



---

# Generation of visually understandable decision trees from Weka decision tree outputs.

---

Final Report for CS39440 Major Project

*Author:* Jessica Rocton (jer26@aber.ac.uk)

*Supervisor:* Dr. Wayne Aubrey (waa2@aber.ac.uk)

8<sup>th</sup> May 2017

Version 1.0 (Final)

This report is submitted as partial fulfilment of a BSc degree in  
Computer Science (G401)



## **Declaration of originality**

I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the regulations on Unacceptable Academic Practice from the University's Academic Quality and Records Office (AQRO) and the relevant sections of the current Student Handbook of the Department of Computer Science.
- In submitting this work I understand and agree to abide by the University's regulations governing these issues.

Name: Jessica Rocton

Date: 8/5/2017

## **Consent to share this work**

By including my name below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name: Jessica Rocton

Date: 8/5/2017

## **Acknowledgements**

To the guys, who put up with my misery, pessimism and crying and still tried to make me laugh.

To Leon, whose patience I will forever be grateful for.

To my long suffering boyfriend who, on top of the misery, pessimism and crying also weathered the storm of my frustration and a good number of tantrums.

To Stacey, for taking the time to proof read and grammar check this mammoth, even though she probably had no idea what I was talking about most of the time.

And finally, to my tutors, who never gave up on me, even when I wanted to give up on myself.

## Abstract

The web application created for the purpose of this project, allows users to upload Weka J48 classification decision tree text outputs and generates a decision tree diagram based on the data they have submitted. Users may then download the visualisation as a png or a jpeg file. As a web application, it was created using a combination of HTML5, CSS3, JavaScript and JQuery. It is designed to be cross-platform and to run on most common web browsers.

The project was successful in creating a web application that could successfully read in a Weka decision tree text output and generate a tree based on the data within this file. It was also possible to download the tree as a png or jpeg, however, these functionalities were only fully successful in the most recent versions of Google Chrome and Firefox. The project was unsuccessful at being able to read in the Weka text outputs in the same format as Weka provides them in. In order for this to work the file had to be reformatted using a 3<sup>rd</sup> party application. Additionally, while it is possible to generate a visualisation based on the data in a file, this only works for one file currently as it was not possible to find a way to generate the trees without hard coding the elements that would populate each node. Further work is still required before this project could be used in the capacity envisaged.

# Contents

Table of Figures.....	6
1. Background, Analysis & Process .....	8
1.1. Background .....	8
1.1.1 Project Overview .....	8
1.1.2 Technologies Used .....	12
1.2. Analysis.....	14
1.2.1 A comparison between a website and JavaScript command line tool .....	14
1.2.2 A comparison of text and XML formats.....	14
1.2.3 Learning Undertaken .....	16
1.2.4 Functional Requirements .....	17
1.3. Process .....	17
1.3.1 Development Methodology .....	17
2. Design .....	19
2.1 User Interface Design .....	19
2.2 Decision Tree Design .....	20
3. Implementation .....	21
3.1 Create the User Interface .....	21
3.2 Read in the text file.....	21
3.3 Manipulate the text file and print to screen.....	22
3.4 Implementation of Chosen Library .....	23
3.5 Data reformatting .....	24
3.6 Download Function.....	28
4. Testing.....	28
4.1 CodePen .....	29
4.2 Incorrect File Type.....	29
4.3 Browser compatibility .....	29
4.3.1 Google Chrome, version 57.0.2987.133.....	30
4.3.2 Microsoft Edge (ME), version 38.14393.1066.0 .....	31
4.3.4 Internet Explorer (IE), version 11.1066.14393.0. ....	32
4.3.5 Mozilla Firefox version 53.0 .....	32
4.3.6 Safari version 10.10 "Yosemite" .....	33
5. Evaluation .....	35
5.1 Were the requirements correctly identified? .....	35
5.2 Were the design decisions correct?.....	35

5.3 Could a more suitable set of tools have been chosen? .....	36
5.4 How well did the software meet the needs of those who were expecting to use it? .....	36
5.5 How well were any other project aims achieved? .....	37
5.6 If you were starting again, what would you do differently? .....	37
5.7 Future work.....	37
<b>6. Appendices .....</b>	<b>39</b>
A. Third-Party Code and Libraries .....	39
B. Ethics Submission .....	40
C. Code Samples .....	42
D. Testing Table .....	43
Annotated Bibliography .....	47

# Table of Figures

Figure 1: An example of part of a Weka J48 decision tree output, copied from Weka and viewed in Notepad++ .....	8
Figure 2: A screenshot example of a visualised J48 decision tree generated by Weka.....	9
Figure 3: An example of the use of a phylogenetic tree visualisation tool available from BioJS .....	10
Figure 4: Shows a Weka J48 decision tree output generated using example data provided by Weka displayed in a dot file format which is then used by the software application Graphviz, to generate a graphical representation of the data .....	11
Figure 5: Image shows the layout of CodePen along with some features that it offers .....	13
Figure 6: This shows part of the output of the web application used to try and convert a Weka decision tree output into XML .....	15
Figure 7: This figure shows the output that was provided by the WekaText2XML program when run with the test set of Weka decision tree data.....	16
Figure 8: This image shows a screenshot of the Trello dashboard (an online task manager) being used during an early phase of the project. Tasks can be created and are represented cards which can then be moved from one column to another as they are completed. ....	18
Figure 9: This image shows the landing page of the site and will be the initial point of contact between the user and the program.....	19
Figure 10: This figure shows the different options for tree structures that were available with the tree generation library, Treant.js.....	20
Figure 11: Depicting the output of the program once the user has successfully selected a file of the correct format and submitted it. ....	20
Figure 12: Screenshot displaying the error message the user will receive if they attempt to upload an incorrect file type.....	22
Figure 13: This figure shows the first part of the text output provided by Weka, displayed in Notepad++ .....	25
Figure 14: This shows the second part of the text output provided by Weka, displayed in Notepad++ .....	26
Figure 15: The output from the use of ParseJ48.py which reformatted a Weka J48 output. This was opened in Visual Studio Code and Beautify was applied to the file to improve the clarity of information being displayed. ....	27
Figure 16: The result of downloading the file from the webpage by selecting the download as a jpeg file button, viewed in Microsoft Photos (version 17.313.10010.0) .....	28
Figure 17: This shows the result when a file type that contains plain text but is not a text file, such as a CSS file, is uploaded to the form. ....	29
Figure 18: Shows the result of reading in a Weka decision tree test file being logged to the console. This was displayed by using the inspect function in Google Chrome.....	30
Figure 19: Homepage view in Microsoft Edge .....	31
Figure 20: Result generated by running the program and submitting a file using Microsoft Edge.....	31
Figure 21: Homepage view when opened in Mozilla Firefox.....	32
Figure 22: Result generated by submitting a file and generating a tree from the content, displayed in Mozilla Firefox. The bottom of the image also shows the array of the file that was read in being printed to the console.....	33
Figure 23: Homepage view in Apple Safari .....	34
Figure 24: Result generated by submitting a file to the program and generating a tree from the content, displayed in Apple Safari. ....	34

Figure 25: This figure shows the original site concept when it was generated using Mega Boilerplate .....	39
Figure 26: Showing the Jade (now known as Pug) that was used to code the page as well as the requests being made to the server upon opening the page. ....	40
Figure 27: The code implemented to generate the tree from submitted data. This shows the creation of each node, the assignment of the parent node, assignment of indexes to input the information that would be displayed on each node and the initialisation of the library. ....	42
Figure 28: Implementation of the code used to enable the tree output to be downloaded as a jpeg and png. The functions are passed the element that stores the tree and converts it to a blob/dataURL .....	42



# 1. Background, Analysis & Process

This section outlines the background of the project and analysis of the problem. Included, are justifications for the selected solution, technologies used and initial research that was carried out.

## 1.1. Background

### 1.1.1 Project Overview

Weka is a common data mining suit and data processing tool used in machine learning and a number of other scientific fields [1]. One aspect of the program allows users to generate decision trees as a way of visualising data using the J48 classification [2]. However, the decision tree outputs provided by Weka are outputted as plain text, as seen in Figure 1. This makes them visually difficult to interpret and unsuitable for use in academic publications.

```
1  === Run information ===
2
3  Scheme:      weka.classifiers.trees.J48 -C 0.25 -M 42
4  Relation:    Cleaned_Mouse
5  Instances:   8411
6  Attributes:  18788
7               [list of attributes omitted]
8  Test mode:   10-fold cross-validation
9
10 === Classifier model (full training set) ===
11
12 J48 pruned tree
13 -----
14
15 GO:0001707 = 1: lethal (85.0/3.0)
16 GO:0001707 = 0
17 |   GO:0048663 = 1: lethal (83.0/3.0)
18 |   GO:0048663 = 0
19 | |   GO:0035148 = 1: lethal (248.0/26.0)
20 | |   GO:0035148 = 0
21 | | |   GO:0035282 = 1: lethal (66.0/6.0)
22 | | |   GO:0035282 = 0
23 | | | |   GO:0045687 = 1: lethal (56.0/6.0)
24 | | | |   GO:0045687 = 0
25 | | | | |   GO:0007369 = 1: lethal (68.0/8.0)
26 | | | | |   GO:0007369 = 0
27 | | | | |   GO:0048514 = 1: lethal (132.0/12.0)
28 | | | | |   GO:0048514 = 0
29 | | | | |   GO:0008589 = 1: lethal (77.0/9.0)
30 | | | | |   GO:0008589 = 0
31 | | | | |   GO:0042476 = 1: lethal (63.0/9.0)
32 | | | | |   GO:0042476 = 0
33 | | | | |   GO:0007568 = 1: lethal (138.0/24.0)
34 | | | | |   GO:0007568 = 0
35 | | | | |   GO:0048568 = 1: lethal (142.0/20.0)
36 | | | | |   GO:0048568 = 0
37 | | | | |   GO:0009790 = 1: lethal (380.0/60.0)
38 | | | | |   GO:0009790 = 0
39 | | | | |   GO:0005643 = 1: lethal (44.0/8.0)
40 | | | | |   GO:0005643 = 0
41 | | | | |   GO:0030900 = 1: lethal (59.0/11.0)
```

Figure 1: An example of part of a Weka J48 decision tree output, copied from Weka and viewed in Notepad++

A decision tree is a diagrammatic tool used to inspect the various outcomes of a decision [3]. Currently, there are very few existing tools that make it possible to convert the Weka

decision tree outputs into a visual tree which can then be exported as an image. Weka does make it possible to view the text output in a more traditional treelike structure, as can be seen in figure 2. However, this does not scale well with larger models and there is not currently a way of outputting this in a different format, such as an image. In order to obtain this image it was necessary to take a screenshot and crop the image from it. This would be unrealistic to do with a larger tree as key details could be lost in the attempt of capturing the whole tree.

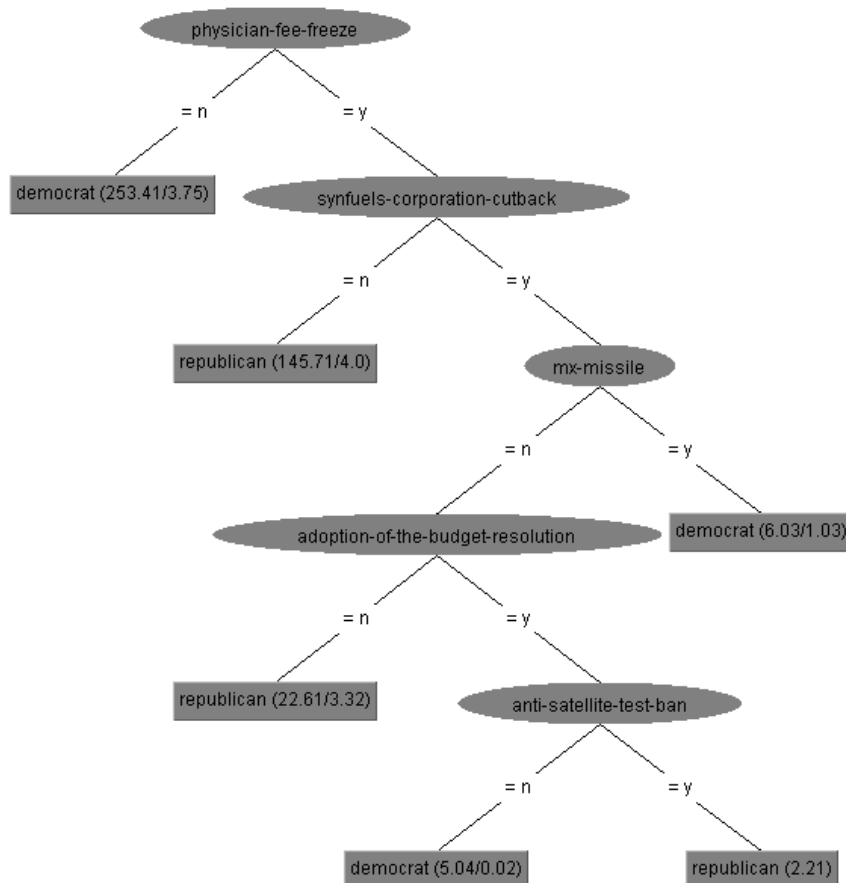


Figure 2: A screenshot example of a visualised J48 decision tree generated by Weka

There are libraries and tools for the visualisation data that could be used to generate decision trees. BioJS is a community based, open source project for creating and sharing JavaScript components purely for the visualisation of biological data [4]. The project came about to address one of the most common issues faced by the biological community which was the need for easily accessible and reusable bioinformatics visualisation tools [5]. One such example of a tree that is visualised based on data inputs was used to create a phylogenetic tree that displays the evolutionary relationships between organisms [6].

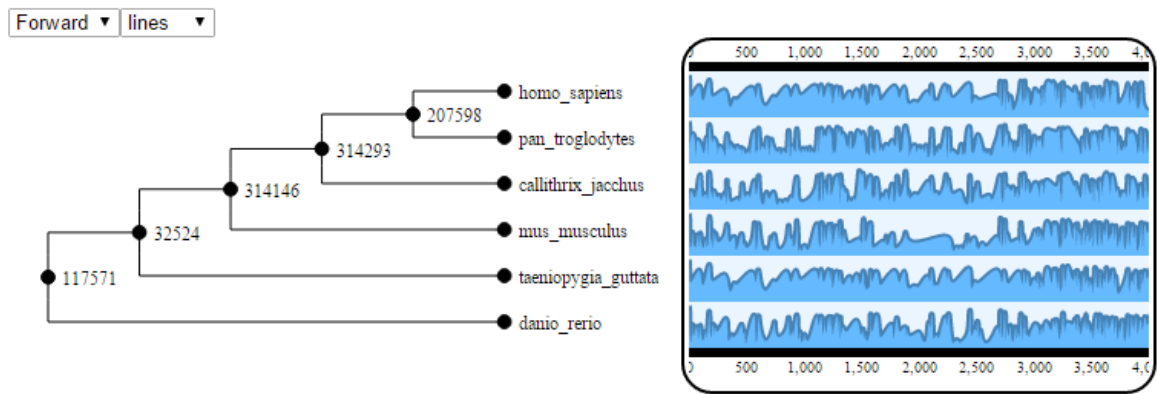


Figure 3: An example of the use of a phylogenetic tree visualisation tool available from BioJS

Figure 3 shows an example of the output of the phylogenetic tree visualisation tool, as can be seen, the tree structure that is visualised shares some similarities with traditional decision trees. However, the data used to generate the tree was completely different from the data provided by Weka's J48 classification. Additionally, in this case, the tree itself was not a decision tree, as a result, this did not make it an appropriate example to base this project on.

