**COMP338 - Project #2 Due: Saturday, Jan 12, 2025**



**DEPARTMENT OF COMPUTER SCIENCE**

**COMP338 - Artificial Intelligence**

**Course Project 2**

**Prepared by**

**Name:** Malak Mustafa **– Student ID:** 1212986

**Name:** Layal Abed **– Student ID:** 1220480

**Instructor’s Name:** Mohammad Helal

**Section: 2**

**Date: 12/1/2024**

**Table of contents**

**[Abstract: 1](#_Toc10728)**

**[Theory : 1](#_Toc13767)**

**[Procedure and Discussion: 1](#_Toc27287)**

[1. How we Print out the distribution of the target class 1](#_Toc3988)

[2. How we Split dataset into training and testing : 2](#_Toc12355)

[3. How Evaluate the model and print the accuracy and the F1-Score of the test set : 3](#_Toc20850)

[4. How we generated decision tree of model M1 and M2: 3](#_Toc3491)

**[The result : 5](#_Toc15751)**

**Table of Figure**

[Figure 1 : Print the distribution of the target class 2](#_Toc18111)

[Figure 2 : Split dataset into training and testing 2](#_Toc16941)

[Figure 3 :Evaluate the model and print the accuracy and the F1-Score of the test set 3](#_Toc29889)

[Figure 4 :Plot mode 2 4](#_Toc17359)

[Figure 5 :Plot mode 1 4](#_Toc5272)

**Abstract:**

Regularization tree (decision tree) algorithm will be used to classify the data using Weka library. Training model using C4.5 algorithm, Data for weight analysis across metrics such as Accuracy and F1-Score. Training of the NRF dataset will be planned according to different layouts (70-30% and 50-50%).

**Theory:**

A decision tree is a machine learning technique used for both classification and regression tasks. It works by dividing the data into smaller subsets based on the values of features, creating a tree-like structure. In this structure, each node represents a decision based on a specific feature, while the leaf nodes represent the final prediction. Decision trees are easy to understand and are typically built using algorithms such as C4.5, which selects the most relevant feature to split the data. These models are effective for making predictions, particularly when there are complex relationships between features and outcomes, and can handle both numerical and categorical data.

**Procedure and Discussion:**

1. How we Print out the distribution of the target class

The printClassDistribution method calculates and prints the distribution of the target class in a dataset. It first initializes an array called counts to keep track of the frequency of each class. Then, it iterates through all the instances in the dataset, incrementing the count for the corresponding class of each instance. Once all instances are processed, the method calculates the percentage of each class by dividing its count by the total number of instances and multiplying the result by 100. Finally, it prints out the name of each class along with its corresponding percentage, providing a clear view of how the classes are distributed in the dataset.

This is the result we got:

Class Distribution:

unacc: 70.02314814814815

acc: 22.22222222222222

good: 3.9930555555555554

vgood: 3.761574074074074

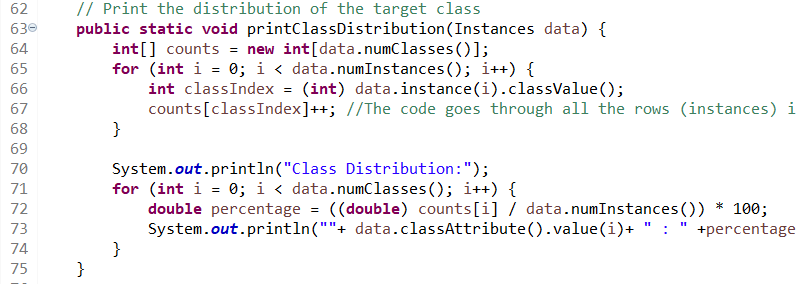


Figure 1: Print the distribution of the target class

1. How we Split dataset into training and testing:

The splitDataset method is designed to divide a dataset into two subsets: a training set and a test set. First, the method shuffles the data to ensure randomness before splitting. This is achieved using the Randomize filter from Weka. After shuffling, the method calculates the size of the training set by multiplying the total number of instances by the specified trainingPercentage. In this case, a value of 0.7 (70%) is used, meaning 70% of the dataset is allocated to the training set. The remaining 30% is automatically assigned to the test set. The dataset is then split into two Instances objects: one for training (train) and one for testing (test). The method ultimately returns an array containing these two subsets, allowing for efficient model training and evaluation.

