Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Dražen Perić

Modeliranje relacije med tehničnimi parametri dizajna spletnih strani in njegovo estetiko

magistrsko delo

študijski program druge stopnje
računalništvo in informatika

Ljubljana, 2015

University of Ljubljana
Faculty of Computer and Information Science

Dražen Perić

Modeling relation between technical parameters of web page design and its aesthetics

masters thesis

second cycle masters program
computer and information science

Supervisor: Assoc. Prof. Zoran Bosnić, Ph.D.

Ljubljana, 2015

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Dražen Perić

Modeliranje relacije med tehničnimi parametri dizajna spletnih strani in njegovo estetiko

magistrsko delo

študijski program druge stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Zoran Bosnić

Ljubljana, 2015

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Izjava o avtorstvu magistrskega dela

Spodaj podpisani Dražen Perić sem avtor magistrskega dela z naslovom:

Modeling relation between technical parameters of web page design and its aesthetics

S svojim podpisom zagotavljam, da:

- sem magistrsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Zorana Bosnića,
- so elektronska oblika magistrskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko magistrskega dela,
- soglašam z javno objavo elektronske oblike magistrskega dela v zbirki "Dela FRI".

V Ljubljani, 5. januarja 2015 Podpis avtorja:

I am deeply grateful to everyone who supported me during the process of writing this thesis. I would particulary like to thank to my supervisor Zoran Bosnić for all the professional help and assistance, without which this thesis would not be possible. For all the support during the study, I sincerely thank to my parents, sister, all my friends and coworkers.

Contents

List of Figures

List of Tables

List of abbreviations

abbreviation	english
HTML	hypertext markup language
HTML5	hypertext markup language - fifth version
(X)HTML	extensible hypertext markup language
XML	extensible markup language
CSS	cascading style sheets
CSS3	cascading style sheets - third version
DOM	document object model
RNN	recurrent neural network
RSS	rich site summary
W3C	world wide web consortium
JS	JavaScript
RT	regression tree
LM	linear model
NN	neural network
SVM	support vector machines
RF	random forests
KNN	k - nearest neighbors
MSE	mean squared error
RMSE	relative mean squared error
MAE	mean absolute error
RMAE	relative mean absolute error
PHP	PHP: hypertext preprocessor

JSON	JavaScript object notation
URL	uniform resource locator
CSV	comma-separated values
SQL	structured query language

Abstract

Web design has been an important part of websites for almost two decades now. It gives identity to websites and tends to deliver information to end users in a structured, organized and elegant way. Occasionally it also fails, delivering poor experience to end users who want to access information they are looking for. In this thesis, we model a relationship between the attributional description and user experience of web design, focusing on the aesthetic point of view. Our goals are the following: to define attributes which can describe web page design in a suitable way, to collect a set of websites, calculate their descriptors, supplement them with human-based aesthetics ratings through crowd-sourcing, and model the relationship between the design and the ratings by using machine learning models. We use the best performing model to develop a Chrome extension prototype that will give users the possibility to evaluate and analyze the design of websites.

Keywords:
web design, machine learning, HTML, CSS, evaluation, chrome extension

Povzetek

Spletno oblikovanje je, že dve desetletji, pomemben del spletnih strani. Podaja jim neko identiteto in se nagiba k temu, da končnemu uporabniku izroči informacije na strukturiran, organiziran in eleganten način. Ker so lahko izkušnje uporabnika tudi slabe, bomo v tem delu magistrske naloge skušali vzpostaviti odnos med spletnim oblikovanjem in uporabniško izkušnjo-predvsem z vidika estetike. Cilj je definirati in predlagati merila za ocenjevanje spletnega oblikovanja, s pomočjo ocenjenih spletnih strani (pristop "crowd-sourcing") določiti kakovost le-tega ter prikazati odnos med oblikovanjem in uporabniško oceno s pomočjo modelov strojnega učenja. Najuspešnejši model bomo uporabili za gradnjo razširitve v brskalniku Google Chrome, ki ocenjuje spletne strani in oblikovalcu oz. razvijalcu podaja priporočila za izboljšavo spletnega oblikovanja.

Ključne besede:
spletno oblikovanje, strojno učenje, HTML, CSS, ocenjevanje, chrome razširitev

Razširjeni povzetek

Splet je pomemben in vpliven komunikacijski medij, ki gosti številne spletne strani, katere se določenim uporabnikom zdijo bolj privlačne kot druge. Čeprav v praksi obstajajo smernice o razvoju učinkovitega dizajna, viri izhajajo iz ad-hoc priporočil in ne temeljijo na sistematičnih analizah. Iz slednjega izhaja potreba po sistematičnem naboru kvantitativnih lastnosti, s katerimi bi bilo možno vrednotiti oblikovanje, zanimivosti in uporabnost spletnih strani. Cilj naše naloge je poiskati odnos med tehnično zasnovo spletne strani in kakovostjo dizajna le-te, ki ga bodo subjektivno (z uporabo "crowd-sourcing" pristopa) ocenili uporabniki.

Glavni prispevki magistrske naloge so:

1. sistematičen pregled smiselnih atributov za opisovanje dizajna,
2. razlaga in interpretacija najbolje ocenjenega modela,
3. razširitev za spletni brskalnik Google Chrome, ki oblikovalcu posreduje priporočila za izboljšavo dizajna.

Za oblikovanje spletnih strani smo postavili ključne lastnosti. Pri določanju smo uporabili tako lastne ideje kot tudi usmeritve, ki se uporabljajo pri izdelavi spletnih aplikaciji, katere smo izbrali iz različnih spletnih in pisnih virov. Zgradili smo učno množico z uporabo pristopa "crowd-sourcing", kjer so uporabniki ocenili, ali jim je trenutna stran všeč ali ne. Na podlagi teh ocen (dokumentiranega modela in izvorne kode) smo zajeli podatke izbranih

lastnosti oblikovanja ter zgradili učno množico, ki je sestavljena iz 176 ocenjenih in razčlenjenih spletnih strani.

Na osnovi zbranih podatkov smo atribute ocenili z uporabo algoritma ReliefF in izbrali le 20 najbolje ocenjenih, s katerimi smo zgradili modele strojnega učenja. Z napovednimi modeli smo ocenili še neocenjene spletne strani. Za gradnjo modelov smo uporabili regresijske metode, kot so regresijska drevesa (RT), linearna regresija (LM), umetne nevronske mreže (NN), metoda podpornih vektorjev (SVM), naključni gozdovi (RF) in k najbližjih sosedov (kNN). Modele smo ocenili z uporabo 10-kragnega prečnega preverjanja in smo najboljšemu uvrščenemu modelu s pomočjo metode za razlago modelov (Štrumbelj in Kononenko, 2010) izračunali tudi razlago, ki pove katere pomembne vrednosti atributov vplivajo na dizajn. Na osnovi te razlage smo razvili "pametno" razširitev (extension) za spletni brskalnik Google Chrome, ki hrani napovedni model za kakovost dizajna, z metodologijo razlage pa zna svetovati uporabniku o možnih izboljšavah.

Glavni deli Google Chrome razširitve so:

- razširitev, ki na uporabniški strani komunicira z odjemalcem,
- jedro aplikacije, ki komunicira z vsemi komponentami, strukturira podatke in razširitvi vrne zahtevane odgovore,
- razčlanjevalnik, ki razčlenjuje spletno stran in se uporablja tudi pri grajenju učne množice,
- ocenjevalec, ki na podlagi razčlenjenih podatkov vrne oceno in razlago spletne strani,
- aplikacijski podatki, kjer se gre za statične podatke, ustvarjene med procesom analize podatkov – najboljše ocenjeni model, prvotna učna množica in seznam najbolj pomembnih vrednosti atributov.

Razširitev je preprosta za uporabo ter od uporabnika zahteva, da odpre spletno stran, ki si jo želi oceniti in da za tem pritisne gumb za evalvacijo.

Ob zagonu se podatki pošljejo v obdelavo, avtomatsko se odpre nova stran in čakajo se rezultati obdelave. Po prejemu rezultatov se le-ti prikažejo, poleg rezultatov pa se izpišejo tudi preprosta navodila za izboljševanje.

Struktura naloge je v drugem poglavju namenjena pregledu sorodnega dela in prejšnjim poskusom na področju našega raziskovanja. V tretjem poglavju definiramo mere, ki nam pomagajo pri iskanju povezave med spletnim oblikovanjem in uporabniško izkušnjo. Opisali bomo tudi proces zbiranja in gradnje podatkov. Četrto poglavje je posvečeno evalvaciji – predstavili bomo zbrane podatke o ocenjevanju atributov, gradnji modelov strojnega učenja in ocenjevanju modelov. Izbrali, opisali in razložili bomo najboljši model in ga kasneje v razvoju tudi uporabili. Proces gradnje končnega izdelka in izdelavo ter proces uporabe Google Chrome razširitve bomo opisali v petem poglavju. Zaključek in ideje za nadaljnje delo pa bodo predstavljene v šestem poglavju.

Chapter 1

Introduction

The Internet is an important and influential communication medium that hosts numerous websites, of which some are more attractive to users than others. In practice, there are guidelines for developing an efficient web design, but those are derived from ad-hoc rules and are not based on systematical analysis. Because of this, there is a need to define a systematic set of quantitative criteria in web design, the use of which would enable to evaluate the attractiveness and usability of websites.

Web design development has been around for more than two decades and it is one of the most important elements of every website online - both for users and developers. Visual standards of web design have changed a lot from the beginning and have been frequently adapting to user needs and to all kinds of new and popular devices - from personal computers, notebooks, to smart phones, tablets and everything in between [1].

Today, many skills and techniques are influencing web design - typography, page layouts and graphics [2] are just some of those. Standards and trends are changing fast and web professionals need to be in touch with all updates on daily basis. A lot of research is spent on user experience, analysis of how users interact with websites and what can be done to make website experience even better.

In this thesis, we aim to model the relation between technical description

of website design and its aesthetic subjective rating, given by web users. In the first step - we propose a set of attributes which are used for measuring and describing web design. We collect a list of websites and a list of their ratings (using the crowd-sourcing [3] approach) that are afterwards used for building prediction models using machine learning methods - and from those models, we select the one with the best performance. Next, explanation methodology (Štrumbelj and Kononenko, 2010) is used to define important attribute values that are affecting attractive design, based on which we can suggest possible design improvements. As the final product, we develop a Google Chrome extension that can help developers during website production. We will be focusing on three main contributions, which are:

1. a systematic overview of meaningful attributes for describing and measuring web design,
2. explanation and interpretation of the best evaluated model - we plan to explain how individual attributes affect the estimated design,
3. a Google Chrome extension that will help developers during website production.

