Green University of Bangladesh

Department of Computer Science and Engineering (CSE)
Faculty of Sciences and Engineering (FSE)
Semester: (Spring, Year: 2025), B.Sc. in CSE (Day)

ARFF Tree Explorer

Course Title: Artificial Intelligence Lab
Course Code: CSE 316
Section: 222-D2

Students Details

Name	ID
Md. Zehadul Islam	222902069
Md. Abdullah Al Moin	222902070

Submission Date: 15.05.2025
Course Teacher's Name: Md. Abu Rumman Refat

[For teachers use only: Don't write anything inside this box]

Lab Project Status	
Marks:	**Signature:**
Comments:	**Date:**

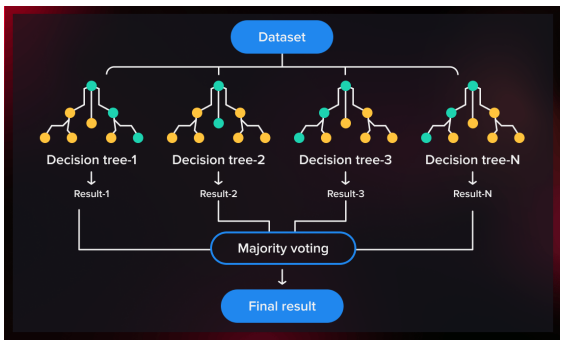
Contents

Chapter 1

Introduction

1.1 Overview

This project **"ARFF Tree Explorer"**focuses on building an interactive desktop-based application for visualizing machine learning models using ARFF (Attribute-Relation File Format) data. This application is built entirely in Python, using technologies such as Tkinter for the GUI, Pandas for data handling, and Scikit-learn for model training and visualization. It allows users to upload ARFF files and visualize various decision-making models like J48 (Decision Tree), Random Forest, REPTree, Decision Stump, and Logistic Model Tree through clear and interactive tree diagrams. The tool supports data preprocessing (including encoding categorical features), model training, and tree visualization using Matplotlib. The goal is to provide a user-friendly interface for educational, analytical, or research purposes where users can explore how decision trees are formed from ARFF datasets. [**?**] [1] [2]



1.2 Motivation

With the increasing use of machine learning and data mining techniques in education and research, there is a need for tools that make it easier to understand complex models. While platforms like Weka provide GUI-based access to ARFF files, many users prefer open-source, code-based environments. The ARFF Tree Explorer was developed to bridge this gap — offering a lightweight, Python-only tool that lets users quickly load

ARFF files, train decision models, and visualize the results in an intuitive manner. This project is especially useful for students, educators, and data science enthusiasts who want to explore classification models visually without switching between platforms.

1.3 Problem Definition

There are many tools for machine learning, but very few of them provide visual interaction with decision tree models using ARFF files in a pure Python environment. Most either require advanced coding or lack graphical representation. The challenge is to create a solution that is both technically robust and easy to use, especially for those working with ARFF files, which are common in Weka and academic datasets.

1.3.1 Problem Statement

The core problem addressed in this project is the lack of simple, open-source tools for visually exploring decision tree models trained on ARFF files. By offering model training, visualization, and support for multiple tree types in a single Python interface, the project aims to provide a practical and accessible solution for both learning and analysis.

1.3.2 Complex Engineering Problem

To further elaborate, Table 1.1 summarizes the attributes of the complex engineering problem addressed by this project:

Name of the P	**Attributes**	**Explanation of how to address**
P1: Depth of knowledge required	Machine learning, Python GUI, ARFF format	Understanding how to preprocess data, train ML models, and visualize trees in Python
P2: Range of conflicting requirements	Simplicity vs. functionality	Designing an interface that is simple yet supports multiple ML models and visualizations
P3: Depth of analysis required	Tree structure understanding	Ensuring that users can explore tree depth, splitting criteria, and node decisions interactively.
P4: Familiarity of issues	GUI design, data handling	Handling ARFF parsing errors, encoding, and GUI responsiveness
P5: Extent of applicable codes	Use of open-source libraries	Ensuring that all components use reliable Python libraries (like sklearn, pandas, matplotlib)
P6: Extent of stakeholder involvement and conflicting requirements	Educators, learners, analysts	Gathering feedback from students and researchers for usability improvements
P7: Interdependence	Integration of multiple components	Linking GUI, data loading, model training, and visual output into a cohesive app

