GODOT BENDED-ATTRIBUTE TREES VISUALIZATION TOOL

Central Washington University
CS556 Data Mining Final Project
Student: Andrew Dunn
Instructor: Dr. Boris Kovalerchuk

TABLE OF CONTENTS

1 Introduction

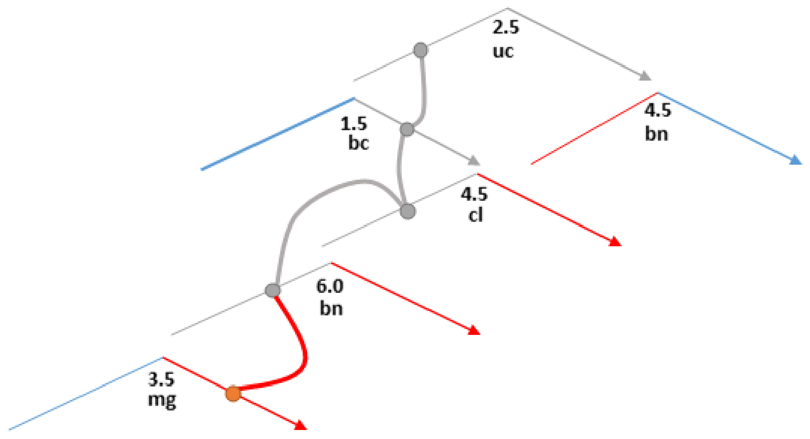


Figure 1: Visual Example of Shifted Bended Attributes

Bended Attributes is a method of visualizing decision tree models and their decision-making process. Figure 1 above shows a general representation of Shifted Bended Attributes, which is a version of Bended Attributes that visually encodes the distances of a sample's attributes to the tree node's threshold values. In figure 1, the bended line that starts as a gray color, and then ends in a red color at one of the tree leaves represents a single sample from the dataset after it has been overlayed on the tree. The circle dots that lie upon the tree branches represent the sample's attribute values. The further the dot is from the parent tree node; the further the sample's attribute value is from the threshold.

2 Tools and Source Code

We have come up with a generalized implementation of Bended Attribute Trees, in the form of a computer program that can display both binary classification decision trees and their associated data samples. This program was designed and implemented using a tool named Godot 3.2.3. Godot is a free, open source game engine written in C++ that supports OpenGL rendering on all major operating systems, including Windows, MacOS, and Linux. Godot utilizes a built-in script engine called GDScript, which is a programming language very similar in syntax to Python. GDScript allows for rapid development and prototyping, and greatly reduces the amount of time spent on implementing utility functions and boiler-plate code that would be required in other programming languages. The biggest strength of Godot is that it has a built-in OpenGL rendering engine that greatly simplifies the process of drawing things such as lines, shapes, and textures to the screen. This makes adding new visualization features to the application much easier than it would be in other programming languages.

GDScript does have some limitations, and these are primarily a lack of external data science libraries, slower performance than more traditional programming languages, and only single-precision floating point number support. Despite these limitations, we deemed GDScript to be adequate for implementing Bended Attribute Trees. If these limitations are deal-breaking however, then Godot also allows code to be written in C# and C++ that interfaces directly with the engine. Thus, a Godot project can be designed to utilize any of the three supported programming languages, and they can interact with each other. For the reasons specified earlier, our current implementation is fully written in GDScript, but this could be changed in the future if needed.

To open the included source code and Godot project files, you must download the Godot Editor from their homepage at https://godotengine.org/. The usage of the Godot editor is outside the scope of this paper, so anyone who is interested on modifying this project should take some basic tutorials online. We recommend the tutorial here, or a shorter one found here. Since Godot is a game engine, most tutorials online focus on the game making aspects of the program. Despite this, these tutorials do still teach you the basics of Godot's scene tree structure, the GDScript programming language, and how to draw objects on the screen, which is what our Bended Attributes application does.

3 Features and Limitations

Our Bended Attribute Visualization Tool inherently supports Windows, MacOS, and Linux. Since Godot supports OpenGL 2.1, it should run on most hardware. The current implementation of our program only supports binary classification decision trees. The decision tree structure must be in a specific JSON file format, the specifications of which you can find below in Section 6 JSON Decision Tree Format Specifications. Currently, our program only supports real value thresholds for the tree nodes, and does not support Boolean, or categorical attribute tree nodes. There is no hard-coded limitation for the size, number of tree nodes, or complexity of the decision tree if it is in the correct JSON format.