However, one application that does exist that makes it possible to do what this project aims to do, is Graphviz. This is an open source visualisation software that has the capability of transforming Weka J48 decision trees into an image output that can be saved by the user. This is done by converting the Weka output into a type dot file and then converting that dot file into another format [7, 8] such as a jpeg or a png file. Dot is a language that reads graph text files and writes them as drawings [9], this can be seen in figure 4. While this application is a good example of how this project can be successful it must be noted that it is a command line tool which limits user interaction with the final output. In addition, it is known to have problems running with larger files so there is a need for an alternative solution.

```

1 digraph J48Tree {
2 N0 [label="physician-fee-freeze" ]
3 N0->N1 [label==" n"]
4 N1 [label="democrat (253.41/3.75)" shape=box style=filled ]
5 N0->N2 [label==" y"]
6 N2 [label="synfuels-corporation-cutback" ]
7 N2->N3 [label==" n"]
8 N3 [label="republican (145.71/4.0)" shape=box style=filled ]
9 N2->N4 [label==" y"]
10 N4 [label="mx-missile" ]
11 N4->N5 [label==" n"]
12 N5 [label="adoption-of-the-budget-resolution" ]
13 N5->N6 [label==" n"]
14 N6 [label="republican (22.61/3.32)" shape=box style=filled ]
15 N5->N7 [label==" y"]
16 N7 [label="anti-satellite-test-ban" ]
17 N7->N8 [label==" n"]
18 N8 [label="democrat (5.04/0.02)" shape=box style=filled ]
19 N7->N9 [label==" y"]
20 N9 [label="republican (2.21)" shape=box style=filled ]
21 N4->N10 [label==" y"]
22 N10 [label="democrat (6.03/1.03)" shape=box style=filled ]
23 }
24
25

```

Figure 4: Shows a Weka J48 decision tree output generated using example data provided by Weka displayed in a dot file format which is then used by the software application Graphviz, to generate a graphical representation of the data

The aim of this project is to create a tool that allows users to input the decision tree file outputted by Weka in the form of a text file and generate a more visually understandable and pleasing decision tree. The first use of the project, if successful, would be by a current PhD student at Aberystwyth University who would include the newly formatted decision trees in his PhD thesis.

The solution that was selected was the creation of a website that would allow users to upload their Weka text outputs and map the data to a visual representation. Users would then have the option to download the newly generated tree as an image. To an extent, this project was successful in that a file could be read in and mapped to a tree structure based on the file content. Unfortunately, the tree could only be created by hard coding the indexes of the elements that were to be displayed from an array in which they were stored. Additionally, it was possible to download an image of the generated tree as both a png and a jpeg file however, this too had problems with the download not matching the produced image.

This document explores the process that was undertaken in order to provide a solution to the problems discussed. This includes the justification behind design choices, the process by which the software was implemented, the testing that was applied as well as a critical evaluation of the success of the overall project and the choices that were made.

### 1.1.2 Technologies Used

Before discussing the technologies used it is important to note that this application was developed using a Windows OS (Windows 10), as such, all software that was downloaded was Windows specific.

**Visual Studio Code (Version 1.11.2):** One decision that was necessary to make before starting the software development element of this project was the choice of IDE (Integrated Development Environment) which would be most appropriate to support development. Originally Sublime Text editor was considered for its simplicity and familiarity. However, after some research into various IDE's it was decided that Microsoft's open source Visual Studio Code (VS Code) [10] would be better suited to this particular project. VS Code offers a number of features that would make the coding process more efficient, this includes integrated debugging, direct access to Git via the editor and extensions that made the coding experience more efficient. A number of extensions were used in this project to make the coding process smoother, the first such extension was Beautify, [11] which was used to reformat the code to make it more readable and easy to understand. The other extension used was HTML Snippets [12] which was mainly used for automatically completing and adding closing tags.

**Mega Boilerplate:** One option that was explored early on in the project conception was the use of Mega Boilerplate to generate the website. A website would provide an easy means for users to interact with the application and would make the program usable without having to install software such as Python or Java in order to run it. Mega Boilerplate is an open source starter project generator [13]. It provides customisable set ups for web projects, this includes a basic website which can be selected to be static or which could use Node.js. If the Node.js option is selected then Express will be included as the web application framework to support this choice and the site will be locally hosted on port 3000. Additional frameworks can also be selected for styling and for use of JavaScript. There are also options to set up unit testing and a database if the application requires it and the tool offers the choice of structured and unstructured database setups in the form of MongoDB and MySQL. Depending on the options selected, the further choices you can access will vary in order to best support the needs of the application

This was considered initially as the website was not to be the main focus of this project and so, having much of the basic site pre generated would save time. However, one requirement of use was that any mark-up changes would have to be made using Jade [14], a templating language, which would then be compiled into HTML. This was as a result of Mega Boilerplate using Node.js to run server side JavaScript. This added additional complexities to the project and after further assessing the requirements it was decided that many of the features offered by Mega Boilerplate were not necessary to the project. One example of this was the provision of a database and server support however, as the program could be run on the client side, this was unnecessary. As such, it was decided that this was not appropriate for use in the scope of the project and that the site would be created manually. However, images and code examples of its trialled use can be found in the appendix of this document.

**CodePen:** CodePen is a virtual space which allows new web code or web applications to be tested and run securely [15], in the scope of this project it was used to demo code before implementing it into the main code base. It allowed for the input of HTML, CSS and JavaScript and was a convenient, easy way to manipulate and test code to identify potential

issues and solutions. This tool was incredibly simple to use and was helpful when testing the implementation of Application Programming Interfaces (APIs) or functions, as changes could be made and tested in quick succession. However, this was less helpful when trying to implement 3rd party libraries as it was not possible to integrate them with the tool, as a result they had to be directly added to the main code base and tested through the web browser.

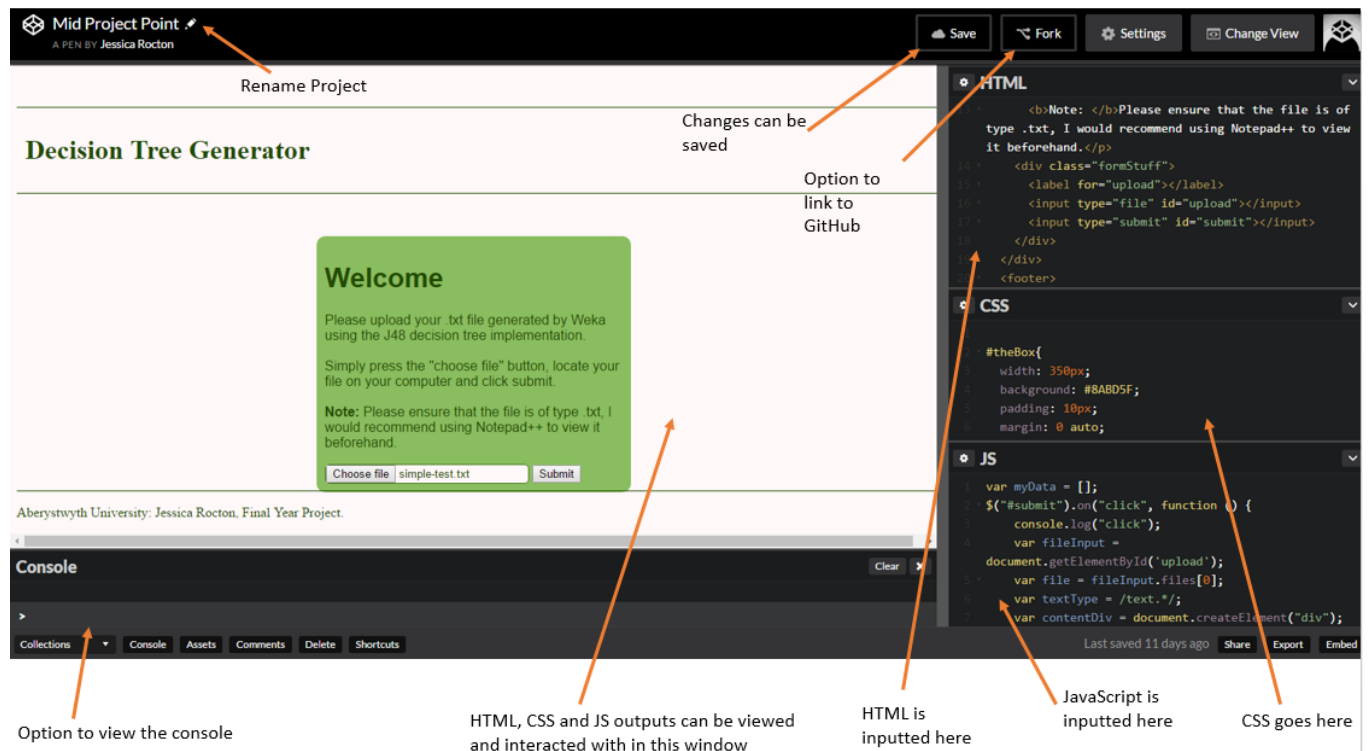


Figure 5: Image shows the layout of CodePen along with some features that it offers

**GitHub:** GitHub is an open source version control system that is based on the command line tool Git [16]. It was primarily selected as the method of version control for this project as it synchronized well with the chosen IDE and is a well-used platform for storing, sharing and managing code. Using GitHub would allow for the code and all resources associated with it to be made publically available in a repository. This would permit it to be reused and modified freely in the future by the coding community. In addition, this made it possible for the write up document to also be subject to version control which improves the reliability of both the software and documentation as either may be rolled back to a previous version if issues were found.

**Google Docs:** Google Docs is a real time collaborative writing tool that was selected to be used as a sharing platform for the write-up of this project. The project was to be written up in Microsoft Word and having a means by which a working document could be shared with the project supervisor allowed for ease of access on their part. This also provided the added benefit of quicker feedback owing to the user's ability to alter the document and make comments in real time. This solves the problem of accidentally editing or providing feedback on an out-of-date document (as there is only one working version) and prevents the need to merge changes from two documents. While the concept itself is incredibly effective it was more difficult to format the document as there were less formatting tools available than in Microsoft Word.

LaTeX was another option for document generation that was considered. LaTeX is a document preparation tool that is commonly used in the creation of large/medium technical documents [17]. This tool also has a real time, collaborative editor known as ShareLaTeX [18], which could have been used instead of Google Docs if this option had been selected.

Microsoft Word was ultimately selected for the final formatting of the document due to its familiarity and flexibility for formatting and typeset. Additionally, it was considered that the addition of learning how to use LaTeX, while useful, could be potentially time consuming and would detract from the main technical work.

## 1.2. Analysis

This section outlines the decisions that were made before the coding process took place. These decisions enabled better planning and understanding of the tools that would be needed going forward with the project.

### 1.2.1 A comparison between a website and JavaScript command line tool

An early decision that had to be made upon analysis of the problem was the format that the application would take. One solution would have been to create a JavaScript programme that could be run from the command line, but would also require the user to have node.js installed. This would have taken the Weka file as a parameter when the script was run and then run a series of processes on the data to provide an output file. However, as mentioned previously, Graphviz already exists to fulfil this need so it was decided that a client side web app would be a better fit. Not only would it provide a better user experience and allow the user to interact with the output but users could be sure that they were satisfied with the output before deciding to commit to downloading it. In addition to this, as the app would be run on the user's client, it could be used without needing to access the internet. The hope is that additional functionality such as the ability to zoom in on aspects of the tree and rearrange the layout will be added to further enhance the user's experience. These are capabilities that would not have been as effective to implement if creating a command line tool.

### 1.2.2 A comparison of text and XML formats

One question that was raised during the course of this project was in relation to the file input type. As mentioned, Weka outputs produced in version 3.8 are presented as raw text and so the initial file format that will be handled will be a plain text file.