This paper is structured as follows. In Chapter 2, we review the related work and we talk about previous attempts that were similar or somehow related to our work. In Chapter 3, we define the attributes to help us find relationship between web design and user experience, and we describe the process of collecting the data and building a final data set. Chapter 4 is dedicated to the evaluation - where we present the collected data and where we talk the evaluation of attributes, the building of learning machine models and the evaluation of those. We also choose, describe and explain the best evaluated model, which we use later in development. Chapter 5 is dedicated to the building process of the final product - where we explain the Chrome extension, how it is done and how it should be used. To summarize, we finish with a meaningful conclusion.

Chapter 2

Related work

Machine learning has already been used in the field of web design, but with different goals. In the following chapter, we overview different approaches to modeling and analyzing web page design using machine learning approaches.

The Webzeitgest tool [4] was built for understanding design demographics, automating design creation and supporting data-driven design tools. It is intended for crawling over 100,000 web pages, their attributes, structural properties and vision descriptors. It offers direct access to specific page elements using a query-based access type - thus allowing to search on different web pages for elements such as horizontal layouts, canvas, custom fonts, etc. Because of that, it has been also used to create data sets for other projects related to machine learning and web design [5, 6].

At the Stanford University, M. Lim, A. Satyanarayan and S. R. Klemmer have built a platform [5] that uses a web crawler and proxy server to build a complete corpus of raw web page resource content which is afterward processed with Bento, a page segmenting algorithm that uses the Document Object Model (DOM) tree of the page as the starting point for segmentation. The final goal was to build a platform, that is able to help designers to find relative examples based on the searched keywords. This enables users to find websites based on their complexity and form - such as modern/creative/minimalistic websites, or even websites that use just dark colors, etc.

The project "Structural Learning for Web Design" [6] has similar aims as our thesis. Its idea was to create an automatic predictor which can decide whether a given web page looks minimal or modern. They use web elements provided by Webzeitgeist and Recurrent Neural Networks (RNN) to build an underlying classifier.

Google also did some research in this field back in 2005, named "Web Authoring Statistics". There are no written papers about it but it is all covered on Google's developers website [7]. They have aimed at discovering which HTML elements are primarily used and which attributes are primarily used on those HTML elements. It is good to mention that Google's study was also based on some previous studies by individuals like Marko Karppinen ("Marko Karppinen did a study in 2002, looking at which of the then 141 W3C members had sites that validated") [8] and John Allsopp ("John Allsopp's study is the most recent one we are aware of, where he looked at class and id attribute values on 1315 sites.") [9].

Another Stanford University project in which they tried to learn structural semantics of the web is also worth mentioning [10]. Their goal was to build a classifier that can help recognizing DOM objects. They took the crowd-sourcing approach using Amazon's Mechanical Turk, where each of the participants semantically labeled at least ten page elements of five given web pages. The biggest problem while building a classifier was that the web is still largely unstructured, and even if there are HTML5 semantic tags like article, nav, figure etc, it is still hard to rely on web designers and web developers to annotate pages with semantic markup elements such as these.

What is interesting for users and what is considered to be a good design, is described by Perkowitz and Etzioni [11], who refer to user experience and use of artificial intelligence for adaptive web sites - sites that automatically improve website organization and presentation based on user access data. Even if this reference is a bit older, it is interesting for us because of the described attributes which were taken into account.

Chapter 3

Data preparation

In this chapter we define and describe 37 attributes which help us to find a relationship between web design and user experience. From the list of 200 websites, we build a learning data set by using the crowd-sourcing [3] approach and a custom made website parser. The resulting learning data contains 176 evaluated and parsed websites.

3.1 Definition of attributes

The first step of website parameterization is to define attributes that can be used for describing web design and afterwards for finding the relationship between user design and user experience in terms of aesthetics. To define suitable attributes, we have drawn from various web design guidelines [12], user experience principles and common sense. We selected 37 attributes altogether. They are as follows:

1. **Amount of text**

   As websites are completely different from books, it is necessary that they have to adjust to different standards. The amount of text mostly depends on the website's content type (for example, blogs will contain more text than company websites). An effective website experience allows the user to reach the essential website content as quickly as

possible, categorize all the content, and use multiple heading levels. The amount of text is simply used to describe how much letters of text (clean text without any HTML tags) is used on the website.

2. **Number of HTML elements**

   We simply count how many HTML elements are used altogether on the webpage.

3. **Number of headings**

   Since users may orient themselves within web pages using headings, it is important to use them to emphasize the document structure. Also, search engines use headings to index the structure and content of web pages. Headings are defined with *h* tags, from *h1* to *h6*, where *h1* defines the most important heading and *h6* defines the least important heading. *h1* headings should be used as main headings, followed by *h2* headings, then the less important *h3* headings, and so on [13]. Number of headings is the second attribute that we have taken into account.

4. **Number of paragraphs**

   A lot of users find too much text not interesting - however, this also depends on their interests. The most common way to draw users into important segments of a text and to support readability is to split large amount of texts into shorter paragraphs. There is no formula to calculate how long the paragraph should be so we decided only to count the number of paragraphs on the website.

5. **Number of images**

   Another way to split large text blocks is using images, and since images are easier to remember, we have taken the number of images as another attribute. We count images by checking how many *img* tags are used.

6. **Number of font families**

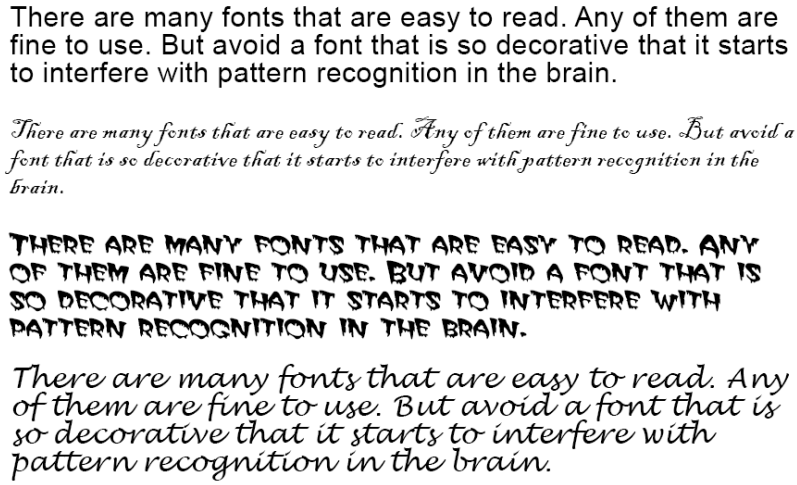

Figure 3.1: Different decorative fonts

People have been debating for a long time which fonts are better, easier to read, or the most appropriate. Designers are using fonts to raise the mood, express the brand, or to make associations with the text. In terms of readability, the chosen font is not of critical importance as long as it is readable; some fonts interfere with the brain's ability to recognize patterns. Figure 3.1 shows different decorative fonts. The first font is relatively easy to read; the others become progressively more difficult. The lack of typographic consistency is one of the biggest mistakes that web designers make. A large number of font families used on websites confuses website readers. Good practice is to use only two font families - one for regular text and the second to emphasize important text. The number of font families is therefore another attribute we considered for describing web design.

7. **Number of font sizes**

In a book about design principles and how people interact with them [14], Susan Weinschenk said: "When it comes to fonts, size matters a lot.

The font size needs to be big enough for users to read the text without strain. And it's not just older individuals who need fonts to be bigger - young people also complain when font sizes are too small to read". Some fonts can look bigger but they can actually have the same size - and this is due to the x-height which refers to the distance between the baseline and the mean line - or literally, as the height of the small letter x (as well as the u, v, w and z) in the selected font family. Designers should choose a point size that is large enough for people of various ages to read comfortably and use a font with a large x-height for online viewing so that the type will appear to be larger. The number of font sizes is another attribute which describes web design. For the number of font sizes we count all font sizes in pixels (px) and relative measurement units in percents (%) and units em (which is a unit of measurement equal to the currently specified font-size property). We say relative because those measures depend on the current font-size. For example, if the current font-size is 16px, 80% will be 12,8px and for 32px will be equal to 2em (em's are scalable).

8. **Number of colors**

"When applied to Web design, color may impact the user's expectations about navigation, links, and content, for example, as well as overall satisfaction" is what Wendy Barber and Albert Badre wrote in their article about the merging of culture and usability [15]. Colors have a huge impact on user experience and communication, and should be carefully combined so that they make visual sense and attract users. Colors should also be carefully picked depending on the target audience. As Badre and Barber also mentioned in their article, some colors have different meanings in different countries - so for example, if American bank wants to use green on their website to promote services for French investors, they should better avoid that, because some French can associate green with criminal. On the other hand, for Muslims and Middle Eastern people, green carries a positive connotation. On web

pages, colors can be used in different contexts and can be applied to all HTML elements. Colors can not only bring joy and make for a better user experience, but also destroy it if they are not used properly. We used the number of different colors as another attribute.

9. **Number of links**

   Hyperlinks are part of the HTML standard from the beginning [16], and are used as internal connections to sections within a page, or as external connections to other pages. Links should be useful and understandable so that users can effectively achieve the objective. We chose the number of links on a website as another attribute for describing web design.

10. **Number of div elements**

    One of the prime goals of transition from HTML to (X)HTML/CSS is to make HTML documents more meaningful by removing code such as font tags, div tags and similar. Div elements should only be used when grouping related elements together. For example, on a news website, the story contains a title, paragraphs, tables, lists and so on, so it seems reasonable to wrap all of these in one div. On the other side, it is recommended to replace *div*s with leaner and more meaningful elements that were introduced in HTML5 [17], such as *article*, *section*, *header*, *footer*, *nav*, etc. The number of *div* elements used on a website is another attribute that we use for describing web design.

11. **Number of id and class elements**

    When we replace div elements with more meaningful mark-up, it is easier to apply a style without using a particular element id. Namely, the class selector allows us to set multiple styles to the same element or tag in a document. We consider the number of id and class elements as another two attributes that we use to describe web design. The number of id elements tells us how many unique elements are there on the website; and the number of class elements tells us how many

repetitive elements there are on the web page.

12. **Number of CSS sources**

    CSS (Cascading Style Sheets) is independent of HTML and can be used with any XML-based markup language. The separation of HTML from CSS makes it easier to maintain sites, share style sheets across pages, and tailor pages to different environments. This is referred to as the separation of structure (or content) from presentation. To derive attributes, we separately count the number of external CSS sources, internal CSS sources and the number of inline CSS definitions.

13. **CSS prefixes**

    Although modern browsers support most of CSS3 properties, most designers and developers seem to focus only on a small set of common properties that are well documented, well tested and frequently used. Browser makers can use the so-called CSS vendor prefixes (such as "-moz-border-radius" which works only in Firefox, or "-webkit-border-top-left-radius" which works in Chrome and Safari) or CSS browser prefixes to add support for new CSS features that are still in the testing and experimentation period. Browser prefixes are used to add new features that may not be part of the formal specification and to implement features in a specification that has not yet been finalized. Despite that these properties are not official; many designers use them to achieve more advanced design. We use this fact to check if a designer is using experimental CSS properties, and store the finding as an attribute.

