UNIVERSITY OF TARTU

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

INSTITUTE OF COMPUTER SCIENCE

Nataliia Semenenko

Accurate diagnosis of cross-browser compatibility issues via machine learning

Master thesis (30 ECTS)

Supervisor(s): Marlon Dumas, University of Tartu

Tõnis Saar, STACC

Author: _____ "___" May 2013

Supervisor: _____ "___" May 2013

Approved for defense

Professor: _____ "___" May 2013

Tartu, 2013

Acknowledgement

First of all, I would like to express my deepest gratitude to my supervisor Marlon Dumas for his encouragement and support throughout the research. He guided me from the initial to final stage of the study enabling me to develop a better understanding of the subject. He inspired me to extend the research and to go deeper into it. I acknowledge his great contribution into the research behind this Master thesis.

I am thankful to my co-supervisor, Tõnis Saar, for his help, guideline and valuable comments. He has started working with me from the very beginning, explaining me the refinements of the subject, details of the techniques and background of our research. He was also helpful with the abstract translation into Estonian language.

I would like thank STACC and Browserbite that employed me and without which the research would not be possible.

Abstract

Due to the rapid evolution of Web technologies and the failure of Web standards to uniformize every single technology evolution, Web developers are faced with the challenge of ensuring that their applications are correctly rendered across a broad range of browsers and platforms.

While abidance to Web standards may reduce the chances of Web documents being inconsistently rendered across multiple browsers, in practice cross-browser compatibility issues are recurrent and range from minor layout bugs to critical functional failures such as a button being invisible in a given browser-platform combination.

To detect cross-browser incompatibilities, developers often resort to visually checking that each document produced by their application is consistently rendered across all relevant browser-platform combinations. This manual testing approach is time consuming and error-prone. Existing cross-browser compatibility testing tools speed up this process by automating the rendering of a Web document in multiple browsers and platforms, and applying either image analysis or Document Object Model (DOM) analysis to highlight potential cross-browser incompatibilities.

However, existing tools in this space suffer from over-sensitivity, meaning that they produce a large number of false positives as they tend to classify even insignificant differences as potential incompatibilities. Reducing the number of false positives produced by cross-browser compatibility testing tools is challenging, since defining criteria for classifying a difference as an incompatibility is to some extent subjective.

This Master's thesis presents a machine learning approach to improve the accuracy of two techniques for cross-browser compatibility testing – one based on image analysis (Browserbite) and one based on DOM analysis (Mogotest). To this end, we selected over 140 Web pages, each rendered in 10 to 14 browser-system combinations and built statistical classifiers to differentiate between true incompatibilities and false alarms. Two classification algorithms were used, namely classification trees and neural networks. An extensive experimental evaluation shows that neural networks produce highly accurate classifiers, both when post-processing the outputs of the image-based and the DOM-based technique. An attempt to combine image and DOM-based analysis is also reported.

Table of Contents

List of abbreviations

DOM	Document Object Model
IUT	Image under the test
ROI	Region of interest
ROIT	Region of interest on the browser under the test
ROIB	Region of interest on the baseline browser
ROC	Receiver operating characteristic
AUC	Area under the curve

1 Introduction

Web applications are increasingly widespread nowadays, and are used daily both for work and personal activities, such as shopping, networking, banking, entertainment etc., by people all over the world. Currently users can choose between a wide range of Web browsers and this fact challenges Web developers – Web applications are expected to behave consistently across all browsers and platforms that the intended user might have chosen.

The earliest instances of cross-browser compatibility problem have roots in the parallel evolution of different browsers and lack of standardization of Web content. Currently there exist standards for basic Web content (e.g. HTML 4.0) and scripting languages such as JavaScript, Jscript etc. However, mostly all Web browsers typically implement their own extensions to the Web standards which play a significant role in their behavior. These inconsistencies lead to cross-browser incompatibilities – differences in the way a Web page looks and behaves in different browsers.

Cross-browser incompatibilities range from minor and even imperceptible cosmetic problems to critical failures that constitute layout, formatting and behavioral incompatibilities. Such incompatibilities can result in the inability of the user to access one or more services provided by a Web application. Thus maintaining compliance across different browsers and platforms is a crucial task for Web application developers.

Manual detection of these incompatibilities by means of visual inspection of Web pages rendered in multiple platforms is highly labor-intensive and even error-prone, as Web testers tend to miss some incompatibilities due to fatigue.

Automated cross-browser testing tools significantly reduce the amount of required manual effort. Currently there are several solutions and services both open-source (such as CrossCheck [27], WebDiff [5] etc.) and commercial (i.e. Browserbite [28], Mogotest [7], Browsera [34] etc.), that claim to address the problem of cross-browser compatibility checking. However, these systems suffer from over-sensitivity and produce an excessive amount of false positives. For example, Choudhary et al in [27] had run an experiment with two cross-browser compatibility testing solutions CrossCheck and WebDiff. According to the experiment, these solutions were exercised on same Web applications; manually confirmed differences between rendered Web applications were compared to the results provided by tools. An evaluation has shown 64% of false positives for CrossCheck and 79% false positives in WebDiff. Fundamentally, this is due to the fact that what constitutes an incompatibility, as opposed to a simple difference, is to some extent subjective [27]. Thus setting a specific threshold to classify a difference as an incompatibility is not trivial.

In this setting, this Master thesis addresses the problem of providing accurate diagnosis of cross-browser incompatibilities. The goal in this context is to decrease the amount of false positive results in the outcomes of cross-browser compatibility testing tools. The working hypothesis is that, rather than setting arbitrary thresholds in order to classify a difference as an incompatibility or a false positive, more accurate diagnosis can be obtained by starting from an over-sensitive solution and applying supervised machine learning methods to construct models that classify potential incompatibilities into actual incompatibilities and false positives.

The contribution of the research is a machine learning technique based on single-page cross-browser compatibility testing approaches (specifically that of Browserbite and Mogotest) in order to improve their accuracy. Both tools – Browserbite and Mogotest – are good examples of single-page cross-browser compatibility testing technique. In addition, these tools are representatives of two different methodologies for Web pages comparison. While Mogotest is based on Document Object

Model (DOM), Browserbite uses image-based segmentation and comparison technique, but both suffer from false positives in their results. In this context we empirically evaluate the classification and ranked accuracy of classification trees and neural networks based on DOM parameters of elements on the Web page and features produced during automated image comparison. The training and testing of machine learning models is based on 140 Web pages rendered across more than 10 different configurations (browser-platform combinations).

The rest of the document is structured as following. Section 2 presents principles and limitations of existing cross-browser compatibility testing techniques. Next, in Section 3 we discuss the problem of over-sensitivity and its consequences for single-page compatibility testing techniques on example of Browserbite and Mogotest. In Section 4 we introduce the machine learning approach to post-process the outputs of image-based and DOM-based cross-browser compatibility testing techniques, while in Section 5 we are combining features of these two techniques with the purpose to examine the combination's influence on the accuracy. Section 6 concludes and discusses the direction for future work.

2 State of the art

For the Web application to be successful it is important to work correctly on different configurations (combination of browsers and platforms). A Web application is called cross-browser compatible if it is rendered identically and works correctly on these different configurations. However, in practice maintaining cross-browser compatibility is a challenging task. In total there are five major browser vendors [17], each suggests more than one browser available version and also some browsers are provided for different operating systems. The distribution of market shares that is presented in Table 1 shows that for many browsers several of its versions are actively used.

Table 1. Market shares of the major browsers for the first quarter 2013 [17]

Browser version	Market share [%]	Total [%]
Microsoft Internet Explorer 8.0	23.47	23.47
Microsoft Internet Explorer 9.0	21.28	44.75
Chrome 24.0	9.72	54.47
Firefox 18	9.57	64.04
Microsoft Internet Explorer 6.0	6.53	70.57
Chrome 23.0	3.68	74.25
Firefox 17	3.21	77.46
Safari 6.0	2.85	80.31
Microsoft Internet Explorer 7.0	2.07	82.38
Firefox 19	1.71	84.09
Safari 5.1	1.65	85.74
Chrome 25.0	1.56	87.3
Opera 12.x	1.51	88.81
Microsoft Internet Explorer 10.0	1.43	90.24
Firefox 16	0.81	91.05
Firefox 12	0.68	91.73
Firefox 3.6	0.65	92.38
Safari 5.0	0.61	92.99
Firefox 14	0.49	93.48
Firefox 15	0.48	93.96
Chrome 21.0	0.38	94.34
Firefox 10	0.38	94.72
Firefox 13	0.34	95.06

As a consequence, depending on popularity of the Web page, the number of required configurations can go up to 20-30 in order to cover 90-95% of users. In other words, developers of Web application have to test their product on 20-30 different browser-system combinations. There is no way to avoid such testing due to the fact that different browsers and also different version of the same browser can behave differently despite that there are existing standards for the most important technologies. For quality assurance this has serious consequences. For instance, with the aim of testing of a small Web application with just 10 pages, a Web testers would have to manually com-

pare 230 pages rendered in 23 most popular browsers for the first quarter 2013 (see Table 1), and to make this testing more comprehensive these browser should be run on different operating systems, so that the number of test pages would be increased. Moreover, for avoiding errors, these tests would have to be conducted after each change made in the Web application.

In practice, developers of Web application normally use a fixed browser to run tests while implementing the application. Hence, this fixed browser is called reference browser, oracle or baseline browser, and for this browser the application is expected to work correctly.

A number of automated cross-browser testing tools significantly reduce the amount of required manual effort. There are many tools available to help check cross-browser compatibility issues. Currently there are several tools and services both open-source and commercial, that claim to address the problem of cross-browser compatibility checking.

Among them there are services that provide screenshots of how particular URL, specified by the user, would be rendered under various client-side platform combinations of specific browsers, operating systems and screen resolutions. Amid them there are IE NetRenderer [13], ieCapture [23], BrowserShots [32] and Browser Photo [16]. Such tools do not offer the automatically comparison service and thus Web testers still have a big branch of work to do

There also exist tools that provide emulation environments which allow the user to interact with particular Web application as if it is being served on particular configuration. Representatives in this category include CrossBrowserTesting.com, which span a wide variety of client-side platforms. In addition there are more specific services like BrowserCamp [29] targeted for Mac OS browsers and Xenocode Browser Sandbox [1] for Windows-based environments.

However, none of the above tools and services checks automatically the cross-browser compatibility of the particular Web application. It is left to the human user, which makes the whole process very ad hoc and hence the quality of the check itself very suspect.

More comprehensive cross-browser compatibility testing techniques are automated and can be broadly divided into two main categories: testing techniques based on the Web application behavior and single-page testing techniques. In Section 2.2 we describe behavioral testing techniques which focus on analyzing the possible states and transitions of Web applications, with the aim of detecting differences of state transitions of the Web page (among them there are sequences of pages that can be reached via Web application under different platforms), while in Section 2.1 we present single-page testing techniques that consider those differences that could be captured when analyzing the single Web page appearance.

2.1 Single-page cross-browser compatibility testing techniques

Single-page cross-browser testing techniques can be divided into those based on Document Object Model (DOM) techniques and those based on image comparison. In general the single-page cross-browser testing techniques can be described in two phases: (i) segmentation of the Web page and (ii) matching and comparison of page segments. Each of these two phases relies on DOM-based analysis or image processing.

2.1.1 DOM-based techniques

Document Object Model is a platform independent and language-neutral convention for representing and interacting with objects in HTML, XHTML and XML documents [2]. To render a document such as HTML page, most browsers use an internal model similar to the DOM. The

nodes of each document are organized into a tree-structure called DOM tree, which topmost node is named ‘Document object’. When a Web page is rendered in browser, it downloads the HTML into the local memory and automatically parses it to display the page on the screen. The example of the DOM tree structure is presented on the Figure 1.

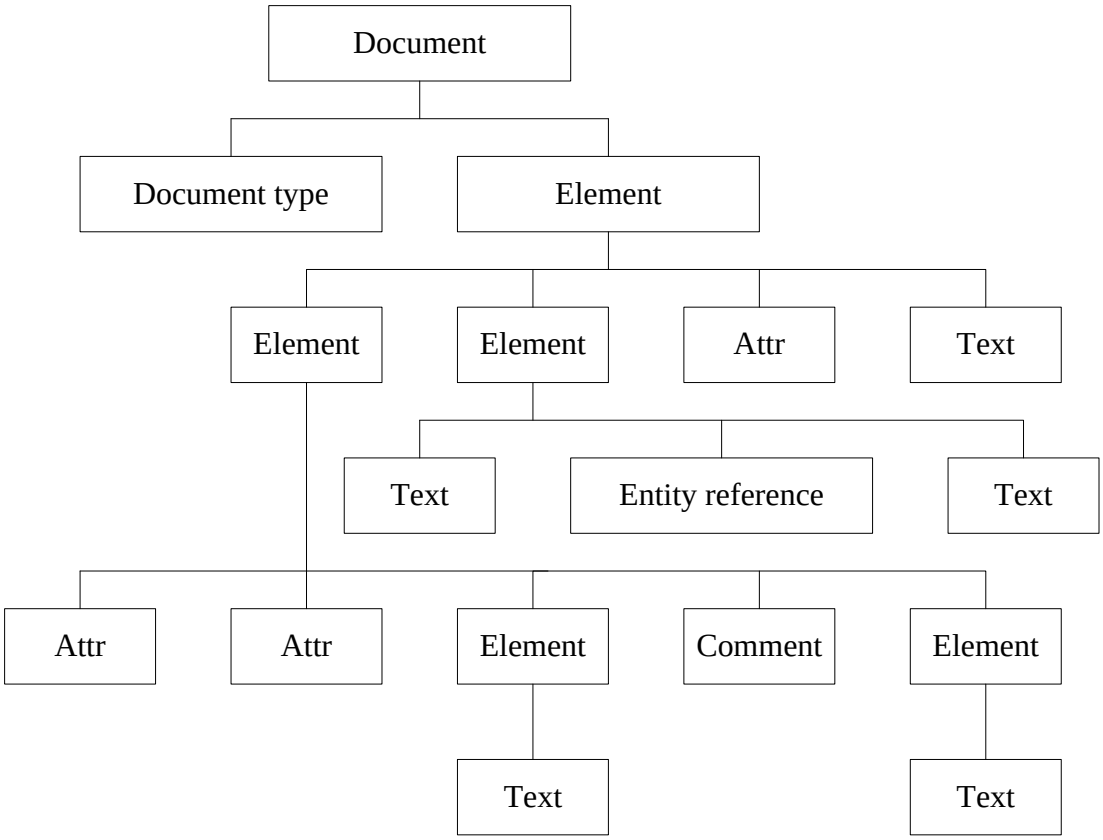

Figure 1. DOM structure representing XML [19]

State-of-the-art DOM-based technique is proposed by Choudhary et al. [5]. This method is implemented in the tool called WebDiff, the overall goal of which is to find the dissimilar Web page elements across different browsers. For this reason Choudhary et al. use the baseline browser and compare data about rendered Web page on all other browsers against it. This approach works in four phases: (i) DOM data and screenshot collection, (ii) variable elements identification, (iii) DOM elements matching and (iv) matched information comparison and differences identification.

During the first phase WebDiff launches the Web page under test in all considered browsers and adjust the browsers’ window size such as they are equal. After this, tool extracts the DOM information (specifically, name of the DOM element tag, unique identifier of the DOM node, node’s X-path in the structure, absolute screen position, visibility and clickability parameter, and DOM element’s screen stack order) of the Web page and captures the screenshot for each browser window.

In the DOM-based techniques elements that change when the page is reloaded (variable elements, such as advertisements and page statistics) may results in false positives, thus they should be ignored during the analysis. During the second step WebDiff detects variable elements by loading the Web page twice in the baseline browser and compares the DOM and screenshot information obtained in two cases. All DOM nodes that reveal either structural or visual differences are marked and ‘variable’ and ignored in the further steps.

