Introduction to Machine Learning Coursework 1: Decision Trees

Reni, Ishaan (ir320)
Khan, Haaris (hhk20)
Neagu, Alexandra (an720)
Louie, Charmaine (ccl19)

Contents

Introduction

The aim of the coursework is to produce a machine learning model which can determine a device's location depending on the strength of WIFI signals detected by the device. To create this model, a decision tree was used. This was a multilabel classification problem, as the task included 4 different rooms. To create a decision tree, the splits were determined using entropy calculations. Since the dataset was relatively small, cross validation was applied.

Our Approach

The program uses an object-oriented class structure to improve extensibility and modularity. The trees and nodes are instances of their respective classes, which helped especially in cross-validation.

The data was split using the *find_split()* function. For each router column, the rows are split to form two groups, an upper and lower group. As the data is sorted, the upper group contains larger values than the lower group and each iteration choses a new value to split the data by that is smaller than the previous value that was used. For each group, the entropy is calculated and weighted appropriately. Summing the weighted entropy of the upper and lower group results in an overall entropy. The lowest possible overall entropy is recorded for each router column and then the lowest possible entropy out of all the router columns is chosen along with the split value that resulted in the best entropy. Both the router and the split value form the decision node.

The trees are pruned using the validation set, using post-order (bottom-up) traversal. Post-order traversal is often used to recursively delete a tree. For this reason, only one traversal through the tree is necessary to check every node for pruning. Nodes are pruned by turning them into leaves which take the value of the majority class from the training dataset. If the accuracy decreases, that operation is undone. This continues until every node is checked.

In this scenario there are four class labels, meaning evaluation metrics can be derived using 4×4 confusion matrix. Using the matrix, accuracy can be calculated (proportion of all correctly identified labels). Precision (proportion of true positives among all predictions for that class) and recall (proportion of that class that were predicted as positive) can also be calculated. F1 measure is also used, which combines precision and recall. Since the dataset is perfectly balanced, and no specific room is deemed to be the 'true' class, we expect the precision, recall, and F1 to be similar for each room, and for each room to have similar precisions, recalls and F1s (symmetrical matrix transpose).

Visualising the Decision Trees

To be able to get a general understanding of the decision tree models based on the given datasets, we set up a visualising script. The script uses a pre-order depth first traversal algorithm, that creates the edges separately from the nodes and leaves. By making this division, we can gather more information regarding the tree and present it in the legend table in the top left corner.

Matplotlib allows the user to interact with the graph by zooming or panning, hence being able to obtain every detail of the tree.

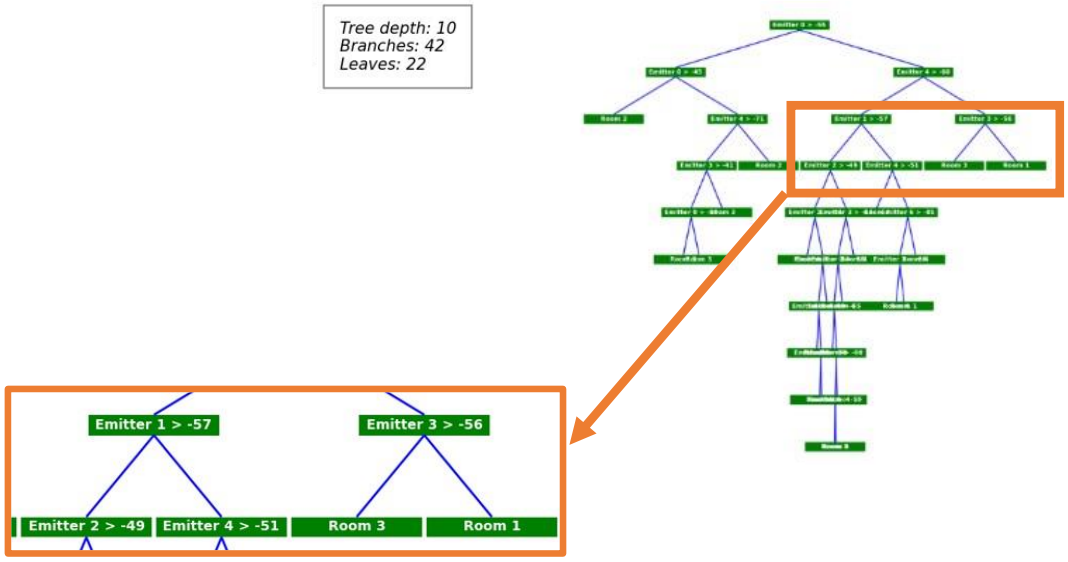

Figure 1: Tree printed using Matplotlib

The Figure 1 presents a decision tree and a zoomed in section for easier comprehension of the lower levels of the tree. More diagrams presenting some examples of pre-pruned and pruned decision trees can be found in the Appendix.

