UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Decision Tree Simulator Technical documentation

Presentado por Daniel Drefs Fernandes
en Universidad de Burgos — June 11, 2024
Tutores: Carlos López Nozal
Ismael Ramos Pérez

Contents

List of Figures

List of Tables

Listings

Appendix A

Software Project Plan

A.1 Introduction

This section presents how time management was handled in the course of this project. Throughout the development, bi-weekly meetings were held that served the purpose of reviewing what had been done and discussing what the tasks for the next sprint would be. To give better insight, a burndown of every sprint will be displayed. They are automatically created by Zube, the project agility tool that was used. The issues which these graphs are based on are all found in the "Issues" section[1] of the project's GitHub repository.

A.2 Time planning

Sprint 1 (29/02/2024 - 13/03/2024): Kick off project

Objectives: The main objectives of this sprint were to set up the Github repository structure, link it to Zube for a better overview of each sprint's tasks, learn about decision trees and to create a first web application displaying a tree using SVG.

Results: Almost all the tasks that were intended for this sprint were completed, except for the documentation of the Decision Trees concept in the Memoria.

Figure A.1 shows the burndown of the sprint.

[1] GitHub issues: https://github.com/danieldf01/TFG-decision-trees-sim/issues

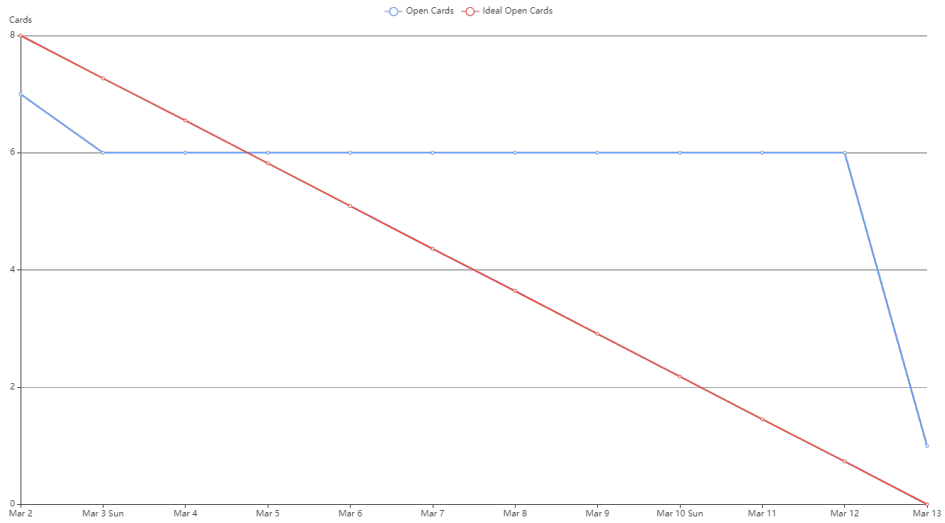


Figure A.1: Burndown Sprint 1

Sprint 2 (14/03/2024 - 03/04/2024): Implementation of tree graphics

Objectives: For this sprint, the intention was to create the first two prototypes, one displaying the entropy function with a calculator and the other one displaying a decision tree, both making use of the D3.js library. To display these prototypes, a GitHub Pages repository was to be created. Solidifying knowledge about conditional entropy and making entries to the "Theoretical concepts" section of the Memoria were also part of this sprint.

Results: As seen on the burndown in figure A.2, everything was completed except for the prototype displaying a decision tree. Due to sickness during the sprint, this task was left unfinished and pushed back to a later sprint for the time being.

Figure A.2: Burndown Sprint 2

Sprint 3 (04/04/2024 - 17/04/2024): Prototype for conditional Entropy

Objectives: During this sprint, the main tasks were to refactor the GitHub repository structure, upgrade the visual presentation of the Entropy prototype using the Bootstrap framework, start documenting technical tools used in the Memoria and to create a prototype displaying a calculator for conditional Entropy.

Results: As figure A.3 shows, all the tasks of this sprint were completed in time.

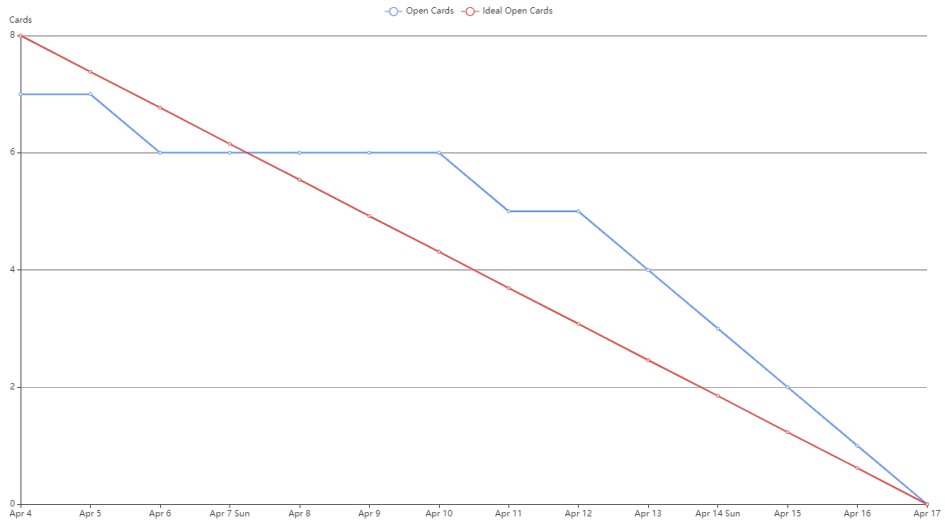


Figure A.3: Burndown Sprint 3

Sprint 4 (18/04/2024 - 02/05/2024): Prototype Decision Tree

Objectives: The main tasks of this sprint were to, on one hand, improve the existing prototypes with exceptions and enhance the overall code quality and, on the other hand, create a prototype that displays a decision tree based on an example dataset. Besides that, it was also asked to continue working on the Memoria by documenting some technical environments that were used.

Results: As seen in figure A.4, all tasks were completed except for two issues regarding the documentation of related works and a theoretical concept. This shortcoming was due to time constraints caused by assignments and exams in other classes.

Figure A.4: Burndown Sprint 4

Sprint 5 (03/05/2025 - 16/05/2024): step-by-step Decision Tree simulation

Objectives: This sprint's main objective consisted of implementing a step-by-step visualization for the decision tree prototype that was created in the previous sprint. To achieve that, the decision tree creation had to be made dynamic, which, at the time, it was not. Other tasks included the creation of a header and footer for the web application and documenting relevant aspects of the development.

Results: Figure A.5 displays this sprint's burndown which shows that all the proposed tasks were done in time.

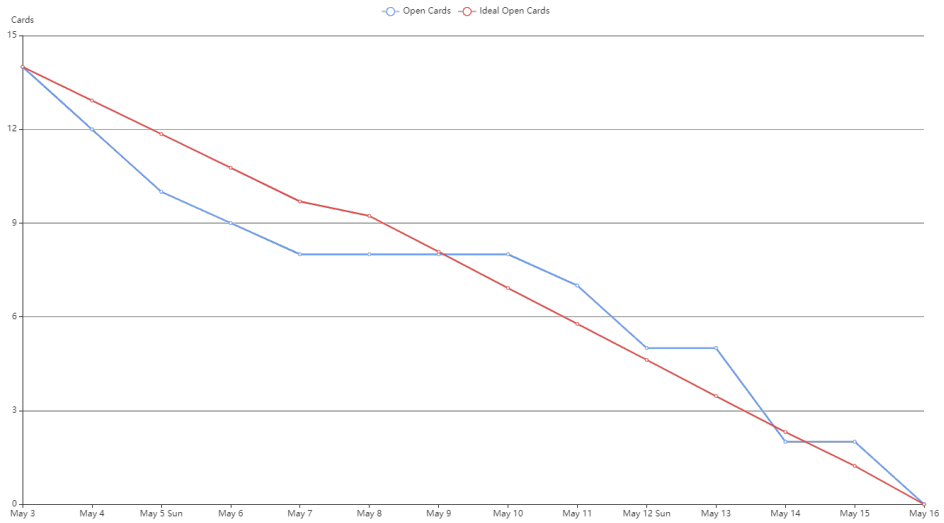


Figure A.5: Burndown Sprint 5

Sprint 6 (17/05/2024 - 30/05/2024): Decision Tree value table, CSV data loading, interactive data

Objectives: One of this sprint's main goals was to upgrade the decision tree's prototype by adding a dynamic value table that would display relevant values, like each feature's information gain, at each step. The other main objectives were to make it possible for the user to use their own datasets in CSV file format and to allow them to add and remove rows and columns from a currently loaded dataset.

Results: Figure A.6 shows that, due to the sprint having been during the final exam phase, not all tasks were completed. Besides issues like scaling text sizes based on their width and a cleanup of the project layout, one of the main objectives was left unfinished. While the addition of user-uploaded CSV datasets was successful, the "interactive data" goal was not met. In the end, it was discarded altogether as other refinements took priority due to the lack of time.

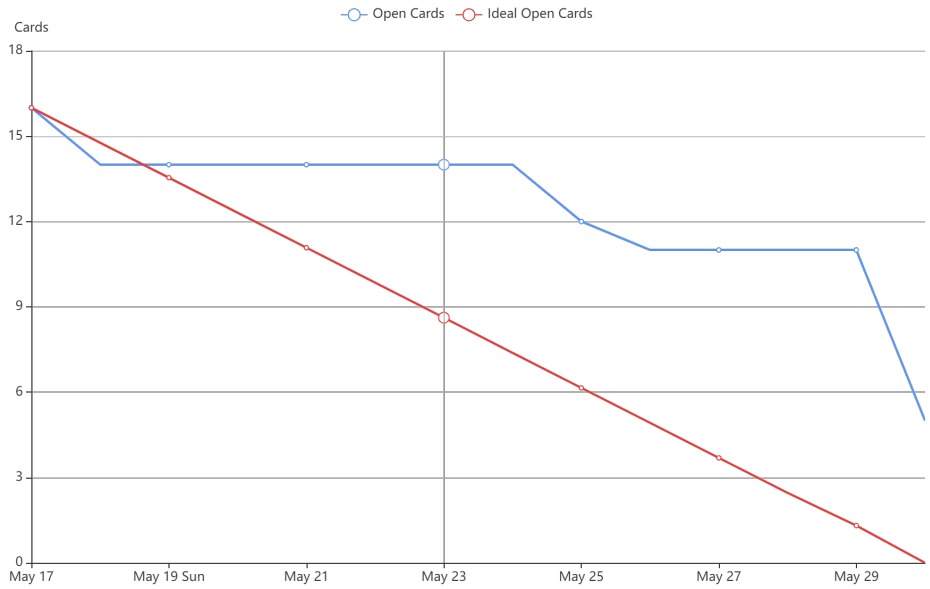


Figure A.6: Burndown Sprint 6

Sprint 7 (31/05/2024 - 06/06/2024): Decision Tree selectable example data, CSV file requirements

Objectives: The final sprint of this project's development was used to refine some of the already existing parts of the application. One issue was to add the functionality of being able to choose between different example datasets for the decision tree ID3 simulation. Another was to formulate requirements that a user-chosen CSV dataset had to meet and display them.

Results: Figure A.7 shows a burndown of the final sprint. As this sprint still took place during the exam phase, not all tasks could be finished here either. However, those were only minor issues like an improvement of the repository's README file which could were completed in the final days before the deadline.

