Imperial College London

Imperial College London

Department of Computing

Introduction to Machine Learning Coursework 1 Decision Trees Report

Authors:

- Summit Bajaj, 02538746
- Marcus Hooi, 02542289
- Ho Han Yu, 02538866
- S Jivaganesh, 02541413

Date: 3 November 2023

Table of Contents

1. Method of Implementation

1.1 Decision Tree Algorithm

With the datasets provided, we implemented a decision tree through 6 main functions. These functions work by splitting the datasets recursively into more left and right datasets while maximising information gain and reducing entropy. The following briefly explains the steps we took:

- Firstly, we initialised the tree to be empty, and the maximum depth of the tree which is provided as a parameter in the constructor of the Decision Tree class
- After which, we take the input matrix and maximum depth parameter to fit the decision tree. We do so by recursively constructing the decision tree by finding the best attribute to split the data which maximises information gain. This continues until it reaches the specified max depth or if all the labels in the leaf node are the same. This method is crucial for determining how to divide the data into two subsets at each internal node of the decision tree.
- Finally, we make predictions for new data points using the constructed decision tree by recursively traversing the decision tree from the root to the leaf node, returning a predicted class label.

1.2 10-fold Cross Validation

Now that the decision trees have been constructed, we validate them through a general K-fold algorithm.

- Firstly, we split the dataset into K = 10 equal-sized folds, with each fold containing the indexes of the train, validation, test data from the input matrix, where the training data gets 80% of data in each fold while the validation and testing data gets 10% of the fold each.
- Next, for each of these 10 equal-sized folds, we create a decision tree and fit it with the training data. The tree's predictions on the validation data are used to calculate the validation accuracy, for each maximum depth ranging from 1 to 20.
- After looping through all maximum depths, the function will then select the decision tree with the highest validation accuracy as the best model for each fold.
- We then obtain the accuracy scores, confusion matrix and decision trees for each fold.
- Lastly, we take the average of the confusion matrix and calculate the accuracy, precision, recall and F1 score of the decision tree algorithm.

2. Visualisation of Tree

2.1 Visualisation Function Methodology

In order to create a visual representation of the tree, our function traverses the tree from the root node to leaf node. Whilst traversing, it creates a plot.

For each node in the tree, the function determines whether it's a **leaf node** (representing a prediction) or **an internal node** (representing a split on a feature). If it's a leaf node, the function displays the prediction room label. If it's an internal node, it shows information about the feature on which the split occurs and the split value.

In addition, to prevent overlapping of the nodes, our function calculates an offset to ensure they can fit within a row.

2.2 Function Structure

Code	Purpose

`#Function counts the number of children nodes from a parent node`
`def countChildren(tree):`
`    if tree['isLeaf']:`
`        return 1`
`    else:`
`        left_tree = tree['left']`
`        right_tree = tree['right']`
`        return countChildren(left_tree) + countChildren(right_tree)`
	Count the number of children nodes in a decision tree.

`#Function uses recursion to plot the nodes in the decision tree and links them with arrows`
`def plotDecisionTree(tree, x_cord, y_cord, offset):`
`    #If leaf node is reach`
`    if tree['isLeaf']:`
`        #Print the predicted room`
`        node_label = f"Predict:\n Room{int(tree['max_attribute']+1)}"`
`        plt.text(x_cord, y_cord, node_label, ha='center', va='center', fontsize=20)`
`        return 1`
 	
`    #Else if node is a child node`
`    else:`
`        left_tree = tree['left']`
`        right_tree = tree['right']`
`        #Print the feature the node will be split on and the split value`
`        node_label = f"Split on: Room{ tree['attribute']+1}\nSplit by: {tree['split']}"`
`        plt.text(x_cord, y_cord, node_label, ha='center', va='center', fontsize=20)`
 	
`        #Creates an offset so that all the children nodes can fit in a single row`
`        left_offset = right_offset = offset / 2.5`
 	
`        #Coordinates of the elft and right children node`
`        left_child_x = (x_cord - left_offset)`
`        left_child_y = (y_cord - 0.5)`
 	
`        right_child_x = (x_cord + right_offset)`
`        right_child_y = (y_cord - 0.5)`
 	