Our program also supports the loading and displaying of samples from a dataset. Currently, the dataset must be in a comma separated values (.CSV) or tab separated values (.TXT) format, with the first row reserved for the column labels. For the dataset to be loaded successfully, the column labels must match the tree node's attribute names exactly. If our program cannot find one of the attributes used in the decision tree in the dataset, an error message will be displayed. The dataset can have additional attributes other than the ones used in the decision tree, and our program will load them alongside the other attributes used in the tree. This is useful if you want to filter samples based on their ground-truth versus predicted classes, for example. There is no hard-coded limitation on the size of the dataset, nor number of samples (rows) that can be loaded. Note however that loading thousands of samples may severely degrade the application's performance, therefore it is highly recommended that the number of samples loaded at any one-time number in the hundreds. We have included a simple Python script that can randomly split a CSV dataset for you into training and testing sets, which is useful if you want to load one or the other into our program. You can read more about this tool in Section 5 Additional Included Tools below.

4 Application UI and Usage

4.1 Main Application Screen

To use the Godot Bended-Attribute Trees Visualization Tool, execute the appropriate included binary for your operating system. A window should open that displays the text: "Loading …". Once loading has finished, you should be greeted with the programs main screen, which looks like figure 2 below.

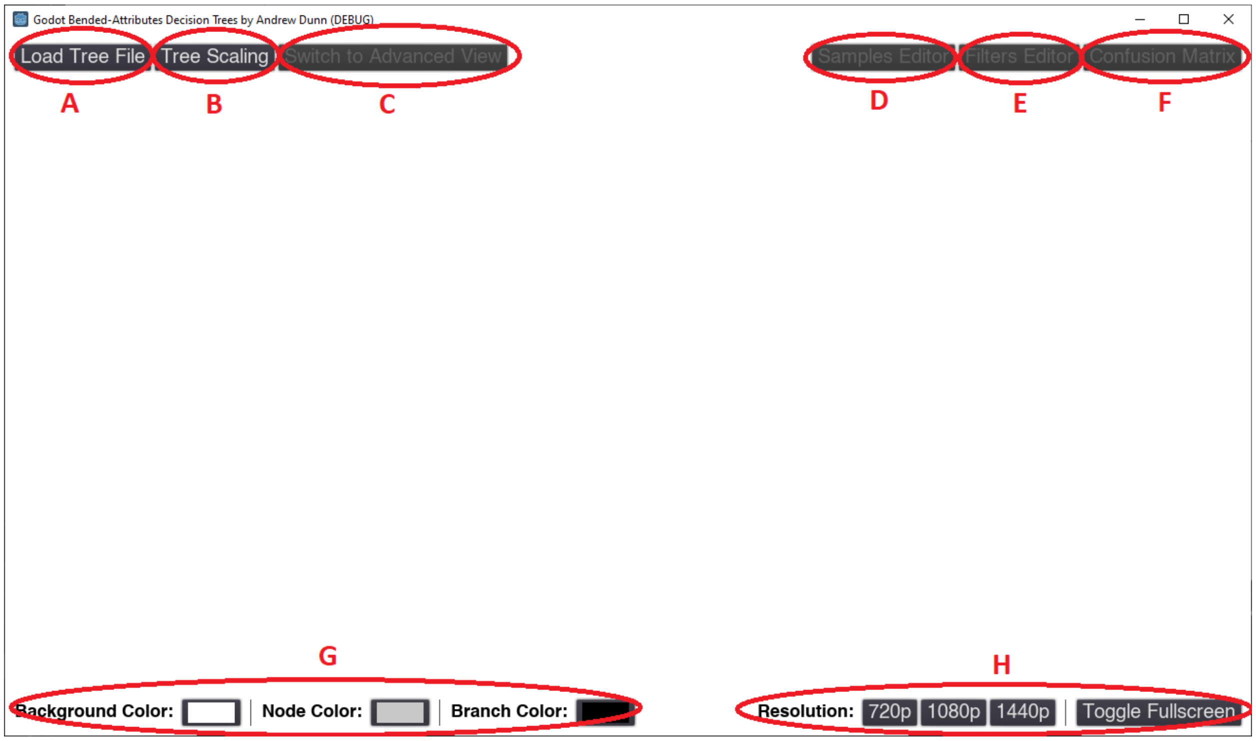


Figure 2: Application Main Screen With Important UI Elements Circled and Labeled

In figure 2 you can see the basic UI with the important elements circled in red and labeled. A short description of each UI element can be found in the table below:

Fig. Label	Name	Description
A	Load Tree File Button	Allows you to load a JSON decision tree
B	Tree Scaling Button	Changes how far apart the tree nodes are drawn to the screen
C	Advanced/Simple Mode Button	Toggles the data samples view mode
D	Samples Editor Button	Opens the Samples Editor window to manage the currently loaded samples dataset
E	Filters Editor Button	Opens the Filters Editor window to manage sample filters
F	Confusion Matrix Button	Opens the Confusion Matrix window to display sample metrics
G	UI Color Controls	Allows you to change the display color of the screen background, tree noes, and tree branches
H	Display Controls	Allows you to change the rendering resolution of the application window, and toggle Fullscreen mode. A larger resolution will make UI elements and windows appear smaller.

