quickfind code: 1226235 ★★★★★ [Read 1 review](#)

Hover image to zoom

View larger image

You save: £189.99

# £610.00

Add to Basket »

✓ **In Stock**, Order within **5 hrs, 2 mins, 23 secs** for Delivery from tomorrow

✓ **FREE Delivery**

+ [See all delivery & collection options](#)

---

- **Why buy me**

- Unlocked & SIM Free
- 4G Ready for ultra fast network speeds
- Groundbreaking TriCam System - High Resolution 23MP, Motion Tracking Depth Sensing 8 Megapixel selfie camera
- 5.7 2K Super AMOLED Display
- Super fast Snapdragon 821 Octa Core Processor + 6GB of RAM
- Android 7.0 Nougat OS
- 128GB Storage + 200GB micro SD card slot
- Worlds 1st AR VR phone!
- Quick Charge 3.0: 60% battery life in under 40 minutes!

[More Info »](#)
[Tech Spec »](#)

---

+ [Valid promotions & discounts](#)

---

+ [Ways to pay](#)

### Product Information

---

**Asus Zenfone AR Black 5.7" 128GB 4G Dual SIM Unlocked & SIM Free**

**Key Features:**

- Groundbreaking TriCam System - High Resolution 23MP, Motion Tracking & Depth Sensing & 8 Megapixel selfie camera
- 5.7" 2K Super AMOLED Display
- Super fast Snapdragon™ 821 Octa Core Processor + 6GB of RAM
- Android 7.0 Nougat OS
- 128GB Storage + 200GB micro SD card slot

**The world's first Tango enabled and Daydream-ready phone**

ZenFone AR is the world's first 5.7-inch smartphone with Tango and Daydream by Google. Tango is an exciting new augmented reality (AR) technology that changes the way you interact with the world and expands your vision. And with Daydream, you can experience high-quality, immersive virtual reality (VR) with your phone.

### Tech Spec

---

Technical Specifications	
Network Speeds	4G
Rear Camera Quality	23 Megapixel
Front Facing Camera Quality	8 Megapixel
Screen size	5.7 Inches
Screen Resolution	2560 x 1440 pixels
Operating System	Android 7.0 Nougat
Processor Type	Snapdragon 821
Processor Speed	2.3GHz
RAM	6GB
Onboard Storage	128GB
SIM card type	Nano SIM
Dual Sim	Dual SIM
Charging Cable Type	USB Type C

### More Asus Zenfone products

---

Asus Zenfone 4 Max Deepsea Black 5.5" 32GB 4G Dual SL...

£239.00

Asus Zenfone AR Black 5.7" 128GB 4G Dual SIM Unlocked ...

£610.00

See all Asus Zenfone products »

### Recommended Accessories

---

Bluetooth Key finder Phone finder Wallet finder

£14.99

Include in Order

Figure 4.1: Example of product detail page of DebenhamsPlus.com e-shop (cropped).

# 5. Obtaining Product Data

As mentioned in the Chapter 4, the goal of this thesis is to propose an improved version of Odalic semantization algorithm which will be more effective in semantization of product data from e-shops. In order to achieve this, it is necessary to at first gather such data. Since this improved algorithm will take the advantage of machine learning classification technique, it is necessary to have some data for the purposes of evaluation of various classification algorithms. This evaluation is required in order to determine the algorithm with best performance. This chapter describes the process of downloading, or *scraping*, of HTML documents representing product data from several e-shops. Since — as described in chapter 3 — the Odalic system supports only input data in form of CSV files, downloaded HTML documents need to be converted to the CSV format before they can be actually passed as input to the semantization algorithm. The process of HTML to CSV conversion is also described in this chapter.

## 5.1 Downloading HTML Product Details

The HTML files of product details were scraped from e-shop websites using a tool called ItSucks<sup>1</sup>. This is an open-source — created in Java<sup>2</sup> programming language, and therefore running on a wide variety of platforms — simple-to-use application for downloading the web pages as HTML documents. It has many useful features, such as proxy support or variety of filters which tells the application what criteria the URL of the page must meet in order to be downloaded.

In order to scrape data (HTML documents) from a website, it is necessary to create a configuration file tailored specifically to the website whose data are going to be scraped. This can be done easily using the GUI<sup>3</sup> of the application.

After several attempts to scrape the product data from multiple e-shops. Various issues with the scraping were discovered in this phase, mostly because chosen e-shops took use of JavaScript code to asynchronously fetch and insert the data into HTML template. In some cases, portions of the data were even loaded only after user action, such as clicking on “more details” button. These e-shops could not be scraped because of the JavaScript usage.

Product detail pages of following e-shops were successfully scraped for purposes of this thesis:

- **Debenhams Plus UK** (<https://www.DebenhamsPlus.com/>)
- **GearBest** (<https://www.GearBest.com/>)
- **MobileShop EU** (<https://www.MobilEshop.eu/>)

Configuration files used to scrape product data from these e-shops — their product detail pages — can be found in the folder “/files/itSucksEshopTemplates” on the *attached CD*.

---

<sup>1</sup><http://itsucks.sourceforge.net>, source codes of the application can be found at: <https://sourceforge.net/projects/itsucks/files/itsucks/>

<sup>2</sup><https://www.java.com/en/>

<sup>3</sup>Graphical User Interface

### 5.1.1 Scraped HTML Documents

The scraping tool introduced in Section 5.1 crawles websites and downloads the content of pages that meet the configured constraints in form of HTML documents. These HTML documents contain only the raw HTML structure of the page. No additional linked resources, such as images, styles or script files are downloaded. These are actually not needed, as all information important for conversion to the machine-readable format are contained in the HTML structure itself, and the role of these resources is just to make the web page look more “human” friendly.

As an example, an original product detail page of *DebenhamsPlus.com* e-shop can be seen on the Figure 4.1 in Chapter 4.

The same product detail page without additional resources can be seen in Figure 5.1.

As seen in the given examples, its clear that all of the important information (such as product name, features, etc.) are contained in scraped HTML documents.

After the scraping tool finished, following number of HTML documents were scraped by the scraping tool for each e-shop of interest:

- **Debenhams Plus UK** - 1111 HTML documents.
- **GearBest** - 598 HTML documents.
- **MobileShop EU** - 264 HTML documents.

HTML files downloaded from all e-shops can be found in the folder “/files/inputData/raw” on the *attached CD*.

## 5.2 HTML to CSV Conversion

The Odalic system does<sup>4</sup> only accept input files in the form of CSV files. Therefore, in order to be able to process the scraped data using the Odalic system, it is necessary to convert these data from HTML to CSV format first.

For the purpose of the conversion, the author of this thesis designed and implemented a custom application which takes the HTML files as its input and produces CSV files containing important information retrieved fetched from the input files, as its output.

The conversion process takes the advantage of the fact that all of the HTML documents represent the “same” type of page - *a detail of a product in the e-shops catalog*. These pages do have more or less similar structure even among different e-shops: They all contain the *name of the product* as one of the page headings, the *price of the product*, and some sort or *enumeration of important properties and features of the product*. These properties / features are typically represented in HTML as e.g. a *table*, or an *unsorted list* elements. Figure 5.2 shows the sample of scraped HTML feature list of *Apple MacBook Pro* laptop (represented by a HTML table), as published by *DebenhamsPlus.com*. Other e-shops feature lists (tables) are usually similar.

---

<sup>4</sup>At the time of writing of this thesis.

The conversion application is called *HtmlToCsvTool*. It is implemented in Java programming language and is designed as modular. The conversion logic of each site is encapsulated in one simple parser class. The advantage of this approach is, that even though there needs to be separate parser provided for each type of website whose HTML data need to be converted, it is relatively easy to extend the application to support new types of websites (e.g. for different e-shops) by providing a new implementation of the “parser” Java class<sup>5</sup> for the new page type which then can be used by the application.

The application also takes advantage of the current situation on the hardware field, especially the fact that multi-core processors are very common these days. The conversion process of this application is able to utilize these multiple processor cores in a way that the input HTML documents are split to multiple sets of independent documents. Each of the sets is then assigned to one of the multiple threads and can be processed in parallel to other document sets. The number of maximum parallel threads is configurable. This results in better performance of the application.

The HTML to CSV file conversion focuses on the important parts of the page (such as product title and specifications of product properties/features) which are extracted by the parser implementation for given website using the CSS selectors of corresponding HTML elements (such as elements containing the product title, or elements which represent a table of product specifications). All other unnecessary content (e.g. advertisements, page header, footer, recommended similar products, etc.) is discarded.

Many of the e-shops use the multiple tables for the description of product properties/features on a single page (for a single product), e.g. separate table for *CPU* properties, *Display* properties, and so on. The *HtmlToCsvTool* respect this, and it keeps a list of properties for each of found categories. All features/properties of all categories that are found, are then exported into a CSV file, with the headings in the form of “category.feature”. The category with no heading assigned (usually the one containing general information, is called **root**).

Each CSV document contains all product information retrieved from the HTML documents of one e-shop.

A quick start guide on how to get the sources of the application, compile, configure and run the application can be found in Attachment A.

Job configuration files for the e-shops mentioned earlier in this chapter can be found in the “/html-to-csv-tool/conf/jobs/” folder on the *attached CD*.

### 5.3 Converted CSV files

The result of HTML to CSV conversion using the *HtmlToCsvTool* tool described in Section 5.2 is a set of output documents in CSV format. For simplicity reason, the pipe (“—”) symbol was used as a separator for these CSV files. These documents represent the important properties and features of the products contained in the scraped HTML data, one file per e-shop. After the conversion, these CSV files also contain a first column with URIs of the product HTML pages. Since

---

<sup>5</sup>For more information about adding new parser see section B.6 in the Attachment B.




these URIs were not be needed in the rest of this thesis, this column was removed from all CSV files. The e-shop based files were also manually split into several files, according to the product category (“phones”, “laptops”, etc). After this preprocessing, the files are now in a form which is valid to be given to the Odalic Core system. Figure 5.3 shows an example (cropped, as it is too large to fit on the page) of a converted and preprocessed CSV file.

- [Smartphones](#)
- [ASUS](#)

## Asus Zenfone AR Black 5.7" 128GB 4G Dual SIM Unlocked & SIM Free 90AK0021-M00660

Quickfind code: 1226235

You save:  £189.99

 £610

[Add to Basket](#)

### Product Information

#### Key Features:

- 4G Ready for ultra fast network speeds
- 5.7" 2K Super AMOLED Display
- Super fast Snapdragon™ 821 Octa Core Processor + 6GB of RAM

#### The world's first Tango enabled and Daydream-ready phone

ZenFone AR is the world's first 5.7-inch smartphone with Tango and Daydream by Google. Tango is an exciting new augmented reality (AR) technology that changes the way you interact with the world and expands your vision. And with Daydream, you can experience high-quality, immersive virtual reality (VR) with your phone.

[Show more](#)

### Tech Spec

#### Technical Specifications

Network Speeds	4G
Rear Camera Quality	23 Megapixel
Front Facing Camera Quality	8 Megapixel
Screen size	5.7 Inches
Screen Resolution	2560 x 1440 pixels
Operating System	Android 7.0 Nougat
Processor Type	Snapdragon 821
Processor Speed	2.3GHz
RAM	6GB
Onboard Storage	128GB
SIM card type	Nano SIM
Dual Sim	Dual SIM
Charging Cable Type	USB Type C
Device Status	Unlocked

Figure 5.1: Example of scraped product detail page of DebenhamsPlus.com e-shop (cropped).

## Tech Spec

Key Specification	
Screen size	13.3in
Processor manufacturer	Intel
Processor Model	Intel Core i5
Processor Number	5257U
RAM Memory	8GB
Storage	128GB
Screen Resolution	2560 x 1600
Operating System	OS X 10.12 Sierra
Optical Drive	No optical drive
Battery Run Time	12hours
Touchscreen	Non Touch
Warranty	1 year warranty
Display and Keyboard	
Screen size	13.3in
Screen Resolution	2560 x 1600
Touchscreen	Non Touch

Figure 5.2: Example of product feature list of DebenhamsPlus.com e-shop (cropped).

root_title	Body.Dimensions	Body.Weight	Memory.Internal	root_price
nokia 3310 (2017) dual sim dark blue	115.6 x 51 x 12.8 mm (4.55 x 2.01 x 0.50 in)	109 g	16 mb	94 €
nokia xl dual sim blue	141.4 x 77.7 x 10.9 mm	190 g	4 gb, 768 mb ram	133 €
nokia xl dual sim black	141.4 x 77.7 x 10.9 mm	190 g	4 gb, 768 mb ram	133 €
nokia xl dual sim orange	141.4 x 77.7 x 10.9 mm	190 g	4 gb, 768 mb ram	133 €
nokia 3310 (2017) dual sim gray	115.6 x 51 x 12.8 mm (4.55 x 2.01 x 0.50 in)	109 g	16 mb	
nokia 3 dual sim 16gb black	143.4 x 71.4 x 8.5 mm (5.65 x 2.81 x 0.33 in)	140 g (4.94 oz)	16 gb, 2 gb ram	163 €
nokia xl dual sim yellow	141.4 x 77.7 x 10.9 mm	190 g	4 gb, 768 mb ram	133 €
htc desire vt t328t black	119.5 x 62.2 x 9.6 mm	118 g	4 gb storage, 512 mb ram	
htc desire vt t328t red	119.5 x 62.2 x 9.6 mm	118 g	4 gb storage, 512 mb ram	177 €
htc desire 630 dual sim d630n dark gray	146.9 x 70.9 x 8.3 mm (5.78 x 2.79 x 0.33 in)	140 g (4.94 oz)	16 gb, 2 gb ram	170 €
apple iphone 7 128gb gold	138.3 x 67.1 x 7.1 mm (5.44 x 2.64 x 0.28 in)	138 g (4.87 oz)	128 gb, 2 gb ram	753 €
apple iphone se 128gb rose gold	123.8 x 58.6 x 7.6 mm (4.87 x 2.31 x 0.30 in)	113 g (3.99 oz)	128 gb, 2 gb ram	496 €
apple iphone 7 256gb rose gold	138.3 x 67.1 x 7.1 mm (5.44 x 2.64 x 0.28 in)	138 g (4.87 oz)	256 gb, 2 gb ram	818 €
apple iphone se 32gb rose gold	123.8 x 58.6 x 7.6 mm (4.87 x 2.31 x 0.30 in)	113 g (3.99 oz)	32 gb, 2 gb ram	382 €
apple iphone se 128gb gray	123.8 x 58.6 x 7.6 mm (4.87 x 2.31 x 0.30 in)	113 g (3.99 oz)	128 gb, 2 gb ram	496 €
apple iphone 6s 32gb silver	138.3 x 67.1 x 7.1 mm (5.44 x 2.64 x 0.28 in)	143 g (5.04 oz)	32 gb, 2 gb ram	610 €
apple iphone 7 128gb rose gold	138.3 x 67.1 x 7.1 mm (5.44 x 2.64 x 0.28 in)	138 g (4.87 oz)	128 gb, 2 gb ram	753 €
apple iphone 7 256gb black	138.3 x 67.1 x 7.1 mm (5.44 x 2.64 x 0.28 in)	138 g (4.87 oz)	256 gb, 2 gb ram	
apple iphone 7 32gb black	138.3 x 67.1 x 7.1 mm (5.44 x 2.64 x 0.28 in)	138 g (4.87 oz)	32 gb, 2 gb ram	658 €
apple iphone 7 128gb red	138.3 x 67.1 x 7.1 mm (5.44 x 2.64 x 0.28 in)	138 g (4.87 oz)	128 gb, 2 gb ram	764 €
apple iphone se 128gb gold	123.8 x 58.6 x 7.6 mm (4.87 x 2.31 x 0.30 in)	113 g (3.99 oz)	128 gb, 2 gb ram	496 €
apple iphone se 32gb gold	123.8 x 58.6 x 7.6 mm (4.87 x 2.31 x 0.30 in)	113 g (3.99 oz)	32 gb, 2 gb ram	382 €
apple iphone 6s 32gb gold	138.3 x 67.1 x 7.1 mm (5.44 x 2.64 x 0.28 in)	143 g (5.04 oz)	32 gb, 2 gb ram	610 €
apple iphone 5s 16gb white	123.8 x 58.6 x 7.6 mm (4.87 x 2.31 x 0.30 in)	112 g (3.95 oz)	16/32/64 gb storage, 1 gb ram ddr3	312 €
apple iphone 5s 16gb gray	123.8 x 58.6 x 7.6 mm (4.87 x 2.31 x 0.30 in)	112 g (3.95 oz)	16/32/64 gb storage, 1 gb ram ddr3	301 €
apple iphone 7 128gb black	138.3 x 67.1 x 7.1 mm (5.44 x 2.64 x 0.28 in)	138 g (4.87 oz)	128 gb, 2 gb ram	753 €
apple iphone 7 128gb silver	138.3 x 67.1 x 7.1 mm (5.44 x 2.64 x 0.28 in)	138 g (4.87 oz)	128 gb, 2 gb ram	768 €
apple iphone se 32gb silver	123.8 x 58.6 x 7.6 mm (4.87 x 2.31 x 0.30 in)	113 g (3.99 oz)	32 gb, 2 gb ram	
apple ipod shuffle 4th generation 2gb silver	29 x 31.6 x 8.7 mm	12 g	2 gb storage	88 €
apple ipod touch 6th generation 32gb blue	123 x 58.6 x 6.1 mm	88 g	32 gb storage, 1 gb ram	288 €
apple iphone se 128gb silver	123.8 x 58.6 x 7.6 mm (4.87 x 2.31 x 0.30 in)	113 g (3.99 oz)	128 gb, 2 gb ram	
apple ipod touch 6th generation 32gb silver	123 x 58.6 x 6.1 mm	88 g	32 gb storage, 1 gb ram	288 €

Figure 5.3: Example of converted and preprocessed CSV file (cropped).

# 6. Evaluating Machine Learning Classifier Algorithms

This chapter has several main aims. At first, it introduces the reader to the field of supervised machine learning algorithms - especially the group of classification algorithms. Then, it gives an overview of algorithm chosen for evaluation by the Author of this thesis. After chosen algorithms are introduced, actual process of experiments and evaluation of these classifier algorithms is described, together with the *ml-experiments* application implemented by the author of this thesis. The chapter is concluded with the determination of the classification algorithm and its configuration which proved to have the best results during evaluation.

## 6.1 Machine Learning & Classification

Machine learning is a technique of programming which allows computers to solve certain problem, however, instead of providing the computer with an explicit algorithm with exact step by step guide which the program follows, the machine learning program should infer what to do based on either training data or past experience [Alpaydin, 2010].

Machine learning techniques are distributed into several categories, such as - *supervised learning*, *unsupervised learning* and *reinforcement learning* [Kotsiantis, 2007]. For the purposes of this thesis, it is necessary to at least briefly introduce the *supervised learning* category.

According to Kotsiantis [2007], machine learning is called *supervised* when the algorithm at first receives a sample (usually called *training set* of data) of data with already known results. This way, the algorithm can analyze the training samples and *learn to predict* the results for *unknown* values based on their features.

A typical group of *supervised learning* algorithms is a group of algorithms which are able to determine the *class* (or a type) of — to *classify* — an entity represented using its features. These algorithms are called *classifiers*.

Mentioned *features* are defined by Kotsiantis [2007] as values which can be *continuous*, *categorical* or *binary*. When any object instance from a dataset is going to be used by a machine learning algorithm, it needs to be at first converted into a representation of a set of features. Each instance needs to be represented using *the same* set of features as others. The Definition 9 explains the intuitive difference between *binary* and *multi-class* classifiers.

As an example of classifier and set of features Kotsiantis [2007] introduces an algorithm for classification of images containing hand-written digits to actual numbers 0 – 9 (ten classes) which are understandable for computers. An (*object*) *instance* is a single grey-scale image file, its features are the numeric values of the individual pixels (e.g. value 0-255 of the shade of grey of the pixel) of the image.

