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)

8th May 2017

Version 1.0 (Draft)

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 …………………………………………

Date ……………………………………………

**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 …………………………………………

Date ……………………………………………

**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 proofread 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 tutor’s, who never gave up on me, even when I wanted to give up on myself.

**Abstract**

Include an abstract for your project. This should be no more than 300 words.

**Contents**

1. List of Figures

1. Background, Analysis & Process........................................................ 4

1.1. Background................................................................................................... 4

1.2. Analysis.......................................................................................................... 4

1.3. Process.......................................................................................................... 4

2. Design.................................................................................................................. 4

2.1. Overall Architecture.................................................................................. 4

2.2. Detailed Design............................................................................................. 4

2.2.1. Even More Detail..................................................................................... 4

2.3. User Interface Design................................................................................. 4

2.4. Other Relevant Sections............................................................................ 4

3. Implementation.............................................................................................. 4

4. Testing............................................................................................................... 4

4.1. Overall Approach to Testing..................................................................... 4

4.2. Automated Testing....................................................................................... 4

4.2.1. Unit Tests................................................................................................. 4

4.2.2. User Interface Testing............................................................................. 4

4.2.3. Stress Testing.......................................................................................... 4

4.2.4. Other Types of Testing............................................................................ 4

4.3. Integration Testing...................................................................................... 4

4.4. User Testing.................................................................................................. 4

5. Critical Evaluation..................................................................................... 4

6. Appendices........................................................................................................ 4

A. Third-Party Code and Libraries.................................................................. 4

B. Ethics Submission......................................................................................... 4

C. Code Samples................................................................................................ 4

Annotated Bibliography................................................................................. 4

List of Figures

**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 implementation [2]. However, the decision tree outputs provided by Weka are outputted, as plain text, as seen in figure X, which makes them visually difficult to interpret and unsuitable for use in academic publications.

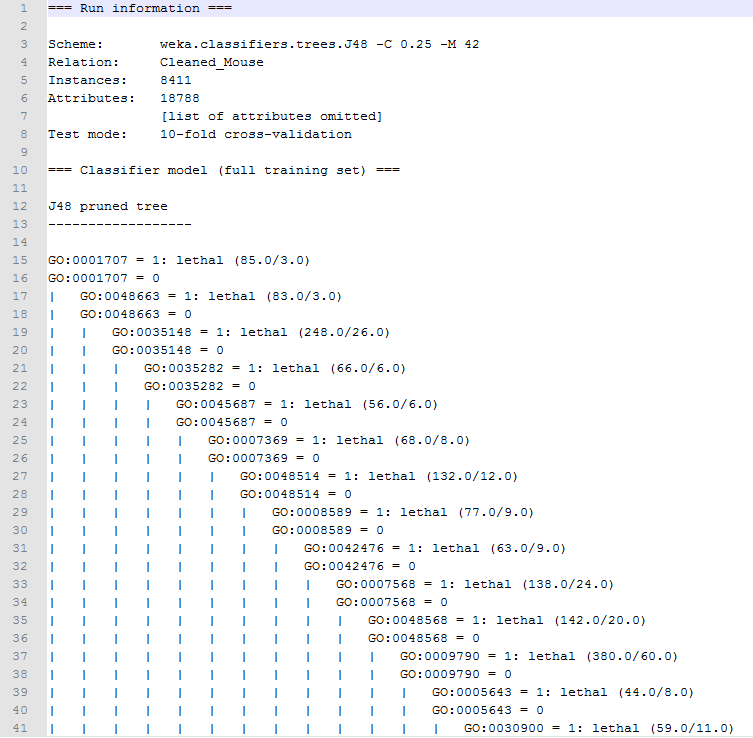


Figure X: An example of part of a Weka J48 decision tree output, viewed in Notepad++

Currently, it does not appear that there is any specific application that takes Weka text outputs and converts them into a visual tree which can then be used as images. Weka does make it possible to view the text output in a more traditional treelike structure, as can be seen in figure X. However, this does not scale well with larger datasets 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 this. This would be unrealistic to do with a larger tree as key details could be lost in the attempt of capturing the whole tree.