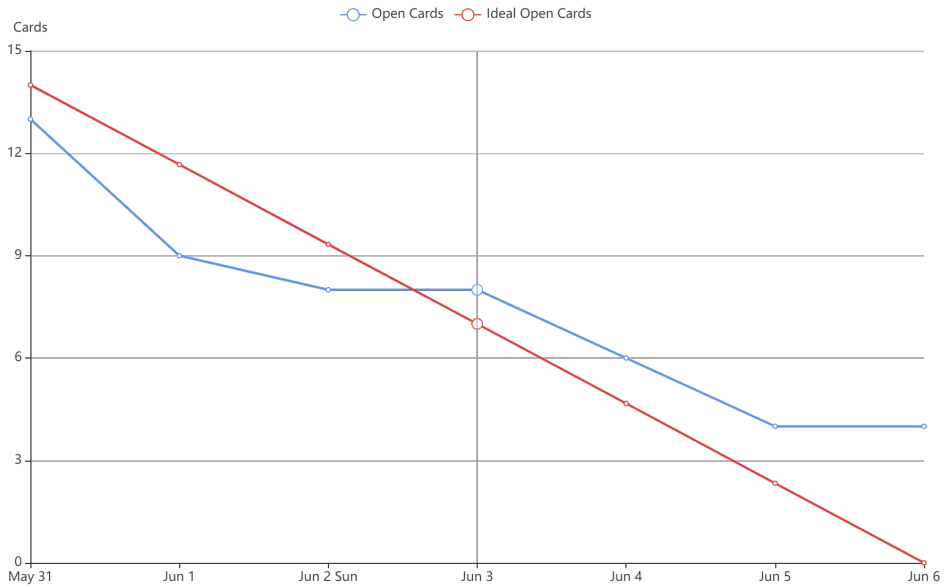


Figure A.7: Burndown Sprint 7

Overview

Figure A.8 displays an overview of the amount of issues that were resolved during each sprint.

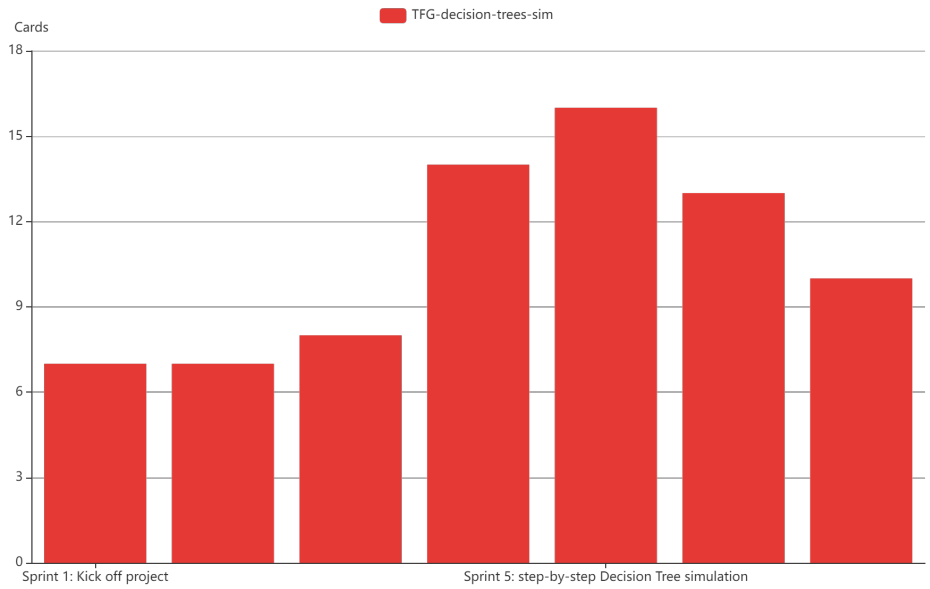


Figure A.8: Sprints overview

The workload was visibly increased starting at sprint 4 (A.2), which marked the start of the implementation of the decision tree prototype. Before that point, a lot of the sprints' work included learning and getting used to new technologies in order to be able to use them. Most of this process was not captured in the form of issues.

A.3 Viability study

Economic viability

Staff costs

This project has been carried out by one programmer over the course of approximately 3 months. Taking into consideration the developer's status of full-time student who also had to spend time in preparation for other classes, the overall development time can be estimated to a part-time employment. Considering the monthly minimum wage for general workers in Spain in 2024 of 1134,00€ per month [5], the estimated salary would be:

$$3m * 1134€/m = 3402€$$

As this represents only the gross wage, social security contributions have to be added, as well. As of 2024, these make up 36,85%, with the employee having to contribute 6,45% and the employer 30,4% [4]. Adding this to the gross wage totals up to:

$$3402€ + 3402€ * 0,3685 = 4655,637€$$

Software and hardware costs

The software cost of this project is equal to 0 as only free software has been used. As for hardware, a device with a value of 863,27€ has been used to carry out this project. Depreciation does not have to be considered as the device was bought at the beginning of development. Having used this device for the project over the course of 3 months, it makes up for a total cost of:

$$3m * 863,27€/m = 2589,81€$$

Total cost

Considering staff and hardware costs together, the total cost is summed up to:

$$2589,81€ + 4655,637€ = 7245,447€$$

Legal viability

Table A.1 shows every used dependency, its version and license

Dependency	Version	License
jQuery	3.7.1	MIT
jest	29.7.0	MIT
jest-environment-jsdom	29.7.0	MIT
D3	7.9.0	ISC
Bootstrap	5.3.3	MIT
bootstrap-icons	1.11.3	MIT
MathJax	3.2.2	Apache-2.0
Polyfill service	3.25.3	CC0-1.0
@popperjs/core	2.11.8	MIT
papaparse	5.4.1	MIT

Table A.1: Every dependency used in the project and its license

The most restrictive of these licenses would be the Apache-2.0 license, which is still a permissive license that allows, e.g., the software being used, modified and distributed in a commercial context.

Appendix B

Requirements Specification

B.1 Introduction

This section will explain the requirements of the application by specifying (non-)functional requirements and use cases.

B.2 General objectives

The main objective of this project has been to create a web application under the name of "Decision Tree Simulator" and with the purpose of helping users learn the concept of decision trees, how they are created and all necessary surrounding topics in an intuitive and simple way.

It provides dynamic calculators for entropy and conditional entropy which let the user input values to they can observe how different values affect the results. There is also a visual representation of the binary entropy graph that uses SVG and responds with markers to the user's input, if they used two classes to calculate the entropy.

The decision tree ID3 simulation presents a step-by-step visualization of the ID3 algorithm so that each user can follow the steps at their own pace. They can choose between selecting one of the example datasets or loading their own dataset in CSV file format. With a combination of a decision tree that is dynamically created using SVG, a dataset table, a value table, and visual cues at each step, the goal was to make the user's learning experience simple and intuitive.

B.3 Requirements catalog

Functional Requirements

- **FR-1** From the web, it must be possible to run the Entropy calculator for calculating the entropy of given input values
  - **FR-1.1** The user must be able to enter values into the presented input fields which are positioned in the column that is given the name “Nr. of instances” by the respective column header.
  - **FR-1.2** The user must be able to add classes by clicking on the button labeled “+”.
  - **FR-1.3** The user must be able to remove the row that represents the class that was last added by clicking on the button labeled “-”.
  - **FR-1.4** The user must be able to initialize the calculation of the entropy by clicking on the button labeled “Calculate Entropy”.
  - **FR-1.5** The application must, given valid input values, correctly calculate each class’s p-value and the feature’s entropy and display those values on the corresponding Entropy table.
  - **FR-1.6** The application must, if only 2 classes were used, show the results of the entropy calculation through a red dot on the x-axis of the presented coordinate system and a red line pointing to the corresponding point on the presented Binary Entropy graph.

- **FR-2** From the web, it must be possible to run the Conditional Entropy calculator for calculating the conditional entropy of given input values.
  - **FR-2.1** The user must be able to enter values into the presented input fields which are positioned in the columns that are given the name “Class 1" and “Class 2" by the respective column headers.
  - **FR-2.2** The user must be able to add categories by clicking on the button labeled “+”.
  - **FR-2.3** The user must be able to remove the row that represents the category that was last added by clicking on the button labeled “-”.

  - **FR-2.4** The user must be able to initialize the calculation of the conditional entropy by clicking on the button labeled “Calculate Conditional Entropy”.
  - **FR-2.5** The application must, given valid input values, correctly calculate each category’s ratio, entropy, and the feature’s conditional entropy and display those values on the table.

- **FR-3** From the web, it must be possible to run the Decision Tree ID3 simulator for executing a step-by-step simulation of the ID3 algorithm.
  - **FR-3.1** The user must be able to choose a dataset from one of the example datasets that are provided by the web application.
  - **FR-3.2** Given that the button labeled "Load own CSV dataset" was clicked, the application must present a modal to the user in which the requirements for a CSV file and an input form are displayed.
  - **FR-3.3** The user must be able to select their own dataset in a CSV file format through an input form.
  - **FR-3.4** The application must, given a valid CSV file, load the dataset that is contained in the file.
  - **FR-3.5** The application must, following a successful load of a dataset, display an information card in regards to the chosen dataset, the root node of the dynamically created decision tree, a data table presenting the dataset, and a value table that presents values that are relevant to the decision tree’s creation at each step.
  - **FR-3.6** The user must be able to navigate through the step-by-step simulation with the use of the four buttons that represent the four functions “Initial step”, “Step back”, “Step forward”, and “Last step”, respectively.
  - **FR-3.7** The application must, given that the “Initial step” button was clicked by the user, go to the first step of the simulation.
  - **FR-3.8** The application must, given that the “Step back” button was clicked by the user and the simulation had not already been at the first step, go back one step in the simulation.
  - **FR-3.9** The application must, given that the “Step forward” button was clicked by the user and the simulation had not already been at the last step, go forward one step in the simulation.

  - **FR-3.10** The application must, given that the "Last step" button was clicked by the user, go to the last step of the simulation.

Non-functional requirements

- **NFR-1** The user interface must be simple and intuitive.
- **NFR-2** The application must be responsive to different screen sizes.
- **NFR-3** For the Entropy calculator and Conditional Entropy calculator, the application must recognize any positive integer value as valid input.
  - **NFR-3.1** The user must be warned through appearing alerts if any of the user-made inputs is invalid.
- **NFR-4** For the Entropy calculator, if more than 2 classes are used, the user must be informed through an appearing alert about the fact that the calculated results will not be displayed on the Binary Entropy graph.
- **NFR-5** For the Decision Tree ID3 simulator, the application must recognize CSV files that meet the file requirements that are displayed in the application as valid.
  - **NFR-5.1** The user must be warned through an appearing alert if the proposed CSV file fails to meet any of the requirements and is therefore recognized as invalid.

B.4 Requirements specification

Use case diagram

Figure B.1 displays the use case diagram.

