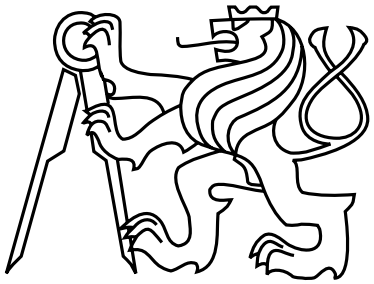
Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science and Engineering



Master's Thesis

Sociopath: automatic local events extractor

Galina Alperovich

Supervisor: Ing. Jan Drchal, PhD

Study Program: Open Informatics

Field of Study: Artificial Intelligence

February 26, 2019

Aknowledgements

I would like to express my sincere gratitude to my supervisor, Ing. Jan Drchal for his patient guidance and immense to help on my thesis. I want to specially thank my husband Simon for supporting and encouraging me in a way that no other ever could.

Declaration

I hereby declare that I have completed this thesis independently and that I have listed all the literature and publications used.
I have no objection to usage of this work in compliance with the act §60 Zákon č. 121/2000Sb. (copyright law), and with the rights connected with the copyright act including the changes in the act.

Prague, May, 2017 .....

Abstract

The Internet is large data source which is mostly unstructured from the semantic point of view. Despite the fact there are many attempts to unify the way how information is presented, there is still no general format for it. For the computer program, it is easy to read the Web page as HTML code, but it's hard to understand the meaning and extract the semantic structure. It makes the automatic information extraction be the challenging problem.

Automatic extraction of the information from Web pages is a common task in data mining. It is used in many modern services and strongly related to the structure of the webpage and the properties of the content itself.

The thesis is focused on Web information extraction about local social events. Social events include various cultural events, sports events, and any other activities.

One of the biggest problems in Web Extraction field is collecting the training data. In this thesis, we presented the approach with the use of Microdata semantic markup for automatic collecting the labeled training dataset. We built the system which automatically collects the training samples with comprehensive features including visual, textual, spatial and DOM-related. Also, this thesis is focused on various techniques on data processing, cleaning and building the classification model for every extracted event component.

Contents

List of Figures

List of Tables

List of Source Code Listing

Chapter 1

Introduction

Information extraction (IE) is a complicated problem, and its goal is the automatic extraction of structural information from unstructured or semi-structured documents. Web Extraction (WE) is a particular case of IE where the corpus of documents are webpages. In the thesis, we are solving the problem of WE for the specific types of the data, namely the local (social) event extraction from a webpage.

1.1 Motivation

Besides the obvious application of WE in the search engine, there are also so-called vertical web services where WE is widely used too. Such websites allow to explore and work with the information on a specific topic. Such services usually do not produce the content by themselves, and rather they automatically collect the data from various sources, processes this data in some way and present to the user in the appealing form. Such services also called *aggregators*. A broad list of aggregators topics includes news, retail other advertisements, reviews, flight tickets, videos and pictures, recipes, social networks and many other. The list is very extensive, virtually for any subject discussed and presented on the Internet there are aggregators exist which try to grasp all other sources into one.

Aggregators are usually very popular because they provide the user with the big database, rich functionality, flexibility and instant updates, what helps a user to save time. Due to popularity of aggregator the 'original' content maker is becoming popular too and therefore the content maker has resources to produce new interesting content. The other side of this practice is traffic reduction on the original content maker websites. It happens for example if the aggregator is unfair and doesn't show the source of information.

The relationship between aggregators and original source makers today reminds the problem of chicken and egg. That's why search engines are very accurate with automatic extraction at the moment of showing the result for the search query. If Google shows you the correct answer right when you type the question, the website with this particular answer where Google took the information from will lose the user. Since this site loses the user, it also loses monetizing and resources to produce the 'correct answers' in future; hence Google

will lose the sources of correct answers. That's the reason why WE among Internet industry players is a controversial topic.

In the research area, WE is typically used in the tasks related to natural language processing and text mining. The reason for this is that the web page content is the semi-structured text mixed with the HTML markdown. Many research project centered around the analysis of various news sources as news articles, tweets, social networks, etc. Some of the possible tasks are sentiment analysis of news, topic recognition, summarization.

Some projects aim to build the system which collects huge amount of data, extract entities from them, determine relationships between them and my answer on human-language question (IBM Watson [**?**], Calais by Reuters [**?**], Wolfram Language [**?**]).

The extraction of data from the web page is closely linked with the Document Object Model (DOM) processing and implies the understanding of the web page structure from the 'browser point of view'.

In this way, WE is becoming a quite interesting, actual and challenging task. It includes both the technical background for the data retrieving and manipulation as well as theoretical background for the reasonable model building. This thesis includes all necessary parts of this task.

1.2 Problem definition

In the thesis, we aim to create the system which automatically extracts the *event components* of the social event from a webpage. The social events might include musical concerts, meetings, performances, festivals, cinema and other social activities. Such events may be published on the different entertainment websites, but usually on a website of the event organizer.

To classify the entity on a web page as an event let's state that it must have the following *event components*:

1. The title
2. The description
3. The date and time
4. Certain location where event is taking place

In this thesis, we won't solve the problem of identifying if the event announcement is placed on the web page since it is the task from Information Retrieval field. We assume that the page already has an event and we want to find and extract the structured information about it (i.e. extract four items above). On the picture 1.1 you can see a typical example of

the page where four main event components are circled in red.

We consider *the webpage* as the rendered page where primary JavaScript already executed (it may be executed in future if the user interacts with a page). Such page has several important components: DOM tree, corresponding CSS, HTML code, resources (e.g. images) and rendered picture which the browser displays us.

For a webpage, we will collect all web elements from a DOM tree, and four of them would be components we need to extract. We will collect more than two hundreds features for every web element and build the model which distinguishes an event components from all other elements. In our problem we will consider four one-vs-all classifiers where each of them will detect associated event component.

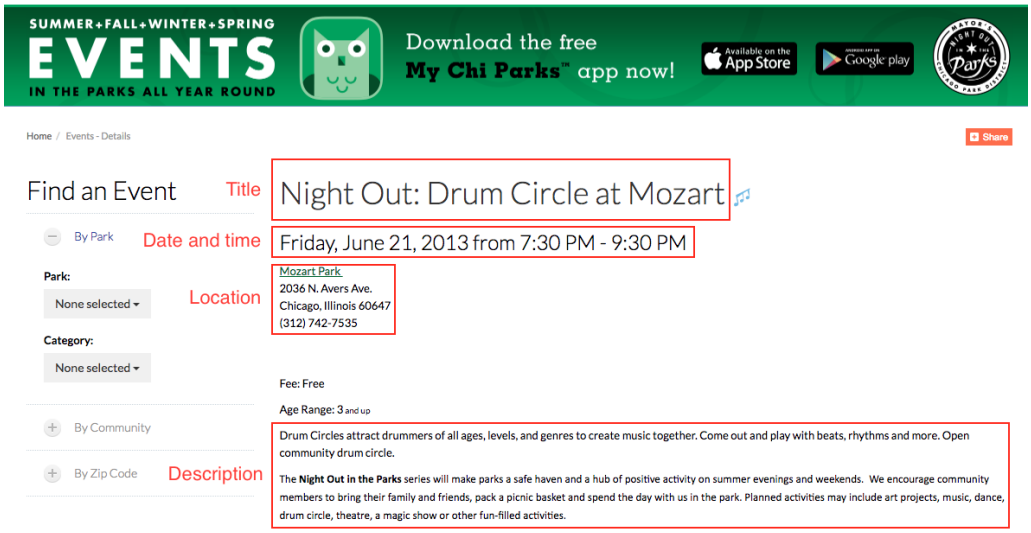


Figure 1.1: An example of the social event on a webpage

1.3 Thesis Outline

The thesis consists of 8 chapters, Appendix and The description of attached CD. The current chapter briefly describes the idea of the thesis and motivation for the task of web extraction. Chapter 2 covers necessary web concepts for this work: webpage, DOM, XPath, Microdata semantic markup, etc. Chapter 3 refers to an overview of state of the art Web Extraction techniques, overview of complementary problems from IE and related work. In Chapter 4 we explain the structure of our system together with the description of its main components. The method which we propose for the collecting of the training dataset will be discussed in details in Chapter 5.

Last three chapters are devoted to an iterative technical process of the dataset analysis. In Chapter 6 we will do several cleaning procedures in order to make the dataset reliable.

In Chapter 7 we show the visualization and precise analysis of the extracted features and its relationships. In Chapter 8 we will build and evaluate several classification models. Conclusion will sum up our most valuable contributions and propose ideas for further work.

Chapter 2

Background from Web technologies

In this thesis, we often operate with such concepts as a webpage, HTML, CSS and JavaScript, web element, DOM, XPath, etc. All these terms refer to the field of web development. In this chapter, we will discuss the meaning of these concepts. We will start from the webpage and its main components. Then we will explain in details the algorithm of web page rendering, so called the critical rendering path. At the end of the chapter, we will talk about the idea of Semantic Web and, in particular, semantic annotations.

2.1 Web page

A Webpage is primarily text document formatted and annotated with Hypertext Markup Language (HTML). A website is a collection of related webpages together with corresponding media files. To view a rendered webpage, we need to make use of a web browser, which provides communication between the user, webpage resources and web server where the associated website hosted. To get the information needed for a displaying a webpage the browser retrieves the data from a web server by sending HTTP requests. We won't discuss the process of requesting, rather we will explain the intermediate steps between the stage when the browser retrieved the information from the server and displayed it to the user.

The access to the webpage might be done directly through the HTTP protocol (for example from the terminal) or with the browser which will render the page. In case we do it through the HTTP request, we will get the raw information the same as a browser gets before it does all complicated prepossessing steps and displaying. In the current thesis, we will use both approaches and take advantage of each of them.

There are two main types of the website and therefore the pages - *static* and *dynamic*. A static webpage is usually written in a plain HTML, whereas dynamic page is using server-side scripts for the content generation. The main points which make the differences between them are scalability, simplicity to update the content and the price. With dynamic websites, it is easy to create a significant number of similar pages and dynamically update the content based on user actions. Dynamic pages are taking it's content from a database, whereas the static pages already have some content which can't be updated quickly. But at the same

time with a dynamic website, it is more complicated to change the design, because the pages are essentially a template into which content is poured to. The design of static pages is more flexible and can be modified manually. As for the price, static website is relatively cheap and easy to make comparing to dynamic websites.

Anyways, after the browser retrieved the information from the server, for both dynamic and static webpages the browser has the same types of information to display the webpage. Only in static case the browser retrieved original files (HTML, CSS, JavaScript, etc) with the predefined content, and in the dynamic case the web server generated these files and took the data from a database.

A webpage as a set of available information contains a lot of different pieces:

- The textual information which we see
- HTML code with the content, hyperlinks, buttons and other components which the user can interact with.
- Static media files, i.e. images, icons, video files, animated GIF, SVG, textual documents and other attached files.
- Dynamic media as Flash, Java applets.
- Semantic meta-information about the content of the page.
- JavaScript scripts which add interactivity and additional functionality.
- Cascading Style Sheets (CSS) files which define how to rendered the webpage and set such properties as a font size, distances between blocks, colors, etc. There are more than five hundreds different CSS properties there [**?**].

2.1.1 HTML, CSS and JavaScript

The triad of the most important Web technologies are HTML to specify the content of web pages, CSS to specify the presentation of web pages, and JavaScript to specify the behavior of web pages [**?**]. Let's briefly discuss how these three major components looks like and show small examples.

Hypertext Markup Language (HTML) - is the standard markup language for creating web pages and web applications. HTML document has the structure of a tree and contains the configuration of the page, its textual content and links to all associated media files. HTML file is parsed by a browser into Document Object Model and we will consider this process in details in the next section. The building blocks of an HTML is an HTML tags (, <title>, <body>) which after the parsing transformed to an HTML elements. A small example of HTML file you can see in listing 2.1:

```
<!DOCTYPE html>
<html>
<title>HTML Tutorial</title>
<body>

<h1>This is a heading</h1>
<div>This is a block of information.</div>

</body>
</html>
```

Listing 2.1: A small example of HTML file

It is possible to incorporate both CSS and JavaScript into the HTML, but in a modern web development, these three things are normally separated into different files. It allows to maintain and change the content, the visualization, and the interactivity features independently.

Since the HTML document has a structure and consists of these labeled blocks (sometimes even semantic ones as <title>), one can consider a web document as a semi-structured document when compared to a plain text document.

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language [**?**]. CSS is primarily designed to enable the separation of content and presentation of a webpage, including layout, colors, and fonts. Such separation improves content accessibility and allows easily change the visual presentation of the page. CSS also allows controlling how the page will be displayed on different screens and devices. There are more than five hundred available CSS properties which can be specified [**?**]. World Wide Web Consortium (W3C) maintain Both the CSS and HTML specifications. The web browser parses CSS files and creates CSSOM tree which is very conceptually similar to DOM. See a small example of CSS file in listing 2.2:

```
body {
    background-color: lightblue;
}

h1 {
    color: white;
    text-align: center;
}

div {
    font-family: verdana;
    font-size: 20px;
}
```

Listing 2.2: A small example of CSS file

JavaScript (JS) is a powerful multi-paradigm language, supporting object-oriented, imperative, and functional programming styles. 95% of 10 million most popular web pages

use JS, and all modern web browsers support it without the need for plug-ins [?]. Scripts are embedded in or included to HTML pages and interact with the DOM and CCSOM. JS can interact and modify both DOM and CCSOM, and it adds interactivity to original static webpages. There is an examples of the JS code in listing 2.3.

Some actions which JS can perform of the page the following [?]:

- It allows loading data from the server without needs to reload the page (AJAX technology)
- It adds animation, interactivity and media content
- It can track the user behavior on the website and collect the information for the web analytic, ad tracking, and penalization purposes

```
1  var canvas = document.createElement('canvas');
2  canvas.width = 100;
3  canvas.height = 100;
4
5  var image = new Image();
6  image.width = 120;
7  image.height = 150;
8  image.onload = window.setInterval(function() {
9      rotation();
10 }, 1000/60);
```

Listing 2.3: A small example of JavaScript code

2.2 How does the browser render the page

Every time the browser retrieves the information from the web server, it makes the series of important actions to display the initial page to the user. This sequence of steps is named *the critical rendering path*. For web developers, it is very important to understand these actions because optimized websites are much easy to use and furthermore, they have a higher rank in search engines. Optimized in this case means that the browser can load and render the page quickly and the internal structure of the code is clear.

The algorithm of how does the browser render the page as follows [?]:

1. The browser retrieves the HTML, read it and see that there are CSS and JavaScripts files for this HTML needed.
2. The browser decides if it's possible to render the page and if yes it loads all necessary resources including CSS, JavaScript and media files.
3. The browser parses the HTML code and builds the *Document Object Model* (DOM) tree. The DOM contains the objects which define the structure and the content of the page.

4. The browser parses the CSS files and builds the CSSOM tree. It is very much like DOM for HTML. The CSSOM contains the objects which define the style of all elements in DOM.

5. JavaScript can modify existing DOM and CSSOM, change any elements of both trees.

6. The DOM and CSSOM trees are combined to form *the Render tree*. Render tree doesn't include the hidden nodes as a script, meta tags, and so on. Also, some nodes are hidden because it was reflected in the CSS files (explicit property "display: none"). Render tree contains only the nodes which required to render the page.

7. The browser is building the layout and computes the exact position and size of each object.

8. The painting stage is when the browser takes the render tree and renders (meaning draws) the pixels to the screen of the user.

On the figure 2.1 you can see how the DOM, CSSOM and Render Tree look like on the 6th step of the rendering algorithms.

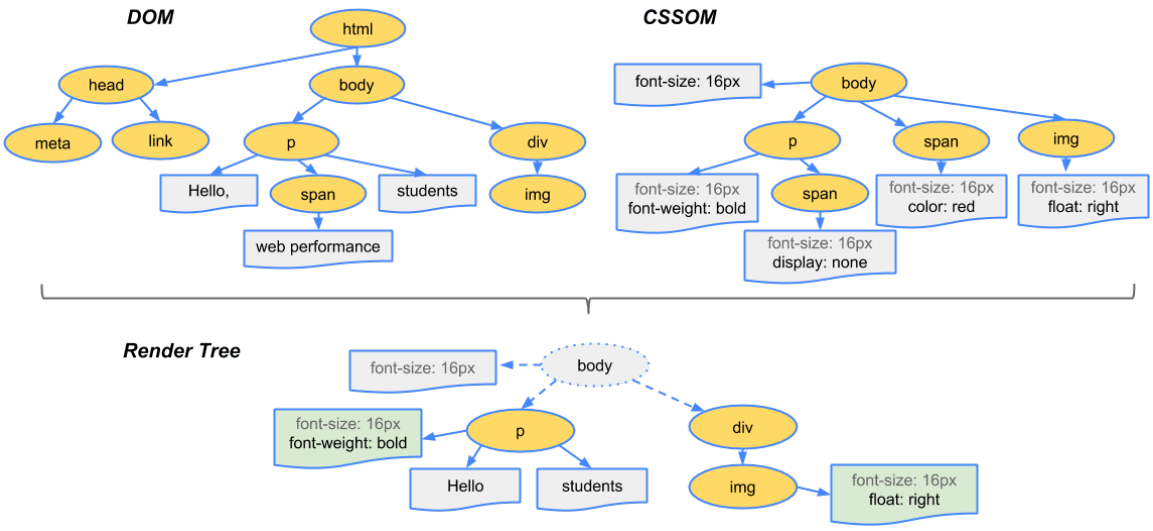


Figure 2.1: The DOM and CSSOM trees are combined to form the Render tree

Source: https://developers.google.com/

2.2.1 XPath and CSS selectors

XPath is a query language for selecting nodes from an XML document based on its tree structure. *CSS selectors* are patterns used to choose the element for styling, and it's also based on a tree structure of an XML document. Since a browser parses and translates an

HTML document to a DOM tree, both the XPath and CSS selectors provide the ability to navigate around the tree and select nodes by a variety of criteria. Both CSS and XPath are the languages developed and recommended by World Wide Web Consortium (W3C). Even though we see the similarity between them and they aim to do the same things, XPath and CSS locators have important technical differences:

- XPath engines are different in each type of the browser. That means this language is inconsistent and expressions must be rewritten for every browser.
- Some browsers even don't have native support for XPath (Internet Explorer).
- XPath has more complex syntax comparing to CSS, which expressions are usually shorter and more clear.
- XPath supports boolean functions which evaluate an expression and returns true or false. CSS doesn't support this functionality.
- CSS is more flexible and have better performance than XPath in all browsers [**?**].
- XPath can traverse down, and up the DOM, CSS can traverse only down.
- CSS fully supports HTML, for example, 'id' and other unique HTML properties of the element can be used [**?**].