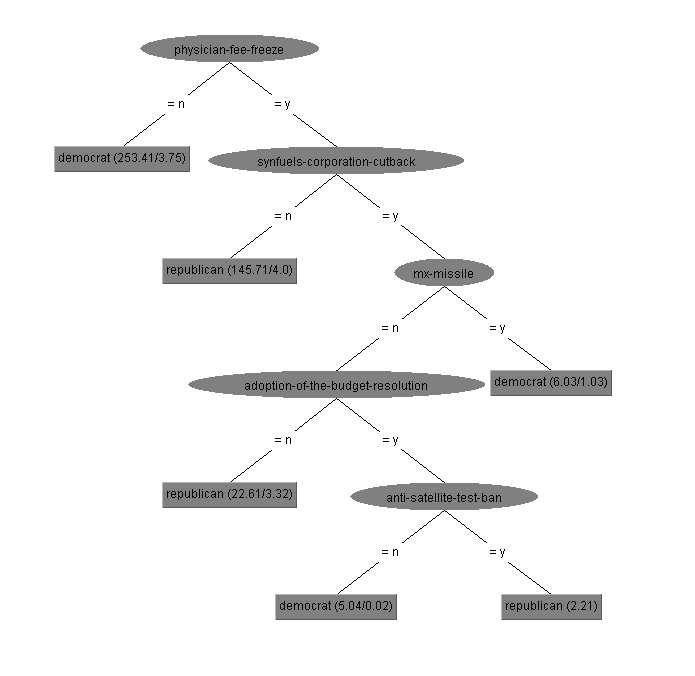


Figure X: A screenshot example of a visualised J48 decision tree generated by Weka

There do exist 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 []. 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 []. One such example of a tree that was visualised based on data inputs was used to create a phylogenetic tree that is used to display the evolutionary relationships between organisms [].

However, the data used to generate the tree was completely different from the data provided by Weka decision trees. Additionally, in this case, the tree itself was not a decision tree although it can share similarities in structure so was not an appropriate example to base this project on.

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 should 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, only the top three nodes were successfully created from the data that was processed by the application. Additionally, it was possible to download an image of the generated tree 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 further 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) [1] 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 easier. A number of extensions were used in this project to make the coding process smoother, the first such extension was Beautify, [2] which was used to reformat the code to make it more readable and easy to understand. The other extension used was HTML Snippets [3] 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 []. It provides useful customisable set up for a web project, 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 option 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 pregenerated would save time. However, one requirement of use was that any markup changes would have to be made using Jade[], 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 Maga Boilerplate were not necessary to the project such as a database and server support the program could be run on client side. 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 trialed use can be found in the appendix of this document.

**CodePen:** CodePen is a virtual space which allows new code or software to be tested and run securely [], 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 code 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 APIs (Application Programming Interface) 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.