14. **Number of JavaScript sources**

    JavaScript is a scripting language commonly implemented as a part of web browser in order to create enhanced user interfaces and dynamic websites. JavaScript can be used for user interaction (animations, validations, etc.), downloading or sending information via Ajax, etc. To derive an attribute, we count the number of used JavaScript source fields,

which are included using *script* tags with attribute set to "JavaScript".

15. **Number of meta tags, and checking whether meta keywords and meta descriptions are in use**

    Meta tags provide metadata about the HTML document. Metadata are not displayed on the page, but they can be parsed. Meta elements are typically used to specify page description, keywords, author of the document, and last modified time. Most search engines use metadata a part of the algorithm for ranking websites. To define attributes, we count how many meta tags are used altogether, and whether website is using meta keywords and meta description tags.

16. **HTML errors**

    For describing web design we also check if the website has a valid HTML. We do this by using Application Programming Interface from the World Wide Web Consortium (W3C).

17. **RSS feed**

    By checking if there is a link tag with type "application/rss+xml", we detect whether the website is using a RSS feed. RSS or Rich Site Summary is actually an XML-based format for sharing web site content such as blog posts, status updates or similar [18].

18. **Import CSS rule**

    @import command is used to import one CSS file into other, and is often used in modern CSS techniques. To define an attribute, we check if there exist any *@import* rules in the CSS source code.

19. **Twitter bootstrap**

    Bootstrap is a front-end framework, developed and maintained by Twitter. It is mostly used by developers in the initial step of the projects and it contains HTML and CSS based design templates. It is also one of the most popular open source projects nowadays [19]. To detect

whether this framework is used, we check whether any of the link tags contain "bootstrap" keyword.

20. **HTML5**

    The HTML5 standard has been around for quite a few years already and its core aim was to improve user experience on the Internet and to give more power to the developers [17]. Within our attribute, we check whether HTML5 is used at all and how many HTML5 are used altogether.

21. **CSS transitions**

    CSS transitions were introduced in the CSS3 standard and are used to add effects to HTML elements without using Flash or Javascript. We defined an attribute that states whether CSS rules that apply to transitions are used.

22. **Flash**

    Adobe Flash is a quite mature technology that is being used less and less. We counted objects and *embed* tags which are a type of "application/x-shockwave-flash".

23. **Page size**

    We measure page size in kilobytes. Size can depend on assets, amount of code etc.

24. **Media queries**

    CSS3 media queries [20] are a basis of responsive web design and are used to adjust the content to specific range of output devices without the need to change the content itself. We check if media queries are used at all.

25. **Conditional comments**

Conditional comments work only in Internet Explorer (IE) and are used to give instructions to old IE browsers. Those comments are supported from IE5 to IE9 and are mostly used to load different kind of CSS styles for different version of IE. We store information about whether conditional comments are used or not.

26. **Included multimedia**

    Multimedia can be embedded using several methods. We check whether the website uses some of the tags for including multimedia, like *video*, *embed*, *object*, *audio* and *source*.

27. **Minified CSS**

    Since the minification of CSS sources is increasingly popular and saves a lot of bandwidth, our attribute states if websites use this kind of approach.

28. **Reset CSS**

    Eric Meyer's "Reset CSS" [21] is also a quite popular technique, the main goal of which is to reduce browser inconsistencies in default line heights, margins, font sizes of headings, etc. We defined an attribute that checks whether "reset.css" is included in the source code.

29. **Normalize CSS**

    Normalize.css is actually just an alternative to reset.css [21] and is a customizable CSS file that makes browsers render all the HTML elements more consistently. We check whether "normalize.css" is included.

30. **CSS pseudo elements**

    Pseudo elements are used to define styles only for specified parts of elements, such as first lines, first letters, first child etc. We check whether those elements are used in the CSS code.

31. **no-js**

"no-js" is a Modernizr [22] technique which is used to detect whether Javascript support is enabled or not. This class is assigned to the *html* tag of an website. If Javascript is supported, Modernizer replaces "no-js" class with "js", otherwise it leaves "no-js".

All 37 defined attributes are listed in Table 3.1, sorted alphabetically by attribute name, with short description in the second column and data type in the third column.

Attribute	**Description**	**Type**
Classes	Number of class attributes in the HTML code	Numeric
Colors	Number of colors	Numeric
Conditional comments	Checks whether CSS conditional comments are in use	Boolean
CSS prefixes	Checks whether CSS browser prefixes are used	Boolean
CSS pseudo elements	Checks whether CSS pseudo elements are in use	Boolean
CSS transitions	Checks whether CSS transitions are in use	Boolean
Divs	Number of *div* HTML tags	Numeric
Elements	Number of all HTML elements	Numeric
External CSS sources	Number of external CSS sources	Numeric
Flash	Checks whether Flash is in use	Boolean
Font families	Number of all font families	Numeric
Font sizes	Number of font sizes (assigned with pixels, % and em)	Numeric
Headings	Number of headings (*h1*, *h2* etc.)	Numeric
HTML errors	Number of HTML errors found by the W3C CSS validator	Numeric

HTML5	Checks whether HTML5 elements are in use	Boolean
HTML5 elements	Number of all HTML5 elements that are in use	Numeric
Ids	Number of *id* attributes in HTML code	Numeric
Images	Number of images included with *img* tag	Numeric
Import	Checks whether CSS *@import* command is in use	Boolean
Included multimedia	Checks whether website is using some of the multimedia tags	Boolean
Inline CSS definitions	Number of inline CSS definitions	Numeric
Internal CSS sources	Number of internal CSS sources	Numeric
JS sources	Number of all JavaScript sources	Numeric
Links	Number of links	Numeric
Media queries	Checks whether media queries are in use	Boolean
Meta description	Checks whether meta description is in use	Boolean
Meta keywords	Checks whether meta keywords are in use	Boolean
Meta tags	Number of meta tags in the *head* section of HTML file	Numeric
Minified CSS	Checks whether CSS is minified	Boolean
no-js	Checks whether no-js technique is in use	Boolean
Normalize CSS	Checks whether normalize.css is in use	Boolean
Page size	Page size in kilobytes	Numeric

Paragraphs	Number of p tags	Numeric
Reset CSS	Checks whether reset.css is in use	Boolean
RSS	Checks whether RSS is used or not	Boolean
Text	Counts the amount of letters in text	Numeric
Twitter bootstrap	Checks whether Twitter bootstrap framework is in use	Boolean

Table 3.1: List of defined attributes

3.2 Construction of learning data by extraction of popular websites

In this section we describe how we constructed our learning set that included values of previously described attributes for a large set of websites. We started by downloading the data of about one million websites [23] from Alexa[1], which is a company that updates the database of website urls and ranks o a daily basis.

We proceeded by manually filtering available websites to select only their subset for further processing. We removed all websites that were not fully available online, all non-English websites and websites that included pornography or had security issues. Our extensive manual effort resulted in selection of 200 websites that we described using attributes defined in Section 3.1.

To receive website ratings, which we planned to correlate to our attributes, we decided to use the crowd-sourcing [3] approach. For this task we developed a web application using the Ruby on Rails framework, which we named Wote (Figure 3.2) and used it to receive website ratings. The applica-

[1] Alexa - Alexa Internet, Inc. is a California-based subsidiary company of Amazon.com which provides commercial web traffic data

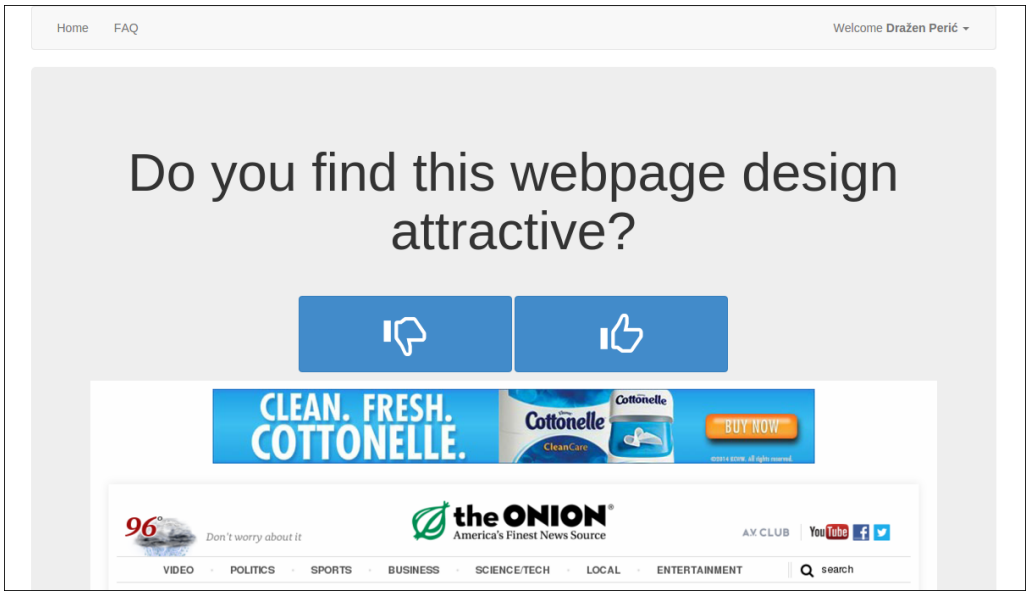

Figure 3.2: Screenshot of the Wote application

tion's basic functionality was to provide "upvote" and "downvote" buttons to users and enable tem to express whether they found the displayed web design attractive or not. Each user got to see each website once.

We set up the Wote application at Heroku[2] and announced it on social networks such as Facebook and Twitter, as well as on the Faculty's mailing list. We collected 3448 votes from 104 unique users for 186 valid websites (14 websites were excluded from the survey due to technical problems with screenshot generation).

3.3 Parsing websites

To extract attributes from the websites, we developed a custom automated parser. The parser was built partially with PhantomJS WebKit [24], which enabled easy extraction of data with jQuery selectors, and partially with

[2]Heroku - cloud based platform as a service (PaaS), that supports several programming languages

Python programming language, which enabled secondary tasks such as generating and analyzing website screenshots.

We implemented the parser to output all the attributes listed in Table 3.1 for each of the input websites. Additionally, we also used an open source project called colorific[3] (built with Python) to detect the number of different colors. This library detects the color palette used in a given image - which was the website screenshot in our example. After developing the parser, we published it as an open source project at Github[4].

After using the parser on all available websites, we transformed the calculated attribute values into our learning set. To do this, we supplemented each attribute vector with an aggregated rating for that site, which was computed as an number of all votes (upvotes and downvotes), divided with the number of upvotes, as shown in Equation 3.1. This result was then rounded to two decimals and normalized so that the lowest rating value is 0.1, and the highest rating value is 1.

$$finalgrade = \frac{upvotes + downvotes}{upvotes} \tag{3.1}$$

Since the relative difference between upvotes and downvotes conceals information about the total number of votes, we decided to weigh the data so that websites with more votes achieve higher weight, and websites with less votes achieve lower weight. We performed this by multiplying the rows as many times as the number of votes that the website in that row received. At the end, our extended final dataset had 3265 rows altogether.