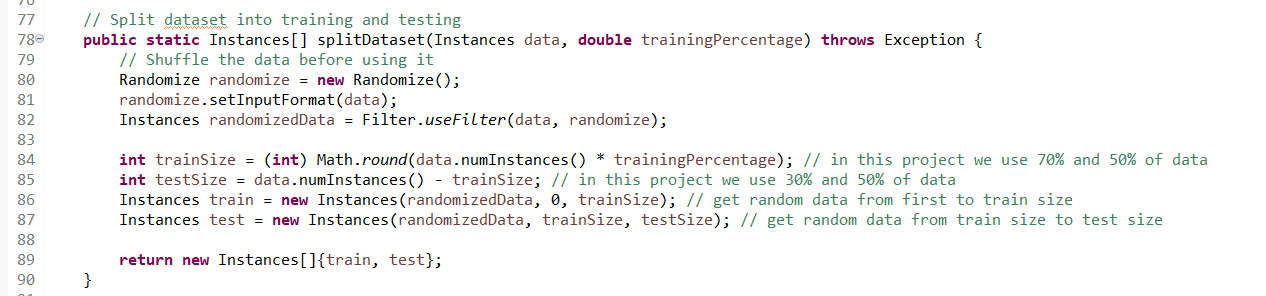


Figure 2 : Split dataset into training and testing

1. How Evaluate the model and print the accuracy and the F1-Score of the test set :

The evaluateModel method is designed to assess the performance of the trained J48 decision tree model on a test dataset. It leverages the Evaluation class from the Weka library to carry out this evaluation. Initially, the method creates an Evaluation object for the test data, then it uses the evaluateModel function to compare the model's predictions with the actual values in the dataset. It calculates and prints two key metrics:

**1.** **Accuracy**: The percentage of correctly classified instances in the test set.

**2.** **F1-Score (Weighted)**: The harmonic means of precision and recall, adjusted for class imbalance.

This is the result we got:

Evaluating Model M1 (70-30 Split):

Accuracy: 92.27799227799228

F1-Score (Weighted): 0.9216496932219128

Evaluating Model M2 (50-50 Split):

Accuracy: 90.85648148148148

F1-Score (Weighted): 0.9078953783031594

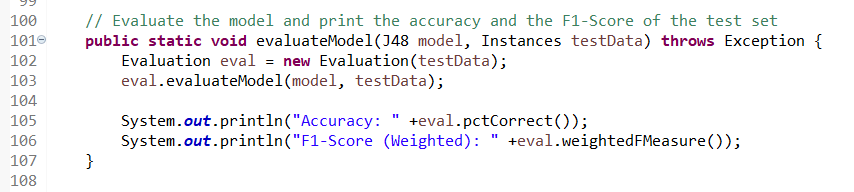


Figure 3 :Evaluate the model and print the accuracy and the F1-Score of the test set

1. How we generated decision tree of model M1 and M2:

The plotTree method visualizes the decision tree model generated by the J48 algorithm. It creates a JFrame window to display the tree, using the TreeVisualizer class. The visualizer is initialized with the model's graph and a PlaceNode1 layout for positioning the nodes. The tree is fitted to the screen with the fitToScreen() method

M2:

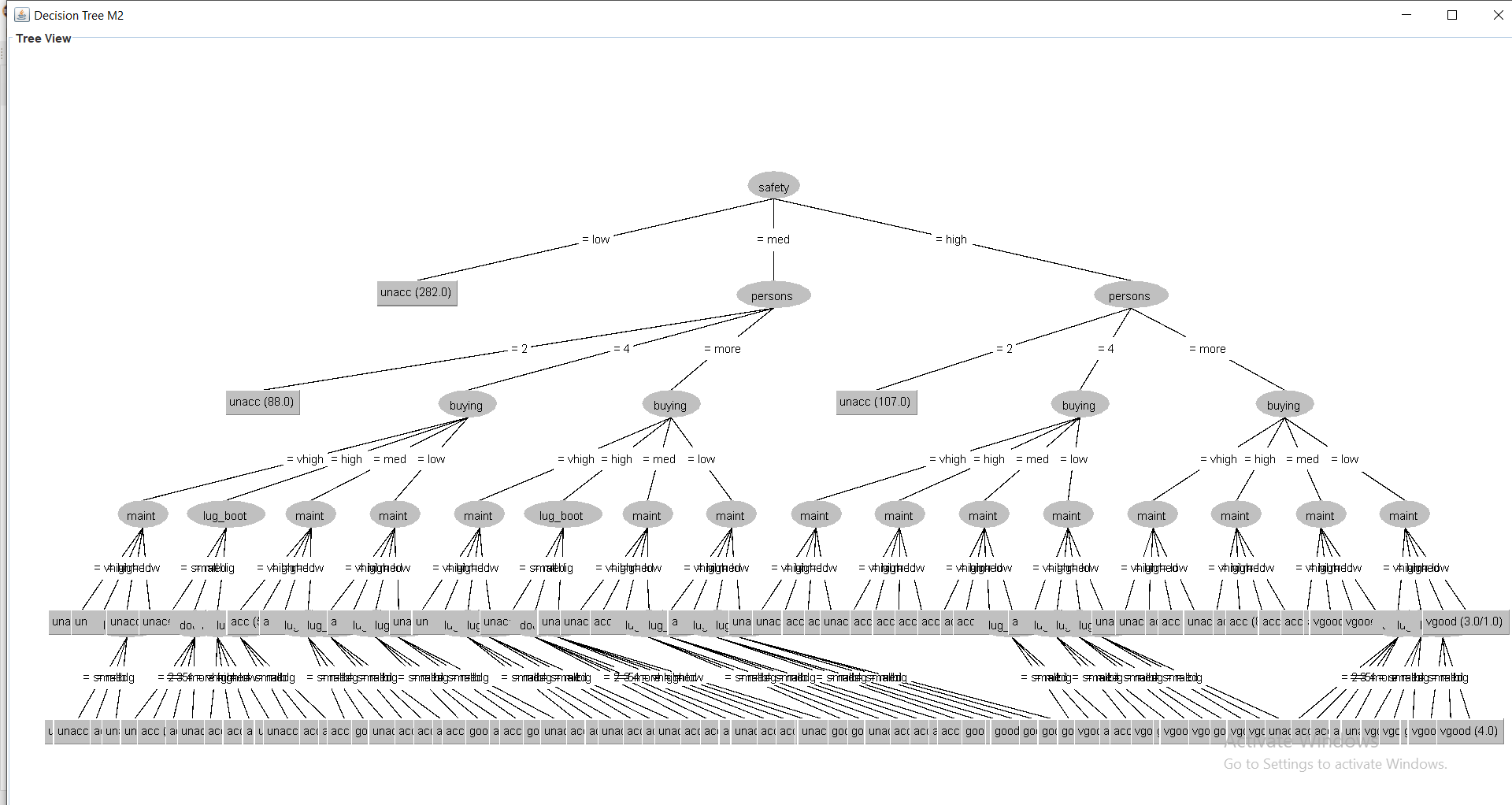


Figure 4 :Plot mode 2

M1:

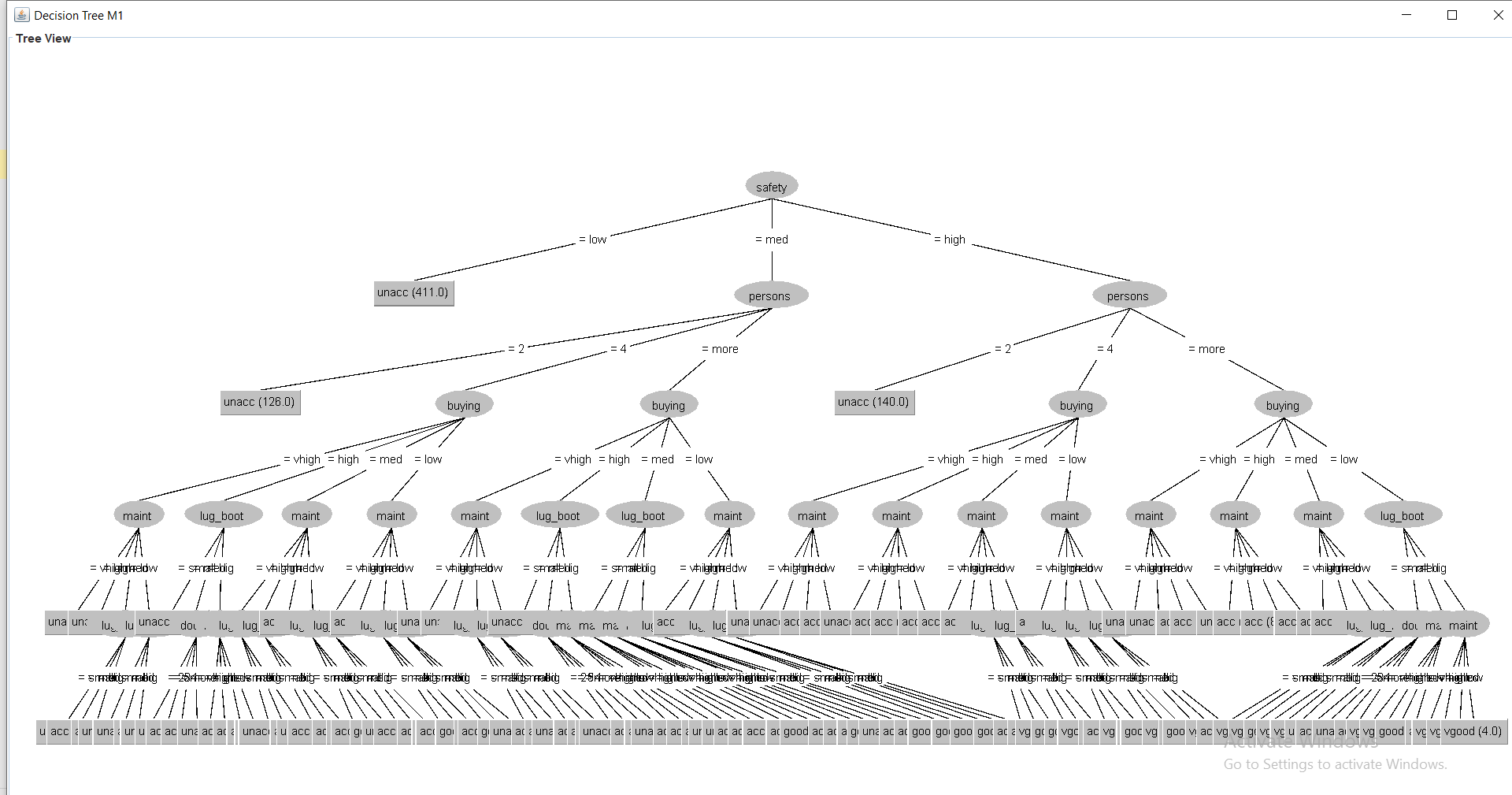


Figure 5 :Plot mode 1

**The result:**

The differences observed in the results between the two models, M1 and M2, are primarily due to the varying proportions of training and testing data.

For **Model M1 (70-30 Split)**, 70% of the dataset was used for training, while 30% was reserved for testing. This resulted in an accuracy of 92.28% and a weighted F1-score of 0.92. The larger training dataset (70%) likely enabled the model to capture more patterns in the data, leading to improved performance across both metrics.

In contrast, **Model M2 (50-50 Split)** utilized an even split of 50% for training and 50% for testing. This led to slightly lower accuracy (90.86%) and F1-score (0.91). With fewer training instances, the model had less data to learn from, which can affect its ability to generalize well to the test data.

To sum up, the larger training set in **Model M1** contributed to its better performance, while the smaller training set in **Model M2** resulted in slightly lower performance metrics.