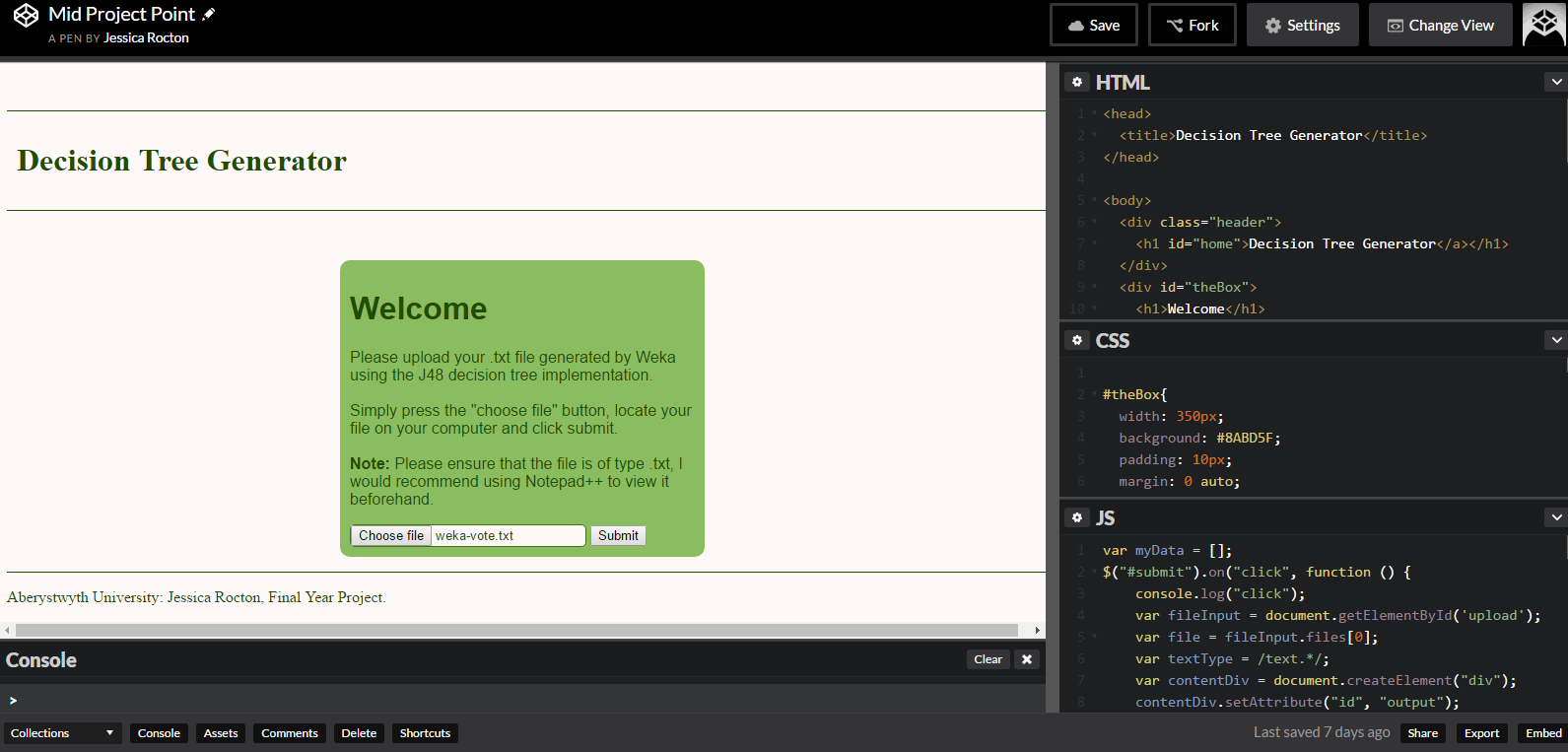


Figure X: Image shows the layout of Codepen along with some features that it offers

**GitHub:** GitHub is an open source version control system that is based on the command line tool Git []. 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 application’s ability to alter the document and make comments. This solves the problem of accidentally editing or providing feedback on an out-of-date document 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.

Another option for document generation that was considered was LaTeX. LaTeX is a document preparation tool that is commonly used in the creation of large/medium technical documents []. LaTeX also has a real time, collaborative editor known as ShareLaTeX [], which could have been used instead of Google Docs if this option had been selected. Microsoft Word was ultimately selected for its familiarity as 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 Website vs 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. 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, it was decided that a web app was a better fit as it would provide a better user experience and allow the user to interact with the output and manipulate it before deciding to commit to downloading it. This would ensure that the user is satisfied with the visual layout of the tree and can clearly see what the output will look like. 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 experience. These are capabilities that would not have been as effective to implement if creating a command line tool.

1.2.2 Text vs XML

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 [1]. 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. Initially, an online Text to XML converter was trialed [2] however, the output from this was unhelpful as it simply enclosed each line in <p> tags, this can be seen in figure X. 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.



Figure X: 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 [3] was used with a limited extent of 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 X.

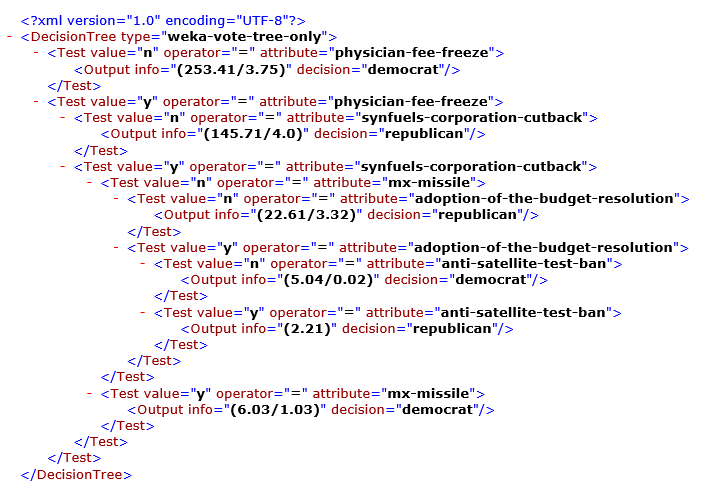


Figure X: This figure shows the output that was provided by the WekaText2XML program when run with the test set of data

While the application did fundamentally achieve what it needed to, it was time consuming and fiddly to use which is less than ideal for users. The biggest problem with this solution was that the software was not open source, but freeware 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 to the proposed website. As a result, 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 should not be pursued as a part of this project.

1.2.3 Learning Undertaken

Before being able to proceed with this project 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, CSS3 (Cascade Style Sheet), 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 [] which made them a sensible choice of technologies when considering browser compatibility.

This included various tutorials on HTML, CSS, JQuery and JavaScript, these were largely undertaken via FreeCodeCamp.com [] 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 decided solution would aim to deliver.

1. A function 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 development of the project could commence, it was important to decide on which development methodology would be used to support the projects growth, 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) and Scrum to plan driven methods such as Waterfall, 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. Reports have shown that the most common techniques employed by business who use Agile are:

* Iteration planning
* Daily standup
* 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 superviser 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 tool that was found to be particularly useful for iteration planning was Trello [], 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 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 X 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) [] and was partially inspired by this. It does differ to an extent, as it does not expressly state due dates, however the general purpose remains the same it that it was a useful tool for tracking progress at a glance but also at a more in-depth level as was used in the planning of this project.