The cross-browser comparison phase in the technique suggested by Choudhary et al. is divided into two steps: structural analysis and visual analysis. The structural analysis aims to match the DOM nodes obtained from different browsers. For instance, two nodes with the same id and/or X-path are marked as perfect match. For each pair of nodes a match index is calculated, which is the number between zero and one that represents how similar are two nodes. Although there is an official definition of the DOM format [2], Web browsers often deviate from such standard. Moreover, since most Web pages have browser specific code to make them work on different browsers and platforms, the DOM generated by different browsers can be different. Therefore WebDiff also performs the visual analysis of the screenshots of Web page rendered on different browsers. In this step, WebDiff uses the structural information to identify corresponding elements in the browser screenshot and perform graphical matching of these elements. This is done using the histogram-based analysis [30]. WebDiff checks for four classes of issues: positional, size, visibility and appearance differences. If any difference is found during these two steps of cross-browser comparison analysis it is reported to the user.

Although using DOM-based techniques to detect differences allows reporting the specific HTML tags that are responsible for an issue, which can help developers to understand and eliminate the problem, these techniques have some limitations. A fundamental limitation of DOM-based methods is that the same HTML code can be interpreted very differently on different browsers. In other words, the DOM of the same Web page for two platforms can have very different properties. As a result, segments extracted using DOM analysis may have different sizes (granularities) depending on the browser, which makes the comparison browser-specific, meaning that both the extraction and matching of segments needs to be tuned on a browser-per-browser basis and possibly re-adjusted as browsers evolve. The technique suggested by Choudhary et al. tries to avoid this limitation by using DOM-based analysis for segmentation and comparison and also image processing for comparison. However, this approach also possesses some limitations. Among them there is a disability to process embedded objects, such as Adobe Flash elements, and WebDiff ignores their internal details. Also the detection of variable elements is done by reloading Web page twice in a small period of time, which means that tool misses certain variable elements and thus causes more false positives. As the image comparison is based on the DOM segmentation and as it was mentioned earlier the same extracted segments may have different granularities depending on the browser, this may also cause the false positive results to appear.

There exists several commercial DOM-based cross-browser compatibility testing tools. Among them there are Mogotest [7] and Browsera [34]. On the help page of the later the main issue of using DOM-based testing techniques is also mentioned or, rather, stated that the tool is not able to properly process Web pages the DOM structure of which differs. These differences in the DOM tree can be caused not only by different interpretation of the same HTML code by multiple browsers, but also, as it is stated by Browsera's developers, in most cases this is a result of either improperly closed or illegally nested tags. The recommendation is to keep the structural HTML as similar as possible for each browser and ensure all tags are closed and nested properly, which is another branch of manual work for human testers.

2.1.2 Image-based techniques

Image-based techniques take as input screenshots of a given Web page on the baseline browser and compare them to screenshots on a number of other browsers (or platforms) under test.

As the segmentation and comparison is performed on the basis of images, these techniques are largely browser-independent as only the screenshot capturing is browser-specific.

State-of-the-art image-based techniques include Browserbite [28]. The Browserbite's cross-browser testing method is built upon image processing techniques, specifically image segmentation and image comparison. The overall workflow of Browserbite is depicted on the Figure 2. During the first step the static image of the tested Web page is generated from each browser-platform combination. One of these images is considered to be the baseline image, while others are called images under the test (IUT). The user of the tool is free to select the baseline, but every time before testing the user has to check whether the baseline image is rendered correctly as all other images will be compared against it.

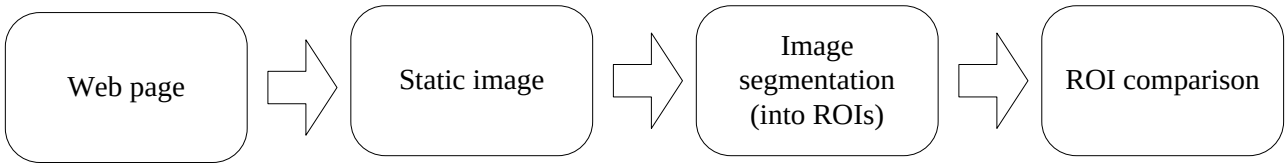

Figure 2. Principle layout of visual testing method

In computer science, image segmentation is a process of partitioning an image into multiple segments. The goal of this segmentation is to simplify the analysis of an image. In case of Browserbite this is done for the comparison purposes. In Browserbite the image segmentation is done on basis of image feature technique. According to this technique, the image features such as edges, corners and other unique transitions on Web page images are analyzed to find regions with dense (packed) content. These regions are then joined into bigger sections, based on which Web page screenshot is segmented into *regions of interest* (ROIs). These regions contain graphical elements which are most important for visual testing

The set of ROIs which was extracted from the baseline image is then matched with the corresponding set of ROIs extracted from each IUT. For a given IUT, this matching step leads to an injective partial function from the set of ROIs of the baseline image (ROIB) to the set of ROIs of the IUT (ROIT). In other words, an ROI in the baseline image is mapped to at most one ROI in the IUT, but it may happen that ROI in the baseline image cannot be mapped to most similar ROI in the IUT (called missing ROI) or that an ROI in the IUT has no corresponding ROI in the baseline image (additional ROI). The comparison of each (ROIB, ROIT) pair is performed using a correlation-based image comparison technique, according to which the cross-correlation coefficient is calculated for every pair of images that are compared. The correlation coefficient value is between zero and one, where one means that images are absolutely identical and zero means that images are uncorrelated. On basis of calculated correlation coefficient and extracted features for each ROI, the decision whether ROIT matches ROIB is done.

If differences are detected between an ROI of the baseline image and a matching ROI of an IUT, Browserbite reports the pair (ROIB, ROIT) as a potential incompatibility. Similarly even missing or additional ROI detected during the matching of a baseline image and an IUT is also reported as an incompatibility.

When reporting differences via its user interface, Browserbite superposes ROIB to the ROIT using a transparency of 50%, thus enabling the user to check the extent of the incompatibility (see Figure 3).

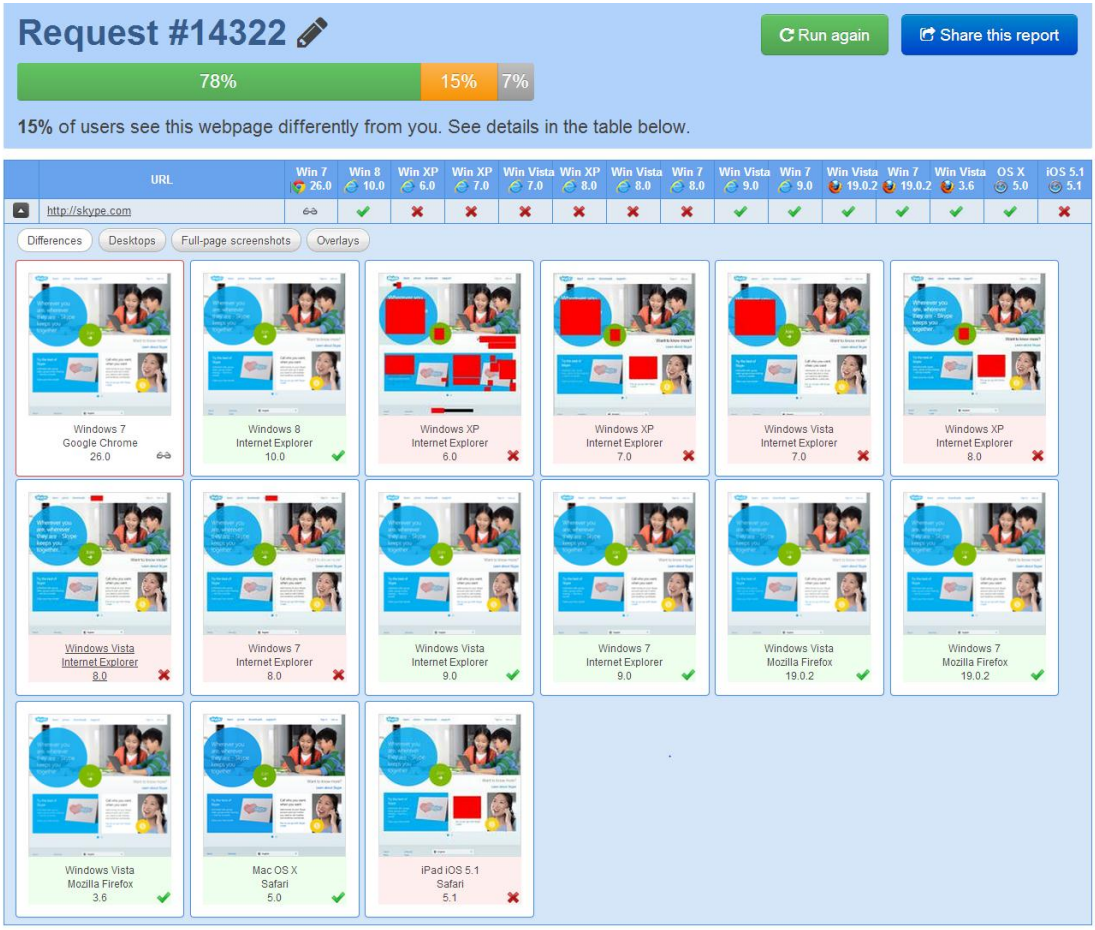

Figure 3. Processing www.skype.com home page by the Browserbite

2.2 Behavioral cross-browser compatibility testing techniques

State-of-the-art behavioral cross-browser testing techniques include works of Mesbah et al [33] (CrossT) and Dallmaier et al [3] [4] (WebMate). These methods are based on Web page crawling, during which the behavior of a Web application on a given platform is reverse-engineered and represented as a machine navigation model. Then the machine navigation models extracted from different platforms are compared.

The first phase in the behavioral cross-browser testing techniques is a state space exploration that is done by Web page crawling. The crawling approach of Mesbah et al in CrossT is based on the Crawljax [20], which by firing events on the user interface elements and analyzing the effects on the DOM tree in a real browser before and after the event, builds a state machine capturing the states of the user interface and the possible event-based transitions between them. A similar approach is used by Dallmaier et al who state that "in essence, WebMate is a crawler for Web 2.0 applications with the additional feature of being able to replay interactions to reach different states" [4]. When analyzing a Web application, WebMate repeatedly triggers actions on the Web page and extracts the state of the DOM tree after each action. The main difference between WebMate and CrossT crawling approach is in the depth of DOM tree level that is analyzed, as CrossT uses a much more low level representation of the DOM to distinguish states.

The navigation model that is produced by crawling a Web application is a finite-state machine. Each state in this model represents the screen observed by the end-user on the web browser, and each transition represents the users' actions (e.g. button click) that results Web application to transform from one screen into another.

The second phase in the behavioral cross-browser testing techniques is dedicated to the comparison of the navigation models created for the baseline browser and other browsers under the test. For the purpose of meaningful and efficient comparison of multiple navigation models Mesbah et al in the CrossT view and analyze these models hierarchically. The top level in their hierarchy is a graph representation of the finite-state machine with states (screens) embodied as unnamed vertices, while the second level is a full programmatic representation of each state (screen). Conceptually, the first level of the CrossT navigation model analysis captures the set of traces, i.e. alternating sequences of users' actions and screen transitions without referencing to the details of each screen, whereas the CrossT's second level captures precisely these details without any knowledge of transitions leading up or out of this screen. Basically the second level of CrossT navigation models analysis refers to the DOM tree of each screen on different browsers. Detection of trace-level differences is done by equivalence check of two navigation models, which produces the output as list of edges and nodes that mismatched in either of the two compared models. In order to compare the labels of two edges the method implemented in CrossT retrieves the DOM element and the corresponding event type from each edge and tries to reconcile the two labels. Two labels are considered as match if the edit distance between the two based on a combination of event types and tag names on the corresponding DOM trees is lower than a threshold. The second step for the CrossT analysis is a screen-level comparison which refers to the single-page cross-browser comparison techniques. For CrossT screen-level comparison is DOM-based and is built as an extension of the differencing engine in XMLUnit.

The navigation model created by Dallmaier et al in the WebMate is similar to those created by Mesbah et al. In addition to the DOM tree collected for each state of the model, WebMate also gathers additional parameters (i.e. elements' position) that could affect the visual representation for each state. In the second step WebMate compares previously collected data between the baseline browser and browser under the test. Before the checking of the child elements in the navigation model, Dallmaier et al normalize its position against the parent node thus it enables WebMate to report the consequential errors.

Although CrossT and WebMate both implement not only the behavior cross-browser compatibility testing techniques, but also DOM-based single-page cross-browser testing methods, it cannot be stated these tools represent some kind of hybrid technique. This is due to the fact that single-page cross-browser compatibility testing techniques implemented in mentioned tools suffers from a surplus of false positives and false negatives. More comprehensive hybrid techniques is implemented in CrossCheck, which combines Web application crawling and differencing to better detect cross-browser incompatibilities and is suggested by Choudhary et al at [27].

CrossCheck is a tool which was implemented using the combination of approaches that were previously used in WebDiff and CrossT which were described earlier in this chapter. While WebDiff detects screen-level cross-browser incompatibilities and CrossT's method finds trace-level cross-browser incompatibilities, CrossCheck employs these techniques in the respective capacities. The benefit of such a combination is the ability to provide visual comparison of screens that were dynamically generated during the crawling. In addition to using DOM-based method for identification of differences, CrossCheck uses the machine learning technique to build the classifier for the visual comparison of screen elements. The reason of such additional feature is to improve results of the visual comparison, as a simplistic DOM-differencing that is used in CrossT and WebDiff's hand-crafted heuristic are difficult to re-target and tend to miss many visual differences. Basically,

CrossCheck combines DOM-differencing features with strict visual-comparison ones to provide a more comprehensive predictor.

The approach of CrossCheck consists of three phases. During the first phase the crawling of a target Web application is done automatically within each of the two browsers (baseline and browser under the test). This phase results into two navigation models created for each of the browsers. The crawling is performed in the identical fashion under each browser for exercising precisely the same set of user-interaction sequences within the Web application in both cases.

The second phase in the CrossCheck method compares received two navigation models to check whether they are equivalent and extract the set of models' differences. This phase consists of three steps: trace-level comparison, DOM comparison of matched screen pairs and visual comparison of matched screen-elements pairs.

During the third final phase the retrieved model differences are being analyzed and collated into a set of cross-browser incompatibilities, which are then presented to the end user. For this analysis CrossCheck uses a decision tree to decide whether two screen elements being compared are different. The decision tree is built upon five features: size different ratio which detects difference in sizes, displacement that captures the Euclidean distance between the positions of corresponding screen elements, area, leaf DOM text difference which is a Boolean-valued feature that detects differences in the text content of screen elements, and image distance.

Although approach implemented by Choudhary et al in CrossCheck is efficient and its results outperform WebDiff and CrossT [27], there are still several limitations exist. The crawler does not support all kinds of user interactions with all kinds of widgets on all browsers, thus the improvement of this part of the CrossCheck system is needed which will lead to the overall improvement of the tool's results. The CrossCheck's core technique relies on DOM information obtained from the browser to detect cross-browser incompatibilities. However, in some cases the DOM node's information provided by the browser is inaccurate (i.e. inaccuracy in the geometric information of a node), which can result in false positives. The problem of the DOM-based techniques, as was mentioned in the previous section, also lays different interpretation of the same HTML by different browsers, which in turn also can cause false positives existence. The extension of the number of features for training the classifier and, probably, using more comprehensive classification methods (i.e. artificial neural networks) can lead to the better outcomes and decreasing of false positive and false negative results of the current technique.