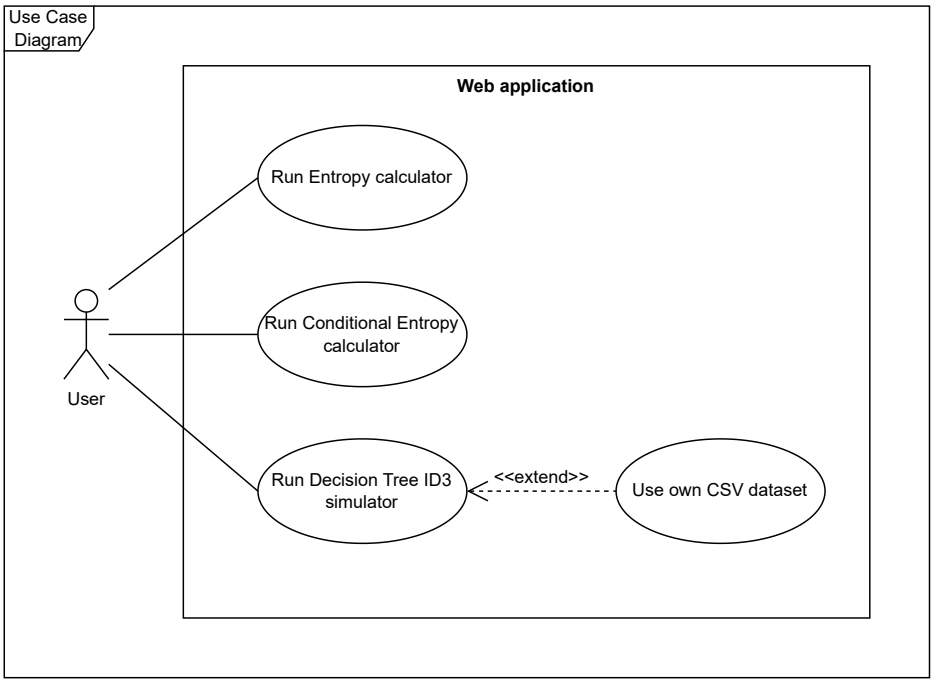


Figure B.1: Use case diagram

Use cases

The textual use cases are displayed in tables B.1, B.2, B.3, and B.4.

UC-1	**Run Entropy calculator**
Version	1.0
Author	Daniel Drefs Fernandes
Associated requirements	FR-1, FR-1.1, FR-1.2, FR-1.3, FR-1.4, FR-1.5, FR-1.6
Description	The user runs the Entropy calculator with the desired input values and receives the results in a visual format on the website.
Precondition	The input values introduced by the user are valid.
Actions	1. The user opens the application. a) The user adds one or multiple class by clicking the button with the label "+". 2. The user fills the input fields with the desired values and clicks on the button with the label "Calculate Entropy". 3. The application calculates each class's p-value and the feature's entropy and displays the results on the corresponding table. a) If the user has not added any classes, the application will show a visualization of the calculated results in SVG format on the Binary Entropy graph.
Postcondition	The results are displayed on the Entropy table.
Exceptions	If the user has introduced invalid values, the application will display an alert and inform the user to only use positive integer values.
Importance	High

Table B.1: UC-1 Run Entropy calculator.

UC-2	**Run Conditional Entropy calculator**
Version	1.0
Author	Daniel Drefs Fernandes
Associated requirements	FR-2, FR-2.1, FR-2.2, FR-2.3, FR-2.4, FR-2.5
Description	The user runs the Conditional Entropy calculator with the desired input values and receives the results in a visual format on the website.
Precondition	The input values introduced by the user are valid.
Actions	1. The user opens the application. a) The user adds one or multiple categories by clicking the button with the label "+". 2. The user fills the input fields with the desired values and clicks on the button with the label "Calculate Conditional Entropy". 3. The application calculates each category's ratio, entropy, and the feature's conditional entropy and displays the results on the table.
Postcondition	The results are displayed on the table.
Exceptions	If the user has introduced invalid values, the application will display an alert and inform the user to only use positive integer values.
Importance	High

Table B.2: UC-2 Run Conditional Entropy calculator.

UC-3	**Run Decision Tree ID3 simulator**
Version	1.0
Author	Daniel Drefs Fernandes
Associated requirements	FR-3, FR-3.1, FR-3.4, FR-3.5, FR-3.6, FR-3.7, FR-3.8, FR-3.9, FR-3.10
Description	The user runs the Decision Tree ID3 simulator with the desired dataset, receives the results in a visual format on the website and goes through the step-by-step simulation.
Precondition	None
Actions	1. The user opens the application. 2. The user clicks on the dropdown element labeled "Choose example dataset". 3. The user chooses a dataset from one of the example datasets that are provided by the application. 4. The application loads the dataset. 5. The application displays an information card designated for the dataset, the root node of the decision tree, a data table corresponding to the dataset, and the value table. 6. The user uses the four presented buttons to navigate through the step-by-step simulation.
Postcondition	The decision tree, data table, and value table are displayed at the user's desired step of the simulation.
Exceptions	None
Importance	High

Table B.3: UC-3 Run Decision Tree ID3 simulator.

UC-4	**Use own CSV dataset**
Version	1.0
Author	Daniel Drefs Fernandes
Associated requirements	FR-3, FR-3.2 FR-3.3, FR-3.4
Description	Within the Decision Tree ID3 simulator, the user selects their own CSV dataset that the application loads.
Precondition	The user has opened the application.
Actions	1. The user clicks on the button that is labeled "Load own CSV dataset". 2. The application displays a modal in which the user is presented with the file requirements and an input form. a) The user reads the file requirements. 3. The user clicks on the input form and selects the desired CSV file from their file system. 4. If given a valid CSV file, the application closes the modal and loads the dataset.
Postcondition	The user's desired dataset is loaded.
Exceptions	If the user has introduced an invalid CSV file, the application will display an alert that informs the user which requirement was not met and tells them to check the file requirements.
Importance	Low

Table B.4: UC-4 Use own CSV dataset.

Appendix C

Design specification

C.1 Introduction

C.2 Data design

Data storage

For the decision tree ID3 simulator, each example dataset is stored in a separate CSV file in a directory designated to the example data. When a certain dataset is selected by a user, a series of functions are triggered that load that set to be ready to use for the step-by-step simulation. The detailed process will be discussed in C.3.

When a user requests to load their own dataset for the simulation, that data is stored in the page's session storage. As no server was set up for this project, it had to be stored locally in the user's session storage. This brings advantages like faster access and the data being stored only for the duration of the page session. After storing the user's data locally in their page session, it is accessed by the application to load the dataset and process it.

CSV datasets

CSV files were chosen as a way to store all datasets in this application. They are in a format that is relatively simple to understand and create files from. In addition, CSV files can be parsed and processed quickly, which enhances the application's quick response to user input.

However, as the application is a relatively small one, some requirements had to be set in order to prevent users from uploading files that are too big or complex for the application to handle. Time constraints during development brought upon limits like, e.g., no numerical values being allowed and the datasets being limited to containing only 2 distinct class categories. Size limitations in the form of a maximum of 150 instance rows and 25 columns were added in order to keep the contents of the decision trees created by the application readable.

Figure C.1 shows an example of an acceptable CSV file.

```
1   Outlook,Temperature,Humidity,Windy,Play Golf
2   Sunny,Hot,High,False,No
3   Sunny,Hot,High,True,No
4   Overcast,Hot,High,False,Yes
5   Rainy,Mild,High,False,Yes
6   Rainy,Cool,Normal,False,Yes
7   Rainy,Cool,Normal,True,No
8   Overcast,Cool,Normal,True,Yes
9   Sunny,Mild,High,False,No
10  Sunny,Cool,Normal,False,Yes
11  Rainy,Mild,Normal,False,Yes
12  Sunny,Mild,Normal,True,Yes
13  Overcast,Mild,High,True,Yes
14  Overcast,Hot,Normal,False,Yes
15  Rainy,Mild,High,True,No
```

Figure C.1: Example of an acceptable CSV file

Decision tree design

The decision tree was designed with the thought that each node should not only be drawn with its respective attribute or, in the case of a leaf node, class label, but also contain additional information that helps the user understand each step of the ID3 algorithm. Figure C.2 shows an example of a decision node.

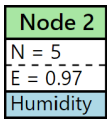


Figure C.2: Example of a decision node

Besides the node number, it includes the following information:

N: number of instances the node represents

E: entropy of the underlying dataset. It shows the impurity of the node and the need for it to be split

Attribute name: the name of the attribute that this node represents

Figure C.3 shows an example of a leaf node.

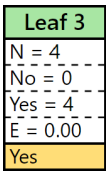


Figure C.3: Example of a leaf node

Besides the leaf number, N and E that represent the same properties as in a decision node, it contains the following additional information:

Class 1: number of instances possessing the first of the two possible class labels, in this case "No"

Class 2: number of instances possessing the second of the two possible class labels, in this case "Yes"

Label value: the assigned class label

The decision of displaying the number of instances belonging to each class was made with the purpose of putting emphasis on the relation between the purity of most leaf nodes and their underlying datasets being homogeneous.

The branch paths that go from one node to another have nothing worth mentioning besides each one containing the label of the attribute category on which the source node's dataset was split on in order to create a subset for the destination node. Figure C.4 shows an interaction between all three elements.

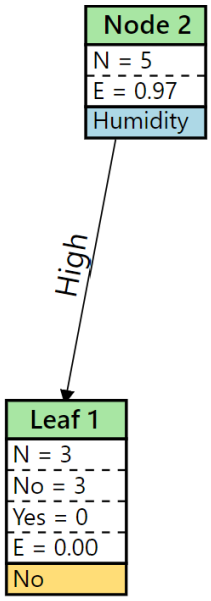


Figure C.4: Example branch

C.3 Procedural design

The most interesting case in regards to the procedure and sequence of execution would be the case of a user running the decision tree ID3 simulator and attempting to load a dataset. Figures C.5 and C.6 show sequence diagrams of how the application handles such a case.

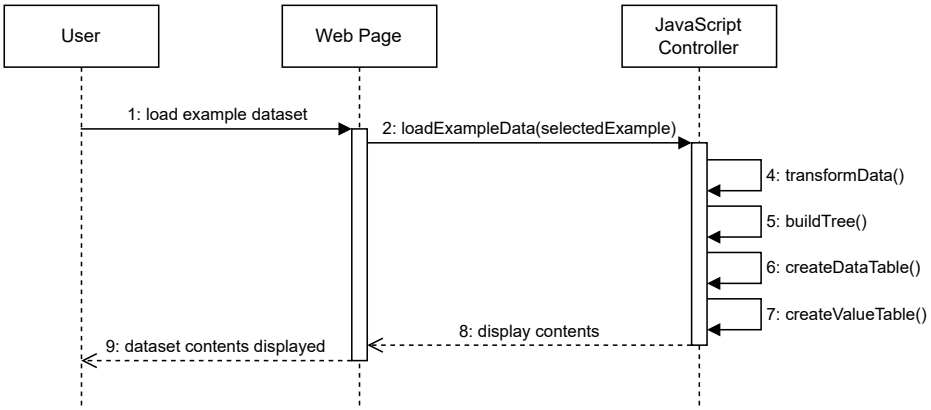


Figure C.5: Sequence diagram of user loading an example dataset

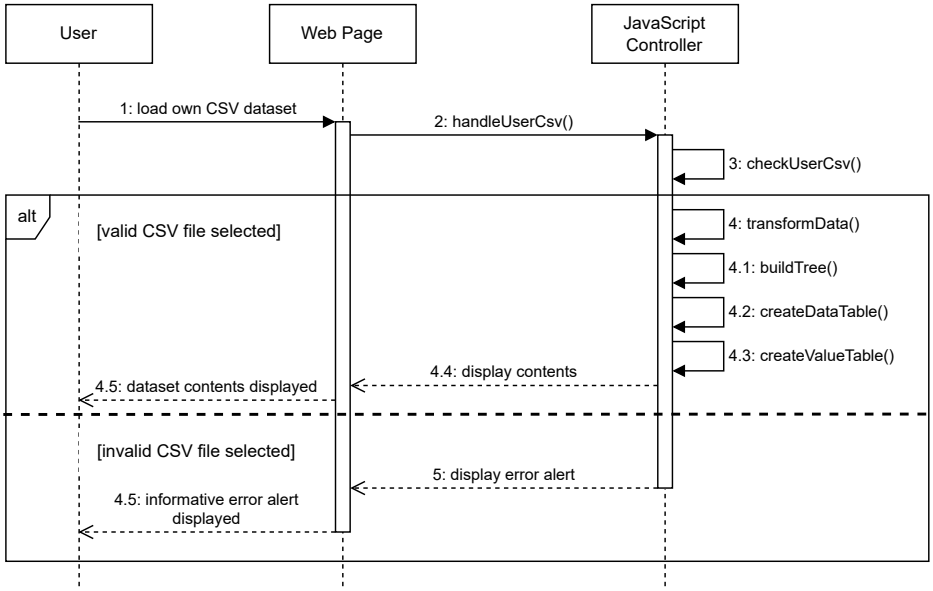


Figure C.6: Sequence diagram of user loading own CSV dataset

In both cases, if a valid CSV dataset has been selected, the program makes sure to first load all of the required resources before displaying them. This way, the user does not have to wait for, e.g., the data table to load when the SVG decision tree is already built and displayed.