One option that was explored was whether converting the file to a different format would make it easier to process, one such format was XML. XML stands for eXtensible Markup Language and it is used to store and transfer data in a plain text format with the addition of custom descriptive tags that describe the data contained within the file [19]. While it is possible for Weka to output a file as XML, this can only be done when using the command line version of the tool. As many users prefer to use the Graphical User Interface (GUI), this option is not immediately available as an output type. In order to compensate for this, users could convert the text file to XML externally. A number of methods were explored when assessing the viability of this option. Firstly, an online Text to XML converter was trialled [20] however, the output from this was unhelpful as it simply enclosed each line in <p> tags, this can be seen in figure 6. Furthermore, the site sometimes would not work and would not read in all text files submitted for conversion. Often, it would only work if given just the tree section of a small file, this made it unreliable and not suitable for use in the context of the project.



```

<?xml version="1.0" encoding="UTF-8"?>
- <Content>
  <p>=== Run information ===</p>
  <p/>
  <p>Scheme: weka.classifiers.trees.J48 -C 0.25 -M 2</p>
  <p>Relation: vote</p>
  <p>Instances: 435</p>
  <p>Attributes: 17</p>
  <p>handicapped-infants</p>
  <p>water-project-cost-sharing</p>
  <p>adoption-of-the-budget-resolution</p>
  <p>physician-fee-freeze</p>
  <p>el-salvador-aid</p>
  <p>religious-groups-in-schools</p>
  <p>anti-satellite-test-ban</p>
  <p>aid-to-nicaraguan-contras</p>
  <p>mx-missile</p>
  <p>immigration</p>
  <p>synfuels-corporation-cutback</p>
  <p>education-spending</p>
  <p>superfund-right-to-sue</p>
  <p>crime</p>
  <p>duty-free-exports</p>
  <p>export-administration-act-south-africa</p>
  <p>Class</p>
  <p>Test mode: 2-fold cross-validation</p>
  <p/>
  <p>=== Classifier model (full training set) ===</p>
  <p/>
  <p>J48 pruned tree</p>
  <p>-----</p>
  <p/>
  <p>physician-fee-freeze = n: democrat (253.41/3.75)</p>
  <p>physician-fee-freeze = y</p>
  <p>| synfuels-corporation-cutback = n: republican (145.71/4.0)</p>
  <p>| synfuels-corporation-cutback = y</p>
  <p>| | mx-missile = n</p>
  <p>| | adoption-of-the-budget-resolution = n: republican (22.61/3.32)</p>
  <p>| | | adoption-of-the-budget-resolution = y</p>
  <p>| | | anti-satellite-test-ban = n: democrat (5.04/0.02)</p>
  <p>| | | anti-satellite-test-ban = y: republican (2.21)</p>
  <p>| | mx-missile = y: democrat (6.03/1.03)</p>
  <p/>

```

Figure 6: This shows part of the output of the web application used to try and convert a Weka decision tree output into XML

Later, an application called WekaText2Xml [21] was used with limited success. While the application did convert the Weka decision tree output to XML, it was first necessary to isolate just the tree section of the file as this program would not read in the file as a whole. This was then saved as a separate file and inputted into the program which then outputted it as a new file with the appropriate tags, as can be seen in figure 7.



```

<?xml version="1.0" encoding="UTF-8"?>
- <DecisionTree type="weka-vote-tree-only">
  - <Test value="n" operator="=" attribute="physician-fee-freeze">
    <Output info="(253.41/3.75)" decision="democrat"/>
  </Test>
  - <Test value="y" operator="=" attribute="physician-fee-freeze">
    - <Test value="n" operator="=" attribute="synfuels-corporation-cutback">
      <Output info="(145.71/4.0)" decision="republican"/>
    </Test>
    - <Test value="y" operator="=" attribute="synfuels-corporation-cutback">
      - <Test value="n" operator="=" attribute="mx-missile">
        - <Test value="n" operator="=" attribute="adoption-of-the-budget-resolution">
          <Output info="(22.61/3.32)" decision="republican"/>
        </Test>
        - <Test value="y" operator="=" attribute="adoption-of-the-budget-resolution">
          - <Test value="n" operator="=" attribute="anti-satellite-test-ban">
            <Output info="(5.04/0.02)" decision="democrat"/>
          </Test>
          - <Test value="y" operator="=" attribute="anti-satellite-test-ban">
            <Output info="(2.21)" decision="republican"/>
          </Test>
        </Test>
      </Test>
    </Test>
    - <Test value="y" operator="=" attribute="mx-missile">
      <Output info="(6.03/1.03)" decision="democrat"/>
    </Test>
  </Test>
</Test>
</DecisionTree>

```

Figure 7: This figure shows the output that was provided by the WekaText2XML program when run with the test set of Weka decision tree data.

While the application did fundamentally achieve what it needed to, it was time consuming and awkward to use which would make it less than ideal for users. The biggest problem with this solution was that the software was freeware not open source and so the code was not available for reuse. As a result, it could not be adapted to be used to internally convert a user's text file within the proposed website. Therefore, users would have to go through a number of tedious steps before even having a suitable format that the website could utilise. This would be a less than ideal solution and for this reason, it was decided that converting the file to XML (using this tool) should not be pursued as a part of this project.

### 1.2.3 Learning Undertaken

Before the technical aspects of this project could commence, a series of learning processes had to be undertaken in order to improve on current knowledge in the areas that would be used. As it had been decided that a web application was to be created, gaining knowledge on the use of web technologies was vital. It was decided that a combination of HTML5, Cascade Style Sheet (CSS3), JQuery and JavaScript would be used as these are the best known web languages and there are many tools available to support those who would wish to learn about them. The majority of HTML and CSS functions are supported by most modern browsers [22] which made them a sensible choice of technologies when considering browser compatibility.

The learning process included various tutorials on HTML, CSS, JQuery and JavaScript, these were largely undertaken via FreeCodeCamp.com [23] which provided me with the basic web skills I would initially need. I found this to be a better learning resource than others I have previously used such as Codecademy as the examples and tasks that were provided were more extensive, better explained and appropriate to the learning that was being undertaken.

### 1.2.4 Functional Requirements

Once the analysis of the problem had been carried out it was possible to create a list of functional requirements that the proposed solution would aim to deliver.

1. A functional website that would enable the user to upload their Weka J48 decision tree text outputs, process the input and output a visual representation of the data in an easy to understand format.
2. A way in which the user can then obtain this output in as an image.
3. Well documented and reusable code that can be made openly available to the coding community for future use.

## **1.3. Process**

Before the development of the project could commence, it was important to decide on which development methodology would be used to support the growth of the project, this was necessary to ensure that the software would be robust and of a good quality.

### 1.3.1 Development Methodology

There are many varied development methodologies available for use in the development of a project. From Agile methodologies such as eXtreme Programming (XP) [24] and Scrum [25] to plan driven methods such as Waterfall [26], the decision of which process to follow ultimately needs to be best suited to the type of project being undertaken.

For this project, it was decided that an agile approach would be taken as it allowed for greater flexibility than plan driven approaches, such as Waterfall. This was important as any changes in project direction could be implemented more easily without having to restructure documentation and design. A yearly report produced by VersionOne [27] has shown that the most common techniques employed by business who use Agile are:

- Iteration planning
- Daily stand-up
- Retrospectives
- Iteration reviews
- Short iterations

From these, short iterations, iteration planning and iteration reviews were used most frequently. While these techniques are largely associated with Scrum, it was unrealistic to implement all functions that are associated with this methodology due to the nature of this project. Short iterations were used for development, as this allowed the coding solution to be coherently broken down into smaller, more manageable tasks. In adopting this approach it allowed for some margin of error as it meant that if one attempted method of implementation was not working then this could be reviewed during the iteration review and a new method could be attempted without having to change the entire project to fit with the new decision.

Iteration reviews were done on a weekly basis with the project supervisor and were closely linked to the sprint retrospectives. During these meetings, current progress would be reported which improved project transparency so that realistic assessments of progress could be made. Additionally, any issues that were encountered were discussed, along with how they were resolved. In the cases where a particular problem persisted, decisions could be made as to whether a different approach should be taken to resolve it. A list of tasks for

the following iteration was compiled and agreed on during these meetings and then reviewed after the iteration was completed.

Planning is an important practice that can affect the success of a project. There are a number of tools to facilitate this process however, the tool that was found to be particularly useful for iteration planning was Trello [28]. This online project management tool allowed for the documenting and tracking of tasks. Trello was selected for use in this project as it is a free tool that could be readily accessed on a number of devices. I had used it in the past for previous projects and found it to be easy to use and also allowed for others to be added to a working board so that they could add or view tasks if they wished. Figure 8 shows a snapshot of the tool being used in the context of this project. It was both helpful and useful to be able to clearly see which tasks needed to be done and at what stage they were at as it allowed for better planning and prioritisation of work. This is particularly reminiscent of the wall chart concept used in Feature Driven Development (FDD) [29] and the Scrum Board used in Scrum, and was partially inspired by this. It does differ to an extent, as it does not expressly state due dates. Despite this, the general purpose remains the same in that it was a useful tool for tracking progress at a glance but also at a more in-depth level could be used in planning which tasks needed to be carried out at different stages in this project.

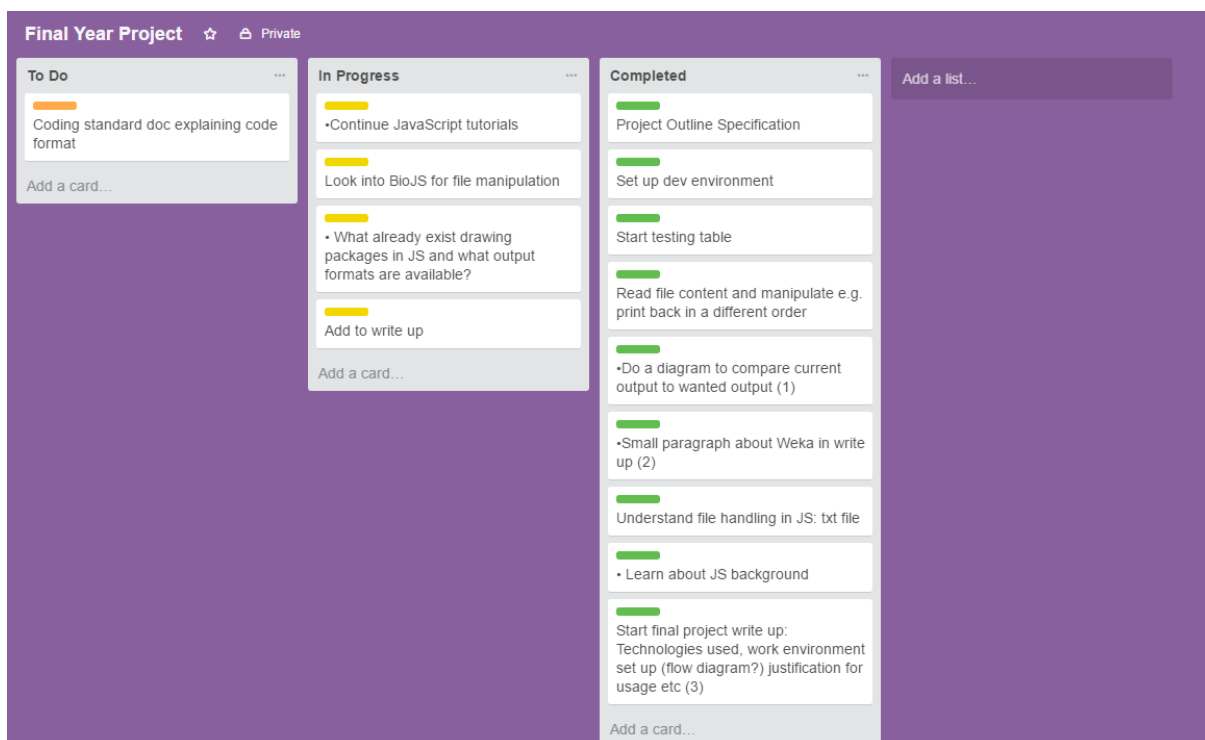


Figure 8: This image shows a screenshot of the Trello dashboard (an online task manager) being used during an early phase of the project. Tasks can be created and are represented cards which can then be moved from one column to another as they are completed.

The method employed throughout this project did not fall under any definitive agile methodology. Rather, techniques from a number of different practices were used in a way that best suited the project's needs. In some cases, certain Agile techniques such as pair programming and the daily stand-up that are associated with specific methods were not realistic for use in the context of this endeavour. As such, it would be incorrect to state that a single methodology was used as not all aspects of just one agile practice would have necessarily been appropriate.

## 2. Design

The discussions in this section relate to the design choices made and the justification behind these decisions. The main point of focus for design in this project was the user interface (UI) as this would be the user's way of interacting with the program.

### 2.1 User Interface Design

Figure 9 shows the main page of the site. The design is purposely simple, this is a tool that provides a means to an end and additional, unnecessary features could make it less user friendly in the long term. This is supported by the acronym YAGNI (You ain't gonna need it) which is commonly cited when implementing agile practices. This phrase emphasises that software should be kept simple and that presumptive features that could have value in the future might actually never be necessary and lead to a waste of time and resources during development [30]. The role of the site is purely to allow the user to select a file from their local file store, submit it, display an output generated by the file and provide a way that users can obtain the output. For this reason there was little need to make it more complex than necessary.

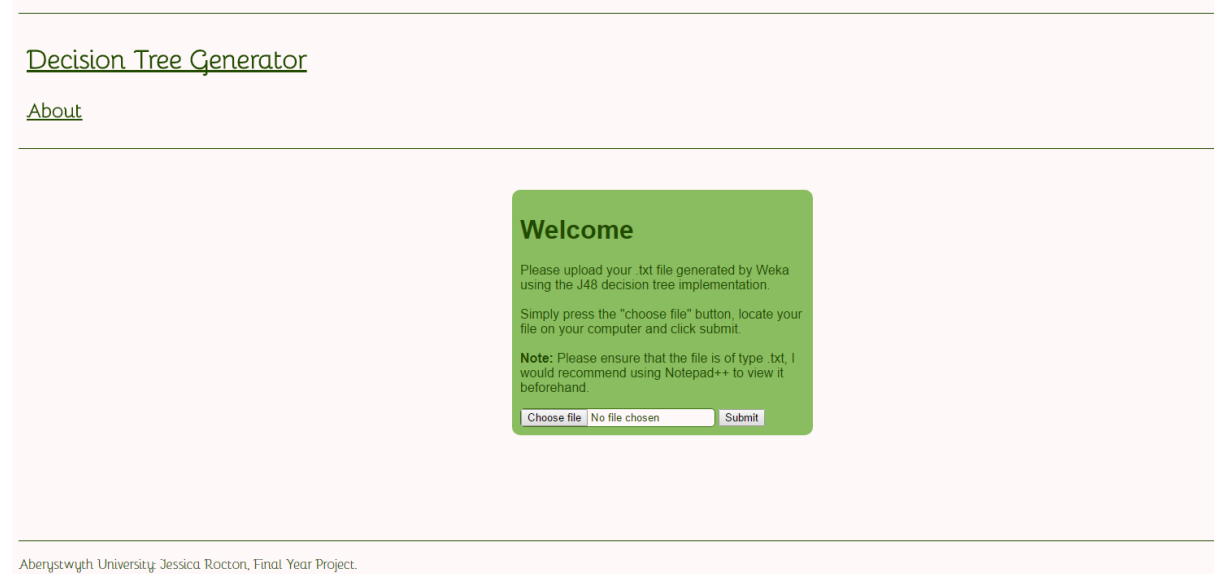


Figure 9: This image shows the landing page of the site and will be the initial point of contact between the user and the program

A number of different shades of green were used and were selected as it related to the theme of trees. A light background was chosen to contrast with the darker colours used to emphasis content, this would draw the eye to the more important parts of the page. While the choice of colours were thematically appropriate this could cause problems for certain users with colour blindness in which the colour green can be problematic and could affect the site usability. One solution to this that was considered, but not implemented due to time constraints, was to give the user an option to change the colour scheme by clicking a "colour-blind mode button". This would have used JQuery to apply a different style to the page once the button had been clicked.

The main font used is called Bellota-Light and was imported from an external source [31] by downloading the oft file and including the following code in the main style sheet. This was chosen as the curved nature of the font was reminiscent of vines, once again relating to the theme of trees.

```
@font-face {
  font-family: Bellota-Light;
  src: url("Bellota-Light.otf") format("opentype");
}
```

## 2.2 Decision Tree Design

The library that was used to generate the tree offered a number of different layouts and options based on the needs of the data that was to be visualised (see figure 10), however, none of these formats seemed appropriate for the data that would be inputted. There was an option to generate a simple tree, seen in figure 11, which was ultimately decided on as it suited the required layout best due to its clarity.