3 Experimental comparison of image-based and DOM-based techniques

In previous Section we described two techniques for a single-page cross-browser compatibility testing. First is based on DOM elements usage, while the background of the second is in the image-processing. For the purpose of comparison in terms of accuracy of these techniques' results, we selected two tools – DOM-based Mogotest and image-processing based Browserbite. The maturity and robustness of these tools was the main criteria for their selection. Both of them are greedily available, easy in usage and support several browser-platform combinations. The brief description of Browserbite and Mogotest is presented in Table 2.

Table 2. Brief description of Browserbite and Mogotest

Tool	Technique used	# of configurations	Baseline browser	Commercial	SaaS
Browserbite	Image-processing	14	Chrome, last version	✓	✓
Mogotest	DOM	10	Chrome, last version	✓	✓

Cross-browser incompatibilities can be divided by analogy with tools for their detection into two main categories: (i) behavioral which are defined in terms of Web page functionality and (ii) Web page visual issues. Visual (or layout) issues are very common in Web application and result in differences in rendering the Web pages across various browsers on different platforms and are visible to the user. These incompatibilities can be further classified as differences in element size and position, visibility or appearance. While difference in element position and size are self-explanatory, differences in visibility consist of elements that are visible in one browsers and being non visible in others. The difference in the element appearance are defined when element's style (font, font size, color scheme etc.) or content is different across browsers. Behavioral incompatibilities involve the functionality of the Web application and are often caused by the different way of executing script elements by different browsers. Behavioral incompatibilities typically limit the ability of the user to access the specific Web page elements, such as widgets. Although the user would find the problem when trying to exercise the affected element, these incompatibilities are often harder to identify, because they may not have any visible effect (e.g. button is displayed correctly but does not work). This also means that visual incompatibilities often are also symptoms of behavioral issues. In other words, the behavioral incompatibility is not supposed to be expressed in visual issue (as the example with not working button), but if there is a visual incompatibility there is also very often behavioral issue beyond it (e.g. button is missing on the browser under the test, thus this may lead to impossibility using its functionality). Current Master Thesis is dedicated to the accurate diagnosis of cross-browser incompatibilities caused by the visual issues.

Although automated tools for cross-browser compatibility testing lighten the work of human testers, the problem of over-sensitivity still exists. Fundamentally, this is due to the fact that what constitutes an incompatibility, as opposed to a simple difference, is to some extent subjective [27]. Thus, setting a specific threshold to classify a difference as an incompatibility is far from trivia.

Table 3. Possible results provided by cross-browser compatibility testing tools

	Actual results		
Predicted results		***Incompatibility***	***Not incompatibility***
	Incompatibility	True positive	False positive
	Not incompatibility	False negative	True negative

All results generated by cross-browser compatibility testing tools can be divided into four categories – true positive, false positive, true negative and false negative – on basis of comparison of actual and predicted outcomes (see Table 3). If the actual incompatibility is found then the result belongs to the true positive category, if incompatibility does exist but it was not marked by cross-browser compatibility testing tool then this outcome is false negative. False positive results are those differences that were classified as an incompatibility by the cross-browser compatibility testing tool, but in fact they are not, in other words, user would not classify these differences to be an incompatibilities. Figure 4 shows the example of false positives generated by Browserbite (image comparison-based technique) and Figure 5 presents false positives generated by Mogotest (DOM-based technique).

Figure 4. On the left difference that user did not classify as incompatibility on the left (www.rik.ee) due to very insignificant minor shift of the ROIT in comparison to the ROIB and on the right (www.skype.com) due to insignificant shift in representing the text by browser under the test in comparison to the baseline (Browserbite)

Figure 5. Difference that user did not classify as incompatibility (www.getresponse.com). A minor layout difference when a button on the browser under the test is slightly moved from the position of the same button on the baseline browser (by Mogotest)

In the ideal cross-browser compatibility testing tool the amount of false results (both positives and negatives) equals to zero. However, nowadays there is no such ideal tool. One of the reasons of large amount of false positive results is, as was already mentioned earlier, over-sensitivity. As the influence of non-captured incompatibilities (false negatives) is much higher than the effect of false positive results, the step for getting reinsured is to increase the sensitivity of the tool and capture even minor differences that user would not classify as incompatibility. Such ‘conservative’

decision means that the tool would rarely miss incompatibilities, but on the other hand it may cause the generation of significant number of false positives.

To confirm this observation, the preliminary experiment was conducted. During the experiment we analyzed top 100 Web pages for Estonia according to Alexa [8] rendered in three different browsers (Chrome 22.0 as a baseline, IE8 and Firefox 16.0.1 as browsers under the test on Windows 7). The produced screenshots were manually compared with the aim to find visual incompatibilities (see Figure 6). For this purpose we created the following criteria to determine layout incompatibilities:

- Position and size differences[1]:
  - For the first 300 pixels from the top of the page, the position of element may not differ over 20 pixels in vertical or horizontal axes, otherwise this is an incompatibility;
  - After the first 300 pixels from the top of the page, the position of element may not differ over 40 pixels in vertical or horizontal axes, otherwise this is an incompatibility;
  - The position of element on horizontal axes may not differ over 40 pixels, otherwise this is an incompatibility;
  - The size difference (height or/and width) of the same element rendered in different browsers may not exceed 15 pixels.
- Visibility differences: if the element is visible (is presented) in one of the browser among baseline and browser under the test, but it is not visible in the other then the incompatibility is detected;
- Appearance differences:
  - The element's color is not visibly different, otherwise the incompatibility is detected;
  - The element's font, font style (bold, italic, underlined etc.) and font size are the same in two browsers that are compared, otherwise the incompatibility is detected;
  - The content of the element is the same in the baseline and browser under the test, otherwise incompatibility is detected.

[1] All values associated with the position and size differences were empirically determined, in other words these values were defined on basis of subjective perception of the Web page rendered on different browsers. Similar empirical rules were also used in [6]

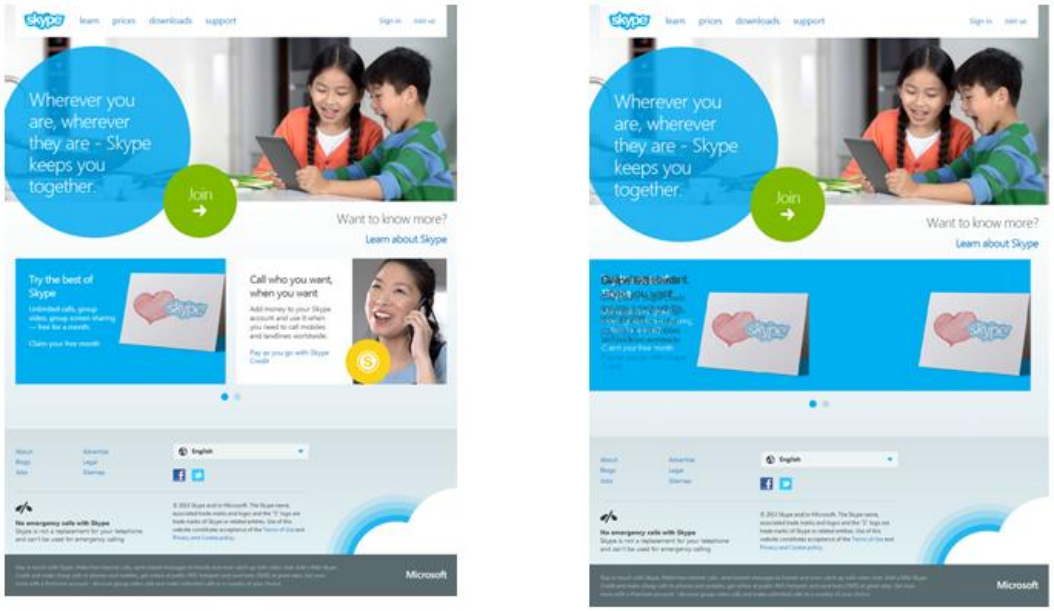

Figure 6. The www.skype.com home page rendered in Chrome 22.0 (on the left) and IE8 (on the right) on Windows 7

After the manual step of detecting incompatibilities was done, the same Web pages were analyzed by Browserbite and Mogotest cross-browser compatibility testing tools. The results provided by these tools were then compared to incompatibilities detected manually with the purpose of finding the amount of real incompatibilities that were not captured and captured potential incompatibilities that are not classified as so (see Table 4). The accuracy of these results was measured in terms of precision (1), recall (2) and F-score (3) [24].

Precision represents a fraction of retrieved instances that are relevant; in the classification task precision is a number of true positives divided by the total number of elements labeled as belonging to the positive class. This is a proportion of predicted positive cases that are correctly classified as real positives. Conversely, recall is the fraction of relevant instances that are retrieved and is the amount of true positives divided by the total number of elements that actually belong to positive class (sum of true positives and false negatives). A measure that combines precision and recall is a harmonic mean of these two and is called F-score.

$$P = \frac{TP}{TP + FP}, \tag{1}$$

$$R = \frac{TP}{TP + FN}, \tag{2}$$

$$F - score = 2 \times \frac{P \times R}{P + R}, \tag{3}$$

where TP – true positive results,
FP – false positive results,
FN – false negative results,
P – precision,
R – recall.

Table 4. Results of comparison manual incompatibilities detection and the usage of Browserbite and Mogotest tools

	# of detected incompatibilities	True positive	False positive	False negative	Precision	Recall	F-score
Manual	2854	-	-	-	-	-	-
Browserbite	3068	1988	1023	57	0.66	0.98	0.79
Mogotest	2790	1794	598	398	0.75	0.82	0.78

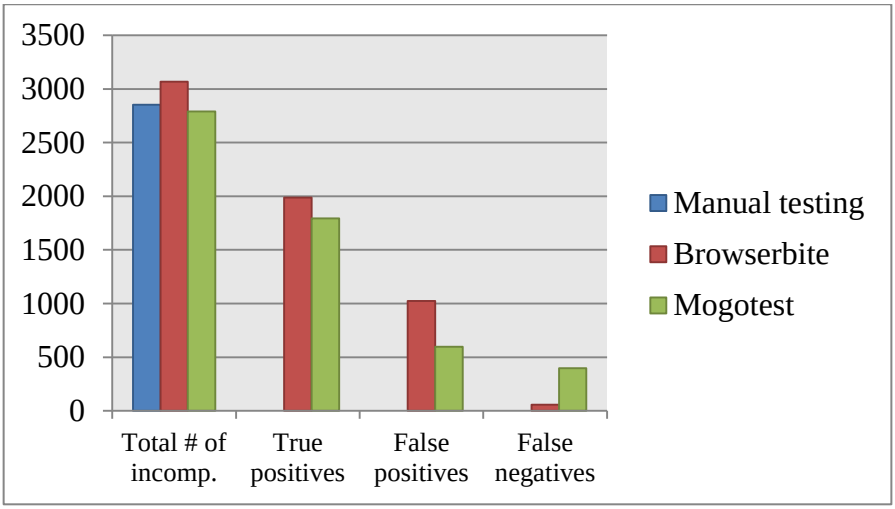

Figure 7. Results of comparison manual incompatibilities detection and the usage of Browserbite and Mogotest tools

The received comparison results have confirmed the observation of cross-browser compatibility testing tools 'conservatism'. However, while Browserbite rarely misses incompatibilities, as only 2% of them were unidentified, but instead has significant amount of false positives (34% of all detected incompatibilities), the Mogotest testing tool has more balanced results with less amount of false positives and bigger and more significant number of false negatives. Nevertheless, the overall accuracy of tools' results is comparable in terms of F-score (see Figure 7).

The decreasing of number of false negatives – the actual incompatibilities that were not detected – is a matter of improving the technique that is used by cross-browser compatibility testing tools.

4 Improving cross-browser incompatibilities detection with machine learning

Results of comparison of manual detection of cross-browser incompatibilities with its detection using cross-browser compatibility testing tools (Browserbite and Mogotest were taken as baseline techniques) has shown the low accuracy of tools' outcomes. The performance of Browserbite is characterized by very few false negatives and big amount of false positives which is 34% of all potential incompatibilities detected by this tool. The results of Mogotest are more balanced meaning almost the same percentage of false positives and false negatives. Mogotest's accuracy can be ameliorated by diminishing the ratio of both false results (positives and negatives), while improving Browserbite's recall is not reasonable on this step. To this end the decision of improving the accuracy of cross-browser compatibility testing tools by decreasing of the amount of false positives was done.

Reducing false positive amount problem was explored in two variants:

- Binary classification: We aimed to classify the potential incompatibilities reported by Browserbite and Mogotest into two categories: true positives (the potential incompatibility is perceived as such by a user) and false positives (the potential incompatibility is not perceived as such by a user);
- Quaternary classification: In this variant we aimed to classify potential incompatibilities reported by tools into four finer-grained categories (more detailed description is presented next).

For the purpose of decreasing the amount of false positives we processed the outputs of Browserbite and Mogotest (which are called potential incompatibilities) using binary and quaternary classification and built machine learning classification models for data handling. The rest of this section is described as following. Section 4.1 presents the experimental setup which is about datasets that are used for the classification models and classification principles that were applied. Section 4.2 depicts classification models and their configuration parameters. In Section 4.3 we present the evaluation of machine learning models by calculation its classification and ranked accuracy.

4.1 Experimental setup

For the experiment we analyzed more than 200 home pages from top Estonian Websites according to Alexa [8], among which we selected 140 Web pages that do not contain any significant amount of animation such as videos or flash animation that cause changes of pictures after each reloading of the page. Such selection criterion is explained by the fact that cross-browser compatibility testing tool will mark picture changing as an incompatibility, because after every reloading of the Web page images are going to be different. However, we are not interested in such evident results.

Each Web page was given as inputs to Browserbite and Mogotest, which generated reports with potential incompatibilities. The report generated by Browserbite for a given Web page consists of (ROIB ROIT) pairs with certain features. Mogotest also generated reports for each Web page and presents potential incompatibilities as DOM regions of page with specific parameters. The purpose of the experiment is to train and test machine learning models to classify the (ROIB, ROIT) pairs produced by Browserbite and Mogotest's outputs into true incompatibilities (true positives) and false alarms (false positives). To this end, we need a set of samples (golden standard) of true posi-

tives and false positives. Sections 4.1.1 and 4.1.2 explain in details how the golden standards were constructed for Browserbite and Mogotest respectively. In Section 4.1.3 we describe features of (ROIB, ROIT) pairs and DOM elements that are used for classification models' building.

4.1.1 Golden standard for Browserbite

Each Web page among 140 selected was given as an input to Browserbite, which generated more than 20×10^3 (ROIB, ROIT) pairs; in other words, Browserbite detected more than 20×10^3 potential incompatibilities.

For the construction of golden standard for training and testing classification models each (ROIB, ROIT) pair was manually classified into positive samples (i.e. correct results returned by Browserbite) and negative samples (i.e. incorrect results returned by Browserbite). During this manual classification the potential incompatibility was considered to be an actual incompatibility (i.e. referred to the "Correct" class) if there is a major layout or formatting difference (see Figure 8 on the right). In other words, the positioning of a fragment (text or visual element) in the ROIB is clearly different from that in the ROIT or vice-versa, or the formatting or colors in one image are distinct from those in the other. Otherwise the sample was labeled as "Incorrect" (see Figure 8 on the left).

Figure 8. Difference captured by Browserbite that is not classified as incompatibility by the user (www.twitter.com) on the left and difference corresponding to the actual incompatibility on the right (www.skype.com), where the text element on the baseline is absent in IUT, instead IUT contains another text

As the number of ROIs is high, only a subset could be manually classified with reasonable effort. Moreover, our aim was to obtain a balanced set of samples (50% true positives and 50% false positives), so affecting the classification models by skewing towards one class will be avoided. Thus, ROIs were manually classified in random order until 1350 positive samples were found. Past this point, we did not retain any further positive results, but continued sampling and manual classifying until 1350 negative samples were found, resulting in a balanced set of 2700 samples.