A community of web developers prefers CSS selectors over XPath due its simplicity and performance. In our work, we will use both CSS and XPath locators since we will need features of both of them: traversing the tree and boolean function as well as specific element locations. You can see the syntax of both languages on the table 2.1.

2.3 Semantic Web

Currently, the Internet, in general, can be considered as *"Web of documents"*, where every document is an HTML page. With Semantic Web, W3C is helping to build a technology stack to support a *"Web of data"*, where all available data on those HTML pages produce the database and anyone can retrieve that data similarly working with databases. So *the Semantic Web* is an extension of the Web through the list of specific standards and technologies supported by the W3C [**?**].

As we know, today not only the humans read the pages on the Internet. A lot of computers with comprehensive programs read the Internet too, and the easiest example is a search engine. It continuously scans the Web pages, processed an enormous amount of data with complicated algorithms to understand the meaning of the text written in natural language. It is a very complex task to understand the context of the document, its semantic nature and the relationship between included concepts. To do so one needs special software and significant computational power, that's why only big corporations as Google, IBM Watson [**?**], and other can afford it. The root of this complexity lies in the fact that the Web is

Selector	Xpath (1.0 - 2.0)	CSS (1-3)
Whole web page	`xpath=/html`	`css=html`
Whole web page body	`xpath=/html/body`	`css=body`
All text nodes of web page	`//text()`	`NA`
Element E by absolute reference	`xpath=/html/body/.../E`	`css=body>...>E`
Element E by relative reference	`//E`	`css=E`
Second E element anywhere on page	`xpath=(//E)[2]`	`NA`
Element E with attribute A	`//E[@A]`	`css=E[A]`
Element E with attribute A containing text 'A' exactely	`//E[@A='t']`	`css=E[A='t']`
Element E with attribute A containing text 'A'	`//E[contains(@A,'t')]`	`css=E[A*='t']`
Element E whose attribute A ends with 't'	`//E[substring(@A, string-length(@A) - string-length('t')+1)='t']`	`css=xpath=E[A$='t']`

Table 2.1: Comparison of XPath and CSS selectors syntax

Source: Michael Sorens

semi-structured text written in natural language, initially made for simple human needs, not for a computer or complicated queries about the underlying concepts.

The Semantic Web acts as an integrator of different content and systems. The stack of major technologies as follows [**?**]:

1. Resource Description Framework (RDF). It is a language for expressing data models, which refer to objects and their relationships. RDF Schema (RDFS), is a vocabulary for describing properties and classes of RDF-based resources.
2. SPARQ - is a query language for semantic web data sources.
3. Web Ontology Language (OWL), a family of knowledge. representation. It adds more vocabulary and relations for describing properties and classes.
4. XML provides basic syntax for the structure of content.

The biggest problem with Semantic Web is its practical feasibility. While learning of HTML is relatively easy, the theoretical background needed to implement the ideas from Semantic web requires the understanding of knowledge representation subject which is quite complicated.

2.3.1 Semantic markup

Semantic markup (or semantic annotation) is an extension of HTML which implies using special tags with meta information. It helps search engines, web crawlers, and browsers automatically retrieve the concepts with detailed information from an HTML page. Search engine relies on this markup to improve search results. For example, it can automatically show the details of a person or a movie right below the search query in case this information appears with the corresponding meta tags on a reliable webpage.

The list of concepts and its relationships is very broad and includes for example person, address, dates, events, etc. There are three the most popular semantic markup: Microdata, RDF, and Microformat. Also, the last version of HTML (HTML5) has embedded tags which created especially for some semantic annotations. The core of the thesis is using *Microdata* for collecting the initial training set. Let's discuss how does this markup look like and what functionality provide.

Microdata

Microdata vocabularies itself does not provide the semantics or meaning of an Item [**?**]. Instead of this, web developers can create a custom vocabulary or use vocabularies available on the Web. *Schema.org* schemes provide the most commonly used markup vocabularies, which includes: Person, Place, Event, Organization, and many others.

In our work, we regularly use *Event Shcema* vocabulary of the Microdata markup. Let's consider a small example and its primary fields. Look at two listings: plain HTML file 2.4 and the same HTML with Microdata annotations 2.5.

```
<div class="event-wrapper">
  <div class="event-date">Sat Sep 14</div>
  <div class="event-title">Typhoon with Radiation City</div>
  <div class="event-venue">
    The Hi-Dive
    <div class="address">
        7 S. Broadway<br>
        Denver, CO 80209
    </div>
  </div>
  <div class="event-time">9:30 PM</div>
</div>
```

Listing 2.4: Event information without semantic markup

```
<div class="event-wrapper" itemscope itemtype="http://schema.org/
   Event">
  <div class="event-date" itemprop="startDate" content="
     2013-09-14T21:30">Sat Sep 14</div>
  <div class="event-title" itemprop="name">Typhoon with Radiation
      City</div>
  <div class="event-venue" itemprop="location" itemscope itemtype
     ="http://schema.org/Place">
    <span itemprop="name">The Hi-Dive</span>
    <div class="address" itemprop="address" itemscope itemtype="
       http://schema.org/PostalAddress">
      <span itemprop="streetAddress">7 S. Broadway</span><br>
      <span itemprop="addressLocality">Denver</span>,
      <span itemprop="addressRegion">CO</span>
      <span itemprop="postalCode">80209</span>
    </div>
  </div>
  <div class="event-time">9:30 PM</div>
</div>
```

Listing 2.5: Event information annotated with Microdata

As you can see, the second code block looks more cumbersome because it has a lot of additional annotations.

- **itemscope** – creates the Item and indicates that descendants of this element contain information about it [**?**].
- **itemtype** – a valid URL of a vocabulary that describes the item. In our case it is "http://schema.org/Event" and if you go by thins link, you will find all possible fields and information.

- **itemprop** - contatining tag holds specific property of the item, for example "location", "startDate", "name".

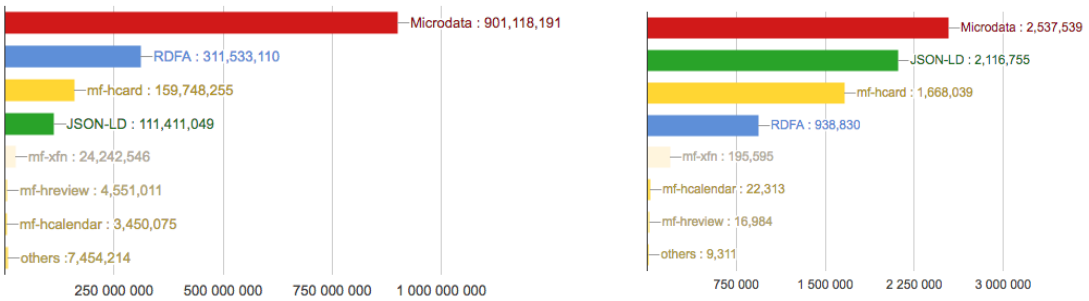
There are dozens of Event properties in "http://schema.org/Event", but in the thesis we consider only four main and the most popular properties "name", "startDate", "description" and "location". We will call them *event components*.

You can mention, that the item can contain a property as well as another item. For example, property "address" includes the item type "http://schema.org/PostalAddress." In the thesis, it would be an additional difficulty during the parsing process because we will need to invoke not only the property but also the properties of all included items to collect the event components. We will discuss it in Chapter 5.

2.3.1.1 Semantic markup popularity

Search engines take advantage of the semantic markup to automatically extract the information, incorporate it into its vertical services and provide a richer browsing experience for users. Despite the fact that search engines encourage and promote the websites which use such markup, in fact, not all of them use it. By statistics only 16% of the domains use semantic markup, the rest 84% use standard HTML. That's also the reason why the task of structural information extraction from unstructured web pages is still relevant - most of the data on the Internet is still stored as a plain HTML.

We can collect popularity of semantic annotations from Web Data Commons web service. In the fourth quarter of 2016, it crawled 34 million pay-level-domains and detected that there are 5.63 million (16.5%) domains use RDFa, Microdata or Microformat semantic markup on their pages. See the figure 2.2 to explore the numbers. As you can see, Microdata is leading, that's the main reason why we considered to use this annotation in the thesis.



(a) Across URLs (b) Across domains

Figure 2.2: Popularity of semantic annotations, 2017

Source: Web Data Commons

Conclusion of the chapter

In this chapter, we considered the most important concepts which we use in the thesis. We explained the structure of a webpage, discussed related terms like HTML, CSS, and JavaScript, discussed in detail how does the browser render a webpage and the notions of CSS and XPath locators of the web element. Also, we talked about the idea of Semantic Web, its prevalence and described Microdata semantic annotation which we use for collecting the training dataset.

Chapter 3

Web extraction

Since we know the central concepts related to a webpage and its structure, we can talk about the Web extraction (WE) problem in detail. In this chapter, we will discuss modern techniques used in this area such as crawlers, wrappers and also list the complementary problems as named entity recognition and region extraction. At the end of the chapter, we will discuss related work to the thesis.

Web extraction (also Web Scraping, Web harvesting) is the extraction of structured information from a webpage presented as an HTML, i.e. semi-structured text. It is important to highlight that by saying that we *extract* the information from a webpage, we assume the information is located on the page and readable.

3.1 Information Extraction

The goal of Information Extraction (IE) is an automatic identification and extracting structured information from unstructured or semi-structured documents. Most of the time, in IT one processes human language. The information needed to be extracted must be explicitly specified. IE is a non-trivial task due to the complexity of natural language, and even for a limited domain implies a lot of work.

IE consists of several sub-tasks [**?**]:

1. *Named Entity Recognition* (NER) refers to the problem of the identification and classification of predefined types of named entities, such as organizations, persons, place names, time, etc.

2. *Co-reference Resolution* (CO) requires the identification of co-referring mentions of the same entity.

3. *Relation Extraction* (RE) detects and classifies predefined relationships between identified entities.

4. *Event Extraction* (EE) refers to an identifying event in text and deriving structured information like who did what to whom, when, where, through what methods, and why. Don't confuse this sub-task of IE with the Event Extraction in the thesis, where we consider social events rather that relationships between entities in the natural language text.

These four sub-tasks can be considered as four well-known problems: NER refers to a segmentation and classification, RE is the association and EE is a clustering [**?**].

3.2 Web extraction methods

WE can be seen as a particular case of IE where the entities are extracted not from a plain natural language text, but rather from a semi-structured HTML page. WE implies several additional sub-tasks specifically related to a webpage structure. For example, Region Extraction (RE) helps to filter web elements to focus only on relevant information. Also, since WE works with an HTML, it must entail its structure and process the tree rather than plain text.

World Wide Web Consortium (W3C) is an international standard organization tries to get all the players on the Web implement the same set of core principles and components. That is why all websites use relatively the same stack of technologies, and browsers seek to follow all changes and support major updates in traditional web tools.

Despite the fact, web developers use the same tools, HTML pages of different websites most of the time are completely different. Every site is following its design and structure, presenting the information in a different way for the particular audience of the website. That means if one method of WE works well on a specific site, it won't probably work for other sites with the same information. For that reason, WE needs to solve the problem of heterogeneity of the websites, which is the most difficult part of WE field.

3.2.1 Extracting with XPath and CSS

One of the popular methods of WE from a webpage is using XPath and CSS selectors for identifying and extracting the information. The algorithm is quite simple: you look to an HTML code, find those blocks which you need to extract and compose correct XPath or CSS selectors for these web elements. After that, you can obtain the information on any webpage where such path in a DOM tree exists. This method is the most appropriate and fast if you have one website with a big number of static pages with identical structure and design.

In our thesis, we use this approach as one of the steps of dataset collecting. We use Scrapy Python framework [**?**] for identifying semantic Metadata tags and crawling the text corresponding to each tag. See an example of code with Scrapy in the listing 3.1.

```
from scrapy.selector import Selector
import urllib.request as request

def get_item_types(url):
    item_types = []
    page = request.urlopen(url)
    if page.code == 200:
        html_body = page.read()
        selector = Selector(text=html_body, type="html")
        elements = selector.xpath('.//*[@itemscope]')
        for item in elements:
            item_type_text = item.xpath('@itemtype').extract()
            item_types.append(item_type_text)
    return item_types
```

Listing 3.1: Example of meta tag text scrapping with Scrapy Python framework

The code is self-explainable, but let's briefly discuss the main steps. Firstly, you open the webpage with urllib, that's where we send an HTTP request. If the status code is 200, meaning the positive answer from a server, we read the page. We create Scrapy Selector, and in this case, we use XPath locator. We are identifying all elements which have 'itemscope' attribute meaning we want to extract all elements which tagged with any semantic label. Then in a loop we collect all types of these semantic tag names, for example 'http://schema.org/Person', 'http://schema.org/Movie', 'http://schema.org/Event', 'http://schema.org/Place', etc. The name of the Metadata tag takes the form of URL for simplicity.

Obviously, the problem with this approach is no scalability, because the path in a DOM for different websites would be different.

3.2.2 Wrappers

Wrapper is a general name for a program for data extraction from a webpage. In the practical application such programs are also called *web crawlers* (or web spiders, web scrapers). Typically, wrappers are implemented manually by a programmer, and she needs to create a new specified wrapper for every document which has to be processed. The output of a wrapper is a structured information in some machine readable format.

Wrappers can be divided into three segments depending on the level of automation:

- Manual wrapper: the user observers a webpage, its source code and writes down a program code to extract data (for example, using the XPath and CSS).
- Wrapper induction: supervised learning where extraction rules are learned from a manually labeled data records [?], [?], [?], [?], [?] [?] [?]. The inductive learning process is the generalization of wrappers that are obtained from human labeled documents.

- Automatic wrapper induction: unsupervised learning with the use of tree matching algorithms to find repetitive patterns on single or multiple pages [?], [?], [?], [?], [?], [?]. Automatic wrapper induction is possible because a lot of web objects follow fixed templates.

Typical usage of the wrapper as follows: a user defines set of desired attributes and indicates associated fragments of the example page. Then the program learns a wrapper for the current website. After it is done, the program uses the learned wrapper to extract new information from the same source automatically. Therefore, every step of this process may be automated to some degree.

Wrappers usually work well on template-based web pages, for example, a list of addresses, product catalogs, and telephone directories. There is a problem with a more unstructured page with custom design, but this kind of webpages are common on the web.

3.3 Adaptive Information Extraction

Adaptive information extraction methods can handle less structured webpages. There are three main approaches:

1. First approach considers extraction task as a Classification problem which has the learning and prediction stages. For this method, one needs to have a dataset with labeled entities which needed to be extracted.
2. In the Sequential Labeling approach, a webpage is viewed as a sequence of words (tokens). This method can model the dependencies between target entities on the page while the classification task considering each object independently.
3. Visual Information Extraction relies on rendering a webpage in a browser. This approach helps in extracting objects from complex web pages that may exhibit a visual pattern, but a lack of patterns in the HTML code.

The current thesis falls into the Classification-based approach with one exception: we generate the training dataset automatically and don't imply human labeling procedure. Also, we use visual features from rendered webpage, means use ideas from Visual Information Extraction approach.

More extensive surveys on different methods and works in Web Extraction field can be found in [?], [?], [?].

3.4 Related problems

3.4.1 Information retrieval

IE is closely related to Information Retrieval (IR). The task of IR is to select from a collection of textual documents a subset which is relevant to a particular query, based on key-word

search and possibly augmented by the use of a thesaurus [?]. A simple example of IR application is a search engine websites. You set the query, and as an output of IR algorithm, you have a list of relevant documents, ranked in descending order. In general case, there is no extracted answer from a corpus, only the documents where the answer might be. To obtain the facts or meaningful structured information from a document one needs to understand the semantic of the text on a webpage. And that's the primary goal of IE.

IE task considered to be harder than IR. However, IR and IE techniques are complementary and may be combined in different ways. IE often uses the IR as a filtering step, because usually, the corpus of texts is very large. IR often uses IE for the feature extraction. See figure 3.1 to see the difference between IE and IR.

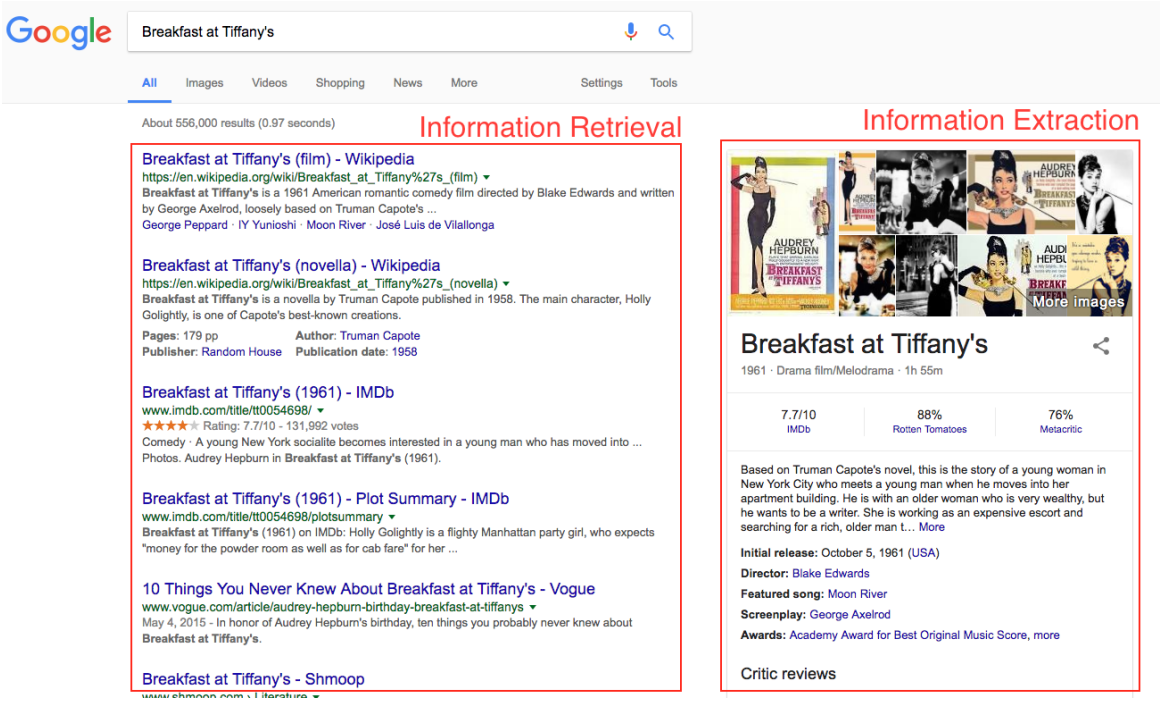


Figure 3.1: The difference between Information Extraction and Information Retrieval

3.4.2 Named Entity Recognition

Named Entity Recognition (NER) is the task of extracting entities, such as proper name (person, location, and organization), time (date and time), and numerical values (currency and percentage), from the source text, and mapping them into predefined categories, such as person, organization, location name or "none-of-the-above" [?]. The two most important components of NER are *to find* the entity and then *classify* this entity.

