Bachelor thesis
Computer Science

Radboud University

Comparing Website-layouts using Edit Tree Distance

Author:
Spaendonck, P.H.M. van
S4343123
p.h.m.spaendonck
@student.ru.nl

First supervisor/assessor:
Prof. dr. ir., Arjen P. de Vries
A.deVries@cs.ru.nl

Second assessor:
title, name
e-mail adress

June 27, 2016

Abstract

The rapidly increasing size of the internet demands an automated solution for retrieving data about websites. While we are already able to analyze websites on their contents, an automated solution for analyzing and comparing layouts does not yet exist.
Thus we have designed a program that translates website layouts into data trees, and uses those to calculate the edit-tree-distance between them, so that we are able to do layout-analysis.

Contents

Chapter 1

Introduction

The internet is a growing source of revenue and information. Being able to analyze this collection of data is very useful. Most website analysis is done by looking at the content of a website, but not at the website layout itself. However being able to analyze and compare website layouts might be more worthwhile then we think. We could for example remove redundant backups of websites by looking if their layout has changed or use clustering to group up different websites with similar designs.
To be able to use modern analysis tools, we have to be able to quantify distance between two elements. If we're able to calculate the distance between different websites we can see which websites are similar and which ones differ the most.
A simple solution would be retrieving metrics about the websites design, for example most used color, however the outcome is entirely dependent on the chosen metrics where chosen, and no clear guidelines on constructing such guidelines exist. Unfortunately, when quantifying distance over large and complex data elements, it becomes more difficult to find a purely objective metric. Because, while we'll usually still be able to say whether two websites differ, being able to say what website is more different becomes more of a subjective decision.
Because of this, we will be looking at adapting an already existing technique instead of inventing and testing a new one. In our research we will be using the edit tree distance, while it is not known if this objective approach of measuring the distance between layouts is consistent with human judgment, we will assume it is sufficiently close to be meaningful.
However to do this, we will have to find a way of translating websites into data trees, so that we can then use our edit-tree-distance to analyze our data.

1.1 Focus of this research

Thus our research will be focusing on the following two questions:

- Is it possible to gain useful information about websites, while only analyzing the website's layout?
- Is Edit-Tree-Distance a good proxy for comparing website layouts?

By answering these, we hope to be able to show that analyzing website-layouts is possible and worth the effort.

Chapter 2

Related Work

Hier zal ik kijken naar vergelijkbaar onderzoek.

2.1 Preliminary knowledge

Before we start diving deeper into our research, it might be useful to explain certain aspects of the information-domains we're using. A websites layout is defined by `div`'s. A `div` is basically a box in which content (like text or an image) can be displayed. A div can also contain another div which again can contain more content. This structure is very similar to a tree; every node (`div`) of the tree might have one or more leaves (pieces of content) attached to it or might have another node attached to itself. For this research we will convert websites into data-trees so we can calculate the edit distance between them.
The elements of the websites we will be studying, have style properties assigned to them, either using a style-sheet or using inline attributes. These rules explain how they should be displayed, and this is what we'll be using to calculate the attributes we want to be stored.

Chapter 3

Method

The first part of the final product is able to transform websites into data. While it currently does not take every css attribute in consideration when calculating the corresponding values, it does for most of them, and shows it is possible to turn website layouts into data elements.
The other part of the product is able to calculate distances between the generated elements, and is also able to generate averages between data elements, which allows us to use analysis tools such as clustering.
The total product shows us that it is also possible to analyze website on their layout alone. And while it could certainly be improved on certain aspects, as will be explained later in this chapter, it does give a working step-by-step process of converting website-layouts into data-elements and analyzing these.
It is worth noting that, during the development process, we've used the top 25 most-visited sites in the US to test our software. These websites have been chosen, because not only are they widely used, they also use varying design principles. For example, google.com uses mostly inline css declaration, while live.com uses almost no inlinecss and refers to an external domain for style declarations.

3.1 design

As explained above, the created piece of software is combination of two pieces.

3.1.1 the translator

The first piece is the part of the program that is responsible for turning websites into data elements. This is done by opening a virtual browser window using the Ghost.py library for python. The window that we've used for this research was $1920x1080$, however different resolutions can be used

when needed.
Once the window is generated a website can be loaded in. On every website we use two pieces of JavaScript code. The first, "pre.js" is used to make XMLHttpRequests to load in style-sheets from external domains, and turning them into style-nodes. This is done because it is no longer possible to use data from a different domain than the domain at which the JavaScript is executed.
When all external style-sheets have been converted "core.js", the second piece of JavaScript, is executed. First all the internal style-sheet rules are assigned to their corresponding elements, by making them inline. This is done so that we don't have to look through all the css-rules every-time we want to look up one of the style properties of an element. Once we've done this, we start recursively calculating the width and height of every element, and their placement relative to their parent element.
Thus for every element we keep the following information:

- the elements width
- the elements height
- the elements horizontal offset, relative to its parent
- the elements vertical offset, relative to its parent
- the children (and their values) belonging to this element

We also have an extra slot we use to save information that we can use for debugging. This value is however never used later.
Once calculated, the elements and their values are saved to our local database, so that they can be used (by the second piece of our program) for analysis.

3.1.2 the analyzer

The second part of the program can then be used to analyze the new data elements using our own adaption of the edit-tree distance metric. [1] When we look at an element in our tree, we can simplify it to having two properties, the four values (width, height, vertical offset and horizontal offset) and its possible children. The way we calculate the distance between two branches is by looking at these two properties. We can project the four values to two vectors, and then calculate the rational distance between the original vector and the vector of the other branch, this gives us the cost of substituting the original element with the other one.
Then we calculate the distance between all the children of our element and all of the children of the other element. We use the shortest distance found,

[1] The adaption of the Edit-Tree-Distance algorithm can be found in the "LooseEditComparer.java" class.

or use the cost of creating the specific child if it's smaller than the shortest distance. The distance between two branches is than the total of these two costs.
For calculating the average we do a similar thing. Since the four values are actually vectors we can calculate these averages via conventional mathematical means. For the children, we try to find the closest pairs again, and then calculate the averages of those pairs.