[3]colorific - https://github.com/99designs/colorific

[4]wparser - https://github.com/peric/wparser

Chapter 4

Evaluation

In this chapter we evaluate the attributes from the Section 3.1 and build the prediction model using the best attributes. After building the best evaluated model, we use the explanation methodology [25] to explain and interpret how individual attributes affect estimated design.

4.1 Data set statistics

Tables 4.1 and 4.2 show basic statistics of the used dataset. Additionally, Figure 4.1 displays the distribution of web page ratings in the dataset, where we can see that the majority of the websites has ratings close to the middle.

Attribute	**Mean**	**Median**	**Min**	**Max**
Classes	120.09	102	0	547
Colors	8.25	7	1	32
Divs	209.48	119.50	0	2591
Elements	951.50	720.50	6	11845
External CSS sources	2.79	2	0	14
Font families	4.43	4	1	20
Font sizes	10.72	10	1	23
Headings	27.61	16	0	313
HTML errors	67.15	31	0	980
HTML5 elements	15.06	1	0	169
Ids	72.66	51	0	1021
Images	34.22	22.50	0	247
Inline CSS definitions	54.61	31	0	1114
Internal CSS sources	3.38	1	0	240
JS sources	21.68	16	0	119
Links	190.99	139	0	1080
Meta tags	10.56	8.5	0	39
Page weight	122.23	72.98	0.25	3314.14
Paragraphs	27.18	12	0	932
Text	17556.19	8779.50	4	259942

Table 4.1: Data set statistics for numeric attributes

Attribute	Yes	No
Conditional comments	128	48
CSS prefixes	96	80
CSS pseudo elements	65	111
CSS transitions	74	102
Flash	7	169
HTML5	99	77
Import	12	164
Included multimedia	16	160
Media queries	124	52
Meta description	139	37
Meta keywords	78	98
Minified CSS	25	151
no-js	29	147
Normalize CSS	2	174
Reset CSS	36	140
RSS	5	173
Twitter bootstrap	120.09	102

Table 4.2: Data set statistics for binary (boolean) attributes

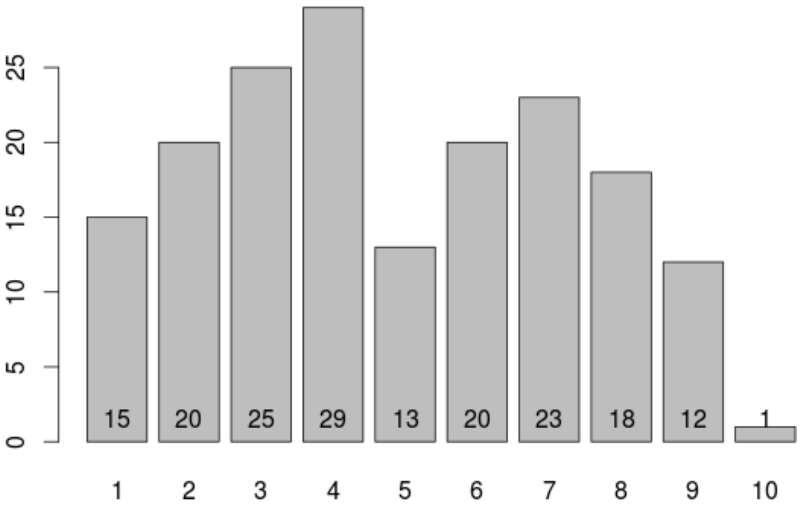

Figure 4.1: Histogram of ratings

4.2 Selection of best attributes

For the evaluation and selection of attributes, we used the ReliefF algorithm [26], more specifically - the RReliefF algorithm [27], which is adapted for regression problems.

ReliefF and RReliefF algorithms are both based on the Relief feature selection algorithm and were introduced by Kononenko et al. [26]]. The main difference between the two is in the distance measure (The Manhattan norm is used instead of the Euclidien norm). RReliefF also considers the context of other attributes, which means that it is not myopic. For each attribute, RReliefF finds the closest attribute in the same class and the closest attribute in the opposite class and rewards the attribute if the nearest example from the other class has a different value of that attribute. Conversely, algorithm punishes an attribute if the nearest example from the same class has a different value of that attribute. RReliefF returns attribute quality estimates from 0 to 1, where important attributes carry larger weights. We decided to use three implementation of the RReliefF algorithm, which are implemented in the statistical package R (they are RreliefFdistance, RreliefFexpRank and RReliefFbestK [28]). RReliefFdistance algorithm weights k nearest instances with their inverse distance from the query instance. RreliefFexpRank is an algorithm which exponentially decreases the weight of the k nearest instances with increasing rank; and RReliefFbestK algorithm considers all neighbors as possible nearest instances and returns the highest found score.

Table 4.3 shows the final results of attribute evaluations using the mentioned RReliefF algorithms, along with the average result. Results are ordered descending by the average.

	RReliefF			
Attribute	**distance**	**expRank**	**bestK**	**Average**
External CSS sources	0.468	0.481	0.889	0.613
Font sizes	0.477	0.510	0.768	0.585
Font families	0.380	0.373	0.768	0.507

Colors	0.487	0.494	0.521	0.501
Meta tags	0.466	0.451	0.520	0.479
Classes	0.280	0.290	0.768	0.446
JS sources	0.377	0.402	0.468	0.416
Links	0.286	0.296	0.480	0.354
Images	0.3	0.290	0.459	0.350
Media queries	0.121	0.112	0.768	0.334
Ids	0.101	0.113	0.768	0.327
Divs	0.113	0.120	0.614	0.282
Headings	0.184	0.198	0.257	0.213
Text	0.123	0.136	0.310	0.190
HTML5 elements	0.189	0.179	0.180	0.183
Meta keywords	0.190	0.179	0.178	0.182
Conditional comments	0.143	0.155	0.232	0.177
HTML errors	0.183	0.173	0.173	0.176
Meta description	0.119	0.107	0.232	0.153
RSS	0.141	0.157	0.159	0.152
CSS prefixes	0.094	0.092	0.268	0.151
no-js	0.118	0.131	0.187	0.145
HTML5	0.158	0.104	0.154	0.139
Inline CSS definitions	0.096	0.109	0.197	0.134
Elements	0.094	0.092	0.213	0.133
Minified CSS	0.128	0.102	0.158	0.129
Page weight	0.043	0.050	0.243	0.112
CSS transitions	0.078	0.080	0.142	0.100
Included multimedia	0.072	0.090	0.114	0.092
Import	0.082	0.081	0.098	0.087
CSS pseudo elements	0.090	0.055	0.087	0.077
Paragraphs	0.063	0.061	0.059	0.061
Twitter bootstrap	0.055	0.057	0.059	0.057

Flash	0.031	0.028	0.037	0.032
Normalize CSS	0.013	0.015	0.021	0.016
Internal CSS sources	0.006	0.007	0.029	0.014
Reset CSS	0.004	0.004	0.006	0.008

Table 4.3: Results of attribute evaluations

We selected the first 20 best-ranked attributes from Table 4.3 and used them to build machine learning models. In the set of best evaluated attributes, we can see that there are attributes of various kinds: some of them are entirely visual attributes, and there are also some that are not visible on the websites at all. Among the former we consider the font sizes and font families, the number of colors, number of images, number of links, and classes; and among the latter we consider the number of external CSS sources, number of meta tags, etc.

4.3 Regression

Since we are dealing with ordinal target value (ratings have values from 1 to 10, where 1 represents the worst and 10 represents the best design), we used regression prediction models. We used six different machine learning algorithms, which are implemented within R: regression trees (library rpart [29]), linear regression (library lm [30]), neural networks (library nnet [31]), support vector machines (library e1071 [32]), random forests (library randomForest [33]), and k-nearest neighbors library kknn [34]). For building models we used the 20 best evaluated attributes, as it was mentioned in Section 4.1, and we used the extended data set, mentioned in Section 3.2, that contains 3265 rows.

4.3.1 Regression Tree (RT)

Regression trees are similar to decision trees. They consist of internal nodes that correspond to attributes, edges that correspond to subsets of attribute values, and leaves that correspond to function mapping from attribute space to continuous target variable. In leaves, we can have a constant, linear or arbitrary function. Usually, a constant function that calculates the mean of the target variable of learning examples is applied in the leaves [35]. We use the rpart [29] library to build regression tree.

4.3.2 Linear Model (LM)

The linear model is based on linear equation and tells how much of an effect x has on y. In our case, we deal with a multi-variable linear model that aims to predict the website's rating based on the variables that are actually attributes and their values. The model is represented by such a line through data points so that the total distance of all points to the line is minimized. The linear model seeks for a best fit between dependent and independent variables using the learning data:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + ... + \beta_n x_0 n \tag{4.1}$$

where x_i's are independent variables (number of links, number of images, etc.), β_0 is everything that is still left out and all the remaining β_i's are coefficients belonging to the attributes. Here, we use the already built-in R method to build a model [30].

4.3.3 Neural Network (NN)

Artificial neurons, which are small, independent building blocks, are the main component of artificial neural networks. Those building blocks are abstractions of biological neurons. The artificial neurons are interconnected and can compute values from inputs that are transformed with a threshold function f into an output signal. Neurons are grouped into several layers (input layer,

zero or more hidden layers, and output layer). Each layer forwards its input signals to all neurons within the next layer, and the output signals of the last layer represent the model's prediction [35]. In our case, we use artificial neural network with a single hidden layer, and nnet [31] library to build it.

4.3.4 Support Vector Machines (SVM)

Support vector machines, although initially developed for classification, were also adapted for regression. They are defined by support vectors, which are learning examples that lie closest to the hyperplane that is positioned through the example space to minimize the examples' margin. The margin within regression SVMs is defined in such a way that the correctly predicted learning examples lie inside of it. Criterion function in regression SVMs is also optimized with respect to the prediction error of the target variable, as well as with respect to the complexity of the criterion function [35]. We use the e1071 [32] library to build the SVM model.

4.3.5 Random Forests (RF)

Random Forests appeared as a result of combining and improving various decision tree algorithms; nevertheless, they are likewise used for regression trees as well. Random Forests for regression generate a series of regression trees that limit the selection of the best attribute in each node to a relatively small subset of randomly chosen candidate attributes. They are among the best algorithms, however, they have a problem with interpreting the combined answers from the set of 100 or more decision/regression trees [35]. We use the randomForest [33] library to build model.