Table 1: Description of Application Main UI Elements

4.1 Load Tree File Button (Fig. Label A)

After loading the application, the first thing you need to do is load a binary classification decision tree from a JSON file. A few examples of JSON tree files have been included with this program, as well as their associated datasets. If you wish to create your own JSON file for some other decision tree, use the included Tanagra output parser, or create the JSON file by hand. A detailed description of both methods can be found in Section 5.1 tana2tree_godot.py and Section 6 JSON Decision Tree Format Specifications. After loading a decision tree, the tree nodes are clickable and display more information about the node, as well as allow you to display how many dataset samples are currently passing through that node.

4.2 Camera Mouse Controls

After a decision tree JSON file is loaded, you can move the tree around by left-clicking the blank background behind the tree and dragging the mouse while holding the button. You can zoom in by scrolling up using your mouse scroll wheel while the mouse is hovering above the blank background. Scroll down to zoom out.

4.3 Tree Scaling Button (Fig. Label B)

After loading a decision tree JSON file, use this button to open the Tree Scaling window to adjust the distances that tree nodes are drawn from each other. Scaling the tree up can be useful when displaying many samples that are overlapping each other on the screen.

4.4 Advanced/Simple Mode Button (Fig. Label C)

After loading a decision tree JSON file and adding at least one sample (either entered manually or loaded from a CSV dataset), you can switch between the simple and advanced view modes. The simple view mode only shows how samples travel down the decision tree to the prediction leaf node. Advanced mode displays the samples as Shifted Bended Attributes, thus visually encodes each samples attribute in relation to the tree node thresholds. If a sample connects with a tree branch close to the parent node, then that samples attribute is relatively close to the node's threshold. Likewise, if a sample connects with a tree branch far from the parent node, then the sample's attribute is relatively far from the node's threshold. See figures 3 and 4 below for example screenshots of the two modes, each displaying the same set of 8 samples. Note that in advanced mode, the little white diamonds drawn on top of the tree branches are clickable, and will open an editor window for that particular sample.