Table 1.1: Attributes of the complex engineering problem

1.4 Design Goals/Objectives

The main goal of the ARFF Tree Explorer is to provide an effective and interactive way to explore decision-making models from ARFF files in Python. Below are the key design objectives:

- **Objectives Of My Project Are :**

User-Friendly Interface:

- Provide a clean and intuitive interface using Tkinter
- Allow easy file uploading and model selection

Model Training and Visualization:

- Train models like J48, REPTree, Random Forest, Stump, and LMT
- Visualize decision trees using Matplotlib.

Data Processing:

- Handle categorical data using Label Encoding.
- Ensure robustness in ARFF data parsing.

Modularity and Scalability:

- Structure the code for easy addition of more models.
- Support multiple trees (e.g., from Random Forest) with separate visualizations.

Educational Utility:

- Serve as a learning tool for students to understand how decision trees work.
- Make ML more accessible without coding in depth

Platform Independence:

- Ensure the application runs on any OS with Python installed.
- Use only Python-native libraries.

1.5 Application

The ARFF Tree Explorer has a variety of practical applications:

- **Educational Use:**
  - Teach students how decision trees are formed based on real datasets.
- **Model Debugging:**
  - Understand how models make decisions by visually inspecting splits and nodes.
- **Research and Prototyping:**
  - Quickly test ARFF datasets with various models before deeper experiments.
- **Weka Alternative:**
  - Use as a lightweight alternative to Weka for basic decision tree analysis in Python.
- **Data Visualization:**
  - Present decision logic visually for reports, presentations, or academic writing.
- **Learning Tool:**
  - Help new data science learners get comfortable with classification models and tree structures.

Chapter 2

Design/Development/Implementation of the Project

2.1 Introduction

This project, "ARFF Tree Explorer," is a Python-based graphical application designed to explore, compare, and visualize classification algorithms using ARFF (Attribute-Relation File Format) datasets. The project integrates several machine learning models from the scikit-learn library and provides a simple interface built with Tkinter. Users can load ARFF files, compare different classification models, and visualize decision trees to better understand the structure and performance of classifiers.

2.2 Project Details

The project integrates two major functionalities:

1. A classification algorithm comparison module.
2. A decision tree visualization module.

The application processes ARFF-formatted datasets and allows users to interactively evaluate machine learning models.

2.2.1 Classifier Comparison Module

- **Input:** ARFF dataset selected via file dialog.
- **Process:** Data is encoded and passed into various classification models. Each model is evaluated using 5-fold cross-validation to compute accuracy.
- **Output:** Displays the accuracy results of each classifier in a Tkinter text widget.

2.2.2 Decision Tree Visualization Module

- **Input:** User-selected classifier model from a dropdown list.
- **Process:** The selected model is trained on the dataset, and the tree is visualized using matplotlib.
- **Output:** A visual representation of the decision tree appears in a separate pop-up window.

2.3 Implementation

2.3.1 Workflow

The implementation consists of the following key steps:

1. **Data Loading:**
   - Prompt the user to select an ARFF file.
   - Load the dataset using the `scipy.io.arff` module and convert it to a pandas DataFrame.
   - Use `LabelEncoder` to handle categorical attributes.
2. **Algorithm Comparison:**
   - Define multiple classifiers: J48 Tree, REPTree, Random Forest, Decision Stump, Logistic Regression.
   - Perform 5-fold cross-validation on each classifier.
   - Display accuracy scores in the GUI.
3. **Decision Tree Visualization:**
   - Let user select a model from a combo box.
   - Train the model on the dataset.
   - Display the decision tree using `plot_tree()`.

2.3.2 Tools and Libraries

- Python: Main programming language.
- Tkinter: GUI toolkit for creating the user interface.
- Pandas: For handling and processing the dataset.
- scikit-learn: Provides classification models and evaluation tools.
- matplotlib: Used for tree visualization.
- scipy.io.arff: For loading ARFF files.