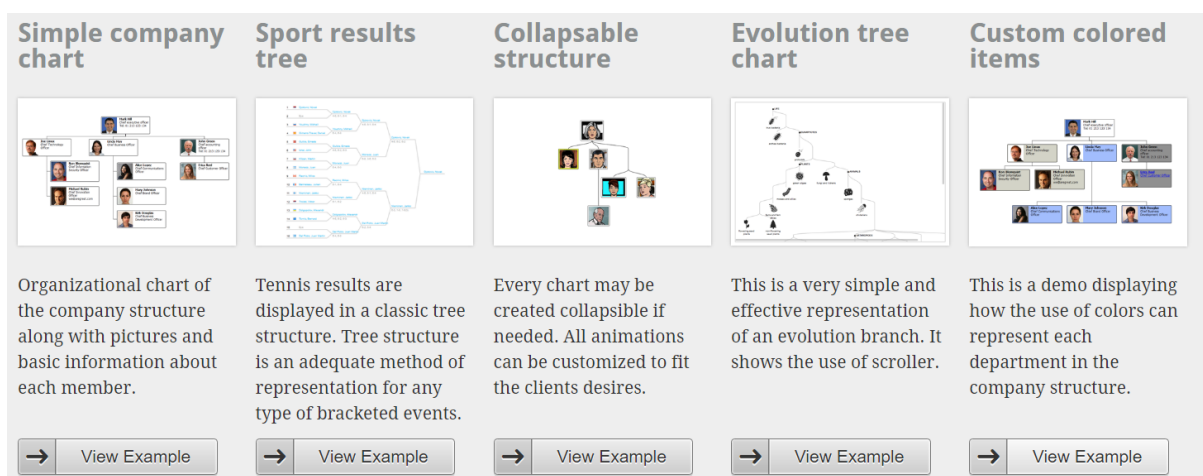
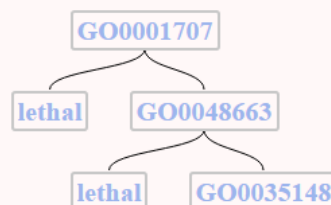


Figure 10: This figure shows the different options for tree structures that were available with the tree generation library, Treant.js

## Decision Tree Generator

### About



[Download as JPEG](#) [Download as PNG](#) [+](#) [-](#)

Aberystwyth University: Jessica Rocton, Final Year Project.

Figure 11: Depicting the output of the program once the user has successfully selected a file of the correct format and submitted it.

## 3. Implementation

The project was implemented incrementally through a series of coding sprints, as explained in the development methodology section of this report. These sprints were carried out to meet the following milestones, the success of which allowed for the continuation onto the next sprint.

### 3.1 Create the User Interface

After it was decided that the program was to be a web based application, the initial step was to create a basic UI that would permit users to interact easily with the program. This was done through the use of HTML5 and CSS and included the creation of a form that would allow users to browse for the file they wished to convert and select it to be processed by pressing a submit button. Once the user was able to select a file, the next step would be to read it in it so that the data within the file could be accessed as this is what the decision tree would be generated from in later sprints.

### 3.2 Read in the text file

JavaScript's FileReader API [32] was used in order to read in the text file to the program. This process can allow for the file to be read in in a number of different formats such as text, binary string, DataURL and as an array buffer. As the data that is being dealt with is a plain text file, it made sense to use the readAsText method which reads in the file as a plain text string.

The first step towards reading in the file was to create a JQuery event listener that would trigger the read function in the event that the submit button was clicked. It was necessary to determine where the file input was coming from by assigning the HTML element that was holding it to a variable known as fileInput. Another variable named file is then created that holds the value of the file that was selected. A further variable called textType is then fabricated that is given the value of the regular expression for the text file extension, this allows the program to check that the correct file type has been selected. If the file extension does not match then the user will receive an error message to the screen, as seen in figure 12, informing them of this. If the file extension does match, a new filereader object is created which runs the onload event handler which triggers when the file has been successfully read [33, 34]. From this point, the data is now able to be accessed and manipulated.

## Decision Tree Generator

### About

File not supported! Please Choose a .txt file

#### Welcome

Please upload your .txt file generated by Weka using the J48 decision tree implementation.

Simply press the "choose file" button, locate your file on your computer and click submit.

**Note:** Please ensure that the file is of type .txt, I would recommend using Notepad++ to view it beforehand.

Choose file Bellota-Light.otf

Submit

Aberystwyth University: Jessica Rocton, Final Year Project.

Figure 12: Screenshot displaying the error message the user will receive if they attempt to upload an incorrect file type

### 3.3 Manipulate the text file and print to screen

At this point, `console.log(reader.result)` was used to check that the file in its entirety had successfully been read in and could be printed to the console. Once this was confirmed, the next step was to attempt to manipulate the data. This was important as the essence of the problem that is trying to be solved is to access a file, extract the required information and then carry out a process on this information. Consequently, testing that basic manipulation could be carried out was an important step.

The initial file that was used to test the basic functionality of the program was a simple file that contained 3 words separated by a space character. It was decided that the most logical way to split the file was based on this character and then adding each individual word as a separate element to an array. Therefore, a new array was created and populated by taking the result from reader and using the returned array, from the split method to break up the file on the space character and then add each element to an array. At this point it was possible to start manipulating the file, this was tested by printing the text within the file back in a different order to which it was read, in this case, backwards. This was achieved by looping through the array starting with the last element and assigning it to a variable, this variable was then added to a new array. This was repeated until index 0 (the first item) was reached. The new array was then printed to ensure this process had been successful.

Up to this point, the data had been viewed only by logging it to the command line. This was not suitable for the end user, so it was necessary to be able to display this information on the webpage. This was done originally by creating a new node (element) and adding it to the existing HTML Document Object Model (DOM) [35]. The DOM is the standard by which HTML elements can be accessed and manipulated by programs and scripts [36]. Once the new node has been created, a text node must be made and appended to the node. After this, an already existing element in the DOM must be selected and the new node is then appended to it.



JQuery was then used to hide the web form that was previously visible to the user before submitting the file. In this way, the user would only then see the output of the file displayed on the page.

### 3.4 Implementation of Chosen Library

One consideration that had to be taken into account was the means by which the data, once extracted, would be visualised. An early option that was considered was to manually create and draw each node and branch using HTML canvas. This was discarded early on however upon realising the complexities that would accompany this method, especially when working with larger datasets that would require the generation of bigger trees. The decision was reached that a 3rd party library would be used as the quality of the tree was likely to be better and it would save time in the development process.

A number of libraries were considered before deciding on the one that was best suited to this project. Initially, D3 was investigated as a potential solution. D3 is “a JavaScript library for manipulating documents based on data” [37] and provided a wide range of examples for data visualisation. Despite this, very few possibilities were identified as being potentially applicable for this project. The first of such was a binary tree visualisation [38] however this was found to be unsuitable for a number of reasons. Firstly, this would not be appropriate for data that was non binary i.e. a tree with more than two decisions, secondly, it would not have displayed labels and while it looked visually appealing it would not have been suitable for cases where the data would cause the tree to become weighted on one side. Finally, while it is given as a D3 example, the example itself does not appear to be open source and so the source code would not be available for use.

The next option that was explored was jqTree [39], while this did in a sense create a tree like structure it was not as visually pleasing as envisioned. Furthermore, the tree itself was created by using JSON data and loaded via Ajax. JavaScript Object Notation (JSON) is a storage method utilising key-value pairs, unfortunately at this time the data that was being used was not formatted in this way so meant that this library was unsuitable for use in this context. A further discussion about JSON formatting will be covered in the following section: Data Formatting. Ajax is used to exchange data between a web page and a server [40] thus allowing for the web page to be updated to reflect new information. In the scope of this project, it was decided that the project would not use a server as the application would run locally, this made this library incompatible with this project.

The library that was finally chosen was Treant.js [41], this was deemed to be the best selection for a variety of reasons. Not only did it provide a number of different decision tree styles but it was also possible to implement using an array or using JSON structure. The documentation that was provided was clear and easy to follow which was also a deciding factor in this selection. It was implemented by linking to the scripts raphael.js, treant.js and the style sheets treant.css and super-simple.css in the index.html document. The array approach that is given as an example in the documentation was then included in script.js so that it would run after the file was read in. At this point, the index values that would populate the nodes were hard coded into the provided code. Unfortunately, this library does not contain a way to label the test value that has led to the decision, which can result in lack of clarity in what exactly the diagram is depicting. This could be solved by adding a legend that explains this.



### 3.5 Data reformatting

Thus far, it was possible to read in a file, split it and add each element to an array. It was also possible to extract the elements required and apply them to the Treant library in order to produce a simple tree with a root node and two children nodes, with each node containing the values of the text file stored at a unique index. Until this point the file that had been used was purely for testing purposes and not represented in the same way as the actual data that would be submitted by users. The next process that had to occur was to apply what had been achieved so far to an actual example.

The first problem that had to be addressed was how to extract only the data that was needed for the generation of the tree from the file. Figures 13 and 14 show an example of one of the files that the application was being built to process. This example was generated using test data that is provided by Weka and shows not only the tree but further information provided about the tree. The outputs vary based on the data that is used to generate them and so the first step was to identify any similarities between files that could be used to form a pattern that could be coded. It was identified that the lines: "J48 pruned tree" and "=== Stratified cross-validation ===" were consistent throughout the different files, though would appear at different line numbers. However, the spacing between these phrases and the start and end of the tree were always the same so it was attempted to isolate only the lines between these phrases as this made up the body of the tree.

From there, it was hoped that the data could be further broken up to take just the information that was required. In order to attempt this, the file was read in and then split on the new line character so that every line of the file would be a new index in the array. One issue that was encountered was that different file systems (Windows, OS X) had different line endings, which made splitting on any single character difficult. To remedy this it was necessary to standardise the new line character in all files. This was achieved by changing the setting of the file in Notepad++. This could be done by selecting "Edit", "EOL Conversion" and choosing Unix (LF). Once this issue had been remedied, the indexes of the phrases were then searched for using JavaScript's `indexOf` method [42], as it was known that while the indexes may change from file to file, the spacing between them and the actual tree was consistent. These indexes were then saved to new variables as they signified the start and end points of the data that needed to be extracted. JavaScript's `slice` method [43] was then used to remove each index between the start and end points of the array and these were stored in a new array. At this point, the data that was needed had been successfully extracted.

The next problem that became apparent was the need to further split the data as there were various characters that were not needed such as the "|" character. One idea that was trialled was to use the pipes ("|") as a count to identify how deep into the tree level the program was and thus to possibly help determine parents and children. The attempt to implement this can be seen on the following CodePen: <http://codepen.io/jessijinx/pen/mWQJbo>. In this case, only one line of the file was used to test this idea. The theory was that each line in the array could be split further on the space character. If the character at index 0 was a "|" then a count was created and incremented for each pipe that was encountered. The `splice` method was then used to remove the character from the array, the same was done for blank characters. While this did eventually return just the values that could be used it was unclear about how this could be used going forward with the library and the full file rather than just one line. In the full file the number of values on each line would change thus changing the indexes needed, this would also change further from file to file depending on the size of the

data set that it had been generated on. Consequently, there would be no consistency and it would not be reliable to assign a static index to the library.

```

1  === Run information ===
2
3  Scheme:      weka.classifiers.trees.J48 -C 0.25 -M 2
4  Relation:    vote
5  Instances:   435
6  Attributes:  17
7               handicapped-infants
8               water-project-cost-sharing
9               adoption-of-the-budget-resolution
10              physician-fee-freeze
11              el-salvador-aid
12              religious-groups-in-schools
13              anti-satellite-test-ban
14              aid-to-nicaraguan-contras
15              mx-missile
16              immigration
17              synfuels-corporation-cutback
18              education-spending
19              superfund-right-to-sue
20              crime
21              duty-free-exports
22              export-administration-act-south-africa
23              Class
24  Test mode:   2-fold cross-validation
25
26  === Classifier model (full training set) ===
27
28  J48 pruned tree
29  -----
30
31  physician-fee-freeze = n: democrat (253.41/3.75)
32  physician-fee-freeze = y
33  |   synfuels-corporation-cutback = n: republican (145.71/4.0)
34  |   synfuels-corporation-cutback = y
35  |   |   mx-missile = n
36  |   |   |   adoption-of-the-budget-resolution = n: republican (22.61/3.32)
37  |   |   |   adoption-of-the-budget-resolution = y
38  |   |   |   |   anti-satellite-test-ban = n: democrat (5.04/0.02)
39  |   |   |   |   anti-satellite-test-ban = y: republican (2.21)
40  |   |   |   |   mx-missile = y: democrat (6.03/1.03)

```

Figure 13: This figure shows the first part of the text output provided by Weka, displayed in Notepad++

```

41
42 Number of Leaves :      6
43
44 Size of the tree :  11
45
46
47 Time taken to build model: 0.02 seconds
48
49 === Stratified cross-validation ===
50 === Summary ===
51
52 Correctly Classified Instances      415           95.4023 %
53 Incorrectly Classified Instances    20           4.5977 %
54 Kappa statistic                    0.9039
55 Mean absolute error                 0.0809
56 Root mean squared error            0.1985
57 Relative absolute error             17.0509 %
58 Root relative squared error        40.7651 %
59 Total Number of Instances          435
60
61 === Detailed Accuracy By Class ===
62
63          TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
64          0.948   0.036   0.977    0.948   0.962     0.905   0.974    0.976    democrat
65          0.964   0.052   0.920    0.964   0.942     0.905   0.974    0.953    republican
66 Weighted Avg.   0.954   0.042   0.955    0.954   0.954     0.905   0.974    0.967
67
68 === Confusion Matrix ===
69
70      a   b   <-- classified as
71  253  14 |   a = democrat
72    6 162 |   b = republican
73

```

Figure 14: This shows the second part of the text output provided by Weka, displayed in Notepad++

Once it became apparent that using the data in the format it was originally in was not ideal one solution that was explored was to manually reformat the text file. This was attempted by converting the relevant information into JSON format. However, this presented unexpected complexities such as standardising it so that it would work with files that were larger than the test data file as various factors such as data names and indexes changed depending on the file content, this led to this option being discarded.