NER is an essential sub-task of IE and refers to a Natural Language Processing field.

The uses of NER as follows [?]:

- Named entities can be indexed, linked off, etc.
- Sentiment can be attributed to companies or products
- A lot of IE relations are associations between named entities
- For question answering, answers are often named entities.

There are three standard techniques to solve NER:

1. Hand-written regular expressions.
2. Using classifiers:
    - Generative: classification algorithms to classify substrings of document as "to be extracted" or not [?].
    - Discriminative: Maximum Entropy Modeling [?]
3. Sequence models relies on sequential nature of natural language text:
    - Hidden Markov Models [?]
    - Conditional Markov Models and Maximum-entropy Markov model [?]
    - Conditional Random Fields [?]

3.4.3 Region extraction

Web documents are getting more and more sophisticated, which complicates the task of IE. The goal of Region Extraction (RE) is filtering irrelevant information and help WE to focus only on important data records.

The difference between IE and RE is that IE focuses on extracting structured data about specified entity with their attributes, whereas the RE identifies the HTML block which contains this entity.

All region extraction methods rely on the repetitive structure of DOM tree of a webpage. The majority of region extractors are unsupervised and use tree matching, string matching, and clustering.

Mining Data Records (MDR) RE system [?] is based on the hypothesis that repetitive data records are usually rendered inside tables and forms. Embley et al. [?] extract the largest unique region in a web document, supposing that this region contains relevant multiple data records. Yi et al. [?], Wang and Lochovsky [?] and Kang [?] showed that RE has a positive impact on both efficiency and effectiveness in IE. Vision-based Page Segmentation

(VIPS) [?] built on the idea that web designers visually distinguishes the major blocks which need to be extracted. Kohlschutter [?] utilizes the notion of text-density as a measure to identify the individual text segments of a web page.

The extensive list of methods on RE can be found in [?].

3.5 Web extraction as a service

Since the task of automatic web extraction might be very useful, there are many commercial applications written for solving this task. There are both desktop and web, free and paid applications with different functionality for the extraction of specific entities. Let's consider those services which provide this functionality as an API.

Such applications usually offer their services on a paid basis (per number of requests) and allow to use it from any other application through HTTP requests. These APIs typically extract only a specific list of structured information. So it might be an API only for reviews or for example, only for product description in the online store. Below you will find popular web services and their main features.

Diffbot's Automatic APIs [?] automatically extracts the content from supported page types: articles, discussions, images, video and products. It combines a variety of comprehensive techniques such as computer vision and machine learning. It accurately extracts clean and structured data from a webpage to JSON format. One particular thing about Diffbot is that it can parse text in different languages because of extracting visual features, not only text content.

ParseHub [?] parses the structured data after one example given by the user. It has an interface where the user needs to select several elements which she wants to parse, compose the rule using the visual builder and then run this scenario for the entire page. ParseHub automatically explores all elements which the user meant and parse it to JSON format. After that, this script is available as an API.

Apifier [?] is similar to Diffbot extracts structured information for specific topics as a product, social networks details, booking hotel websites and search engine. It presents the data in JSON format.

IBM Watson [?] has an Alchemy Language (part of Natural Language Understanding API) component which analyzes the data on a webpage, extracts specific entities such as people, places, and organizations with NLP methods and answer questions about these entities and its relations. IBM Watson is a huge project and includes a lot of different components related to natural language understanding, speech, and visual recognition.

Other services. There are dozens of web services which offer article content extraction, especially the name and description. Also there several interesting services which extract

entities and analyze the relationships based on unstructured plain text: Ambiverse Natural Language Understanding [?], Google Cloud Natural Language [?], Calais by Reuters [?]. In these projects entities have types such as a person, location, organization, product, etc.

3.6 Related work

There are not much work about social event extraction. The closest work is made by Google Research [?] which similarly to us used semantic markup together with a large human-labelled dataset. Petrovski et al. [?] use Schema.org annotations for product ads to learn regular expressions. Gentile et al. [?] work on dictionary-based wrapper induction methods that learn XPaths using linked data. Becker [?] was identifying and characterizing events in social media channels. Gogar et al. [?] presented an automatic method based on the deep neural network to extract information from e-commerce websites.

Conclusion of the chapter

In this chapter, we discussed theoretical background in IE and WE fields, reviewed the main works and presented related problems as region extraction, named entity recognition and information retrieval. Also we listed several commercial IE web services and briefly explained how they work.

Chapter 4

Design and Implementation

This chapter will present the design and technical details on the implementation of Sociopath event extraction system. We will recapitulate the information on Microdata from the chapter 2 and explain how it is relevant in the context of event extraction process. Also, in this chapter, we will show the scheme of the project and describe the building blocks. In the end of the chapter we will list technologies and tools which we use in the thesis.

4.1 Methodology and Approach

As we mentioned in Introduction, the Sociopath system aims to extract the following four event properties from the web page:

1. The title
2. The description
3. The date and time
4. The location where event is taking place

As also said we assume that the page already contains an event announcement, and we don't solve the task of information retrieval to list those pages which contain the events. Our goal is to find and extract the structured information about the event knowing it's there.

We consider the event extraction problem as several binary classification tasks, where every model runs on every element of the webpage and makes its decision. In our approach, we explicitly build one-vs-all classifiers for every event component. Thus we have four different models that independently process the elements of the web page.

On the web page, there is a significant number of elements which are not relevant by default because they can't be an event property. The list of such elements includes advertisement blocks and pop-ups, footers, sidebars, media and other items which are expected to be on a regular web page. We don't want our classifiers to spend computational time, so we applied the filtering procedure to leave only relevant web elements for further analysis.

About the dataset

For building the meaningful statistical model, one needs to have a relatively large training set. As it we said before the collecting of such dataset is one of the biggest problems in Informational Extraction and often implies human involvement into labeling procedure. In our case, we must have a big set of web pages with event announcement for which we know exactly where every component is located.

In chapter 5 we will discuss the collecting process of training dataset. Briefly speaking, we exploited the set of web pages which annotate their HTML code using semantic markup Microdata with the schema *Event* from Schema.org vocabulary system. We gathered 80GB of URLs with Microdata annotations from the service Web Data Commons and then started the crawler for feature extraction. The process of crawling was the most challenging part of data collection. Firstly, because we had a big number of URLs to parse and many of them were already unavailable or damaged. Secondly, the extracting the features takes around 30 seconds for every page. To deal with these difficulties, we built robust and continuously running parallel parser on *MetaCentrum* [**?**] distributed system.

4.2 Architecture of the system

The system is composed of 4 logical modules:

- **I. URLs Collection** - download data from Web Data Commons, clean it, extract the URLs with Event Microdata semantic markup.
- **II. Creating a dataset** - send these URLs to MetaCentrum and set parallel crawling process for extracting web elements of the event components and corresponding features.
- **III. Dataset cleaning** - processing of the raw data, in the end, we'll have a final dataset for analysis.
- **IV. Analysis and modeling** - data exploration, analysis, building and evaluation of models.

On the picture 4.1 you see the detailed schema of the application. Below you will find the description of every component from this schema.

Common Crawl is a project which collects and maintain the most comprehensive public corpus of web crawl data. Common Crawl's web archive consists of more than 250 TiB of content from more than 3 billion webpages [**?**]. It completes the crawls every month, and the history is available for the last eight years. The corpus contains raw webpage data, extracted meta-data, and text.

Web Data Common (WDS)[**?**] is built on Common Crawl. It extracts the structured data (RDFa, Microdata, Microformat, and Embedded JSON-LD) from the Common

Crawl corpus quarterly and shares the dataset for public download to support researcher and companies.

1. URLs extractor. This component takes as an input the data from WDC which is a huge text file which consists of N-Quad rows. An example of such format you will see in the section 5.2.2. The goal of URLs extractor is collecting only URL addresses of those pages which contain the *Event* Microdata semantic markup together with all its successor as *SocialEvent*, *SportsEvent*, etc.

2. URLs cleaner. This component ensures that all URLs exists, available, and have the correct structure. After we collected the URLs, we put them in the storage of URLs chunks. Every piece consists of 100 URLs with Event Microdata. We keep the data like this in the **3. List of URLs** component because it would be more convenient to process later.

4. Multiparser and **5. Raw dataset**. The Multiparser is the most time-consuming part of the workflow. It is a parallel program which runs several independent processes for crawling the URLs. Every crawler does three things: loading, feature extraction, and features writing. The raw dataset contains two types of records: features of the event components and features of the random elements of the page to collect negative examples too. We run the parser on MetaCentrum distributed computer system.

6. Data loader and **7. Data cleaner**. Data loader component loads the raw data from the MetaCentrum through the SSH connector. Data cleaner executes extensive cleaning procedure such that the number of records has decreased ten times. We will discuss the cleaning procedure in chapter 6.

8. Final dataset. The raw dataset contains event components and random elements together with associated web features and identifiers.

9. Exploratory data analysis. In this part, we visualized and learned every feature, its basic statistics, the form of distribution and its specificity. Also, we experiment with dimensionality reduction methods.

10. Feature engineering and selection. On this stage, we made a feature engineering and created features that can be divided into three parts: spatial, textual and visual. Also, we run several Random Forests classifiers to reveal the importance of the features.

11. Model building and 12. Evaluation In this part we experimented with different classification models, optimized their parameters and compare results on previously unseen data.

4.3 Tools

Here is a list of the primary tools and frameworks which we used for building the Sociopath event extraction system. All used tools are free and open-source.

Used programming languages: *Python* for all processing, parsing and data analysis procedures; *JavaScript* for DOM manipulation inside the virtual browser.

Python stack: *scikit-learn, pandas, numpy* libraries for data manipulation and model building; *NLTK* toolkit for textual features engineering; *xgboost* framework for boosted trees algorithm; *multiprocessing* library for parallel parsing process.

Webpage crawling: *PhantomJS* virtual browser; *urllib2, Scrapy* frameworks for retrieving data from a URL webpage with CSS and XPath selectors.

Visualization: *Matplotlib, Seaborn and Bokeh* - Python libraries for dataset visualization, *draw.io* - web service for creating diagrams.

Computation: *Local computer* – Mac OS, Processor 2,7GHz Intel Core i5, RAM 16 GB; *Remote computation – MetaCentrum* virtual distributed computing infrastructure (free for CTU students).

SSH connection: *Paramiko* for Python code, *Cyberduck* and terminal for a desktop.

IDE: PyCharm IDE and Jupyter Notebook.

Conclusion of the chapter

In this chapter, we presented the scheme of the Sociopath event extraction system, and briefly explained its main components. In the next sections, we will cover all parts of the system. Also, we provided the list of technologies and tools which we actively use during the work.

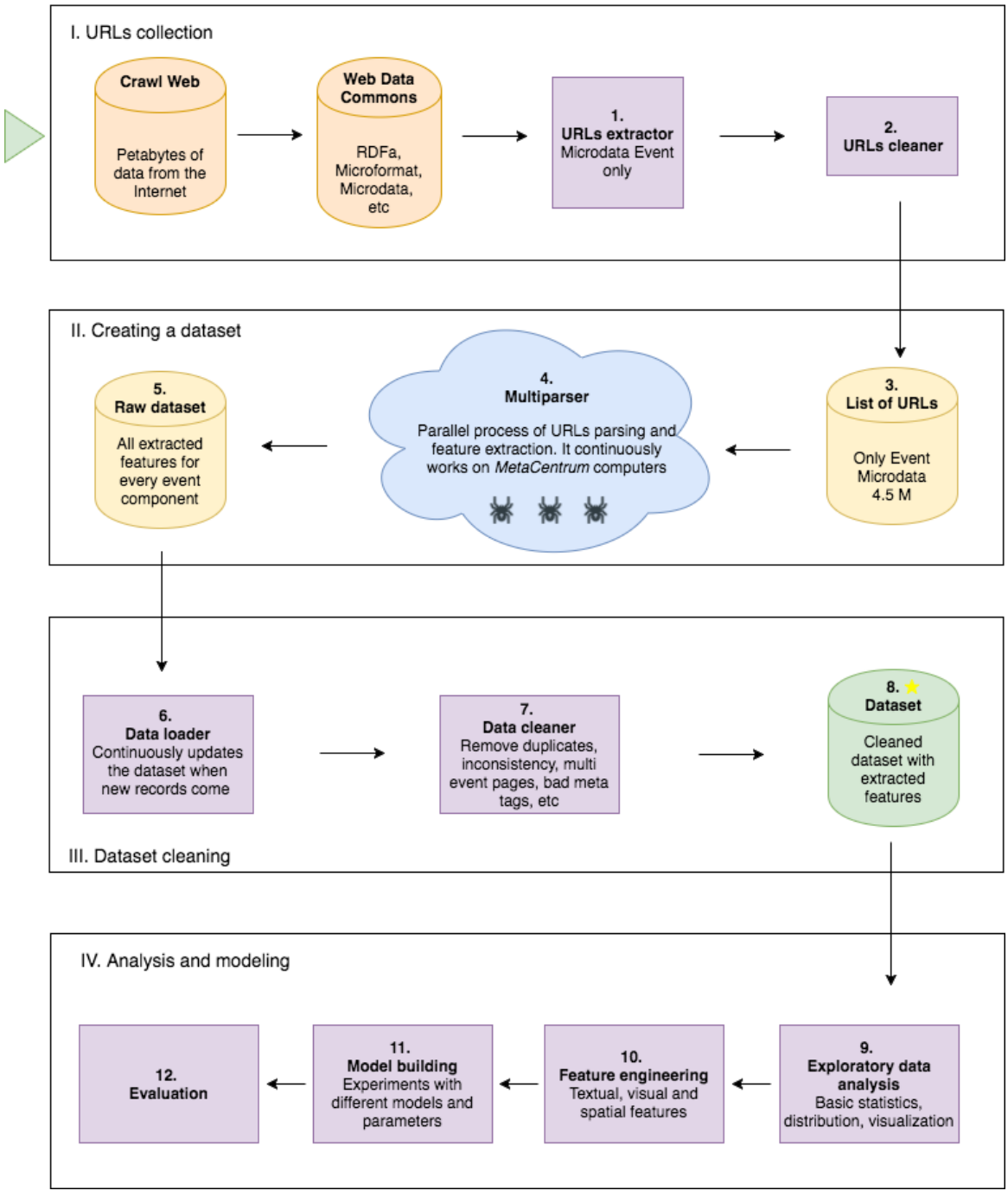


Figure 4.1: The components of Sociopath event extraction system

Chapter 5

Collecting the data

To build the significant statistical model one needs to have a relatively large training set. Usually, collecting the training set is the main stumbling block in the tasks related to Information Extraction (IE) and often implies human involvement into labeling procedure. For solving the problem of Event Extraction (EE) from the webpage we need to create or find publicly available well-labeled dataset.

Firstly in this chapter, we will discuss requirements for the dataset we need. Then we will present the method which we used to create the dataset using Microdata semantic markup. Then we will talk about the process of URL crawling and event web feature extraction for a collecting the dataset.

5.1 Requirements for a dataset

As we said, collecting the labeled dataset in IE is hard. As an example, Google hired several particular companies where hundreds or thousands of people are labeling the data as the full-time job[**?**]. Search engines and big corporations which work on this task can afford the large labeled dataset with the high quality, but it is much more challenging for researchers.

The rule "garbage in, garbage out" is saying that the quality of the learning algorithm is directly tied to the quality of data it takes as an input. Since we consider Event Extraction (EE) task as a set of binary classification tasks, it is critical for us to have representative and relatively large dataset for building the model.

In our case the ideal training dataset would look as follows: for one URL we want to have four main *event components* (title, date time, description and location) together with their *web feature* values and place identifiers on a webpage. XPath or CSS selectors are one of the 'location' identifiers of the web element on a page. We discussed it in detail in section XPath and CSS selectors.

By the *web features* we mean the values describing the event component as a web element on the page. The list of such features might include a tag, text of the element, the position

on the page, number of children in a DOM tree, the CSS property of the font size, etc. All these features in one way or another characterize the meaning of the element, knowing that an event announcement is actually on the page. We consider the task of IE, and we imply that the event is on the page, but we don't know where exactly its components are placed.

There are various types of design and structure of an event page. The event components might be located in different places relatively to each other. Furthermore, there are different HTML code styles, which depend on the developer who programmed the front-end, used web frameworks and technologies. To build the model which will work on previously unseen web pages, we must train the classifiers on many various pages to consider all these cases. That's the reason why we didn't consider the option of collecting the training dataset through the human labeling process.

Therefore, the list of requirements for a dataset we need is the following:

1. It is free
2. It is relatively big (thousands of rows)
3. It reflects different visual structure of the webpage with an event announcement
4. It has significant amount of constructive features: spatial, textual and visual and DOM-related
5. It is fairly fresh (the webpage should be available)

There is one non-free well-formed dataset of labeled webpages named *The ClueWeb12 Dataset* [**?**]. Researchers in related projects have been using these datasets and showed the high performance [**?**]. In this thesis, we didn't consider paid datasets as an option. Rather we set a goal to create a technique for collecting such dataset automatically.

There are also several datasets with annotated webpages from other fields, for example University dataset [**?**], Internet Advertisement [**?**], Interested or not web pages [**?**], Phishing websites [**?**], Rumor dataset [**?**], Amazon product review [**?**], Real state from website [**?**]. There is no publicly available dataset for event extraction problem. One of the goals of this thesis is to create such dataset and shared with community.