2.3.3 Implementation Details

ARFF File Loading

- Opens a file picker dialog using `filedialog`.
- Loads the ARFF file and parses it into a DataFrame.
- Encodes categorical columns using `LabelEncoder`.

Classifier Comparison

- **Models:**
  - J48 Tree → `DecisionTreeClassifier` with entropy
  - REPTree → `DecisionTreeClassifier` with max depth and leaf constraints
  - Random Forest → `RandomForestClassifier`
  - Decision Stump → Shallow `DecisionTreeClassifier`
  - Logistic Regression (as LMT approximation)
- Cross-validation is done using `cross_val_score`.

Tree Visualization

- Based on user selection from the combo box.
- Trains model and uses `plot_tree()` to visualize the tree.
- Random Forest shows the first estimator in the forest.

2.4 Algorithms

2.4.1 Classifier Comparison Algorithm

1. Load ARFF file and convert to DataFrame.
2. Encode categorical columns using `LabelEncoder`.
3. Define machine learning models (DecisionTree, RandomForest, etc.).
4. Perform 5-fold cross-validation for each model.
5. Display accuracy results in the output area.

2.4.2 Tree Visualization Algorithm

1. Load and encode data.
2. Let the user select a model (e.g., J48, REPTree).
3. Train the selected model.
4. Use `matplotlib` to show the decision tree diagram.

