Automatic Web Page Robustness Grading
E. Gradjanin*, I. Prazina* and V. Okanovic*
* Faculty of Electrical Engineering, University of Sarajevo, Bosnia and Herzegovina
egradanin1@etf.unsa.ba, iprazina1@etf.unsa.ba, vokanovic@etf.unsa.ba

Abstract—This paper represents a solution to the problem of automatization of a web page robustness score grading. Robustness of a web page is best defined as a property of a specific web page to keep its layout and style of elements after applying different modifications. The rapid development of web pages has enabled a quick creation of numerous web pages, but the question is what is the quality of those web pages in terms of robustness. Automatic grading enables a relatively fast way of creating a metric in terms of the score that specific web pages get after being tested for the level of robustness. The research framework consists of different technologies and concepts that have been used during the implementation of a practical solution. The paper describes data structures that have been used to represent web pages as well as the machine learning methods such as neural networks, used to calculate the robustness score.

Index Terms—Neural networks, Robustness, DOM, KNN algorithm, Machine Learning, Trees

I. INTRODUCTION

Web page robustness can be an important factor in the success of a web page. If a web page layout is easily broken when new content is added or a component of the page is changed, then the maintenance of the page will be expensive and hard. Assessing the robustness can be an equally challenging problem. A web developer needs to have the experience to know how to effectively create a web page so it is easy to maintain and predict potential changes in the future. A web page that is rigid and hard to change will become obsolete quickly.

In this paper, one solution for automatic robustness of a web page is created. This solution is created with Python, because of its good support for different programming data structures, especially matrix-based data structures. It was a good choice for the implementation. A review of results is presented in the result interpretation section. This paper is structured as follows. First, the related work is analyzed, after that a description of used tools and concepts is given. The last two sections give a detailed description of algorithm implementation details and interpretation of results.

A. Related Work

Several related papers have served as a motivation for implementing building blocks of the algorithm. In [1] authors use hidden Markov model on top of potentials derived from DOM tree features using convolutional neural networks. Using ideas from this paper, a representation of web page in a tree form generated from the Document Object Model (DOM) has been derived. The core of the algorithm is comparing original web page and pages generated from a set of transformations. Comparing comes down to deriving a distance method which quantifies the differences between the original page and transformed pages. In [2,3,4,5,6] authors present different ways of comparing two web pages including visual, structural and content similarity between pages. In [9] author designed an algorithm that translates web page layouts into trees, using them to calculate tree distance that can be used as similarity metric. Authors [11] propose a theoretically guaranteed linear-time kernel computation algorithm. The concepts from this paper have been used for solving bottleneck for common computation when manipulating with tree data structure. In [12] authors present a system that uses neural representations to separate and recombine content and style of arbitrary images, providing a neural algorithm for the creation of artistic images. Values from different layers of neural network have been used to calculate the similarity between layouts and style of two web pages. Fig. 1 shows how the neural network can be used to generate art.


II. RESEARCH FRAMEWORK

A. Tools

Python programming language was used for implementation of the building blocks of the algorithm. Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built-in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components. Selenium is an umbrella project for a range of tools and libraries that enable and support the automation of web browsers. Selenium has been used for taking screenshots of web pages and for transforming web pages using a combination of HTML changes via Javascript and changing the dimensions of the browser which opens
the page. The lxml XML toolkit is a Pythonic binding for the C libraries libxml2 and libxslt. It is unique in that it combines the speed and XML feature completeness of these libraries with the simplicity of a native Python API, mostly compatible but superior to the well-known ElementTree API. Parsel is a BSD-licensed Python library for extraction and removal of data from HTML and XML by using XPath and CSS selectors, optionally combined with regular expressions. The combination of these two libraries with another Python package called BeautifulSoup makes HTML manipulation easy and straightforward. NetworkX is a Python package for creation, manipulation, and study of the structure, dynamics, and functions of complex networks. These packages were used to manipulate trees generated from the web page DOM. Keras has been used for Machine Learning parts of the algorithm. Keras follows best practices for reducing cognitive load: it offers consistent simple APIs, minimizes the number of user actions required for common use cases, and provides clear actionable error messages. Keras also offers out-of-box models like VGG16 neural network that has been used to extract layout and style features.