4.3.6 k - nearest Neighbors (kNN)

Nearest neighbor methods use different approach than other machine learning methods as they store the entire set of examples. After a new example is presented to the predictor, a subset of the most similar learning examples

	RT	**LM**	**NN**	**SVM**	**RF**	**K-NN**
MSE	0.103	0.066	0.078	0.086	0.057	0.126
RMSE	1.744	1.110	1.316	1.446	0.955	2.131
MAE	0.261	0.213	0.225	0.236	0.203	0.286
RMAE	1.228	1.0	1.056	1.109	0.954	1.344

Table 4.4: Results of evaluated regression models

is used to make a prediction. Since there is hardly any learning, nearest neighbor methods are characterized as lazy learning methods. The kNN method is actually the simplest of nearest neighbor methods [35]. Parameter k defines how many nearest examples are taken into account for prediction of a new example, which is then calculated using a simple equation that averages the target values of the neighbors:

$$C_x = \frac{1}{k} \sum_{i=1}^{k} C_i \tag{4.2}$$

where C_i denotes a particular attribute value of the nearest example. We use kknn [34] library and three $k = 3$ nearest neighbors.

4.4 Results with regression

To decide which model works best we needed to evaluate them. We calculated mean squared error (MSE), relative mean squared error (RMSE), mean absolute error (MAE) and relative mean absolute error (RMAE) by using 10-fold cross validation [36]. In each step, the dataset is split into ten subsets. Then, nine subsets are used as the learning set as the model performance is evaluated on the remaining subset. Results of evaluations are shown in Table 4.4.

Values for RMSE and RMAE usually take values that are higher than zero and lower than one (values greater than one indicate that the model

performs worse than the default predictor that always predicts the mean value). The perfect model is the one that has RMSE/RMAE values equal to zero. In our case, we can see that the best performing models are the Random Forests (RF) and the Linear model (LM). Neural Networks (NN) and Support Vector Machines (SVM) follow their performance with slightly worse results.

4.5 Explanation methodology

Based on our results we decided to use the Random Forests to further explain the induced knowledge and use it to recommend improvements to webpage developers. To calculate explanations, we used the methodology, developed by Štrumbelj and Kononenko [25]. This methodology was developed to explain whole regression models as well as the predictions for individual examples.

4.5.1 Explanation of the entire prediction model

We started by explaining the whole prediction model and continued by explaining individual predictions (in the next Section). Figures 4.2 and 4.3 show the explanation for the model built with Random forests, where we can see how values of individual attributes influence the model's predictions. Due to space limitations, each figure contains explanations for ten attributes. Values for continuous attributes are split in 5 intervals, and values for discrete attributes are split in two. These intervals are marked on the left side (annotation of the vertical axis).

The figures reveal how individual attributes contribute to the whole model: each plotted bar represents a contribution of that attribute value. Negative contribution represents negative impact (lower prediction of a rating) and positive value means positive impact (higher prediction of a rating). For example, we can see how a small number of "links" had big a big positive impact and how large number of links had big negative impact. The first

bar of each attribute (the bar on the top, which is in line with the attribute name), shows average negative and positive contributions of all attribute values and serves as a summary for that attribute.

We can quickly see that the greatest positive impact has been given to smaller values of "links", "text", "meta_tags", "js_sources", "images", "ids", "html_errors", "html5_tags", "headings" and "colors". Positive impact has been also given by larger number of "classes", "font_sizes", "font_families", "divs", "css_external" and "classes". In case of binary attributes, we can see that there is not a lot of difference whether the value is true or false. But we can still say that true values of "conditional_comments", "has_meta_description", "media_queries" and "rss" bring more positive impact then the negative ones, and "has_meta_keywords" has basically the same impact for both values. We can also summarize the following:

- RSS is mostly used by websites that publish content (such as news) on a daily basis. Those websites are nowadays rather piled up and full of ads, images, links and similar - which affects the user experience from the aesthetic point of view,
- If there are many external CSS files in use, then the website is probably not using modern CSS technologies such as preprocessors and minifiers. Consequently, the website gets heavier as more and more content is loaded. We can reach a similar conclusion in regard to external JS files,
- Too many colors, font sizes, font families and so forth, can easily have a negative impact on user experience,
- It is better to use less text, images, links, external JS sources, and different headings.

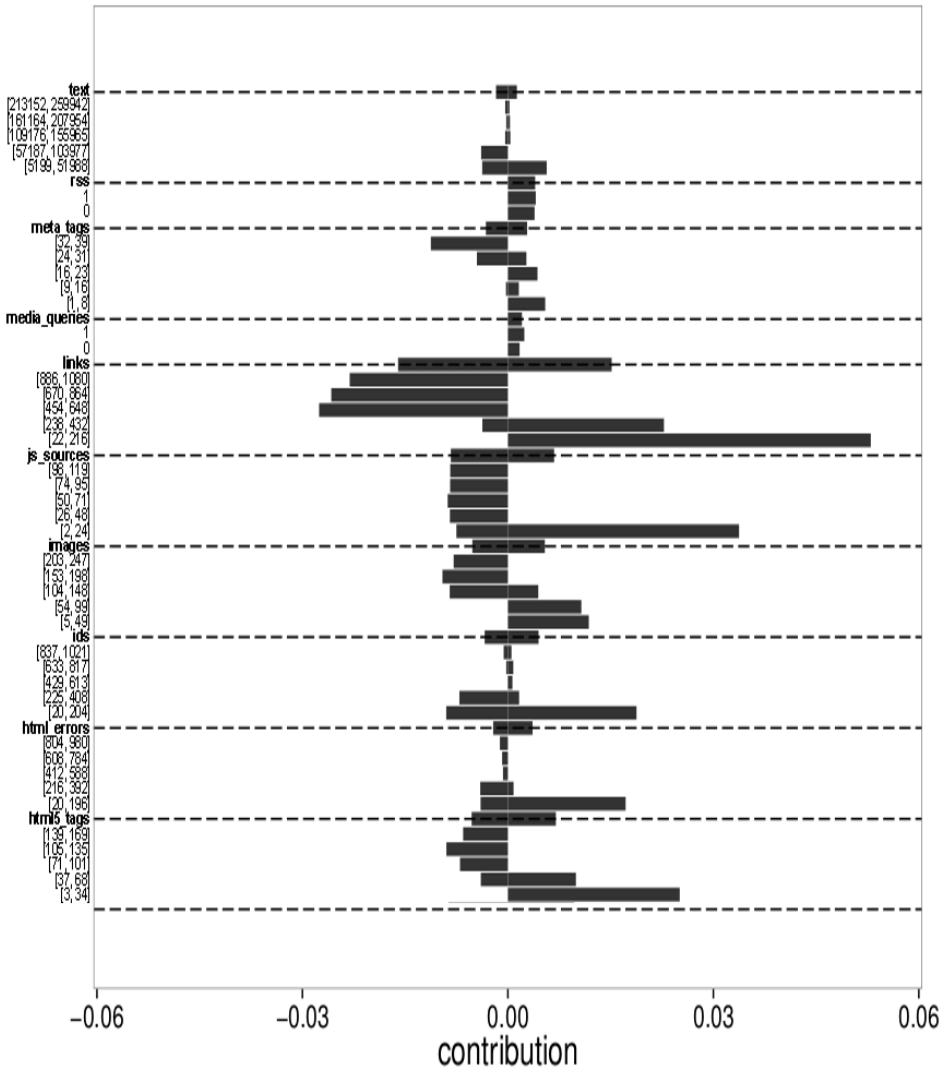

Figure 4.2: Model explanation

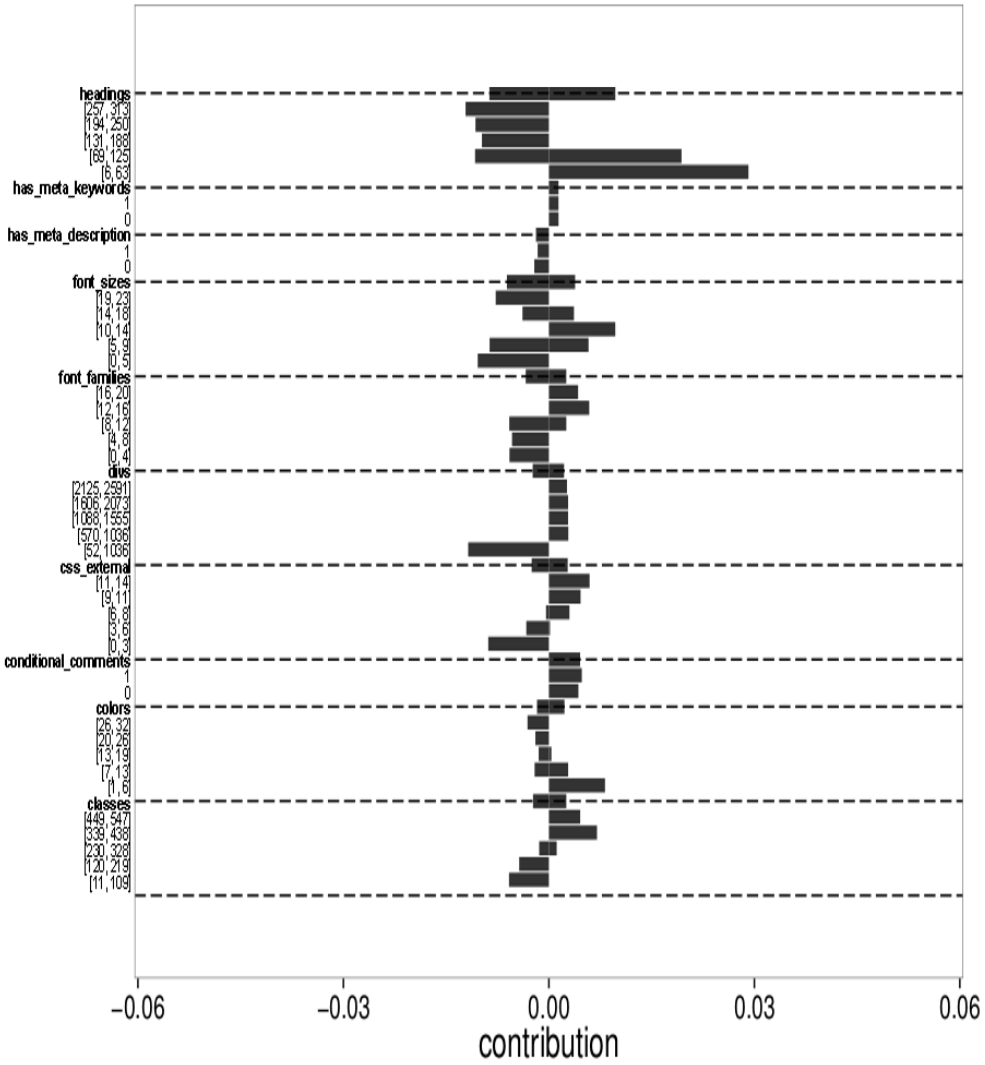

Figure 4.3: Model explanation

4.5.2 Explanation of individual examples

To perform the explanation of individual examples, we first selected three individual examples from the dataset. With the aim of using varied examples, we took one website that received a bad rating (tmz.com, rated with 1), one that received an average rating (piriform.com, rated with 5) and one that received a good rating (getpocket.com, rated with 8.8).