2.5 Flow-Chart

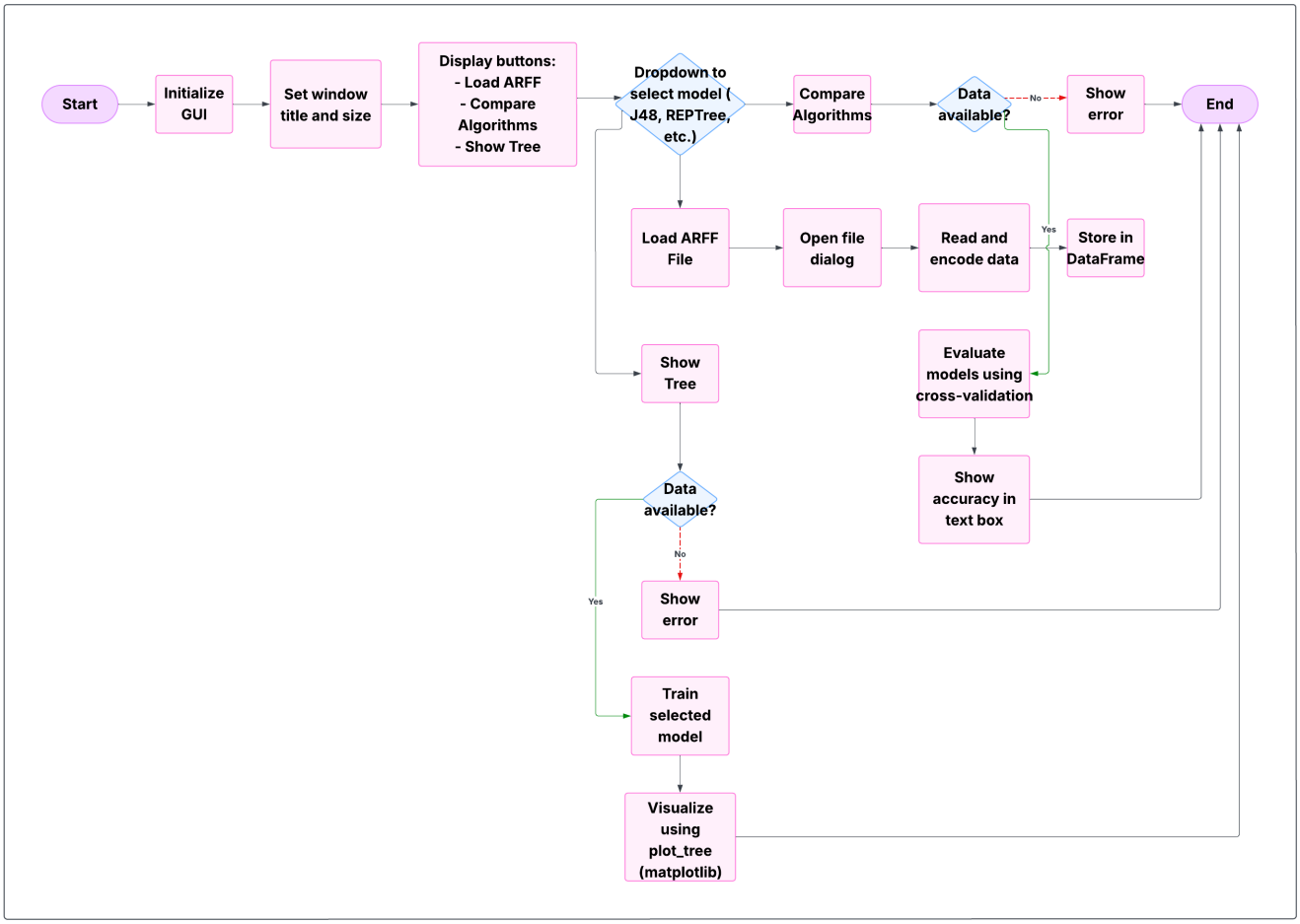


Figure 2.1: Flow-Chart for Arff Tree Explorer project

2.6 Code Implementation

The project is implemented in a modular way, using object-oriented principles.

2.6.1 ARFF File Loading

- GUI allows users to select ARFF files using file dialogs.

```
def load_arff(self):  1 usage
    file = filedialog.askopenfilename(filetypes=[("ARFF files", "*.arff")])
    if file:
        self.file_path = file
        with open(file, 'r') as f:
            raw_data = arff.load(f)
        df = pd.DataFrame(raw_data['data'], columns=[attr[0] for attr in raw_data['attributes']])
        for col in df.columns:
            if df[col].dtype == object:
                df[col] = LabelEncoder().fit_transform(df[col])
        self.data = df
        self.output_text.delete( index1: 1.0, tk.END)
        self.output_text.insert(tk.END,  chars: f"\u2705 Loaded ARFF file: {file}\n")
```

Figure 2.2: Loading and Preprocessing ARFF Files

2.6.2 Algorithm Comparison

- Cross-validation is applied to multiple models.
- Outputs are printed to a multi-line text widget.

```
Arff.py
12    class TreeApp:  1 usage
56        def compare_algorithms(self):  1 usage
57            if self.data is None:
58                messagebox.showerror( title: "Error",  message: "Please load an ARFF file first.")
59                return
60
61            X = self.data.iloc[:, :-1]
62            y = self.data.iloc[:, -1]
63
64            models = {
65                "J48 Tree": DecisionTreeClassifier(criterion='entropy'),
66                "REPTree": DecisionTreeClassifier(max_depth=5, min_samples_leaf=5),
67                "Random Forest": RandomForestClassifier(n_estimators=5),
68                "Decision Stump": DecisionTreeClassifier(max_depth=1),
69                "Logistic Model Tree (LMT approx)": LogisticRegression(max_iter=1000)
70            }
71
72            self.output_text.delete( index1: 1.0, tk.END)
73            self.output_text.insert(tk.END,  chars: f"{'Model':<40}{'Accuracy (CV)':>20}\n")
74            self.output_text.insert(tk.END,  chars: f"{'-'*60}\n")
75
76            for name, model in models.items():
77                try:
78                    scores = cross_val_score(model, X, y, cv=5)
79                    avg_acc = scores.mean()
80                    self.output_text.insert(tk.END,  chars: f"{name:<40}{avg_acc*100:>17.2f}%\n")
```

Figure 2.3: Comparison Output in GUI

2.6.3 Tree Visualization

- Displays decision trees graphically using matplotlib.