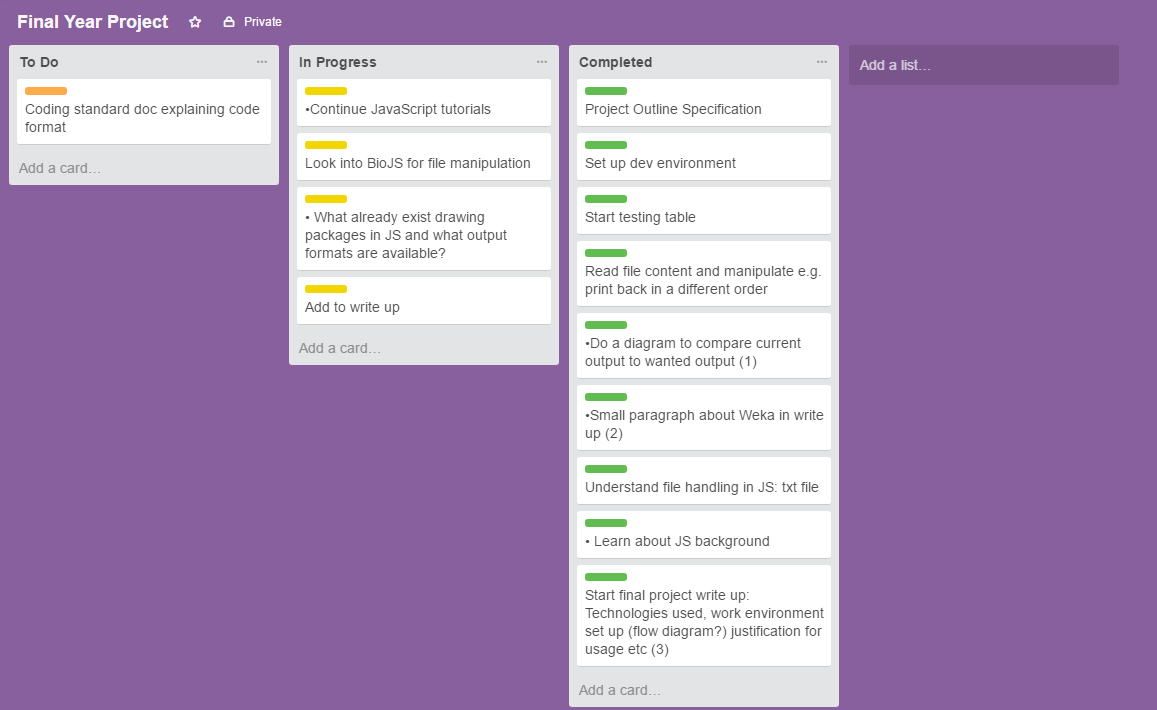


Figure X: This image shows a snapshot of Trello being used during this project

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 Standup that are associated with specific methods were just 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**

You should also identify any support tools that you used. You should discuss your choice of implementation tools - programming language, compilers, database management system, program development environment, etc.

Some example sub-sections may be as follows, but the specific sections are for you to define.

## **2.1.** **Overall Architecture**

## **2.2.** **Detailed Design**

### **2.2.1.** **Even More Detail**

## **2.3.** **User Interface Design**

Figure X shows the main page of the site. The design is purposely simple as the role of the site is purely to take a file and then display an output generated by the file, for this reason there was little need to make it more complex than necessary.