C.4 Architectural design

No packages were created for this project, as the different parts of the web application were simply stored in different directories.

However, it was made sure to extract functions that were used in multiple JavaScript files into separate ones. This way, the specific functions could be imported by the files that needed them.

Appendix D

Technical programming documentation

D.1 Introduction

This section will include a more in-depth description of how the program works. The focus will be on the functionality of the SVG decision tree creation so that it can be modified or expanded upon more easily in the future.

D.2 Directory structure

In the following, all the project's directories will be listed along with a brief description of their content. The most important files will be listed, as well.

```
/
└── applications ...  The different parts of the application
    ├── conditional entropy ...  The conditional entropy calculator
    │   ├── css ...  CSS file with style properties unique to the
    │   │   conditional entropy calculator
    │   ├── js ...  JavaScript files
    │   ├── test ...  Test files
    │   └── index.html ...  Content of the web page
    └── decision tree ...  The decision tree ID3 simulator
        ├── css ...  CSS file with style properties unique to the
        │   decision tree ID3 simulator
        └── exampledata ...  CSV files that contain the example datasets
            and a JavaScript file containing information about them
```

```
        img ...  Images that are only displayed in the decision
        tree ID3 simulator
        js ...  JavaScript files
        test ...  Test files
        index.html ...  Content of the web page
    entropy ...  The entropy calculator
        css ...  CSS file with style properties unique to the
        entropy calculator
        js ...  JavaScript files
        lib ...  Libraries that are only used by the entropy
        calculator
        test ...  Test files
        index.html ...  Content of the web page
    img ...  Images that are shared by multiple parts of the
    application
    lib ...  Libraries that are shared by multiple parts of
    the application
css ...  CSS file with style properties unique to the front
page
doc ...  The Memoria and anexos
    Latex ...  The corresponding Latex files
    anexos.pdf ...  This file
    Memoria.pdf ...  The Memoria
img ...  Images that are only displayed on the front page
index.html ...  Content of the front page
```

D.3 Programmer manual

In the following, it is described how the ID3 algorithm and the SVG tree's creation was programmed.

For both algorithms, the same "TreeNode" class was used. Its fields include:

id: The node's id, e.g. "node1".

attribute: The node's attribute. In case of a leaf node, it is given the value null.

nodeValues: This describes an object of another class called "NodeValues". It is used to save a node's values that are displayed in the SVG tree:

class1: The number of instances belonging to the first of the two distinct target classes.

class2: The number of instances belonging to the second of the two distinct target classes.

n: The number of instances present in the node's underlying (sub-)dataset.

entropy: The entropy value of the node's underlying (sub-)dataset.

isLeaf: Boolean value that indicates whether a node is a leaf or not.

label: In case of a leaf node, the label value is stored.

prevBranchVal: The label of the branch path of which this node is the destination node. It is null for only the root node of a decision tree.

parent: The parent object of the node. It is null for only the root node of a decision tree.

children: An array of a node's children objects

depth: The depth of a node in the decision tree.

x: The absolute x position of the node within the SVG container.

y: The absolute y position of the node within the SVG container.

mod: The amount by which a node's children will be shifted to the right in the creation of the SVG tree.

ID3 algorithm

This algorithm is designed to create a decision tree structure made up of objects of the previously explained "TreeNode" class.

The implementation can be found in the file applications/decision tree/js/tree.js and is run using the function id3(data, attributes, prevBranchVal, nodeId, leafId), where:

data: the current node's dataset

attributes: the attributes considered for the current node

prevBranchVal: the label of the branch path for which the current node serves as destination node

nodeId: the current node id

leafId: the current leaf id

It must be noted that, as the algorithm creates the tree through a top-down approach, the node id and leaf id have to be increased using different variables the node id is only increased when a decision node has been created and the leaf id otherwise.

Listing D.3 shows the first step of the algorithm, which is the calculation of the "nodeValues":

Listing D.1: Calculation of "nodeValues"

```
let allPositive = true;
let allNegative = true;
let class1 = 0;
let class2 = 0;
let n = data.length;

// Save the labels in an array
let datasetLabels = [];
data.forEach(function (row) {
    datasetLabels.push(row.label);
});

// Caclulate entropy for the dataset
let e = entropyLabels(datasetLabels).toFixed(2);

for (const row of data) {
    if (row.label == labelValues[1]) {
        class2++;
        allPositive = false;
    }
    if (row.label == labelValues[0]) {
        class1++;
        allNegative = false;
    }
}
```

Variables "allPositive" and "allNegative" are then used to determine whether the algorithm has reached a leaf node, in which case, it creates a

TreeNode object with the calculated "nodeValues" and returns it. It also does so if the current set of attributes is empty.

It must be taken into account that the storing of values that are to be displayed in the data and value tables after the tree creation also happens in this function. Listing D.3 shows an example of that.

Listing D.2: Storing of relevant values for the value table

```
if (allPositive) {
    valTableGroup = [class1, class2];
    valueTableGroups.push(valTableGroup);
    ...
}
```

Here, if a leaf node has been reached, the number of instances belonging to both classes are stored in an array "valTableGroup" and pushed into another array "valueTableGroups" that saves relevant values for the value table at each step.

While it would have also been possible to do another traversal to give higher priority to a separation of concern, it was decided not to do that as this would not have only meant another traversal of the tree, but another calculation of all the values that had been calculated, but not stored, during the first traversal.

The next steps are to calculate the best attribute based on their information gain value, split the current dataset into subsets using the attribute's categories, and doing a recursion call. Alternatively, if any of the created subsets, a leaf node is created with the most common label of the dataset before the split. Listing D.3 shows that part of the code.

Listing D.3: Recursion call for each subset

```
// Do a recursive call for each value of the
selected attribute or add a leaf node if the
value's subset is empty
for (const value of bestAttributeValues) {
    let subset = data.filter(instance =>
    instance.attributes[bestAttribute]
    === value);

    ...

    if (subset.length === 0) {
```

```
            valTableGroup = [class1, class2];
            valueTableGroups.push(valTableGroup);

            tree.children.push(new TreeNode(leafId,
            null, new NodeValues(class1, class2, n,
            e), true, mostCommonLabel(data),
            prevBranchVal));
        } else {
            tree.prevBranchVal = prevBranchVal;
            let returnVals = id3(subset,
            remainingAttributes, value, nodeId,
            leafId);
            tree.children.push(returnVals[0]);
            nodeId = returnVals[1];
            leafId = returnVals[2];
        }
    }
    return [tree, nodeId, leafId];
```

It is worth mentioning that the node id and the leaf id have to be returned as well for them to be increased in an appropriate way.

In the end, a tree structure will be created that represents the decision tree that will be drawn using SVG afterwards.

SVG tree creation

Before explaining the calculation of each node's position, it must be mentioned that the tree is created using HTML SVG elements. A SVG container is placed on the web page, its size changes dynamically when the browser window is resized. All following calculations are done relative to the current SVG container size. SVG symbols were created that work as templates for the individual nodes that are dynamically created. The symbols are cloned and changed to display each node's individual properties. For this reason, upon creation of these symbols, a standard height and width was set for decision nodes and leaf nodes. Figure D.1 shows the symbol that was created for decision nodes.

```
<symbol id="node" viewBox="0 0 82 92">
    <rect x="1" y="1" width="80" height="26"
        style="fill: #A8E6A3; stroke-width:2; stroke: black;" />
    <text id="nodeNr" x="12" y="20" font-size="18" fill="black" font-weight="500">Node
    </text>
    <rect x="1" y="27" width="80" height="42"
        style="fill: white; stroke-width:2; stroke: black;" />
    <text id="nodeN" x="5" y="43" font-size="16" fill="black">N = </text>
    <path d="M 1 48 81 48" stroke="black" stroke-width="1.3" stroke-dasharray="5.3" />
    <text id="nodeE" x="5" y="64" font-size="16" fill="black">E = </text>
    <rect x="1" y="69" width="80" height="21"
        style="fill: #ADD8E6; stroke-width:2; stroke: black;" />
    <text id="nodeAttribute" x="5" y="84" font-size="16" fill="black"></text>
</symbol>
```

Figure D.1: Decision node symbol

The implementation can be found in the file applications/decision tree/js/tree.js and is run using the function buildSvgTree().

Setup

The first step in calculating the node's positions is to determine how many rows and columns the SVG tree will possess. One row contains either a certain amount of columns of nodes or branch paths. Columns contain nodes.

The amount of nodes at each level/depth is counted. To calculate the amount of rows, the number of levels is multiplied by 2 and 1 is subtracted. A leaf node's height is used as a measure. Figure D.2 shows an example decision tree which contains 5 rows.

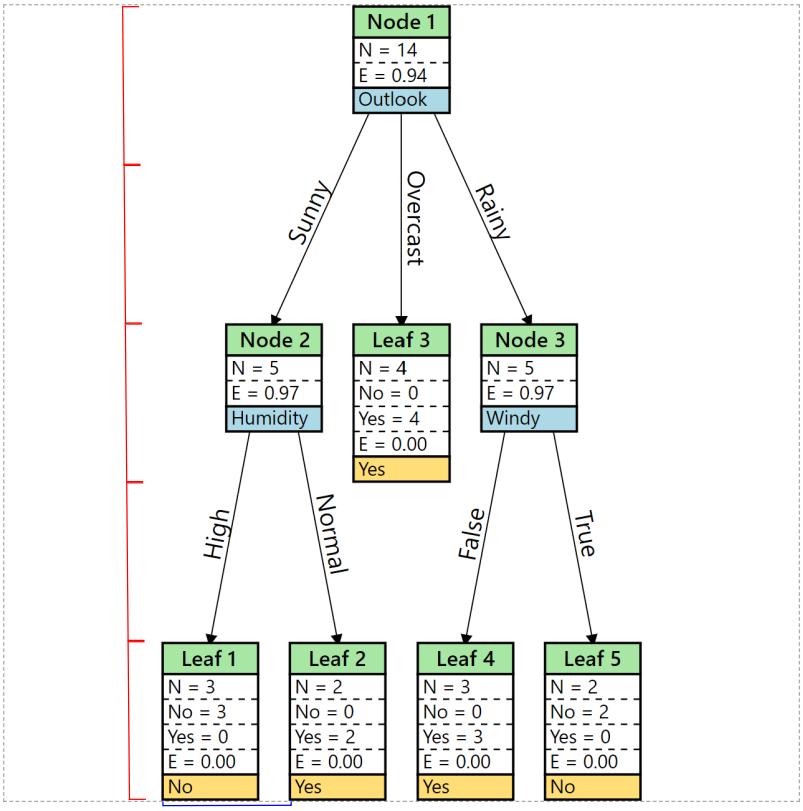


Figure D.2: SVG decision tree containing 5 rows

As the red markers indicate, exactly 5 leaf nodes stacked on top of each other would fit into the SVG of which the borders are marked by a dotted line. The amount of columns is determined by calculating the maximum amount of nodes on one level. In figure D.2, the amount of columns would be 4 as the last level contains 4 nodes.