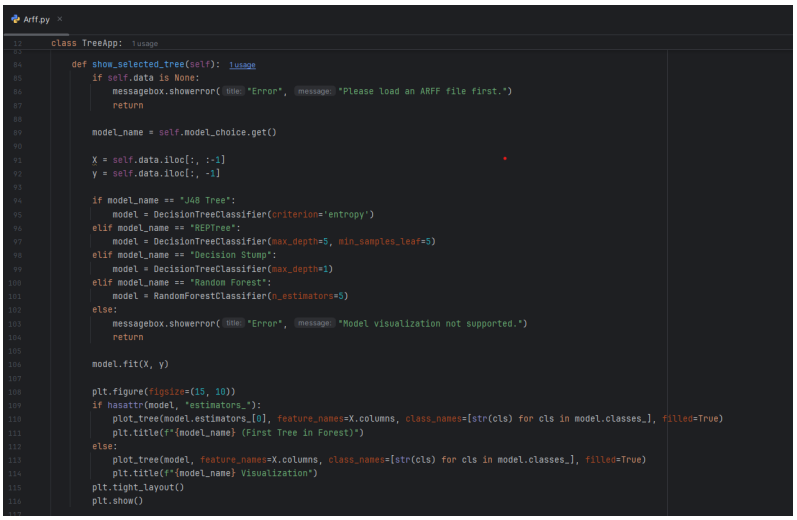


Figure 2.4: Visualization of Selected Decision Tree

Chapter 3

Performance Evaluation

3.1 Simulation Environment/Simulation Procedure

The ARFF Tree Explorer project was developed in Python and tested using various ARFF datasets in a local desktop environment. The system was built with a GUI that allows interactive user input, real-time classifier evaluation, and decision tree visualization.

3.1.1 Setup and Testing Procedure

The following setup steps were performed during development and testing of the project:

- **Development Environment:**
  - Language: Python 3.x
  - IDE: VS Code
  - Libraries: `pandas`, `scikit-learn`, `scipy.io.arff`, `matplotlib`, `tkinter`
- **Setup Steps:**
  - Installed required libraries using `pip`.
  - Implemented the GUI using Tkinter for interaction.
  - Connected back-end classification models using scikit-learn.
  - Verified ARFF file compatibility with various datasets.
- **Testing Procedure:**
  - Tested with multiple ARFF datasets from the UCI repository.
  - Verified classifier comparison results using known datasets.
  - Ensured decision tree visualization displayed correct tree structure.
  - Checked GUI responsiveness and exception handling.

3.2 Results Analysis/Testing

3.2.1 Home-page:

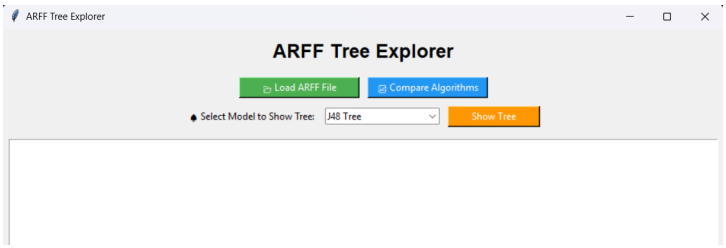


Figure 3.1: Welcome to Tree Explorer

3.2.2 Classifier Comparison Module

- **Interface:** Allowed users to load ARFF files and compare classifiers.
- **Functionality:** Performed 5-fold cross-validation on each model and displayed accuracy.

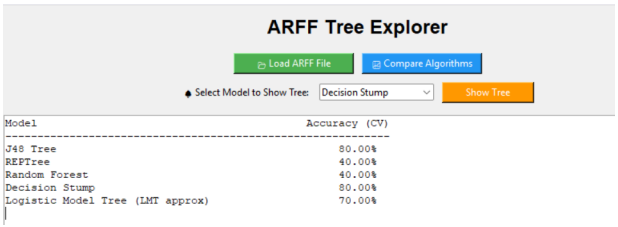


Figure 3.2: Visualization of Comparison Module

3.2.3 Decision Tree Visualization Module

- **Interface:** Provided a dropdown to select a model for visualization.
- **Observation:** Visualized trees showed branching based on feature splits and matched the training data.