**Definition 9** (Binary Classifier and Multi-Class Classifier). Binary classifier *is a classifier which determines whether the input instance belongs to one of two classes, usually TRUE or FALSE.*

Multi-Class classifier is a classifier which allows to classify the instance into more than two classes.

## 6.2 Evaluated Classification Algorithms

For the classification algorithm evaluation phase, the *Weka 3*<sup>1</sup> framework was used. This framework contains a bunch of already pre-implemented machine learning algorithms, and techniques for their evaluation. For the purpose of this thesis, following classifier algorithms have been evaluated:

- *Decision Tree (J48)*<sup>2</sup> - Wekas J48 algorithm is an implementation of decision tree building *C4.5* algorithm<sup>3</sup>, developed by *Ross Quinlan*. The algorithm was listed as one of *Top 10* data mining algorithms by Wu et al. [2008]. From the given *training set* of data, the algorithm infers a decision tree according to the values of the instance features. The *unknown* instance is then classified by traversing the decision tree and selecting the next node according to the rules for the features. The list of a tree contains the predicted class for the instance. An example of decision tree can be found in Figure 6.1.

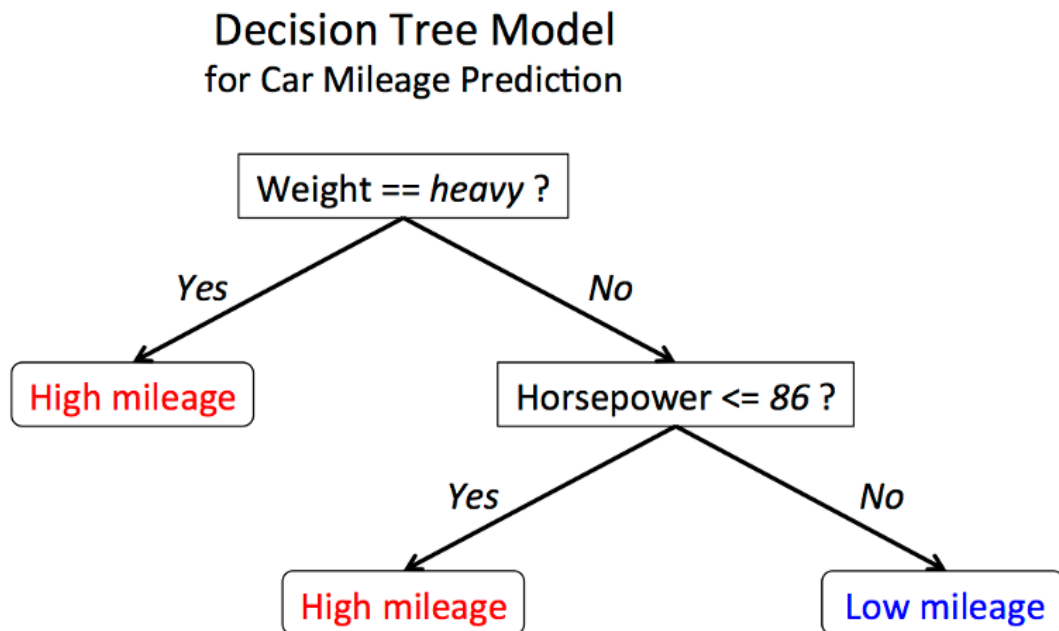


Figure 6.1: Example of a decision tree. Source: Amde and Bradley [2014]

- *Decision Stump*<sup>4</sup> - This is a classifier implementation which generates the rules in form of *decision stump*-s. Iba and Langley [1992] defines *Decision Stump* as *One-Level Decision Tree* — a decision tree having just a *root*

<sup>1</sup><http://www.cs.waikato.ac.nz/ml/weka/index.html>

<sup>2</sup><http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/J48.html>

<sup>3</sup>C4.5 is an improved version of Quinlan's ID3 (Iterative Dichotomiser 3) algorithm.

<sup>4</sup><http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/DecisionStump.html>

*node* representing a rule on single instance feature, and *leaves* representing *classes* according to the rule evaluations. That means, a decision stump decides on the class just by evaluating a single feature of the instances.

- *PART Decision List (PART)*<sup>5</sup> - This algorithm, described in Frank and Witten [1998], uses the *separate-and-conquer* paradigm for recursive building of decision list of rules for the training dataset instances. *Decision list*, as defined by Rivest [1987], can be viewed as a sequence of evaluation of rules in form described in Code 7.1. Once the instance fulfills a rule, it is classified with a defined class. The algorithm builds a partial decision tree (using *C4.5* algorithm) in each iteration, and makes a new rule from the “*strongest*” leaf.

```

if (RULE1) then
    return CLASS1
else if (RULE2) then
    return CLASS2
...
else
    return CLASSX

```

Code 7.1: Pseudo-code of PART Decision List rules. Source: Rivest [1987].

- *Decision Table*<sup>6</sup> - This is an implementation of classifier which builds a set of rules represented as rule-mappings to a majority class. This representation is called *Decision Table Majority (DTM)* and — as Kohavi [1995] states — contains two components: a *table schema*, which represents set of features, and *table body*, which contains labeled instances from *training set* of data, represented by the features described in the table schema. For a given *unknown* instance, the classifier algorithm searches the table for the exact match of features. If there is a match (or matches), the majority class of the matched instances is used as the class of the instance. Otherwise, if no instance matches, the *majority class of the table* is used.
- *Random Forest*<sup>7</sup> - Random forests are described by Breiman [2001] as a classifier which relies on growing a large number of decision trees — a *forest*. Each of the trees has a single vote, and they vote together for the most popular class for a given *unknown* instance. The trees are grown with an assistance of generated random vectors. The random vectors for all of the trees should be *independent identically distributed*.
- *Naive Bayes*<sup>8</sup> - John and Langley [1995] states that *Naive Bayesian Classifier* is a probabilistic classifier which is a simplified version of Bayesian

<sup>5</sup><http://weka.sourceforge.net/doc.dev/weka/classifiers/rules/PART.html>

<sup>6</sup><http://weka.sourceforge.net/doc.dev/weka/classifiers/rules/DecisionTable.html>

<sup>7</sup><http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/RandomForest.html>

<sup>8</sup><http://weka.sourceforge.net/doc.dev/weka/classifiers/bayes/NaiveBayes.html>

Network. It assumes that attributes are independent — thus the name *naive* — which uses the Bayes’s theorem<sup>9</sup> to determine the probability of the instance belonging to a given class based on the vector of observed values and then predict the most probable class. Naive Bayes algorithm was also listed as one of *Top 10* data mining algorithms by Wu et al. [2008].

- *Support Vector Machine (SMO)*<sup>10</sup> - An implementation of *Sequential Minimal Optimization* algorithm which can be used for training a *Support Vector Machine Classifier*. A *Support Vector Machine Classifier* is, according to Osuna et al. [1997], a classifier which tries to find the most optimal hyperplane which would split the points — called *supported vectors* (which represent data features) — to discrete classes, with maximum margin between the separated classes. More information about the SMO algorithm itself can be found in Platt [1998]. SVM was also listed as one of *Top 10* data mining algorithms by Wu et al. [2008]
- *FeedForward Neural Network (Multilayer Perceptron)*<sup>11</sup> This algorithm represents an implementation of *Neural Network*, namely *FeedForward Neural Network*. A *Neural Network* is a network of interconnected neurons, also called *Perceptrons* (Kotsiantis [2007]). A perceptron is a representation of a *step* or *sigmoid* function which takes  $n$  inputs and produces one output. Each input of a perceptron has a *weight* assigned. This weight determines which inputs are more “important” for given perceptrons. The network forms a graph, where perceptrons are represented as nodes, and edges between nodes represent the connection between the output of first perceptron and input of the second perceptron. An example of a neural network structure can be seen in Figure 6.2. The network can consist of *single* or *multiple* layers of perceptrons. For a classifier network, the *input layer* contains one perceptron for each instance feature, and *output layer* contains one perceptron for each possible class. A layer which is not *input* or *output* is called *hidden layer*. Zell [1997] defines that neural network is considered as *FeedForward*, when there is no cycle within connections between *neurons*.

## 6.3 Classifier Evaluation

In order to proceed with actual evaluation of the classification algorithms, it is necessary to at first define some concepts and evaluation metrics, knowledge about whose is required to understand the evaluation methodology. The actual methodology follows after these definitions.

An important concepts in classification algorithm evaluation are *True Positive*, *False Positive*, *True Negative* and *False Negative*. These concepts are used by the evaluation metrics. Definition of those concepts can be found in Definition 10. Definitions are based on the definitions in Witten and Eibe [2005].

---

<sup>9</sup>Bayes’s Theorem: <https://www.britannica.com/topic/Bayess-theorem>

<sup>10</sup>Sequential Minimal Optimization: <http://weka.sourceforge.net/doc.dev/weka/classifiers/functions/SMO.html>

<sup>11</sup><http://weka.sourceforge.net/doc.dev/weka/classifiers/functions/MultilayerPerceptron.html>

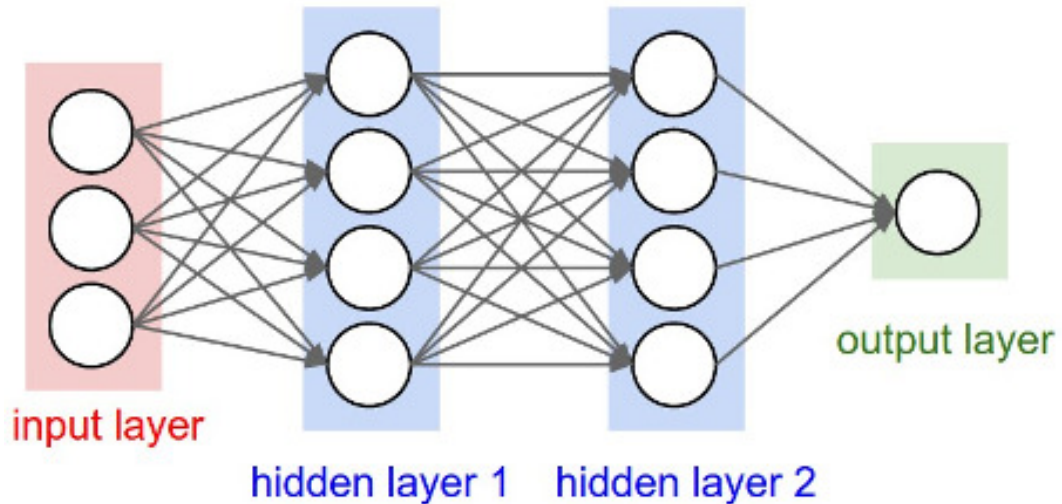


Figure 6.2: Example schema of a neural network. Source: <http://cs231n.github.io/neural-networks-1/>

**Definition 10** (True/False Positive/Negative). For binary classifier define:

- True Positive (TP) - When instance actually belongs to class *TRUE*, and was correctly classified as *TRUE*.
- True Negative (TN) - When instance actually belongs to class *FALSE*, and was correctly classified as *FALSE*.
- False Positive (FP) - When instance actually belongs to class *FALSE*, but was incorrectly classified as *TRUE*.
- False Negative (FN) - When instance actually belongs to class *TRUE*, but was incorrectly classified as *FALSE*.

For multi-class classifier define:

- True Positive for class A ( $TP_A$ ) - When instance actually belongs to class *A*, and was correctly classified as *A*.
- True Negative for class A ( $TN_A$ ) - When instance actually does not belong to class *A*, and was correctly classified as not belonging to *A*.
- False Positive for class A ( $FP_A$ ) - When instance actually belongs to different class than *A*, but was incorrectly classified as *A*.
- False Negative for class A ( $FN_A$ ) - When instance actually belongs to class *A*, but was incorrectly classified as belonging to different class.

These *True/False Positive/Negative* concepts are used to express important metrics used for the classifier algorithm evaluation - such as (*weighted*) *precision*, (*weighted*) *recall*. These metrics are defined in Definitions 11 and 12. Definitions are based on the definitions in Witten and Eibe [2005].



**Definition 11** (Precision and Weighted Precision). *For the binary classifier, define precision as:*

$$\text{Precision} = \frac{TP}{TP + FP}$$

*Thus, precision determines the ratio of correctly classified instances, among all classified instances.*

*For multi-class classifiers, it is possible to either calculate precision for each class separately ( $P_i$  for  $i$ -th class), or to calculate the Weighted (average) Precision ( $P_w$ ) across all of the classes. Assume there is a total of  $n$  classes,  $C_i$  is the number of instances of  $i$ -th class and  $T$  is the total number of instances in dataset, then:*

$$P_w = \frac{\sum_{i=1}^n (P_i \cdot C_i)}{T}$$

**Definition 12** (Recall And Weighted Recall). *For the binary classifier, define recall as:*

$$\text{Recall} = \frac{TP}{TP + FN}$$

*Thus, recall determines the ratio of actually correctly classified instances, from all instances that should have been classified with given class.*

*For multi-class classifiers, it is possible to either calculate recall for each class separately ( $R_i$  for  $i$ -th class), or to calculate the Weighted (average) Recall ( $R_w$ ) across all of the classes. Assume there is a total of  $n$  classes,  $C_i$  is the number of instances of  $i$ -th class and  $T$  is the total number of instances in dataset, then:*

$$R_w = \frac{\sum_{i=1}^n (R_i \cdot C_i)}{T}$$

Using *precision* and *recall* it is possible to introduce more complicated metric, called (*weighted*) *F-measure* which — as can be seen in Definition 13 — is dependant on them.

**Definition 13** (F-measure and Weighted F-measure). *For binary classifiers define F-measure as a harmonic mean of precision and recall:*

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

*For multi-class classifiers, it is possible to either calculate the F-measure for each class separately ( $F_i$  for  $i$ -th class), or to calculate the Weighted (average) F-measure ( $F_w$ ) across all of the classes. Assume there is a total of  $n$  classes,  $C_i$  is the number of instances of  $i$ -th class and  $T$  is the total number of instances in dataset, then:*

$$F_w = \frac{\sum_{i=1}^n (F_i \cdot C_i)}{T}$$

Witten and Eibe [2005] also introduces a metric, called *Error Rate* which, as they claim, is a natural way to express the performance of a classifier. Definition of error rate can be found in Definition 14.

**Definition 14** (Error Rate). *Each result of classification is either success (True Positive, True Negative) or error (False Positive, False Negative). The Error Rate is represented as*

$$er = \frac{\#errors}{\#instances}$$

### 6.3.1 The ML-Experiments Framework

For the purpose of experiments with the classification algorithms, the author of this thesis implemented a framework called *ml-experiments*. It is a set of proof-of-concept Java applications which utilize implementation of classifier algorithms from the Weka framework<sup>12</sup> for the purposes of experiments and evaluations. The application runs from command line, and has no graphical user interface. The framework contains a set of applications, which are actually Java classes, that were added gradually to the framework, as they were needed while doing the actual algorithm evaluation by the author of this thesis:

- *sk.kadlecek.mle.SingleDataset* - An application which takes the dataset provided in its input and performs a *10-fold cross-validation*<sup>13</sup> of the specified classifier (with its default settings).
- *sk.kadlecek.mle.SeparateDatasets* - An application which as an input takes both training and testing datasets, and performs the evaluation of the specified classifier (with its default settings). The N-fold cross-validation Since the training and testing datasets are same
- *sk.kadlecek.mle.Evaluation* - An application which as an input takes both training and testing datasets, and performs the evaluation (using same method as the *Separate Datasets* application) of the specified classifier with various configurations of the algorithm. The aim of this application is to find the best algorithm configuration for given input data. The parameter combinations are hard-coded into the application.
- *sk.kadlecek.mle.CrossValidationEvaluation* - An application which is very similar to the *Evaluation* application, however instead of taking both training and testing datasets on its input, only one dataset is expected as input. This dataset is then used to perform N-fold cross-validation evaluation of the specified algorithm.
- *sk.kadlecek.mle.Prediction* - An application which trains the specified classification algorithm using training set given as input, and then classifies the instances of the testing dataset from input. The results are stored in a specified output file. However, just Weka index of classified class is assigned to the result, it is not being translated to the actual class label.

---

<sup>12</sup>Introduced in Section 6.2 of this chapter.

<sup>13</sup>See the Definition 15 below.

- *sk.kadlecek.mle.InteractivePrediction* - An application which is similar to the *Prediction* application, but after training the classification algorithm using training set given as input, it then exposes its shell to the user. The user can interactively type in phrases (one per line), which then will be classified and the resulting class label is printed back to the user.

The main application, which was used to perform all experiments described in this chapter, is the *CrossValidationEvaluation* application.

Descriptions of several applications from the framework used the term "N-fold Cross-Validation". This term is explained in detail in the Definition 15. The definition is based on the definition in [Witten and Eibe, 2005].

**Definition 15** ((N-fold) Cross Validation). *A method of evaluation of the classification algorithm, where the dataset is split into two: training and testing dataset. The classifier is then built by learning instances from the training dataset. After the classifier has learned all instances from training dataset, instances from the testing dataset are used to evaluate the classifier.*

*In the N-fold Cross Validation, the validation is run N times. The dataset is also randomly split into training/testing datasets N times, so in each "fold" the training/testing datasets are different, and the results are averaged.*

*For the most accurate results, it is recommended to use 10-fold cross validation, and run the cross validation 10 times (that means 100 algorithm evaluations in total).*

Each of the applications introduced above is expecting *one* or *two* dataset files on input. These dataset files are expected to be in the *ARFF*<sup>14</sup> format supported by Weka. The framework contains a preprocessing application "sk.kadlecek.mle.preprocessing.Preprocessor" which can be used for conversion of CSV files to ARFF format. However, the application is more like a proof-of-concept, as there are lots of things hardwired in the code, including input/output file paths and features to be detected, and was used by the author of this thesis from the IDE with manual adjustments of the parameters according to the experiment needs.

The conversion process consists of following steps:

1. *Class Extraction* - As the input file is in CSV format, names of all classes are present in the columns in the header row. The conversion algorithm collects them into a *set*.
2. *Feature Calculation* - In this phase, all cell values (except the ones in header row) in the input file were processed, and for each value, set of features was determined. During this phase, all redundant cell values are detected and removed (so they are not propagated to the resulting dataset). Empty cells are removed as well.
3. *Dataset File Creation* - In this phase, all of the features collected for non-redundant cell values were printed into ARFF file in the form:

F1, F2, . . . , Fn, CLZ
-------------------------

<sup>14</sup>Attribute-Relation File Format: <https://www.cs.waikato.ac.nz/ml/weka/arff.html>

Where  $F_i$  denotes a value of  $i$ -th feature and  $CLZ$  denotes a label of a class the instance belongs to.

The resulting dataset file clearly contains a classification for each of the input instance represented by features. This is required in order to use the dataset with Weka framework.

Exhaustive documentation of the application, together with guide on how to compile the application from source codes, can be found in the “/ml-experiments/documentation/ml-experiments-doc.pdf” file on the attached CD.

Source codes of the *ml-experiments* application can be found in the GitHub repository <sup>15</sup> of the author of this thesis or in the “/ml-experiments/src” folder on the attached CD. Pre-built *jar package* can be found in the “/ml-experiments/target/ml-experiments.jar” file on the attached CD.

### 6.3.2 Experiments

Using the *ml-experiments* framework described in the beginning of this section, the author of this thesis performed several experiments with the classification algorithms. These experiments can be grouped into two groups: *evaluation of features*, which should represent the input data in order to achieve the best results, and, *evaluation of algorithm configurations* in order to find the best performing algorithm configuration.

The classification works as follows: the framework receives input data in form of ARFF files, where each instance represents the value of a single table cell, represented by the features calculated from that value, and a class which is the value of the header column, to which the cell belongs in the table. The dataset file given at the input of the application is split into the training and testing datasets. The training contains a known class for each of its instances, the classes of the instances from the testing dataset are considered as unknown.