We evaluated the three chosen examples using the provided by explanation methodology. Results are shown in Figures 4.4 - 4.6, where we can see how values of individual attributes influence the rating that particular website has received. Negative contribution means negative impact (lower rating) and positive contribution means positive impact (higher rating).

Figure 4.4 shows instance explanation for tmz.com. The actual (evaluated) rating of that website was very bad (1) and the predicted rating was 3.5, as shown at the top of the figure. From the explanation, we can see that only the following attributes had a small positive impact: “text”, “media_queries”, “images”, “font_sizes” and “colors”. On the other hand, we can see that the largest negative impact was caused by values of attributes “links”, “ids”, “headings”, “divs”, “classes” and “js_sources”. We can also see that these findings are consistent with the explanation of the entire model that is shown in Figure 4.3. We can conclude that the site tmz.com features too many links (these can be either true links or ads), which might be too associative with spam. There are obviously also too many headings which affect readability, search-ability and general user experience.

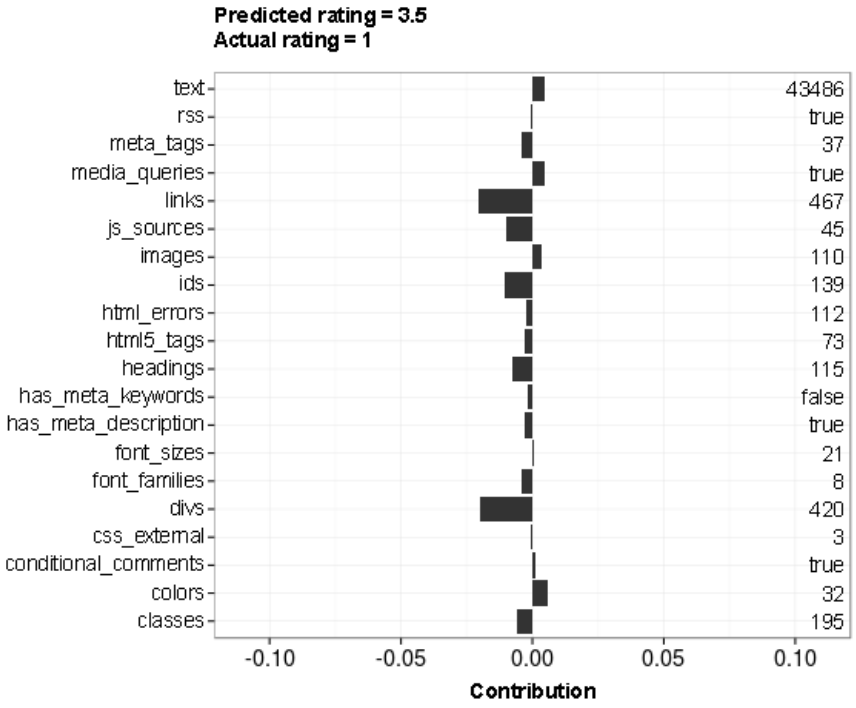

Figure 4.4: Instance explanation for tmz.com

Figure 4.5 shows the explanation for website piriform.com that was evaluated with a moderate rating (5) and predicatively rated with 5.2. In contrast with the previous example, we can observe that this website had more attributes with positive impact, and that there are just a few attributes with negative impact. Attributes with largest positive impacts are "html_errors", "js_sources", "ids", "html5_tags", "headings", "links", "headings", "font_sizes" and "text". There are also attributes that had negative impact: "rss", "meta_tags", "font_families", "divs" and "classes". The findings are again consistent with the computed explanation of the entire model.

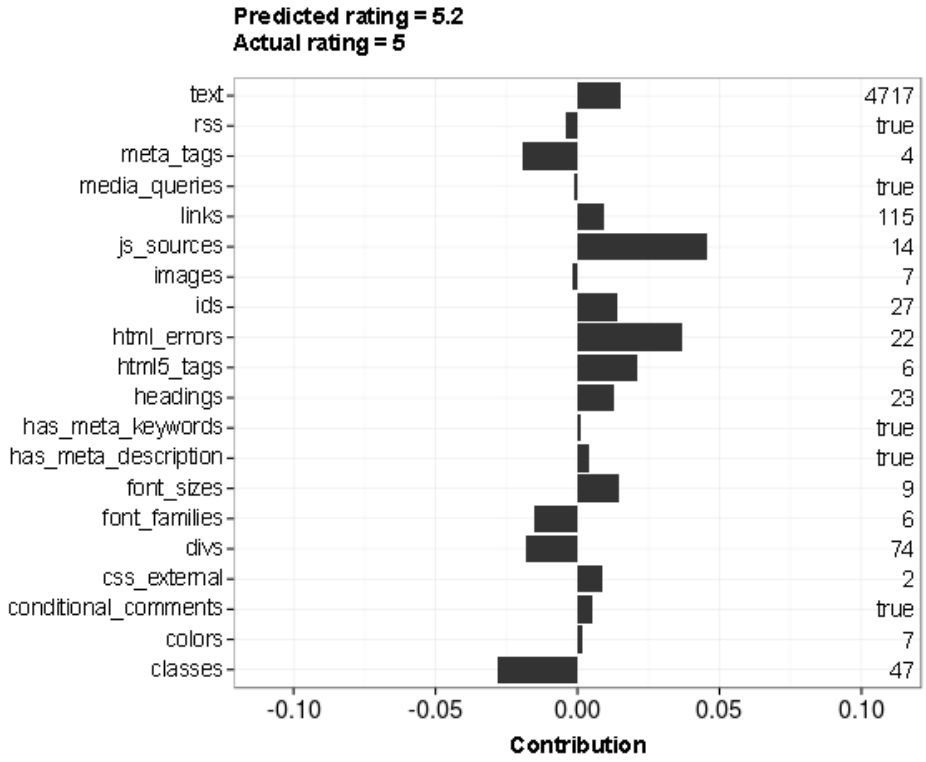

Figure 4.5: Instance explanation for piriform.com

Figure 4.6 shows the explanation for the third website, getpocket.com, that was evaluated with a high rating (8.8) and predictively rated with 6.3. We can see that similar attributes as in our previous example (Figure 4.5) had positive and negative impacts on this website, as well. Most of the positive impact was contributed by values of attributes “links”, “js_sources”, “html_errors”, “font_sizes”, “html5_tags”, “headings”, “ids”, “text”, “conditional_comments” and “images”.

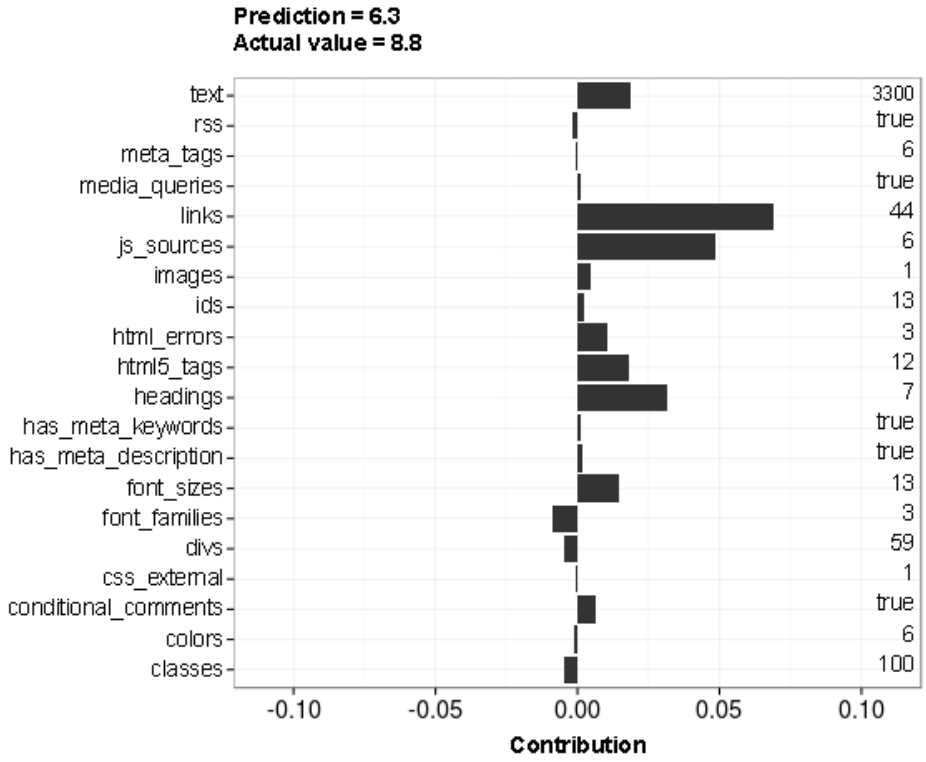

Figure 4.6: Instance explanation for getpocket.com

Chapter 5

Browser extension

In this chapter, we describe the development and idea of the final product - a Google Chrome extension that can help developers during website production. We named the Chrome extension "Flasknose", a wordplay on "Bottlenose dolphins" (flask is a type of bottle), which are among the most intelligent animals.

5.1 The architecture of the developed extension

The extension itself is just a client-side application for the end user. Nevertheless, there is a lot of processing that happens on the server-side, which we describe in the following paragraphs. We plan to use our predictive model, which we evaluated and explained in the previous chapters, as the core of the extension, along with the data about model explanation.

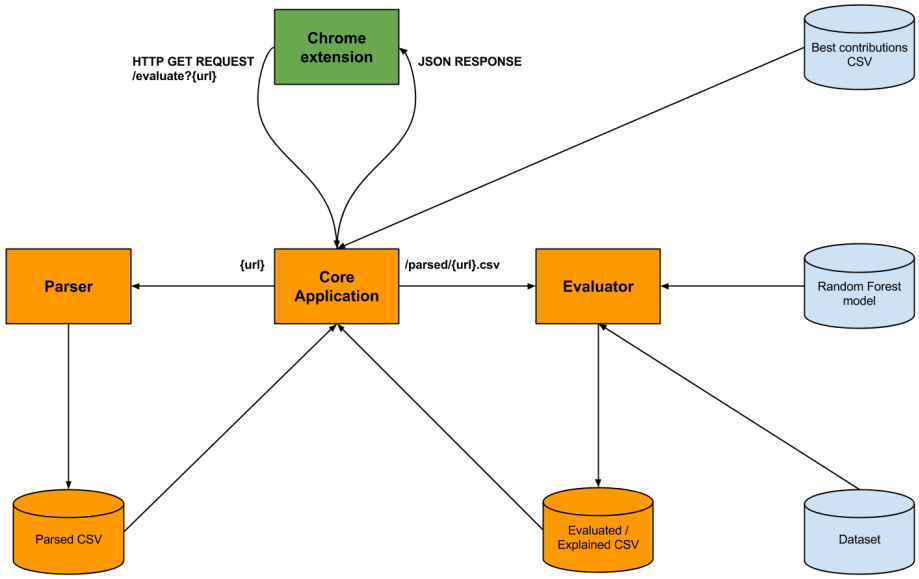

Figure 5.1: Scheme of a browser extension architecture. Green - client-side, Orange - server-side, Blue - application data

Therefore, in Figure 5.1 we can see the scheme of the architecture that can be split into five most important parts that are described below, where we separately describe client-side Chrome extension, core application, parser, evaluator and application data.