Based on these values, the actual height of a row is determined by dividing the height of the SVG container by the number of rows. The same is done for the width of a column, using the width of the SVG container.

However, only one ratio can be used to down-scale the width and height of the nodes according to the sizes of the tree the SVG container. The smaller, more restrictive one between the two ratios is determined and chosen. If the bigger one would be chosen, the nodes could go out of bounds of the SVG container.

With the determined ratio, the final widths and heights of decision and leaf nodes is calculated. It is made sure that a separate column width is calculated to ensure that there will be a minimum space between nodes on the same level. That column width can be seen as the blue marker between "Leaf 1" and "Leaf 2" in figure D.2.

Calculating nodes' positions

This algorithm partially implements the Reingold-Tilford algorithm [3], which was originally proposed as a method for drawing binary trees in a compact and aesthetically pleasing manner. This implementation expands on the idea by making it work with non-binary trees.

As a first step, the algorithm does a post-order traversal of the tree and calculates the nodes' initial x-positions. It does so by positioning the leaf nodes next to each other, each at a distance of the width of a column. Nodes that have one child are positioned above their child and nodes with more than one child are placed in the middle between their children. Additionally, it is checked if a parent node has any siblings left of it. If yes, it is placed one column width apart from their left sibling. Its mod value is also set which, at a later step, will shift its children to the right.

For better understanding, example images [2] using letters as node descriptors will be used. Figure D.3 shows each node's x-positions after the described first step.

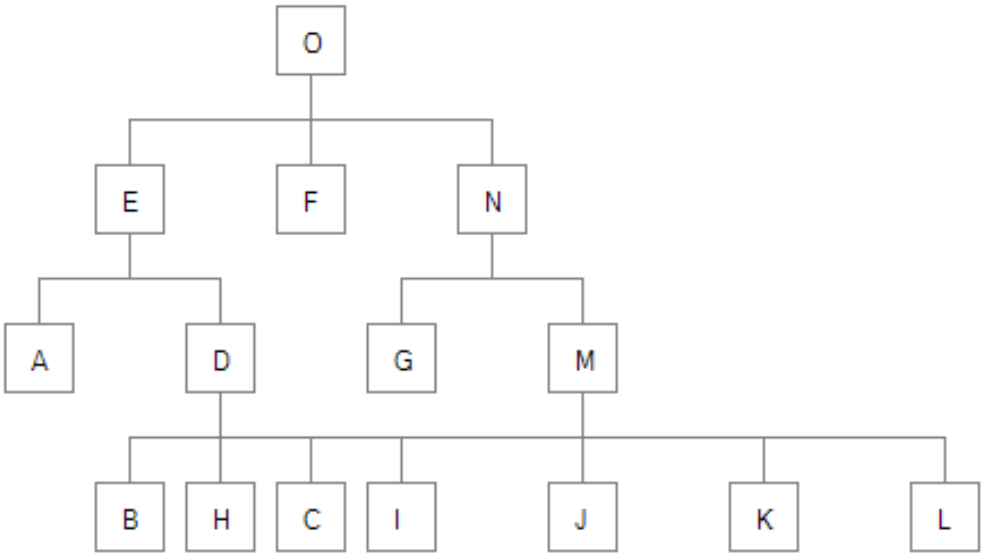


Figure D.3: Initial nodes' x-positions

However, the tree has overlapping nodes in this state, so they need to be corrected. Still traversing the tree in a post-order, each node's left edges are calculated and checked if they overlap with any of its siblings' right edges. Figure D.4 shows node N's left edge in red and its sibling's E's right edge in blue.

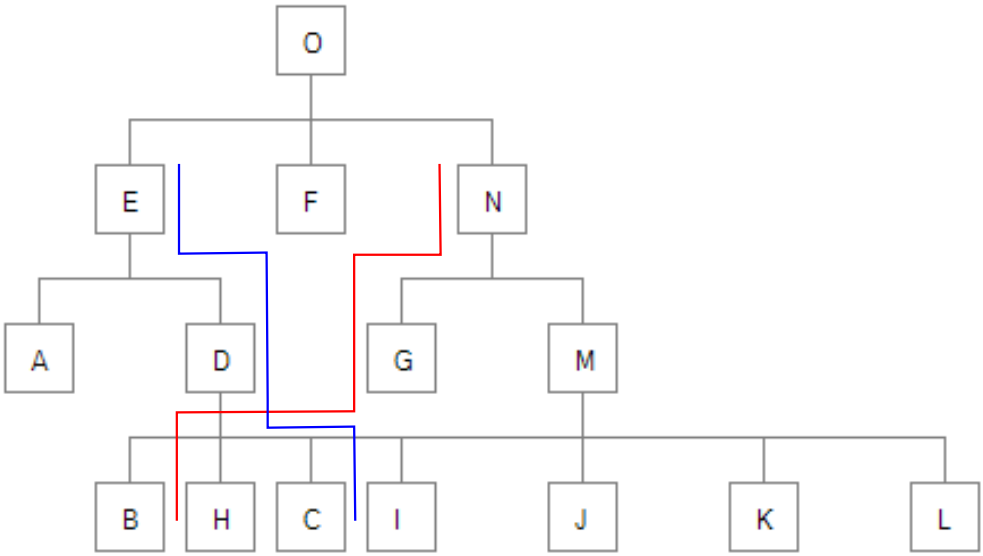


Figure D.4: Initial nodes' x-positions

It can be seen that node H of node N's subtree is overlapping with node C of node E's subtree. To avoid conflicts like this, each of the current node's sibling's right edges are checked for whether all nodes are at least one column width apart. If that is not the case, the node (in this case N) is shifted by the necessary amount and its mod property is increased by the same amount so that its children will be shifted later, too.

To make the tree more aesthetically appealing, the algorithm makes sure to also center nodes in between of subtrees that were overlapping and had to be shifted. In this case, F would have to be shifted. Figure D.5 shows the results of this step.

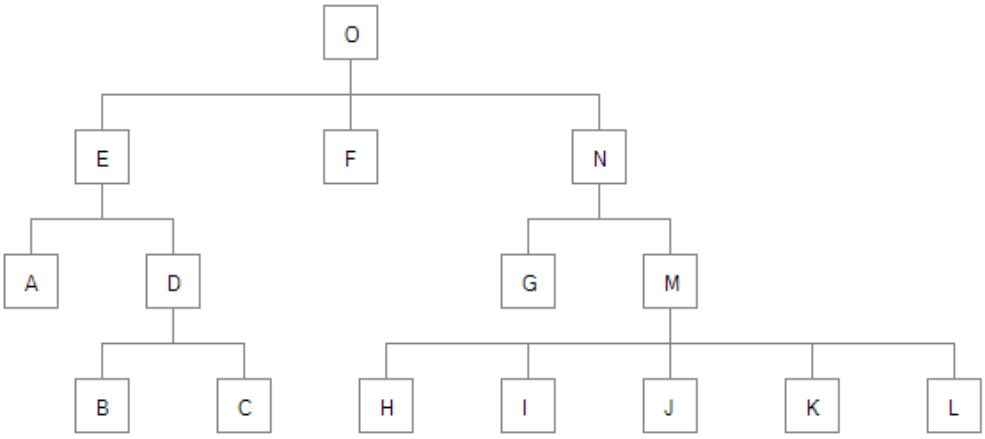


Figure D.5: Nodes' x-positions after resolved conflicts

With that, the first traversal of the tree has been completed. However, a node's mod value can be of negative value, which could potentially position its child nodes out of bounds of the SVG container. That is why, in addition, the left edges of the root node are calculated again and shifted by the necessary amount. That same amount is added to the mod value to ensure all children will be in bounds.

Inversely, the root node's right edges are calculated, as well, to check if there is still free space between the tree's right edge and the right border of the SVG container. If there is, the tree is shifted to the right.

As a final step, the algorithm does a top-down traversal of the tree and calculates each node's final x-position by adding the mod value of all its parent nodes to its x-position. Their y-values are also calculated in this step, which are based on their depth and the height of a row, which is equal to a leaf node's height.

Symbol cloning

With all nodes' positions inside the SVG container calculated, the aforementioned symbols are cloned to store each node's and branch path's individual properties. Ultimately, use elements are created that display these cloned symbols and indicate each node's position. Figure D.6 shows an example of a use element displaying a leaf node.

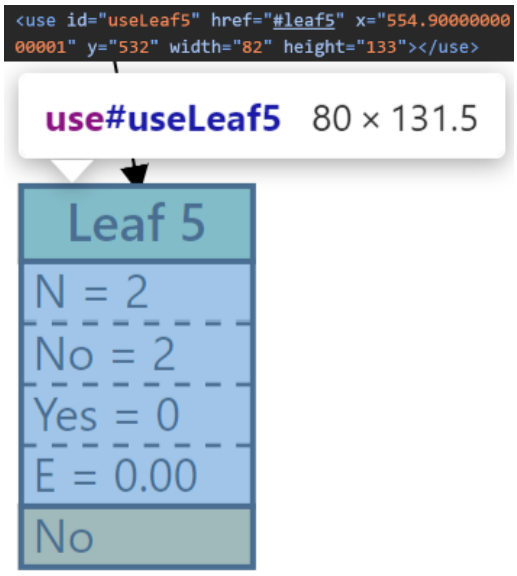


Figure D.6: Example of a leaf node use element

These use elements and all cloned symbols are made sure to be deleted before a different dataset is loaded.

Ultimately, an aesthetically pleasing decision tree is created using SVG. An example is shown in figure

D.4 Project compilation, installation and execution

There are no executables or scripts that need to be run to execute this project as it is based on HTML pages with JavaScript files as the underlying scripts that make the web pages interactive. However, to make use of the locally included libraries used in this project, it needs to be cloned from the GitHub repository.

This project was developed on a machine operating on the Windows 11 OS. Git would have to be installed on the machine, which can be done through a package download from the official git website[1].

The project would have to be cloned to the user's file system. From that point on, no further compilation installation would have to be done. The

[1]git download for Windows: https://www.git-scm.com/download/win

project would be ready to be run in a browser and modified in any code editor.