B. Concepts

When you write an HTML document, you put the HTML content into another HTML content. By this procedure you set up a hierarchy that can be represented as a tree. Often this hierarchical or encapsulated system is visually indicated by the identification of the HTML document. When loading HTML documents browser breaks and parses this hierarchy to create a tree of nodes which is called DOM. The Jaccard similarity index (sometimes called the Jaccard similarity coefficient) compares members for two sets to see which members are shared and which are distinct. It’s a measure of similarity for the two sets of data, with a range from 0% to 100%. Jaccard’s similarity is successfully applied as a method of finding textually similar documents in a large corpus. A tree is a nonlinear data structure, compared to arrays, linked lists, stacks and queues which are linear data structures. A tree can be empty with no nodes or a tree is a structure consisting of one node called the root and zero or one or more subtrees. The tree has been used in the algorithm to represent web pages. They have been generated from web pages DOM. A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm that can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and able to differentiate one from the other. One example for CNN is VGG16 which is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford. The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. This neural network has been used along with transfer learning to extract layout and style features from the web pages. Fig. 2 shows the building blocks of this network.

C. Implementation

NetworkX and BeatifulSoup libraries have been used to make it easy to work with HTML source code, and its conversion and representation in tree form. Selenium library enabled the retrieval of the original page source code which Beautiful soup uses to find the "body" tag. This generated tree is passed as an argument to a function that visits each time from the current root of the tree to each leaf and builds a set that represents a collection of subpaths. The collection of subpaths is then used to calculate kernel distance between two trees, by calculating the scalar product. Identical subpaths increase the similarity between the two trees, while the different subpaths do not affect the similarity. In other words they don’t change the level of similarity.

A custom function for recording the current state of the web page as a screenshot has been used. Selenium library has been used for visiting the page with the given link in a headless browser and taking a screenshot. This custom function has also allowed us to send a Javascript code that will be executed in the browser after the page load. Also an optional config object can be set to modify the browser’s dimensions which allows us to check web page responsiveness. These two options are building blocks for creating page modification and testing page robustness. First, it is necessary to define what a modification represents. Every modification used in the algorithm is a “dictionary” in the Python program language, with two keys “modificationScript” and “deviceMetric”. The modification may be a manipulation of the DOM website using Javascript scripts defined as a "modificationScript" key value. The modification can also have a defined value for the second key "deviceMetric" to describe the dimensions such as width, height and resolution of the environment in which the page is loaded.

Since both the original page and transformed pages representations are defined, a distance between them can be calculated. Using the following libraries "difflib", "io", "parsel" and "lxml.html" structural and style similarity is calculated.

First, a web page is decomposed to a collection of HTML tags and the ratio of the structural similarities between the two pages is calculated by using the functions from the library "difflib". In other words, pages with a similar tag order will
have a high value of similarity score, while pages with a different order will have a relatively small similarity, and thus the low similarity score. The same way CSS classes are extracted from the source code of both pages and using Jaccard similarity the distance between those two collections is calculated. Each collection represents a unique set of class names that have been extracted from the original website code. Finally, combining these two similarity measures, a structural and style similarity between two pages is derived. The similarity measure is between 0 and 1, where values close to 0 can serve as an indicator that two pages are not structurally similar, while the score close to 1 indicates high similarity between two pages.

VGG16 neural network has been used to extract features from specific layers. Both screenshots, one for the original page and one for the transformed page after applying one of the transformations, are used as inputs to the neural network. Structural similarity between pages is calculated as the difference between activations in deeper layers, since deeper layers of the neural network represent properties at higher abstraction. The same way style similarity is derived using a combination of several shallow layers. Style similarity measures how similar the pages are in terms of visual appearance. In combination with an already defined measure that shows the structural similarity of the two images, style difference represents a good indicator for analyzing website changes.