The algorithm at first build its classification model based on the already known instances from the training dataset, and then it “guesses” (estimates) the class for the instances in the training dataset based on this trained model.

Since the application knows the correct classes of the instances from the testing dataset (as they were provided in input, and were removed from the dataset at the moment of creating testing dataset from a part of input dataset), it can, based on the original and classified values then compute evaluation metrics for the algorithm.

Results of these experiments were evaluated based on following metrics:

- *Average Error Rate*
- *Average F-Measure*
- *Average Precision*
- *Average Recall*

---

<sup>15</sup><https://github.com/rkadlec/ml-experiments>

## Input Data Preparation

In order to perform the experiments, some preprocessing of the input data was required. The main issue with the data was that in some of the e-shops the structure of the properties is not same across all of the products, and in some cases, also some typos were present in the column headers. That means, in the CSV files there were multiple columns representing the same property with each of them having a portion of the values. In these cases, the author of this thesis merged the columns together manually.

The author of this thesis also altered the headers of the columns of input CSV files to simplify them. This means that instead of a column header such as “TechnicalSpecifications.OperatingSystem”, simpler “OPERATING\_SYSTEM” header was used.

Also, all input files were simplified in a way that they contained the same columns (properties) — if available. Columns with just a small amount of non-empty cells were removed.

For the purpose of classifier algorithm evaluation, the updated CSV documents were merged into one dataset file containing information about all products from all e-shops. This complete dataset was then used as an input for experiments.

The input data for can be found in the “/ml-experiments/resources/experimentX/input” folders (where X is the number of the experiment) on the attached CD.

The output data from experiments can be found in the “/ml-experiments/resources/experimentX/output” folders (where X is the number of the experiment) on the attached CD.

## Features

Since the author of this thesis was not able to find any existing feature sets relevant to the problem of classification of string values as specific as properties of the products during the research of existing works, a customer set of features was proposed. These features can be found in Table 6.1.

<b>Numeric Features</b>	<b>Boolean Features</b>
Total Number of Words	Is the String Numeric Value
Total Number of Characters	Is the String Integral Numeric Value
Total Number of Letters	Is the String Decimal Numeric Value
Total Number of Digits	Is the String Prefixed Number (e.g. “\$100”)
Total Number of Whitespace Characters	Is the String Postfixed Number (e.g. “2 GB”)
Total Number of Other Characters	

Table 6.1: Proposed Feature Set.

In order to evaluate, if this set of properties behaves well on the given input data, experiments 1 - 15 were performed. All of these experiments run all of the classifiers on the the same input dataset, with different combinations of extracted

features. Experiment 1 starts with single numeric feature (*Total Number of Words in String*), and Experiments 2 - 11 then add more features — one at a time — until the full feature set described in Table 6.1 is reached. Experiments 12 - 15 then remove several chosen features from the full feature set in order to evaluate that the features are not redundant.

Each feature set was evaluated using the 10-fold cross validation method which was run 10 times for each algorithm except the *MultilayerPerceptron* algorithm. For this algorithm, the 10-fold cross validation method was run just once, because of the time complexity of the algorithm. This complexity is illustrated in the Table 6.2 which shows the average times of a single 10-fold cross validation run for all algorithms in the Experiment 13 on the same PC. It is clearly visible that the *MultilayerPerceptron* has by the order of magnitude higher time complexity compared to other algorithms.

Algorithm	Average Time (ms)
Decision Tree	758,10
Decision Stump	99,70
Decision List	2693,60
Decision Table	7953,50
Naive Bayes	630,10
Random Forrest	8324,20
Support Vector Machine	15801,40
Multilayer Perceptron	498092,00

Table 6.2: Average times of single 10-fold cross validation (= 10 validations) run of classification algorithms in Experiment 13 on the same PC.

The experiments were performed using the *CrossValidationEvaluation* application from the *ml-experiments* framework. The Code 6.1 shows an example of the command to run the Experiment 13 with the NaiveBayes algorithm:

```
$ java -jar "target/ml-experiments.jar" \
    "sk.kadlecek.mle.CrossValidationEvaluation" \
    -a NaiveBayes -e -f 10 -r 10 \
    -d "resources/experiment13/input.arff"
```

Code 6.1: Example shell command to start the Feature experiment 13.

In almost all experiments, the *RandomForest* classifier proved as the best performing one, except the Experiment 2, where the *J48 Decision Tree* performed slightly (by 0.8%) better.

The evaluation of experiments also shown that the best performing feature set is the feature set evaluated in Experiment 13, features contained in which are shown in Table 6.3. That means, the *Is the String Integral Numeric Value* feature from the originally proposed feature set is *redundant* for classifiers *Decision Tree*, *Decision Stump*, *Decision Table* and *Random Forest* which achieved the same

results as in Experiment 11 with the complete proposed feature set. The property also made the score slightly *worse* for *Decision List*, *Support Vector Machine* and *Multilayer Perceptron* classifiers. However, the score *Naive Bayes* classifier was slightly improved (by 0.08%).

The results of the evaluation of the feature set 13 dataset can be found in Table 6.4.

<b>Numeric Features</b>	<b>Boolean Features</b>
Total Number of Words	Is the String Numeric Value
Total Number of Characters	Is the String Decimal Numeric Value
Total Number of Letters	Is the String Prefixed Number
Total Number of Digits	Is the String Postfixed Number
Total Number of Whitespace Characters	
Total Number of Other Characters	

Table 6.3: Experiment 13: Used Feature Set.

Since for most of the classifiers the feature set 13 behaved better, or at least the same as the full proposed feature set, the author of this thesis decided to use the feature set 13 as the feature set for the next set of algorithm evaluation experiments.

<b>Algorithm</b>	<b>Average Error Rate (%)</b>	<b>Average Precision (%)</b>	<b>Average Recall (%)</b>	<b>Average F-measure</b>
Decision Tree	11,76	87,65	88,24	87,78
Decision Stump	50,32	25,42	49,68	33,40
Decision List	11,67	87,80	88,33	87,93
Decision Table	19,57	84,14	80,43	80,45
Naive Bayes	34,12	68,25	65,88	65,90
Random Forrest	9,45	90,06	90,55	90,18
Support Vector Machine	39,05	51,19	60,95	52,96
Multilayer Perceptron	18,88	79,91	81,12	79,88

Table 6.4: Experiment 13: Results.

The detailed description of features used in particular experiments, together with the results of the Experiments 1 - 15 can be found in the Attachment C.

### **Evaluation of Algorithm Configurations**

After the data features were defined, various algorithm configuration needed to be evaluated. The purpose of this configuration evaluation is to detect the best behaving configuration of each algorithm on given input data, in order to be able to select the best algorithm to be used in the proposed Odalic algorithm.

All of the algorithm configurations were evaluated using the 10-fold cross validation method which was run 10 times for each algorithm, except the *Multilay-*

*erPerceptron* algorithm. For this algorithm, the 10-fold cross validation method was run just once, because of the time complexity of the algorithm.

The experiments were performed using the *CrossValidationEvaluation* application from the *ml-experiments* framework. The Code 6.2 shows an example of the command to run the evaluation of the NaiveBayes algorithm configurations (the actual configuration options to evaluate are hard-coded in the application):

```
$ java -jar "target/ml-experiments.jar" \
  "sk.kadlecek.mle.CrossValidationEvaluation" \
  -a NaiveBayes -f 10 -r 10 \
  -d "resources/experiment16/input.arff"
```

Code 6.2: Example shell command to start the CrossValidation evaluation of the Naive Bayes algorithm.

The detailed description of evaluated algorithm properties and all of their configurations that were subjected to the evaluation can be found in the Attachment D, together with the best performing configuration of each algorithm.

The detailed results of this experiment for each classification algorithm can be found in form of CSV files in the “/ml-experiments/resources/experimentX/output” folder on the attached CD. The files are sorted using following expressions:

- *Average Error Rate* - Descending
- *Average F-Measure* - Ascending
- *Average Precision* - Descending
- *Average Recall* - Descending

Table 6.5 shows the actual metrics comparison result for the best algorithm configurations.

Algorithm	Average Error Rate (%)	Average Precision (%)	Average Recall (%)	Average F-measure
Decision Tree	10,73	88,90	89,27	88,99
Decision Stump	50,32	25,42	49,68	33,40
Decision List	10,92	88,78	89,08	88,84
Decision Table	17,56	84,51	82,44	82,78
Naive Bayes	23,81	77,08	76,19	75,94
Random Forrest	9,39	90,07	90,61	90,21
Support Vector Machine	19,42	79,49	80,58	78,37
Multilayer Perceptron	14,57	84,52	85,43	84,75

Table 6.5: Comparison of best configurations of all evaluated algorithms



### 6.3.3 Experiments Conclusion

According to the evaluation of performed experiments described in this chapter, the best classification algorithm on the product input data is the *RandomForest* classification algorithm with configuration mentioned in Table 6.6. Thus, this algorithm was chosen to be used by the proposed improved Odalic algorithm which will be described in the Chapter 7.

Parameter	Value
Batch Size	80
Max Depth	15
Number of Features	3
Number of Bagging Iterations	90
Break Ties Randomly	true

Table 6.6: Random Forest - configuration parameters with best results.

# 7. The New Odalic Algorithm

The Chapter 6 described the process of evaluation of selected machine learning classification algorithms and their various configurations, and features used by these algorithms. The best performing feature set, algorithm and its configuration were concluded at the end of the chapter. This chapter describes the improved Odalic algorithm proposed by the author of this thesis which takes the advantage of the chosen classification algorithm in order to classify the columns of the input tables and discover relations between them even in cases when the amount of existing knowledge base data is very limited, for example for product data. This chapter also describes the integration of the algorithm into the Odalic system and evaluation of the new algorithm vs the original Odalic algorithm. In the end of this chapter, a process of semantization of the product data into RDF format, using the new Odalic algorithm, is described.

## 7.1 Algorithm Description

The updated algorithm flow which now depicts also the *ML PreClassification* phase and all the other algorithm phases that are affected by the results of this new phase can be seen in Figure 7.1. Added parts are highlighted with blue color.

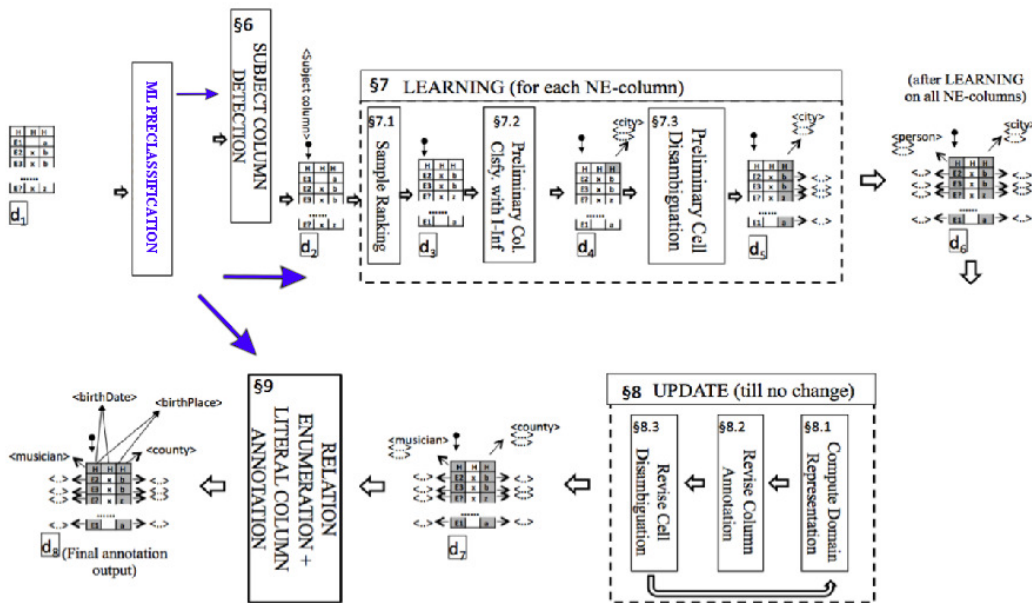


Figure 7.1: The proposed algorithm flow.

### 7.1.1 The ML PreClassification Phase

Before proceeding with actual description of the ML PreClassification phase, it is necessary to introduce terms *Ontological Class* and *ML Class* so its possible to distinguish between these two different class concepts. These terms are defined in Definitions 16 and 17.

**Definition 16** (Ontological Class). *A class defined as a part of an ontology. E.g. class with URI “<http://dbpedia.org/ontology/Software>”.*

**Definition 17** (ML Class). *A class which is returned as a result of classification of an instance by the classifier algorithm. A ML class is, in context of this thesis, represented by the header value of a column. The section 6.3.2 of chapter 6 mentions that the headers of input data files were simplified to a strings such as TITLE, CPU, MODEL and others. These simplified strings are considered as ML classes for the purposes of this thesis.*

From the high level point of view, the newly added *ML PreClassification Phase* uses the classification algorithm chosen as the best performing one based on the results of experiments in the Chapter 6 in order to classify columns of a table into an ontological class or predicate, based on the classification of values in the table cells present in given column.

The classification algorithm is trained on the training dataset<sup>1</sup>. The training is performed just once, during the component initialization, even before the actual semantization algorithm starts.

The *ML PreClassification* algorithm (or *ML PreClassifier*) iterates over each of table columns, except the columns for which the user provided manual feedback<sup>2</sup>. The algorithm classifies all of table cells in given non-feedback columns into their *ML classes*. For each column the algorithm determines a list of candidate ML classes — a list of classes which were returned by the ML classifier at least for one cell value in given column — together with their score. The score of a ML class is a number of cell values in a column classified as given ML class. Retrieved ML class candidates for a column are then sorted according to the score. If there is a classification manual feedback for a column, following two situations are possible:

1. A ML class mapping exists for feedback classification - In this case, the given ML class is assigned to the column with highest score possible (*Double.MAX\_VALUE*). This allows the algorithm to do further manipulations with this column, such as list it in the list of suggested *subject columns*, or perform relation discovery for this column.
2. A ML class mapping does not exist for feedback classification - In this case, the column is ignored - as the classification is given in the feedback, and the *ML PreClassifier* does not support manipulations with this class.

The pseudo code of this operation can be found in Code 7.1.

```
Map[Integer, String] feedbackMLClasses =  
    feedback.classifications mapped to ML class  
Set[Integer] feedbackNoMLClass =  
    feedback.classifications not mapped to ML class  
column[] columnsToClassify =
```

<sup>1</sup>Traning Dataset is described in more detail in the *Training Dataset* sub-section further in this section.

<sup>2</sup>Either set the automatic classification as invalid, or configured that the whole column should be ignored by the algorithm.

```

    allColumns - feedback.ignoreColumns - feedbackNoMLClass

List[ColumnWithSortedClasses] colsWithSortedClasses

// get list of candidate ML classes for column
for (columnToClassify in columnsToClassify):

    if (feedbackMLClasses contains columnToClassify):
        // use manually provided class
        mlClass = feedbackMLClasses.get(columnToClassify)
        colsWithSortedClasses.add(
            ColumnWithSortedClasses(
                columnToClassify,
                List(MLClassWithScore(mlClass, Double.MAX_VALUE))
            )
        )
    else:
        Map[String, MLClassWithScore] classesWithOccurrences = Map.empty

        for (cell in columnToClassify.cells):
            String mlClass = classifier.classify(cell)
            if (classesWithOccurrences contains mlClass):
                increase score of mlClass in map
            else
                add new mlClass to map with score 1
            end if
        end for
        colsWithSortedClasses.add(
            ColumnWithSortedClasses(
                columnToClassify,
                sortColumnClassesByScore(classesWithOccurrences)
            )
        )
    end if
end for

// select winning classes for columns
Map[Integer, MLClassificationWithScore] columnClassifications =
    getColumnClassifications(colsWithSortedClasses);

```

Code 7.1: Pseudo code of the column ML classification in the ML PreClassification phase.

After candidate ML classes of each column are resolved, the algorithm assigns the winning class for each column iteratively, until all columns are either assigned with a class, or they “run out” of available classes. The process works in a following steps:

1. *Building the ML class map.* In each iteration, a *current top* ML class is retrieved for each of the columns which are being processed. The current

top class is selected from the list of scored candidate classes of given column, from the classes with highest score to the classes with lowest score. If a column “runs out” of available classes, no class is assigned to it. If the column contains a valid class, the class is added to the map of type (Class, [(ColumnIndex, Score)]) which keeps the columns, to which the class can be assigned, and scores of the class in given columns.

2. *Selecting winning columns of the ML classes.* For each ML class stored in map which was built in previous step a *winning column* — the column where the class achieved highest score — is selected. The mapping of the ML class to the winning column is then saved to the result map.
3. *Updating columns to be processed.* All columns which still have no class assigned — but still have a *top class* available — will be processed again in next iteration of the algorithm.

The algorithm converges when either all of the columns are exhausted - they are either assigned with a class, or they run out of candidate classes. Each class is assigned just to (at most) one of the columns.

The pseudo code of assigning winning ML classes for columns can be found in Code 7.2.

```
List[ColumnWithSortedClasses] columnsToProcess = colsWithSortedClasses
Set[Integer] columnsToIgnore = Set.empty
Set[String] alreadyAssignedClasses = Set.empty

Map[Integer, MLClassificationWithScore] result = Map.empty;

while (columnsToProcess is not empty):

    Map[String, (ColumnIndex, Score)] classOccurrences = Map.empty

    for (columnToProcess : columnsToProcess):
        // in each iteration retrieves "next" class
        // in order from best score to worst score
        classWithScore = columnToProcess.getNextTopClass
        if (classWithScore != null)
            if (!alreadyAssignedClasses contains classWithScore):
                add class occurrence to the classOccurrencesMap
            else
                column will not be processed this iteration
            end if
        else
            // ignore column for the rest of the algorithm run
            columnsToIgnore.add(columnToProcess.columnIndex)
        end if
    end for

    List[ColumnWithSortedClasses] newColumnsToProcess = List.empty
    for (classOccurrencesEntry: classOccurrences):
        winningColumn = from the classOccurrences map select column
```

```

        with highest score of given class

    add the ML class mapping of winning column to the result

        newColumnsToProcess =
            classOccurrencesEntry.candidateColumns - winningColumn
    end for

    columnsToProcess = newColumnsToProcess - columnsToIgnore
end while

return result

```

Code 7.2: The process of assigning winning ML classes to columns.

The actual classification algorithm is wrapped inside the “uk.ac.shef.dcs.sti.core.algorithm.tmp.ml.classifier.RandomForestMLClassifier” class and its parent “MLClassifier” class located in the same package. The algorithm at first converts the given cell value into the Wekas input instance (this includes detecting the features of the value) and retrieves the list of candidate ML classes together with their *probability* scores. The algorithm then iterates over these candidate ML classes, find the one with highest score, and if the score is higher<sup>3</sup> than configured treshold<sup>4</sup>, returns that ML class. The pseudo code of this operation can be found in Code 7.3.

```

instance = detectFeaturesAndConvertToInstance(cellValue)
mlClassesWithScores = mlClassifier.distributionForInstance(instance)
mostProbableClass = findClassWithHighestScore(mlClassesWithScores)

if (mostProbableClass.score > threshold):
    return mostProbableClass.mlClass
else
    return null
end if

```

Code 7.3: Pseudo code of the ML classification of a cell value.

After each of the columns is classified with a ML class, the process of translation of *ML class* into *ontological entity* — *class* or *predicate* — takes place. This process uses the *Ontology Mapping* entity in order to retrieve ontological entities which are mapped to ML classes selected for the table columns. For each column, the algorithm consults the ontology mapping entity to determine if given ML class represents an ontological class or predicate:

- *It is a class* - The algorithm loads required class details, especially label, from the *Ontology Definition* model, and adds a new *TColumnHeaderAnnotation*<sup>5</sup> entity to the *ML PreClassification* result.

<sup>3</sup>The score is a *double value* in range 0 – 1.