5.2 Creating a dataset

To generate the dataset for training, we exploited the set of web pages which format their HTML code using the semantic markup Microdata together with the schema *Event* from Schema.org, which we discussed in detail in section Microdata. If we know the URLs of such pages, it is relatively clear how to extract this information. That means that first of all, we need to find these pages and select only those which contain an event announcement. As we said, there is only small part of websites using semantic markup (16%). Therefore the problem of gathering the pages with particular schema is not simple especially if we want to collect the rich and diverse dataset of URLs.

5.2.1 Collecting URLs

Fortunately for us, we discovered the great open repository *Common Crawl*, which maintains and collects the web crawl unstructured data. Common Crawl is the largest web corpus available to the public, which harvests a vast amount of data from various Internet pages. To find the pages which contain the Event Microdata markup would not be feasible if it wasn't for another great open repository *Web Data Commons* (WDC) which in fact does what we exactly need. It processes the Common Crawl entire database and selects only those URLs which contain any structured data.

To get the clean list of such URLs, we found 80GB file associated with Event schema on WDC and downloaded it. The content of the archive is very verbose and contains a lot of unnecessary information. So we processed the file by selecting only URLs for the *Event* schema and its successors as *SocialEvent*, *SportsEvent*, *TheaterEvent*, etc. The size of the list of URLs reduced to 400 MB.

5.2.2 URLs crawling

The most time-consuming part of data collection was the parsing the list of URLs. To do so, we developed a parallel Multiparser program and set it on MetaCentrum for continuous execution. The parser is processing the files by 100 URLs, and for every URL it is making the same sequence of steps:

Step 1. In the first step the parser goes to the webpage and tries to find all event components. Here we use *Scrapy* framework and explicit HTTP requesting, that's why it's a fast operation. One of the problems we face here is that an Event item may contain not only properties like Date, Location or Description. It also might have another item inside, for example, Place, which has the same Location property. In this way, we had to collect event properties such that they will not intersect with each other. We did it wit the specially constructed XPath locator.

Step 2. If it succeeds in a previous step, then it runs a virtual *PhantomJS* browser on this page. Otherwise, it starts to process a new URL. We need to render the page with the browser to extract visual and spatial features what can't be done with a simple HTTP requesting.

Step 3. The virtual browser opens a page and renders if fully - including images, CSS, even Ajax-delivered content. It is more advanced that using HTTP requesting because we have access to a visual layout and DOM, CSSOM trees.

Step 4. The parser extracts all web features through the executing JavaScript in a browser. This step is the longest one, it takes from 30 seconds to 2 minutes to render and extract all features for one URL. Here we had difficulties with JavaScript coding. The reason for this is that it wasn't possible to debug the code which is being executed in a virtual browser; the

browser doesn't provide with such API.

Step 5. The crawler writes its result to the file.

For every step of a crawler, we set the time limit because sometimes it's processing too long and holding everything back. To control the execution time, we needed to run separate Python process for every time-consuming action. The whole crawling process has been ongoing for three months.

The result of all these steps is a dataset where every row consists of a URL, event component name and the values of corresponding features. To summarize, see the scheme of dataset collecting process on figure 5.1.

Conclusion of the chapter

In this chapter, we focused on collecting the dataset process. We talked about gathering the URLs from Web Data Commons service and comprehensive parallel parsing process, which we run on MetaCentrum. Also, we discussed requirements for a training dataset and presented the list of main features which we extracted.

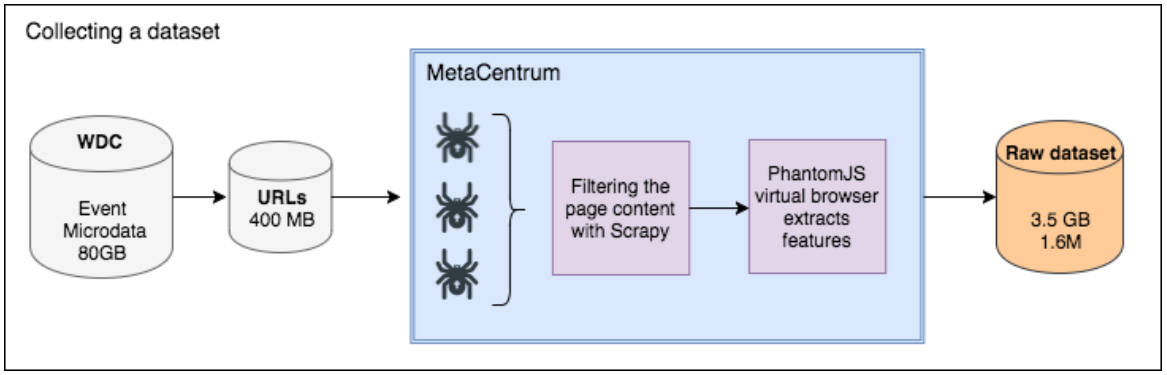


Figure 5.1: The scheme of dataset collecting process, the final number of records is 1.6 million

Chapter 6

Experimental part: Data cleaning

In this chapter, we will present the process of the raw dataset cleaning. As we discussed in previous sections, we extracted raw web features separately for event components (name, date, location, description) and random elements of the page. We aim to construct train dataset with both positive and negative examples for every event component to build binary classifiers.

This chapter opens the block of the chapters which cover the Experimental part of the thesis. All corresponding Python code may be found in corresponding Jupyter notebooks. In this chapter, we cover the main stages and present the scheme of the whole cleaning pipeline. Also, we will show how the raw data is melting, meaning after the every step of the cleaning process the amount of data is reducing dramatically. The less significant results will be available in Appendix chapter. The whole process together with the corresponding code you may see in the notebooks with the prefix *Clean*.

The process of big dataset cleaning usually is very time-consuming, and in our thesis, it was not an exception. The reason for this in our case is that the dataset is artificial and collected from the Internet by our crawlers. During the collecting stage, we could face problems with the web element availability, encoding, duplicates, missing values and all other natural effects. We separated the whole process into four logical components. In the first part, we remove the rows which have a visible damage. In the second part we look closer and work individually with the CSS properties since there are more than 200 features and not all of them are relevant for us. In the third part, we work with *frequent domains* which have a lot of similar URL pages. In the last part, we worked with the outliers and logically not-existing values of the features.

On the picture 6.1 you may see the main stages of the cleaning process. On the upper part of the picture, you find five main steps. In the bottom of the picture, you can see the number of records left in a dataset after each step. After step "CSS properties" nothing changed because at that moment we reduced the number of columns, rather than rows. Let's discuss every part of this scheme.

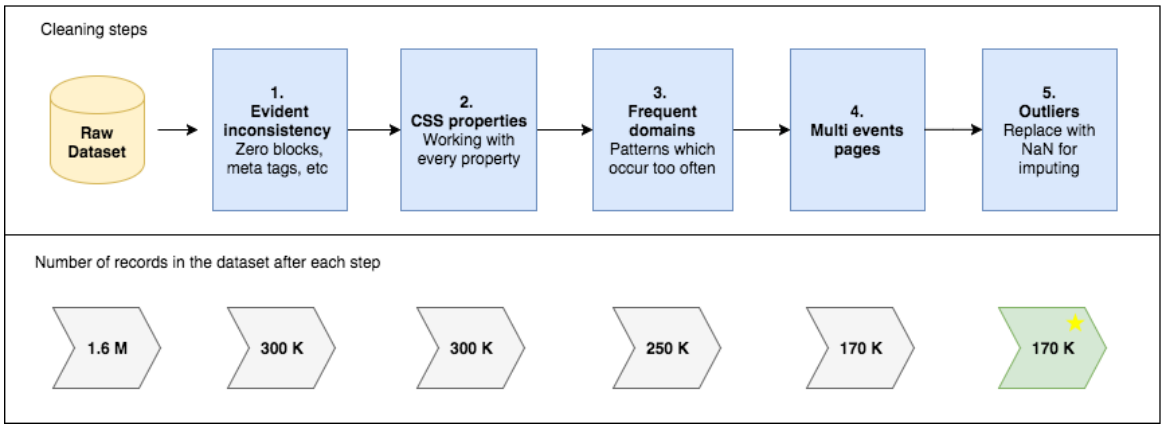


Figure 6.1: Raw data-cleaning workflow

6.1 Obvious Inconsistencies

When we uploaded the data, first of all, we worked with several types of inconsistency which are easy to detect and which usually touch a lot of data:

- Duplicates of rows by numerical features.
- The 'meta' tag. Developers often set the meta information with semantic schema not in the associated real web element, but rather to an artificial auxiliary 'meta' tags. These artificial tags don't fall into a render tree because they are not visible. That means the data like this is unnecessary because it doesn't contain any visual relevant information we extract. *After this filtering, the dataset became approximately four times smaller.*
- Zero block width and height.
- Zero X and Y coordinates
- Empty URL

Corresponding code for this step you may see in the notebook with the prefix *Clean I.*

6.2 Working with CSS properties

In out Raw dataset we have 288 CSS features from a virtual web browser. To get these properties the browser rendered the page, so we have real post-processing values of CSS fields. Many of properties have prefix *-webkit*. Webkit is a rendering engine used by Safari, Chrome, and other browsers. The *-webkit* prefix on CSS selectors are properties that only this specific engine is intended to process. Due to inconsistency in a browser rendering engines, developers need to set multiple rules for the same thing for all popular browsers, that's the reason why web developers community is hoping this goes away. From our point of view, that means we have a lot of duplicated or almost the same features which don't add

new information.

Here is a list of steps we made for working with CSS features:

1. Remove duplicated columns.
2. Remove those columns one where the there is a default value is set for every row in a dataset. For example auto, nonzero, normal, etc.
3. Replaced sub-string 'px' in every cell to extract numerical pixel value.
4. Calculated the proportion of 'NaN' and 'None' values for every column. Then remove those columns we the percentage was greater than 30%. We saw a big jump from 30% to 90% without intermediate values, so it looked like a safe threshold. See the top results for the proportion of 'NaN' and 'None'6.1.
5. Calculated unique values of CSS properties and removed that one where the value is not a categorical one.
6. Then work specifically with the 12 color properties. After we had examined their values, it appears that almost all of them were the same or had default values. So we decided to leave only one main *color* property which has the form *'rgb(number, number, number)'*. We extracted these three values for red, green and blue channels into three new features.
7. Extract a font family from the corresponding column into one separate feature.
8. On this stage we had only 12 CSS properties left. Then we considered categorical features as *display, font_family, text_align, tag, display* and made label encoding with the value between 0 and (number of classes - 1) for every feature.
9. Then we processed two features *font_weight* and *line_height*. These two properties had both numerical and categorical values. For example, *font_weight* has values as 100,200, etc and at the same time 'normal' and 'bold'. We looked at the documentation and replaced the categorical value with the corresponding numerical one.

The final list of 12 CSS features as follows: *x_coords, y_coords, block_height, block_width, tag, num_child, color_r, color_g, color_b, font_size, display, font_weight, width, height, font_family, text_align, line_height, locale.*

Corresponding code for this step you may see in the notebook with the prefix *Clean II.*

6.3 Frequent domain names

During the dataset examining, we found out that many URL pages are belonging to the same domain name. For us, it means that in the dataset we have a lot of records whose corresponding webpages have a similar structure and use the same visual templates. A structure of such HTML pages is also alike, and only the content of the pages differ. It happens

id	CSS property	value
104	-webkit-animation-name	0.999987
90	list-style-image	0.999793
40	-webkit-box-shadow	0.999637
...	...	...
83	border-bottom-style	0.994262
4	text-shadow	0.948314
9	list-style-type	0.209271
15	display	0.035866
78	speak	0.000214
87	pointer-events	0.000052
71	text-indent	0.000000
76	border-bottom-left-radius	0.000000
75	image-rendering	0.000000
74	-webkit-transition-property	0.000000
60	border-right-color	0.000000

Table 6.1: Top result for the proportion of 'NaN' and 'None' values in different CSS properties

because these pages generated automatically by a web-server, as we discussed before such websites are called *dynamic*.

On this step, we get rid of those pages whose domain appears too often. We limited the number of pages for one domain name and tagged every record with the flag *keep/remove* depending on how many times the associated domain name appeared before. Then we removed records with *remove* value.

6.4 Multi-events pages

We found two types of webpages with an event announcement: the page with only one event and the page with several events grouped by date, place, artist, and so one. We will call second group *multi-events pages*. The structure of these two groups differs because the multi-event page usually has a tabular view and visually it looks not alike. We detected URLs with up to 25 events per page. Fortunately for us, there is only 1% of multi-event pages in our dataset, so we removed those pages.

We detected such pages based on appearance of the *name* component of an event. We assumed that the name of the event is the most significant field and if there are more than one event names that mean we deal with a multi-events page. We collected the list of URLs like this and removed those records in our dataset which associated with these URLs. On the picture 6.2 you may see the difference between these two types of pages.

6.5 Outliers

After we formed boxplots of all numerical features and looked at their basic statistics we found out that they have a lot of outliers. See figure with outliers 6.3 for features *width* and *num_siblings*.

We replaced outliers with 'NaN' values for further imputation. We detected outliers similarly as a boxplot but with the difference that we considered $P_{10\%}$ and $P_{90\%}$ instead of $Q_{25\%}$ and $Q_{75\%}$. We calculated percentiles $P_{10\%}$ and $P_{90\%}$, and then we calculated the value IPR by analogy with Interquartile Range (IQR):

$$IPR = P_{90\%} - P_{10\%}$$

$$P_{90\%} + 1.5 * IPR^* \leq outliers \leq P_{10\%} - 1.5 * IPR^*$$

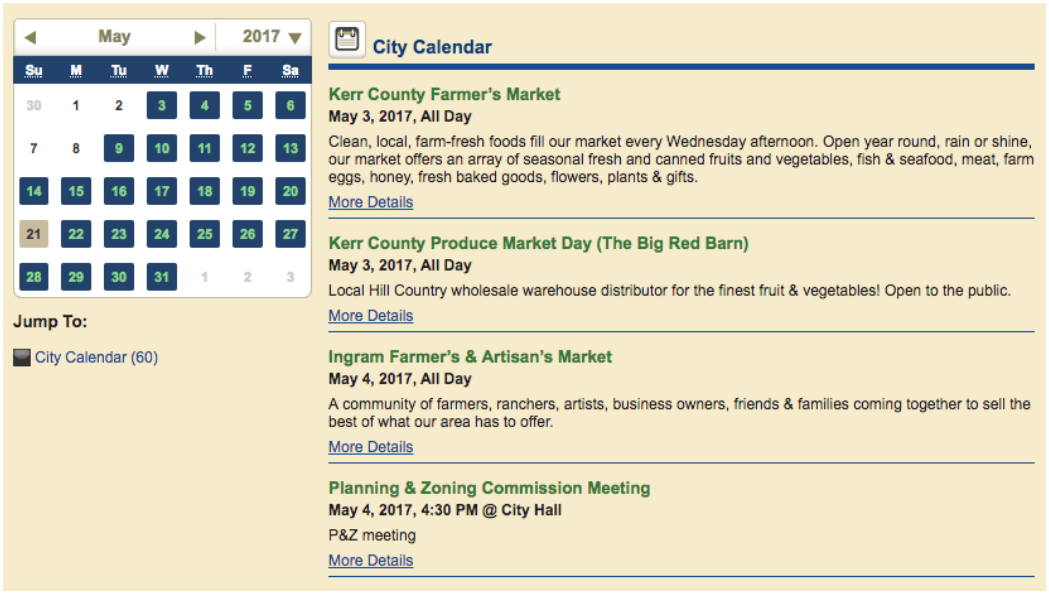
We didn't set original values of percentiles to $Q_{25\%}$ and $Q_{75\%}$ because otherwise, we would lose too much data, so we decided to consider weaker condition. As an example, see 6.2 where the number of original values are 32 207.

After we had replaced outliers, the basic statistics of the features became much better. Also, the numbers we saw accord with our understanding of the feature properties and their meaning.

Feature name	**Number of outliers**
num_siblings	476
x_coords	585
y_coords	680
block_height	837
block_width	124

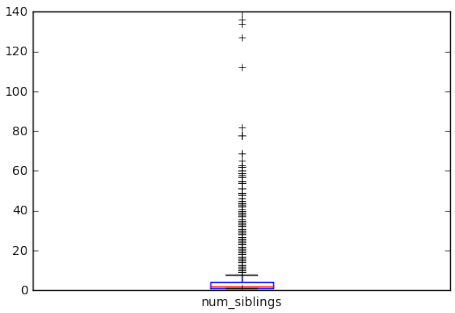
Table 6.2: Number of outliers for several features

(a) One-event webpage

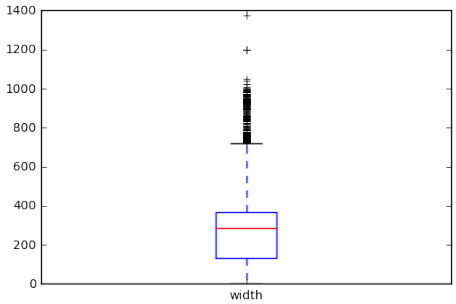


(b) Multi-events webpage

Figure 6.2: Visual difference between one-event and multi-events pages



(a) Outliers of *num_ siblings* feature



(b) Outliers of *width* CSS property

Figure 6.3: Examples of outliers

Conclusion of the chapter

In this chapter, we discussed the five steps on data cleaning. After these actions, we reduced the number of records in the dataset significantly. In original dataset, we had 1.6M rows with the both event components and random elements of the page. After the last step, we gained 170K where 140K is random web element features, and 30K are an event components.

The largest part was reduced after the first step where we removed the obvious inconsistency. That happens due to a big number of web elements without width, height and with zero coordinates. These elements usually have *meta* tag and they include all event information but not associated with any real web elements which contain the name, location, date and description. Also, we removed multi-events pages, pages with frequent domains, detected outliers and replaced them with 'NaN' value. We performed CSS feature selection and cleaning, after this step the number of CSS features decreased from 288 to 12.

Chapter 7

Experimental part: Exploratory data analysis

In this chapter, we will explore the data we have, create several new features from existing ones, visualize and discover the relationships between them. Most of the figures and tables for this chapter you can find in Appendix.

Here we often refer to a *meta name* field in the dataset. It may takes five values: *name*, *startDate*, *location*, *description* and *noEvent*. Also we use the term *event component* which is applied to all meta names excluding *noEvent*.

7.1 Feature engineering

Before we start exploring the data, we would like to extract several new features. We will work on the following types of features:

- *Textual features* related to an original text of the event components.
- *Spatial features* related to X and Y coordinates of the web element on the page. X and Y are coordinates of a visual block on the real rendered page.