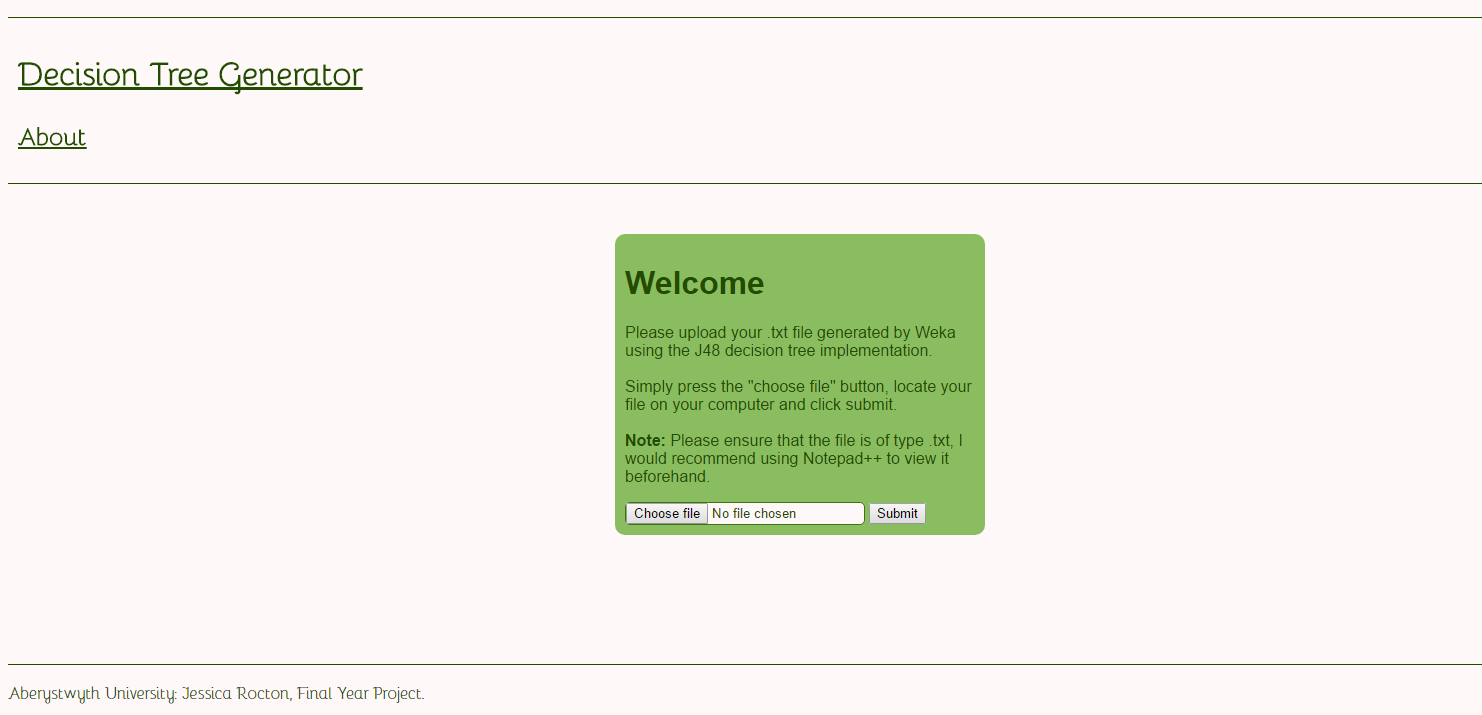


Figure X: This image shows the landing page of the site and will be the initial point of contact between the user and the program

A number of different shades of green were used and were selected as it related to the theme of trees. The main font used is called Bellota-Light and was imported from an eternal source [] by downloading the oft file and including the following code in the main style sheet.

@font-face {

font-family: Bellota-Light;

src: url("Bellota-Light.otf") format("opentype");

}



Figure X: Depicting the output of the program once run and will be what the user’s result looks like.

## **2.4.** **Other Relevant Sections**

# **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 User Interface (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 [] 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 that would trigger the read function in the event that the submit button was clicked. It was necessary to determine where the file input is coming from by assigning the HTML element that is 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 X, 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 []. From this point the data is now able to be accessed and manipulated.

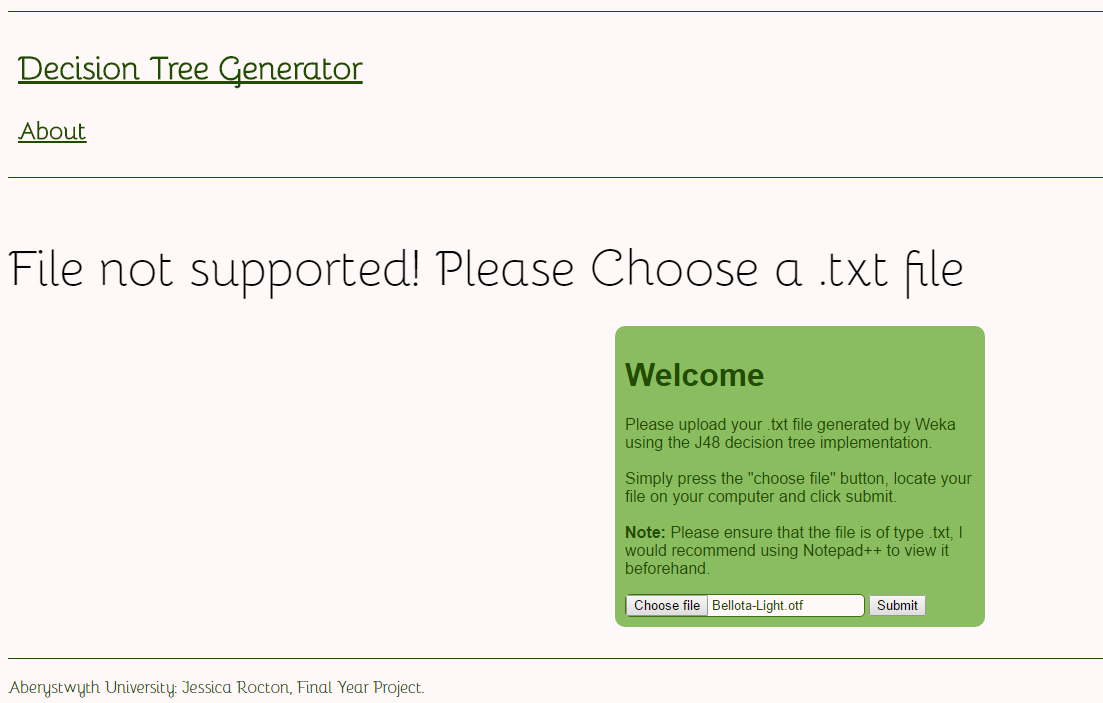


Figure X: Screenshot showing the error message the user will receive if they try to upload an incorrect file type