<sup>4</sup>Threshold is configured in “sti.tmp.ml.confidence.threshold” configuration option of the Odalic system which is by default located in the “ml.properties” file in the configuration folder.

<sup>5</sup>Instance of “uk.ac.shef.dcs.sti.core.model.TColumnHeaderAnnotation” class, used by Odalic to represent column classes.

- *It is a predicate* - The algorithm loads required predicate details, such as predicate domain<sup>6</sup>, from the *Ontology Definition* model, and adds a new *MLPredicate*<sup>7</sup> entity to the *ML PreClassification* result.

All identified *TColumnHeaderAnnotation* and *MLPredicate* entities are encapsulated into the instance of *MLPreClassification*<sup>8</sup> class which represents the result of *ML PreClassification* phase of the algorithm.

The *Ontology Mapping* and *Ontology Definition* entities mentioned above are described in following sub sections.

The pseudo code representation of the *ML PreClassification* phase of the algorithm can be found in Code 7.4.

```

column[] columnsToClassify = allColumns - ignoreColumns
MLPreClassification result = new MLPreclassification

for (columnToClassify in columnsToClassify):
    String mlClass = columnToClassify.mlClass
    columnToClassify.mlClass = mlClass

    if (ontologyMapping.containsClassMapping(mlClass)):
        OntoEntityURI classUri = ontologyMapping.getClass(mlClass)
       Clazz clazz = ontologyDefinition.loadClass(classUri)
        result.addClassHeaderAnnotation(
            new TColumnHeaderAnnotation(clazz)
        )

    else if (ontologyMapping.containsPredicateMapping(mlClass)):
        OntoEntityURI predicateUri =
            ontologyMapping.getPredicate(mlClass)
        Set predicateDomainUris =
            ontologyDefinition.loadPredicateDomain(predicateUri)

        result.addPredicate(
            new MLPredicate(predicateUri, predicateDomainUris)
        )

    else
        log "Mapping not found for ml class"
    end if
end for

return result

```

Code 7.4: High-level pseudo code of the ML PreClassification algorithm phase.

<sup>6</sup>Domain of a predicate is basically a set of class URIs, to which this predicate can be assigned.

<sup>7</sup>Instance of “uk.ac.shef.dcs.sti.core.algorithm.tmp.ml.MLPredicate” class, used to represent the identified predicate.

<sup>8</sup>“uk.ac.shef.dcs.sti.core.algorithm.tmp.ml.config.MLPredClassification”

Odalic system extends the TableMiner<sup>+</sup> algorithm in a way that its possible to query multiple knowledge bases. Or, more precisely, the system runs the TableMiner<sup>+</sup> algorithm separately for each knowledge base configured for the task and gathers the best results from these runs. Since the *ML PreClassification* phase is not dependant on any of the knowledge bases, it runs just once, before the actual TableMiner<sup>+</sup> phases are executed.

The source code of the *ML PreClassification* phase of the algorithm can be found in the “uk.ac.shef.dcs.sti.core.algorithm.tmp.ml.DefaultMLPreClassifier” class in the “odalic/sti” source codes on attached CD.

## Training Dataset

The training dataset is a set of instances that is used to give the classification algorithm some kind of insight on how the feature values affect the class. The training dataset is loaded from a training dataset file which is provided by the user of the Odalic system at the time of creating a new task. The expected format of the training dataset is a CSV file (so the training dataset can have similar format as the actual *unknown* input data which will be classified. Since the Weka classifier accepts the datasets only in the ARFF format, the CSV file needs to be at first converted to ARFF in a pre-processing stage. This is done automatically by the new Odalic algorithm. The principle of this conversion is described in the Section 6.3.1 of Chapter 6.

## Ontology Mapping

The ontology mapping entity — implemented by the “uk.ac.shef.dcs.sti.core.algorithm.tmp.ml.config.MLOntologyMapping” class — is responsible for “translation” of *ML classes*, assigned to table columns by the classification algorithm to the actual entities (classes, predicates) defined in an ontology.

The ontology mapping is loaded from a JSON file configured in the “sti.tmp.ml.ontology.mapping.file.path” configuration property of the Odalic system which is by default located in the “ml.properties” file in the configuration folder, and contains 2 groups of *ML CLASS* to *Entity URI* mappings: *class mappings* and *predicate mappings*. The exact structure of the file is described in the Code 7.5.

```
{
  "classMappings": [
    {
      "mlClass": "TITLE",
      "uri": "http://dbpedia.org/ontology/Device"
    },
    {
      "mlClass": "OS",
      "uri": "http://dbpedia.org/ontology/Software"
    },
    ...
  ],
}
```



```

"predicateMappings": [
  {
    "mlClass": "CPU",
    "uri": "http://dbpedia.org/ontology/distanceLaps"
  },
  {
    "mlClass": "BRAND",
    "uri": "http://dbpedia.org/property/manufacturer"
  },
  ...
]
}

```

Code 7.5: Example of the Ontology Mapping file which illustrates its format.

An example of complete ontology mapping file can be found in the “odalic/sti/resources/ml/mapping.json” location on the attached CD.

## Ontology Definition Model

The Ontology Definition Model — implemented by the “uk.ac.shef.dcs.sti.core.algorithm.tmp.ml.config.MLOntologyDefinition” class — is a RDF model which contains RDF triples representing the definition of chosen ontologies. It is important, because it contains more detailed information about the ontological entities mapped to the ML classes, e.g. *domains* or *ranges*<sup>9</sup> of predicates.

The ontology definition model supports multiple ontologies and is loaded from RDF files stored in a folder, path to which is configured in the “sti.tmp.ml.ontology.definition.folder” configuration property of the Odalic system which is by default located in the “ml.properties” file in the Odalic configuration folder. The brief example of ontology definition taken from the DBpedia ontology definition file in the *N-Triples* format can be found in the Code 7.6.

So far, *RDF/XML*, *Turtle* and *N-Triples* formats are supported for the Ontology Definition files. The file reader determines the format of the file using the file extension: *.rdf*, *.xml*, *.owl* for *RDF/XML*, *.ttl* for *Turtle* and *.nt* for *N-Triples*.

Since the heart of the Ontology Definition Model entity is a model consisting of RDF triples<sup>10</sup>, the entity also contains implementation of methods for querying of the model which are used in *ML PreClassification* and *Relation Discovery* phases.

```

<http://dbpedia.org/ontology/>
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
    <http://www.w3.org/2002/07/owl#Ontology> .

<http://dbpedia.org/ontology/>
  <http://purl.org/vocab/vann/preferredNamespacePrefix>
    "dbo" .

```

<sup>9</sup>Range of a predicate defines “legal” values which this predicate can obtain.

<sup>10</sup>Implementation is using the rdf4j library: <http://rdf4j.org/>.

```

<http://dbpedia.org/ontology/Software>
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
    <http://www.w3.org/2002/07/owl#Class> .

<http://dbpedia.org/ontology/Software>
  <http://www.w3.org/2000/01/rdf-schema#label>
    "software"@en .

<http://dbpedia.org/ontology/operatingSystem>
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
    <http://www.w3.org/2002/07/owl#ObjectProperty> .

<http://dbpedia.org/ontology/operatingSystem>
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
    <http://www.w3.org/1999/02/22-rdf-syntax-ns#Property> .

<http://dbpedia.org/ontology/operatingSystem>
  <http://www.w3.org/2000/01/rdf-schema#label>
    "operating system"@en .

<http://dbpedia.org/ontology/operatingSystem>
  <http://www.w3.org/2000/01/rdf-schema#domain>
    <http://dbpedia.org/ontology/Software> .

```

Code 7.6: An example extract of the Ontology Definition file. Source: DBpedia<sup>11</sup>

In order for the algorithm to run properly, it is recommended to provide definitions of ontologies with defined domains and ranges of the predicates.

The complete ontology definition file of DBpedia ontology can be found in the “odalic/sti/resources/ml/ontology/dbpedia\_2016-10.nt” location on the attached CD.

## 7.1.2 Subject Column Detection

The original Odalic algorithm contains the Subject Column Detection method as introduced by the TableMiner<sup>+</sup> algorithm, but adds the option to *specify* (or *override*) the subject columns manually by the user - In this case, the subject columns provided by user are used instead of the one *detected* by the algorithm. If there are no subject columns specified by the user, the TableMiner<sup>+</sup> subject column detection routine is used to retrieve subject column candidate. In time of writing of this thesis the algorithm proposes just one subject column candidate.

The new algorithm proposed by this thesis improves the Subject Column Detection in following way:

1. In case the subject columns are provided by the user, these provided subject columns are used.

<sup>11</sup>[http://downloads.dbpedia.org/2016-10/dbpedia\\_2016-10.nt](http://downloads.dbpedia.org/2016-10/dbpedia_2016-10.nt)

2. If there are no subject columns provided by the user, the result of the *ML PreClassification* phase of the algorithm is queried in order to retrieve subject column candidates. The chosen subject columns are passed to the algorithm in the same way as user-provided subject columns would be - using the *Constraints* instance.
3. If no subject column is provided by the user or no candidate proposed by *ML PreClassification*, the standard TableMiner<sup>+</sup> subject column detection algorithm is used.

### ML PreClassification - Choosing Subject Columns

The detection of subject columns on the result of *ML PreClassification* iterates through all columns that were classified as *ontological class* columns and for each of these columns the algorithm tries to find a predicate column whose predicate can be used with given *ontological class* (the predicate contains given class in its domain). If at least one of such predicate columns is found, the class column is considered as one of subject columns.

The pseudo code of the Subject Column Detection algorithm within the result of *ML PreClassification* can be found in Code 7.7.

```

Set[Column] subjectColumns = Set.empty

for (identifiedClassColumn in identifiedClassColumns):

    const clazz = identifiedClassColumn.class
    if (exists identifiedPredicate which has clazz in its domain):
        subjectColumns += identifiedClassColumn
    end if
end for

return subjectColumns

```

Code 7.7: Pseudo code of the ML PreClassification Subject Column Detection.

### 7.1.3 Column Classification & Entity Disambiguation

The column classification using the ML classification algorithm is performed in the *ML PreClassification* phase of the algorithm. Since the motivation behind this algorithm improvements is that the presence of product data in knowledge bases is limited, the standard knowledge-base-based classification is not performed. The ML classification of columns is provided to the algorithm in the form of Constraints (as if it was a manual feedback from the user).

Since the column classification can be also provided manually by the user in form of Constraints, the classification made by the ML system needs to be merged with these user constraints properly. The pseudo code of this merging process can be found in Code 7.8. The manual classification constraints have preference over the classifications made by the ML PreClassification phase of the algorithm.

```

Set mergedClassifications = Set.empty

for (column in tableColumns):
  if (userClassifications.exists(column)):
    mergedClassifications.add(userClassifications.get(column))
  else if (mlClassifications.exists(column)):
    mergedClassifications.add(mlClassifications.get(column))
  end if
end for

return mergedClassifications

```

Code 7.8: Pseudo code of the User and ML PreClassification column classifications merge process.

Even though the standard knowledge-base-based column classification is not taken into account in case of successful ML classification, the *Entity Disambiguation* runs as usual, in order to try to identify at least a minimum of entities from the table cell values which could possibly be in the knowledge base.

#### 7.1.4 Relation Discovery

The Relation Discovery phase has been modified in a way that it at first tries to utilize the information discovered in the *ML PreClassification* phase in order to discover relations between subject column(s) and other table columns. However, this can be done only if the subject column, which is being processed, contains a valid class header annotation (information about the Ontological class of the column). If this condition is satisfied, the relation discovery uses following steps to try discover relations between subject column and each object column:

1. *Object column is an ontological class* - Try to find a predicate which has the class of subject column in its domain, and the class of the object column in its range<sup>12</sup>, and which also was not assigned to any other (subjectColumn, objectColumn) pair. If such predicate is found, relation annotation between columns is created. If not, the column is added to the list of columns, on which the classic relation discovery will be performed later.
2. *Object column was classified by ML classifier as an ontological predicate* - The algorithm checks, if the class of the subject column complies to the domain of the object column predicate. Also, a check whether the given predicate was not yet assigned by the ML Relation Discovery to a different (subjectColumn, objectColumn) pair is performed. If these conditions are satisfied, relation annotation between columns is created. If not, the column is added to the list of columns, on which the classic relation discovery will be performed later.

---

<sup>12</sup>This process is described in the “Finding Predicate Between Subject and Object Entities” subsection below

In case that the object column is neither an ontological class, nor a predicate, it is added to the list of columns, on which the classic relation discovery will be performed.

The algorithm tries to prevent discovering invalid duplicate relations by checking if the discovered property is not yet assigned to different (subjectColumn, objectColumn) pair. These invalid duplicate relations are discovered often in case, when there are multiple subjectColumns in the table, and the domain of the predicate is defined too loosely, or not restricted at all. In this case, the relation would be made between the object column, which represents the value of given predicate, and multiple (or even all) subject columns. This is most cases — if not in all cases — not correct.

After all of the object columns are processed (or in case that the subject column does not contain a header annotation), the classic relation discovery is performed between the subject column and all object columns, for whose no relation has been found.

The pseudo code of the proposed Relation Discovery phase can be found in the Code 7.9.

```
otherColumns = columns for whose the relation between
  them and the subjectColumn need to be discovered
for (subjectColumn in subjectColumns)
  if (subjectColumn has ML classification)
    for (objectColumn in otherColumns)
      if (objectColumn.hasClassHeaderAnnotation):
        predicates = ontologyDefinition.findPredicateForSubjectObject(
          subjectColumn.class.uri, objectColumn.class.uri
        )
        unAssignedPredicate = predicates.filter(
          predicate not yet assigned to any
            (subjectColumn, objectColumn) pair
        )
        if (unAssignedPredicate is found):
          add relation annotation between columns
        end if
      else if (objectColumn.hasPredicateAnnotation):
        if (objectColumn.predicateDomain contains subjectColumn.class
          and objectColumn.predicate not yet assigned to any
            (subjectColumn, objectColumn) pair):

          add relation annotation between columns
        end if
      end if
    end for
  end if
  run classic relation discovery on columns with no relation found
end for
```

Code 7.9: Pseudo code of the proposed Relation Discovery phase of the algorithm.

The Relation Discovery Phase is implemented in the “uk.ac.shef.dcs.sti.core.algorithm.tmp.TMLColumnColumnRelationEnumerator” class.

### 7.1.5 Lookup of Predicate Between Subject and Object Entities

In order to describe the process of looking up of predicate between given subject and object classes, it is necessary to introduce the *direct domain (range)* of a *predicate* terms. Definition of these terms can be found in Definition 18.

**Definition 18** (Direct Domain (Range) of a Predicate). *Direct domain of a predicate is a set of URIs of classes, to which the predicate can be assigned, and this domain is defined for the predicate directly (the domain is not inherited from super property).*

*Direct range of a predicate is a set of ranges / values which can be assigned as the value of this predicate, and which is defined directly for given predicate - is not inherited from a super property.*

The lookup tries to find predicates of two types:

1. Look up predicates that have defined direct domain and range, their direct domain contains the class of the subject column, and their range contains the class of the object column. These predicates are added to ordered list of retrieved predicates.
2. Look up predicates that have defined domain — the constraint on the direct domain is dropped in this step, the domain can be also inherited from super properties — and direct range, their inherited domain contains the class of the subject column, and their range contains the class of the object column. These predicates are added to ordered list of retrieved predicates.

The ordered list (based on the score of the predicates) of predicates is then provided to the relation discovery algorithm. This algorithm selects the predicate with highest score which has not been assigned to any (subjectColumn, objectColumn) pair yet.

If no predicate is found during the two steps above, the algorithm will give up and no additional lookup for given subject / object column pair is performed.

A predicate needs to have a *range* defined in order to allow this lookup to find it. Predicate with no defined range will not be found.

The pseudo code of this lookup process can be found in Code 7.10.

```
subjectClass = subjectColumn.class
objectClass = objectColumn.class

directPredicates = predicates, for which:
    (p.directDomain contains subjectClass) and
    (p.range contains objectClass)

inheritedPredicates = predicates, for which:
    // in this case, domain can be also inherited
```

```

// from super properties
(p.domain contains subjectClass) and
(p.range contains objectClass)

foundPredicates = directPredicates ++ inheritedPredicates
return foundPredicates

```

Code 7.10: Pseudo code of the process of predicate between given subject and object entities lookup.

### 7.1.6 Adjustment of the thesis sub-goals

Because of the presented algorithm design and the way the algorithm works following sub-goals of this thesis could not completely satisfied by the algorithm:

- *Detection of classes of the products based on the detected predicates.*
- *Detection of predicates for the rest of the attributes. If the algorithm detects that that certain product is a laptop, then try to match the rest of the predicates according to properties which are usually assigned to laptops.*

Both the class of the product and the predicates of the attributes are resolved by the ML classifier during the ML PreClassification phase of the algorithm, when the columns of the input file are classified to their corresponding ML classes. The resulting ontology class of the product / attributes are then determined based on the provided ontology mapping file.

## 7.2 Odalic Integration

The proposed algorithm has been integrated into the Odalic system as an optional extension of the existing Odalic algorithm. That means, the user of the system can decide, whether he / she wants to use the new algorithm, or use the original Odalic algorithm at the time of creation (or modification) of the Odalic task. The API of the Odalic Core component, responsible for creating new or updating existing tasks has been extended in order to allow to pass the algorithm choice, and additional parameters for the new algorithm (in case its selected). Because of this change, the Odalic UI needed to be extended as well, to allow the user to provide these new options to the API - especially the *Add new task* and *Modify task* forms. The modified *Add new task* form can be seen in Figure 7.2, the added section of the form is highlighted in red color. The *Training Dataset* component of the form is very similar to the *Input file* configuration component, and it is able to upload a new file, if needed, and also to configure some properties of the file, such as *field separator*.

The screenshots of the example results screen of an Odalic task which was configured to use the new algorithm can be seen in following Figures 7.3 (detected column classifications), 7.4 (detected subject columns) and 7.5 (detected relations between columns). In the screenshots, the actual ordering of the columns in the table was changed in order to allow better projection on the cropped screenshots.

The image shows the 'Add new task' form in the Odalic UI. The form is organized into several sections:

- Basic properties:** Includes fields for 'Task identifier' and 'Description'.
- Input file:** Features a 'Source' section with 'Local file' selected, an 'Upload a new file' section with a 'Choose File' button, and an 'Identifier' field with an 'Upload' button. Below this is a 'Selected file' dropdown menu and a 'Configure' button.
- Machine Learning (highlighted):** Contains a 'Use Machine Learning' checkbox (checked), a 'Training Dataset' section with 'Local file' selected, and another 'Upload a new file' section with a 'Choose File' button, an 'Identifier' field, and an 'Upload' button. It also includes a 'Selected file' dropdown and a 'Configure' button.
- Used knowledge bases:** Includes a search bar for 'Knowledge bases' and a dropdown for 'Primary knowledge base'.
- Processing:** Features radio buttons for 'Lines processed' (set to 'All') and a checkbox for 'Statistical data'.

At the bottom of the form are three buttons: 'Save and run', 'Save', and 'Cancel'. The footer of the page indicates 'Odalic v1.1'.

Figure 7.2: Odalic UI: Add New Task form with added Machine Learning section highlighted.

## 7.3 Evaluation

For an evaluation of the actual benefits of the improved Odalic algorithm described in the beginning of this section, a couple of tasks based on the input data obtained for this thesis<sup>13</sup> have been proposed.

The input data files were, for the purpose of evaluation, combined into *three categories* according to the product type:

- Phone data
- Tablet data
- Computer data

<sup>13</sup>Input data retrieval and manipulation is described in Chapters 5 and 6.