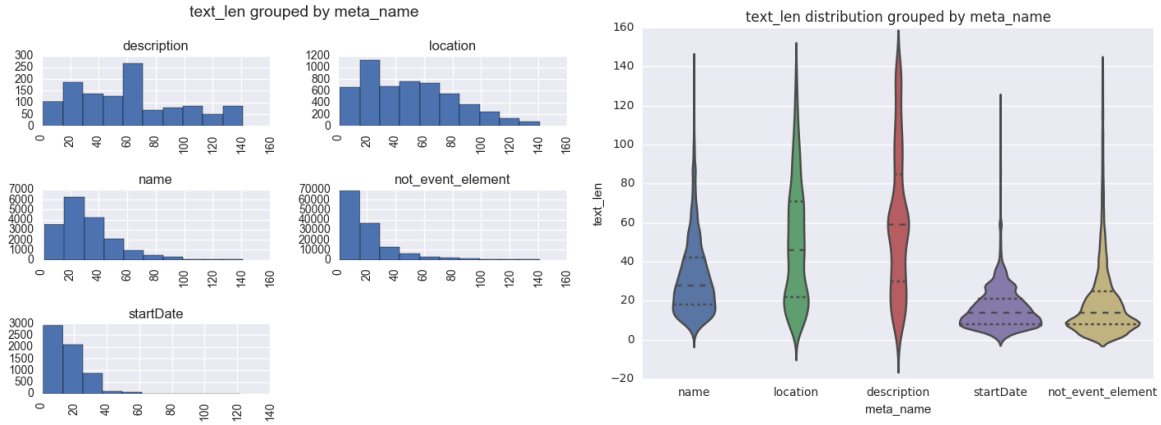
In section Working with CSS properties we already extracted several *Visual features* from the CSS properties. We divided into three parts a value of color, the font family and properties measured the pixels.

7.1.1 Textual features

An original textual feature is a text contained in a web element. For the event name, location and start date, it's a quite short string. For a description is much longer text. That's why the first extracted feature is a *length of the text*. On the figure 7.1 you distribution of the text length differs among meta names.

After we had extracted the text length, we noticed that the date and location meta names usually contain more digits than name and description. So we derived also the number of digits in a text. See figure 7.2.

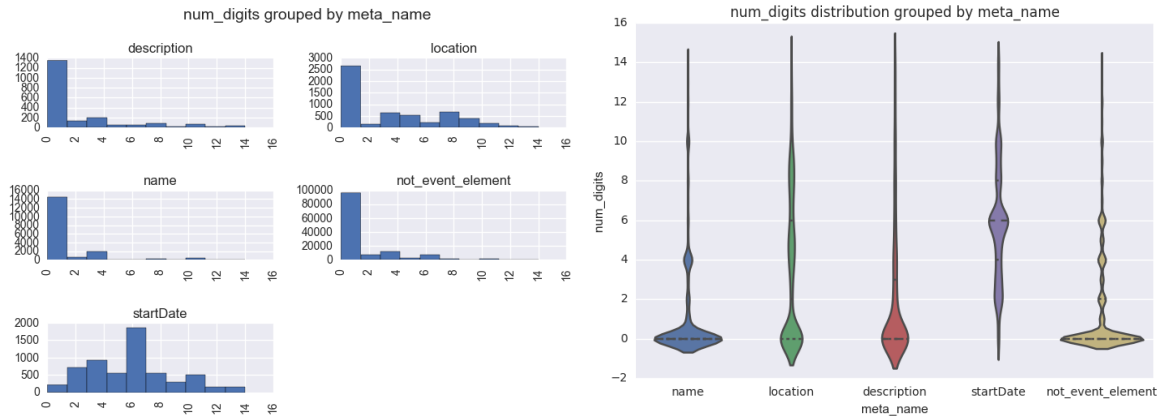
By analogy, we also extracted the number of uppercase letter, white spaces, digit proportion and the number of punctuation marks. See the rest of the figures in Appendix.



(a) Histogram

(b) Violin plot

Figure 7.1: Distribution of the text length by meta name



(a) Histogram

(b) Violin plot

Figure 7.2: Distribution of the digits number by meta name

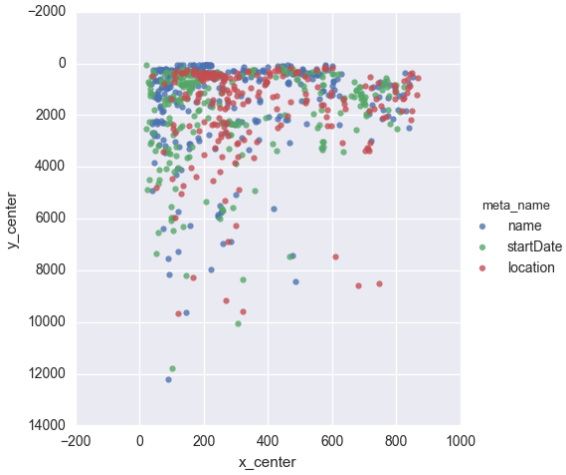
Therefore we created six additional features from one textual field: text length, the number of punctuation marks, the number of digits, digits share, the number of upper case letters, the number of white spaces. You can see that visually they distinguish meta names well. The difference in basic statistics you can find in table 10.1.

When we build different classification models in the next chapter, we will also calculate a *tf-idf weighting matrix* in order to reflect the importance of every word in a text.

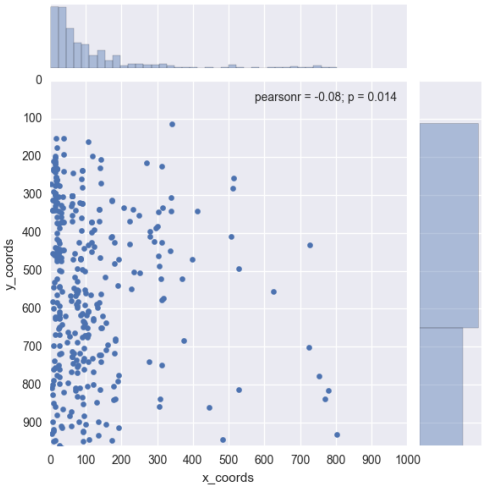
7.1.2 Spatial features

Here we will work on two original coordinates X and Y of visual blocks. The visual block is a rectangle on a webpage where the web element is located. Also, we will use two other related features: block width and block height. The precise meaning of X, Y features as follows: X pixels means horizontal position on a page from the left side, and Y pixels means vertical position on a page from the top side. That means that X and Y are coordinates of the left upper corner of the corresponding visual block. An example of X-Y scatter plot you can see on figure 7.3b. The axes on inverted such it looks the same as on a webpage.

The first feature we created, is an ultimate X and Y which are centers of the visual block rather than left the upper corner. Our guess was that these two features better represent the location on the page. We visualized these two coordinates, and it appears that there is no clear pattern here, rather the elements can appear almost anywhere on a page with only small deviation in means. See the figure 7.3a.



(a) Centers of block coordinates for different meta names which appear on the same page



(b) Original X and Y for an event start date

Figure 7.3: The distribution of X and Y coordinates

Then we applied K-means clustering algorithm to assign the cluster label for every row. The idea here was to attempt the visual block on a page where every component might appear. We learned the model for sub-datasets for every meta name. Unfortunately, this experiment wasn't successful, because it seems that centered X and Y don't have a clustering structure. There is some difference in basic statistics of X and Y for different meta names, but the standard deviation is so high that it can't be considered as a significant result, see the table 10.2.

We can make several conclusions on the spatial features:

- Visually X and Y coordinates badly separate the meta names.

- The average of X coordinate differs for all meta names, but the standard deviation is very high. Y coordinates are more or less the same for all meta names except the description, usually, it's located lower than other components.
- Both, the visualization and summary statistics may be interpreted as follows: generally, the name is located higher, the start date is more on the left side, the start date is usually more left than the location, and description is below. From page to page the design is different, and that's why there is no clear picture on component locations. Since in our work we build one-vs-all classifiers, we won't use the relative positions of the event components.

7.2 The final dataset

The final dataset after the cleaning and feature engineering is a table of 168K rows and 32 columns. Let's look at its main properties:

- The list of features in the dataset can be seen in table 7.1.
- Every row in a dataset associated with one of the following meta names for every parsed URL: event name, event date, event location, event description or random element on the page. The row contains URL, meta name and all web features we collected: textual, spatial, visual and DOM-related.
- It's cleaned of outliers, frequent domains, multi-events pages, and inconsistency.
- It contains 33K unique URLs and 4000 domain names.
- There are in average 45 URLs per one domain name.
- We have 137K rows related to random elements on the webpage, 18K for an event's name, 6K for a start date, 6K for location, and 2K for description. In sum, 32K rows containing features of event components.
- There are in average 1.8 event components out of 4 on a webpage.
- There are the most popular combinations of event components fro one URL.
  - *name* 8335
  - *location, name* 2863
  - *name, startDate* 2522
  - *location, name, startDate* 1735

  Top 10 list of combinations you may find in table 7.2.
- We have 30 numeric features and 4 textual. Additionaly, we will include *idf-idf* matrix later.
- We replaced outliers of numerical features with the NaN. We fill the missing values with the corresponding average values.

DOM	**CSS (Visual)**
URL of the page	Color (three chanells)
Meta name (name, startDate, description, location, no_event)	Text alignment (center, left, right, etc)
Text of the web element	Property of the block view
HTML tag	The level font weight
Number of siblings in a DOM tree	Padding
Number of childs in a DOM tree	Font family (Verdana, Arial, etc)
	Font size
	The language of the page
Spatial	**Textual**
X and Y coordinate on the rendered picture	Length of the text inside the block
X and Y coordinate of the center of a visual block	Number of punctuation marks in text
Visual block height	Number of digits in text
Visual block width	Number of digits/text length
Height in a browser	Number of upper case letters
Width in a browser	Number of whitespaces
	+ *tf-idf matrix*

Table 7.1: The final list of features

7.3 Analysis of features

Let's briefly mention other valuable features which separate the meta name well enough. We will present the visualization with the grouping by the meta name and some summary tables.

On the figure 7.4b we can see that the number of uppercase letters is the highest for location meta name and the lowest for a start date.

The figure 7.4a shows that the distribution of a font size has a very different shape for different meta names. For description, it is more or less normal, for the name and start date it is right-skewed distribution, and for a random element it is almost stable which says that the random elements of the page in average has the same font size.

Basic statistics for the different numerical features for all meta names you can see on table 7.3.

On the figure 7.5a you will notice that the tag usage is different for different meta names.

Combination	**Count**
['name']	8335
['location', 'name']	2863
['name', 'startDate']	2522
['location', 'name', 'startDate']	1735
['description', 'name']	730
['description', 'name', 'startDate']	548
['description', 'location', 'name']	485
['description', 'location', 'name', 'startDate']	293
['name', 'startDate', 'startDate']	246
['location', 'name', 'startDate', 'startDate']	125

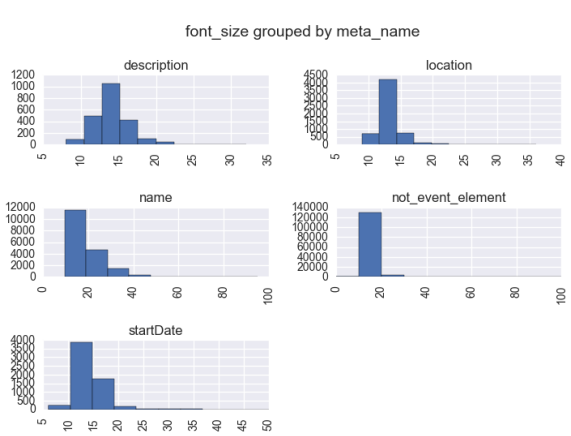
Table 7.2: Top 10 combinations of event component for every URL

We encoded the tag name by the number from 0 to 35.

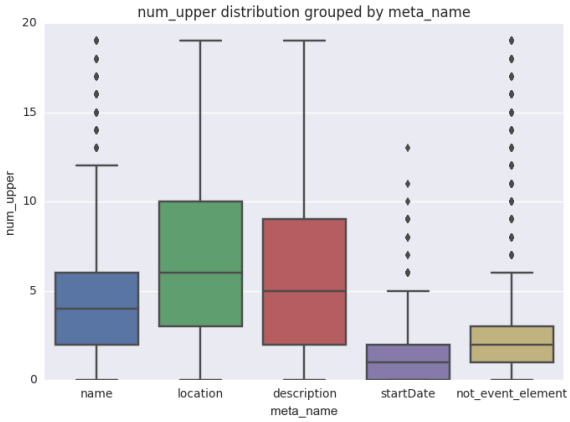
Despite the fact, there are many features which visually distinguish the different meta names, dimensionality reduction output didn't show good separation. We standardized the values and performed both PCA and t-SNE approaches on all numeric features to see if the clusters appear there. Especially after unsuccessful clustering, it would be good to see it is there. PCA showed a small separation but t-SNE not. The result of PCA you can see on figure 7.5b

Conclusion of the chapter

In this chapter, we performed exploratory data analysis by visualizing the features and grouping by the meta names. As we see, there features which separate well the meta names (the tag, font size, digits share, text length, etc.) and which not (x and y coordinates, color). We performed dimensionality reduction with PCA and saw a small separation on clusters whereas k-means clustering on X and Y coordinates didn't perform well.

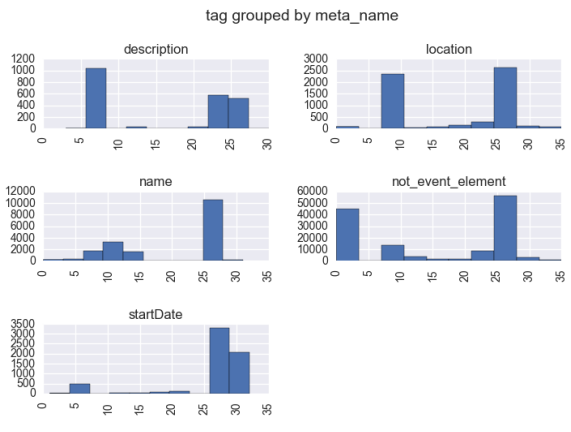


(a) The font size by meta names



(b) The number of uppercase letters by meta names

Figure 7.4: Distribution of digits number by meta name



(a) Distribution of HTML tag value by meta names

(b) PCA on standardized numeric values for different meta names.

stat	color_r	font_size	font_weight	digits_share	block_height
count	167872.0	167872.0	167872.0	167872.0	167872.0
mean	76.396201	14.69536	455.510746	0.112503	26.359833
std	84.989934	4.15742	129.067880	0.225118	18.612600
min	0.0	0.0	100.0	0.0	0.0
25%	4.0	13.0	400.0	0.0	16.0
50%	51.0	14.0	400.0	0.0	19.0
75%	125.0	16.0	400.0	0.117021	31.0
max	255.0	100.0	900.0	1.0	194.0

Table 7.3: Summary statistics for some numerical features

Chapter 8

Experimental part: Models and evaluation

In this chapter, we will experiment with the different classification models. We will consider Random Forest, SVM, Extreme Random Forest [**?**] and Logistic regression for comparison. Also, we will analyze the feature importance for every event component with Random Forest. We will actively use great Python library scikit-learn (sklearn).

We conducted three experiments with different preferences and input features. The result of every experiment is a summary table with the following metrics: accuracy, precision, recall and F_1 score. In this chapter, we also will briefly remind the definition and intuitive meaning every metric.

8.1 Feature importance

To gather the feature importance for every component, we will use Random Forest classifier and Boosted Trees. Random Forest's implementation in sklearn automatically calculates the "Gini importance" or "mean decrease impurity" which is defined as the total decrease in node impurity, weighted by the probability of reaching that node and then averaged over all trees of the ensemble.

For every event component, we trained Random Forest classifier with one-vs-all strategy. That means that as positive examples we choose the records from a dataset associated with the target event component and as negative examples the rest of records. After we had trained the Random Forest (the number of trees is 1000), we took the feature importance from it. The detailed procedure of building the model will be discussed in the next section.

Remarkably that the **Top-5** features for every component totally makes sense:

- **Event name**: font family, tag, block width, font size, the number of uppercase letters. Usually, the title of the event is bigger and probably has a tag <h1>. The name is

rather short, and hence the block width is small. The font size is large and the number of uppercase letters often higher for attracting the user. To see the importance of features for a name component see figure 10.10. The rest of related pictures you will find in Appendix.

- **Event date**: digits share, the number of digits, font family, tag, number of punctuation. It also logical since the date, of course, has more digits than other web elements. The number of punctuation is rather small because the text is usually short. See figure 10.5.
- **Event location**: number of uppercase letters, tag, font family, the number of siblings, text length. See figure 10.6.
- **Event description**: block width, text length, font family, number of punctuation. The description, typically, is a wider block of text, that's why block width and text length are the most important features. The number of punctuation is also correlated with the text length and means significant. See figure 10.7.

Similarly, we extracted feature importance from the trained Boosted trees model which is also a great tool for classification tasks but requires intensive parameter tuning. See figures to see result for a date of the event, location, description and name.

8.2 Comparison of different models

8.2.1 Evaluation metrics

We used standard metrics for binary classification problem and Information extraction task. All the values reach its best value at 1 and worst score at 0. See below brief description each of them.

The mean accuracy is the average number of incorrect predictions:

$$mean\_accuracy = \frac{tn + fp}{tp + tn + fp + fn}$$

where tp, tn, fp, fn are accordingly true positives, true negatives, false positives and false negatives records from a predicted labels.

The precision is the ability of the classifier not to label as positive a sample that is negative:

$$precision = \frac{tp}{tp + fp}$$

The recall is the ability of the classifier to find all the positive samples:

$$recall = \frac{tp}{tp + fn}$$

The F_1 score, also known as balanced F-score or F-measure is as a weighted average of the precision and recall. The contribution of precision and recall to the F_1 score are equal. The formula for the F_1 is:

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

8.3 Experiments

In this section, we will describe our experiments on building the binary classification models for every event component. For every test, we will change the preferences to see the difference and influence of each step.