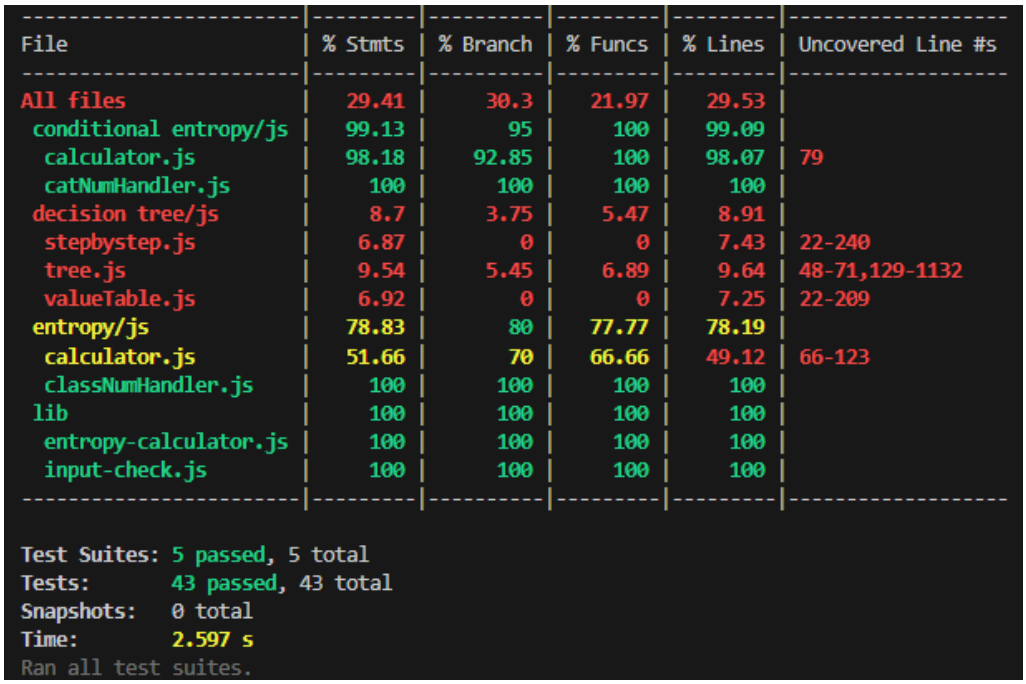
However, to run the tests included in the project, Node.js and npm would need to be installed, which can be done in multiple way, as indicated on the official website[2]. To install the relevant dependencies for the testing framework, the command "npm install" would have to be run from the terminal. The current directory would have to be the project directory in the user's file system. When everything is installed, the tests could be run with the command "npm test".

D.5 System testing

Jest

Jest [1] is a JavaScript testing framework that allows the user to write simple JavaScript tests without a complicated setup. This framework has been used as a main device for testing the project's functionality. Code coverage reports are also created every time a test is run. However, due to time constraints towards the end of the project, not all files and functions could be formally tested. Figure D.7 shows the code coverage of the created tests for this project with Jest.

[2]Node.js and npm installation: https://docs.npmjs.com/downloading-and-installing-node-js-and-npm



File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	29.41	30.3	21.97	29.53	
conditional entropy/js	99.13	95	100	99.09	
calculator.js	98.18	92.85	100	98.07	79
catNumHandler.js	100	100	100	100	
decision tree/js	8.7	3.75	5.47	8.91	
stepbystep.js	6.87	0	0	7.43	22-240
tree.js	9.54	5.45	6.89	9.64	48-71,129-1132
valueTable.js	6.92	0	0	7.25	22-209
entropy/js	78.83	80	77.77	78.19	
calculator.js	51.66	70	66.66	49.12	66-123
classNumHandler.js	100	100	100	100	
lib	100	100	100	100	
entropy-calculator.js	100	100	100	100	
input-check.js	100	100	100	100	

```
Test Suites: 5 passed, 5 total
Tests:       43 passed, 43 total
Snapshots:   0 total
Time:        2.597 s
Ran all test suites.
```

Figure D.7: Jest code coverage

Manual testing

In addition to the usage of the Jest testing framework, the biweekly sprint review meetings were also used as a method of manually testing the web application. In this way, the tutors often stumbled upon errors the presenter of this work had not considered. The tutors also reviewed the web application on their own before a meeting, which brought the advantage of having been able to discuss shortcomings or mistakes during the meetings.

Appendix E

User documentation

E.1 Introduction

This appendix shows what requirements the user has to meet to run the application and how to install and use it.

E.2 User requirements

As this project consists of a web application, the user only needs to be able to run a browser that supports JavaScript, CSS sheets, Bootstrap, jQuery, and D3. Some examples would be:

- Google Chrome
- Mozilla Firefox
- Microsoft Edge
- Apple Safari
- Opera

E.3 Installation

No installation is necessary for this project as it consists of a web application.

E.4 User manual

First, the user needs to open the application through the following URL: https://danieldf01.github.io.

Front page

They are presented with the front page that is shown in figure E.1.

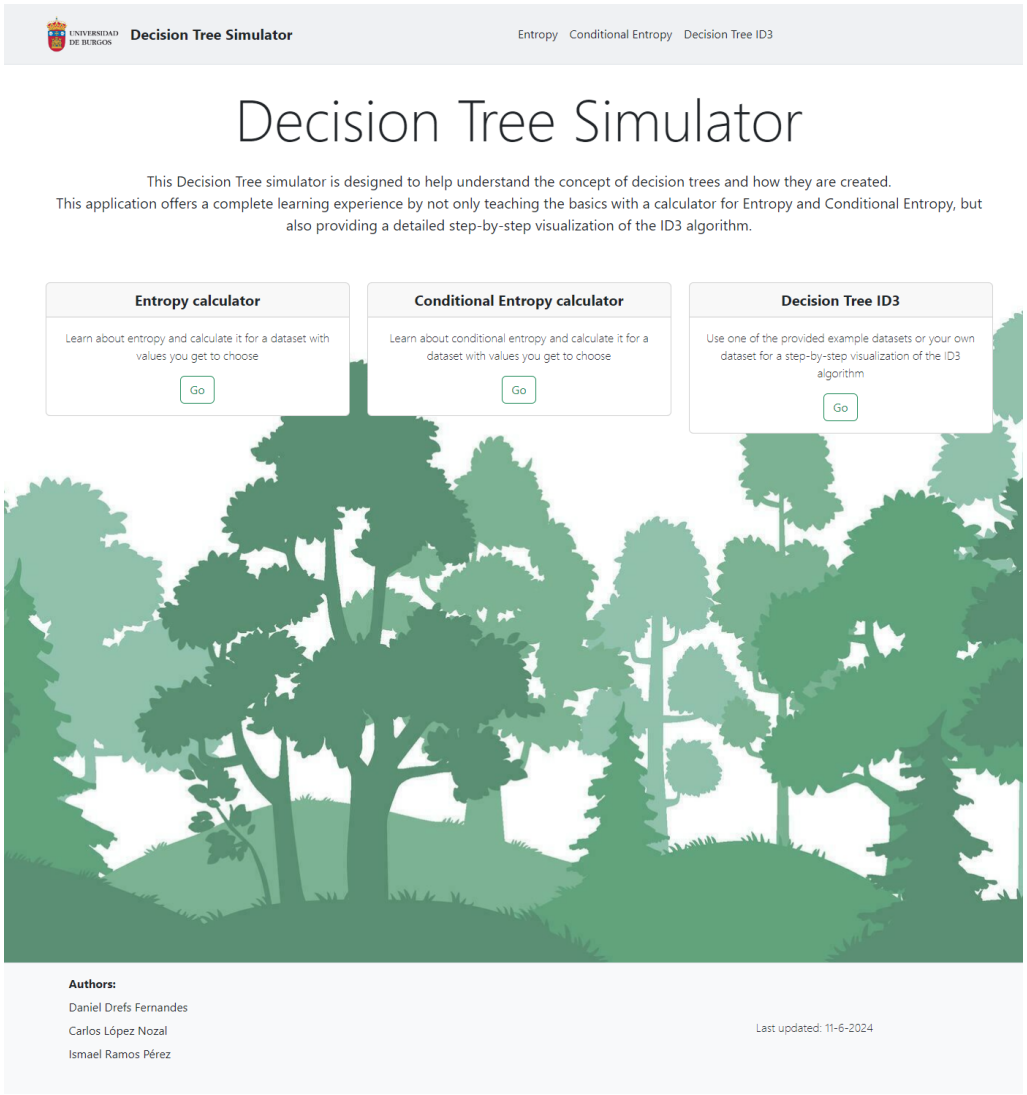


Figure E.1: Front page of the web application

Besides the title and a short introduction, three information cards are displayed, each one corresponding to a functionality of the web application with the opportunity to open either one: “Entropy calculator”, “Conditional Entropy calculator”, and “Decision Tree ID3”.

Entropy calculator

If they choose to open the Entropy calculator, they are presented with the page shown in figure E.2.

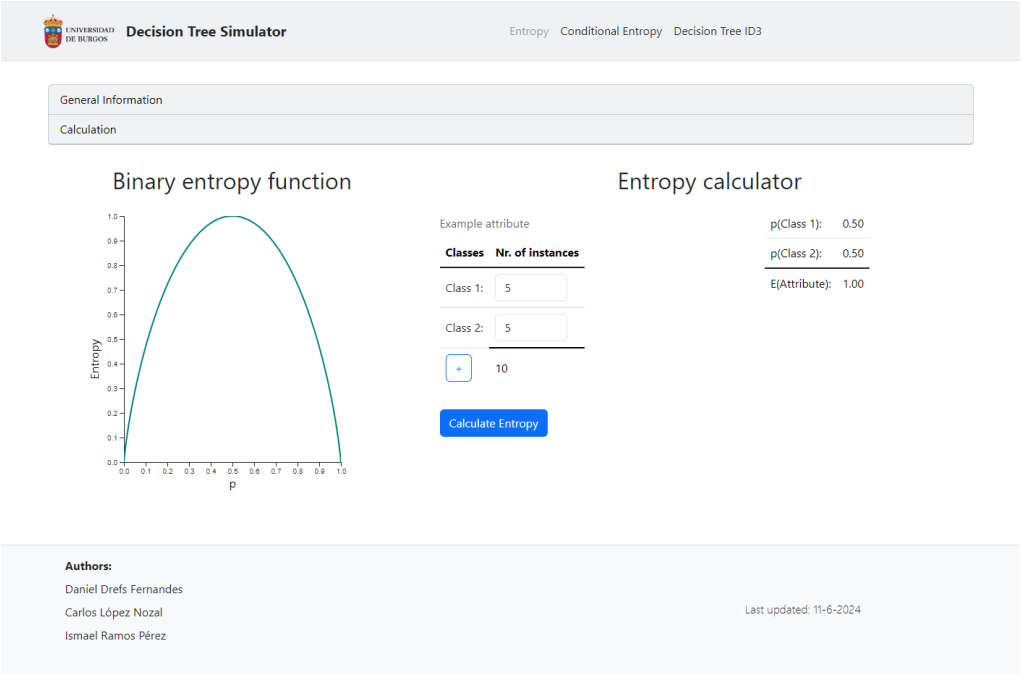


Figure E.2: Entropy calculator page

The page consists of the following components:

- Information cards at the top.

  General Information: provides a brief introduction to the concept of entropy.

  Calculation: displays the entropy formula and explains each part of it.

- A graph displaying the binary entropy function on the left.
- The entropy calculator on the right, consisting of two tables.

  Left table: represents an example attribute whose instances are distributed among 2 classes.

  Right table: shows the proportion of each class in the dataset and the entropy of the attribute.