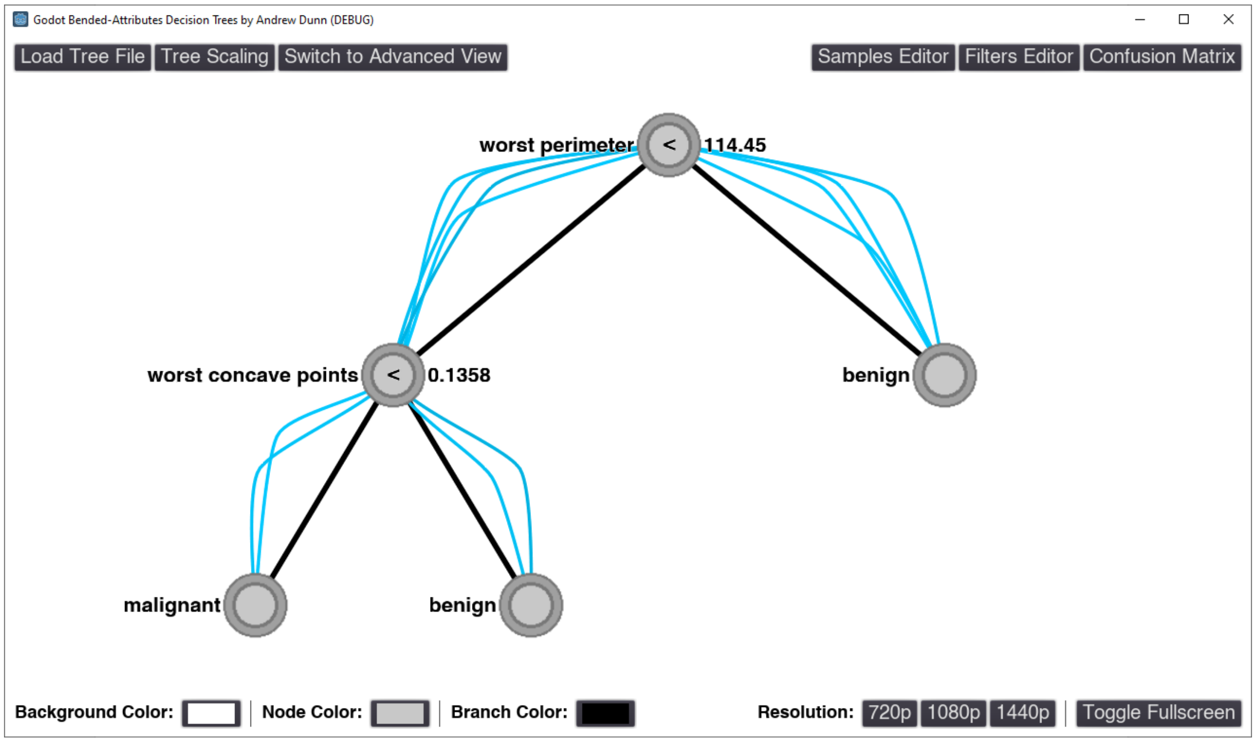


Figure 3: Example of Simple View Mode (Bended Attributes)

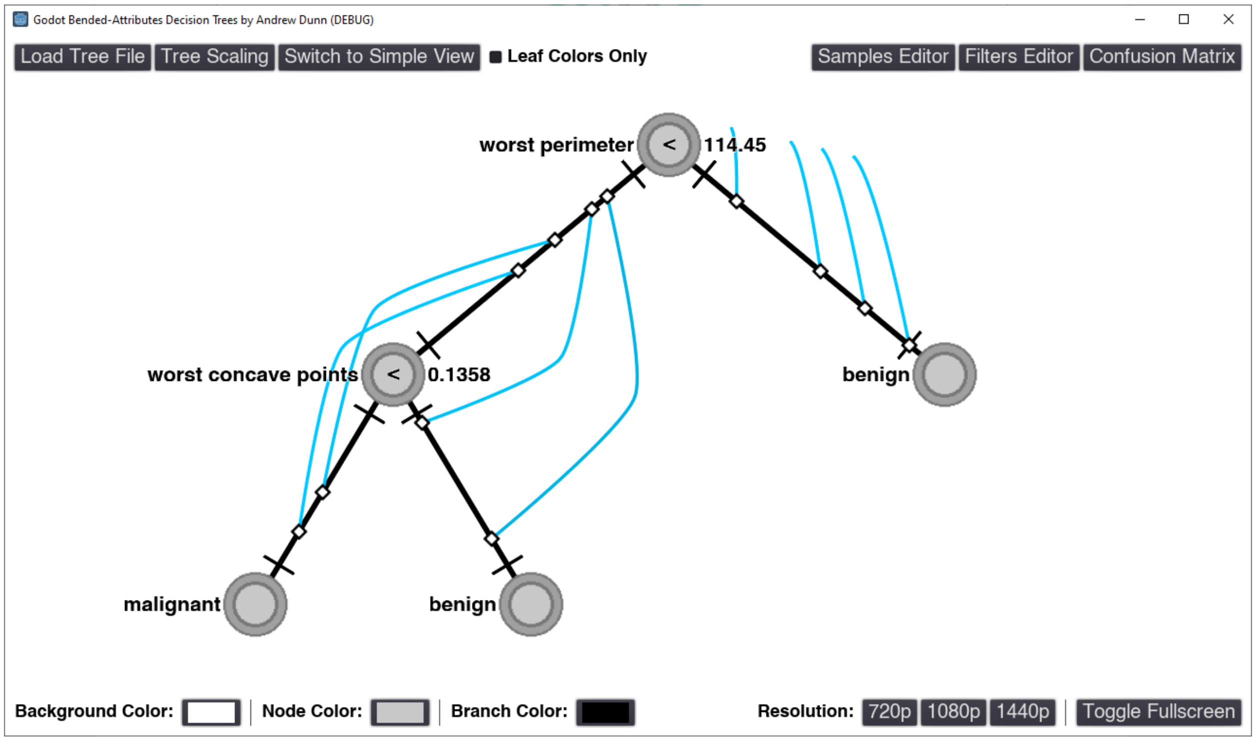
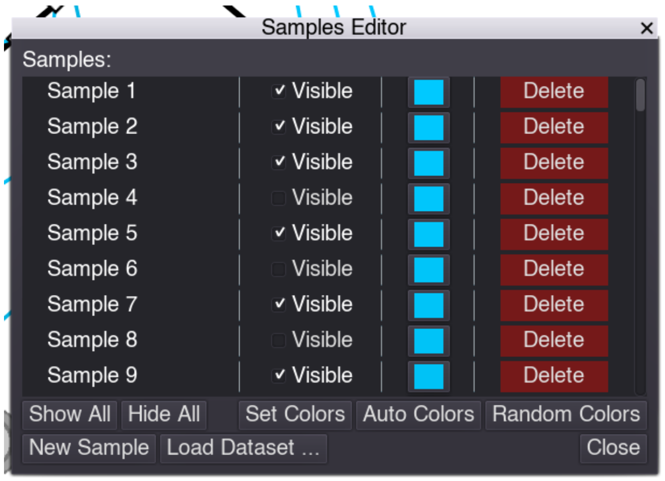