For every binary classification model and target event component, we will do the following actions:

1. We select the *target event component* for which we build the model (for example, date of the event)
2. As positive examples, we take all records containing the target component in an original dataset ("startDate")
3. As negative examples, we take the rest of the records and mark them as "no_startDate" (records "name", "location", "description", "no_event" will became "no_startDate")
4. We perform stratification procedure what means that we balance the dataset in such way that the number of examples for both classes would be the same.
5. We impute missing values with the mean (we replaced outliers with NaN in the previous steps)
6. We train the model
7. We calculate four performance metrics like the average values after cross-validation procedure
8. We build summary table and compare the results

For all experiments except the first one, we performed two additional procedures. Firstly, we do standardization of features with the mean and standard deviation. Secondly, we implemented custom cross-validation function which tracks the following important property: for every train-test split, domain names in both sets should not overlap.

The reason for performing this procedure is the following: we have many different URLs, but some of them come from the same domain name. Usually, the pages in one domain look

similar, and it is not fair to train classifiers on certain pages and test on the pages from the same domain names. Our experimenting shows that if we don't do this procedure, the result of the classifier is overestimated. In our task and source code, we call this procedure *fair train test split*.

8.3.1 Experiment 1

Preferences: Original numeric features as an input without tracking domain names intersection.

Result: All metrics value are very high, and the result is obviously overestimated. The reason for this is that both train and test dataset contained the same domain names. Therefore trained classifier has already known the design and properties of the website on the testing stage. See the table 10.3 to see extremely high metric values. The only SVM performs not well because it implies standardized values, but we didn't do it in the first experiment.

Conclusion: It is critical to keep training and testing examples independently and not allow to leak the information from the training stage to the testing stage. If you see the high values of performance metrics, be suspicious and try to understand why it happens.

8.3.2 Experiment 2

Preferences: Original numeric features with the tracking of domain names intersection while splitting into train and test subsets.

Result: After we added the restriction on domain names intersection, all the metrics decreased from .97 to .85 in average, but this estimation is much fair than previous. The result shows us that the model is working, but it's not the best, so we are motivated to tune it and experiment with the features further. Extreme Random Forest shows the best results for all event components. The event date component with SVM has the worst results by all metrics. Simple logistic regression shows the relatively good result for all models. See results of this experiment on the table 10.4.

Conclusion: Even if metrics decreased after applying some restrictions, it's better to have a fair model and optimize it.

8.3.3 Experiment 3

Preferences: Original numeric features with the tracking of domain names intersection while splitting into train and test subsets. Also, we extracted textual field and constructed a tf-idf [**?**] matrix which shows how important every word in a text relatively to the whole corpus. In our case, the corpus of documents is all textual fields from the dataset. We applied tf-idf transformation separately for train and test set. tf-idf matrix is sparse, that's why then we used Incremental PCA [**?**] dimensionality reduction, which is more memory

efficient than the traditional PCA.

Result: After these transformations the metric results for each event component and classifiers increased. Almost all metrics became greater than .8 for all components and classifiers. F_1 score, precision, and accuracy are the highest for Extreme Random Forest almost for all event components. The random forest has the highest recall for all components. See results of this experiment on the table 8.1.

Conclusion: Working with textual field and applying the combination of tf-idf and dimensional reduction methods can increase the result very much.

The code for all three experiments can be found on attached CD in the folder notebooks/ with the prefix *"Model Building."*

f1_score	accuracy	precision	recall	meta_name	model
0.86	**0.86**	**0.86**	0.86	name	Random forest
0.82	0.83	0.82	0.84	name	SVM
0.75	0.75	0.74	0.78	name	Logistic regression
0.85	**0.86**	0.81	**0.90**	name	Extreme Random Forest
0.81	0.80	0.77	0.87	location	Random forest
0.81	**0.81**	**0.81**	0.81	location	SVM
0.76	0.76	0.77	0.77	location	Logistic regression
0.82	**0.81**	0.75	**0.91**	location	Extreme Random Forest
0.84	0.83	0.82	0.88	description	Random forest
0.82	0.82	0.78	0.87	description	SVM
0.77	0.77	0.74	0.83	description	Logistic regression
0.86	**0.87**	**0.83**	**0.91**	description	Extreme Random Forest
0.91	0.90	**0.90**	0.92	date	Random forest
0.87	0.87	0.89	0.86	date	SVM
0.80	0.81	0.86	0.79	date	Logistic regression
0.91	**0.91**	0.88	**0.95**	date	Extreme Random Forest

Table 8.1: Experiment 3: Metrics values for a different classification models and event components. PCA components calculated on extracted tf-idf matrix were added as additional features. Cross-validation with k = 5.

Conclusion of the chapter

In this chapter we showed three experiment with four different classification models and initial preferences on input features. We saw that additionally constructed tf-idf matrix and PCA improved the performance, hence textual information is important factor for event extraction task. For every event component there are different classifiers which maximize the metrics, but generally ensemble model Extreme Random Forest works the best. Also, it appears, that font family is one of the most important features based on feature importance from Random Forest and Boosted trees. Other important features are tag, font size, digits share and text length.

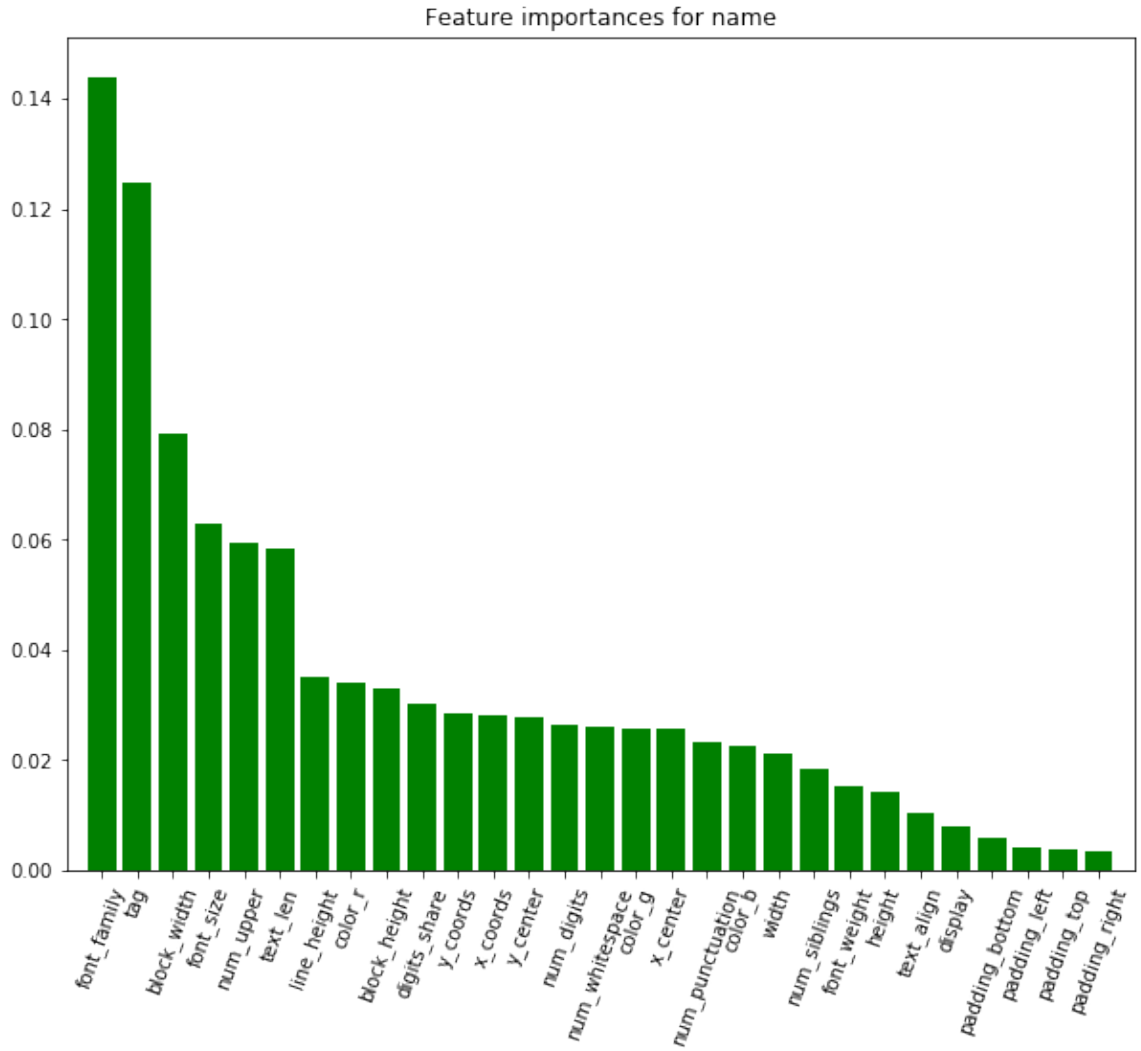


Figure 8.1: Feature importance from Random Forest for the Event name. Top-5: font family, tag, block width, font size, number of uppercase letters

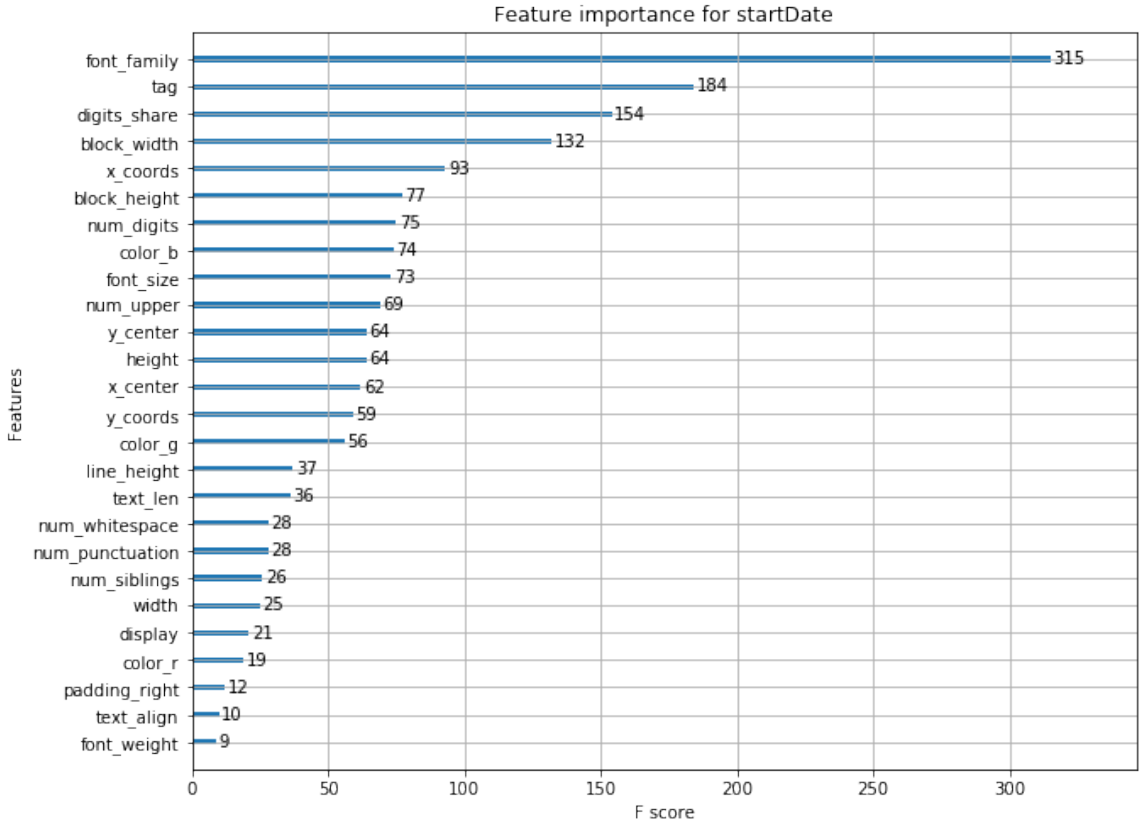


Figure 8.2: Feature importance from Boosted Trees for the Event name. Top-5: font family, tag, digits share, block width, x_coords

Chapter 9

Conclusion

9.1 Summary

The primary results of this thesis are the following:

1. We made a review of existing modern Web Extraction methods, considered all main complementary tasks and listed work related to social events extraction.

2. We developed the original system which in parallel collects training examples for an event extraction problem with the use of Microdata semantic markup.

3. Hence, we automatically created the training labeled dataset which contains various web features: visual, textual, spatial and DOM-related.

4. We performed extensive cleaning procedure on the original dataset what reduced the number of records and columns dramatically.

5. We engineered many features from original ones.

6. We made comprehensive exploratory data analysis for every event component including visualization of extracted features and their relationships, dimensionality reduction and clustering.

7. We made the final dataset public and therefore it is the first publicly available dataset for social event extraction problem. The data with the code are available in a GitHub repository: https://github.com/galinaalperovich/Ms-Thesis-CVUT

The main conclusion of the thesis is also that we demonstrated that it is possible to automatically extract training dataset, prepare and build meaningful models which can recognize the event components on a web page. The list of items annotated with the semantic markup is very large, and hypothetically our approach can be extended to any of them.

9.2 Future work

This work can be improved and optimized in many ways indefinitely.

Data collecting: It is possible to collect even more training examples, while the upper bound for the number of such examples is the amountf URLs which have the Microdata semantic markup in their HTML code. It is possible to extract more features, especially DOM-related ones. Theoretically, it could be any features associated with the tree structure of the page. Also, since we used the virtual browser it is possible to consider the page as an image and apply computer vision methods on it.

Cleaning procedure might be incorporated into crawling process to filter pages and web elements. Region Extraction methods would help in this task because they can identify important regions on the page.

Feature engineering: If we have more extracted web features, is it possible to create more relevant features from existing ones to improve the quality of classifiers.

Approach: In the thesis, we considered the local event extraction problem as a set of binary classification tasks which work on every event component independently. A logical continuation of this approach might be the using of the relative information about the event components. Also, it would be a good idea to process the page content more like a natural language text and exploit its sequential structure.