Such classification clearly involves a certain level of subjectivity and potential bias, as for example, some differences in rendering a Web page can be perceived as incompatibilities by some persons and for others they will be not significant. Accordingly, for the construction and evaluation of classification models, we recruited 40 subjects through social media and asked them to classify the above (ROIB, ROIT) pairs into the mentioned classes. Respondents were University students in the range of 20-25 years from 6 countries (Estonia, Ukraine, Russia, Germany, Italy and Hungary). The subjects came from different specialties: 60% with the IT background, 20% with economics and business background, 10% with philological background, and 10% others.

There was no point in asking respondents to classify each (ROIB, ROIT) pair as this would have caused decreasing of effort and thus unreliable results. So each respondent was asked to classi-

fy between 200 and 400 (ROIB, ROIT) pairs randomly sampled from the dataset with replacement into two categories – positive and negative samples.

In practice, the notion of incompatibility is not clear-cut. Some incompatibilities are rather minor and do not affect in any significant way the functionality or structure of a Web page, while others do. Accordingly, we also attempted to construct quaternary classification models according to the following class definitions:

- *Class C1*: there is no difference at all and thus (ROIB, ROIT) pair cannot be considered to be an incompatibility (see Figure 8 on the left);
- *Class C2*: there is a minor layout or formatting difference, but it is not significant, so the (ROIB, ROIT) pair cannot be considered to be an incompatibility (see Figure 9 on the left);
- *Class C3*: there is a major layout or formatting difference, so (ROIB, ROIT) pair is an actual incompatibility (see Figure 9 on the right);
- *Class C4*: there is critical difference that affects navigation or other functionality of the Web page (e.g. a missing button) or that affects its aesthetics in a significant manner (see Figure 8 on the right where text elements are different).

Figure 9. Non-significant (on the left) and significant (on the right) layout difference captured by Browserbite (www.rik.ee)

The level of subjectivity for the quaternary classification is obviously high, as for instance, the difference between major and minor layout differences can arguably differ across subjects. To decrease the subjectivity level 40 respondents were asked to classify the given pairs of ROIs also into four categories. If respondent classified an (ROIB, ROIT) pair as "Incorrect" (negative sample), he was asked to assign one of two categories – C1 or C2 – to this pair; and vice-versa if respondent classified (ROIB, ROIT) as "Correct" (positive sample), he was asked to assign class C3 or class C4 to this pair of ROIs. In such a way we obtained the quaternary classification for each (ROIB, ROIT) pair in the dataset.

As a result we obtained between 8 and 15 classifications per pair. To achieve uniformity, we randomly trimmed the dataset so that each (ROIB, ROIT) pair had only 8 classifications (i.e. 8 subjects per pair). The inter-rater reliability of the resulting dataset in binary case classification is 0.96[2] and in quaternary classification equals 0.94, which indicates little disagreement between judges. Accordingly, we assigned numeric value for each class (i.e. 1 for class "Correct" and 2 for class "Incorrect" in binary case classification, and from 1 to 4 for classes from C1 to C4 in quaternary classification). We calculated the average class for each (ROIB, ROIT) pair (rounded to the nearest integer) and used the resulting "average class" as a source of truth.

At the beginning we aimed to obtain the balanced set of samples and thus our manual classification into two classes flew out into 1:1 proportion. However, the reclassification done by 40 judges caused some imbalance in the initial dataset. In other words for binary classification the

[2] This value was calculated using the online Inter-Rater Reliability Calculator at http://www.med-ed-online.org/rating/reliability.html which implements the measure in [25]

number of samples belonging to class “Correct” equals 1361 and for class “Incorrect” equals 1339. However, such small imbalance is not significant. The number of instances of four categories was not selected purposely and also was achieved by manual classification by list of subjects. So that the number of instances of classes C1, C2, C3 and C4 are different for every class (see Figure 10).

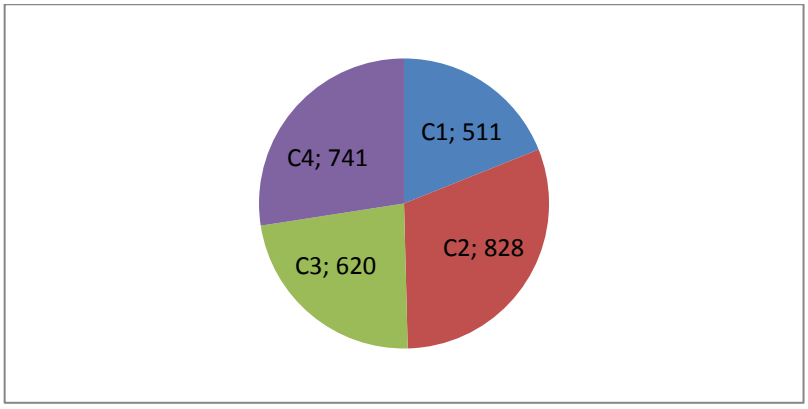

Figure 10. Number of instances of different classes for quaternary classification (Browserbite)

4.1.2 Golden standard for Mogotest

Among 200 home pages from top Estonian Websites according to Alexa, that were described in the previously, we selected 140 Web pages, each of which was given as an input to Mogotest cross-browser compatibility testing tool. The example of processing a Web page by this tool is presented in the Figure 11. While processing, Mogotest runs the Web page on different browser-platform combinations and performs a DOM-based comparison of the Web page rendered on the baseline browser and browsers under the test. When reporting difference via its user interface, Mogotest displays the baseline image of the Web page and its image rendered on the browser under the test side-by-side (however, it is also possible to superposes baseline image on the image rendered on the browser under the test) and reports the list of DOM elements that are potential incompatibilities, allowing the user to click on each element of the list. After clicking the specific list element, the corresponding DOM element is highlighted in the images rendered on the baseline and browser under the test (see Figure 12).

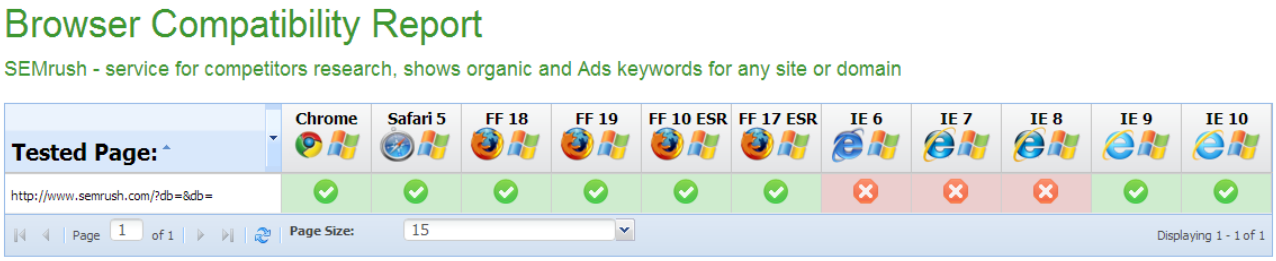

Figure 11. Processing www.semrush.com home page by Mogotest cross-browser compatibility testing environment

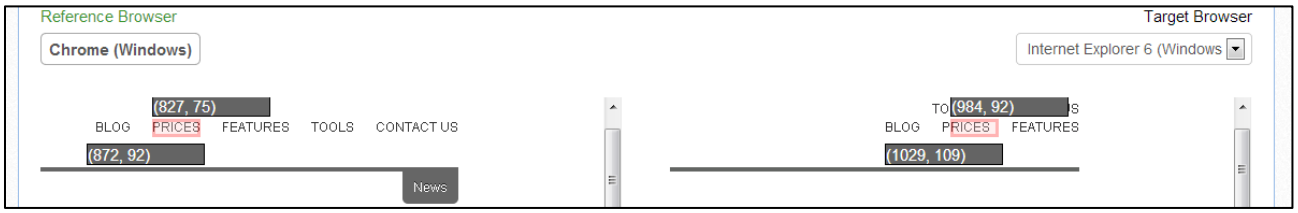

Figure 12. Highlighting DOM elements by Mogotest that are potential incompatibilities (www.semrush.com)

After processing selected Web pages, Mogotest generated more than 12000 potential incompatibilities. In order to construct a standard for training and testing classification models each DOM-element was manually classified, in the same way as it was done for Browserbite, into positive samples (i.e. correct results returned by Mogotest) and negative samples (i.e. incorrect results returned by Mogotest). In this manual classification the potential incompatibility was deemed to be an actual incompatibility (i.e. put in the "Correct" class) if there is a major layout or formatting difference (see Figure 13 on the right). In other words, the positioning of DOM element on the baseline is clearly different from that in the browser under the test or vice-versa, or the formatting and color of the DOM element are distinct from those in the other. Otherwise the sample was labeled as "Incorrect" (see Figure 13 on the left).

Figure 13. Difference captured by Mogotest that is not classified as incompatibility by the user (www.semrush.com) on the left and difference corresponding to the actual incompatibility on the right (www. semrush.com)

Our aim was to obtain a balanced set of samples (50% true positives and 50% false positives) so as to avoid affecting the classification models by skewing towards one class. As the number of true results produced by Mogotest is bigger than the number of false positives, we started with classifying DOM elements in random order until, for the sake of time, 500 positive samples were found. Past this point, we did not retain any further positive results, but continued sampling and manually classifying until 500 negative samples were found, resulting in a balanced set of 1000 samples.

By analogy to Browserbite, the quaternary classification was done for the Mogotest's dataset. However, for to sake of time, both binary and quaternary classifications were done without multiple judges, but only by us.

While the proportion of classes in the binary classification in 1:1 meaning that the amount of positive and negative samples is the same which was achieved by manual selection of necessary samples, the number of instances of four categories was not selected purposely and was achieved by manual classification. So that the number of instances of classes C1, C2, C3 and C4 for Mogotest are different for every class (see Figure 14).

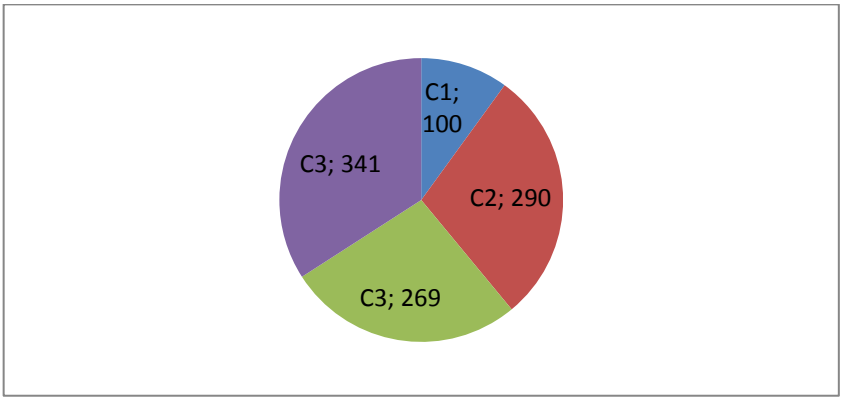

Figure 14. Number of instances of different classes for quaternary classification (Mogotest)

4.1.3 Feature set

We recall that an incompatibility reported by Browserbite consists of pair (ROIB, ROIT) where ROIB is an ROI in the baseline image and ROIT is a corresponding ROI in the IUT.

Given a pair (ROIB, ROIT), we extracted 17 features to build a sample for constructing classification models (these features can be derived in accordance of the image segmentation and comparison techniques that are used by Browserbite):

- 10 histogram bins (h0, h1… h9). For the classification purposes Browserbite calculated image histograms for each ROI which are encoded by 10 integers, providing ROIs with additional ten parameters. These 10 discrete bins represent pixel intensity distribution across the entire ROI image [30];
- Correlation coefficient between the ROI in the baseline image and ROI in the IUT. This is a number between zero and one. It is zero in case of low correlation between ROIB and ROIT, in other words, in case when two compared images are absolutely different;
- Horizontal and vertical position of the ROI (X and Y coordinates) on the baseline image;
- Horizontal and vertical size of the ROI (width and height) of the baseline image;
- Configuration index, which is a numerical identifier of the browser-platform combination of the IUT. Browserbite supports 14 browser-platform combinations, thus this number is an integer between 1 and 14;
- Mismatch Density (4)

$$MD = \frac{E}{T}, \tag{4}$$

where E is a number of ROIs in the IUT that are not matched 100% to an ROI in the baseline image, and T is a total number of ROIs in the IUT. Note, that this is a feature for an ROI inside the IUT. However, for the sake of convenience when constructing the machine learning models, we make the MD a feature of each (ROIB, ROIT) pair. All ROIs extracted from the same IUT will have the same MD (the MD of their enclosing IUT).

Mogotest uses DOM-based segmentation and comparison technique. Each DOM element has large amount of properties that characterize it. The decision was to extract for usage in the classification models the following parameters:

- Clickability: the property of the DOM element that equals to one if a DOM element has a click handler and zero otherwise. Static elements of the Web page does not play role in the behavior of the application while clickable elements do. As soon as some behavior incompatibilities usually cause visual incompatibilities on Web applications, the difference between clickable elements on the page can be cause also by behavioral issues;
- Tagname: name of the tag associated with the DOM element (e.g. div, p, h1 etc.). In the dataset that was created for Mogotest we used 20 different tagnames. On basis of this parameter there is a potential possibility to identify what types of DOM elements causes bigger amount of significant incompatibilities;
- Horizontal and vertical size (width and height) of the DOM elements on the baseline, as probably bigger elements can cause more significant differences;
- Horizontal and vertical position (X and Y coordinates) of the DOM elements. In the dataset we used coordinates for DOM element on the baseline and coordinates of the corresponding DOM element on the browser under the test;
- Font and font size of the DOM elements on the baseline. For image elements these value equal null;
- Configuration index, which is a numerical identifier of the browser-platform combination of the browser under the test. Mogotest supports 10 browser-platform combinations, thus this number is an integer between 1 and 10;
- Difference between X and Y coordinates of the DOM element on the baseline and X and Y coordinates respectively of the corresponding DOM element on the browser under the test.

The selection of the above parameters was driven by the following considerations. Elements' tagname, clickability and coordinates were taken by the analogy to the features that were used in WebDiff [5]. Size and configuration index were selected by analogy to the choice made in Browserbite. Font features were included because a significant portion of false positives produced by Mogotest contain text.

On basis of the extracted parameters it is possible to build, train and test classification models that are used to filter out false positives from the results that are produced by cross-browser compatibility testing tools.

4.2 Machine learning techniques

Possible decision for the described problem of filtering out false positives is to build a classification model for prediction whether the potential incompatibility is identified correctly on basis of its parameters (image-based features of (ROIB, ROIT) pairs or DOM parameters for Browserbite and Mogotest respectively). Classification tree (i.e. decision tree) [31], [9], [22] and artificial neural network are potential solutions.

We selected two classification methods namely classification trees and neural networks. Classification trees were selected due to their ability to produce human-interpretable results that can be shed some insight into the features that play the important role in the classification and their val-

ues. Neural networks were selected due to their ability to handle non-linear problems. Given the complexity of the domain, we considered that the resulting classification model could be non-linear. Specifically, in this Master Thesis the implementation of this machine learning techniques provided by Matlab is used. Configuration parameters of classification trees and neural networks that are used for the current problem we describe in Sections 4.2.1 and 4.2.2 respectively.

4.2.1 Classification tree

Classification tree is a predictive model which brings observations about an item into correlation with conclusions about the item's target value. In the tree structure, "leaves represent class labels and branches represent conjunctions of features that lead to those class labels" [31]. The use of classification tree in the context of described problem is motivated by the fact that it provides a convenient way to interpret the results. By analyzing the classification tree, we can obtain insight into thresholds and determine whether a given reported potential incompatibility is an actual incompatibility or not.