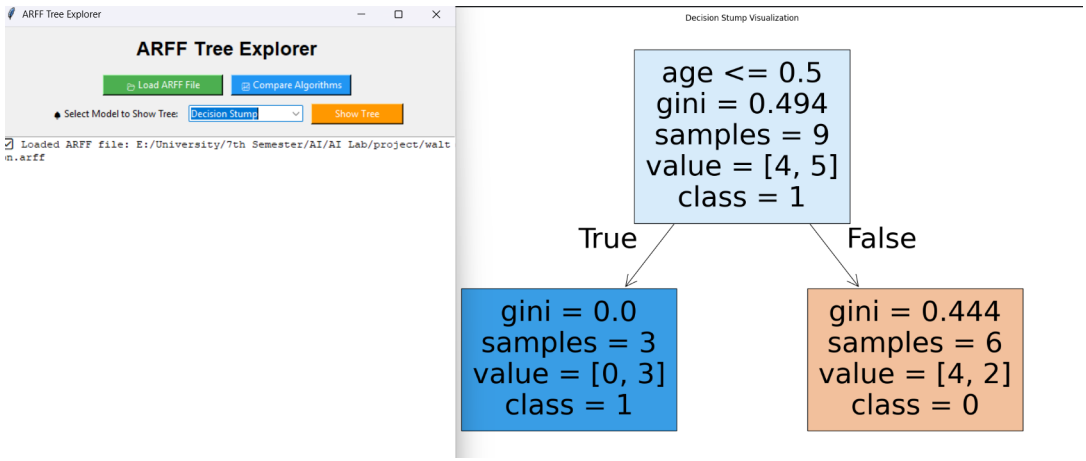


Figure 3.3: Decision Tree Visualization

3.2.4 J48 Tree

– **Description:** The J48 tree is an implementation of the C4.5 decision tree algorithm. It builds a tree by recursively splitting the data based on information gain.

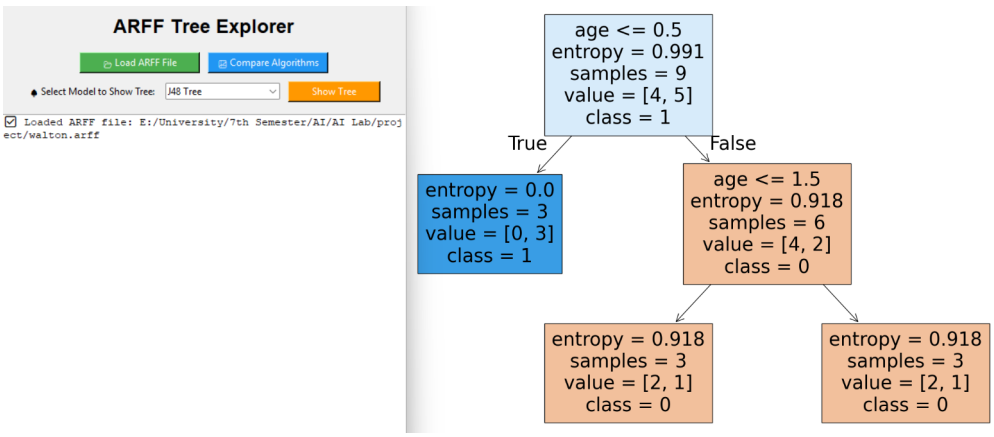


Figure 3.4: J48 Tree Visualization

3.2.5 REP Tree

– **Description:** REPTree uses reduced-error pruning and builds a fast decision tree based on variance reduction.

– **Observation:** This tree had a simpler structure and fewer branches due to pruning.