Chapter 10

Appendix

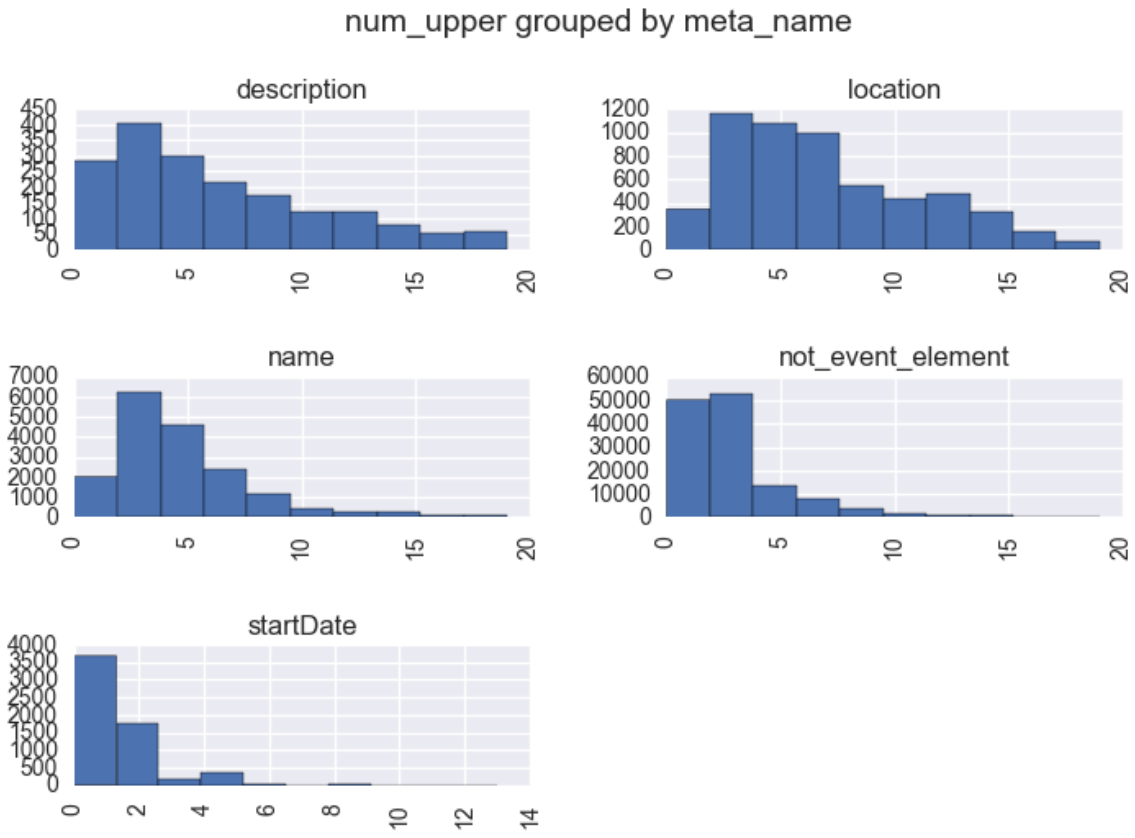


Figure 10.1: The distribution of upper case letters number by meta name

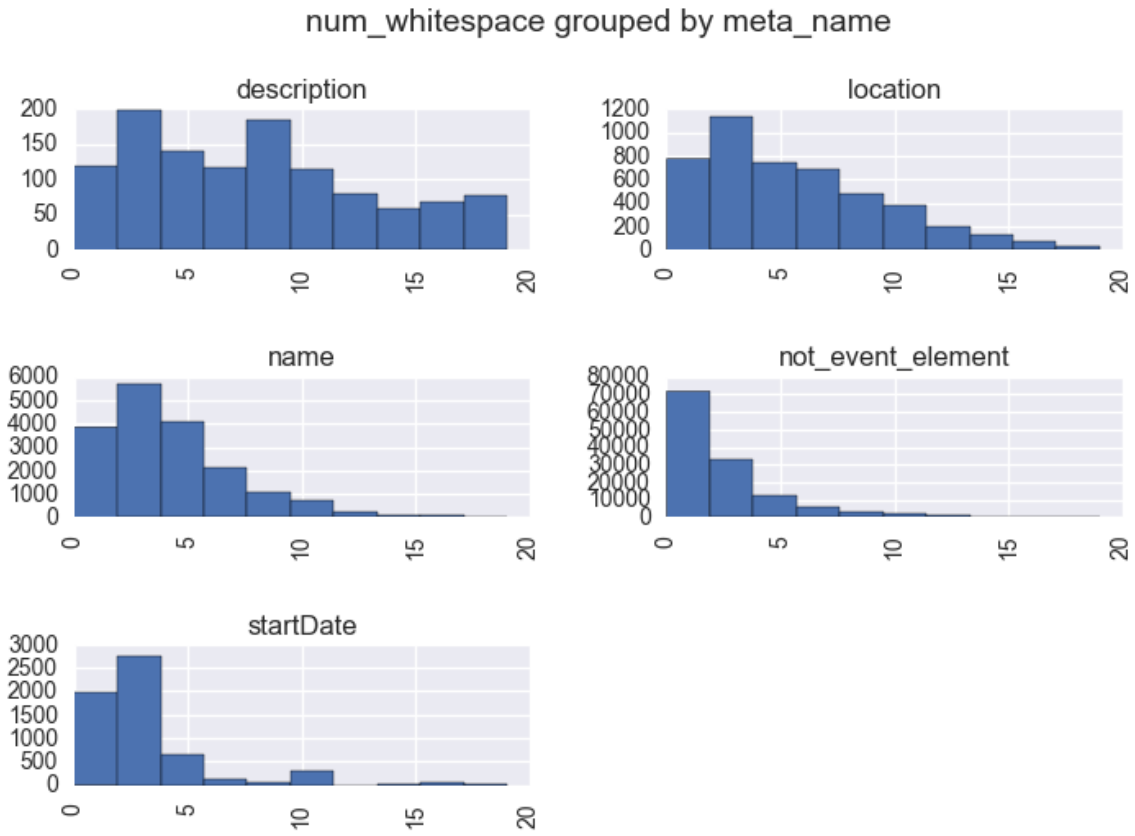


Figure 10.2: The distribution of white spaces by meta name

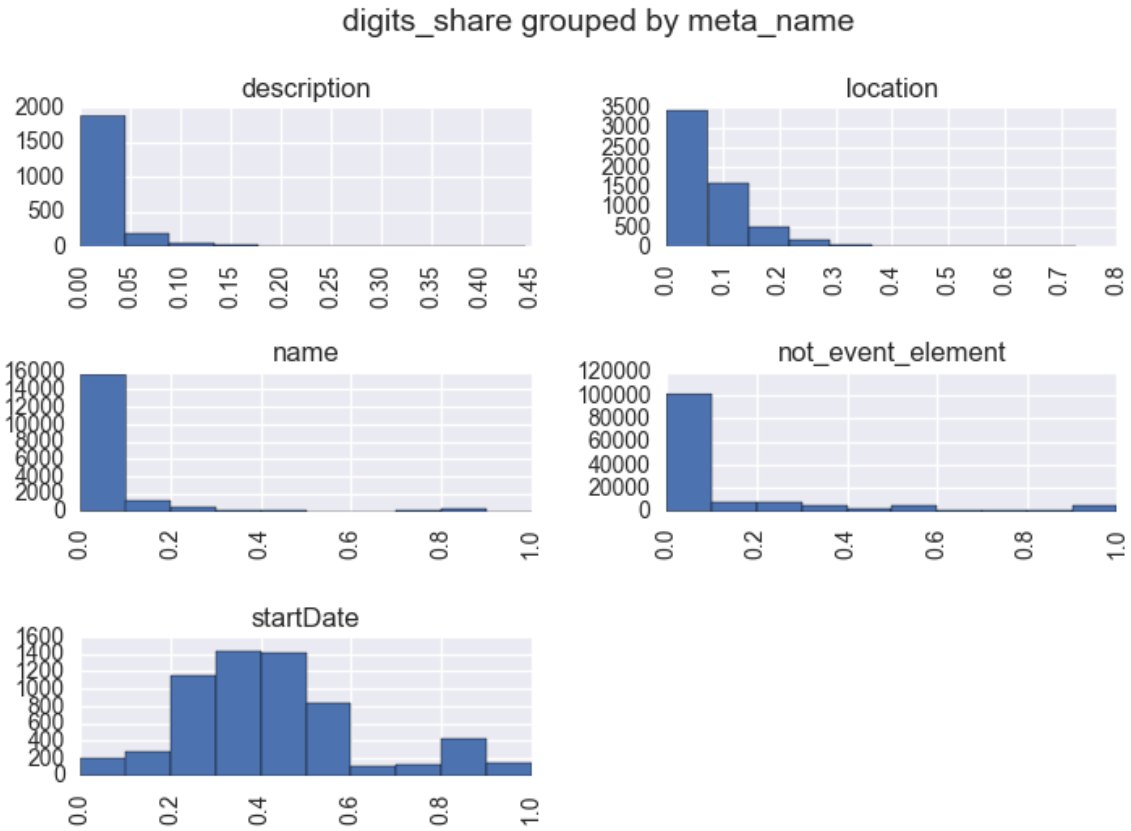


Figure 10.3: The distribution of digit proportion by meta name

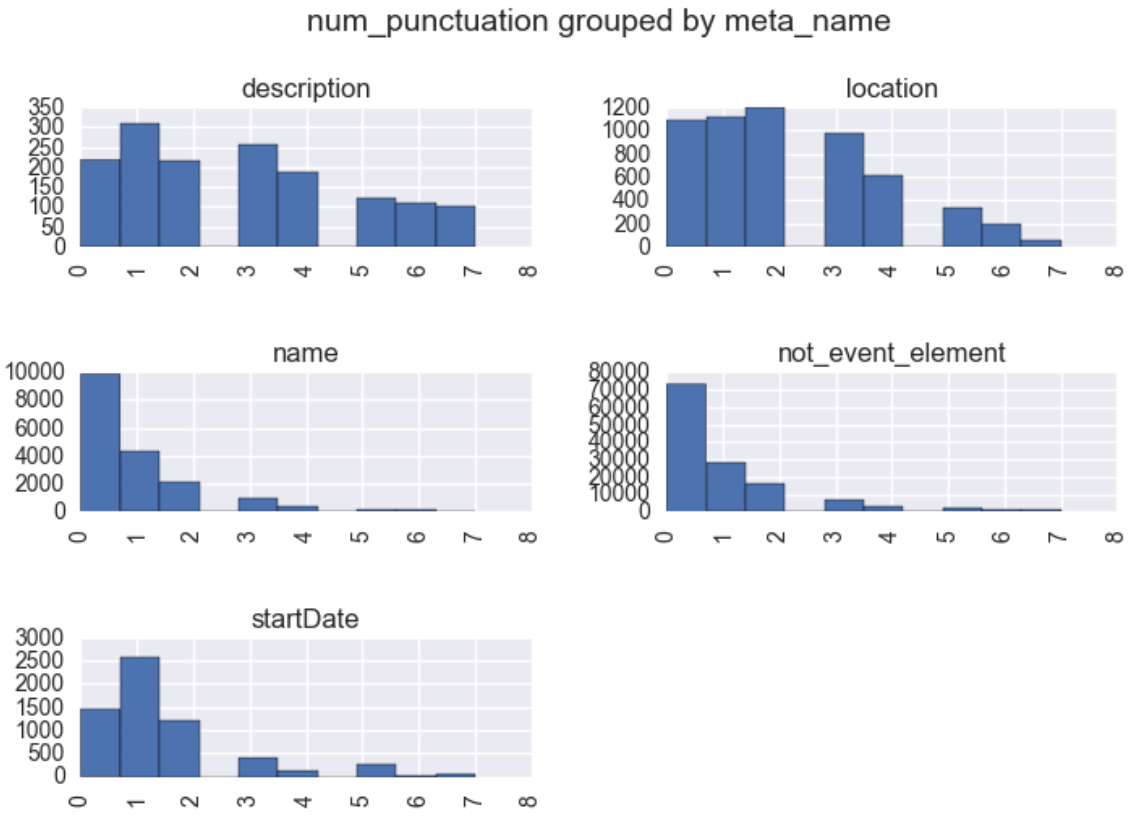


Figure 10.4: The distribution of punctuation marks number by meta name

		digit_share	num_digit	num_punc	num_upp	num_white	text_len
description	count	2206.00	2206.00	2206.00	2206.00	2206.00	2206.00
	mean	0.02	1.99	2.23	5.70	5.41	43.14
	std	0.05	3.28	1.94	4.59	4.72	32.55
	min	0.00	0.00	0.00	0.00	0.00	1.00
	25%	0.00	0.00	0.96	3.00	2.65	23.02
	50%	0.00	0.00	1.00	4.00	2.65	23.02
	75%	0.02	2.00	3.00	8.00	8.00	60.00
	max	0.44	14.00	7.00	19.00	19.00	141.00
location	count	5780.00	5780.00	5780.00	5780.00	5780.00	5780.00
	mean	0.06	3.27	2.12	6.74	4.87	47.45
	std	0.08	3.65	1.69	4.35	3.85	30.90
	min	0.00	0.00	0.00	0.00	0.00	1.00
	25%	0.00	0.00	1.00	3.00	2.65	23.00
	50%	0.04	1.62	2.00	6.00	3.00	42.00
	75%	0.11	6.00	3.00	10.00	7.00	69.00
	max	0.73	14.00	7.00	19.00	19.00	141.00
name	count	18116.00	18116.00	18116.00	18116.00	18116.00	18116.00
	mean	0.05	1.07	0.85	4.46	3.97	32.32
	std	0.14	2.41	1.25	3.25	3.11	20.12
	min	0.00	0.00	0.00	0.00	0.00	2.00
	25%	0.00	0.00	0.00	2.00	2.00	18.00
	50%	0.00	0.00	0.00	4.00	3.00	27.00
	75%	0.00	0.00	1.00	6.00	5.00	42.00
	max	1.00	14.00	7.00	19.00	19.00	141.00
not_event	count	135665.00	135665.00	135665.00	135665.00	135665.00	135665.00
	mean	0.11	1.43	0.88	2.70	2.33	20.70
	std	0.23	2.60	1.31	2.83	3.08	21.49
	min	0.00	0.00	0.00	0.00	0.00	1.00
	25%	0.00	0.00	0.00	1.00	0.00	8.00
	50%	0.00	0.00	0.00	2.00	1.00	14.00
	75%	0.10	2.00	1.00	3.00	3.00	24.00
	max	1.00	14.00	7.00	19.00	19.00	141.00
startDate	count	6105.00	6105.00	6105.00	6105.00	6105.00	6105.00
	mean	0.41	5.74	1.40	1.38	2.75	16.54
	std	0.20	3.01	1.36	1.36	2.92	10.98
	min	0.00	0.00	0.00	0.00	0.00	1.00
	25%	0.29	3.00	1.00	0.00	1.00	9.00
	50%	0.39	6.00	1.00	1.00	2.00	14.00
	75%	0.50	8.00	2.00	2.00	3.00	22.00
	max	1.00	14.00	7.00	13.00	19.00	122.00

Table 10.1: Summary statistics for a textual features grouped by meta name

		x_center	y_center
description	count	2206.00	2206.00
	mean	262.80	3112.37
	std	135.67	2599.73
	min	23.50	120.50
	25%	200.00	805.62
	50%	200.00	2247.50
	75%	279.75	6151.55
	max	920.50	13226.00
location	count	5780.00	5780.00
	mean	283.90	2017.20
	std	192.64	2072.98
	min	35.50	120.50
	25%	189.50	602.00
	50%	200.00	1210.50
	75%	312.12	2635.12
	max	900.00	13168.50
name	count	18116.00	18116.00
	mean	248.27	2157.42
	std	152.47	2354.20
	min	14.50	27.00
	25%	167.00	479.50
	50%	200.00	1163.00
	75%	284.50	3010.62
	max	920.50	13248.50
not_event_element	count	135665.00	135665.00
	mean	305.94	7076.33
	std	210.60	10007.35
	min	0.00	4.50
	25%	153.50	1397.00
	50%	235.00	3060.50
	75%	419.00	6957.50
	max	1418.00	53955.00
startDate	count	6105.00	6105.00
	mean	180.83	2465.70
	std	145.79	2326.51
	min	19.00	34.50
	25%	89.00	676.00
	50%	137.50	1570.50
	75%	203.50	3540.50
	max	887.50	13239.50

Table 10.2: Summary statistics for a spatial features grouped by meta name

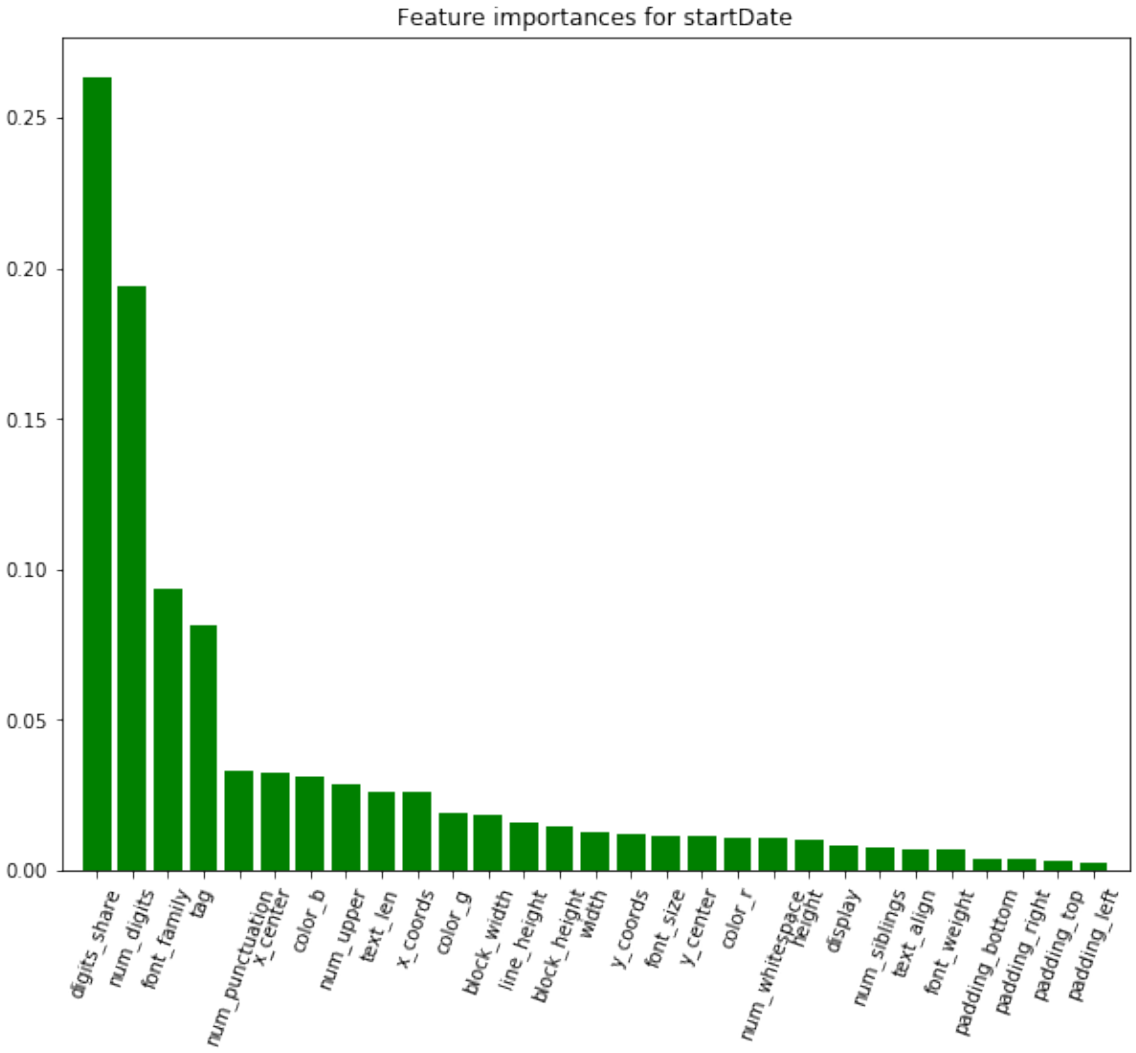


Figure 10.5: Feature importance from Random Forest for the Event date, Top: digits share, number of digits, font family, tag, number of punctuation

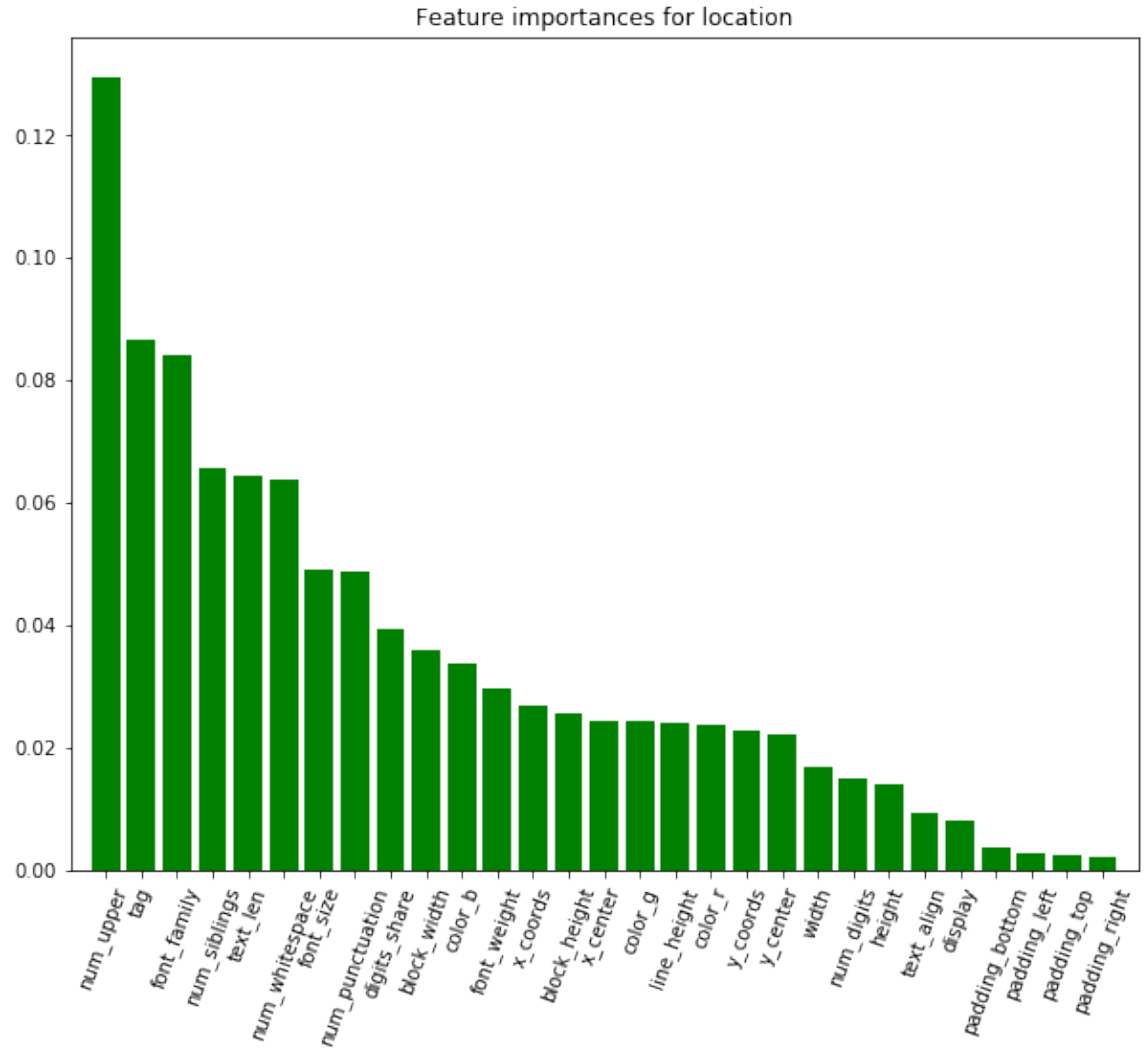


Figure 10.6: Feature importance from Random Forest for the Event location, Top: number of uppercase letters, tag, font family, number of siblings, text length