TITLE_2	GPU_2	CPU_2	OS_2	PRICE_2	RAM_CAPACITY_2	RAM_TYPE_2
device (dbpedia-owl:Device)	GPU (http://www.kadlecek.sk/dt/ontology#GPU)	CPU (http://www.kadlecek.sk/dt/ontology#CPU)	software (dbpedia-owl:Software)			
hp zbook 15u g3 core i7-6500u 16gb 256gb ssd windows 10 professional laptop		intel core i7 6500u	windows 10 pro	£99.97	16gb	ddr4 sdram
asus k556lun core i7-7500 12gb 512gb ssd geforce gtx 940m dvd-rw 15.6 inch windows 10 laptop	geforce gtx 940m	intel core i7 7500u	windows 10 Windows 10@en (dbpedia:Windows_10)	£114.97	12gb	ddr4 sdram
hp probobook 650 g2 core i5-6200u 4gb 500gb dvd-rw 15.6 inch windows 10 professional laptop		intel core i5 6200u	windows 10 pro	£820	4gb	ddr4 sdram
hp elitebook folio g1 core i5-6200u 4gb 500gb dvd-rw 15.6 inch windows 10 professional laptop		intel core m5 6y54	windows 10 pro	£291.97	8gb	lpddr3 sdram

Figure 7.3: Odalic: Example of suggested classifications.

A result of this operation are three files, representing data from all scraped e-shops for given product category. These three files represent the input data for the “legacy” Odalic algorithm, however, for the new algorithm, additional pre-processing was required.

In order to create appropriate input data for the new Odalic algorithm, all records in each of the files described above were randomly shuffled and the resulting shuffled files were then split into two files - the training and testing dataset (Odalic input) files. The datasets were split using following ratio:

- Training Dataset: 20%
- Testing Dataset: 80%

In order to make the structure of the testing dataset files different to the training dataset files, in all of the testing dataset files the columns were randomly shuffled, and the column headings were re-named (by appending the “\_2” suffix to the column header).

Three Odalic tasks were then created - one for each product category. These tasks were then processed by both, legacy and new Odalic algorithms, and following metrics were evaluated for each of the semantization tasks - column classification, subject column detection, and discovery of relations between the columns:

- *Percentage of True Positives* - Percentage of columns (relations) which should have been classified (discovered) and also were correctly classified (discovered) by the algorithm.
- *Percentage of True Negatives* - Percentage of columns (relations) which should have not been classified (discovered) and also were not classified (discovered) by the algorithm.

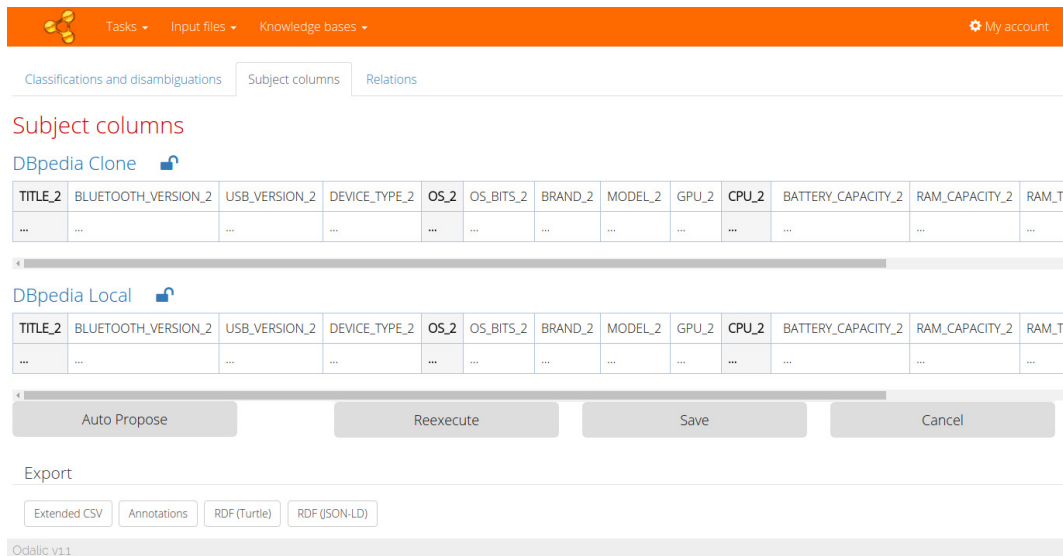


Figure 7.4: Odalic: Example of detected subject columns.

- *Percentage of False Positives* - Percentage of columns (relations) which should have not been classified (discovered), but were classified (discovered) by the algorithm.
- *Percentage of False Negatives* - Percentage of columns (relations) which should have been classified (discovered), but were not classified (discovered) by the algorithm, or were classified (discovered) by the algorithm, but the classification (discovery) was incorrect.

The results for each of the Odalic tasks were then averaged.

The actual classes of columns or relations between columns represented in testing datasets (input files) are unknown to the algorithm. The algorithm relies on the machine learning classifier in order to determine the actual classes of the columns and relations between them. However, these classes and relations were known to the author of this thesis, since the testing dataset is in fact a CSV file which contains headers of columns. These headers, despite the fact that they are different (renamed) than in the training dataset, uniquely represent certain class or predicate, to which the column should be mapped. Because of this, it was possible for the author of this thesis to objectively determine the correctness or incorrectness of the classes / relations discovered by the algorithm.

### 7.3.1 Original Odalic Algorithm Results

Since the original Odalic algorithm does not take training / testing datasets into account, the three files containing data for all products, split by category were used to create three Odalic tasks (one task for each product category). These tasks were run against the *DBpedia.org* knowledge base.

A result of all of these tasks is very similar - *no column* was classified into any ontological class and thus also *no relations* between columns were found. For one task, one subject column was determined, however, also this subject column was incorrect.

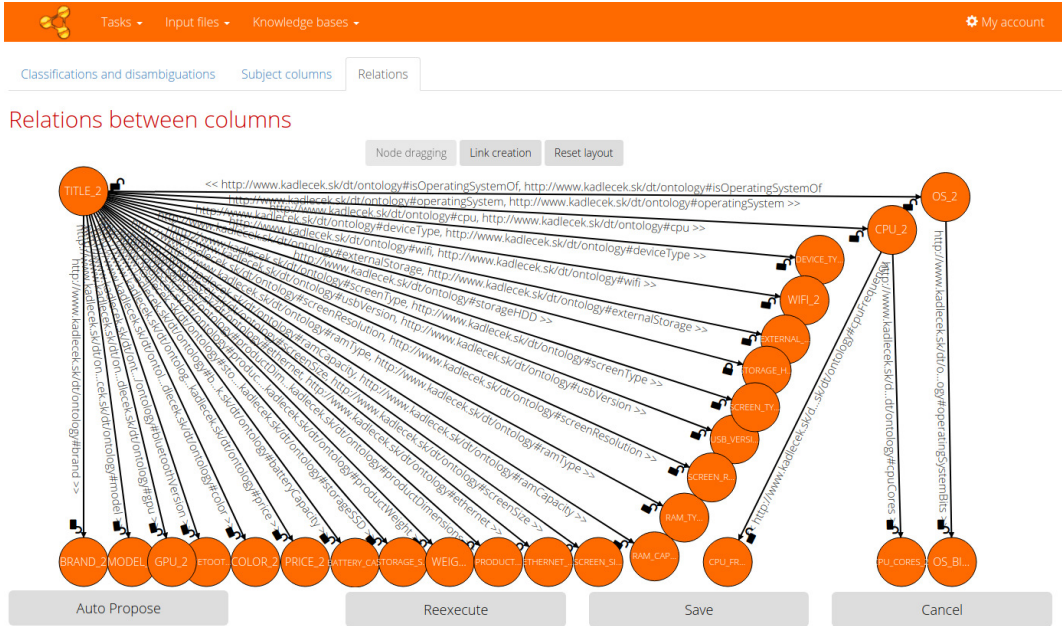


Figure 7.5: Odalic: Example of suggested relations.

### 7.3.2 New Odalic Algorithm Results

The averaged results of the tasks run by new Odalic algorithm can be found in Table 7.1.

	True Positive (%)	True Negative (%)	False Positive (%)	False Negative (%)
Column Classification	86.67	97.1	2.9	13.33
Subject Columns Detection	72.23	97.38	2.62	27.77
Relation Discovery	83.26	99.46	0.54	16.74

Table 7.1: Average results of tasks performed by the new Odalic algorithm.

The main outcome from Table 7.1 is that 86.67% of the columns were classified into correct ontology class, 72.23% of subject columns were correctly detected and 83.26% of the relations were correctly discovered. However, based on the description of the algorithm, it is clear that the subject column detection and relation discovery phases depend heavily on results of the column classification phase. Thus, for each of the tasks, where the column classification was not 100% successful, the classifications were manually corrected, and all tasks were re-run again.

Table 7.2 shows the average results of the tasks with manually corrected column classifications:

Results in Table 7.2 suggests that in case of completely correct column classification, the subject column detection phase achieves also 100% correct results, and results of the relation discovery are also improved by almost 10%.

In order to determine, whether having a larger training dataset would improve the algorithm results, additional three “inverse” tasks were performed. These

	<b>True Positive (%)</b>	<b>True Negative (%)</b>	<b>False Positive (%)</b>	<b>False Negative (%)</b>
Subject Columns Detection	100	100	0	0
Relation Discovery	92.18	99.66	0.34	7.82

Table 7.2: Average results of tasks with corrected column classifications, performed by the new Odalic algorithm.

tasks used the 80% datasets as training datasets, and 20% datasets as testing datasets. In order to make the 80% datasets work as a training dataset, its column headers needed to be renamed back to their original values. The column headers of the 20% datasets were also renamed, so they do not match the column headers in the training datasets.

The averaged results of the “inverse” tasks run by new Odalic algorithm can be found in Table 7.3.

	<b>True Positive (%)</b>	<b>True Negative (%)</b>	<b>False Positive (%)</b>	<b>False Negative (%)</b>
Column Classification	86.67	98.48	1.52	13.33
Subject Columns Detection	83.33	98.67	1.33	16.67
Relation Discovery	84.3	99.56	0.44	15.7

Table 7.3: Average results of “inverse” tasks performed by the new Odalic algorithm.

The results of the “inverse” tasks, especially the subject column detection phase, are improved in comparison with the results of the original tasks in Table 7.1. However, since there was hardly any improvement in the column classification phase, the results are still not as good as in the experiment with corrected classifications which can be seen in Table 7.2.

For the sake of completeness, Table 7.4 shows the average results of the “inverse” tasks with manually corrected column classifications:

	<b>True Positive (%)</b>	<b>True Negative (%)</b>	<b>False Positive (%)</b>	<b>False Negative (%)</b>
Subject Columns Detection	100	100	0	0
Relation Discovery	90.76	99.77	0.23	9.24

Table 7.4: Average results of “inverse” tasks with corrected column classifications, performed by the new Odalic algorithm.

Table 7.4 confirms that having correct column classifications has significant impact on the results of subject column detection and relation discovery phases. It also shows that the relation discovery phase performed slightly worse in com-

parison with results in Table 7.2. This was caused by several false-positive relation discoveries by a *fallback to legacy relation discovery* mechanism.

The exhaustive description of performed evaluation tasks and their results can be found in the Attachment E.. All dataset files used in the tasks referenced in this section can be found in the “/evaluation” folder and its subfolders on the attached CD.

The ontology used by the *ML PreClassification* and *Relation Discovery* phases of the algorithm was a combination of DBpedia ontology<sup>14</sup> and a custom ontology created by the author of this thesis — which focuses on the product data attributes<sup>15</sup>.

## Custom Ontology

For the purpose of describing the product data and improved algorithm evaluation, the author of this thesis created a new ontology which is based on the format of the input data tables, prepared within this thesis.

The prefix of this ontology is *rkdto*, with the “*http://www.kadlecek.sk/dt/ontology*” URI. The ontology is based on the DBpedia ontology whose predicates it extends. The purpose of this extension is that the *domain-s* and *range-s* of properties of the DBpedia ontology are not very well defined. This can decrease the effectiveness of the proposed algorithm. Also, some new properties — such as *rkdto:simCardType*) — and classes — such as *rkdto:CPU* — are introduced in this ontology.

The ontology also introduces a new object predicate, “*rkdto:isOperatingSystemOf*”. This predicate extends the “*dbo:operatingSystem*” predicate from DBpedia ontology which has the *dbo:Software* class in its domain. However, it is used across DBpedia as if the *dbo:Software* was in its range (invalid associativity). The introduced “*rkdto:isOperatingSystemOf*” predicate reflects this “reversed” associativity in better way.

The ontology definition file, in *Turtle* format, can be found in the “*odalic/sti/resources/ml/ontology/kadlecek\_onto.ttl*” file on the attached CD.

## 7.4 Semantization of Obtained Product Data

The Section 7.3 described the evaluation of the input data which were split into the 80% and 20% datasets. Since the evaluation was based on the results of Odalic tasks run on these datasets, during this evaluation the columns of the datasets were classified, subject columns were detected and some of the relations between the columns were discovered. Since these classifications were in case of most tasks not 100% correct, the author of this thesis corrected manually these wrong classifications / relations. These corrected task results served as a good base for the data semantization.

However, one last piece of the puzzle was missing in order to semantize the data properly - the disambiguation of the cells of table columns, which were classified as ontological classes, into knowledge base entities. Although the algorithm

---

<sup>14</sup>The ontology definition file can be found in the “*odalic/sti/resources/ml/ontology/dbpedia.2016-10.nt*” file on the attached CD.

<sup>15</sup>More information about the custom ontology can be found in the Section 7.3.2

was even able to disambiguate several entities in the table cells to existing knowledge base entities during the evaluation task runs, it was however only a small amount of them, since the product data were mostly not present in the used knowledge bases. Despite the fact that the columns are correctly classified and subject columns are correctly detected, the most of the data from the input files would not be semantized (exported) by the export functionality of the Odalic system, because of the missing disambiguations of the cell values.

There is a solution of this issue in the Odalic system. The user can *manually* propose a new knowledge base entities to the unknown cell values using the Odalic UI. This needs to be done individually for each table cell, so it would take significant amount of time and manual labor to do this for all of the unknown entities.

For this purpose, the author of this thesis implemented a new functionality into the Odalic system - *the entity auto-proposer*. This functionality adds a new ('Auto Propose') button to the results screen of the Odalic UI. The page footer, containing the new button (highlighted) can be seen in the Figure 7.6.

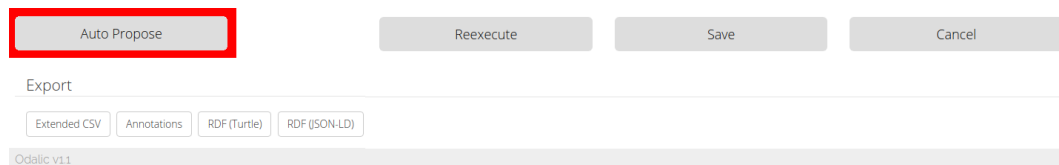


Figure 7.6: Odalic UI: Added “Auto Propose” button (highlighted).

By clicking this button, a new API call will be performed which triggers the auto-proposal functionality of the Odalic Core component. This functionality inserts all unknown (non-disambiguated) cell values of column, which are classified as ontology class, into the knowledge base creating a new entity records. The pseudo-code of the algorithm used by the auto-proposal functionality can be found in Code 7.10.

```
List proposals = List.empty
List rows = inputTable.rows
uriSuffix = new resource uri suffix from Odalic configuration

for each row in rows:
  for each column in row:
    cell = rows[row][column]
    if (column is classified as ontology class ||
        column should be ignored):

      continue
    end if

    if (cell not disambiguated && cell.value not yet proposed):
      label = cell.value
      uri = uriSuffix + UUID.randomUUID()
      proposals.add(new ResourceProposal(label, uri)
    end if
  end for
```

```

end for

List batches = split proposal to batches
for batch in batches:
    insert batch of proposals to knowledge base
end for

```

Code 7.10: Pseudo code of the entity auto-proposal feature.

The source code of the auto-proposal functionality can be found in the “cz.cuni.mff.xrg.odalic” class in the “odalic/sti” source directory on the attached CD.

After the auto-proposal is completed, it is required to re-execute the Odalic task which will now match newly proposed entities during the disambiguation phase of the algorithm.

An example result of a task after the auto-proposal was completed (and task was re-executed) can be seen in Figure 7.7.

TITLE_2	SCREEN_RESOLUTION_2	CPU_2	RAM_CAPACITY_2
device ( dbpedia-owl:Device ) device ( dbpedia-owl:Device )		CPU ( http://www.kadlecek.sk/dt/ontology#CPU )	
apple macbook pro 5th gen core i5 8gb 128gb ssd 13.3 inch retina intel iris 6100 laptop + electric globetrotter trolley bag apple macbook pro 5th gen core i5 8gb 128gb ssd 13.3 inch retina intel iris 6100 laptop + electric globetrotter trolley bag (http://odalic.eu/resource/d214a8bc-dcb9-43f6-a2e5-2b2312af4581)	2560 x 1600	intel core i5 5257u intel core i5 5257u (http://odalic.eu/resource/7b04423d-81f5-4eda-9466-21d5b197d117)	8gb
hp zbook 15 g3 intel xeon e3-1505m5 16gb 256gb ssd 15.6 inch windows 7 professional laptop hp zbook 15 g3 intel xeon e3-1505m5 16gb 256gb ssd 15.6 inch windows 7 professional laptop (http://odalic.eu/resource/33ac0bc7-dfc3-400a-bad7-efedc9c538c6)	1920 x 1080	intel xeon 1505m5 intel xeon 1505m5 (http://odalic.eu/resource/3a4b020f-a8bb-4094-a57e-3f6dd4e29a81)	16gb
asus vivomini vm65 core i3 6100u 4 gb 1 tb windows 10 desktop asus vivomini vm65 core i3 6100u 4 gb 1 tb windows 10 desktop (http://odalic.eu/resource/078add8e-e5bf-4e02-9f5f-3cbfa1cfc6bb)		intel core i3 6100u intel core i3 6100u (http://odalic.eu/resource/ad648413-18ea-4278-9664-e518cd56f18f)	4gb
acer predator g9-593 core i7-6700u 16gb 1tb+128gb ssd geforce gtx 1070 dvd-rw 15.6 inch windows 10 g acer predator g9-593 core i7-6700u 16gb 1tb+128gb ssd geforce gtx 1070 dvd-rw 15.6 inch windows 10 g (http://odalic.eu/resource/f32002a0-650d-436f-bd93-158cee33af2d)	1920 x 1080	intel core i7 i7-6700hq intel core i7 6700hq@en (http://odalic.eu/resource/17196f5b-87c0-48b2-9f26-e71d30ed0db2)	16gb
voyo v1 mini pc intel celeron n3450 voyo v1 mini pc intel celeron n3450 (http://odalic.eu/resource/43828f8c-601d-458e-a4e7-d3c58691f294)		intel celeron n3450 intel celeron n3450@en (http://odalic.eu/resource/69ebf573-857f-43f9-9588-21f34f1ab)	4g

Figure 7.7: Odalic UI: Example of task results with entity auto-proposal.

However, even after the auto-proposal is completed, Odalic system was not able to disambiguate some of the column cells into valid entities. These missing entities were matched manually by the author of this thesis.

The tasks were then auto-proposed and re-executed one after another in order to semantize the data. To prevent inserting the same entity into the knowledge base multiple times, after the auto-propose feature was used on a task, all remaining tasks were re-executed before using the auto-proposal feature to propose their undisambiguated cell values.

The semantized data from all tasks were then exported by the Odalic export feature into RDF files in *Turtle* format. These exported files containing the semantized product data can be found in the “semantized-data” folder on the attached CD.

# Conclusion & Future Work

The beginning of this thesis provided a necessary introduction into concepts of the Web 2.0, the Semantic Web, and also provided necessary information about technologies used within those concepts. A problem of table semantization (semantic table interpretation) was then introduced, together with an overview of existing approaches towards solving of this problem - together with detailed explanation of the TableMiner<sup>+</sup> algorithm and the Odalic system.

After providing necessary theoretical background, input data — html documents representing product detail pages — were collected for four different product categories — phones, tablets, desktop computers and laptops — from three different e-shops. A custom application, html-to-csv-tool, was implemented in order to extract the interesting parts (product specifications) of these HTML documents into CSV files suitable as an input to the Odalic system.