`        #Plot arrows from the parent node to the left and right child node`
`        plt.arrow(x_cord, y_cord-0.03, left_child_x - x_cord, left_child_y - y_cord+0.08,`
`         length_includes_head=True, width=0.005)`
`        plt.arrow(x_cord, y_cord-0.03, right_child_x - x_cord, right_child_y - y_cord+0.08,`
`         length_includes_head=True, width=0.005)`
 	
`        #Recursively calls the plotDecisionTree function to plot the decision tree`
`        plotDecisionTree(left_tree, x_cord - left_offset, y_cord - 0.5, left_offset)`
`        plotDecisionTree(right_tree, x_cord + right_offset, y_cord - 0.5, right_offset)`
	Build the tree using recursion

Code	Purpose

`#Call function to plot the final decision tree (Calls countChildren & plotDecisionTree functions)`
`#Function takes in the decision tree and max_horizontal_len`
`#(which is the length of the final depth of the tree,`
`# length must be selected such that all the final depth nodes will not overlap with each other)`
`def visualise_tree(tree, max_horizontal_len, tree_name):`
`  #Calculates the initial offset to be used in the plotDecisionTree function`
`  num_of_nodes = countChildren(tree.tree)`
`  initial_span = max_horizontal_len / (num_of_nodes - 1)`
`  plt.figure(figsize=(max_horizontal_len, max_horizontal_len))`
`  plotDecisionTree(tree.tree, 0.5, 2, initial_span)`
`  plt.axis('off')`
`  # Save the plot as a PNG file`
`  plt.savefig(tree_name, dpi=500)`
`  plt.show()`
	Function to visualise tree using matplotlib

`#The tree to be printed is the the one that yielded the highest accuracy`
`#score in the cross validation done previously`
 	
`#Call the function visualise_tree to visualise the clean tree`
`visualise_tree(clean_trees[4], 40, "clean_decision_tree")`
`#Call the function visualise_tree to visualise the noisy tree`
`visualise_tree(noisy_trees[0], 40, "noisy_decision_tree")`
	Call the visualise_tree function to build the clean and noisy tree. The

Table 1. Code Overview of Tree Visualisation Function

2.4 Visualisation of Tree With Clean Dataset

Using 10-fold cross validation, a total of 10 trees were plotted with the clean dataset and another 10 trees were plotted with the noisy dataset.

Of the 10 decision trees for the clean dataset, the tree that yielded the highest accuracy and lowest max depth was selected and plotted below in Figure 1.