After this attempt was unsuccessful, the next possible solution that was explored was to find a program that would convert the file into JSON and try to integrate it into this application. Two applications were found that would do this, one python script was tried [45] but would not work even after updating the syntax. A Python application named ParseJ48.py did exist and was designed specifically for converting J48 decision trees into JSON [44]. While it did reformat the file, before being able to pass it to the program the file had to be in a certain format initially which required the alteration the original Weka output. Figure 15 shows the output of the conversion program, beautify was used on the file to make it easier to read, this was because the original output was all on one line.

```

1  ["physician-fee-freeze", [
2      ["=", "n", "democrat"],
3      ["=", "y", ["synfuels-corporation-cutback", [
4          ["=", "n", "republican"],
5          ["=", "y", ["mx-missile", [
6              ["=", "n", ["adoption-of-the-budget-resolution", [
7                  ["=", "n", "republican"],
8                  ["=", "y", ["anti-satellite-test-ban", [
9                      ["=", "n", "democrat"],
10                     ["=", "y", "republican"]
11                 ]]]
12             ]]],
13             ["=", "y", "democrat"]
14         ]]]
15     ]]]
16 ]]
```

Figure 15: The output from the use of ParseJ48.py which reformatted a Weka J48 output. This was opened in Visual Studio Code and Beautify was applied to the file to improve the clarity of information being displayed.

While the output is not JSON as was indicated by the program, it did appear that the data had been separated into an array of arrays and it was hoped that this output could still be used. It was hoped that a solution to converting the file internally into this format could be found at a later date, perhaps by rewriting the Python script to JavaScript.

Over time it became clear that there were additional issues that came with processing the file in its new form. This was because, although the content was that of an array when the file was read in using the FileReader API, it was still being read as a string and was not identified as being an array. This returned to the previous problem of indexes of the elements that were wanted changing from file to file. It does not appear that there is a way to read the content of a file as an array so it would have to be split like any other string. It was later found that characters could be replaced in the file using a combination of the replace method and regular expressions which allowed for the isolation of only the information that was required.

The problem that was posed at this point was how to iterate through the data and identify the parent nodes and their children. The program would then be required to continue to create correct nodes until an end point was reached. This is ultimately where further investigation was required in order to find a suitable solution. One option that was explored is as follows:

1. Identify the test values in each file e.g. Yes/No, 1/0
2. Identify the nodes that were decisions and assign the parent node to it
3. Identify the nodes that were not decisions but new test cases, assign the parent but also make the node a parent in its own right.
4. Carry out a check that every test case could be associated with all test values
5. If a test case is not followed by all test values, identify the missing test value and identify which node is its parent.

Steps 1-3 would be wrapped in a recursive function that would be called recursively on each child of the parent (the node in question for the function). This potential solution was not implemented due to the complexities that the file content could pose, the greatest of these

would be devising a way in which test cases could be associated with the correct parent when they do not directly precede them.

Ultimately, while it wasn't possible to generate a tree from the initial text file, it was possible to hardcode the indexes that held the required information to create a decision tree from the reformatted file generated by the ParseJ48 script. As the indexes of the required values were specific to each file based on a particular data set, no way was identified to accurately and dynamically create nodes correctly based on the data inputs. As such, this cannot be classified as a universal solution to the problem of generating an image of decision trees based on a data input.

### 3.6 Download Function

One of the project requirements was to devise a way in which users could obtain the tree, once it had been generated, as an image. The download function was implemented by using the DOM-to-image library [46] which allows for the conversion of an HTML element into an image. The user is given the option to download the image as a png file or as a jpeg. These formats were selected as they are commonly used. When the download as jpeg button is clicked, the DOM node is converted into a jpeg and downloaded to the user's computer. A similar process is carried out when selecting download as png except a third party library called FileSaver.js [47] is used in conjunction with the DOM-to-image library. This allows the node to be converted into a blob and then a saveAs method is called which initialises the download.

While these methods do work inasmuch as an image of the basic tree is downloaded, figure 16 shows that it does not look as it does when you load the file in as the colours have been altered and the branches of the tree that can be seen when the tree is generated are no longer visible. It is not currently clear what the cause of these changes are, but it alters the representation of the tree significantly and these issues would need to be resolved before this function could be deemed to be fully operational.

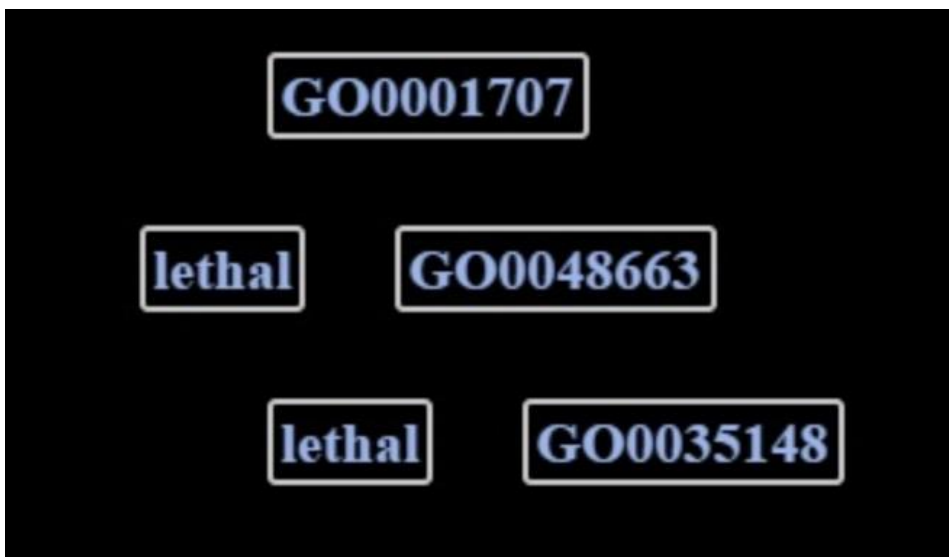


Figure 16: The result of downloading the file from the webpage by selecting the download as a jpeg file button, viewed in Microsoft Photos (version 17.313.10010.0)

## 4. Testing

Testing is an integral process in any software development project as it can identify problems and ensure the quality and functionality of the product. The code was often tested in the online tool CodePen to determine to what extent it worked with the existing functions before adding it to the main code body. Functionality was often checked using a web browser and inspecting the code within the page, this logged any errors to the console that could then be remedied. Finally, testing was carried out across a number of different web browsers to determine how well the application worked in separate web environments.

## 4.1 CodePen

CodePen was used extensively during the development of this project. Before code was added to the main code base it was tested using this tool to ensure that it functioned as expected.

## 4.2 Incorrect File Type

One test that is built into the code is the ability to check that the file a user has selected to upload is the correct file type. As can be seen from figure 12, when an incorrect file type is selected the user will be presented with a message informing them of this. They can then select another file using the form as they normally would. One issue that is currently experienced is if a user uploads a file that is written in plain text, such as CSS files, the program will still attempt to read it but nothing will be displayed to screen. This problem can be seen in figure 17. It is not certain what the cause of this is but one assumption could be that the content of both CSS files and normal text files are read as plain text, which could be why it is not being picked up as an error.

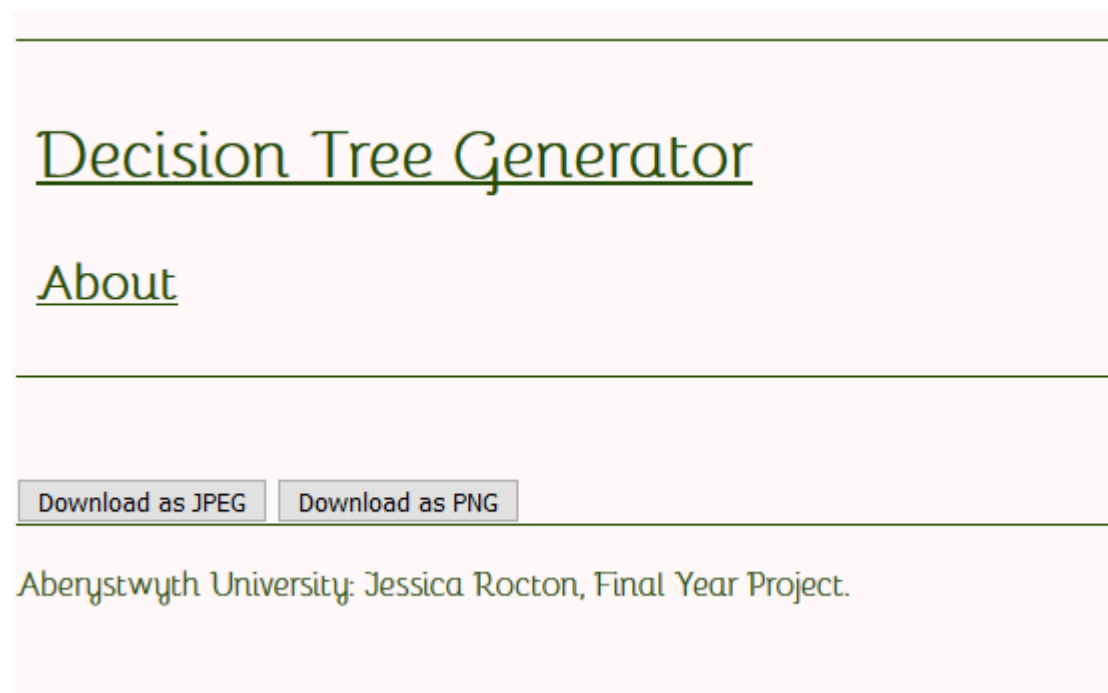


Figure 17: This shows the result when a file type that contains plain text but is not a text file, such as a CSS file, is uploaded to the form.

## 4.3 Browser compatibility

An important test to determine the usability of this application was to test its functionality on a number of different internet browsers as this will be the platform by which users access the

program. This would establish which functions worked in which browsers, and identify the application's level of cross browser compatibility.

### 4.3.1 Google Chrome, version 57.0.2987.133

Google Chrome was the default browser used throughout the development of this web application. This was due to the extensive support it offers for web technologies and its cross platform compatibility with Windows, Linux and Mac OS. Additionally, PC Advisor found Google Chrome to possess 52% of the web browser market share [48], with such a high proportion of users, it was important to ensure the application would run in this environment. Much of the testing of the functionality of implemented code was done using Google's inspect element as this provided run time information such as errors in the console. Figure 18 shows the content of a text file that was read in being logged to the console, this was to ensure that the read had been successful.



```

=== Run information ===
midProjectJS.js:20

Scheme:      weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:    vote
Instances:    435
Attributes:   17
handicapped-infants
water-project-cost-sharing
adoption-of-the-budget-resolution
physician-fee-freeze
el-salvador-aid
religious-groups-in-schools
anti-satellite-test-ban
aid-to-nicaraguan-contras
mx-missile
immigration
synfuels-corporation-cutback
education-spending
superfund-right-to-sue
crime
duty-free-exports
export-administration-act-south-africa
Class
Test mode:    2-fold cross-validation

=== Classifier model (full training set) ===

J48 pruned tree
-----

physician-fee-freeze = n: democrat (253.41/3.75)
physician-fee-freeze = y
| synfuels-corporation-cutback = n: republican (145.71/4.0)
| synfuels-corporation-cutback = y
| | mx-missile = n
| | | adoption-of-the-budget-resolution = n: republican
| | | (22.61/3.32)
| | | adoption-of-the-budget-resolution = y
| | | | anti-satellite-test-ban = n: democrat (5.04/0.02)
| | | | anti-satellite-test-ban = y: republican (2.21)
| | | mx-missile = y: democrat (6.03/1.03)

Number of Leaves :    6

Size of the tree :    11

Time taken to build model: 0.02 seconds

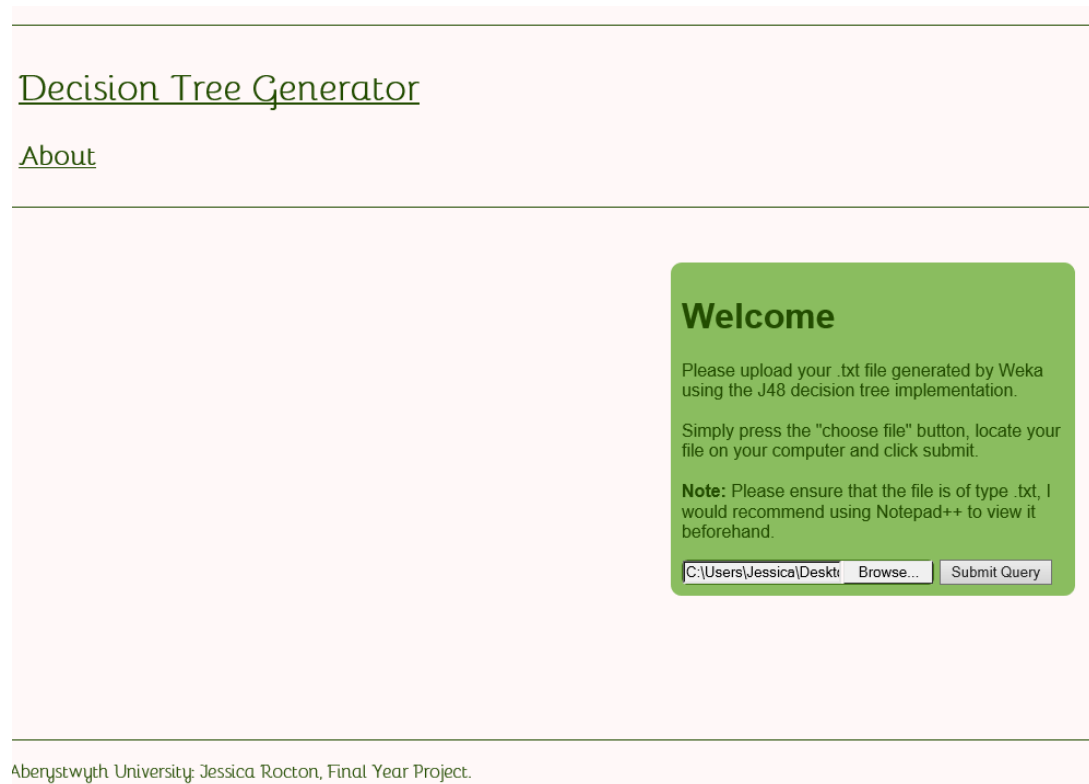
```

Figure 18: Shows the result of reading in a Weka decision tree test file being logged to the console. This was displayed by using the inspect function in Google Chrome.

It has been found that when used in Google Chrome, the program offers the full functionality that was expected.

### 4.3.2 Microsoft Edge (ME), version 38.14393.1066.0

Microsoft's new browser ME was tested as it is offered as a default browser on Windows 10 pc's and so could be selected by some users to run the program. While the basic functionality of being able to read in and generate a tree was retained in this browser, the browser altered the form slightly (see figure 19). The submit button was relabelled to "Submit Query" and the browser button was moved to the right side of the form. Additionally, neither download functions work as the libraries used to generate the download methods do not seem to be compatible for use in ME.



Aberystwyth University: Jessica Rocton, Final Year Project.