Classification trees are prone to over-fitting when too many features are given as input (in our case it is 17 features for Browserbite and 13 features for Mogotest). Accordingly, it is necessary to apply the feature selection for identification a subset of the features with the highest predictive power. Feature importance (a.k.a. predictor importance) was measured using Matlab. For each node of the tree, the estimation of risk, which is node error weighted by the node probability of belonging to the certain class, is done. Feature importance associated with this split is computed as the difference between the risk for the parent node and the total risk for the two children [18].

Measurements were done separately for binary and quaternary classification task for Browserbite and also for Mogotest. Results are presented in Table 5 and Table 6 for Browserbite and Mogotest respectively. The value of feature importance is in range [0; 1]; the higher value corresponds to higher predictive power.

Table 5. Classification tree feature importance for Browserbite

Feature	***Conf.***	***MD***	***Corr.***	***X***	***Y***	***Height***	***Width***	***h0***
2 classes	0.0425	0.0381	0.2025	0.0765	0.1225	0.0448	0.0393	0.0355
4 classes	0.0440	0.0912	0.1162	0.0802	0.0836	0.0413	0.0405	0.0274

Table 5. (continued)

Feature	***h1***	***h2***	***h3***	***h4***	***h5***	***h6***	***h7***	***h8***	***h9***
2 classes	0.0247	0.0337	0.0303	0.0515	0.0179	0.0281	0.0233	0.0206	0.0654
4 classes	0.0391	0.0222	0.0285	0.0387	0.0326	0.0291	0.0363	0.0290	0.0639

Table 6. Classification tree feature importance for Mogotest

Feature	***Click able***	***Tag name***	***Widt h***	***Heig ht***	***Font***	***Font size***	***X base- line***	***Y base- line***	***X test***	***Y test***	***Conf.***	***Diff X***	***Diff Y***
2 classes	0.008	0.001	0.082	0.071	0.002	0.001	0.095	0.046	0.02	0.01	0.001	0.12	0.041
4 classes	0.024	0.016	0.18	0.095	0.001	0.001	0.063	0.078	0.01	0.02	0.005	0.24	0.008

From the results presented in Table 5, it can be seen that the most important features for binary classification include correlation, Y and X coordinates, while the importance of all other parameters of (ROIB, ROIT) pairs are approximately the same. These results are unexpected due to

the fact that the comparison of ROIB and ROIT is done using correlation-based technique, which implies that correlation index is playing the main role in image comparison. As it can be seen, the position of ROIB (its X and Y coordinates) also has a big influence on creation of the classification tree. For the quaternary classification, according to Table 5, we can define four most important features, among them there are correlation, mismatch density and horizontal and vertical position of the ROIB. The importance of histogram bins in building the classification tree is approximately the same for all bins, which means that it is impossible to select the most significant color intensity in the images that are compared.

The results presented in Table 6 claim that for building the classification tree on basis of DOM element parameters the most significant role play difference between horizontal positioning of DOM elements that are compared, size (both width and height) of the element on the baseline browser and its horizontal coordinate on the baseline. Other parameters are less crucial. For quaternary classification case for Mogotest, according to Table 6, the most important features are also the difference between horizontal coordinates of elements on the baseline and browser under the test, the size of the element. The only difference from the binary case is in the higher importance of vertical coordinate of the element on the baseline. Other parameters seem to be less important. More detailed description and explanation of the most important features is done in Section 4.3.3, where we present the examples of classification trees both for Browserbite and Mogotest.

4.2.2 Neural network

Artificial neural network is a mathematical model which contains interconnected groups of artificial neurons, and processes information using connections between these neurons. In most cases neural network is an adaptive system that is able to change its structure during the learning stage [15]. Neural networks imitate the brain's ability to sort out patterns and learn from trials and errors, distinguishing and extracting the relationships that underlie the data with which it is presented. Recent advanced researches in neural networks for classification [15] [9] have proved that this is a promising alternative to standard classification models [15]. In this respect a key advantage of neural networks is their ability to adjust themselves to the data without explicit specification of functional or distributional form.

We selected the three-layered feed-forward neural network for the current problem. The feed-forward neural network is a type of artificial neural network where connections between the neurons do not form a directed cycle [15]. In the created neural network the first layer (the input layer) consists of 17 for Browserbite and 13 for Mogotest neurons that correspond to the number of features. The number of neurons in the output layer depends on the problem that is solved – whether this is binary classification of quaternary. In the case of binary classification the number of neurons in the output layer is two, and in case of quaternary classification this number is four (the number of classes the data is classified to). As the dataset is not linearly separable there was a need to introduce one or more additional "hidden" layers. In practice, very few problems that cannot be solved with a single hidden layer can be solved by adding another hidden layer [21]. This general rule was confirmed by our experiments, which showed that adding two hidden layers did not improve the accuracy (F-score) with respect to a single hidden layer. Thus the following work was done with the usage of a three-layered neural network.

The number of neurons in neural network is another important parameter that plays a big role is learning, validation and testing phases. Too few hidden neurons can cause under-fitting so

that the neural network cannot learn the details of the given data. Conversely, a too large number of hidden neurons can cause over-fitting, as the neural network starts to learn insignificant details.

In order to determine the appropriate number of hidden neurons we applied the empirically derived rules-of-thumbs. Some of these rules are as following:

- The number of hidden neurons should be around the mid-point between the size of the input and size of the output layers [14];
- The size of the hidden layer should never be more than twice as large as the input layer [26];
- To calculate the number of hidden nodes the general formula (5) can be used [14]:

$$Number\ of\ hidden\ neurons = \frac{2}{3} \times (Number\ of\ inputs + Number\ of\ outputs). \quad (5)$$

However, the described rules-of-thumb ignore the number of training cases, the amount of noise in the targets and the complexity of the problem, so they should not be used "as it is". As the number of input neurons equals 17 (the number of features) for Browserbite and 13 for Mogotest and the number of output neurons equals 2 and 4 for binary and quaternary classification respectively, we tried to find an optimal number of hidden neurons between 8 and 13. To this end, we trained the neural network with different number of neurons and calculated the F-score for each trained model (see Figure 15 and Figure 16). We experimentally found that the peak in F-score is reaches for a number of hidden neurons of 11 both for binary and quaternary classification cases for Browserbite data, and the number of hidden neurons equals 12 for Mogotest data and 10 for its quaternary classification.

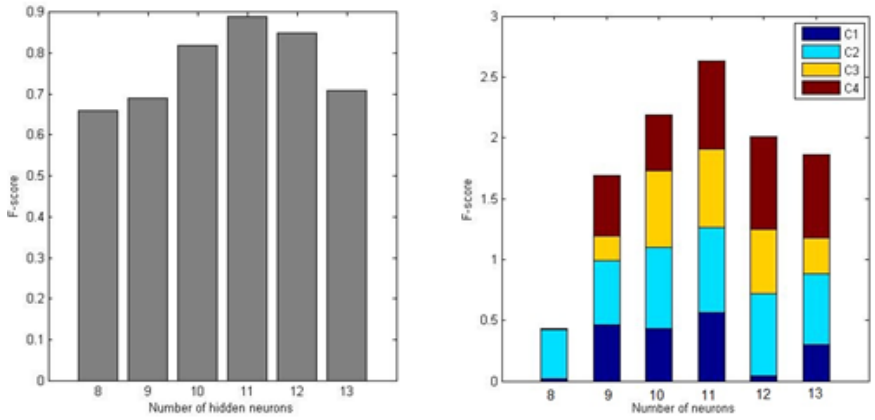

Figure 15. Number of hidden neurons vs F-score for binary (on the left) and quaternary (on the right) classification for Browserbite

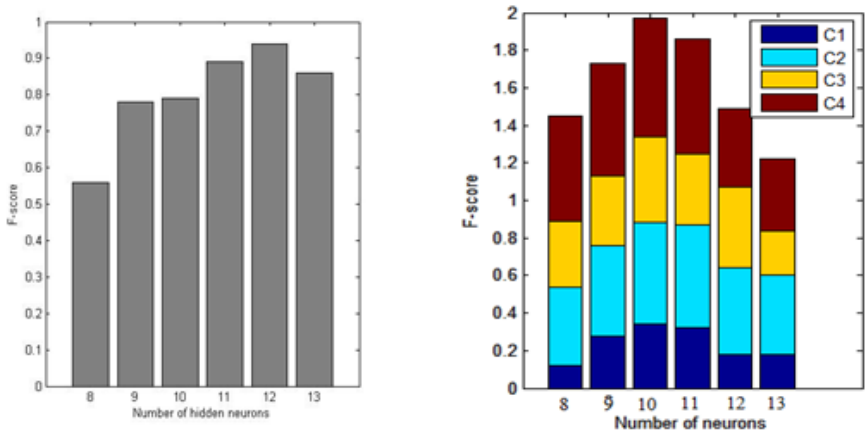

Figure 16. Number of hidden neurons vs F-score for binary (on the left) and quaternary (on the right) classification for Mogotest

The following evaluation of neural network is presented for the 3-layered feed-forward neural network with 11 hidden neurons both for binary and quaternary classification cases for Browserbite and 12 and 10 hidden neurons for Mogotest (see the conventional diagram for binary classification on Figure 17).

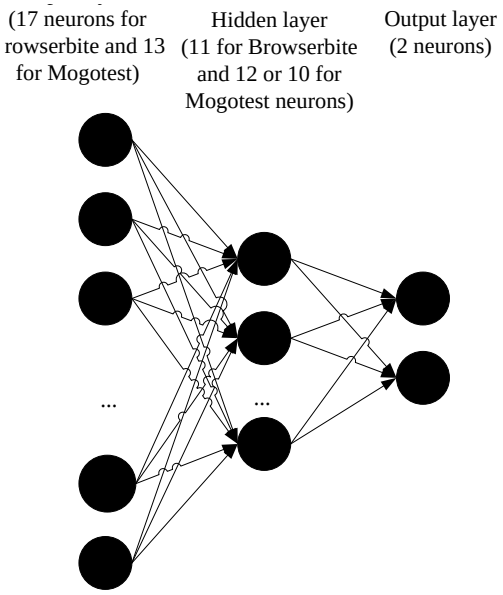

Figure 17. Conventional diagram of neural network for binary classification

As it was mentioned earlier, the implementation of neural network, as well as implementation of classification tree, was done using Matlab functionality. The evaluation of classifiers in the next section is done on basis of Matlab. In addition, for Browserbite neural network both for binary and quaternary cases was created on C++ using the open source computer vision library OpenCV [12]. This implementation was done only for Browserbite and is going to be integrated into the tool for decreasing the amount of generated false positive results of Browserbite.

4.3 Experimental results

Evaluation of machine learning techniques that were selected for filtering out false positives from the outcomes of cross-browser compatibility testing tools was done in terms of classification and ranked accuracy. In Section 4.3.1 the classification accuracy of classification tree and neural network both for binary and quaternary classification is calculated and compared with the bare-

bones techniques of Browserbite and Mogotest. Section 4.3.2 presents ranked accuracy, while the examples of created classification trees are represented in Section 4.3.3 for deeper explanation of the most important features that participate in classification.

4.3.1 Classification accuracy

In the first series of experiment, we aimed at comparing the classification accuracy of Browserbite and Mogotest without any machine learning post-processing, with that of Browserbite and Mogotest post-processed with classification tree and Browserbite and Mogotest post-processed with neural network.

Classification accuracy was measured in terms of precision (1), recall (2) and F-score (3).

The robustness of the evaluation results for Browserbite and Mogotest was validated using five-fold cross-validation method. According to this method, the dataset was partitioned into five equal parts, during one fold four parts were used to train the model and the remaining one was used to test the model. This process was repeated five times with each part playing the testing role ones. The results from each fold were then averaged to produce a single measurement of precision, recall and F-score for each method (classification tree and neural network). We measured execution time for creation of the machine learning model using "tic/toc" function in Matlab.

The results of these experiments are summarized in Table 7, Table 8, Table 9 and Table 10. Binary classification results are in Table 7 and Table 8 for Browserbite and for Mogotest respectively (see Figure 18), while quaternary classification results for Browserbite is presented in Table 9 and for Mogotest – in Table 10. The bare-bones Browserbite and Mogotest results were received from the preliminary experiment that is described in Section 3 (see Figure 7). In the quaternary case we do not provide results for plain Browserbite and Mogotest as these tools on its own do not classify into four categories.

Table 7. Accuracy results for the binary classification and plain Browserbite

Measurement	***Plain Browserbite***	***Classification tree***	***Neural Network***
Precision	0.66	0.85	0.963
Recall	0.98	0.78	0.887
F-score	0.79	0.81	0.923
Exec. Time (sec.)	-	1.43	1.71

Table 8. Accuracy results for the binary classification and plain Mogotest

Measurement	***Plain Mogotest***	***Classification tree***	***Neural Network***
Precision	0.75	0.921	0.951
Recall	0.82	0.644	0.911
F-score	0.78	0.75	0,93
Exec. Time (sec.)	-	1.36	1.76

Table 9. Accuracy results for quaternary classification for Browserbite

Measurement	*Classification tree*				*Neural Network*			
	C1	***C2***	***C3***	***C4***	***C1***	***C2***	***C3***	***C4***
Precision	0.56	0.53	0.65	0.61	0.65	0.69	0.69	0.77
Recall	0.61	0.67	0.53	0.59	0.52	0.75	0.64	0.73
F-score	0.58	0.59	0.58	0.6	0.58	0.72	0.66	0.75
Exec. Time (sec.)	1.72				1.97			

Table 10. Accuracy for quaternary classification for Mogotest

Measurement	*Classification tree*				*Neural Network*			
	C1	***C2***	***C3***	***C4***	***C1***	***C2***	***C3***	***C4***
Precision	0.21	0.35	0.28	0.38	0.38	0.56	0.52	0.64
Recall	0.18	0.34	0.29	0.32	0.31	0.52	0.42	0.63
F-score	0.2	0.34	0.28	0.35	0.34	0.54	0.46	0.63
Exec. Time (sec.)	1.52				1.67			

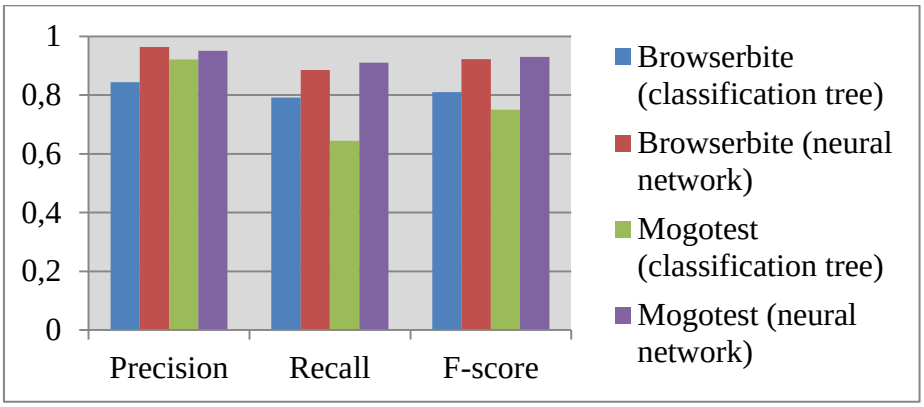

Figure 18. Overview of the results for binary classification for Browserbite and Mogotest

It can be seen from the results that neural network outperform by far the classification tree in both binary cases of classification for Browserbite and Mogotest, and also in the quaternary cases. In binary case neural network achieves a very high precision at the expanse of some degradation in recall. The F-score, while classifying into two classes using neural network, is approximately equal for the datasets collected for Browserbite and Mogotest (0.923 and 0.93 respectively), however, the degradation in recall for Browserbite dataset is slightly bigger, but the recall is still in the acceptable ranges. The improvement in precision for both datasets in binary cases is considerably high. Such improvement states that for Browserbite the false positives rate can be decreased to less than 4% out of all positive results, and for Mogotest – to 5%.