The user is now free to introduce any positive integer values into the input forms presented in the left table and calculate the entropy for those values. The user is informed by an alert if invalid values were put in.

If the entropy was calculated for two classes, a red marker in form of a line and a point will appear on the graph on the left, indicating the proportion of class 1 and the corresponding entropy value of the attribute. This is shown in figure E.3.

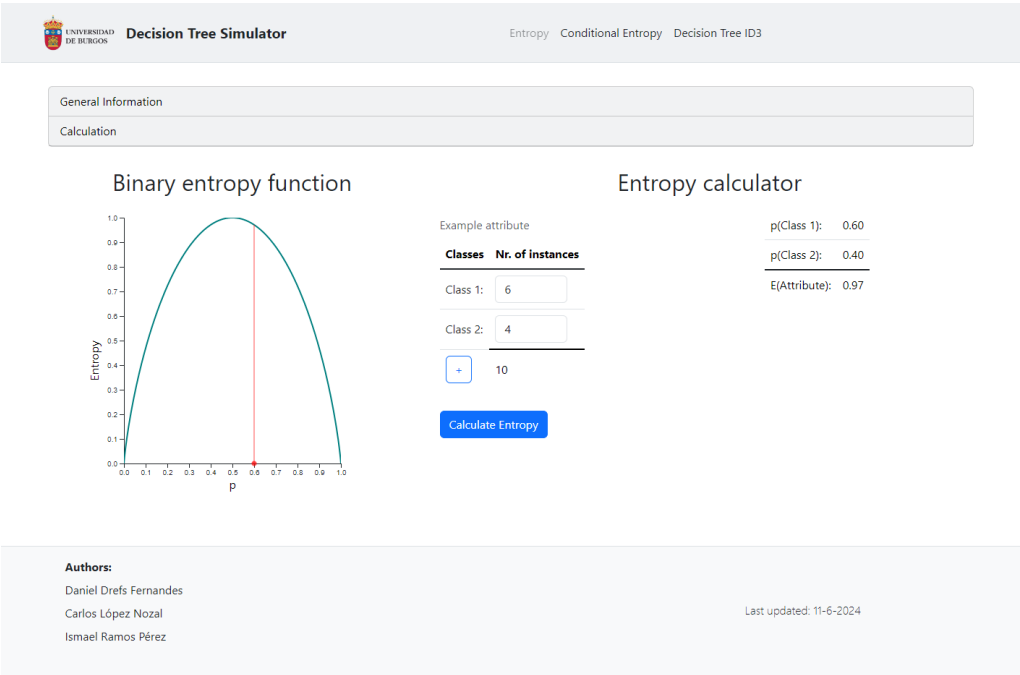


Figure E.3: Entropy was calculated for 2 classes

The user can also add classes with a click of the "+" button and calculate the entropy for an attribute whose instances can belong to more than two

classes. However, the red marker that indicates the results on the graph will not be displayed with more than two classes. The user is also informed about this with the use of an information alert. Figure E.4 shows that.

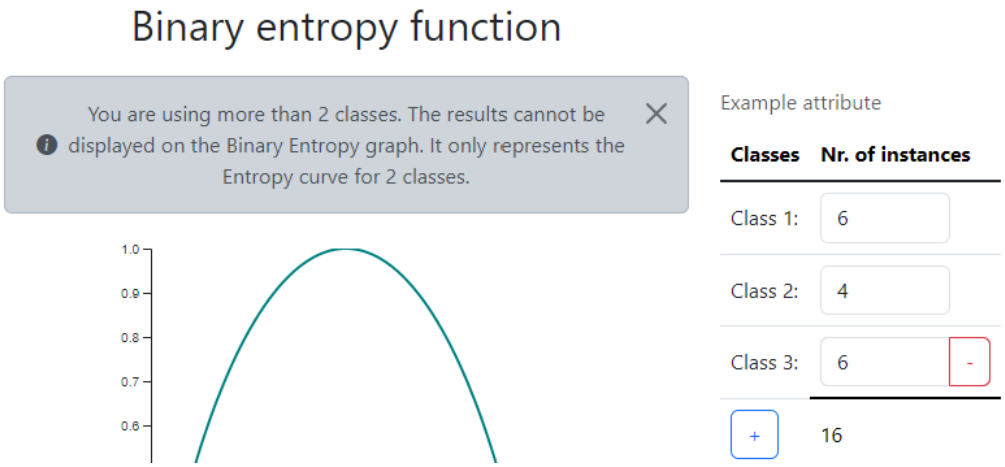


Figure E.4: Entropy is calculated with more than two classes

The user can also remove the class that was last added by clicking the "-" button.

Conditional Entropy calculator

Figure E.5 shows what users are shown upon entering the page for the conditional entropy calculator.

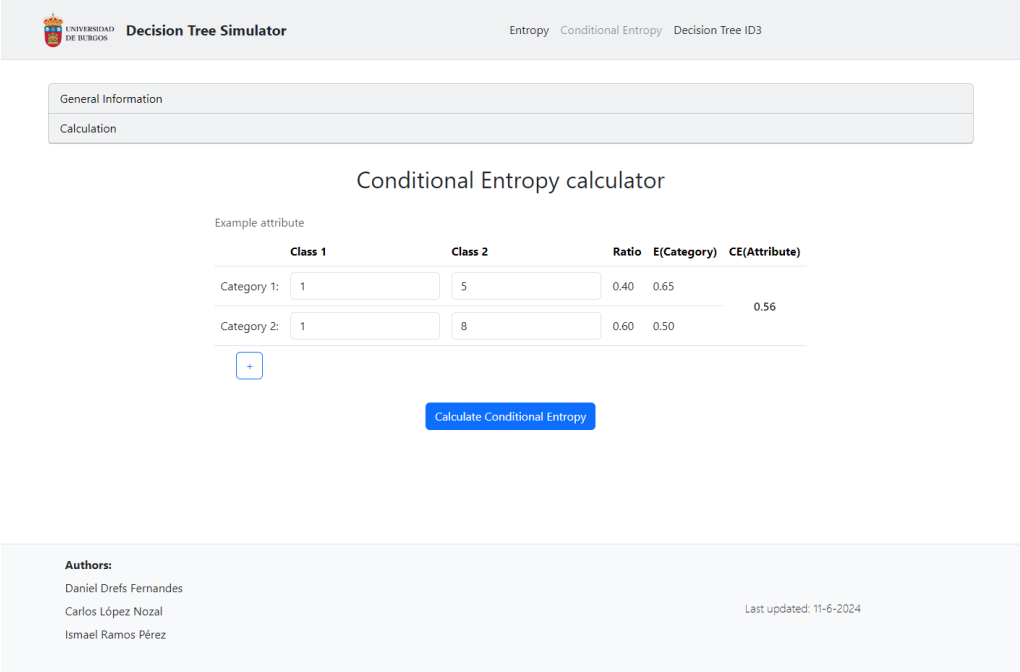


Figure E.5: Conditional Entropy calculator page

The page contains the following content:

- Information cards at the top.

  General Information: provides a brief introduction to the concept of conditional entropy.

  Calculation: displays the formulas used to calculate the conditional entropy and explains each part of them.

- The conditional entropy calculator in the middle.

Much like on the entropy calculator page, the user can introduce any positive integer values into the input forms presented in the table and calculate the conditional entropy for those values. The user is informed by an alert if invalid values were put in.

The user is able to add categories with a click of the "+" button and calculate the conditional entropy for an attribute with more than two categories. They can also remove the category that was last added by clicking the "-" button. This is indicated in figure E.6.

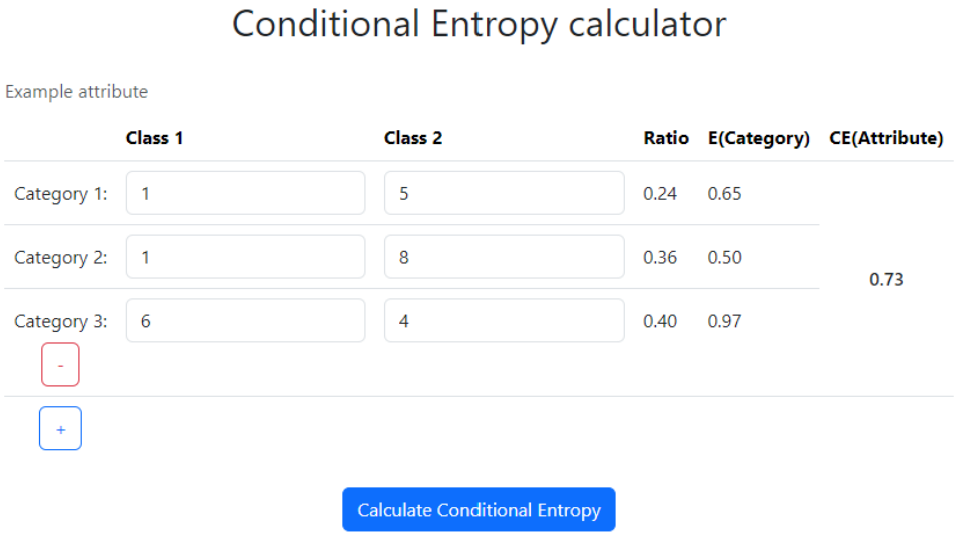


Figure E.6: Conditional entropy calculated for 3 categories

Decision Tree ID3

After first entering the page for the decision tree ID3 simulator, the user is presented with what is depicted in figure E.7.

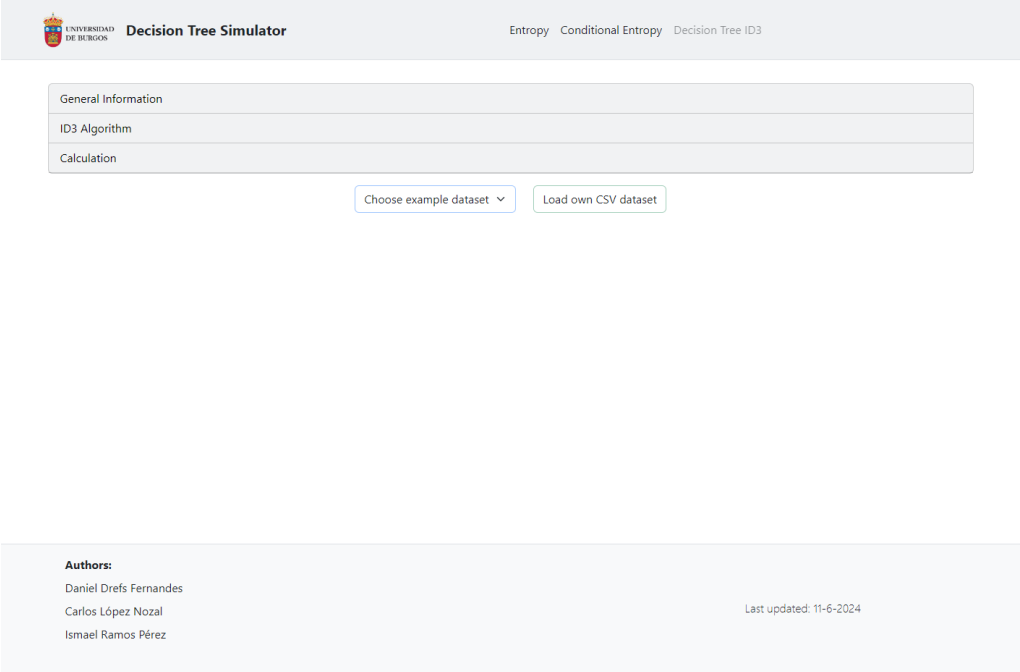


Figure E.7: Decision tree ID3 simulator page

In this state of the page, the contents are:

- Information cards at the top.