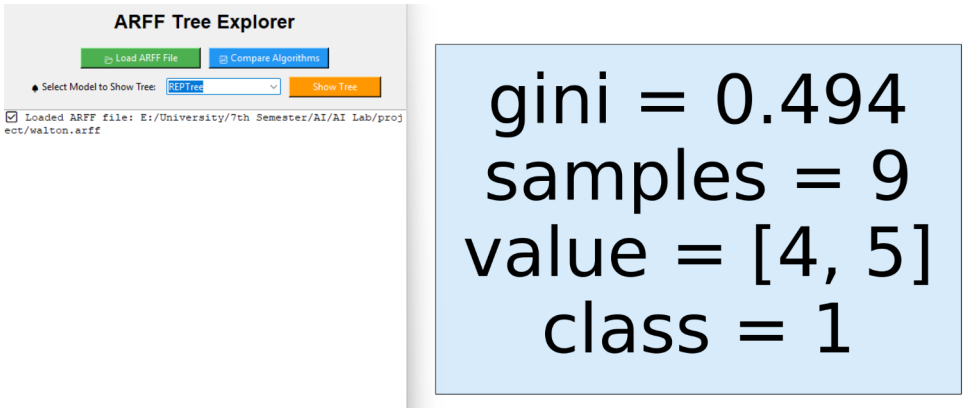


Figure 3.5: REPTree Visualization

3.2.6 Random Forest Tree

- **Description:** Random Forest is an ensemble learning method that combines multiple decision trees for improved accuracy.
- **Observation:** One representative tree from the forest was visualized, showing depth and complexity compared to single trees.

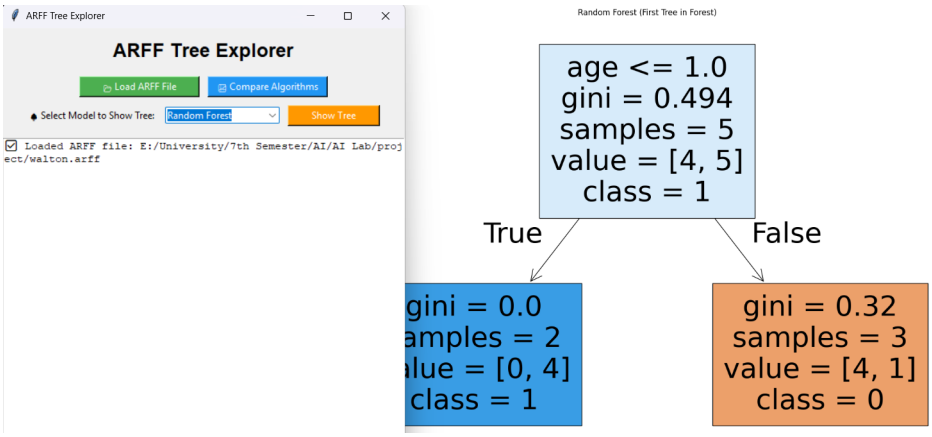


Figure 3.6: Random Forest Tree Visualization

3.2.7 Real-Time Interaction

- The GUI dynamically responded to file input and model selection.
- Errors were handled gracefully, including invalid file types or missing labels.
- The interface updated outputs in real time without requiring a program restart.

3.3 Results Overall Discussion

3.3.1 How It Worked

- The system successfully accepted ARFF files and parsed them into a structured format.
- All classifiers returned accuracy results after cross-validation.
- The decision tree visualizer correctly rendered the selected model.

3.3.2 Problems Found

- **ARFF Compatibility:** Some ARFF files with missing values caused errors.
- **Model Limitations:** Certain classifiers struggled with non-numeric features without encoding.
- **GUI Freezing:** Occasionally froze when processing large datasets due to lack of threading.

3.3.3 Complex Engineering Problem Discussion

- Parsing ARFF data and encoding multiple categorical columns correctly was challenging.
- Designing a modular interface that connects multiple models with a GUI took careful planning.
- Maintaining GUI responsiveness while training models required optimization and event-driven design.

Chapter 4

Conclusion

4.1 Discussion

This project demonstrated the successful development of a GUI-based system to compare classification algorithms and visualize decision trees using ARFF files. The tool enables users to explore machine learning concepts through interactive visual outputs and quantitative model comparisons.

4.2 Limitations

- Dependent on correctly formatted ARFF files.
- Lacked advanced GUI features like threading or progress indicators.
- Did not support real-time model tuning or hyperparameter adjustment.

4.3 Scope of Future Work

- Add support for more file types (e.g., CSV, XLSX).
- Implement threading to avoid GUI freezing on large datasets.
- Integrate hyperparameter tuning options for each classifier.
- Include support for exporting results and saving decision tree images.
- Improve decision tree visuals with node-specific information and interactive zooming.

References

[1] Tkinter gui programming. https://docs.python.org/3/library/tk.html.

[2] Scikit-learn documentation. https://scikit-learn.org/.