In the binary classification case, the improvement in precision provided by the classification tree is considerably less important, and comes at the expense of a drop in recall. Although for Mogotest the value of precision is high, the degradation in recall means that fewer incompatibilities will be identified as so; the impossibility to identify 36% (in comparison to 18% for bare-bones Mogotest) of real incompatibilities is an extremely significant value and the improvement in filtering out false positive results cannot balance this fact. The overall accuracy of the binary classification tree for Browserbite is not significantly changed in comparison to plain Browserbite (0.81 and

0.79 F-score respectively): although the fraction of filtered out false positives is higher than for the bare-bones technique, the amount of false negatives tend to significantly increase.

Execution times for both methods – classification tree and neural network – are comparable both for Browserbite and Mogotest.

In the quaternary case the same observation can be done – the neural network outperforms the classification tree, which is visible for all classes. For Browserbite the classification tree shows approximately equal overall accuracy of the results for all classes – 0.58 for classes C1 and C3, 0.59 for class C2 and 0.6 for C4. This means that so far for some classes, such as C3 and C4, the filtering out of false positive results is performing better than for other classes, however, the recall is then significantly lower than for C1 and C2 classes. Neural network's results for Browserbite are more "widespread". The best and also the worst results are obtained in "extreme" C1 and C4 classes. Insufficient accuracy for C1 class can be explained by the comparable small amount of C1 class instances in the dataset. In other words, the fraction of (ROIB, ROIT) pairs in the dataset that belong to class C1, which states that there is no difference at all between ROIB and ROIT, is lower that fraction of (ROIB, ROIT) pairs belonging to other classes. This is due to the fact that such kind of cases, when Browserbite highlights as an incompatibility sections where there are no difference at all, are rare and harder to be captured. While examples of (ROIB, ROIT) pairs belonging to class C1 are infrequent, in contrary, pairs of ROIs that identified as absolutely different, thus belonging to class C4, are frequent due to the fact that Browserbite's technique is directed to capture even minor incompatibilities, not to mention significant ones. Thus the accuracy for class C4 is higher than the accuracy for all other classes. Another complicated problem is to distinguish between classes C2 and C3 that states for minor and insignificant and major layout and formatting issues to be captured respectively. The difference between these two classes is sometimes subjective, meaning that for some people the same difference can be minor and for others this difference arises as incompatibility. The neural network presents the same precision for two middle classes; however, the recall for class C2 is significantly higher than the recall for class C3 that equal to 0.75 and 0.64 respectively. This means that on basis of parameters of (ROIB, ROIT) pairs, the classifier also can be missed between these two classes.

Although in the quaternary classification for Mogotest neural networks also outperform classification trees, the results even for neural networks are poor. In essence, for class C1 such results can be explained by the small amount of instances of this class in the dataset (only 10%). Thus for the continuation working with the Mogotest's quaternary case classification there is an important need in dataset increasing.

The execution times for both methods in the quaternary classification case are also comparable. It is worth to notice that execution times are significantly low when creating the classifiers (both classification tree and neural network).

4.3.2 Ranked accuracy

Receiver Operating Characteristics (ROC) is another metric for comparing predicted and target values in a classification model, particularly when ordering (ranking) of the produced results is relevant. In case in cross-browser compatibility testing tools such ranking is relevant as testers would typically examine the detected incompatibilities in order of importance. ROC analysis investigates the relationship between sensitivity and specificity of the classifier [24] [11]. Sensitivity states for the true positive rate and measures the proportion of correctly classified instances, while

the specificity is a true negative results which measures the negatives correctly classified. Conventionally, the true positive rate is plotted against the false positive rate. Such kind of plot is called a ROC curve. The best possible prediction methods results into the upper left corner with the coordinate (0;1) of the ROC space, that of representing 100% sensitivity and specificity which means no false positives and no false negatives. This is a so called perfect classification to which all classification models aim to. A completely random guess would results into points along a diagonal line from the left bottom to the right top corner. This diagonal divides the ROC space and all points that lie above the diagonal represent satisfactory classification results, while points below the diagonal represent poor results.

The area under the ROC curve (AUC) [11] measures the discriminating ability of the classifier. Since the curve is located in the square unit, the possible values for AUC are between zero and one. The AUC equals one is achieved if the classifier scores every positive sample higher than every negative and AUC equals zero in vice versa case. If the value of AUC equals 0.5 this may mean different scenarios [11]: (i) the classifier assigns the same score to all test examples without matter they are positive or negative, and consequently ROC curve is ascending diagonal; (ii) the per-class score distributions are similar, which results in the ROC curve close to the ascending diagonal; and (iii) the classifier gives half of a particular class the highest scores, and the other half – the lowest scores. The high value of AUC means the increasing of likelihood that an actual positive sample will get the higher probability of being positive than an actual negative instance. When the outputs of the classifier are ranked by the probability of a results belonging to a given class, a high AUC entails that true positives are most likely to be found at the top of the ranked list.

We built ROC curves for the created classifiers and results are presented on the following figures. Figure 19 shows the ROC plot for the classification tree and neural network for the binary case for Browserbite and Figure 20 – for Mogotest, while Figure 21 shows the same for the Browserbite's quaternary classification and Figure 22 – for Mogotest's. In Table 11 the values of AUC for both methods are presented.

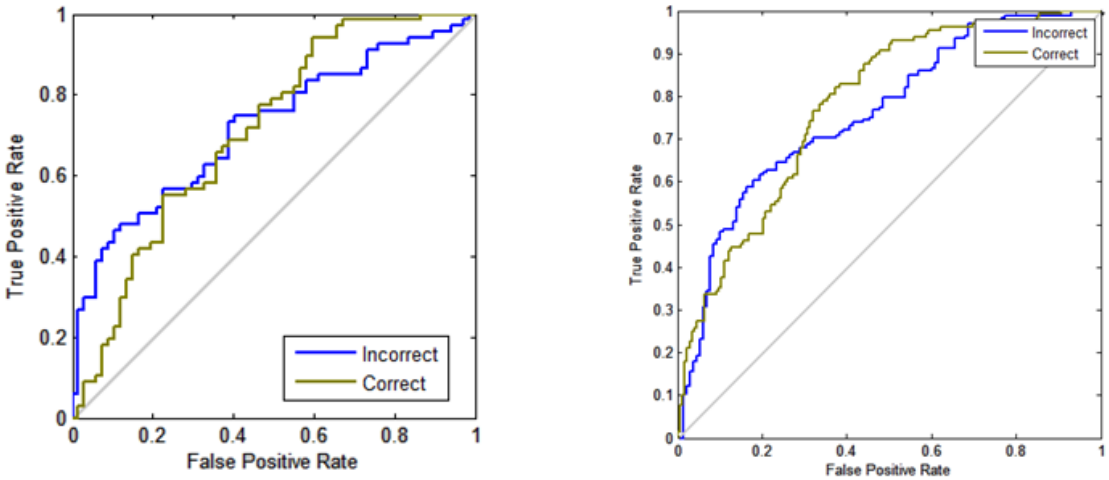

Figure 19. ROC curve for binary classification by classification tree (on the left) and neural network (on the right) for Browserbite

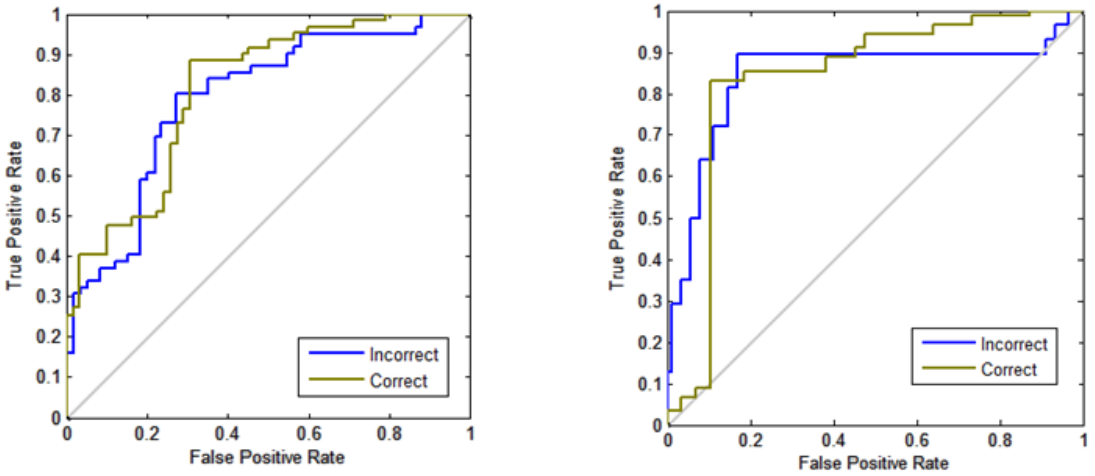

Figure 20. ROC curve for binary classification by classification tree (on the left) and neural network (on the right) for Mogotest

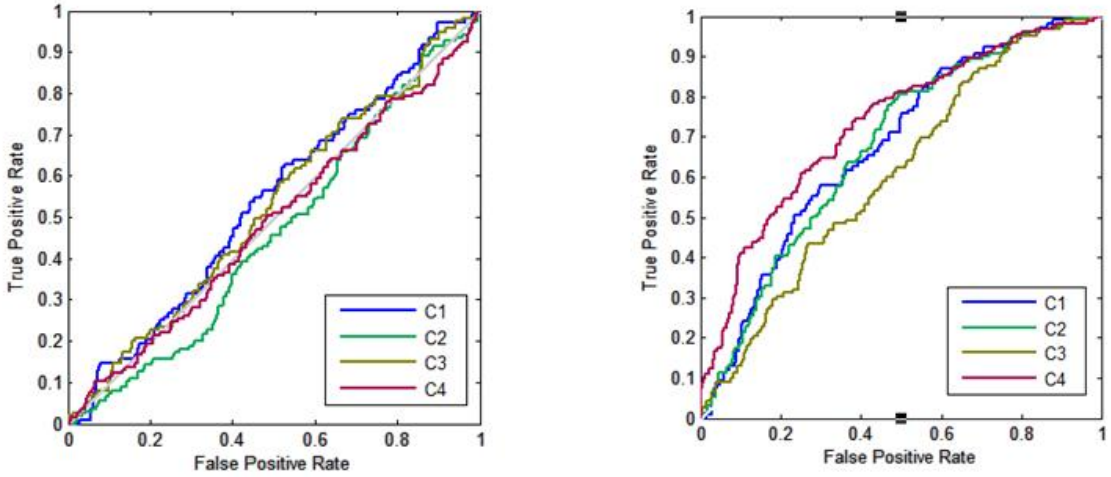

Figure 21. ROC curve for quaternary classification by classification tree (on the left) and neural network (on the right) for Browserbite

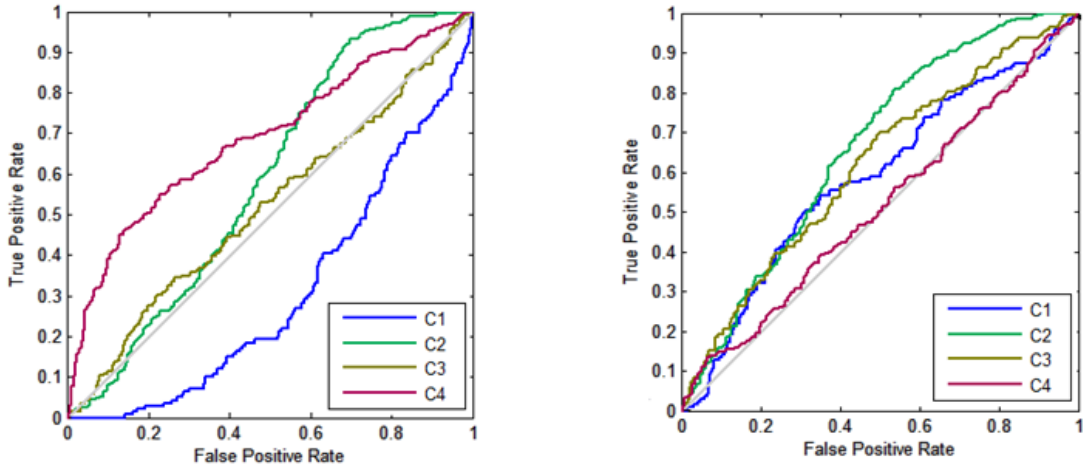

Figure 22. ROC curve for quaternary classification by classification tree (on the left) and neural network (on the right) for Mogotest

The ROC plots and AUCs confirm that neural networks outperform classification trees and achieve an arguably high level of accuracy overall. The AUCs also show that the ranked accuracy of classification trees for the class “Correct results” (a true positive in Browserbite and Mogotest) is

relatively high for both systems (Browserbite and Mogotest), meaning that classification trees could be used to find the most likely incompatibilities among those reported. Neural networks for binary classification for both tools also show that ranked accuracy of "Correct" class is higher than that for "Incorrect" class.

The ROC and AUCs also confirm that in the quaternary case, neural networks outperform classification trees, but still, the results can and should be improved. We hypothesize that other features and/or larger training set are needed to achieve better accuracy in the quaternary case. Nevertheless, for Browserbite neural network's results for "extreme" C1 and C4 classes achieve higher ranked accuracy, while that of for middle C2 and C3 classes. This is explained by the fact that differences that belongs to classes C1 and C4 is easier to reveal than difference belonging to classes C2 and C3, which classification may be a matter of subjectivity. On contrary, quaternary classification for Mogotest by neural network perform almost the same results for all classes, however, "extreme" cases for classes C1 and C4 tend to have smaller AUC resulting in better classification into middle classes C2 and C3.

On basis of previous observations, neural networks can be used to find the most likely incompatibilities among those reported by cross-browser compatibility testing tools.

Table 11. Area under the curve values for the classification systems

		Classification tree	***Neural network***
Browserbite: 2 classes	Correct	0.87	0.92
	Incorrect	0.8	0.88
Mogotest: 2 classes	Correct	0.89	0.94
	Incorrect	0.86	0.89
Browserbite: 4 classes	C1	0.62	0.72
	C2	0.43	0.69
	C3	0.59	0.61
	C4	0.58	0.76
Mogotest: 4 classes	C1	0.34	0.65
	C2	0.62	0.71
	C3	0.51	0.63
	C4	0.68	0.54

4.3.3 Examples of classification trees

In order to understand which features play an important role in the detection of false positives, in this Section the classification trees for binary and quaternary classification problems are presented. Trees were built using Matlab numeric environment.

For the binary classification the number of nodes in the tree equals 527 for Browserbite and 94 for Mogotest. In the quaternary classification task the classification tree consists of 773 nodes for Browserbite and 278 nodes for Mogotest. However, it is possible to perform a tree pruning by merging leaves at the same tree branch. We used the pruning sequence calculated by Matlab by default when building this classification tree. The first five levels of pruned trees are presented in Figure 23 for binary classification for Browserbite, Figure 24 for binary classification for Mogotest

and Figure 25 and Figure 26 for quaternary classification for Browserbite and Mogotest respectively.