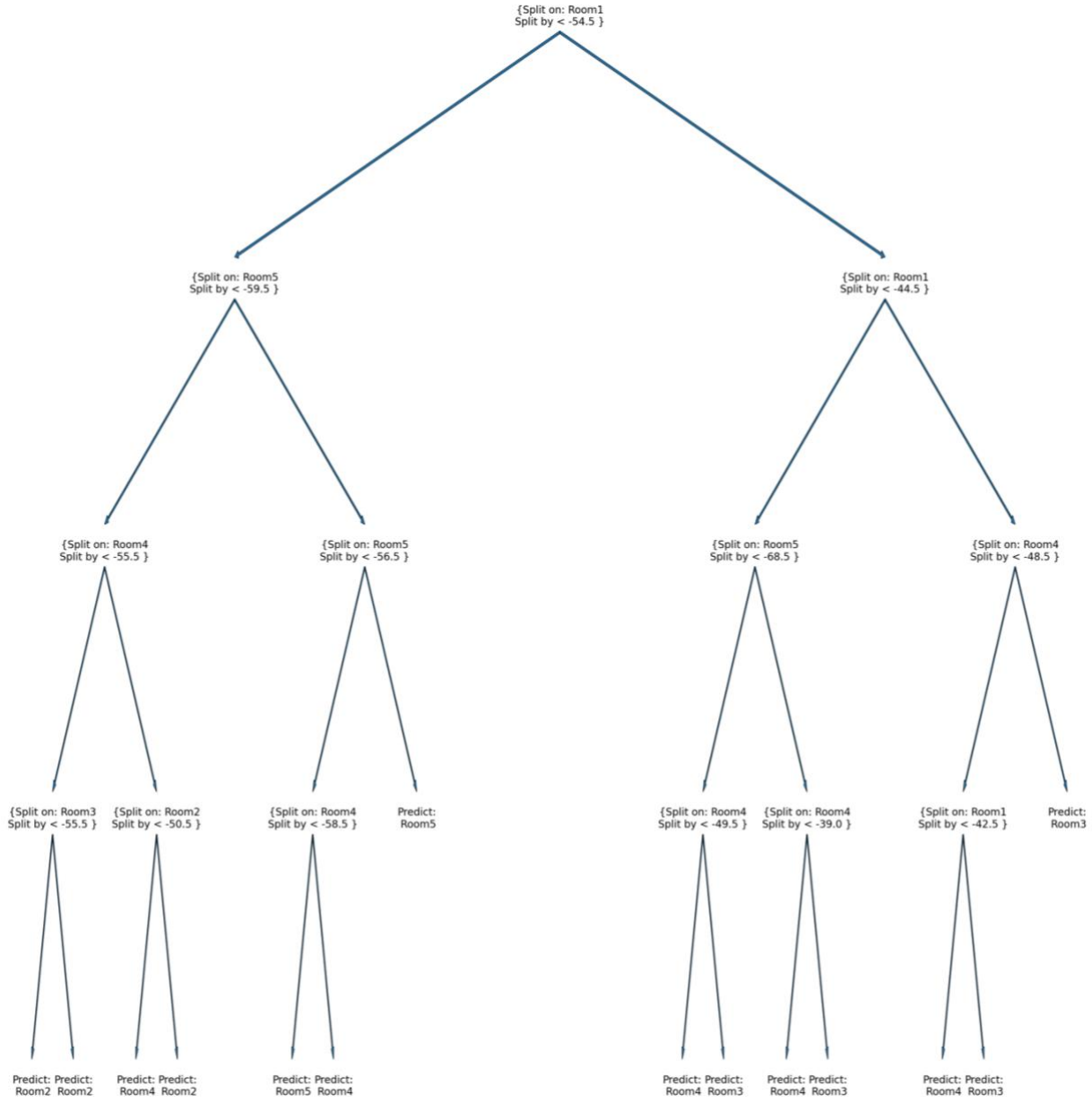

Figure 1: Visualisation of Decision Tree with Clean Dataset

2.5 Visualisation of Tree With Noisy Dataset

Similarly, of the 10 decision trees for the noisy dataset, the tree that yielded the highest accuracy and lowest max depth was selected and plotted below in Figure 2. Due to space constrain, the image in Figure 2 is limited to depth 4.