5.1.1 Chrome extension (client-side)

We developed the extension using client-side techniques based on HTML, CSS and JavaScript. The Chrome extension also contains the manifest file [37] which is just a JSON formatted table of contents that stores the general data about Chrome extension like its name, description, version number and so on. The main task of the Chrome extension is to communicate with the server-side application, by sending request to it and displaying the data to the user when the response is received. In Section 5.2, we will tell more about how to use the extension.

5.1.2 The core (server-side) application

The core of the application is built using a PHP micro-framework named Silex, which is based on the Symfony[1] components. We chose that framework because it is lightweight, good for bootstrapping and good for handling simple requests and responses.

This core layer aggregates all the other layers - it receives a request from the Chrome extension, calls a parser (described in Section 5.1.3), calls an evaluator (described in Section 5.1.3), brings all the data together and then returns a JSON-formatted response back to the Chrome extension that shows the results to the end-user. The JSON response is split into three sections - "website", "attributes" and "contributions". The section "website" (shown in Figure 5.2) consists of data parsed from the processed website, together with the received rating of the page. The section "attributes" (partially shown in Figure 5.3) contains a list of evaluated attributes, their descriptions and their best interval values. The section "contributions" (shown in Figure 5.4) shows whether the attribute values of the processed website had positive or negative impact after the received evaluation.

[1]Symfony - PHP framework for web development

```
{
    "website": {
        "url": "www.moonleerecords.com",
        "css_external": "5",
        "font_sizes": "11",
        "font_families": "7",
        "colors": "9",
        "meta_tags": "7",
        "classes": "45",
        "js_sources": "13",
        "links": "94",
        "images": "38",
        "media_queries": "true",
        "ids": "54",
        "divs": "58",
        "headings": "22",
        "text": "7846",
        "html5_tags": "1",
        "has_meta_keywords": "true",
        "conditional_comments": "false",
        "html_errors": "104",
        "has_meta_description": "true",
        "rss": "true",
        "rating": 3.74
    }
}
```


Figure 5.2: Example of JSON response from the server - section “website”

```
{
    "attributes": {
        "css_external": {
            "description": "Number of external CSS sources",
            "contribution": "0.00594103918563813",
            "bestValue": "11 - 14"
        },
        "font_sizes": {
            "description": "Number of font sizes",
            "contribution": "0.00969506142485781",
            "bestValue": "10 - 14"
        },
            "font_families": {
            "description": "Number of all font families",
            "contribution": "0.00589039084806754",
            "bestValue": "12 - 16"
        },
        "colors": {
            "description": "Number of colors",
            "contribution": "0.00819496872903376",
            "bestValue": "1 - 6"
        },
            "meta_tags": {
            "description": "Number of meta tags in the <head>",
            "contribution": "0.00544882537413989",
            "bestValue": "1 - 8"
        },
        ...
    }
}
```


Figure 5.3: Example of JSON response from the server - section “attributes”

```
{
    "contributions": {
        "text": "0.00225479680546379",
        "headings": "0.0147606988153462",
        "images": "-0.0103420621839931",
        "font_families": "-0.00660889898901358",
        "font_sizes": "0.0126723735638628",
        "links": "0.0278543635944567",
        "divs": "-0.00973004462042928",
        "ids": "-0.00513147026724093",
        "classes": "-0.0311881519105302",
        "css_external": "-0.000537304889333406",
        "js_sources": "0.0199532526576352",
        "meta_tags": "-0.0047761200785613",
        "has_meta_keywords": "0.00178919852172994",
        "has_meta_description": "0.00480294578648505",
        "rss": "-0.00307787837699083",
        "html5_tags": "-0.0218091798419434",
        "media_queries": "-1.76168612516692e-05",
        "conditional_comments": "-0.00728170671635332",
        "html_errors": "-0.00228542459789248"
    }
}
```


Figure 5.4: Example of JSON response from the server - section "contributions"

5.1.3 Parser

Parsing is performed on the server-side and is the first process that is called after the request is received from the Chrome extension (the parser is already described in Chapter 3). The parser crawls the given website, extracts the required attributes and stores the data into a CSV file (a partial example is shown in Table 5.1) that contains attribute names in the header and attribute

text	**html_elements**	**headings**	**paragraphs**	**images**	**...**
2703	208	3	2	18	...

Table 5.1: A partial example of the CSV file that contains parsed data

attribute	**contribution**
text	0.0339801668394493
headings	0.0166580385400552
images	-0.00707481809301546
font_families	-0.0069773719937119
...	...

Table 5.2: Partial example of CSV file that contains contributions

values in the first row. After the parser finishes, the CSV file is forwarded to the Evaluator (described in the Section 5.1.4).

5.1.4 Evaluator

The evaluator is written in R and is called after the parser finishes. The Evaluator receives a parsed example (CSV file), reads the data, predicts the rating of a given example and calculates its explanation. For predictions, we use the Random forests model and the explanation methodology [25] that was described in Chapter 4. After all the processing is done, the result is output into two CSV files - one for contributions (shown partially in Table 5.2), and one for the rating (shown in Table 5.3). The CSV files are then gathered by the core application that brings all the data together and generates a proper JSON-formatted response.

5.1.5 Application data

The Application contains the data that has been generated during the whole process of analysis. This data is written in CSV and RDS (single R object)

url	rating
http://www.fri.uni-lj.si/	0.489791615041364

Table 5.3: Example of CSV file that contains rating

attribute	contribution	interval	contributionType
classes	0.00702380699060079	339 - 438	pos
colors	0.00819496872903376	1 - 6	pos
css_external	0.00594103918563813	11 - 14	pos
media_queries	0.00239990444929369	true	pos
...	...	...	...

Table 5.4: Example of CSV file that contains the best contributions

file formats, and is read and used by the core application and by the R evaluator. Here, we are referring to the generated Random Forest model, original dataset and a list of best contributions (partially shown in Table 5.4).

5.2 How to use the Chrome Extension

Figure 5.5 displays how the developed extension works: it consists of a popup window that contains only the "Evaluate" button. After clicking on the button, a new window (shown in Figure 5.6) opens and a GET Request is sent to the server. The GET Request contains the URL of the website, that is going to be processed. During the processing period, a spinner indicator is shown on the results page and the favicon[2] also changes its background to red - so that the user is notified that the page is being processed on the server.

After the server-side processing is completed, the server returns a JSON formatted Response to the extension which includes the explanation of indi-

[2]Favicon (Favorite icon) - website icon shown in the address bar and next to the page name

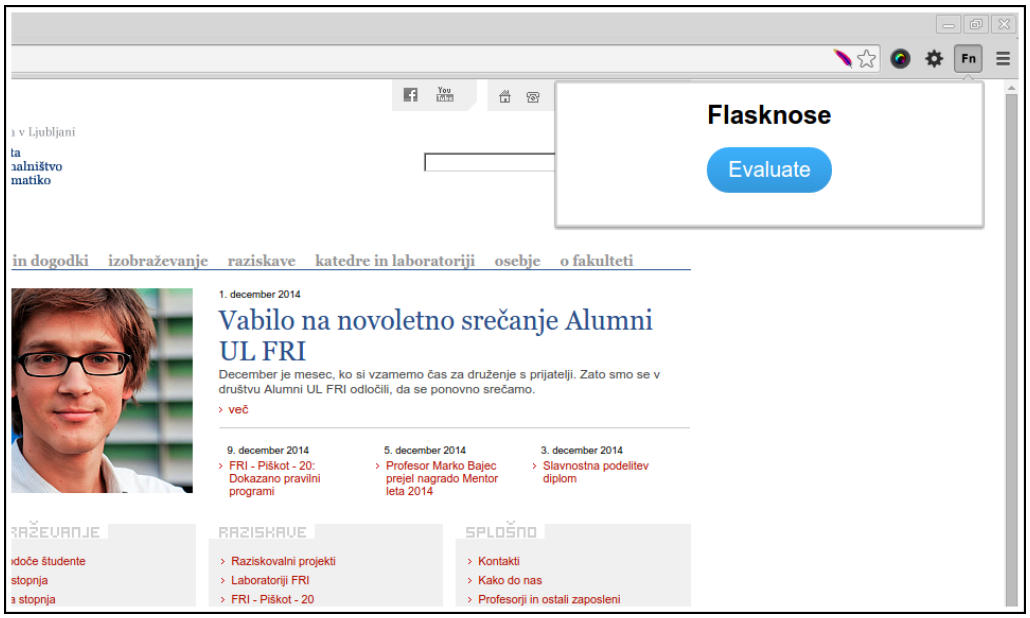

Figure 5.5: Popup window of the Chrome extension

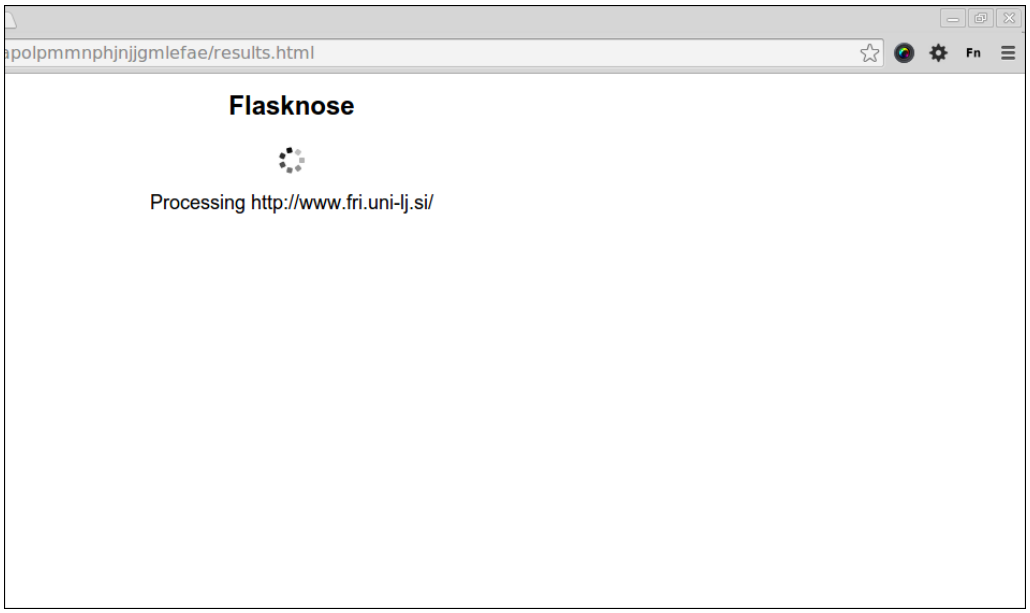

Figure 5.6: Notification of the Chrome extension that the webpage is being processed on the server