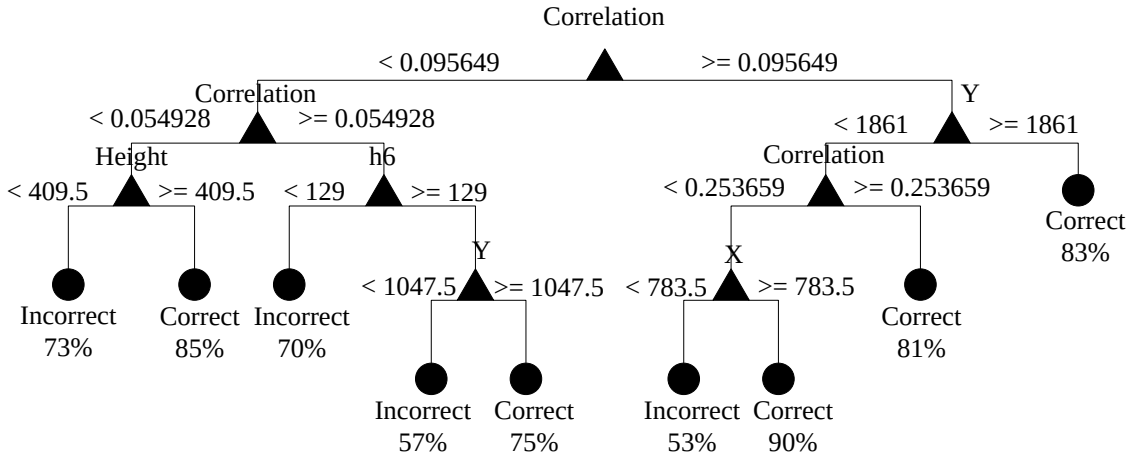

Figure 23. Pruned classification tree for the binary classification (Browserbite)

Although we highlighted three the most important features for the classification tree built for binary classification for Browserbite (see Section 4.2.1), in the pruned classification tree (see Figure 23) two other parameters – height of the ROI and h6 (histogram bin) – play their role. According to the classification tree, when the correlation index of the (ROIB, ROIT) pair is greater than 0.096 then there is a higher probability to classify such pair as Correct. These results are expected as this index captures the difference between two images and image comparison technique that is used by Browserbite is based on the correlation computation. More interesting observation is the fact that the higher value of vertical coordinate Y is associated with true positives, meaning that incompatibilities detected at the bottom of the page are more likely to be true incompatibilities compared to incompatibilities detected at the top of the Web page. This may be due to the fact that small displacements of elements at the top of the Web page ripple down into more visible incompatibilities at the bottom of the Web page, and thus clear incompatibilities are more likely to occur at the bottom of the page. Almost the same conclusion can be done for the value of horizontal coordinate X. Another observation is the influence of height on the incompatibility classification. It can be assumed that the smaller size of the ROI (vertical size) corresponds to the more precise calculation of correlation for this segment of image.

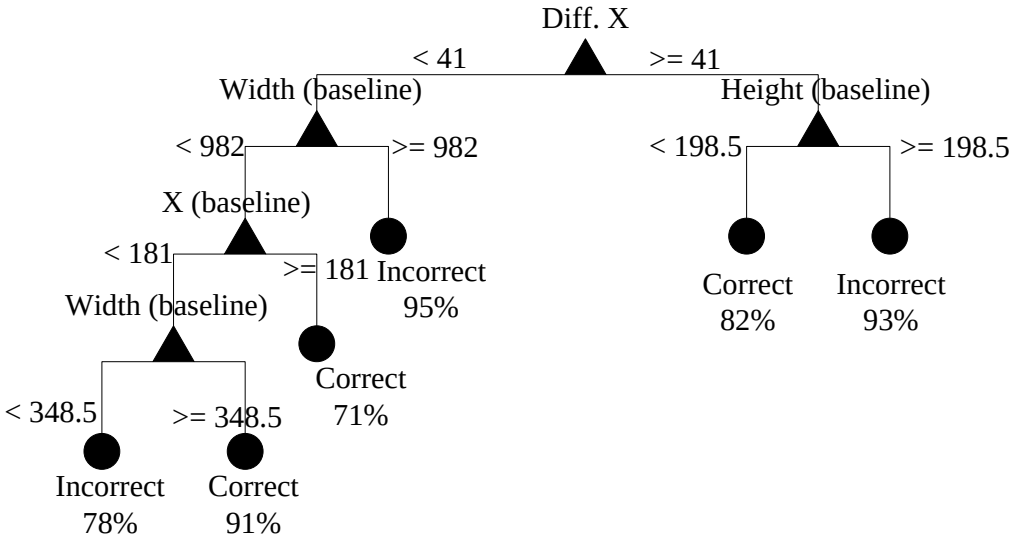

Figure 24. Pruned classification tree for the binary classification (Mogotest)

According to the Figure 24, for Mogotest the important part in binary classification play the difference between horizontal coordinates X of the baseline browser and browser under the test. If this difference is less than 41 pixels then the higher probability to get the false positive results, meaning that the observed difference is not a real incompatibility. These results are explicable as the rules defined for the preliminary experiment (see Section 3) also claim that if the horizontal positioning difference is small (and there are no other visible difference between images) then it should not be classified to be an incompatibility. Interesting to notice, that the smaller size of the elements that are compared tend to associate with true positive results (i.e. if height is less than 198.5 then the belonging to the class "Correct" has 82% probability). This may be due to the fact that while comparing small segments of images fewer mistakes are done. Also as in the previous classification tree, the larger the horizontal coordinate X of the element the higher the probability of belonging it to the true positives. This confirm the hypothesis that the small displacements of elements at the beginning of the Web page can cause more visible differences and thus real incompatibilities at the end of the page (in the current case, the beginning is the left side of the page and the ending is the right side of the Web page).

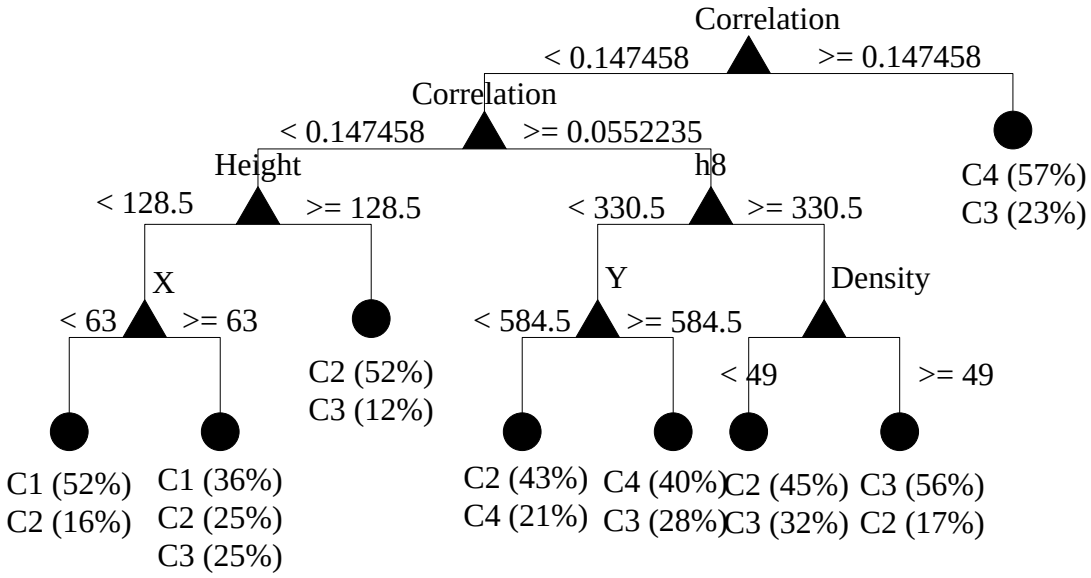

Figure 25. Pruned classification tree for the quaternary classification (Browserbite)

According to the classification tree for the quaternary classification for Browserbite (see Figure 25), the classification into neighbor classes (i.e. C1 and C2, C2 and C3, C3 and C4) has the higher value of "disagreement". In other words, it is unlikely that the same parameters can cause the classification to class C1, for example, with some probability and into class C4 with almost the same probability. Usually classification model "chooses" between classes the difference between which is small, like between C2 and C3. The results presented on the Figure 25 in some way repeat the results presented by the classification tree for binary classification on Figure 23, which is expectable. For instance, the value of correlation >= 0.147458 leads to the classification into class C4 with the probability 57% or class C3 with the probability 23%. As both of these classes corresponds to class "Correct" in binary classification this results repeats the same observation done for the binary classification – the higher value of correlation the higher probability that there is a real incompatibility. However, it can be noticed that the subjectivity level of classifying ROIs into classes C2 and C3 plays its role in the classification tree very clearly. The classification model very often

chooses between these two classes to classify (ROIB, ROIT) pair to. Also the higher value of vertical coordinate leads to the true positive results.

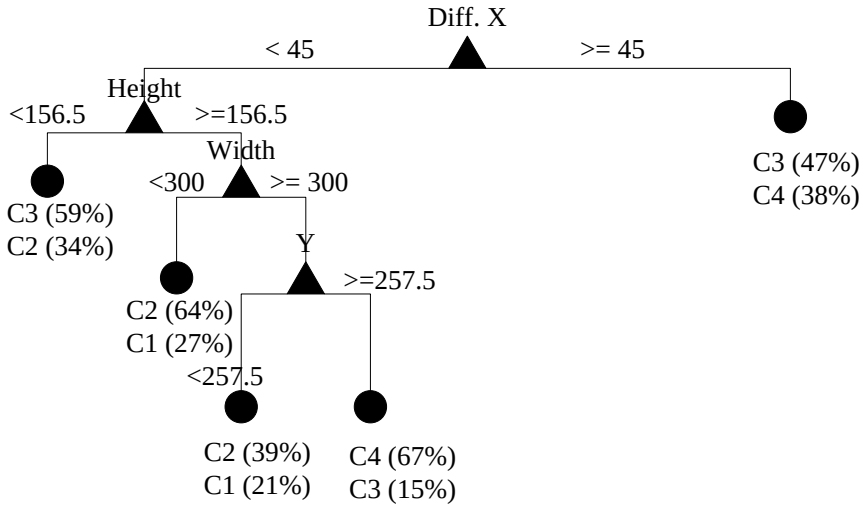

Figure 26. Pruned classification tree for the quaternary classification (Mogotest)

According to Figure 26, for quaternary classification for Mogotest the most important features are almost the same as for binary classification. Difference between horizontal coordinates still defines the element classification and, by the analogy to the binary case, if this difference is higher than 45 pixels than with the higher probability the element will belong to one of C3 or C4 classes (that corresponds to “Correct” class in binary classification). Interesting to notice that for quaternary classification tree smaller sizes of DOM elements corresponds to higher probability of their incorrect classification. Also the observation of rippling down displacements that causes real incompatibilities at the bottom of the page is confirmed.

5 Towards combining image-based and DOM features

As the results of filtering out false positives from the cross-browser compatibility testing tools such as Browserbite and Mogotest were successful and provide a high accuracy in the binary case classification, the question appeared how could one go beyond? The possible direction is to merge the features that are used in image-based technique (histogram bins, correlation coefficient, mismatch density etc.) with the DOM parameters that can be extracted. As the baseline technique for this case we selected Browserbite and defined the additional task as to see whether it is possible to improve results of cross-browser compatibility testing tool that is using image-based technique by including additional DOM-based features.

To this end we conducted the experiment is which the set of 700 (ROIB, ROIT) pairs was extracted from the dataset collected for the Browserbite. For each ROI we found additional DOM parameters that are described in the Section 4.1.3. The classification models (classification tree and neural network) were then built and evaluated for the taken dataset.

By the definition, each ROI segmented by the Browserbite is not a DOM element. For example, there exist cases when ROI is a combination of several DOM elements (Figure 27), or ROI contains some parts of one or more DOM elements but not the whole element (Figure 28). However, in most cases ROIs are slightly different from the corresponding DOM elements (Figure 29). In other words, some ROIs extracted by Browserbite can be perfectly mapped to DOM elements of the Web page.

The Web pages, rendered previously by Browserbite, were run on three different browsers – Chrome 22.0 as a baseline, IE8 and Firefox 16.0.1 as browsers under the test on Windows 7. In each browser the Web pages were segmented into DOM elements by using the browsers' inbuilt tools (i.e. in Chrome it can be done by clicking on the Web page with the right button of the mouse and select "Check element" in the menu). The next step was to compare ROIs for the selected Web page generated by Browserbite with the corresponding DOM elements on the Web pages rendered on different browsers. For the current experiment we selected only those ROIs that correspond to one DOM element of the Web page, building a very nice mapping between ROIs and DOM elements. The reason of selection such kind of ROIs lies in the fact that if one ROI corresponds to multiple DOM elements then there is a need to aggregate features of multiple DOMs, which requires a separate study and is an another complicated problem to be solved. By applying such matching selection, the set of 700 ROIs with corresponding DOM parameters was collected. The extracted set of ROIs is imbalanced, meaning that the number of samples belonging to different classes differs. The number of ROIs belonging to class "Correct" (positive samples) equals 375 and the number of ROIs in class "Incorrect" is 325.

Figure 27. One ROI (on the left) segmented by Browserbite that corresponds to two DOM elements on the right (www.skype.com)

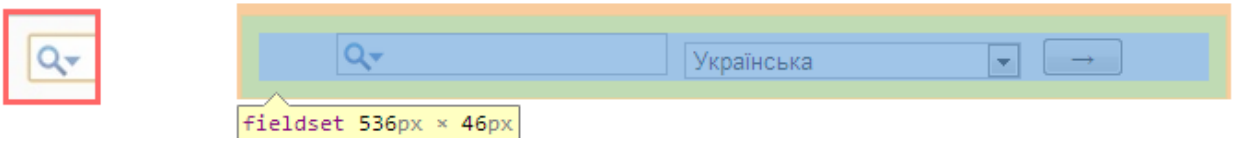

Figure 28. One ROI (on the left) segmented by Browserbite corresponds to the small part of the DOM element on the right (www.wikipedia.org)

Figure 29. One ROI (on the top) segmented by Browserbite corresponds to one DOM element on the bottom without difference (www.wikipedia.org)

In the collected dataset each sample contains the following features:

- Values of 10 histogram bins;
- Correlation coefficient;
- Horizontal and vertical position of corresponding DOM element on the baseline and on the browser under the test;
- Horizontal and vertical position of ROI on the baseline browser;
- Difference between horizontal and vertical positions of the DOM elements on the baseline and browser under the test;
- Horizontal and vertical size of corresponding DOM elements on the baseline;
- Horizontal and vertical size of ROI on the baseline browser;
- Configuration index (the integer number between 1 and 2 as this experiment uses only two browsers under the test);
- Mismatch density;
- Clickability;
- Tagname;
- Font and font size of the corresponding DOM elements on the baseline.

We suggested only binary classification in this case: into positive ("Correct") and negative ("Incorrect") samples. The classification models described earlier – classification tree and neural network – were built using extracted 29 features. The number of layers and number of nodes for the neural network was selected in the same way as it was done in previous experiments and we found out that the optimal number of layers is 3 and he number of hidden neurons equals 21 – using the neural network with such parameters we obtained better results. For the evaluation purposes for both classifiers we measured classification and ranked accuracy. Figure 30 presents ROC curves for the classification tree and neural network classifier respectively. We also measured accuracy for classifying selected 700 ROIs without merging parameters – for image-based features and DOM features separately – to compare these results with the results of combining features (see Table 12).