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, so 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 index to an array. Therefore, a new array was created and populated by taking the result from reader and using the split method to break up the file on the space character and then add each index 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 done by cycling through the array starting with the last element and assigning it to a variable, then adding this variable 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 be 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 DOM (Document Object Model) []. The DOM is the standard by which HTML elements can be accessed and manipulated by programs and scripts []. 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 elements that were previously visible to the user before submitting the file. In this way, the user would only then see the output of the file displayed to 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” [] 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 [] 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[], 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. JSON is a way that data can be formatted using 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 and web page and a server [] 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 [], 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 as 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.

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 be being 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 X and X 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 file though would appear at different lines 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 that would be 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 but instead it was 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 J48 outputs sometimes had different newline characters which meant that sometimes splitting the file wouldn’t work. To remedy this is was necessary to standardise the new line character in all files. This was done by changing the setting of the file in Notepad++. This was done by selecting “Edit”, “EOL Conversion” and choosing Unix (LF). Once this had been remedied, the index of the phrases were then searched using JavaScript’s indexOf method [], as it was known that while the indexes may change from file to file, the spacing between them and the actually tree was consistent. These indexes were then saved to new a variable as they signified the start and end points of the data that needed to be extracted. JavaScript’s slice method [] 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 trialed 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 at <http://codepen.io/jessijinx/pen/mWQJbo>. Just 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 so there would be no consistency and it would not be reliable to assign a static index to the library.

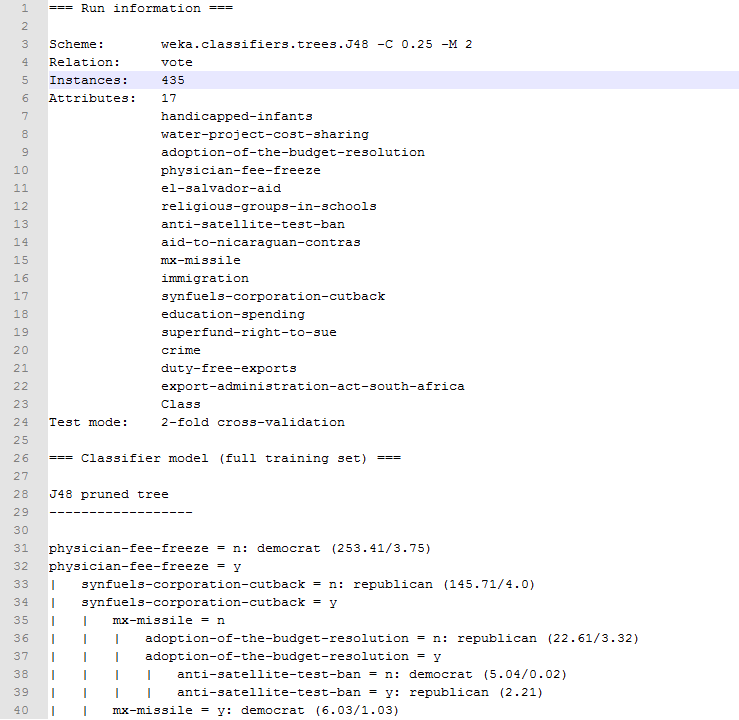


Figure X: This figure shows the first part of the text output provided by Weka, displayed in Notepad++