Figure 4: Example of Advanced View Mode (Shifted Bended Attributes)

4.5 Samples Editor Button (Fig. Label D)

After loading a decision tree JSON file, click the Samples Editor button to open the Samples Editor window. This window allows you to create, load, and manage dataset samples. See figure 5 below for an example screenshot.



Sample 1	✓ Visible	🟦	Delete
Sample 2	✓ Visible	🟦	Delete
Sample 3	✓ Visible	🟦	Delete
Sample 4	Visible	🟦	Delete
Sample 5	✓ Visible	🟦	Delete
Sample 6	Visible	🟦	Delete
Sample 7	✓ Visible	🟦	Delete
Sample 8	Visible	🟦	Delete
Sample 9	✓ Visible	🟦	Delete

Figure 5: Example of Samples Editor with Samples Dataset Already Loaded

Taking up most of the space in the center of the window is the Samples List, which contains a list of all currently loaded dataset samples. The first column contains the sample name. The second column is a checkbox that shows or hides the associated sample bended line on the decision tree. The third column allows you to change the sample's display color. Finally, the last column contains a delete button, and when click will delete the associated sampled from the list. Note that if you delete a sample from the sample list, it will not change the original dataset file.

Below the Samples List are several different buttons. The Show All button will make all samples in the Samples List visible on the decision tree. Likewise, the Hide All button makes all samples in the Sample List invisible. To the right of the Hide All button is three different buttons related to coloring your list of samples.

The Set Colors button allows you to set the color of all visible samples to a specific color. This is useful in combination with the Filters Editor because you can assign specific samples specific colors based on different filtering criteria. You can select two different shades of the same base color to give samples slight color variations, making it easier to differentiate between individual sample lines on the decision tree.

The Auto Colors button will automatically assign sample colors based on a selected dataset attribute. This is useful if you want to assign specific sample colors for each target class in your dataset. You can change the color Hue range assigned to the samples, as well as a saturation color ramp. Samples within the same class will have slight color variations depending on the sample saturation range, while the hue range controls the sample base color.

The Random Colors button will simply randomize the colors of all currently visible samples. This is useful if you want to differentiate between individual sample lines when you have many samples displayed on the decision tree. You can select the darkest and lightest possible colors to be randomly generated.

The New Sample button will open the Enter Samples window and allows you to manually enter sample data which will be added to the Samples List.

The Load Dataset button opens the Load CSV wizard and presents a series of dialogs that assist in loading a dataset of samples into the program. Currently only plain text file format datasets are supported. After picking a dataset file and selecting the correct delimiter used in the file, the Import Samples from CSV window will open. This window allows you to control how the dataset is loaded, including limiting the number of samples loaded, how to handle samples with missing values, and how the new samples should be named in the Samples List. If your dataset has an included name column with unique names for each sample, then it is recommended to name samples after "A column value." Otherwise, you should name the new samples after their row index.

4.6 Filters Editor Button (Fig. Label E)

After loading a decision tree JSON file and creating or loading some dataset samples, click the Filters Editor button to display the Filter Samples window. You should see a collection of filters that were automatically generated for you based on the sample predictions and sample targets (If the target column was correctly found in the dataset). These filters allow you to show or hide samples on the decision tree based on different conditions. For example, clicking the Show Only button for "benign Prediction" will make all samples that are predicted as benign visible while hiding all others. Likewise, the Hide Only button will show all samples except the benign predicted samples. If you click a filter name in the Filters List, a new window will open that allows you to view and edit the conditional expression behind that filter.