Table 12. Accuracy results for the binary classification in case of combination of image-based extracted feature and DOM parameters

Measurement	Classification tree			Neural network		
	Image-based features	***DOM features***	***Combined features***	***Image-based features***	***DOM features***	***Combined features***
Precision	0.78	0.77	0.41	0.91	0.88	0.89
Recall	0.74	0.71	0.53	0.82	0.78	0.71
F-score	0.76	0.74	0.46	0.86	0.83	0.79
AUC for "Correct" class	0.84	0.83	0.57	0.92	0.88	0.86
AUC for "Incorrect" class	0.78	0.74	0.68	0.85	0.82	0.81

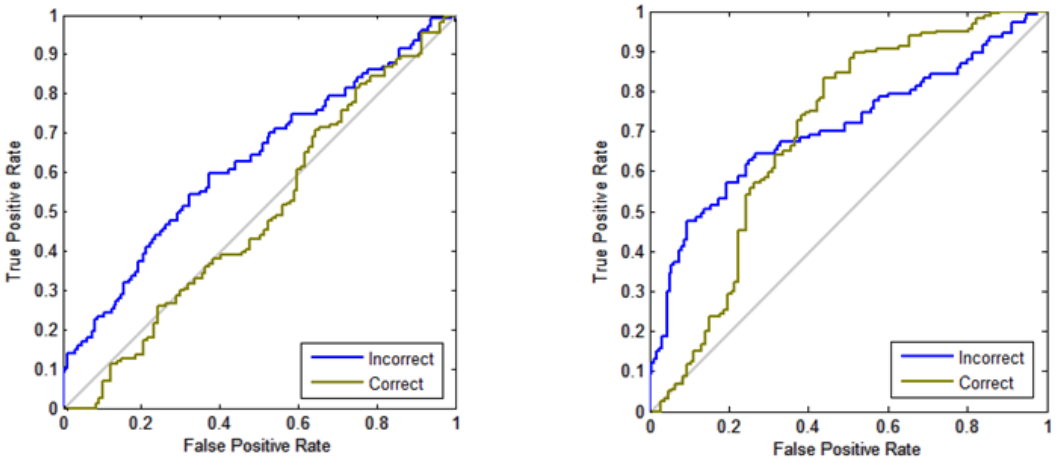

Figure 30. ROC curve for binary classification by classification tree (on the left) and neural network (on the right) for combined parameters

We can observe no improvement in the accuracy when using the combination of features. These may be due to the over-fitting of the classification models, so that the accuracy, both F-score and AUC, are much lower than that without merging features. However, feature selection is this case is not interesting as we were concerned with the fact whether combination of ROI features and DOM parameters can improve the results.

Neural network also outperforms the classification tree in this case, both in its F-score and ranked accuracy. It can be stated that using the described set of features it is possible to determine the most likely incompatibilities among those reported. However, although the precision is in acceptable interval, the drop of recall is significant in comparison with those without features combination. Such drop of recall can be explained by the imbalanced set of samples that are used in the classification.

There are following limitations in usage of the technique of feature combination in the state as it is at the moment. First, the dataset consists of ROIs that corresponds to the DOM elements

almost exactly. Due to this fact the large range of ROIs extracted by the Browserbite is not included into the classification process for filtering out false positive results. Secondly, the size of the dataset is small. Thus there can be suggested the way out which consists in increasing the dataset size by adding also that ROIs that do not meet the requirements described in the rules that are mentioned earlier.

However, the main problem is that it is hard to match DOM element to image. Alternative approach is to extract DOM elements first (for instance, in the way how it is done in Mogotest) and then extract exact corresponding image segment with their features. After such sequential extraction we would obtain perfect one-to-one mapping that can be used in the classification models. This is another way to continue the investigation of the problem how to decrease the amount of false positives, generated by the cross-browser compatibility testing tools.

6 Conclusion

Cross-browser incompatibilities are increasingly prevalent, and achieving consistent behavior across all major browsers has been a serious concern for Web application developers. Existing techniques for detecting and fixing these issues are, however, still immature, meaning that their results should also be checked manually as yet. One of the problems that cross-browser compatibility testing tools face with is a significant amount of false positives in their outcomes. False positive results are those minor and insignificant differences that cross-browser compatibility testing tools classify as real incompatibilities, however, they were not identified as so by humans. Due to the big amount of such results in tools' outcomes, for Web developers and testers it is harder to concentrate on real, important incompatibilities for fixing them. For this reason, filtering out false positive results from the cross-browser compatibility testing tools' outcomes is a relevant and important task.

6.1.1 Summary of findings

Current research has presented and evaluated a machine learning method to improve the accuracy of the output produced by means of single-page cross-browser compatibility testing tools that are using DOM-based and image-based techniques. We selected two machine learning methods – classification tree and neural network.

It was observed that incompatibilities detected at the bottom (at the right side) of the Web page are more likely to be true incompatibilities compared to those detected at the top (at the left side) of the page. This may be due to the fact that small displacements of elements at the top (left side) of the Web page ripple down (right) into more visible incompatibilities at the bottom (right side) of Web page. Also it was observed that size of the elements on the Web page that are compared plays a significant role in the comparison. While smaller elements almost always imply more precise comparison and thus cause less false positive results, the larger elements induce cross-browser compatibility testing tool to make mistakes.

For the binary classification both for image-based and DOM-based cross-browser compatibility testing techniques neural network provide a high level of accuracy in the context of the problem. For the image-based technique it achieves precision of 96% with recall of 89%, and for DOM-based technique the value of precision of 95% with recall of 91%. These results are significant improvement with respect to state-of-the-art techniques such as CrossCheck [27], which reportedly achieves the precision of 36% (64% of false positives) or its predecessor (WebDiff), which achieves an even lower precision.

We also suggested quaternary classification for potential incompatibilities identified by cross-browser testing tools. However, quaternary classification both for Browserbite and Mogotest has not showed significant improvement in accuracy. For Browserbite, F-scores for all four categories are low and approximately the same, and are in range from 58-60% for classification tree, but more "widespread" for neural network (from 58% from class C1 to 75% for category C4). The best and also the worst results for neural network are obtained in "extreme" C1 and C4 cases. Insufficient accuracy for C1 class can be explained by the comparable small amount of C1 class instances in the dataset (19% of all other samples in the dataset) due to the fact that such examples of potential incompatibilities detected by Browserbite are very infrequent. On the other side real incompatibilities are detected by Browserbite and amount of samples of class C4 is sufficient.

The fact that image features alone and DOM parameters alone provided a very high accuracy in binary case of classification, we attempted to combine these features and use their combina-

tion in the classification models. Such parameter mixing was conducted for Browserbite. However, we observed almost no improvement in the accuracy in this case in comparison to using image features or DOM alone, although 89% precision was reached which means that the combination of features is still working better than plain technique.

In light of the considerable improvements in accuracy, the neural network technique (for binary classification) has been productized and is going to be implemented in the Browserbite software-as-service tool.

6.1.2 Future work

Although the accuracy of cross-browser compatibility testing tools such as Browserbite and Mogotest was significantly increased using machine learning techniques for filtering out false positives from their outcomes, there is still a big branch of future work to do and investigate in terms of the current problem.

Although the approach has been framed in the contexts of the Browserbite (image-based) and Mogotest (DOM-based) tools, it is clear that the underlying principles are applicable to other image-based and/or DOM-based cross-browser testing techniques such as CrossCheck, which in fact uses both image- and DOM-based techniques all together. Validating the methodology in other such settings is a possible direction for future work.

Another avenue for future work is to further study the case of quaternary classification, for which the accuracy achieved by neural networks leaves room for the improvement. To this end, we need to consider further features and gather larger dataset. It is important to mention that the proportion of samples belonged to different categories has to be balanced, meaning that the number of samples in different classes has to be the same, so that the classification model wouldn't tilt to any of classes. A related direction is to evaluate the proposed technique with different types of stakeholders involved in Web application development process (e.g. Web designers versus testers versus developers). In this respect, one can hypothesize that classification models for designers would be different than those for developers, for example.

Another way for the future research is to combine features that are used in image-based cross-browser compatibility testing techniques with parameters of DOM elements. Although the achieved accuracy is in acceptable intervals (precision of 89% with poor recall of 71%), we need to consider larger dataset that includes various DOM elements. Also the automatic extraction parameters for DOM elements and in this way enlarging the dataset with additional features can also improve the accuracy of the results. However, the main problem in the current research is that it is hard to match DOM element to image. Alternative approach is to extract DOM elements first (for instance, in the way how it is done in Mogotest) and then extract exact corresponding image segments with their features. After such sequential extraction we would obtain perfect one-to-one mapping that can be used in the classification models. This is another way to continue the investigation of the problem how to decrease the amount of false positives, generated by the cross-browser compatibility testing tools.

Kokkuvõte

Masinõppel põhinev veebilehtede ühilduvusdiagnostika

Magistritöö (30 ECTS)

Nataliia Semenenko

Tulenevalt veebitehnoloogia kiirest arengust ja standardite rakendamise aeglusest, on tulemuseks probleemid veebilehtede ühilduvuses. Veebiarendajad peavad igapäevaselt tegelema ühilduvusprobleemiga, mis seisneb veebilehtede erinevas renderdamises üle laia valiku erinevate veebilehitsejate ja operatsioonisüsteemide.

Kuigi standarditele vastav veebileht tagab suurema veakindluse, siis on praktikas renderdusvead siiski väga sagedased ja vahelduvad vähetähtsatest kuni kriitiliste vigadeni, nagu näiteks mõne nupu puudumine veebilehel, mis on renderdatud konkreetse veebilehitseja ja operatsioonisüsteemi konfiguratsiooni koosluses.

Tuvastamaks renderdamisest tulenevaid ühilduvusvigu peavad arendajad visuaalselt kontrollima veebilehtede korrektsust üle valiku veebilehitseja ja operatsioonisüsteemi kombinatsioonide. Selline testimisviis on aeganõudev ja vigaderohke. Olemasolevad automaatsed testimislahendused kiirendavad testimise protsessi. Nendes lahendustes renderdatakse esmalt veebileht mitmel konfiguratsioonil, misjärel teostatakse kas dokumendiobjektide mudelil (DOM) või pilditöötlusel põhinev analüüs, tuvastamaks potentsiaalseid vigu.

Hetkel olemasolevate lahenduste puuduseks on ülitundlikkus. Selle tulemuseks on suur hulk vale-positiivseid tulemusi. Isegi väikseimad erinevused klassifitseeritakse vigadeks. Vale-positiivsete tulemuste vähendamine on raskendatud, tulenevalt sellest, et vea kriteerium ei ole alati jäigalt objektiivne.

Antud magistritöö pakub välja masinõppel põhinevad lahendused, parandamaks veebilehitsejate ühilduvustestide tulemusi, kas pilditöötluse (Browserbite) või DOM analüüsil (Mogotest) põhineva lahenduse näitel. Klassifikaatorite treenimiseks ja testimiseks valiti üle 140 veebilehe, mis renderdati 10 või 14 konfiguratioonil. Klassifitseerimisel kasutati kahte erinevat algoritmi: klassifikatsioonipuud ja tehisnärvivõrke. Ulatuslikud eksperimentaalsed katsed näitasid, et tehisnärvivõrgul põhinev klassifikaator on täpsete omadustega, nii pilditöötlusel kui ka DOM analüüsil põhinevate tulemuste klassifitseerimisel. Lisaks antakse töös ülevaade pilditöötluse ja DOM analüüsi meetodite kombineerimisest veebilehitsejate ühilduvuse testimisel.

Bibliography

[1] Xenocode Browser Sandbox. [Online]. http://spoon.net/browsers

[2] World Wide Web Consortsium: Document Object Model (DOM). [Online]. http://www.w3.org/DOM/

[3] V. Dallmeier, M. Burger, T. Orth, A. Zeller, "WebMate: Generating Test Cases for Web 2.0," in *Software Quality. Increasing Value in Software and Systems Development*, S. Biffl, D. Winkler ed.: Springer, 2013, pp. 55-69.

[4] V. Dallmeier, M. Burger, T. Orth, A. Zeller, "WebMate: A Tool for Testing Web 2.0 Applications," in *Proceedings of the Workshop on JavaScript Tools*, Beijing, China, 2012, pp. 11-15.

[5] S. R. Choudhary, H. Versee, A. Orso, "WEBDIFF: Automated Identification of Cross-browser Issues in Web Applications," in *Proceedings of the 2010 IEEE International Conference on Software Maintenance*, Timisoara, Romania, 2010, pp. 1–10.

[6] A.-L. Tamm, "Visual testing in different testing approaches," University of Tartu, Bachelor thesis 2012.

[7] The world's only cross-browser visual regression tool Mogotest. [Online]. http://mogotest.com

[8] The Web information company Alexa. [Online]. http://www.alexa.com

[9] T. Hastie, R. Tibshirani, J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed. NY: Springer, 2009.

[10] StatCounter Gglobal Stats.. [Online]. http://gs.statcounter.com

[11] P. Flach, "ROC Analysis," in *Encyclopedia of Machine Learning*, G. Webb C. Sammut, Ed.: Springer, 2011.

[12] Open source computer vision library OpenCV. [Online]. http://opencv.org

[13] Online tool NetRenderer. [Online]. http://netrenderer.com/

[14] A. Blum, *Neural Networks in C++: An ObjectOriented Framework for Building Connectionist*. NY: John Wiley & Sons, 1992.

[15] G. P. Zhang, "Neural Networks for Classification: A Survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 30, no. 4, pp. 451-462, 2000.

[16] NetMechanic, Browser compatibility testing with Browser Photo. [Online]. http://www.netmechanic.com/products/browser-index.shtml

[17] NetMarketShare, Desktop Browser Version Market Share. [Online]. http://www.netmarketshare.com/

[18] Matlab online help. [Online]. http://www.mathworks.se/help/matlab

[19] B. McLaughlin, *Java and XML, Solutions to Real-World Problems*.: O'Reilly Media, 2001.

[20] A. Mesbah, A. van Deursen, "Invariant-based automatic testing of AJAX user interfaces," in *Proceedings of the 31st International Conference on Software Engineering, ICSE '09*, Vancouver, Canada, 2009, pp. 210–220.

[21] J. Heaton, *Introduction to Neural Networks for Java*, 2nd ed.: Heaton Research, 2005.

[22] P.-N. Tan, M. Steinbach, V. Kumar, *Introduction to Data Mining*, 1st ed. Boston: Addison-Wesley, 2005.

[23] IECapt, Command-line utility to capture Internet Explorer's rendering of a web page. [Online]. http://iecapt.sourceforge.net/

[24] D. M. W. Powers, "Evaluation: from Precision, Recall and F-measure to ROC, Informedness, Markedness, and Correlation," *Journal of Machine Learning Technologies*, vol. 2, no. 1, pp. 37-63, 2011.

[25] R. L. Ebel, "Estimation of the reliability of ratings," *Psychometrica*, vol. 16, no. 4, pp. 407-424, 1951.

[26] M. J. Berry, G. S. Linoff, *Data Mining Techniques: For Marketing, Sales, and Customer Relationship Management*.: John Wiley & Sons, 2004.

[27] S. R. Choudhary, M. R. Prasad, A. Orso, "CROSSCHECK: Combining Crawling and Differencing To Better Detect Cross-browser Incompatibilities in Web Applications," in *Proceedings of the 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST)*, Montreal, Canada, 2012, pp. 171–180.

[28] Cross-browser compatibility testing tool Browserbite. [Online]. http://www.browserbite.com

[29] Cross browser compatibility testing tool BrowserCam. [Online]. http://www.browsercam.com

[30] L. G. Shapiro, G. C. Stockman, *Computer Vision*. New Jersey: Prentice Hall, 2001.

[31] L. Breiman, J. Friedman, C. J. Stone, R. A. Olshen, *Classification and Regression Trees*, 1st ed.: Chapman and Hall/CRC, 1984.

[32] BrowserShots. [Online]. http://browsershots.org

[33] A. Mesbah, M. R. Prasad, "Automated Cross-Browser Compatibility Testing," in *Proceedings of the 33rd International Conference on Software Engineering, ICSE'11*, Honolulu, HI, USA, 2011, pp. 561–570.

[34] Automated browser compatibility testing tool Browsera. [Online]. http://www.browsera.com

Non-exclusive licence to reproduce thesis and make thesis public

I, Nataliia Semenenko
(date of birth: 17.08.1990),

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

1.2. make available to the public via the university's web environment, including via the DSpace digital archives, as of 17.05.2013 until expiry of the term of validity of the copyright,

Accurate diagnosis of cross-browser compatibility issues via machine learning
(title of thesis)

supervised by Marlon Dumas, Tõnis Saar,

2. I am aware of the fact that the author retains these rights.

3. This is to certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu/Tallinn/Narva/Pärnu/Viljandi, **17.05.2013**