Figure 19: Homepage view in Microsoft Edge



Aberystwyth University: Jessica Rocton, Final Year Project.

Figure 20: Result generated by running the program and submitting a file using Microsoft Edge



#### [4.3.4 Internet Explorer \(IE\), version 11.1066.14393.0.](#)

Testing found that this application is not suitable for use with IE. While index.html would display and it was possible to select a file to upload, the file was not uploaded. As such, further tests were not conducted as the most basic functionality was not possible.

#### [4.3.5 Mozilla Firefox version 53.0](#)

Mozilla Firefox was tested as it is still considered to be one a popular browser. When tested it was found that the program retained most of its functionality when run. It was possible to select and read in a file, generate a tree that would be displayed to screen (figure 22), and unlike other browsers, it was possible to download the tree as a png file. However, like the other browsers (excluding Chrome), the download and jpeg option also did not work. Similarly to ME, Firefox also renamed the submit button to “Submit Query” (figure 21).

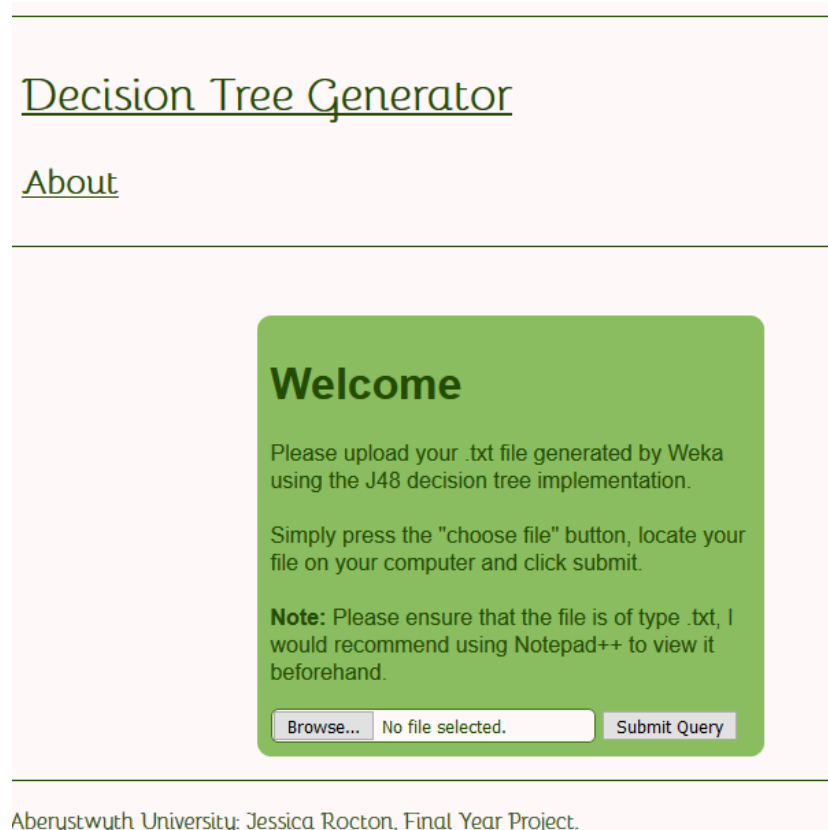
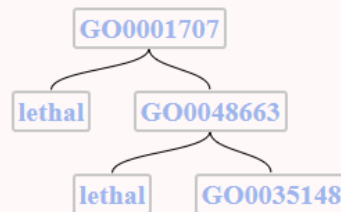


Figure 21: Homepage view when opened in Mozilla Firefox

# Decision Tree Generator

## About



[Download as JPEG](#) [Download as PNG](#)

Aberystwyth University: Jessica Rocton, Final Year Project.

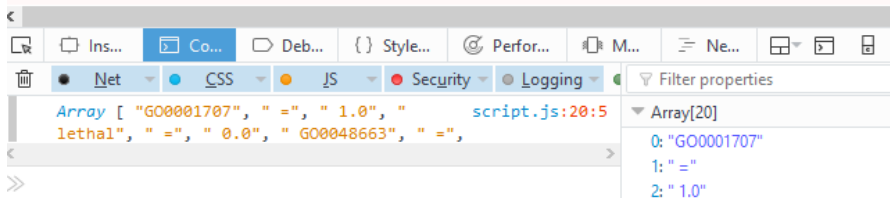


Figure 22: Result generated by submitting a file and generating a tree from the content, displayed in Mozilla Firefox. The bottom of the image also shows the array of the file that was read in being printed to the console.

### 4.3.6 Safari version 10.10 "Yosemite"

Safari is the web browser provided by Apple for Mac computers and Apple portable devices. It was important to test this application on other operating system's (OS) default browsers, particularly Safari, who possess the greatest web browser market share after Google Chrome [48]. It was possible to select a file and generate a tree based on the data (figure 24), however, like Microsoft Edge, neither download functions worked which affects the overall usability of the application in this browser.



Figure 23: Homepage view in Apple Safari

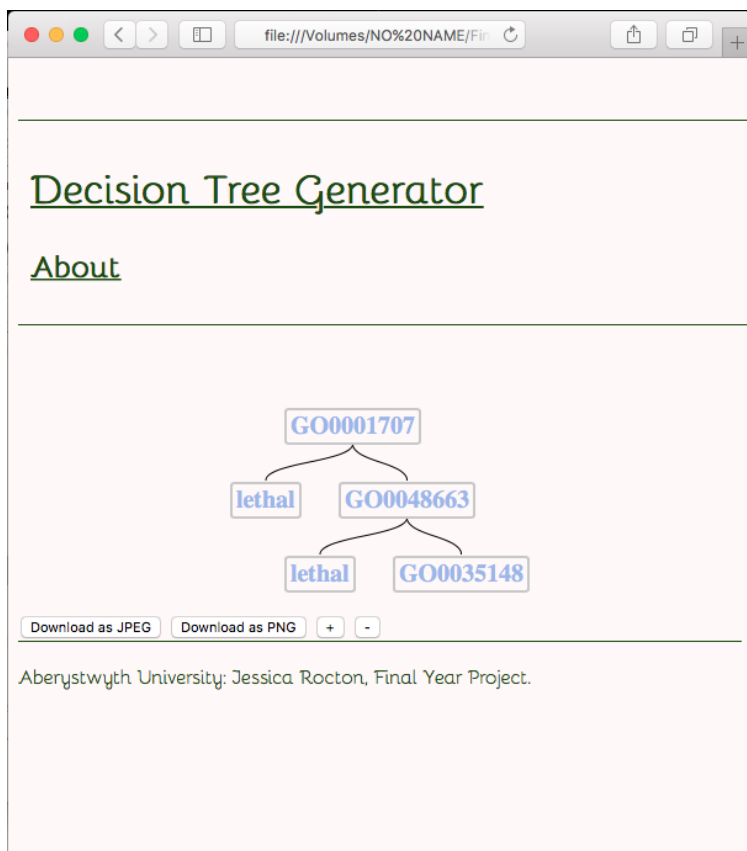


Figure 24: Result generated by submitting a file to the program and generating a tree from the content, displayed in Apple Safari.

Generally speaking, the majority of browsers tested would allow for a file to be selected and read in, a tree in the expected format will be generated and displayed to screen. However in all but one browser, the function that allowed the tree to be downloaded as a jpeg file was not compatible, this was due to the library used to implement this function not supporting the other browsers tested. This was a similar case for the download as png function, however, unlike the jpeg download, this functioned in both Firefox and Chrome. For full functionality of this application, it is recommended that it should be run in Google Chrome version 57.0.2987.133.

## 5. Evaluation

### 5.1 Were the requirements correctly identified?

The key requirements for this project consisted of 3 main functions:

1. Allow users to input their Weka decision tree outputs into the program
2. Generate a visual representation of the inputted data.
3. Provide a way in which the generated tree could be obtained as an image.

The essence of the problem that this project aimed to solve was that there are very limited and reliable ways by which Weka decision tree outputs can be viewed in a pleasing and easy to understand format.

While the creation of a GUI was not a specific requirement, based on the current tools available and the analysis of the problem, it was decided that this would be an effective way to implement a solution.

The requirements outlined, simple as they were, would remedy this if successfully implemented. For this reason, I will argue that the requirements were correctly identified, if not fully achieved.

### 5.2 Were the design decisions correct?

Throughout the course of this project there were design decisions that were made that were correct, but equally, there were some decisions that were questionable and impacted the progress of the project.

I believe the decisions I made in regards to the UI design were successful. The decision to create an application with a UI was based on the wish to develop a way that the user could easily interact with the program. This would also differentiate my tool from Grahpviz, which is run using the command line. Ensuring that the tool was easy to use was a priority as making it unnecessarily complex would be a detriment to the user's experience. Saying this, I do think that the site could look more aesthetically pleasing, for example by making small changes to the style and position of the buttons.

One design decision that I believe was less successful was the choice of tree visualisation design. The tree design and layout were suitable for what I was trying to display, however, the library that was selected did not include labels on the tree's branches. This impacts the clarity of the diagram as it is not possible to tell which decision led to a particular node.

Another design decision that I now question was the decision to create a client side app. While the reasoning behind making the application run on the client side, rather than a

server did have some benefits, if I had been using a server it might have made certain functions, such as downloading, more successful.

### **5.3 Could a more suitable set of tools have been chosen?**

Based on the decision to create a web application, the choice of using CSS, JavaScript and HTML was both a sensible and appropriate one. This is due to their wide usage and therefore the extensive support that is available for them in most browsers and as such, I do not believe that anything more suitable could have been selected in this context.

As mentioned previously, the library that was used to generate the tree does not provide labelling of the branches which affects the clarity and understanding of the output. Despite this, out of the various libraries and programs that were explored, none appeared to offer this functionality other than Weka itself and Graphviz, neither of which were compatible for use with JavaScript. This would have left me with the option of drawing the output manually if I wanted this feature, which would not have been realistic as I did not feel that quality I could create would match that of pre-existing solutions. Based on the existing libraries I had to choose from, only Treant seemed to suit my needs. While it was not a perfect solution, I still believe that it was the best that was available to me.

The choice of VS Code as the IDE for this project was very beneficial due to the support it offered to a wide range of languages and the additional extensions that were available to make the coding process more efficient. Additionally being able to push changes to my GitHub account made version control incredibly quick and simple.

The tool used to convert the Weka data into an alternative format, ParseJ48, was less than ideal. The fact that it was written in Python meant that it would have to be rewritten in JavaScript to be integrated with my application. Due to my limited experience with this language, it was not possible to include this conversion in the scope of the project and this will mean the user, at this time, would have to also run this program to reformat their file so my application can process it. This is not ideal from a usability standpoint as it adds further complications to the process. Despite this, there were very few other tools that were available and that worked consistently, as such, this limited the alternative options that could be considered.

### **5.4 How well did the software meet the needs of those who were expecting to use it?**

The two key aims of this project were to product a piece of software that would allow users to input a Weka text output of a decision tree and create a visual representation of it. Secondly, it was important to provide a way in which the tree visualisation could be obtained by the user in an image format.

At the point that this project reached, it was possible to read in a text file and produce a tree like visualisation based on the file content. However, the biggest shortcomings were that in order for the program to work, the file had to be reformatted using an external 3rd party program and this output was what the program would read in, not the original Weka text output. The implication of using a 3rd party library is that users, at this time, would have to download an additional piece of software and convert their file before being able to use my program. This is not ideal as it includes extra complexities for the user which was something I wanted to avoid. Ideally, this process of reformatting the file into a more usable state would have been internalised. However, as a result of my lack of knowledge of the programming language

python and the lack of other software available that could fulfil this function, I was not able to implement this at this at this time.

Additionally, the tree was generated by hard coding in the number of nodes the tree would need and the indexes at which each element that was needed could be found within a specific data input. Consequently, this means this program can only accurately generate a tree based on one file, this results in limited usability. As mentioned this occurred due to complexities in generating an algorithm that would accurately iterate through the data and assign the correct nodes to their parents.

Finally, while it is possible to download the tree visualisation this function is limited to only two browsers, with the jpeg download not being supported in one of these. There are also known bugs with the download leading it to not display the branches, this is not ideal and would have to be handled before this function could be fully usable.

While this software does not fully meet the needs of those who would be expected to use it, it would offer a good starting point for anyone who would wish to adapt this code in the future.

## 5.5 How well were any other project aims achieved?

The other project aim was to make this program available through BioJS. As the project currently stands I do not feel it is at a suitable level of completion to publish it on this platform and I would not feel confident making it available to users as it would not be able to fully meet their needs.

While the aim to make this software available on BioJS was not fulfilled, I still wanted to make what I had readily available to others. I hope that some, or all of the project might be able to help someone else in the future, for this reason, it is publically available on GitHub.

## 5.6 If you were starting again, what would you do differently?

While BioJS clearly shows how powerful JavaScript can be as a visualisation tool if I were to repeat this project I would look into other languages and the support they offer for data visualisation. For example, whenever I tried to reformat the data the programs that currently existed to do this were written in Python. As such, they could not be directly integrated into my application without being rewritten.

One of the key features that I wanted from my project to distinguish itself from other tools was to include a graphical user interface (GUI) as most other current tools are run from the command line. I knew this would make it easier for the user to interact with the program and would improve their overall experience. While using web tools made creating a UI both quick and easy, other languages could have supported me in doing this. Python offers Tkinter [49] as a library for creating graphical user interfaces and using Python would have allowed me to easily integrate the currently existing 3rd party libraries that were used to reformat the data.

## 5.7 Future work

This project provides significant potential for future development. The first option would be to increase the functionality that is offered by the program. One suggestion would be to add a zoom function, this would allow the user to better inspect larger trees. It could also be linked

to the download function as the user may only wish to display part of the tree and by zooming into the area of interest it could be useful to only download that section.

Further functionality could be added by allowing users to manipulate the tree. For example, this could take the form of providing the ability to move branches and nodes. Once again, this would add to the user's capability to focus on, and highlight, key areas of the tree that are of particular interest and would allow the tree spacing to be customised to the user's preference. An additional opportunity could also be to add colour coding to the nodes, by defining each decision by colour this will improve clarity and understanding for those who might be looking at the image in a publication.

Finally, an interesting option to explore would be the use of Electron to convert the program into a desktop application. Electron is an open source framework which allows programmers to create native apps using web technologies [50]. By making the program available as a desktop application, this would resolve some of the problems that were faced with cross browser compatibility and could provide a smoother user experience.

## 6. Appendices

### A. Third-Party Code and Libraries

**Treant.js**- This library was used to generate the tree structure from the text file that is read into the application. This library is open source and available from GitHub. The only modifications made to the code provided was to insert the required indexes of the items stored in the array that would provide the information to be generated by the tree.