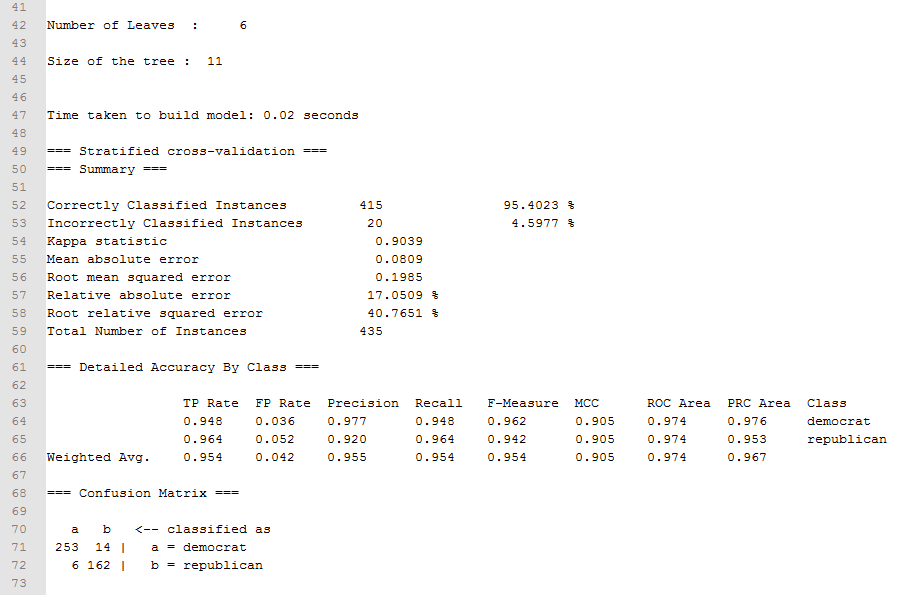


Figure X: This shows the second part of the text output provided by Weka, displayed in Notepad++

Once it became apparent that using the data in the format it was originally in was not ideal one option that was explored was to manually reformat the text file. This was attempted by converting the file 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 a 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 was tried but would not work even after updating the syntax. Another application named ParseJ48.py did exist and was designed specifically for converting J48 decision trees into JSON []. 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 X shows the output of the conversion program, beautify was used on the file to make it easier to read as the original output in all on one line.

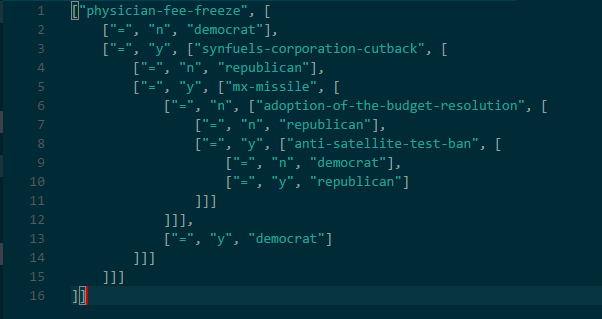


Figure X: The output from the use of ParseJ48.py after being opened in Visual Studio Code and having Beautify applied to it.

While the output is clearly not JSON as was indicated by the program it did appear that the data had been separated into an array and it was hoped that this output could still be used and 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 the file could not be processed as it 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 as an array. It does not appear that there is a way to read in the content of a file as an array so it would have to be split like any other strong. This returned to a previous problem of unwanted characters.

In the end, while it wasn’t possible to generate a tree from the whole file, it was possible to generate the first three nodes. This was only possible however by hard coding the indexes of the values that were needed and it would not be compatible with other files.

3.6 Additional Functionality

In order to improve the user’s experience, additional functionality was added to the program. This takes the form of a download function and a zoom function. The download function was implemented by using the DOM-to-image library [] which allows for the conversion of a HTML element into a image. When the download button is clicked, the DOM node is converted into a JPEG and downloaded to the user’s computer. While this does work inasmuch as an image of the basic tree is downloaded, figure X 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.

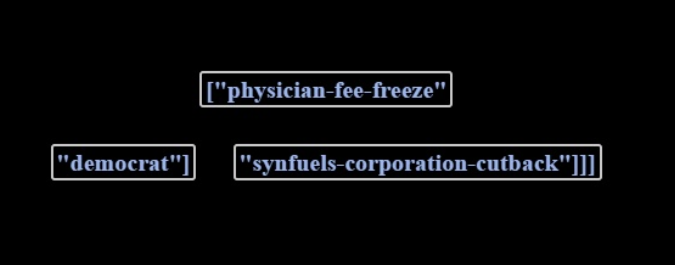


Figure X: The result of the downloaded file as viewed in Microsoft Photos (version 17.313.10010.0)

The implementation should look at any issues you encountered as you tried to implement your design. During the work, you might have found that elements of your design were unnecessary or overly complex; perhaps third party libraries were available that simplified some of the functions that you intended to implement. If things were easier in some areas, then how did you adapt your project to take account of your findings?

It is more likely that things were more complex than you first thought. In particular, were there any problems or difficulties that you found during implementation that you had to address? Did such problems simply delay you or were they more significant?

You can conclude this section by reviewing the end of the implementation stage against the planned requirements.

# **4.** **Testing**

Detailed descriptions of every test case are definitely not what is required here. What is important is to show that you adopted a sensible strategy that was, in principle, capable of testing the system adequately even if you did not have the time to test the system fully.

Have you tested your system on ’real users’? For example, if your system is supposed to solve a problem for a business, then it would be appropriate to present your approach to involve the users in the testing process and to record the results that you obtained. Depending on the level of detail, it is likely that you would put any detailed results in an appendix.

The following sections indicate some areas you might include. Other sections may be more appropriate to your project.

## **4.1.** **Overall Approach to Testing**

Browser compatability

# **5.** **Critical Evaluation**

Examiners expect to find in your dissertation a section addressing such questions as:

· Were the requirements correctly identified?