The last metric calculates the horizontal scroll ratio between the original and transformed page. Combining these four metrics (source sequence similarity, structural, style and horizontal scroll similarity) the initial dataset has been created. For each page ten different transformations have been applied. For each transformed page all four metrics have been calculated by comparing it to the original page. Combining these four metrics a total score for robustness is derived. The dataset is split into three subsets train, test, validation with corresponding sizes 60%, 20% and 20%.

To get a robustness score for the new page, first, the page is transformed to a tree representation. Then KNN algorithm is used to find the closest neighbors in the dataset using the kernel distance. For each page in the dataset a tree representation has been stored in memory to speed up the process. After a set of k closest neighbors are found an average robustness score for the new page is calculated.

![Fig. 3. Web page with high robustness score (36.8/40)](image)

D. Result Interpretation

The final result of the algorithm represents a function that receives an arbitrary page URL and calculates a measure of robustness for the page located at a given URL using the KNN algorithm. When building a machine model, RMSE metrics were used for the selection of hyperparameters, as well as for measuring performance on the holdout set. RMSD (root-mean-square deviation) or RMSE (root-mean-square error) is a commonly used measure to calculate the difference between the prediction model and actual observation values. RMSE represents the square root of the mean residual value, that is, the difference between actual value and prediction. To select hyperparameters during the construction of the machine learning model, a holdout set was used to measure RMSE, and a selection was made for that value of the hyperparameter for which the RMSE is the smallest. As an improvement of this method, on this relatively small dataset, the method of repeated cross-validation could be used in the future. Also, to get better results, a given data set could be expanded with more web pages. The function that calculates the distance for trees generated from the source code could serve to do a certain type of clustering. The measure of robustness is a linear combination of several metrics that are described in the previous section, and the scope of this measure ranges between 0 and 40. Pages that have a "score" close to 0 represent low-grade page robustness, while pages with a size close to 40 represent pages with a high degree of robustness, pages that retain the layout and style of the elements even after the application of modifications. Fig. 3 shows a screenshot of the page with a high score for robustness. The page achieves a total score of 36.8 out of a maximum score of 40. Fig. 4 shows a screenshot of the page with a low score for robustness. The page achieves a total score of 10.3 out of maximum score 40. RMSE on a set that was not used during the construction and validation of hyperparameters is 2.5.
III. CONCLUSION

The focus of this paper was building a model for the automatization of measuring web page robustness. As a final result of this project, after a successful construction of the dataset and machine learning model, a function that measures the degree (level) of the robustness of a randomly chosen web page is obtained. This has, thus, enabled the user of the algorithm to relatively quickly get the measure of the robustness for the particular web page by simply forwarding the URL by command line, and use the metrics as an input to another system or machine learning model. As a secondary result during implementation, the distance between web pages, precisely between trees generated from the original web page code, has been defined. This distance or similarity measure can be used for clustering web pages by their similarity; for different commercial use, as well as automatized extraction (scraping) of data from similar web pages using different methods for different page types. One of the improvements that can be introduced during machine learning model building is the implementation of the repeating cross-validation method. One of many uses of the robustness measure could be in building a reinforcement learning model, where this measure could be used as a reward/punishment function. That way, the model with an option of choosing between different alternatives of the existing web page presented as modifications (Javascript code) can be used for building web pages with a high degree of robustness.

REFERENCES

[11] Daisuke Kimura Hisashi Kashima, "Fast Computation of Subpath Kernel for Trees", Graduate School of Information Science and Technology, The University of Tokyo, Hongo 7-3-1, Bunkyo-ku, Tokyo, 113-8656, JAPAN
[12] Leon A. Gatys and Alexander S. Ecker and Matthias Bethge, "A Neural Algorithm of Artistic Style”