3.2 difficulties

During the design and creation process, we've stumbled on certain problems that are worth mentioning.
As stated before, one big problem with JavaScript is that it is impossible to retrieve data from an external domain. We solved this by using CORS to make the external data local. This is however only possible if the external domain has CORS enabled. Thus the problem could arise that it is no longer possible to correctly retrieve the css-information via the current method.
Another problem is auto generated values that are created when the 'auto' keyword is used. Since the specific implementation is browser specific, the exact result depends on the browser that is used. Since we're using a custom browser, these values are not always computed as we want. For example *'margin auto'* does not center elements in our browser, while it would do so in others. Because of this we don't use the *'Window.getComputedStyle()'* method, but we calculate the values ourselves.

Chapter 4

Discusion

Our current implementation is able to create an objective representation of the websites we are using. We have shown that it is possible to retrieve information from a website, while only looking at the layout. However our current implementation is very crude and could definitely be improved on certain aspects.

4.1 Evaluation

During the development-process, we have left out three websites: Twitter.com and Reddit.com These were left out because they were to big to be handled by our software. However if the html-parser were to be made more efficient, then these websites would also be able to be parsed.
Our current parser, also does not handle every css-attribute, however the current parser is able to retrieve the css-attributes, thus a future version would be able to handle these. This means that while our solution is incomplete, it is still a valid proof of concept, since it still shows us that there is information inside the websites layout.

4.1.1 Usefulness of retrieved information

We have shown that it is possible to retrieve information about website layouts. However we would like this information to be useful. To see whether this is the case we will take a look at different pieces of functionality provided by our program.

Parsed websites				
google.com	Facebook.com	Amazon.com	Wikipedia.org	Ebay.com
Netflix.com	Linkedin.com	Craigslist.org	Pinterest.com	Live.com
Imgur.com	Go.com	Bing.com	Chase.com	Instagram.com
Paypal.com	Tumblr.com	Diply.com	cnn.com	Espn.go.com
Msn.com				

Clustering

The program has the ability to cluster using a nearest-neighbor algorithm and a K-means algorithm.
When clusering using nearest neighbour we gain the following clusters:

- Cluster 0 Pinterest.com, Facebook.com and Paypal.com
- Cluster 1 Instagram.com, Live.com, google.com, Amazon.com, Netflix.com, Craigslist.org, Bing.com and Chase.com
- Cluster 2 cnn.com and Go.com
- Cluster 3 Msn.com , Linkedin.com , Wikipedia.org, Imgur.com, Espn.go.com, Tumblr.com and Ebay.com

Unfortunately the clusters do not seem to correlate with the purposes of the parsed websites; socialmedia and news-sites are divided over different clusters. With only online advertisement and platforms (Craigslist and Ebay) being in the same cluster. However what we do notice is correlation between the usage of headers and alignment of elements.

Finding the most similar tree

Our program is also capable of retrieving the site most similar to another site. When we do this, we notice the following ḿost genericśites (Sites with the most closest sites). With our current pool, there are 5 sites that have Google as their nearest neighbor and 4 sites (one of them being google) with Live.com as nearest neighbor. When we take a look at the designs of these two websites, we notice that these websites consist of mostly empty space and very few displayed content.
When we look at their neighbors (which are the other memebers of Cluster 1), we notice that except for Amazon.com, all of these share a similar grade of complexity. This possible property means that the less complex a website is, the more websites see it as their nearest neighbor.
A possible usage of this, would be creating a directed graph using a lot more websites, one edge coming from every node and going into their respective nearest neighbor. If the bigger graph maintains the same property, we can

see how complex a website is, be seeing how few edges go into the websiteś node.
Further research could then be conducted to whether this complexity correlates with the usability of a website.

Although our current implementation does not seam to yield valuable information through clustering, it does seem that there is useful information to be retrieved by looking at what websites differ the least. We should also be taken into account that due to our relatively small dataset, some correlations might not be apparent and some correlations that now exist, might not be there, once a larger set is used.

4.1.2 The value of Edit-Tree-Distance as a proxy

Edit-Tree-Distance seems to be a good basis for design distance metrics for websites. As we have shown, it is relatively easy to translate website-layouts into data-trees, because of their tree-like nature. However it does fall short at some aspects.
For example, the origal edit-tree-distance algorithm only works in exacts. This means that it can only say whether a node in the tree is different or equal. Because of this we have to come up with our own way of calculating the difference between different nodes, bringing us back to our original problem, but only smaller.
The algorithm is also a very complex distance metric. Because of this calculating the distance between different nodes, can be very time consuming.

However it does seem that this is the best basis we currently have, and thus the best proxy we can use.

4.2 Future work

An unfortunate issue of trying to analyze data-sets that have not been analyzed before, is the lack of a frame of reference for measuring its validity, because of this most of the validity comes from further usage of the methods, that we've designed.
One of the most prominent things of continuing on this research will be analyzing a bigger collection of websites (ca. 500). Due to time constraints we were only able to use a small set websites, however now that we've already laid the groundwork for translating websites, we should be able to use a larger set.
In our current research we have not looked at human interpretation. It might be interesting to use surveys to test the validity of our current implementation.

Appendix A

Appendix

The written code can be found at `https://github.com/VizuTheShaman/Parsie`