· Were the design decisions correct?

· Could a more suitable set of tools have been chosen?

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

· How well were any other project aims achieved?

· If you were starting again, what would you do differently?

Electron\*\*\*\* future consideration

Such material is regarded as an important part of the dissertation; it should demonstrate that you are capable not only of carrying out a piece of work but also of thinking critically about how you did it and how you might have done it better. This is seen as an important part of an honours degree.

There will be good things and room for improvement with any project. As you write this section, identify and discuss the parts of the work that went well and also consider ways in which the work could be improved.

Review the discussion on the Evaluation section from the lectures. A recording is available on Blackboard.

# **6.** **Appendices**

A. Third-Party Code and Libraries

If you have made use of any third party code or software libraries, i.e. any code that you have not designed and written yourself, then you must include this appendix.

As has been said in lectures, it is acceptable and likely that you will make use of third-party code and software libraries. The key requirement is that we understand what is your original work and what work is based on that of other people.

Therefore, you need to clearly state what you have used and where the original material can be found. Also, if you have made any changes to the original versions, you must explain what you have changed.

**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\*\*

**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 []. 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.

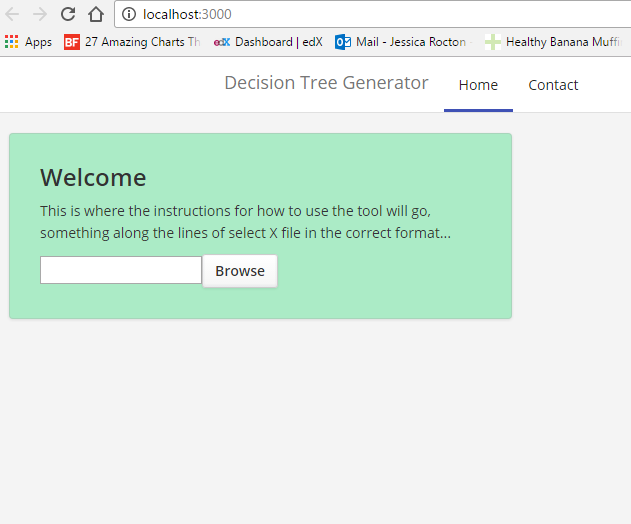


Figure X: This figure shows the original site concept when it was generated using Mega Boilerplate



Figure X: 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

This appendix includes a copy of the ethics submission for the project. After you have completed your Ethics submission, you will receive a PDF with a summary of the comments. That document should be embedded in this report, either as images, an embedded PDF or as copied text. The content should also include the Ethics Application Number that you receive.

C. Code Samples

This is an example appendix. Include as many appendices as you need. The appendices do not count towards the overall word count for the report.

# 

# 

# **Annotated Bibliography**

**Analysis**

[1] Weka 3: Data Mining Software in Java, Machine Learning Group at the University of Waikato, 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] Class J48 Available at: <http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/J48.html> [Accessed 9/3/17]

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

BioJS

[] About BioJS Available at:<https://biojs.net/about/> [Accessed 7/2/17]

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

[] 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.

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.

[] 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 trees.

Technologies used

<https://code.visualstudio.com/>

accessed 7/2/17

<https://marketplace.visualstudio.com/items?itemName=abusaidm.html-snippets> [Accessed 8/2/17]

<https://marketplace.visualstudio.com/items?itemName=HookyQR.beautify> [Accessed 8/2/17]

Mega boilerplate

[] <https://github.com/sahat/megaboilerplate> accessed 6/2/17

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

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

An explanation of Jade and it’s use

CodePen

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

Explains the basic principle and function of CodePen

GitHub

[] 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].

Google Docs

[] 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

[] 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

Text vs XML

[1] 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.

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

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

[3] sorel, l. (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

Learning Undertaken

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

A reference of CSS properties and their compatibilities with browsers

[1] <https://www.freecodecamp.com>

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

Development Methodology

[]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

[] 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.

**Design**

[] 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

[] Dunham, E. (2013). Bellota Font Free by Pixilate » Font Squirrel. [online] Fontsquirrel.com. Available at: 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**

Read in the text file

[1] 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

[2] 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.

[3] 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].

Manipulate the text file and print to screen

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

Explanation of the HTML Document Object Model

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

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

Implementation of Chosen Library

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

[] 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

[] 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

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

An explanation of ajax, it’s use and examples

[] Treant.js available at: <http://fperucic.github.io/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.

Data reformatting

[] 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 [Accessed 27 Apr. 2017].

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

[] 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 [Accessed 27 Apr. 2017].

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

[] 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

[] 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

Additional Functionality

[1] 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.

**Appendices**

3rd party libraries

[1] Treant.js available at: <http://fperucic.github.io/treant-js/> [Accessed 22 Mar. 2017].

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

[2] 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.

[3] 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.

[4] 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.