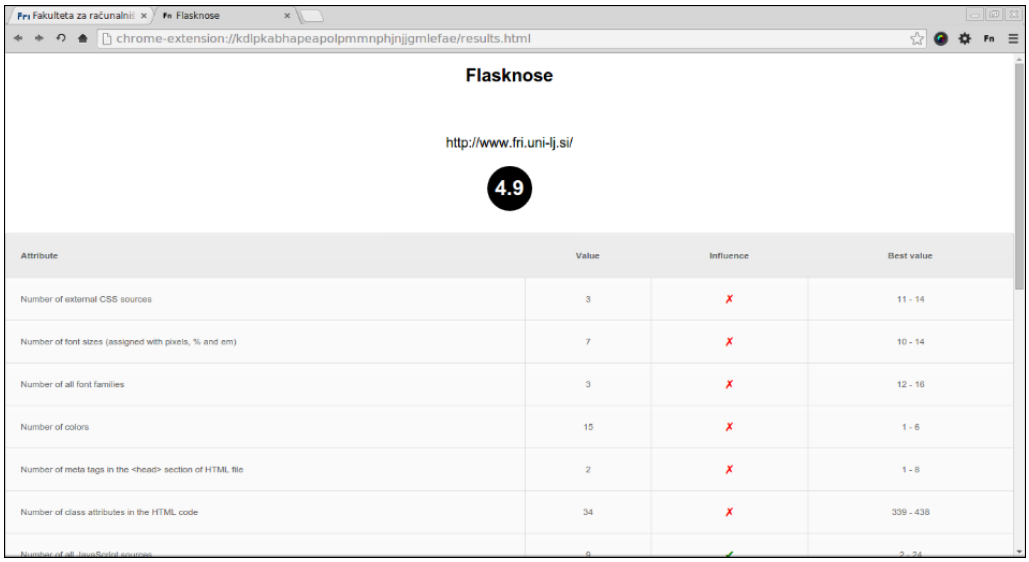

Figure 5.7: Chrome extension - the page with evaluation results

vidual attributes and the overall rating of the page. All these data are shown in the table, as displayed in Figure 5.7.

Chapter 6

Conclusion

We collected a list of websites and built a dataset by using the crowd-sourcing approach. To evaluate and select the best attributes from a list of all defined attributes we have used the ReliefF algorithm. With twenty selected attributes, we have built more different models with different machine learning methods. Then, we have evaluated (with 10-fold cross validation) the received models and took the best one – which was the one built with the Random Forests (RF) algorithm. With the explanation methodology (Štrumbelj and Kononenko, 2010) we explained the best model by using three different instances from the dataset. The idea was to show how individual attributes and attribute values contribute to the whole model and to the individual instance – where we could see whether the analyzed attribute values had a positive or negative impact on the model and the given instances. The best model was afterward used as the basis for our Chrome extension. Another part of the extension is also the explanation methodology, which acts at the same time as the predictor and finds out attribute values contributions for given instances.

Three main contributions, already defined at the beginning, were covered in this project and were already mentioned in the description above. These contributions are:

1. a systematic overview of meaningful attributes for describing and mea-

suring web design,

2. explanation and interpretation of the best evaluated model,

3. a Google Chrome extension that will help developers during website production.

The source code of the Google Chrome extension, together with all the server-side implementation, is available at [38].

One of the things we have been aware of since the very beginning is that our work might become irrelevant already after a short time period. The reason for that lies in the rapid and constant change of trends in web design. Although interesting facts would remain, these would not be completely suitable anymore.

Because of that, we have to deal with a few tasks that we are aware can be continuously improved. The first thing that we have to consider is that the Chrome extension is still in the alpha development version and not production-ready. In the following paragraphs, we will describe some of the present issues and suggest improvements that could bring the entire project to a higher level.

One of the important improvements we have been considering is to take into an account both mobile and desktop website versions and to consider a different analysis approach for each. We were analyzing and considering only desktop websites and did not focus on mobile applications, despite the fact that mobile web is at the moment more interesting than ever. We would need to apply different rules for mobile websites and probably come out with completely different attributes and results.

In relation to the previous point, we would also need to adapt our voting system, so that mobile and desktop websites have separated votes. Another thing that could be improved in the voting system is to make it more fun. The current voting system could be transformed to a game in order to add new examples and to collect new data constantly. Because of that, we would

also need to implement adaptive learning, and build machine learning models that would consider the flow of examples as a data stream.

Adaptive machine learning is also something that would definitely need to be implemented if we frequently introduce new attributes. We believe that current attributes are not the best attributes and that we should constantly search for new and better attributes. This would again require adaptive machine learning.

In the current stage of the application, we are using CSV files to store processed data. We decided for that because we had few different environments and it was more simpler to deal with CSV files in that stage of development. To improve that in the next stage, we would do it with relational database like MySQL, PostgreSQL or even with NoSQL database like MongoDB - whichever we would decide that it is better to use.

A big challenge was to write the perfect parser and to pick perfect tools to build it. Because we wanted a parser that would work automatically (without any adjustments), it was just impossible to get all the data from all the sources – and not just because it is hard to detect all the dynamical changes – in some cases, not all files were directly accessible or readable. To improve it, we would need to try more different tools and test which ones give us the best results, which are the most adaptive and so on.

We learned a lot during this work, we were able to use the knowledge we have learned at the Faculty of Computer and Information Science and we believe that the system we developed is a step forward for the field of website evaluation, analysis and performance.

Bibliography

[1] J. Zeldman, E. Marcotte, "Designing with Web Standards", New Riders, 2009.

[2] J. Nielsen, M. Tahir, "Homepage Usability: 50 Websites Deconstructed", New Riders, 2001.

[3] E. Estellés-Arolas, F. González-Ladrón-de-Guevara, "Towards an integrated crowdsourcing definition", Journal of Information Science, 2012.

[4] R. Kumar, A. Satyanarayan, C. Torres, M. Lim, S. Ahmad, S. R. Klemmer, J. O. Talton, "Webzeitgeist: Design Mining the Web", Stanford University, Massachusetts Institute of Technology, Intel Corporation, 2013.

[5] A. Satyanarayan, M. Lim, S. R. Klemmer, "A Platform for Large-Scale Machine Learning on Web Design", Stanford University, 2012.

[6] M. Lim, A. Satyanarayan, C. Torres, "Structural Learning for Web Design", Stanford University, 2012.

[7] "Web Authoring Statistics" by Google, Last accessed 24.12.2014, available at: https://developers.google.com/webmasters/state-of-the-web/

[8] "Validating sites of W3C members", Last accessed 24.12.2014, available at:
http://www.triin.net/2006/03/05/Validating_sites_of_W3C_members

[9] "Semantics in the wild", Last accessed 24.12.2014, available at: http://westciv.typepad.com/dog_or_higher/2005/11/real_world_sema.html

[10] M. Lim, R. Kumar, A. Satyanarayan, C. Torres, J. O. Talton, S. R. Klemmer, "Learning Structural Semantics for the Web", Stanford University, Intel Corporation, 2012.

[11] M. Perkowitz, O. Etzioni, "Adaptive Web Sites: an AI Challenge", University of Washington, 1997.

[12] Smashing Magazine, "The Smashing Book #4", Smashing Magazine GmbH, 2013

[13] "The global structure of an HTML document", Last accessed 24.12.2014, available at: http://www.w3.org/TR/html401/struct/global.html#h-7.5.5

[14] S. Weinschenk, "100 Things Every Designer Needs to Know About People (Voices That Matter)", New Riders, 2011.

[15] W. Barber, A. Badre, "Culturability: The merging of culture and usability", Georgia Institute of Technology, 1998.

[16] T. Berners-Lee, D. Connolly, "Hypertext Markup Language (HTML): A Representation of Textual Information and MetaInformation for Retrieval and Interchange", IIIR Working Group, 1993.

[17] B. Lawson, R. Sharp, "Introducing HTML5", New Riders, second edition, 2011.

[18] Z. Çelikbaş, "What is RSS and how can it serve libraries?", First International Conference on Innovations in Learning for the Future: e-Learning, 2004.

[19] List of most popular repositories at Github, Last accessed 24.11.2014, available at: https://github.com/search?q=stars

[20] E. Weyl, "What's New in CSS3", O'Reilly, 2012.

[21] "Consistently render HTML elements in all browsers - CSS Reset and Normalize", Last accessed 24.11.2014, available at: http://www.refulz.com/consistently-render-html-elements-in-all-browsers-css-reset-and-normalize/

[22] A. Watson, "Learning Modernizr", Packt Publishing, 2012.

[23] "Does Alexa have a list of its top-ranked websites?", Last accessed 24.12.2014, available at: https://alexa.zendesk.com/hc/en-us/articles/200449834-Does-Alexa-have-a-list-of-its-top-ranked-websites

[24] R. Friesel, "PhantomJS Cookbook", Packt Publishing, 2014.

[25] E. Štrumbelj, I. Kononenko, "A General Method for Visualizing and Explaining Black-Box Regression Models", University of Ljubljana, 2011.

[26] I. Kononenko, E. Simec, M. Robnik-Šikonja, "Overcoming the myopia of inductive learning algorithms with RELIEFF", University of Ljubljana, 1997.

[27] M. Robnik-Šikonja, I. Kononenko, "An adaptation of Relief for attribute estimation in regression", University of Ljubljana, 1997.

[28] M. Robnik-Šikonja, P. Savicky, "Classification, regression, feature evaluation and ordinal evaluation", version 0.9.43, 2014.

[29] T. Therneau, B. Atkinson, B. Ripley, "Recursive Partitioning and Regression Trees", version 4.1-8, 2014.

[30] "Fitting Linear Models", Last accessed 24.11.2014, available at: https://stat.ethz.ch/R-manual/R-devel/library/stats/html/lm.html

[31] B. Ripley, W. Venables, "Feed-forward Neural Networks and Multinomial Log-Linear Models", version 7.3-8, 2014.

[32] D. Mayer, E. Dimitriadou, K. Hornik, A. Weingessel, F. Leisch, C. C. Chang, C. C. Lin, "Misc Functions of the Department of Statistics (e1071)", version 1.6-4, 2014.

[33] L. Breiman, A. Cutler, A. Liaw, M. Wiener, "Breiman and Cutler's random forests for classification and regression", version 4.6-10, 2014.

[34] K. Schliep, K. Hechenbichler, "Weighted k-Nearest Neighbors", version 1.2-5, 2014.

[35] I. Kononenko, M. Kukar, "Machine Learning and Data Mining: Introduction to Principles and Algorithms", Horwood Publishing Limited, 2007.

[36] P. Refaeilzadeh, L. Tang, H. Liu, "Cross-Validation", Arizona State University, 2008.

[37] "Getting Started: Building a Chrome Extension",
Last accessed 24.11.2014, available at:
https://developer.chrome.com/extensions/getstarted

[38] Github repository of the Chrome extension, Last accessed 04.01.2014, available at: https://github.com/peric/flasknose