Having the input data ready, the thesis continued with an introduction of several machine learning classification algorithms which evaluation was then performed. At first, various features of the input data were evaluated in order to determine which features are the best ones for the final feature set used by machine learning algorithms. After the feature set was established, multiple different parameter combinations of machine learning algorithms were evaluated in order to determine which machine learning algorithm and with which parameter combination works the best on the input data. Evaluation was done using a custom evaluation framework, called ml-experiments, implemented for this purpose by the author of this thesis. The RandomForest algorithm was evaluated as the best performing one.

In the next part of the thesis a new, improved, Odalic algorithm, designed by the author of this thesis, was presented. This algorithm was designed to improve performance of the original Odalic algorithm in terms of performing column classification, subject column detection and relation discovery tasks on data with low or no presence in used knowledge bases, such as product data retrieved from e-shops. This should have been achieved by leveraging machine learning RandomForest classification algorithm. Based on the design of proposed algorithm, some of the sub-goals of this thesis needed to be adjusted.

After the algorithm was presented, this thesis described an approach how the algorithm was integrated into the Odalic API and Odalic UI and what steps are necessary to be performed by the user in order to activate the new algorithm for an Odalic task.

When the algorithm was integrated into the Odalic system, a set of tasks was performed in order to evaluate the actual improvement of the new algorithm in comparison with the original Odalic algorithm. For this purpose, input files were combined in a way that data about products of each product category were stored in a single file. A custom ontology, based on the DBpedia ontology was used for semantization. Since the original Odalic algorithm did not actually find any correct semantic information (column classes, correct subject columns or relations between columns), the evaluation part was mostly about measuring of performance of the new algorithm. For this purpose, input files were split into two datasets: training dataset — containing 20% of data — and testing



dataset — containing 80% of data. Using these datasets, average percentage of correct results for semantization tasks are: 86.67% for the column classification, 72.23% for the subject column detection and 83.26% for the relation discovery. However, the subject column detection and relation discovery phases depend heavily on the column classification phase. To demonstrate this, the algorithm was manually hinted with correct column classifications and Odalic tasks were re-executed. The results were as follows: 100% for the subject column detection and 92.18% for the relation discovery phases. In another evaluation experiment, the training and testing datasets were interchanged. The results were improved in magnitude of percents, but since the column classification basically did not change, improvements were not very significant.

The Odalic tasks used during evaluation of the algorithms covered data-wise all of the collected product data. Thus, for semantization of the data, these Odalic tasks were re-used. Some manual corrections of invalid classifications and relations were required. After these corrections were performed, a new functionality for entity auto-proposal has been added to Odalic. This feature allows the system to automatically propose new entities — for cell values which could not be disambiguated in the used knowledge bases — into the knowledge base. After proposing all entities which were unknown to the knowledge base and doing some final manual corrections, the data were correctly semantized and exported into RDF Turtle files. These files can be imported into some existing knowledge base and connected into the Linked Data cloud.

## 7.5 Future Work

There are several directions of possible future work on topics covered by this thesis. One of the directions can be focused on a part of actual HTML product data retrieval, such as development of a scraping tool which can work with web pages that use JavaScript to load the data. Popularity of these web pages is rising, however JavaScript is preventing conventional scraping tools from successfully scraping these web pages.

Another possible direction for future work would be to focus on improving the training dataset. This can be achieved by, for example, including data from more e-shops. With larger training dataset containing more variable data, the algorithm could achieve better results in column classification which would improve also results of other algorithm phases. Alternatively, a future work can focus on creating additional training datasets for different product categories which were not covered by this thesis.

And finally, another possible direction could be an effort to update the DBpedia ontology to make it more compatible with product data. This may include adding more product related classes / predicates and also making definitions of currently existing classes / predicates more precise by e.g. adding missing domains and ranges for predicates. This would improve the semantic meaning of the data and allow new uses of the data by data processing algorithms.

# Bibliography

- S. Aghaei, M. A. Nematbakhsh, and H. K. Farsani. Evolution of the World Wide Web: From Web 1.0 to Web 4.0. *International journal of Web & Semantic Technology [online]*, 3(1):1–10, January 2012. doi: 10.5121/ijwest.2012.3101. URL <http://airccse.org/journal/ijwest/papers/3112ijwest01.pdf>. Visited on 2016-05-29.
- E. Alpaydin. *Introduction to machine learning*. MIT Press, 2 edition, 2010. ISBN 978-0-262-01243-0.
- M. Amde and J. Bradley. Scalable Decision Trees in MLib, 2014. URL <https://databricks.com/blog/2014/09/29/scalable-decision-trees-in-mlib.html>. Visited on 2017-07-13.
- T. Berners-Lee. Linked Data, January 2006. URL <https://www.w3.org/DesignIssues/LinkedData.html>. Visited on 2016-03-20.
- T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. Technical report, Network Working Group, January 2005. URL <https://greenbytes.de/tech/webdav/rfc3986.pdf>. Visited on 2018-04-15.
- C. Bizer, T. Heath, and T. Berners-Lee. Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems [online]*, 5(3):1–22, 2009. ISSN 1552-6283. doi: 10.4018/jswis.2009081901. URL <http://tomheath.com/papers/bizer-heath-berners-lee-ijswis-linked-data.pdf>. Visited on 2016-03-20.
- L. Breiman. Random Forests. *Machine Learning [online]*, 45(1):5–32, 2001. ISSN 08856125. doi: 10.1023/A:1010933404324. URL <http://link.springer.com/10.1023/A:1010933404324>. Visited on 2017-07-11.
- CambridgeSemantics. Introduction to the Semantic Web. URL <http://www.cambridgesemantics.com/semantic-university/introduction-semantic-web>. Visited on 2016-03-20.
- M. Duerst and M. Suignard. Internationalized Resource Identifiers (IRIs), 2005. URL <http://www.rfc-base.org/txt/rfc-3987.txt>. Visited on 2018-04-15.
- R. Fielding and J. Reschke. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing, June 2014. URL <http://tools.ietf.org/html/rfc7230>. Visited on 2018-04-15.
- T. Finin, Z. Syed, V. Mulwad, and A. Joshi. Exploiting a Web of Semantic Data for Interpreting Tables. *WebSci10: Extending the Frontiers of Society On-Line*, 2010. URL <http://journal.webscience.org/322/>. Visited on 2017-07-07.
- E. Frank and I. H. Witten. Generating Accurate Rule Sets Without Global Optimization. *ICML '98 Proceedings of the Fifteenth International Conference on Machine Learning*, pages 144–151, 1998. URL <http://dl.acm.org/citation.cfm?id=657305>. Visited on 2017-07-11.

- I. Herman, B. Adida, M. Sporny, and M. Birbeck. RDFa 1.1 Primer - Third Edition, 2015. URL <https://www.w3.org/TR/rdfa-primer/>. Visited on 2016-05-01.
- W. Iba and P. Langley. Induction of One-Level Decision Trees. *ML '92 Proceedings of the Ninth International Workshop on Machine Learning*, pages 233–240, 1992. URL <http://dl.acm.org/citation.cfm?id=757759>. Visited on 2017-07-11.
- G. H. John and P. Langley. Estimating Continuous Distributions in Bayesian Classifiers. *UAI'95 Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 338–345, 1995. URL <http://dl.acm.org/citation.cfm?id=2074196>. Visited on 2017-07-11.
- T. Knap. Open Data Linker and Classifier - assignment, August 2016. URL <http://www.ksi.mff.cuni.cz/sw-projekty/zadani/odalic.pdf>. Visited on 2018-04-16.
- T. Knap. Towards odalic, a semantic table interpretation tool in the adequate project. In Anna Lisa Gentile, Andrea Giovanni Nuzzolese, and Ziqi Zhang, editors, *Proceedings of the 5th International Workshop on Linked Data for Information Extraction co-located with the 16th International Semantic Web Conference (ISWC 2017), Vienna, Austria, October 22, 2017.*, volume 1946 of *CEUR Workshop Proceedings*, pages 26–37. CEUR-WS.org, 2017. URL <http://ceur-ws.org/Vol-1946/paper-04.pdf>.
- R. Kohavi. The Power of Decision Tables. *ECML '95 Proceedings of the 8th European Conference on Machine Learning*, pages 174–189, 1995. URL <http://dl.acm.org/citation.cfm?id=649649>. Visited on 2017-07-11.
- S. B. Kotsiantis. Supervised Machine Learning: A Review of Classification Techniques. *Proceedings of the 2007 conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*, pages 3–24, 2007. URL <http://dl.acm.org/citation.cfm?id=1566770.1566773>. Visited on 2017-07-10.
- G. Limaye, S. Sarawagi, and S. Chakrabarti. Annotating and Searching Web Tables Using Entities, Types and Relationships. *Proceedings of the VLDB Endowment*, 3(1–2):1338–1347, 2010. ISSN 21508097. doi: 10.14778/1920841.1921005. URL <http://dl.acm.org/citation.cfm?doid=1920841.1921005>. Visited on 2017-07-07.
- Live Internet Stats. Total number of Websites, April 2018. URL <http://www.internetlivestats.com/total-number-of-websites/#trend>. Visited on 2018-04-20.
- Microformats Wiki. microformats 2 - Microformats Wiki, 2015. URL <http://microformats.org/wiki/microformats2>. Visited on 2016-05-01.

- E. Muñoz, A. Hogan, and A. Mileo. Triplifying Wikipedia’s Tables. *LD4IE’13 Proceedings of the First International Conference on Linked Data for Information Extraction*, 1057:26–37, 2013. URL [http://ceur-ws.org/Vol-1057/MunozEtAl\\_LD4IE2013.pdf](http://ceur-ws.org/Vol-1057/MunozEtAl_LD4IE2013.pdf). Visited on 2017-07-07.
- E. Muñoz, A. Hogan, and A. Mileo. Using Linked Data to Mine RDF from Wikipedia’s Tables. *Proceedings of the 7th ACM international conference on Web search and data mining - WSDM ’14 [online]*, pages 533–542, 2014. doi: 10.1145/2556195.2556266. URL <http://dl.acm.org/citation.cfm?doid=2556195.2556266>. Visited on 2017-07-07.
- ODALIC. ADEQUATE, ODALIC Documentation, February 2017. URL <https://odalic.github.io/download/ODALIC.Project.Documentation.pdf>. Visited on 2018-04-16.
- E. Osuna, R. Freund, and F. Girosi. Training Support Vector Machines: an Application to Face Detection. *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition [online]*, pages 130–136, 1997. doi: 10.1109/CVPR.1997.609310. URL <http://ieeexplore.ieee.org/document/609310/>. Visited on 2017-07-11.
- J. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schoelkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1998. URL <http://research.microsoft.com/~jplatt/smo.html>.
- R. L. Rivest. Learning Decision Lists. *Machine Learning*, 2(3):229–246, 1987. ISSN 08856125. doi: 10.1023/A:1022607331053. URL <http://dl.acm.org/citation.cfm?id=637934>. Visited on 2017-07-11.
- StatisticBrain. Total Number of Pages Indexed by Google, August 2015. URL <http://www.statisticbrain.com/total-number-of-pages-indexed-by-google/>. Visited on 2018-04-20.
- The Open Graph protocol. The Open Graph protocol, 2015. URL <http://ogp.me>. Visited on 2016-05-01.
- P. Venetis, A. Halevy, J. Madhavan, M. Paşca, W. Shen, F. Wu, and G. Wu. C. Miao. Recovering Semantics of Tables on the Web. *Proceedings of VLDB Endowment*, 4(9):528–538, 2011. ISSN 21508097. doi: 10.14778/2002938.2002939. URL <http://www.vldb.org/pvldb/vol14/p528-venetis.pdf>. Visited on 2017-07-07.
- W3C. XHTML 1.0: The Extensible HyperText Markup Language (Second Edition), August 2002. URL <https://www.w3.org/TR/xhtml11/>. Visited on 2018-04-15.
- W3C. W3C Technical Report Development Process, October 2005. URL <https://www.w3.org/2005/10/Process-20051014/tr.html>. Visited on 2018-04-15.
- W3C. XHTML 2 Working Group Expected to Stop Work End of 2009, W3C to Increase Resources on HTML 5, December 2009. URL <https://www.w3.org/News/2009#entry-6601>. Visited on 2018-04-15.

- W3C. Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification, 2011. URL <https://www.w3.org/TR/CSS2/>. Visited on 2018-04-15.
- W3C. CSS Syntax Module Level 3, February 2014a. URL <https://www.w3.org/TR/css-syntax-3/>. Visited on 2018-04-15.
- W3C. JSON-LD 1.0, 2014b. URL <https://www.w3.org/TR/json-ld/>. Visited on 2018-07-16.
- W3C. RDF 1.1 Concepts and Abstract Syntax, February 2014c. URL <https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/>. Visited on 2018-04-15.
- W3C. Ontologies - W3C, 2015a. URL <https://www.w3.org/standards/semanticweb/ontology>. Visited on 2018-04-15.
- W3C. Semantic Web - W3C, 2015b. URL <https://www.w3.org/standards/semanticweb/>. Visited on 2016-03-20.
- W3C. HTML 5.2, December 2017. URL <https://www.w3.org/TR/html52/>. Visited on 2018-04-15.
- J. Wang, H. Wang, Z. Wang, and K.Q. Zhu. Understanding Tables on the Web. *Proceedings of the 31st international conference on Conceptual Modeling*, pages 141–155, 2012. doi: 10.1007/978-3-642-34002-4\_11. URL [http://link.springer.com/10.1007/978-3-642-34002-4\\_11](http://link.springer.com/10.1007/978-3-642-34002-4_11). Visited on 2017-07-07.
- I. H. Witten and F. Eibe. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufman, Boston, MA, 2 edition, 2005. ISBN 0-12-088407-0.
- X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Metoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Zhou Z. H., Steinbach M., D. J. Hand, and D. Steinberg. Top 10 algorithms in data mining. *Knowledge and Information Systems [online]*, 14(1):1–37, 2008. ISSN 0219-1377. doi: 10.1007/s10115-007-0114-2. URL <http://dl.acm.org/citation.cfm?id=1566770.1566773>. Visited on 2017-07-11.
- A. Zell. *Simulation neuronaler Netze*. Addison-Wesley, 1997. ISBN 3893195548.
- Z. Zhang. Towards Efficient and Effective Semantic Table Interpretation. *ISWC '14 Proceedings of the 13th International Semantic Web Conference - Part I*, 2014. doi: 10.1007/978-3-319-11964-9\_31. URL [http://link.springer.com/10.1007/978-3-319-11964-9\\_31](http://link.springer.com/10.1007/978-3-319-11964-9_31). Visited on 2017-06-15.
- Z. Zhang. Effective and Efficient Semantic Table Interpretation using TableMiner+. *Semantic Web Journal (Accepted, Tracking: 1339-2551) [online]*, 2016. URL <http://semantic-web-journal.net/system/files/swj1339.pdf>.
- S. Zwicklbauer, C. Einsiedler, M. Granitzer, and C. Seifert. Towards Disambiguating Web Tables. *CEUR Workshop Proceedings*, 1035:205–208, 2013. URL [http://ceur-ws.org/Vol-1035/iswc2013\\_poster\\_7.pdf](http://ceur-ws.org/Vol-1035/iswc2013_poster_7.pdf). Visited on 2017-07-06.

# List of Figures

1.1	RDF graph representation of the information from Example 3. . . . .	15
3.1	TableMiner <sup>+</sup> Algorithm Flow. Source: Zhang [2016] . . . . .	26
3.2	Odalic UI - Feedback screen. . . . .	30
4.1	Example of product detail page of DebenhamsPlus.com e-shop (cropped). . . . .	33
5.1	Example of scraped product detail page of DebenhamsPlus.com e-shop (cropped). . . . .	38
5.2	Example of product feature list of DebenhamsPlus.com e-shop (cropped). . . . .	39
5.3	Example of converted and preprocessed CSV file (cropped). . . . .	39
6.1	Example of a decision tree. Source: Amde and Bradley [2014] . . . . .	41
6.2	Example schema of a neural network. Source: <a href="http://cs231n.github.io/neural-networks-1/">http://cs231n.github.io/neural-networks-1/</a> . . . . .	43
7.1	The proposed algorithm flow. . . . .	54
7.2	Odalic UI: Add New Task form with added Machine Learning section highlighted. . . . .	68
7.3	Odalic: Example of suggested classifications. . . . .	69
7.4	Odalic: Example of detected subject columns. . . . .	70
7.5	Odalic: Example of suggested relations. . . . .	71
7.6	Odalic UI: Added “Auto Propose” button (highlighted). . . . .	74
7.7	Odalic UI: Example of task results with entity auto-proposal. . . . .	75

# List of Tables

1.1	Selected CSS selectors. Source: <a href="https://www.w3.org/TR/CSS2/selector.html">https://www.w3.org/TR/CSS2/selector.html</a> . . . . .	9
6.1	Proposed Feature Set. . . . .	49
6.2	Average times of single 10-fold cross validation (= 10 validations) run of classification algorithms in Experiment 13 on the same PC. . . . .	50
6.3	Experiment 13: Used Feature Set. . . . .	51
6.4	Experiment 13: Results. . . . .	51
6.5	Comparison of best configurations of all evaluated algorithms . . . . .	52
6.6	Random Forest - configuration parameters with best results. . . . .	53
7.1	Average results of tasks performed by the new Odalic algorithm. . . . .	71
7.2	Average results of tasks with corrected column classifications, performed by the new Odalic algorithm. . . . .	72
7.3	Average results of “inverse” tasks performed by the new Odalic algorithm. . . . .	72
7.4	Average results of “inverse” tasks with corrected column classifications, performed by the new Odalic algorithm. . . . .	72
C.1	Experiment 1: Used Feature Set. . . . .	91
C.2	Experiment 1: Results. . . . .	91
C.3	Experiment 2: Used Feature Set. . . . .	91
C.4	Experiment 2: Results. . . . .	92
C.5	Experiment 3: Used Feature Set. . . . .	92
C.6	Experiment 3: Results. . . . .	92
C.7	Experiment 4: Used Feature Set. . . . .	93
C.8	Experiment 4: Results. . . . .	93
C.9	Experiment 5: Used Feature Set. . . . .	93
C.10	Experiment 5: Results. . . . .	94
C.11	Experiment 6: Used Feature Set. . . . .	94
C.12	Experiment 6: Results. . . . .	94
C.13	Experiment 7: Used Feature Set. . . . .	95
C.14	Experiment 7: Results. . . . .	95
C.15	Experiment 8: Used Feature Set. . . . .	95
C.16	Experiment 8: Results. . . . .	96
C.17	Experiment 9: Used Feature Set. . . . .	96
C.18	Experiment 9: Results. . . . .	96
C.19	Experiment 10: Used Feature Set. . . . .	97
C.20	Experiment 10: Results. . . . .	97
C.21	Experiment 11: Used Feature Set. . . . .	97
C.22	Experiment 11: Results. . . . .	98
C.23	Experiment 12: Used Feature Set. . . . .	98
C.24	Experiment 12: Results. . . . .	98
C.25	Experiment 14: Used Feature Set. . . . .	99
C.26	Experiment 14: Results. . . . .	99
C.27	Experiment 15: Used Feature Set. . . . .	99