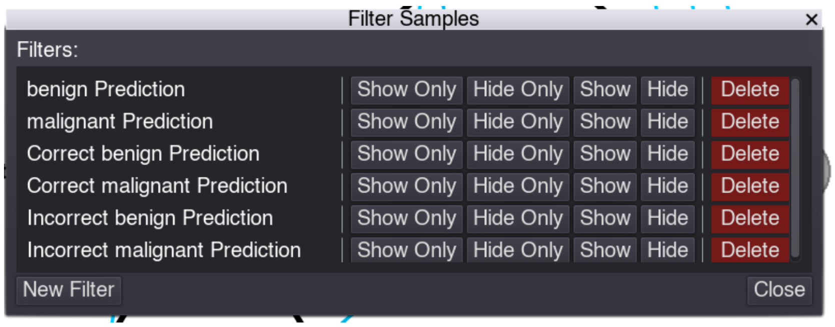

Filter Samples
Filters:
benign Prediction
malignant Prediction
Correct benign Prediction
Correct malignant Prediction
Incorrect benign Prediction
Incorrect malignant Prediction
Show Only
Hide Only
Show
Hide
Delete
New Filter
Close

Figure 6: Example of Filter Samples Window with Samples Dataset Already Loaded

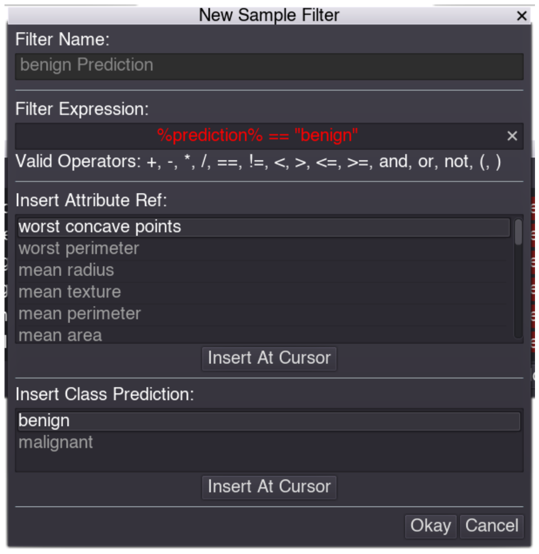
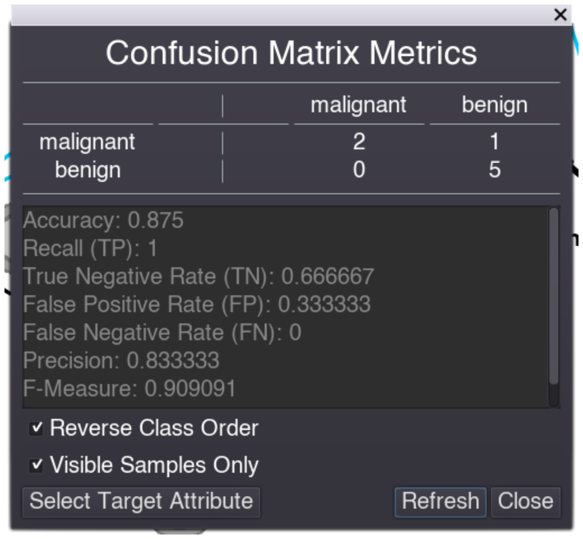


Figure 7: After Clicking "benign Prediction" Filter in Filter List

If you would like to create your own custom filter, click the New Filter button and use the "Insert At Cursor" buttons to insert references to attributes and class predictions into the expression. This allows you to make complex conditional expressions using combinations of attribute and prediction references with the supported operators. For example, if you wanted to filter all "benign" predicted samples with a "worst parimeter" value less-than 114.45, you could enter the following expression: **%prediction% == "benign" and $worst perimeter$ < 114.45**

4.7 Confusion Matrix Button (Fig. Label F)

Click the Confusion Matrix button after loading a decision tree JSON file and sample dataset to display the Confusion Matrix Metrics window. After the window opens, you must click the Select Target Attribute button and select the attribute from your dataset that contains the sample target classes. Note that the selected target column must contain class data in the same format as the leaf-class names used in the decision tree. After selecting a target attribute column, the confusion matrix will be computed for the currently visible samples, as well as the accuracy and other metrics if there are only two classes in your decision tree.



Confusion Matrix Metrics

	malignant	benign
malignant	2	1
benign	0	5

```
Accuracy: 0.875
Recall (TP): 1
True Negative Rate (TN): 0.666667
False Positive Rate (FP): 0.333333
False Negative Rate (FN): 0
Precision: 0.833333
F-Measure: 0.909091
```

✓ Reverse Class Order

✓ Visible Samples Only

Figure 8: Example of Confusion Matrix Metrics for Two-Class Decision Tree

4.8 UI Color Controls (Fig. Label G)

You may use the UI Color Controls to change how the background color and tree colors are displayed. This can be useful if you want to make the tree nodes semi-transparent.