**DOM-to-Image** - This library was used to change the element that contained the tree into an image so that additional functions could be applied to it more easily. This library is open source and available from GitHub. No changes were made to this code

**FileSaver.js** - This library was used to save the image of the generated tree as a PNG file. This library is open source and available from GitHub. No changes were made to this code.

**HTML5 FileReader API Demo** - This is a CodePen example that I used to understand how the FileReader API can be implemented. This particular example was publically available on the platform so its reuse is permitted [51]. Relatively little of this code was used directly as this example was for use with an image, whereas I needed to read in text, but the example of checking for the file type was taken from this although the output was altered.

**Weka-json-parser** - This is the program that was used to convert the Weka J48 decision tree example into JSON. This was publically available on GitHub, the code was changed by adding “()” to update the syntax in the final line so that it would work with Python 3.

**Mega BoilerPlate** - Mega Boilerplate was initially used to generate a basic website and set up the local host on which it would have run, this was used as a proof of concept and was not used going forward in the main part of the project. This is publically available for use via GitHub, the only changes that were made were alterations to the default jade code that was provided in order to alter the landing page appearance.

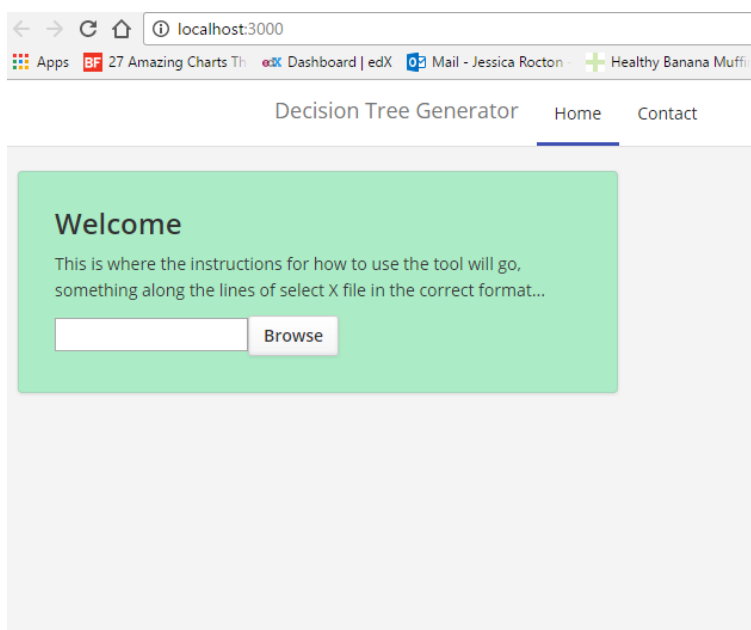
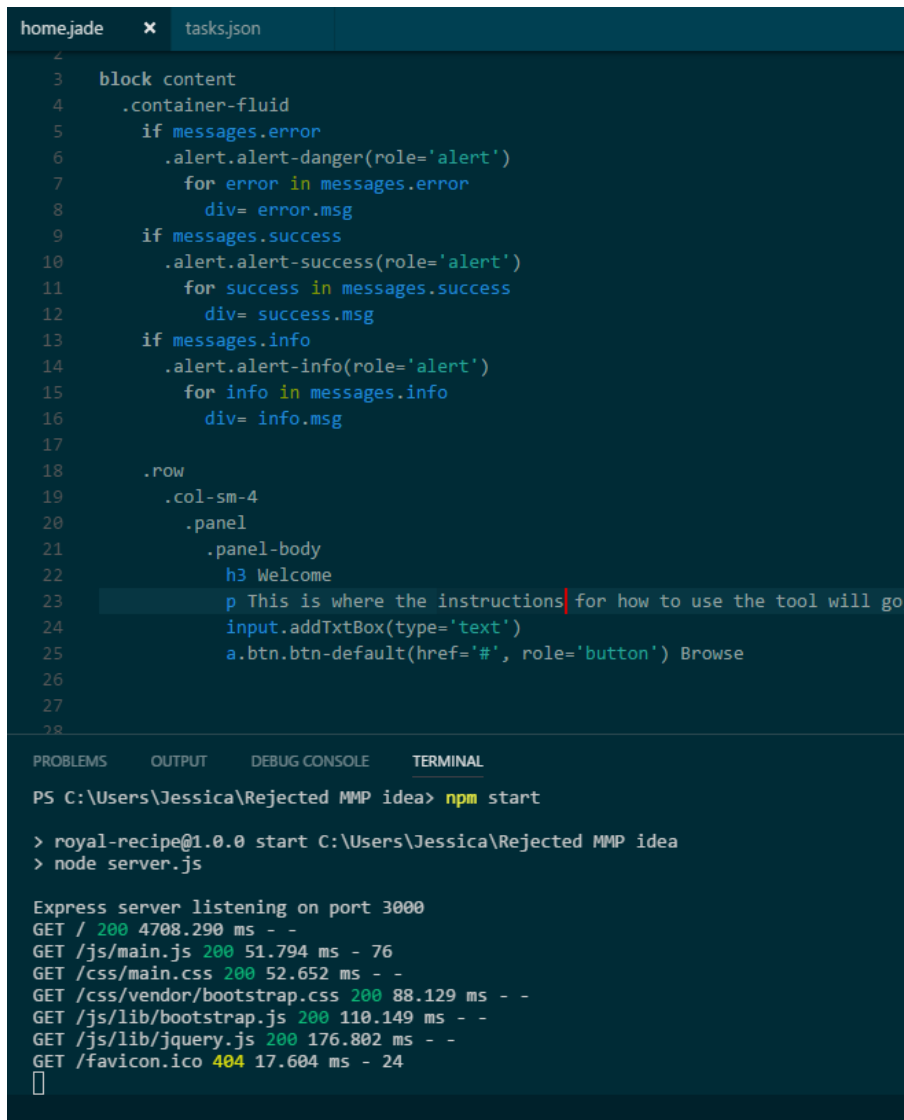


Figure 25: This figure shows the original site concept when it was generated using Mega Boilerplate





The screenshot shows an IDE with two tabs: 'homejade' and 'tasks.json'. The 'homejade' tab is active, displaying Jade/Pug code for a web page. The code includes conditional rendering for error, success, and info messages, a row of columns, a panel body with a welcome message, a text input, and a browse button. Below the code editor, there is a terminal window with tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', and 'TERMINAL'. The 'TERMINAL' tab is active, showing the command 'npm start' and the output of the Express server, including various GET requests and their response times.

```
2
3   block content
4     .container-fluid
5       if messages.error
6         .alert.alert-danger(role='alert')
7           for error in messages.error
8             div= error.msg
9       if messages.success
10        .alert.alert-success(role='alert')
11          for success in messages.success
12            div= success.msg
13      if messages.info
14        .alert.alert-info(role='alert')
15          for info in messages.info
16            div= info.msg
17
18    .row
19      .col-sm-4
20        .panel
21          .panel-body
22            h3 Welcome
23            p This is where the instructions for how to use the tool will go
24            input.addTextBox(type='text')
25            a.btn.btn-default(href='#', role='button') Browse
26
27
28
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS C:\Users\Jessica\Rejected MMP idea> npm start

> royal-recipe@1.0.0 start C:\Users\Jessica\Rejected MMP idea

> node server.js

Express server listening on port 3000

GET / 200 4708.290 ms - -

GET /js/main.js 200 51.794 ms - 76

GET /css/main.css 200 52.652 ms - -

GET /css/vendor/bootstrap.css 200 88.129 ms - -

GET /js/lib/bootstrap.js 200 110.149 ms - -

GET /js/lib/jquery.js 200 176.802 ms - -

GET /favicon.ico 404 17.604 ms - 24

□

Figure 26: Showing the Jade (now known as Pug) that was used to code the page as well as the requests being made to the server upon opening the page.

## B. Ethics Submission

Ethics Application reference number: 6723

### AU Status

Undergraduate or PG Taught

### Your aber.ac.uk email address

jer26@aber.ac.uk

### Full Name

Jessica Rocton

**Please enter the name of the person responsible for reviewing your assessment.**

Reyer Zwiggelaar

**Please enter the aber.ac.uk email address of the person responsible for reviewing your assessment**

rrz@aber.ac.uk

**Supervisor or Institute Director of Research Department**

cs

**Module code (Only enter if you have been asked to do so)**

**Proposed Study Title**

Conversion of Weka decision tree outputs to visually understandable decision trees.

**Proposed Start Date**

30th Jan 2017

**Proposed Completion Date**

6th May 2017

**Are you conducting a quantitative or qualitative research project?**

Mixed Methods

**Does your research require external ethical approval under the Health Research Authority?**

No

**Does your research involve animals?**

No

**Are you completing this form for your own research?**

Yes

**Does your research involve human participants?**

No

**Institute**

IMPACS

**Please provide a brief summary of your project (150 word max)**

Creation of decision trees from Weka data outputs

**Where appropriate, do you have consent for the publication, reproduction or use of any unpublished material?**

Not applicable

**Will appropriate measures be put in place for the secure and confidential storage of data?**

Yes

**Does the research pose more than minimal and predictable risk to the researcher?**

No

**Will you be travelling, as a foreign national, in to any areas that the UK Foreign and Commonwealth**

**Office advise against travel to?**

No

**Please include any further relevant information for this section here:**

**If you are to be working alone with vulnerable people or children, you may need a DBS (CRB) check.**

**Tick to confirm that you will ensure you comply with this requirement should you identify that you require one.**

Yes

**Declaration: Please tick to confirm that you have completed this form to the best of your knowledge and**

**that you will inform your department should the proposal significantly change.**

Yes

**Please include any further relevant information for this section here:**

## C. Code Samples

```
28 |         config = {
29 |             container: "#tree-simple"
30 |         };
31 |         parent_node = {
32 |             text: { name: myData[0] }
33 |         };
34 |         first_child = {
35 |             parent: parent_node,
36 |             text: { name: myData[3] }
37 |         };
38 |         second_child = {
39 |             parent: parent_node,
40 |             text: { name: myData[6] }
41 |         };
42 |         Third_child = {
43 |             parent: second_child,
44 |             text: { name: myData[9] }
45 |         };
46 |         fourth_child = {
47 |             parent: second_child,
48 |             text: { name: myData[12] }
49 |         };
50 |         simple_chart_config = [
51 |             config, parent_node,
52 |             first_child, second_child, Third_child, fourth_child
53 |         ];
54 |         var my_chart = new Treant(simple_chart_config);
55 |     }
56 | }
```

Figure 27: The code implemented to generate the tree from submitted data. This shows the creation of each node, the assignment of the parent node, assignment of indexes to input the information that would be displayed on each node and the initialisation of the library.