  General Information: provides a brief introduction to the concept of decision trees and how they are constructed with the ID3 algorithm.

  ID3 Algorithm: explains the ID3 algorithm in detail

  Calculation: displays the information gain formula and explains each part of it.

- A dropdown menu and a button below the information cards.

The user can now choose between using an example dataset, provided by the application, for the simulation and loading their own dataset in CSV file format.

Example dataset

The user chose one of the example datasets and is now presented with additional content that is shown in figure E.8.

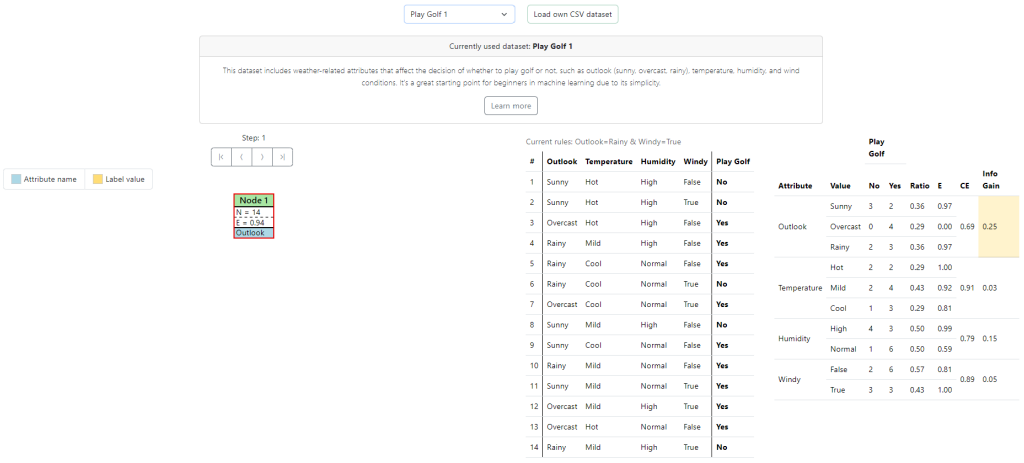


Figure E.8: Loaded dataset

The now visible content includes:

- Information card about the dataset in the middle below the dropdown menu and button.
- On the left, the first node of the decision tree that is built upon the chosen dataset.
- A legend above and left of the node.
- Four buttons representing the functions “Initial step”, “Step back”, “Step forward”, and “Last step”.
- A step counter above the four buttons.
- Two tables on the right:

  Left table: A table representing the loaded dataset.

  Right table: A table representing relevant values at each step.

The user can now choose to move throught the step-by-step simulation by clicking any of the four buttons. As the decision tree is built to completion with the step-by-step simulation, the tables on the right are also changed. E.g., arriving at the last step would look like figure displays.

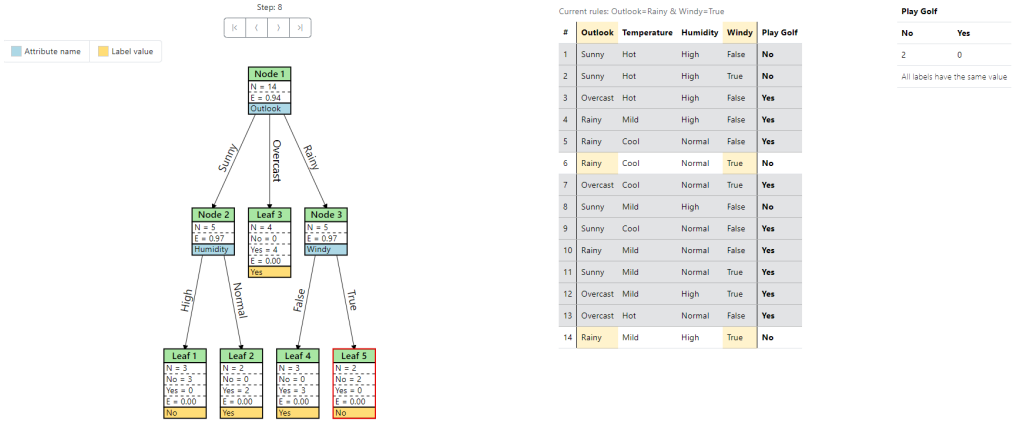


Figure E.9: Final step of the step-by-step simulation

Even with a dataset already loaded, the user can choose to switch to a different dataset.

User CSV dataset

If the user chooses to click on the button labeled “Load own CSV dataset”, a modal appears as is shown in figure .

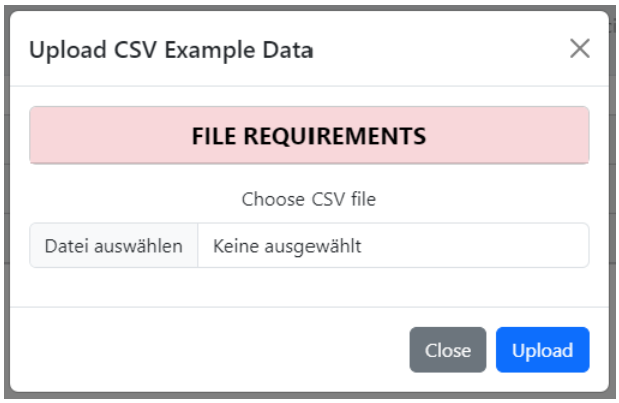


Figure E.10: Modal that appears when user wants to load their own CSV dataset

They are now presented with:

- A collapsible card containing file requirements at the top.
- Below that, an input form for the user to choose their own file.
- Two buttons at the bottom right of the modal, labeled “Close” and “Upload”.

The user can now choose to read the file requirements before trying to upload their own dataset in CSV file format via the suggested input form. If they click on the input form, they get to choose a file from their file system. After clicking on the button labeled “Upload”, the selected file will be checked and, if it is invalid (does not meet the file requirements), the user is confronted with an alert that describes why the file is considered invalid. An example of this is displayed in figure E.11.

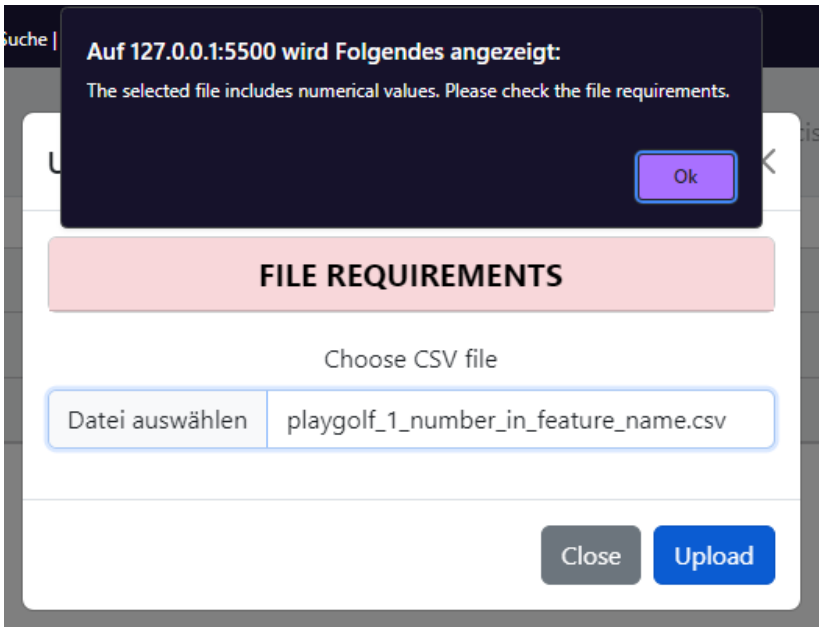


Figure E.11: Alert that appears when a user tried to upload an invalid file

Following that, they can try again with a different file and, if the file is valid, and the upload was successful, they are presented with the same general content as in E.4. However, a description of the dataset will not be provided. And, unless it is the same dataset, the constructed decision tree will look different and the tables will have different contents.

Appendix F

Curricular sustainability annex

F.1 Introduction

This appendix will offer a personal reflection on the sustainability of this project. In the context of a web application, sustainability aims to create solutions that are efficient, user-friendly and environmentally responsible.

F.2 Reflection on sustainability

Client-side processing

The decision of only using client-side processing for an application as small as this one ensures a higher ressource efficiency, as none of the used data needed to be stored on a server. A user's CSV dataset is always stored in their browser page's session, which is erased when the tab or the window is closed.

KISS

The aforementioned structure choice also goes with the KISS (Keep It Simple, Stupid) principle which aims for simplicity and avoiding unnecessary complexity. This results in lower overall energy consumption and lower maintenance costs. It produces fewer bugs and makes the project easier to update or upgrade. Simpler code also proves to show higher adaptability to future changes that might be implemented by possible developers aiming to enhance the application's functionality.

Open-source

only open-source tools and libraries were used in this project, which contributes to their further development and wide-spread usage. Having the code to this project publicly available and giving anyone access to it is another sustainable contribution to the developer community. Not only does it give other programmers the chance to extend this code, it could also prove helpful in teaching students or people who are interested in the topic.

Reducing computational complexity

Throughout this project, it was tried to always keep the computational complexity as low as possible by using efficient recursive algorithms like, e.g., the Reingold-Tilford algorithm for calculating nodes' positions in a tree.

Maintainability

It was made sure to keep this project's code easily readable for future developers who might build upon this project or modify it in some way. This was done by clear comments that explain what is done at each step. Easy-to-understand variable and function names were a priority, too. Additionally, it was tried to keep separation of concern at a high level so that programmers trying to expand upon or edit this project can easily dismantle each component without having to worry about unclear entanglements within the code.

Production cost

As seen in A.3, the only production cost for this project came from the system it was developed on. No additional licenses or tools had to be bought that might have influenced sustainability in an unfavourable way.

Lightweight design

The decision to stick to a more minimalistic design for the web application with a little amount of presented images adds to the project's resource efficiency. The use of SVG contributes to that, as well.

Site-hosting with GitHub Pages

Using GitHub Pages as a site-hosting service to display this web application adds to this project's sustainability as well, as GitHub is owned by Microsoft,

which has made commitments to carbon emissions and using renewable energy.

Responsive Design

Building a web application that works well across different devices allows for a shared experience regardless of the type of device the project is being run on. It reduces the need for multiple versions of the same application.

Potential for improvement

However, there is a part of this project that weighs negatively on its sustainability. That being the unfinished tests. Unfinished tests equal a possibility of an undiscovered error or unexpected behavior that might pose an obstacle to a future developer that wants to expand upon this project. Needless to say, even with a full test coverage, there would be the possibility of unforeseen errors occurring. But it would be smaller than with only parts of the project being covered by tests.

Bibliography

[1] Jest. Jest - delightful javascript testing, 2024. [Internet; visited 11-june-2024].

[2] Rachel Lim. Algorithm for drawing trees, 2014. [Internet; visited 11-june-2024].

[3] Edward M. Reingold and John S. Tilford. Tidier drawings of trees. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING,*, SE-7(2):223–228, 1981.

[4] PwC Spain. Spain individual - other taxes, 2024. [Internet; visited 09-june-2024].

[5] WageIndicator. Salario mínimo – españa, 2024. [Internet; visited 09-june-2024].