4.9 Display Controls (Fig. Label H)

The Display Controls allow you to change the base resolution of the main screen, and toggle Fullscreen mode on or off. Increasing the resolution is useful because it makes the UI elements and windows smaller, thus making them take up less screen real-estate. It is recommended to set the resolution close to your monitor display resolution.

5 Additional Included Tools

Included with the main application are two utility Python scripts: tana2tree_godot.py and csv_test_train_split.py. These Python scripts were created to make the preparation of decision trees and sample datasets for our application easier.

5.1 tana2tree_godot.py

The tana2tree_godot.py Python script can parse the decision tree output from Tanagra into a JSON file than can be opened in our application. This script requires an external library created by one of our project group members, Brad Reeves. To install the library, open a terminal and run: **pip install tana2tree**.

5.1.1 Preparing Tanagra Data

The first step to converting a Tanagra decision tree is to generate a decision tree using Tanagra. The details of this is out of the scope of this paper. We recommend using the ID3 decision tree classifier. After generating a decision tree in Tanagra, view the results then click the “Component” menu item at the top of the window, then select “Copy results”. This will copy Tanagra’s HTML output to your clipboard.

Next, open a text editor such as Notepad and paste the Tanagra output into the window. You should see the HTML code output from Tanagra in the text editor. Next, save this to a file, for example **tanagra_data.txt.**

5.1.2 Converting A Tanagra Decision Tree

After saving and closing the text file, open a terminal in the directory containing our tana2tree_godot.py script. Next, type in the following command: **python ./tana2tree_godot.py ./tanagra_data.txt ./output_tree.json true**

Note that this assumes your tanagra_data.txt file is in the same directory as the tana2tree_godot.py script. The ./**output_tree.json** argument is the output JSON file name. Finally, the **true** at the end tells tana2tree_godot.py that you would like the output tree optimized. This automatically combines unnecessary tree leaves when both leaf siblings are of the same class. If you don't want this functionality, then exclude the **true** argument. After the program finishes running, you should have a new file created in the directory named output_tree.json which contains the tree data in the correct format for our application.

5.2 CSV_TEST_TRAIN_SPLIT.PY

In addition to a Tanagra parser script, we have included another Python script that can automatically split a simple text file dataset (delimited columns) into separate training and testing files. This is useful because training a Tanagra decision tree on the training data set only is a bit of a pain if the dataset is not split into two separate files. After splitting the dataset, you can load the training dataset into Tanagra for training your decision tree, then load either the training or testing (or both) data into our application. To use this script, first you must have a dataset that is in a simple text format, such as CSV (comma separated values) or TXT (tab separated values). You can open a dataset in Excel and save it as either of these formats easily.

Once you have a text file dataset, you can randomly split it into training and testing dataset by opening a terminal in same directory as our csv_test_train_split.py script, then running the following command:

Python ./csv_test_train_split.py ./my_dataset.csv "," 0.2

Note that this assumes your dataset file is named "my_dataset.csv" and is contained within the same directory as the Python script. The **","** argument is the delimiter used in the file. In this case, we are using a comma because it is a CSV (comma separated values) file. The **0.2** argument is the ratio of samples that will be placed in the test set. Here we selected 20%. After running this command, you should see two new files created named "my_dataset_train.csv" and "my_dataset_test.csv" which will contain the randomly split training and testing sample data.

Important note: Tanagra does not support CSV files, only Tab Separated Values (TXT) files. If you want to use this script to split a tab-separated dataset in Windows, you can use **"`t"** to represent a tab character in the Powershell terminal. Just pressing the tab button in the terminal will not work. The little hyphen character is the key directly above the Tab key on a standard QWERTY keyboard.