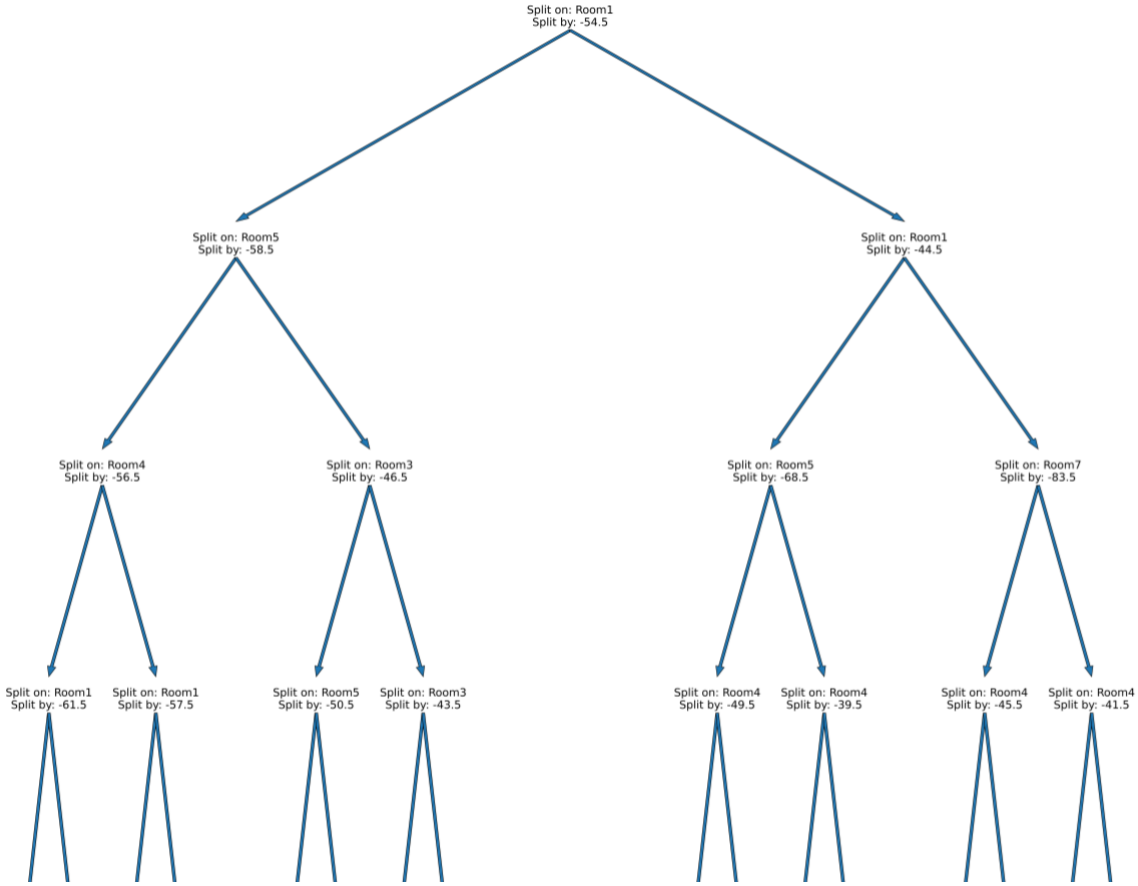

Figure 2: Visualisation of Decision Tree with Noisy Dataset

3. Results and Evaluation

3.1 Clean Dataset Results

The average confusion matrix of the clean dataset is shown in Table. 2, while the accuracy, recall, precision, and F1 scores are shown in Table. 3.

3.1.1 Average Confusion Matrix

Room	1 Predicted	2 Predicted	3 Predicted	4 Predicted
1 Actual	49.8	0.0	0.2	0.4
2 Actual	0.0	47.3	2.7	0.0
3 Actual	0.4	1.5	47.9	0.2
4 Actual	0.5	0.0	0.4	49.1

Table 2. Average Confusion Matrix of Clean Dataset (rounded to 1d.p.)

3.1.2 Accuracy, Recall and Precision Rate, F1 measure per class

Evaluation Metric	Room 1	Room 2	Room 3	Room 4
Accuracy	0.970	0.970	0.970	0.970
Recall	0.996	0.946	0.958	0.982
Precision Rate	0.982	0.969	0.936	0.996
F1-score	0.989	0.957	0.947	0.989

Table 3. Average Confusion Matrix of Clean Dataset (rounded to 3d.p.)

3.2 Noisy Dataset Results

The average confusion matrix of the noisy dataset is shown in Table. 4, while the accuracy, recall, precision, and F1 scores are shown in Table. 5.

3.2.1 Average Confusion Matrix

Room	1 Predicted	2 Predicted	3 Predicted	4 Predicted
1 Actual	44.6	1.0	1.4	2.0
2 Actual	1.8	43.7	3.1	1.1
3 Actual	2.0	2.4	45.4	1.7
4 Actual	2.2	1.3	1.6	44.7

Table 4. Average Confusion Matrix of Noisy Dataset (rounded to 1d.p.)

3.2.2 Accuracy, Recall and Precision Rate, F1 measure per class

Evaluation Metric	Room 1	Room 2	Room 3	Room 4
Accuracy	0.892	0.892	0.892	0.892
Recall	0.910	0.879	0.882	0.896
Precision Rate	0.881	0.903	0.882	0.903
F1-score	0.896	0.891	0.882	0.900

Table 5. Average Confusion Matrix of Noisy Dataset (rounded to 3d.p.)

4. Analysis of Results

4.1 Result Analysis

Generally for both clean and noisy dataset, the rooms are recognised accurately. This is supported by the high true positive count, as seen in the confusion matrix, and F1 Score.

However, occasional misclassifications in the confusion matrix of the clean dataset Table 2 and Table 4 is observed:

- Room 1 gets confused with Room 3 and Room 4
- Room 2 gets confused with Room 3
- Room 3 gets confused with Room 1, Room 2 and Room 4
- Room 4 gets confused with Room 1 and Room 3

4.2 Dataset Difference

Looking at the F1 score, the clean dataset has a higher score than that of a noisy dataset. F1 score was used to balance the results of precision and recall. This result indicates that the clean dataset is more likely to correctly classify a room and less likely to make misclassifications. This might be because a decision tree might tend to overfit values on a noisy data, leading to inaccuracies and suboptimal split.