C.28 Experiment 15: Results. . . . .	100
D.1 List of evaluated configurations for Decision Tree classifier. . . . .	101
D.2 Decision Tree - configuration parameters with best results. . . . .	102
D.3 List of evaluated configurations for Decision List classifier. . . . .	103
D.4 Decision List - configuration parameters with best results. . . . .	103
D.5 List of evaluated configurations for Decision Table classifier. . . . .	103
D.6 Decision Table - configuration parameters with best results. . . . .	104
D.7 List of evaluated configurations for Naive Bayes classifier. . . . .	104
D.8 Naive Bayes - configuration parameters with best results. . . . .	104
D.9 List of evaluated configurations for Random Forest classifier. . . . .	105
D.10 List of evaluated configurations for Support Vector Machine classifier. . . . .	105
D.11 Support Vector Machine - configuration parameters with best results. . . . .	106
D.12 List of evaluated configurations for Multilayer Perceptron classifier. . . . .	106
D.13 Multilayer Perceptron - configuration parameters with best results. . . . .	107
E.1 Task 1: Results. . . . .	108
E.2 Task 1: Results in case the column classification was correct for all columns. . . . .	108
E.3 Task 2: Results. . . . .	109
E.4 Task 2: Results in case the column classification was correct for all columns. . . . .	109
E.5 Task 3: Results. . . . .	110
E.6 Task 4: Results. . . . .	110
E.7 Task 4: Results in case the column classification was correct for all columns. . . . .	110
E.8 Task 5: Results. . . . .	111
E.9 Task 5: Results in case the column classification was correct for all columns. . . . .	111
E.10 Task 6: Results. . . . .	111



# List of Abbreviations

<b>CSV</b>	Comma-Separated Values
<b>UI</b>	User Interface
<b>UX</b>	User Experience
<b>HTML</b>	Hypertext Markup Language
<b>XHTML</b>	Extensible Hypertext Markup Language
<b>XML</b>	Extensible Markup Language
<b>URI</b>	Extensible Markup Language
<b>IRI</b>	Extensible Markup Language
<b>HTTP</b>	Hypertext Transfer Protocol
<b>HTTPS</b>	Hypertext Transfer Protocol Secure
<b>RDF</b>	Resource Description Framework
<b>NE</b>	Named Entity
<b>STI</b>	Semantic Table Interpretation
<b>CPU</b>	Central Processing Unit
<b>GUI</b>	Graphical User Interface
<b>SMO</b>	Sequential Minimal Optimization
<b>ARFF</b>	Attribute-Relation File Format

# A. Attachment 1: CD Contents

CD Path	Description
<b>files</b>	
- <b>inputData/raw</b>	Raw HTML files scraped from e-shops.
- <b>inputData/csv</b>	CSV files converted from scraped HTML files.
- <b>itSucksEshopTemplates</b>	Configuration templates used by the ItSucks tool for e-shop product data scraping.
<b>html-to-csv-tool</b>	
- <b>conf/jobs</b>	Job configuration files for used e-shops.
- <b>src</b>	Source code of the html-csv-tool application.
- <b>target/html-csv-tool.jar</b>	Pre-compiled runnable .jar package of the html-to-csv-tool application.
<b>ml-experiments</b>	
- <b>src</b>	Source code of the ml-experiments application.
- <b>target/ml-experiments.jar</b>	Pre-compiled .jar package of the ml-experiments application.
- <b>documentation</b>	
- - <b>ml-experiments-doc.pdf</b>	ML-experiments documentation.
- <b>resources</b>	Files related to the ML classification experiments.
- - <b>experimentX/input</b>	Input files for experiments, CSV & ARFF.
- - <b>experimentX/output</b>	Output files of experiments.
<b>odalic</b>	
- <b>sti</b>	Base directory of Odalic STI component, including source code of all of its modules.
- - <b>odalic</b>	
- - - <b>target</b>	
- - - - <b>odalic.war</b>	Pre-compiled war package of Odalic STI component with proposed algorithm included.
- - <b>config</b>	Odalic STI configuration folder.
- - - <b>ml.properties</b>	ML PreClassification configuration file.
- - <b>resources</b>	
- - - <b>ml</b>	
- - - - <b>mapping.json</b>	Example of ML class to Ontology concepts mapping file.
- - - - <b>ontology</b>	Ontology definition files.
- - - - - <b>dbpedia_2016-10.nt</b>	DBpedia ontology definition file.
- - - - - <b>kadlecek_onto.ttl</b>	Definition file of custom product ontology.
- <b>odalic-ui</b>	Odalic UI source files.
- <b>documentation</b>	Odalic documentation.
<b>evaluation</b>	Datasets for algorithm evaluation.
- <b>legacy-algorithm</b>	Datasets for legacy algorithm evaluation.
- - <b>input</b>	
- <b>new-algorithm</b>	Datasets for new algorithm evaluation.
- - <b>taskX</b>	Datasets for Task X of new algorithm evaluation.

- - - <b>training</b>	Training dataset
- - - <b>testing</b>	Testing dataset (Odalic input).
<b>semantized-data</b>	Semantized Product Data
<b>thesis</b>	
- thesis.pdf	PDF version of this Master Thesis.

# B. Attachment 2: HTML to CSV Conversion Application

## B.1 Getting the application

Source codes, already pre-compiled “jar” package, sample application configuration and job configurations for all e-shops described in the Chapter 5 can be found in the “/html-to-csv-tool” folder on the attached CD.

## B.2 Compiling application JAR package

HtmlToCsvTool application uses *Apache Maven*<sup>1</sup> to take care of the build process. The *runnable jar* package can be produced by running the command described in Code B.1 in the root directory of application sources. This should produce a runnable jar package “target/html-csv-tool-1.0-SNAPSHOT-full.jar”.

```
$ mvn clean package
```

Code B.1: Command for compilation of HtmlToCsvTool application jar package.

## B.3 Launching the application

The application does not have any graphical user interface, and can be thus launched only from the command line. It accepts *one* additional mandatory argument - the path to the configuration file describing the job which should be launched. Example of running the conversion of HTML files scraped from *GearBest.com* e-shop can be found in Code B.2. The application expects that the configuration file with (relative) path “conf/application.properties” exists.

```
$ java -jar html-csv-tool-1.0-SNAPSHOT-full.jar \  
"conf/jobs/gearbest_config.properties"
```

Code B.2: Example of launching HtmlToCsvTool application from the command line.

## B.4 Application Configuration

The application configuration is a java properties file which supports following configuration options:

- *threads.max.concurrent* - Maximum number of concurrently running processing threads. Value “-1” represents “number of all available processor cores”.

---

<sup>1</sup><https://maven.apache.org/>

- *thread.process.max.files* - Maximum size of a chunk - number of files processed by a single processing thread as a single task.

Example of application configuration file can be found in the “/html-to-csv-tool/conf/application.properties” file on the *attached CD*.

## B.5 Job Configuration

The configuration of a job is a Java properties file and consists of following properties:

- *files.path* - Absolute path to the folder containing the input HTML files. All files present in given directory (and recursively its subdirectories) will be processed by the application.
- *output.path* - Absolute path to the folder, where the output files should be placed.
- *uri.prefix* - Each line in the output CSV file describes a single resource (single product) described by HTML document. This property defines a prefix which will be appended to the URI of the resource, such as “http://example.org/”. The rest of the URI is determined by the HTML file path on the filesystem, relative to the *files.path* folder.
- *output.csv.lineSeparator* - Separator to be used to separate lines in the output CSV files.
- *output.csv.columnSeparator* - Separator to be used to separate columns in the output CSV files.
- *output.csv.encloseBy* - Character which will be used to enclose the values in the output CSV files.
- *website.type* - Type of the website. This setting determines which parser should be used for the job. The application supports website types of all e-shops described in the Section 5.1 of the Chapter 5. The enumeration of values which can be used in the job configuration file are:
  - GEARBEST
  - MOBILESHOP
  - DEBENHAMSPLUS

## B.6 Adding Support For a New Website Type

There are two points in the application code which needs to be updated in order to add a support of a new website type to the application. Both of these update points will be described in following subsections.

### B.6.1 Parsers Package

The “*sk.kadlecek.htmltablefetcher.parser*” package contains implementation of all supported parsers. Every parser should extend the *AbstractParser* class which provides couple of useful helper methods, as well as defines the basic structure of a parser using abstract methods. These methods need to be implemented by each parser. The default parsers use *Jsoup*<sup>2</sup> library for querying of the HTML using CSS selectors, but new parsers can use completely different approach.

### B.6.2 WebsiteType Enumeration

Once the parser implementation is finished, the new *WebsiteType* can be added to the enumeration of all supported *WebsiteTypes*. This is done by adding of a new value to the “*sk.kadlecek.htmltablefetcher.enumeration.WebsiteType*” enumeration, together with the reference to a parser class which is able to process files of the new *WebsiteType*.

---

<sup>2</sup><https://jsoup.org/>

# C. Attachment 3: Feature Experiments

The input data used for experiments can be found in the “/ml-experiments/resources/experimentX/input” folders (where X is the number of the Experiment) on the attached CD.

Experiments in following sections were performed in order to discover the best performing set of features on given input data.

## C.1 Experiment 1

The feature set used in this experiment can be found in Table C.1.

---

<b>Numeric Features</b>
Total Number of Words

---

Table C.1: Experiment 1: Used Feature Set.

Results of the experiment can be found in Table C.2.

---

<b>Algorithm</b>	<b>Average Error Rate (%)</b>	<b>Average Precision (%)</b>	<b>Average Recall (%)</b>	<b>Average F-measure</b>
Decision Tree	38,80	45,41	61,20	51,31
Decision Stump	52,16	25,84	47,84	32,92
Decision List	38,79	45,45	61,21	51,36
Decision Table	38,84	45,27	61,16	51,22
Naive Bayes	48,31	32,14	51,69	38,36
Random Forrest	38,78	45,48	61,22	51,32
Support Vector Machine	50,76	28,03	49,24	35,29
Multilayer Perceptron	39,37	43,70	60,63	50,38

---

Table C.2: Experiment 1: Results.

## C.2 Experiment 2

The feature set used in this experiment can be found in Table C.3.

---

<b>Numeric Features</b>
Total Number of Words
Total Number of Characters

---

Table C.3: Experiment 2: Used Feature Set.

Results of the experiment can be found in Table C.4.

<b>Algorithm</b>	<b>Average Error Rate (%)</b>	<b>Average Precision (%)</b>	<b>Average Recall (%)</b>	<b>Average F-measure</b>
Decision Tree	28,52	64,33	71,48	66,86
Decision Stump	51,23	24,67	48,77	32,63
Decision List	28,71	65,05	71,29	67,07
Decision Table	30,74	62,27	69,26	64,22
Naive Bayes	44,37	44,85	55,63	48,43
Random Forrest	29,31	65,18	70,69	67,00
Support Vector Machine	49,48	27,96	50,52	35,58
Multilayer Perceptron	32,29	57,09	67,71	61,08

Table C.4: Experiment 2: Results.

### C.3 Experiment 3

The feature set used in this experiment can be found in Table C.5.

#### **Numeric Features**

Total Number of Words  
Total Number of Characters  
Total Number of Letters

Table C.5: Experiment 3: Used Feature Set.

Results of the experiment can be found in Table C.6.

<b>Algorithm</b>	<b>Average Error Rate (%)</b>	<b>Average Precision (%)</b>	<b>Average Recall (%)</b>	<b>Average F-measure</b>
Decision Tree	17,44	80,69	82,56	81,13
Decision Stump	50,32	25,42	49,68	33,40
Decision List	17,51	80,89	82,49	81,34
Decision Table	21,71	76,06	78,29	76,29
Naive Bayes	36,92	59,42	63,08	59,42
Random Forrest	15,91	82,97	84,09	83,29
Support Vector Machine	40,88	40,82	59,12	47,15
Multilayer Perceptron	24,52	72,77	75,48	72,90

Table C.6: Experiment 3: Results.

### C.4 Experiment 4

The feature set used in this experiment can be found in Table C.7.

Results of the experiment can be found in Table C.8.



---

**Numeric Features**

---

Total Number of Words  
Total Number of Characters  
Total Number of Letters  
Total Number of Digits

Table C.7: Experiment 4: Used Feature Set.

---

<b>Algorithm</b>	<b>Average Error Rate (%)</b>	<b>Average Precision (%)</b>	<b>Average Recall (%)</b>	<b>Average F-measure</b>
Decision Tree	14,91	83,93	85,09	84,23
Decision Stump	50,32	25,42	49,68	33,40
Decision List	15,05	84,06	84,95	84,33
Decision Table	20,56	80,31	79,44	78,65
Naive Bayes	37,07	62,93	62,93	61,71
Random Forrest	12,57	86,81	87,43	86,96
Support Vector Machine	40,44	41,14	59,56	47,75
Multilayer Perceptron	22,70	75,86	77,30	75,45

---

Table C.8: Experiment 4: Results.

## C.5 Experiment 5

The feature set used in this experiment can be found in Table C.9.

---

**Numeric Features**

---

Total Number of Words  
Total Number of Characters  
Total Number of Letters  
Total Number of Digits  
Total Number of Whitespace Characters

Table C.9: Experiment 5: Used Feature Set.

Results of the experiment can be found in Table C.10.

## C.6 Experiment 6

The feature set used in this experiment can be found in Table C.11.

Results of the experiment can be found in Table C.12.

## C.7 Experiment 7

The feature set used in this experiment can be found in Table C.13.

Results of the experiment can be found in Table C.14.

<b>Algorithm</b>	<b>Average Error Rate (%)</b>	<b>Average Precision (%)</b>	<b>Average Recall (%)</b>	<b>Average F-measure</b>
Decision Tree	14,81	84,02	85,19	84,31
Decision Stump	50,32	25,42	49,68	33,40
Decision List	15,00	84,16	85,00	84,42
Decision Table	20,69	80,54	79,31	78,70
Naive Bayes	36,85	64,19	63,15	62,50
Random Forrest	12,34	87,06	87,66	87,20
Support Vector Machine	39,21	44,23	60,79	49,81
Multilayer Perceptron	21,47	76,88	78,53	76,38

Table C.10: Experiment 5: Results.

### **Numeric Features**

Total Number of Words  
Total Number of Characters  
Total Number of Letters  
Total Number of Digits  
Total Number of Whitespace Characters  
Total Number of Other Characters

Table C.11: Experiment 6: Used Feature Set.

<b>Algorithm</b>	<b>Average Error Rate (%)</b>	<b>Average Precision (%)</b>	<b>Average Recall (%)</b>	<b>Average F-measure</b>
Decision Tree	13,33	86,11	86,67	86,20
Decision Stump	50,32	25,42	49,68	33,40
Decision List	13,36	86,11	86,64	86,21
Decision Table	20,67	83,42	79,33	79,38
Naive Bayes	35,26	67,25	64,74	64,81
Random Forrest	11,41	88,08	88,59	88,21
Support Vector Machine	39,13	44,32	60,87	50,04
Multilayer Perceptron	21,02	77,78	78,98	77,32

Table C.12: Experiment 6: Results.

## **C.8 Experiment 8**

The feature set used in this experiment can be found in Table C.15.

Results of the experiment can be found in Table C.16.

## **C.9 Experiment 9**

The feature set used in this experiment can be found in Table C.17.

Results of the experiment can be found in Table C.18.

<b>Numeric Features</b>	<b>Boolean Features</b>
Total Number of Words	Is the String Numeric Value
Total Number of Characters	
Total Number of Letters	
Total Number of Digits	
Total Number of Whitespace Characters	
Total Number of Other Characters	

Table C.13: Experiment 7: Used Feature Set.

<b>Algorithm</b>	<b>Average Error Rate (%)</b>	<b>Average Precision (%)</b>	<b>Average Recall (%)</b>	<b>Average F-measure</b>
Decision Tree	13,22	86,29	86,78	86,33
Decision Stump	50,32	25,42	49,68	33,40
Decision List	13,22	86,31	86,78	86,34
Decision Table	20,57	83,59	79,43	79,51
Naive Bayes	35,23	67,31	64,77	64,84
Random Forrest	11,30	88,20	88,70	88,31
Support Vector Machine	39,03	44,75	60,97	50,31
Multilayer Perceptron	21,06	78,05	78,94	77,42

Table C.14: Experiment 7: Results.

<b>Numeric Features</b>	<b>Boolean Features</b>
Total Number of Words	Is the String Numeric Value
Total Number of Characters	Is the String Integral Numeric Value
Total Number of Letters	
Total Number of Digits	
Total Number of Whitespace Characters	
Total Number of Other Characters	

Table C.15: Experiment 8: Used Feature Set.

## C.10 Experiment 10

The feature set used in this experiment can be found in Table C.19.

Results of the experiment can be found in Table C.20.

## C.11 Experiment 11

The feature set used in this experiment can be found in Table C.21.

Results of the experiment can be found in Table C.22.

<b>Algorithm</b>	<b>Average Error Rate (%)</b>	<b>Average Precision (%)</b>	<b>Average Recall (%)</b>	<b>Average F-measure</b>
Decision Tree	13,22	86,29	86,78	86,33
Decision Stump	50,32	25,42	49,68	33,40
Decision List	13,27	86,23	86,73	86,29
Decision Table	20,55	83,58	79,45	79,51
Naive Bayes	35,22	67,08	64,78	64,73
Random Forrest	11,32	88,22	88,68	88,31
Support Vector Machine	39,01	45,17	60,99	50,37
Multilayer Perceptron	21,51	78,04	78,49	77,13

Table C.16: Experiment 8: Results.

<b>Numeric Features</b>	<b>Boolean Features</b>
Total Number of Words	Is the String Numeric Value
Total Number of Characters	Is the String Integral Numeric Value
Total Number of Letters	Is the String Decimal Numeric Value
Total Number of Digits	
Total Number of Whitespace Characters	
Total Number of Other Characters	

Table C.17: Experiment 9: Used Feature Set.

<b>Algorithm</b>	<b>Average Error Rate (%)</b>	<b>Average Precision (%)</b>	<b>Average Recall (%)</b>	<b>Average F-measure</b>
Decision Tree	13,22	86,29	86,78	86,33
Decision Stump	50,32	25,42	49,68	33,40
Decision List	13,28	86,19	86,72	86,28
Decision Table	20,55	83,58	79,45	79,51
Naive Bayes	35,17	67,19	64,83	64,80
Random Forrest	11,27	88,25	88,73	88,35
Support Vector Machine	39,01	45,14	60,99	50,36
Multilayer Perceptron	21,17	78,67	78,83	77,47

Table C.18: Experiment 9: Results.

## C.12 Experiment 12

The feature set used in this experiment can be found in Table C.23.

Results of the experiment can be found in Table C.24.

## C.13 Experiment 13

The feature set used in this experiment can be found in Table 6.3 in Chapter 6.

<b>Numeric Features</b>	<b>Boolean Features</b>
Total Number of Words	Is the String Numeric Value
Total Number of Characters	Is the String Integral Numeric Value
Total Number of Letters	Is the String Decimal Numeric Value
Total Number of Digits	Is the String Prefixed Number (e.g. "\$100")
Total Number of Whitespace Characters	
Total Number of Other Characters	

Table C.19: Experiment 10: Used Feature Set.

<b>Algorithm</b>	<b>Average Error Rate (%)</b>	<b>Average Precision (%)</b>	<b>Average Recall (%)</b>	<b>Average F-measure</b>
Decision Tree	12,17	87,33	87,83	87,39
Decision Stump	50,32	25,42	49,68	33,40
Decision List	12,23	87,21	87,77	87,33
Decision Table	19,75	84,33	80,25	80,40
Naive Bayes	34,80	67,38	65,20	65,07
Random Forrest	10,14	89,32	89,86	89,47
Support Vector Machine	38,87	50,94	61,13	53,34
Multilayer Perceptron	19,94	78,72	80,06	78,62

Table C.20: Experiment 10: Results.

<b>Numeric Features</b>	<b>Boolean Features</b>
Total Number of Words	Is the String Numeric Value
Total Number of Characters	Is the String Integral Numeric Value
Total Number of Letters	Is the String Decimal Numeric Value
Total Number of Digits	Is the String Prefixed Number (e.g. "\$100")
Total Number of Whitespace Characters	Is the String Postfixed Number (e.g. "2 GB")
Total Number of Other Characters	

Table C.21: Experiment 11: Used Feature Set.

Results of the experiment can be found in Table 6.4 in Chapter 6.

## C.14 Experiment 14

The feature set used in this experiment can be found in Table C.25.

Results of the experiment can be found in Table C.26.