6 JSON Decision Tree Format Specifications

Our application uses the JSON standard format for storing the decision tree data. More information about the JSON data format can be found on Wikipedia: https://en.wikipedia.org/wiki/JSON

Every JSON decision tree needs to have a root dictionary named "tree_root". For example, every JSON tree file should begin like this:

```
{
    "tree_root" : {

(…)
```

Inside of the “tree_root” dictionary you should have the data for single tree node, representing the root node of the decision tree. The format of a single (non-leaf) node looks like this:

```
“attribute”: “worst perimeter”,
“threshold”: 114.45,
“operator”: “<”,
“child_left”: { … },
“child_right”: { … }
```

Where “attribute” is the name of the dataset attribute that is split on the node. The “threshold” value is the splitting point of the real-numbered attribute. The “operator” is the conditional operator that is used when predicting samples for the node. The left child branch is always followed when the condition expression is true. Likewise, the right child branch is followed if the condition expression is false. In this particular example, if a sample has less than 114.45 for the “worst perimeter” attribute, then the sample will follow along the left child branch. Otherwise the right child branch will be followed. Supported operators are: <, <=, >, and >=. Note that the data format above is reserved solely for non-leaf tree nodes. Leaf nodes use a different format:

“leaf_class”: “benign”

Where “leaf_class” is the predicted class for that child leaf node. Note that this data will be inside a parent node’s “child_left” or “child_right” dictionary. Here is a complete example of a basic JSON tree file to put this all together:

breast_cancer_tree.json

```
{
    "tree_root": {
        "attribute": "worst perimeter",
        "threshold": 114.45,
        "operator": "<",
        "child_left": {
            "attribute": "worst concave points",
            "threshold": 0.1358,
            "operator": "<",
            "child_left": {
                "leaf_class": "malignant"
            },
            "child_right": {
                "leaf_class": "benign"
            }
        },
        "child_right": {
            "leaf_class": "benign"
        }
    }
}
```

The above JSON tree file will produce the following tree in our application:

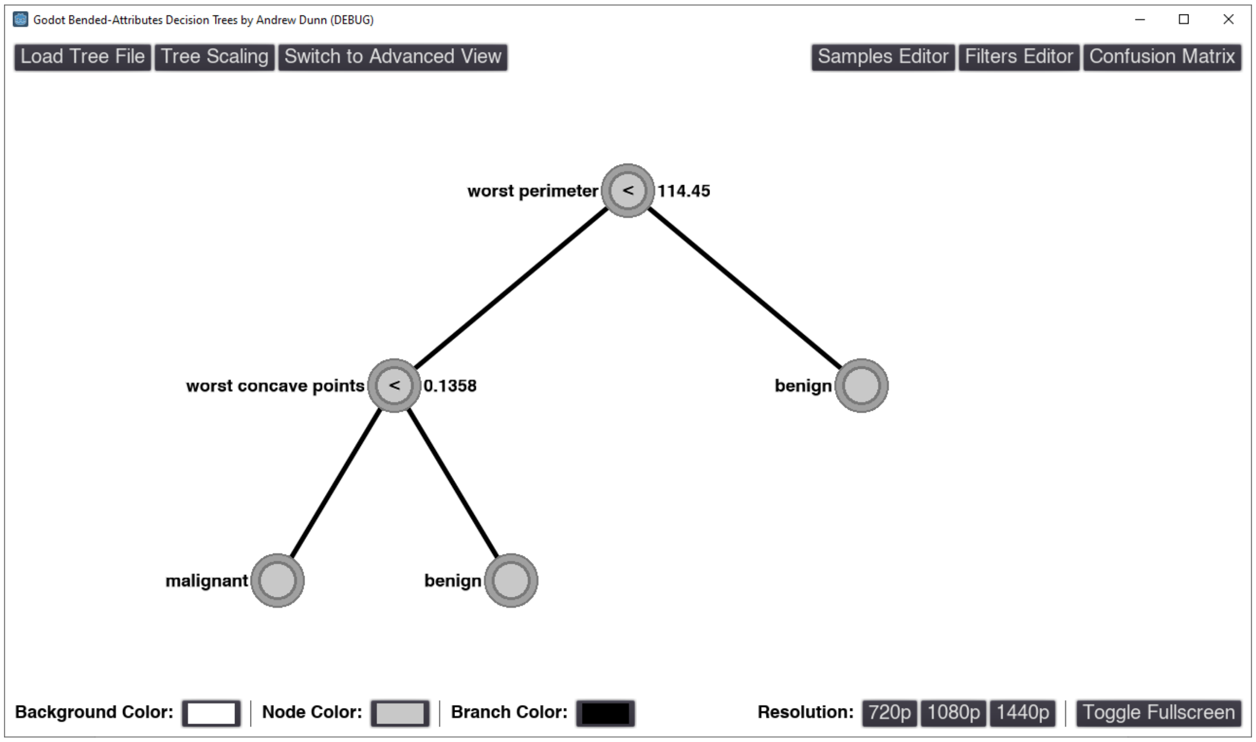


Figure 9: Application with breast_cancer_tree.json Loaded