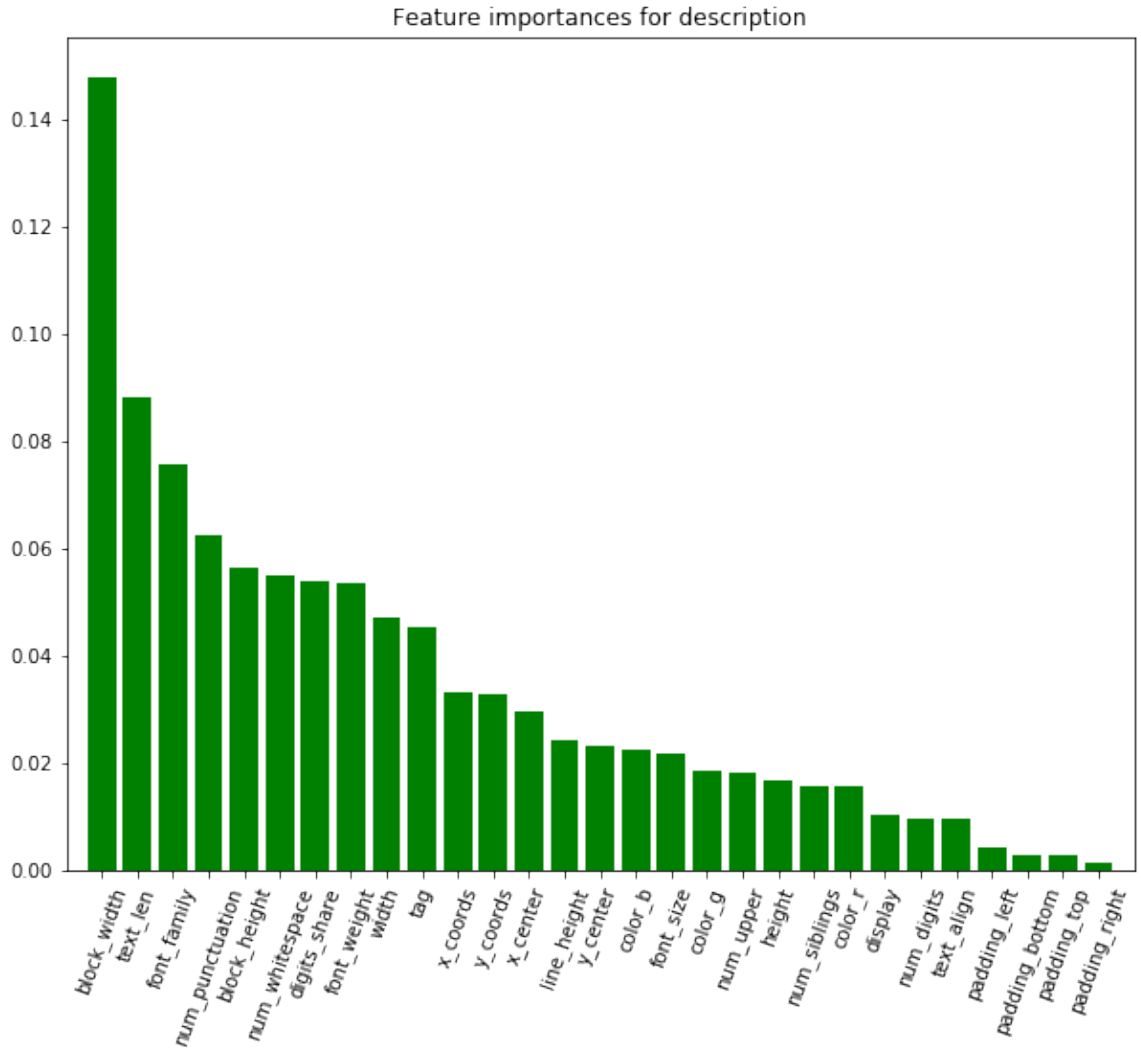


Figure 10.7: Feature importance from Random Forest for the Event description, Top: block width, text length, font family, number of punctuation

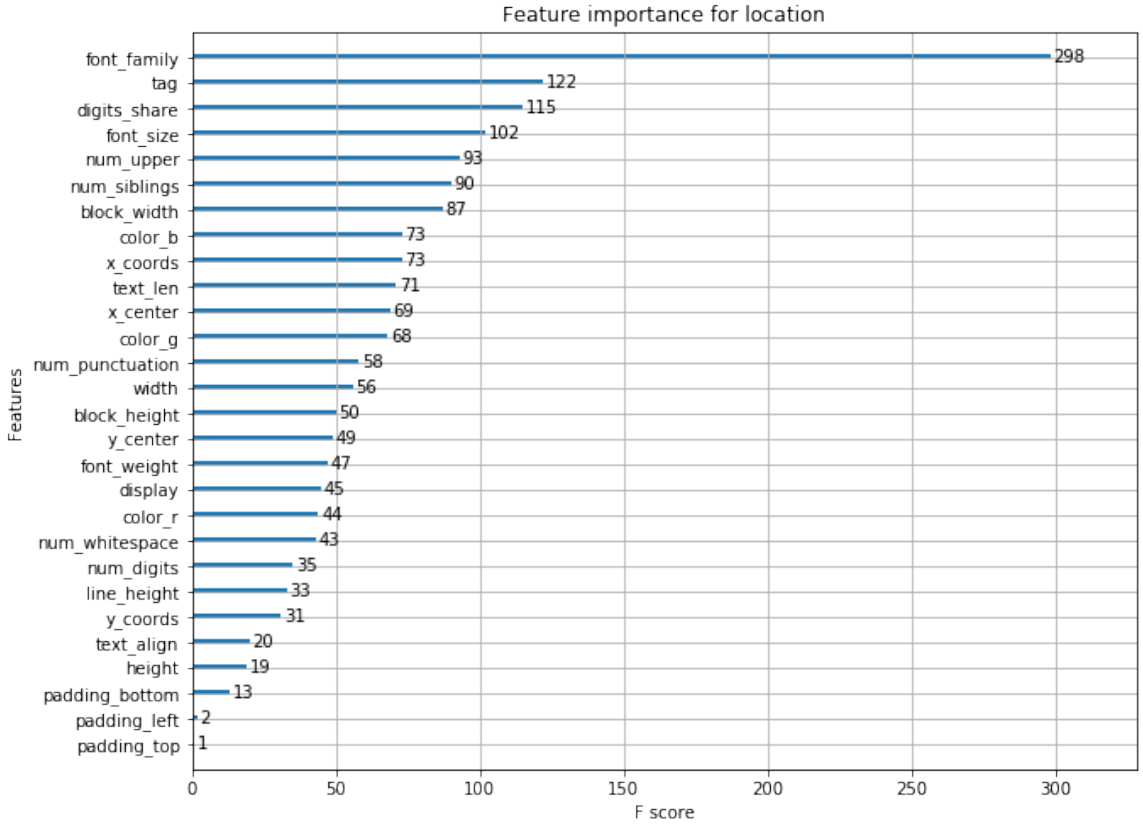


Figure 10.8: Feature importance from Boosted Trees for the Event location. Top-5: font family, tag, digits share, font size, number of uppercase letters

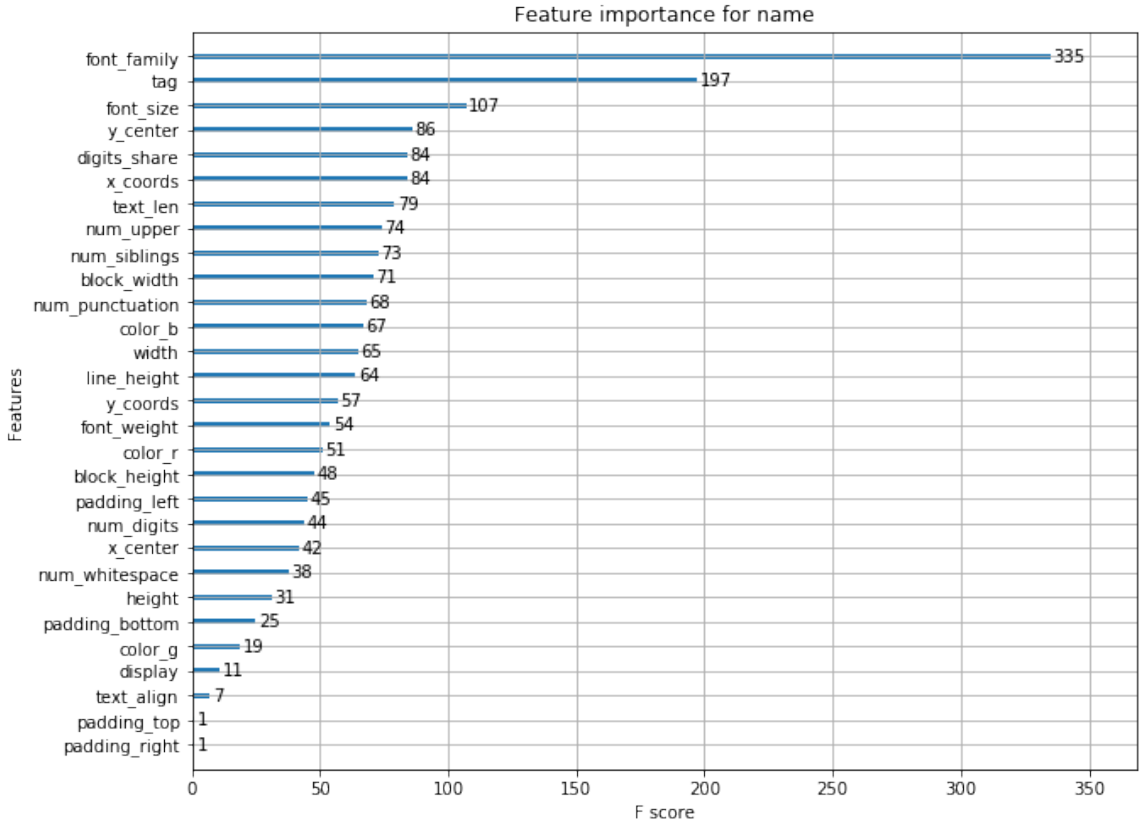


Figure 10.9: Feature importance from Boosted Trees for the Event name. Top-5: font family, tag, block height, font weight, x coords

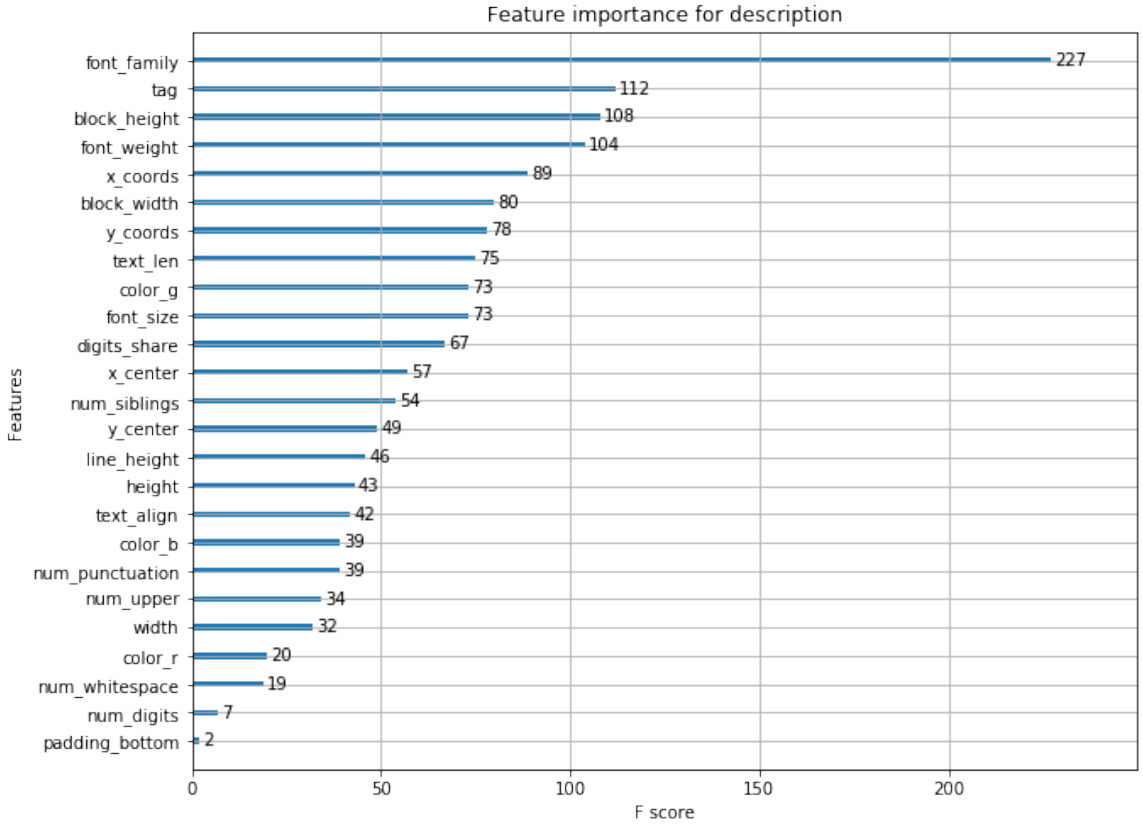


Figure 10.10: Feature importance from Boosted Trees for the Event description. Top-5: font family, tag, block width, font size, number of uppercase letters

accuracy	f1	precision	recall	meta_name	model
0.98	0.98	0.98	0.97	name	Random forest
0.58	0.70	0.54	1.00	name	SVM
0.82	0.82	0.81	0.82	name	Logistic regression
0.98	0.98	0.98	0.97	name	Extreme Random Forest
0.97	0.97	0.97	0.96	location	Random forest
0.59	0.71	0.56	1.00	location	SVM
0.82	0.82	0.81	0.84	location	Logistic regression
0.97	0.97	0.97	0.97	location	Extreme Random Forest
0.95	0.95	0.96	0.94	description	Random forest
0.51	0.66	0.50	1.00	description	SVM
0.87	0.88	0.87	0.88	description	Logistic regression
0.96	0.96	0.98	0.95	description	Extreme Random Forest
0.98	0.98	0.97	0.99	date	Random forest
0.57	0.21	0.99	0.12	date	SVM
0.90	0.90	0.89	0.91	date	Logistic regression
0.98	0.99	0.99	0.98	date	Extreme Random Forest

Table 10.3: Experiment 1: Overestimated metrics values for a different classification models and event components. The result is incorrect because in the train and test sets domain names are overlapping, that means trained classifier knows the structure of a webpage.

f1_score	mean_accuracy	precision	recall	meta_name	model
0.83	0.84	0.84	0.83	name	Random forest
0.81	0.82	0.83	0.81	name	SVM
0.74	0.74	0.76	0.76	name	Logistic regression
0.86	0.87	0.85	0.89	name	Extreme Random Forest
0.83	0.82	0.81	0.87	location	Random forest
0.82	0.82	0.84	0.81	location	SVM
0.77	0.77	0.70	0.86	location	Logistic regression
0.85	0.84	0.79	0.93	location	Extreme Random Forest
0.87	0.86	0.86	0.90	description	Random forest
0.86	0.86	0.84	0.87	description	SVM
0.85	0.85	0.89	0.83	description	Logistic regression
0.88	0.88	0.86	0.90	description	Extreme Random Forest
0.91	0.91	0.93	0.90	date	Random forest
0.89	0.89	0.88	0.90	date	SVM
0.84	0.84	0.87	0.82	date	Logistic regression
0.91	0.91	0.90	0.93	date	Extreme Random Forest

Table 10.4: Experiment 2: Metrics values for a different classification models and event components. Only numeric standardized features were used

The description of attached CD

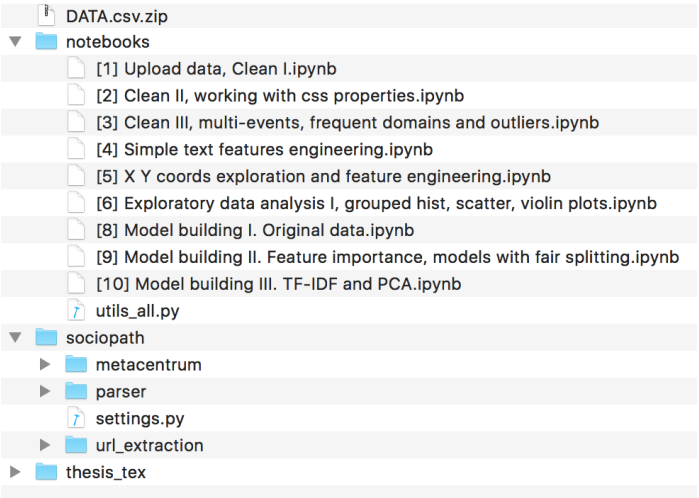


Figure 10.11: Folders structure on attached CD

- **notebooks** - the folder with Jupyter notebooks with analysis and visualizations. Notebooks include some additional images which are not listed in the thesis.
- **sociopath** - the folder with the code of Sociopath parser components
- **DATA** - published training dataset

This project code together with online versions of all notebooks available on GitHub repository: https://github.com/galinaalperovich/Ms-Thesis-CVUT