<b>Algorithm</b>	<b>Average Error Rate (%)</b>	<b>Average Precision (%)</b>	<b>Average Recall (%)</b>	<b>Average F-measure</b>
Decision Tree	11,76	87,65	88,24	87,78
Decision Stump	50,32	25,42	49,68	33,40
Decision List	11,71	87,76	88,29	87,90
Decision Table	19,57	84,14	80,43	80,45
Naive Bayes	34,04	68,23	65,96	65,92
Random Forrest	9,45	90,02	90,55	90,16
Support Vector Machine	39,07	51,11	60,93	52,92
Multilayer Perceptron	18,90	79,90	81,10	79,95

Table C.22: Experiment 11: Results.

<b>Numeric Features</b>	<b>Boolean Features</b>
Total Number of Words	Is the String Numeric Value
Total Number of Characters	Is the String Prefixed Number
Total Number of Letters	Is the String Postfixed Number
Total Number of Digits	
Total Number of Other Characters	

Table C.23: Experiment 12: Used Feature Set.

<b>Algorithm</b>	<b>Average Error Rate (%)</b>	<b>Average Precision (%)</b>	<b>Average Recall (%)</b>	<b>Average F-measure</b>
Decision Tree	11,92	87,43	88,08	87,57
Decision Stump	50,32	25,42	49,68	33,40
Decision List	11,79	87,75	88,21	87,85
Decision Table	19,20	84,04	80,80	80,70
Naive Bayes	33,73	67,92	66,27	65,80
Random Forrest	9,67	89,89	90,33	90,01
Support Vector Machine	39,32	49,99	60,68	52,39
Multilayer Perceptron	18,79	80,17	81,21	80,13

Table C.24: Experiment 12: Results.

## C.15 Experiment 15

The feature set used in this experiment can be found in Table C.27.

Results of the experiment can be found in Table C.28.

<b>Numeric Features</b>	<b>Boolean Features</b>
Total Number of Words	Is the String Prefixed Number
Total Number of Characters	Is the String Postfixed Number
Total Number of Letters	
Total Number of Digits	
Total Number of Whitespace Characters	
Total Number of Other Characters	

Table C.25: Experiment 14: Used Feature Set.

<b>Algorithm</b>	<b>Average Error Rate (%)</b>	<b>Average Precision (%)</b>	<b>Average Recall (%)</b>	<b>Average F-measure</b>
Decision Tree	11,76	87,65	88,24	87,78
Decision Stump	50,32	25,42	49,68	33,40
Decision List	11,71	87,81	88,29	87,90
Decision Table	19,56	84,10	80,44	80,46
Naive Bayes	34,24	68,13	65,76	65,77
Random Forrest	9,48	90,07	90,52	90,18
Support Vector Machine	39,20	50,29	60,80	52,68
Multilayer Perceptron	19,42	79,22	80,58	79,43

Table C.26: Experiment 14: Results.

<b>Numeric Features</b>	<b>Boolean Features</b>
Total Number of Words	Is the String Integral Numeric Value
Total Number of Characters	Is the String Decimal Numeric Value
Total Number of Letters	Is the String Prefixed Number
Total Number of Digits	Is the String Postfixed Number
Total Number of Whitespace Characters	
Total Number of Other Characters	

Table C.27: Experiment 15: Used Feature Set.

<b>Algorithm</b>	<b>Average Error Rate (%)</b>	<b>Average Precision (%)</b>	<b>Average Recall (%)</b>	<b>Average F-measure</b>
Decision Tree	11,76	87,65	88,24	87,78
Decision Stump	50,32	25,42	49,68	33,40
Decision List	11,71	87,79	88,29	87,91
Decision Table	19,57	84,14	80,43	80,45
Naive Bayes	34,14	68,08	65,86	65,80
Random Forrest	9,46	90,04	90,54	90,17
Support Vector Machine	39,05	51,16	60,95	52,95
Multilayer Perceptron	19,44	79,45	80,56	79,46

Table C.28: Experiment 15: Results.



# D. Attachment 4: Algorithm Configurations

This chapter contains the exhaustive descriptions of all configuration properties and values which were tracked during the evaluation. For each algorithm, also the best configuration is listed. At the end of this chapter, a comparison table of results of best configurations of all algorithm is listed.

## D.1 Decision Tree

Following list enumerates all of the tracked algorithm properties of the Decision Tree algorithm. Summary of all evaluated configurations can be found in Table D.1.

- *Minimum Number of Objects per Leaf* - Minimum Number of Objects that needs to be stored in every leaf of the decision tree.
- *Subtree Raising* - Perform, or don't perform, the subtree raising<sup>1</sup> operation.
- *Use Laplace Smoothing for Probabilities* - Do, or don't, apply the *Laplace Smoothing* algorithm on the predicted probabilities.
- *MLD Correction* - Do, or don't, use MDL correction operation on numeric attributes.
- *Tree Pruning* - Sets whether the decision tree should be pruned or not<sup>2</sup>.
- *Confidence Factor* - The value of confidence threshold used for decision tree pruning.

Property	Type	Values
Minimum Number of Objects per Leaf	integer	0 – 10 (step 1)
Subtree Raising	boolean	true / false
Use Laplace Smoothing for Probabilities	boolean	true / false
MLD Correction	boolean	true / false
Tree Pruning	boolean	true / false
Confidence Factor	float	0.1 – 0.4 (step 0.1)

Table D.1: List of evaluated configurations for Decision Tree classifier.

The algorithm configuration which, according to the results, performed the best can be found in Table D.2.

<sup>1</sup>Replaces selected subtree with its child-subtree

<sup>2</sup>Replace selected node with a leaf and assign most common class in original node subtree as the value of that leaf.

Parameter	Value
Minimum Number of Objects per Leaf	1
Subtree Raising	true
Use Laplace Smoothing for Probabilities	false
MLD Correction	true
Tree Pruning	true
Confidence Factor	0.4

Table D.2: Decision Tree - configuration parameters with best results.

## D.2 Decision Stump

Decision stump algorithm does not support any additional configuration.

## D.3 Decision List

Following list enumerates all of the tracked algorithm properties of the Decision List algorithm. Summary of all evaluated configurations can be found in Table D.3.

- *Confidence Factor* - The value of confidence threshold used for decision tree pruning.
- *Minimum Number of Objects per Leaf* - Minimum Number of Objects that needs to be stored in every leaf of the decision tree.
- *Use Binary Splits Only* - Do, or don't, use only binary splits.
- *Make Split Point Actual Value* - Do, or don't, create an actual value out of a split point.
- *MLD Correction* - Do, or don't, use MDL correction operation on numeric attributes.
- *Tree Pruning* - Sets whether the decision tree should be pruned or not<sup>3</sup>.
- *Reduced Error Pruning* - Do, or don't, use Reduced Error Pruning<sup>4</sup>.

The algorithm configuration which, according to the results, performed the best can be found in Table D.4.

## D.4 Decision Table

Following list enumerates all of the tracked algorithm properties of the Decision Table algorithm. Summary of all evaluated configurations can be found in Table D.5.

<sup>3</sup>Replace selected node with a leaf and assign most common class in original node subtree as the value of that leaf.

<sup>4</sup>Prune nodes which will increase the accuracy of the classifier the most

Property	Type	Values
Confidence Factor	float	0.1 – 0.4 (step 0.1)
Minimum Number of Objects per Leaf	integer	0 – 10 (step 1)
Use Binary Splits Only	boolean	true / false
Make Split Point Actual Value	boolean	true / false
MLD Correction	boolean	true / false
Tree Pruning	boolean	true / false
Reduced Error Pruning	boolean	true / false

Table D.3: List of evaluated configurations for Decision List classifier.

Parameter	Value
Confidence Factor	0.1
Minimum Number of Objects per Leaf	0
Use Binary Splits Only	false
Make Split Point Actual Value	true
MLD Correction	true
Tree Pruning	false
Reduced Error Pruning	false

Table D.4: Decision List - configuration parameters with best results.

- *Search Method* - Search Method to use within the Decision Table. Allowed values:
  - **BestFirst**<sup>5</sup> - Can have one of following direction values: *Forward*, *Backward*, *Bidirectional*.
  - **GreedyStepWise**<sup>6</sup> - Can have one of following direction values: *Forward*, *Backward*.
- *Use Nearest Neighbour Instead Of Global Majority (boolean)* - Do, or don't, use the *Nearest Neighbour* method to select the classifier result, instead of *Global Majority* method.

Property	Type	Values
Search Method	object	<b>BestFirst</b> (Forward / Backward / Bidirectional) / <b>GreedyStepWise</b> (Forward / Backward)
Use Nearest Neighbour Instead Of Global Majority	boolean	true / false

Table D.5: List of evaluated configurations for Decision Table classifier.

The algorithm configuration which, according to the results, performed the best can be found in Table D.6.

<sup>5</sup><http://weka.sourceforge.net/doc.dev/weka/attributeSelection/BestFirst.html>

<sup>6</sup><http://weka.sourceforge.net/doc.dev/weka/attributeSelection/GreedyStepwise.html>

Parameter	Value
Search Method	GreedyStepWise (Backward)
Use Nearest Neighbour Instead Of Global Majority	true

Table D.6: Decision Table - configuration parameters with best results.

## D.5 Naive Bayes

Following list enumerates all of the tracked algorithm properties of the Naive Bayes algorithm. Summary of all evaluated configurations can be found in Table D.7.

- *Use Kernel Density Estimator* - Do, or don't, use kernel density operator for numeric attributes, instead of normal distribution.
- *Use Supervised Discretization Factor* - Do, or don't, process numeric attributes using supervised discretization method.

Property	Type	Values
Use Kernel Density Estimator	boolean	true / false
Use Supervised Discretization Factor	boolean	true / false

Table D.7: List of evaluated configurations for Naive Bayes classifier.

The algorithm configuration which, according to the results, performed the best can be found in Table D.8.

Parameter	Value
Use Kernel Density Estimator	false
Use Supervised Discretization Factor	true

Table D.8: Naive Bayes - configuration parameters with best results.

## D.6 Random Forest

Following list enumerates all of the tracked algorithm properties of the Random Forest algorithm. Summary of all evaluated configurations can be found in Table D.9.

- *Batch Size* - Batch Size for the batch prediction.
- *Max Depth* - Maximum depth of each random tree.
- *Number of Features* - Number of features to check (randomly).

- *Number of Bagging Iterations* - Number of iterations to use for the learning process.
- *Break Ties Randomly* - Do, or don't, randomly break ties<sup>7</sup>.

Property	Type	Values
Batch Size	integer	80 – 120 (step 10)
Max Depth	integer	1 – 16 (step 2)
Number of Features	integer	1 – 10 (step 1)
Number of Bagging Iterations	integer	80 – 120 (step 10)
Break Ties Randomly	boolean	true / false

Table D.9: List of evaluated configurations for Random Forest classifier.

The algorithm configuration which, according to the results, performed the best can be found in Table 6.6 in the Chapter 6.

## D.7 Support Vector Machine

Following list enumerates all of the tracked algorithm properties of the Support Vector Machine algorithm. Summary of all evaluated configurations can be found in Table D.10.

- *Complexity Constant* - Complexity parameter, used by SMO to build hyperplane between classes.
- *Kernel* - Kernel implementation to use. Possible values: **PolyKernel**<sup>8</sup>, **Puk**<sup>9</sup>, **RBFKernel**<sup>10</sup>.
- *Build Calibration Model* - Do, or don't, build calibration models to fit to SVM output.

Property	Type	Values
Complexity Constant	float	0.7 – 1.5 (step 0.1)
Kernel	object	PolyKernel / Puk / RBFKernel
Build Calibration Model	boolean	true / false

Table D.10: List of evaluated configurations for Support Vector Machine classifier.

The algorithm configuration which, according to the results, performed the best can be found in Table D.11.

<sup>7</sup>Tie is a situation, when there is no best majority vote (multiple features appear to be equally good)

<sup>8</sup><http://weka.sourceforge.net/doc.dev/weka/classifiers/functions/supportVector/PolyKernel.html>

<sup>9</sup><http://weka.sourceforge.net/doc.dev/weka/classifiers/functions/supportVector/Puk.html>

<sup>10</sup><http://weka.sourceforge.net/doc.dev/weka/classifiers/functions/supportVector/RBFKernel.html>

Parameter	Value
Complexity Constant	1.4
Kernel	Puk
Build Calibration Model	true

Table D.11: Support Vector Machine - configuration parameters with best results.

## D.8 Multilayer Perceptron

Following list enumerates all of the tracked algorithm properties of the Multilayer Perceptron algorithm. Summary of all evaluated configurations can be found in Table D.12.

- *Hidden Layer Types* - Hidden layer types to use<sup>11</sup>.
- *Learning Rate* - Learning rate for the backpropagation algorithm.
- *Momentum* - Momentum rate for the backpropagation algorithm.
- *Training Epochs* - Number of epochs to train through.
- *Decay* - Do, or don't, perform learning rate decay.
- *Normalize Attribute Values* - Do, or don't, normalize feature values.

Property	Type	Values
Hidden Layer Types	string	<b>a, t, i, o, a,t</b>
Learning Rate	double	0.1 – 0.5 (step 0.2)
Momentum	double	0.1 – 0.5 (step 0.2)
Training Epochs	integer	200 – 800 (step 200)
Decay	boolean	true / false
Normalize Attribute Values	boolean	true / false

Table D.12: List of evaluated configurations for Multilayer Perceptron classifier.

The algorithm configuration which, according to the results, performed the best can be found in Table D.13.

## D.9 Best Algorithm Configuration Comparison

The actual comparison of the evaluation metrics achieved by the best configurations of the classifier algorithms can be found in the Table 6.5 in the Chapter 6.

<sup>11</sup>Description of the types can be found in algorithm documentation: <http://weka.sourceforge.net/doc.dev/weka/classifiers/functions/MultilayerPerceptron.html>

<b>Parameter</b>	<b>Value</b>
Hidden Layer Types	a,t
Learning Rate	0.3
Momentum	0.5
Training Epochs	800
Decay	false
Normalize Attribute Values	true

Table D.13: Multilayer Perceptron - configuration parameters with best results.

# E. Attachment 5: Proposed Algorithm Evaluation

This chapter describes the Odalic Tasks used for the evaluation of the new Odalic algorithm. The input data can be found in the “evaluation/new-algorithm/taskX/testing” folder and the training datasets in the “evaluation/new-algorithm/taskX/training” (X is the number of the Task) on the attached CD.

The value of the “sti.tmp.ml.confidence.threshold” threshold which was used for all described tasks was set to 0.5.

The number of possible relations between table columns was determined based on the formula:  $[number\_of\_columns] * ([number\_of\_columns] - 1)$ .

## E.1 Task 1

- *Input File:* 80\_tablets\_shuffled.csv
- *Training Dataset File:* 20\_tablets.csv
- *Number of columns in input file:* 27.

Results of this task can be found in Table E.1.

	<b>True Positive (%)</b>	<b>True Negative (%)</b>	<b>False Positive (%)</b>	<b>False Negative (%)</b>
Column Classification	80	95.45	4.55	20
Subject Columns Detection	50	96	4	50
Relation Discovery	72	99.52	0.48	28

Table E.1: Task 1: Results.

Since the subject column detection and relation discovery phases of the algorithm depend heavily on results of the column classification phase, table E.2 shows results of these algorithm phases in case that the column classification phase was completely correct.

	<b>True Positive (%)</b>	<b>True Negative (%)</b>	<b>False Positive (%)</b>	<b>False Negative (%)</b>
Subject Columns Detection	100	100	0	0
Relation Discovery	84	99.68	0.32	16

Table E.2: Task 1: Results in case the column classification was correct for all columns.



## E.2 Task 2

- *Input File:* 80\_computers\_shuffled.csv
- *Training Dataset File:* 20\_computers.csv
- *Number of columns in input file:* 29

Results of this task can be found in Table E.3.

	<b>True Positive (%)</b>	<b>True Negative (%)</b>	<b>False Positive (%)</b>	<b>False Negative (%)</b>
Column Classification	80	95.83	4.17	20
Subject Columns Detection	66.67	96.15	3.85	33.33
Relation Discovery	77.78	98.96	1.04	22.22

Table E.3: Task 2: Results.

Note: There were four relations evaluated as a false positives which were discovered using the legacy odalic relation discovery algorithm fallback mechanism.

Since the subject column detection and relation discovery phases of the algorithm depend heavily on results of the column classification phase, table E.4 shows results of these algorithm phases in case that the column classification phase was completely correct.

	<b>True Positive (%)</b>	<b>True Negative (%)</b>	<b>False Positive (%)</b>	<b>False Negative (%)</b>
Subject Columns Detection	100	100	0	0
Relation Discovery	92.59	99.41	0.59	7.41

Table E.4: Task 2: Results in case the column classification was correct for all columns.

## E.3 Task 3

- *Input File:* 80\_phones\_shuffled.csv
- *Training Dataset File:* 20\_phones.csv
- *Number of columns in input file:* 32

Results of this task can be found in Table E.5.

Note: There was one relation evaluated as a false positive which was discovered using the legacy odalic relation discovery algorithm fallback mechanism.

	<b>True Positive (%)</b>	<b>True Negative (%)</b>	<b>False Positive (%)</b>	<b>False Negative (%)</b>
Column Classification	100	100	0	0
Subject Columns Detection	100	100	0	0
Relation Discovery	100	99.9	0.1	0

Table E.5: Task 3: Results.

## E.4 Task 4

- *Input File:* 20\_tablets\_renamed.csv
- *Training Dataset File:* 80\_tablets\_shuffled\_renamed.csv
- *Number of columns in input file:* 26.

Results of this task can be found in Table E.6.

	<b>True Positive (%)</b>	<b>True Negative (%)</b>	<b>False Positive (%)</b>	<b>False Negative (%)</b>
Column Classification	80	95.45	4.55	20
Subject Columns Detection	50	96	4	50
Relation Discovery	64	99.36	0.64	36

Table E.6: Task 4: Results.

Since the subject column detection and relation discovery phases of the algorithm depend heavily on results of the column classification phase, table E.7 shows results of these algorithm phases in case that the column classification phase was completely correct.

	<b>True Positive (%)</b>	<b>True Negative (%)</b>	<b>False Positive (%)</b>	<b>False Negative (%)</b>
Subject Columns Detection	100	100	0	0
Relation Discovery	76	99.84	0.16	24

Table E.7: Task 4: Results in case the column classification was correct for all columns.

## E.5 Task 5

- *Input File:* 20\_computers\_renamed.csv
- *Training Dataset File:* 80\_computers\_shuffled\_renamed.csv

- *Number of columns in input file: 27*

Results of this task can be found in Table E.8.

	<b>True Positive (%)</b>	<b>True Negative (%)</b>	<b>False Positive (%)</b>	<b>False Negative (%)</b>
Column Classification	80	100	0	20
Subject Columns Detection	100	100	0	0
Relation Discovery	88.89	99.41	0.59	11.11

Table E.8: Task 5: Results.

Since the relation discovery phase of the algorithm depend heavily on results of the column classification phase, table E.7 shows results of this algorithm phas in case that the column classification phase was completely correct.

	<b>True Positive (%)</b>	<b>True Negative (%)</b>	<b>False Positive (%)</b>	<b>False Negative (%)</b>
Relation Discovery	96.3	99.56	0.44	3.7

Table E.9: Task 5: Results in case the column classification was correct for all columns.

## E.6 Task 6

- *Input File: 20\_phones\_renamed.csv*
- *Training Dataset File: 80\_phones\_shuffled\_renamed.csv*
- *Number of columns in input file: 32*

Results of this task can be found in Table E.10.

	<b>True Positive (%)</b>	<b>True Negative (%)</b>	<b>False Positive (%)</b>	<b>False Negative (%)</b>
Column Classification	100	100	0	0
Subject Columns Detection	100	100	0	0
Relation Discovery	100	99.9	0.1	0

Table E.10: Task 6: Results.