```
113 |         //download as JPEG function implementing the DOM-to-image library
114 |
115 |         $("#button").on("click", function () {
116 |             console.log("click");
117 |             var node = document.getElementById('tree-simple');
118 |             domtoimage.toJpeg(document.getElementById('tree-simple'), { quality: 0.95 })
119 |                 .then(function (dataUrl) {
120 |                     var link = document.createElement('a');
121 |                     link.download = 'J48 Converted Tree.jpeg';
122 |                     link.href = dataUrl;
123 |                     link.click();
124 |                 });
125 |
126 |         })
127 |
128 |         $("#button2").on("click", function () {
129 |             console.log("click");
130 |             domtoimage.toBlob(document.getElementById('tree-simple'))
131 |                 .then(function (blob){
132 |                     saveAs(blob, 'J48 Converted Tree.png');
133 |                 })
134 |
135 |         })
```

Figure 28: Implementation of the code used to enable the tree output to be downloaded as a jpeg and png. The functions are passed the element that stores the tree and converts it to a blob/dataURL

## D. Testing Table

Test Number	Browser	Test Description	Expected Output	Actual Output	Pass (Y/N)
T1	Chrome	Upload a text file using JavaScript FileReader API and print text file content to console	Print text file content to console	Printed text file content to console	Y
T2	Chrome	Upload incorrect file type and print fail message to console	Print fail message to console	Printed fail message to console	Y
T3	Chrome	Add file content to array and print file content as individual array elements to console	Print file content as array elements	Printed file contents as array elements	Y
T4	Chrome	Print file content backwards to show file manipulation	Print file content to console backwards	Printed file contents backwards	Y
T5	Chrome	Print file content to webpage	File will be printed to the webpage in the format it was read in	File was printed to webpage in expected format	Y
T6	Chrome	Upload incorrect file type	Error message will be displayed on the page	Error message was displayed on the page	Y
T7	Chrome	Generate tree and display to webpage	Selected file content will populate tree nodes and be displayed to the webpage using the tree library in the expected format	File was displayed in a tree structure in the expected format	Y
T8	Chrome	Clicking download as jpeg button initialises the download of the tree as a jpeg	Download of a jpeg named J48 Converted Tree.jpeg will be initialised	Download of a jpeg named J48 Converted Tree.jpeg will be initialised	Y

T9	Chrome	Clicking download as png button initialises the download of the tree as a jpeg	Download of a jpeg named J48 Converted Tree.png will be initialised	Download of a jpeg named J48 Converted Tree.png will be initialised	Y
T10	Internet Explorer	Print file content to webpage	File will be printed to the webpage in the format it was read in	File was printed to webpage in expected format	N
T11	Internet Explorer	Generate tree and display to webpage	Selected file content will populate tree nodes and be displayed to the webpage using the tree library in the expected format	File was displayed in a tree structure in the expected format	N
T12	Safari	Selecting a file for upload	Clicking "Choose file" opens local file storage and a file can be selected	Local File storage was opened on click of "Choose file" button Selected file name populates the form	Y
T13	Safari	Upload incorrect file type	Error message will be displayed on the page	Error message was displayed on the page	Y
T14	Safari	Generate tree and display to webpage	Selected file content will populate tree nodes and be displayed to the webpage using the tree library in the expected format	File was displayed in a tree structure in the expected format	Y
T15	Safari	Clicking download as jpeg button initialises the download of the tree as a jpeg	Download of a jpeg named J48 Converted Tree.jpeg will be initialised	Download of a jpeg named J48 Converted Tree.jpeg will be initialised	N

T16	Safari	Clicking download as png button initialises the download of the tree as a jpeg	Download of a jpeg named J48 Converted Tree.png will be initialised	Download of a jpeg named J48 Converted Tree.png will be initialised	N
T17	Mozilla Firefox	Selecting a file for upload	Clicking "Choose file" opens local file storage and a file can be selected	Local File storage was opened on click of "Choose file" button Selected file name populates the form	Y
T18	Mozilla Firefox	Upload incorrect file type	Error message will be displayed on the page	Error message was displayed on the page	
T19	Mozilla Firefox	Generate tree and display to webpage	Selected file content will populate tree nodes and be displayed to the webpage using the tree library in the expected format	File was displayed in a tree structure in the expected format	Y
T20	Mozilla Firefox	Clicking download as jpeg button initialises the download of the tree as a jpeg	Download of a jpeg named J48 Converted Tree.jpeg will be initialised	Download of a jpeg named J48 Converted Tree.jpeg will be initialised	N
T21	Mozilla Firefox	Clicking download as png button initialises the download of the tree as a jpeg	Download of a jpeg named J48 Converted Tree.png will be initialised	Download of a jpeg named J48 Converted Tree.png will be initialised	Y
T22	Microsoft Edge	Selecting a file for upload	Clicking "Choose file" opens local file storage and a file can be selected	Local File storage was opened on click of "Choose file" button Selected file name populates the form	Y

T23	Microsoft Edge	Upload incorrect file type	Error message will be displayed on the page	Error message was displayed on the page	Y
T24	Microsoft Edge	Generate tree and display to webpage	Selected file content will populate tree nodes and be displayed to the webpage using the tree library in the expected format	File was displayed in a tree structure in the expected format	Y
T25	Microsoft Edge	Clicking download as jpeg button initialises the download of the tree as a jpeg	Download of a jpeg named J48 Converted Tree.jpeg will be initialised	Download of a jpeg named J48 Converted Tree.jpeg will be initialised	N
T26	Microsoft Edge	Clicking download as png button initialises the download of the tree as a jpeg	Download of a jpeg named J48 Converted Tree.png will be initialised	Download of a jpeg named J48 Converted Tree.png will be initialised	N

# Annotated Bibliography

## Analysis

[1] Weka 3: Data Mining Software in Java, Machine Learning Group at the University of Waikato [online] Available at: <http://www.cs.waikato.ac.nz/ml/weka/> [Accessed 9/2/17]

A brief explanation of the purpose and use of Weka data mining software.

[2] Weka.sourceforge.net. (n.d.). J48. [online] Available at: <http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/J48.html> [Accessed 9 Mar. 2017].

A summary of the J48 class and its elements within Weka.

[3] Decision Tree. [online] Investopedia. Available at: <http://www.investopedia.com/terms/d/decision-tree.asp> [Accessed 3 May 2017].

Definition of a decision tree and its use

[4] biojs.net. (2016). About BioJS. [online] Available at: <https://biojs.net/about/> [Accessed 7 Mar. 2017].

A brief summary of the purpose and function of BioJS and information about the community and founders.

[5] Corpas M, Jimenez R, Carbon SJ, et al. BioJS: an open source standard for biological visualisation – its status in 2014. F1000Research. 2014;3:55. doi:10.12688/f1000research.3-55.v1. [online] Available at: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4103492/> [Accessed 8/7/17]

A paper explaining in further depth about BioJS, including the aims of the project, the project's background and the direction in which the project is going.

[6] BioJS.io. (2016). TnT library for displaying trees and track-based annotations. [online] Available at: <http://biojs.io/d/tntvis> [Accessed 2 May 2017].

An example of a data visualisation tool to for the generation of phylogenetic trees.

[7] Graphviz.org. (n.d.). About | Graphviz - Graph Visualization Software. [online] Available at: <http://www.graphviz.org/About.php> [Accessed 3 May 2017].

Background on a software package that allows Weka decision tree outputs to be converted into a visual tree that can be saved as an image.



[8] Sehn Körting, T. (2014). How to create elegant decision trees using Weka and Graphviz. [video] Available at: <https://www.youtube.com/watch?v=jIRxlcTbCso> [Accessed 3 May 2017].

A video tutorial demonstrating how to convert Weka decision trees into an image using Graphviz

[9] Gansner, E., Koutsofios, E. and North, S. (2006). Drawing Graphs with dot. 1st ed. [ebook] Available at: <http://www.graphviz.org/Documentation/dotguide.pdf> [Accessed 3 May 2017].

A paper explaining the dot language and how it is used

[10] Code.visualstudio.com. (n.d.). Visual Studio Code - Code Editing. Redefined. [online] Available at: <https://code.visualstudio.com/> [Accessed 7 Mar. 2017].

Link to the download and brief introduction of VS Code.

[11] Marketplace.visualstudio.com. (n.d.). Beautify - Visual Studio Marketplace. [online] Available at: <https://marketplace.visualstudio.com/items?itemName=HookyQR.beautify> [Accessed 8 Feb. 2017].

This is the link that explains and the extension Beautify, used during this project. This includes install instructions

[12] Abusaid, M. (n.d.). HTML Snippets - Visual Studio Marketplace. [online] Marketplace.visualstudio.com. Available at: <https://marketplace.visualstudio.com/items?itemName=abusaidm.html-snippets> [Accessed 8 Feb. 2017].

This is the link that explains and the extension HTML-snippets, used during this project. This includes install instructions

[13] GitHub. (2016). sahat/megaboilerplate. [online] Available at: <https://github.com/sahat/megaboilerplate> [Accessed 6 Feb. 2017].

This links to the mega boilerplate project repository on GitHub which includes the readme explaining the use of the tool.

[14] npm. (2015). jade. [online] Available at: <https://www.npmjs.com/package/jade> [Accessed 21 Apr. 2017].

An explanation of Jade and its use

[15] Codepen.io. (2017). About CodePen. [online] Available at: <http://codepen.io/about/> [Accessed 18.Apr 2017].

Explains the basic principle and function of CodePen

[16] Brown, K. (2014). What Is GitHub, and What Is It Used For? [online] Howtogeek.com. Available at: <https://www.howtogeek.com/180167/htg-explains-what-is-github-and-what-do-geeks-use-it-for/> [Accessed 24 Apr. 2017].

[17] Latex-project.org. (n.d.). Introduction to LaTeX. [online] Available at: <https://www.latex-project.org/about/> [Accessed 1 May 2017].

An explanation of the LaTeX project, its use and availability

[18] Sharelatex.com. (n.d.). ShareLaTeX, Online LaTeX Editor. [online] Available at: <https://www.sharelatex.com/> [Accessed 1 May 2017].

Link to register to use ShareLaTeX as well as an explanation of what it does

[19] Myer, T. (2005). A Really, Really, Really Good Introduction to XML — SitePoint. [online] SitePoint. Available at: <https://www.sitepoint.com/really-good-introduction-xml/> [Accessed 20 Mar. 2017]

An introduction to the basic concept of XML.

[20] Online Documents Converter [online] <http://www.swiftconverter.com> Available at: <http://www.swiftconverter.com/documents> [Accessed 20 Mar. 2017].

Online converter trialled for use of converting the text file used for this project into XML

[21] sorel, I. (2017). Luc Sorel, downloads. [online] Lucsorel.com. Available at: <http://www.lucsorel.com/?page=downloads> [Accessed 20 Mar. 2017].

The source of the tool used to convert Weka text files to XML

[22] W3schools.com. (2017). CSS3 Reference. [online] Available at: [https://www.w3schools.com/cssref/css3\\_browsersupport.asp](https://www.w3schools.com/cssref/css3_browsersupport.asp) [Accessed 1 May 2017].

A reference of CSS properties and their compatibilities with browsers

[23] Free Code Camp. (n.d.). Learn to code and help nonprofits. [online] Available at: <https://www.freecodecamp.com> [Accessed 5 Feb. 2017].

This site was used extensively during the learning process of the web based aspects of this project

[24] Wells, D. (1999). Extreme Programming: A Gentle Introduction. [online] [Extremeprogramming.org](http://www.extremeprogramming.org/). Available at: <http://www.extremeprogramming.org/> [Accessed 3 May 2017].

An introduction to eXtreme Programing and the rules and practices associated with it

[25] Scrumguides.org. (2017). Scrum Guide | Scrum Guides. [online] Available at: <http://www.scrumguides.org/scrum-guide.html#events-planning> [Accessed 2 May 2017].

An online guide summarising the agile methodology of Scrum and its practices.

[26] www.tutorialspoint.com. (n.d.). SDLC Waterfall Model. [online] Available at: [https://www.tutorialspoint.com/sdlc/sdlc\\_waterfall\\_model.htm](https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm) [Accessed 7 May 2017].

An explanation of the Waterfall development methodology and its principles

[27] 11th Annual State of Agile Report. (2017). 1st ed. [pdf] VersionOne, p.10. Available at: <https://explore.versionone.com/state-of-agile/versionone-11th-annual-state-of-agile-report-2> [Accessed 2 May 2017]

Report detailing the use and adoption of agile practices within businesses

[28] Trello.com. (2017). About | What is Trello? [online] Available at: <https://trello.com/about> [Accessed 1 May 2017].

An explanation of Trello and its history

[29] Sherratt, E. (2017). Discipline in Agile Development. [lecture]

A presentation given about discipline in agile development and practices that can be used to maintain discipline

## Design

[30] Fowler, M. (2015). bliki: Yagni. [online] martinfowler.com. Available at: <https://martinfowler.com/bliki/Yagni.html> [Accessed 3 May 2017].

An explanation of the phrase YAGNI and its deeper meaning

[31] Dunham, E. (2013). Bellota Font Free by Pixilate » Font Squirrel. [online] FontSquirrel.com. Available at: [https://www.fontsquirrel.com/fonts/bellota?filter\[download\]=local](https://www.fontsquirrel.com/fonts/bellota?filter[download]=local) [Accessed 15 Feb. 2017].

This is the link to the download of the font that was imported into style.css and used in the main body of the website.

## Implementation

[32] Mozilla Developer Network. (2017). FileReader. [online] Available at: <https://developer.mozilla.org/en/docs/Web/API/FileReader> [Accessed 21 Apr. 2017].

Link to the FileReader API that was initially used to identify how to read in a file with JavaScript

[33] West, M. (2017). HTML5 FileReader API Demo: Text. [online] CodePen. Available at: <http://codepen.io/matt-west/pen/KjEHg> [Accessed 16 Feb. 2017].

This is an example of how a file can be read into Javascript, this was the basis for my own implementation of the JavaScript FileReader API.

[34] West, M. (2013). Reading Files Using The HTML5 FileReader API - Treehouse Blog. [online] Treehouse Blog. Available at: <http://blog.teamtreehouse.com/reading-files-using-the-html5-filereader-api> [Accessed 26 Apr. 2017].

[35] W3schools.com. (n.d.). JavaScript HTML DOM. [online] Available at: [https://www.w3schools.com/js/js\\_htmlDOM.asp](https://www.w3schools.com/js/js_htmlDOM.asp) [Accessed 26 Apr. 2017].

Explanation of the HTML Document Object Model

[36] W3schools.com. (n.d.). JavaScript DOM Nodes. [online] Available at: [https://www.w3schools.com/js/js\\_htmlDOM\\_nodes.asp](https://www.w3schools.com/js/js_htmlDOM_nodes.asp) [Accessed 26 Apr. 2017].

Explanation and examples of how to create a new node and add it to the DOM

[37] Bostock, M. (2017). D3.js - Data-Driven Documents. [online] D3js.org. Available at: <https://d3js.org/> [Accessed 26 Apr. 2017].

[38] Cook, P. (n.d.). D3 Tree. [online] Animateddata.co.uk. Available at: <http://animateddata.co.uk/lab/d3-tree/> [Accessed 26 Apr. 2017].

A data visualisation library that was considered for use in this project

[39] Mbraak.github.io. (2017). jqTree. [online] Available at: <https://mbraak.github.io/jqTree/#examples> [Accessed 26 Apr. 2017].

Another library that was considered for use in this project

[40] W3schools.com. (n.d.). AJAX Introduction. [online] Available at: [https://www.w3schools.com/xml/ajax\\_intro.asp](https://www.w3schools.com/xml/ajax_intro.asp) [Accessed 26 Apr. 2017].

An explanation of Ajax, its use and examples

[41] Fperucic.github.io. (2015). Treant.js - javascript library for drawing tree diagrams. [online] Available at: <https://github.com/fperucic/treant-js> [Accessed 22 Mar. 2017].

This is the link to the download and documentation for the library that was used to generate the decision tree structure.

[42] Mozilla Developer Network. (2017). Array.prototype.indexOf(). [online] Available at: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Array/indexOf?v=example](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/indexOf?v=example) [Accessed 27 Apr. 2017].

Explanation of JavaScript's indexOf method and the parameters it takes

[43] Mozilla Developer Network. (2017). Array.prototype.slice(). [online] Available at: [https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global\\_Objects/Array/slice?v=example](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Array/slice?v=example) [Accessed 27 Apr. 2017].

Explanation of JavaScript's slice method and the parameters it takes

[44] GitHub. (2014). rudi-c/weka-json-parser. [online] Available at: <https://github.com/rudi-c/weka-json-parser> [Accessed 27 Apr. 2017].

Python program used to reformat the J48 decision tree example I had been using

[45] Gist. (2014). Convert Weka J48 decision tree output to structured JSON. [online] Available at: <https://gist.github.com/fairlight1337/5366de28a6ae9316715d> [Accessed 27 Apr. 2017].

Another python program that was tried to convert the J48 decision tree to JSON

[46] GitHub. (2016). tsayen/dom-to-image. [online] Available at: <https://github.com/tsayen/dom-to-image> [Accessed 17 Apr. 2017].

This links to the library that was used to try and convert the element that is populated by the tree into an image so that other functions could be applied to it more easily.

[47] GitHub. (2016). eligrey/FileSaver.js. [online] Available at: <https://github.com/eligrey/FileSaver.js/> [Accessed 3 May 2017].

This links to the library that was used to download the image of the generated tree as a PNG file.

## Testing

[48] Martin, J. (2017). Here are the best web browsers to use, including alternatives to the big names. [online] PC Advisor. Available at: <http://www.pcadvisor.co.uk/test-centre/software/best-web-browsers-for-2017-3635255/> [Accessed 4 May 2017].

A web article discussing the most popular web browsers in 2017

## Evaluation

[49] www.tutorialspoint.com. (n.d.). Python GUI Programming (Tkinter). [online] Available at: [https://www.tutorialspoint.com/python/python\\_gui\\_programming.htm](https://www.tutorialspoint.com/python/python_gui_programming.htm) [Accessed 6 May 2017].

Tutorial explaining the basic functionality and use of Tkinter, Python's library for GUI creation

[50] Electron. (n.d.). Build cross platform desktop apps with JavaScript, HTML, and CSS. [online] Available at: <https://electron.atom.io/> [Accessed 6 May 2017].

Link to the Electron site, this provides further explanation about the framework, access to the download and tutorials.

[51] CodePen Blog. (2017). What Should I Do If Someone Copies My Work? - CodePen Blog. [online] Available at: <https://blog.codepen.io/documentation/faq/copied-work/> [Accessed 18 Apr. 2017].

This explains the licensing around the use of others work that has been publically published on the CodePen platform.