Implementing Cross-Validation

To generally evaluate our algorithm for generating decision tree models, a separate script is provided. It completes a ten-fold nested cross-validation onto any of the datasets the user provides. The dataset is initially randomly shuffled, to avoid any ordering of the data provided. Then the data (by default) is divided into 80% / 10% / 10% splits, which are the training, validation, and test datasets respectively.

Using the proportions, all possible permutations of the three datasets are formed. Hence, for every given test dataset, nine inner folds are computed by shifting the validation subset around the training subset.

This step gets repeated until the test and validation datasets have been in all possible positions. This process is seen in Figure 2.

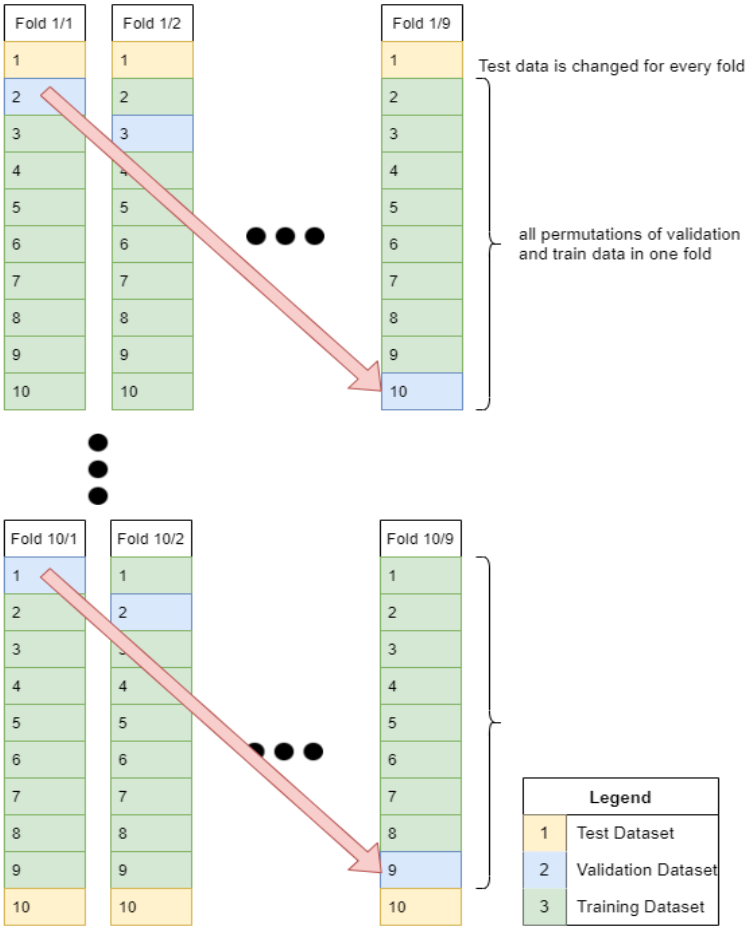

Figure 2: Division of data during cross-validation

For each of the 90 models the script generates, the algorithm calculates all specific values of the evaluation metrics and averages them together at the end to provide an overall view of how the general model performs.

Comparisons

After running cross validation, the averages of accuracy, precision, recall and F1 were obtained for both pre-pruned and pruned trees for each dataset. The average accuracies after cross validation are shown in the table below:

	Pre-pruned tree	**Pruned tree**
Noisy Dataset	0.8028	0.8801
Clean Dataset	0.9738	0.9687

Table 1: Comparison between accuracies

As shown in Table 1, the accuracy for the noisy dataset improved after pruning. This is because while training, the trees learnt the present noise in the training data, resulting in overfitting. Using validation data to prune the trees shorter allows them to generalise to effective patterns shared between both sets (as noise between sets is different). However, the accuracy decreased for the clean dataset after pruning, due to noise not being present in the training data. This produces a tree which is mildly biased due to low node count.

To calculate the other evaluation metrics, we need to evaluate the confusion matrix for pre-pruned and pruned models on all datasets. Table 2 shows example of the confusion matrices.

	Pre-pruned tree	**Pruned tree**
Noisy Dataset	39.04 2.93 3.12 3.90 2.71 40.22 4.20 2.57 3.10 3.63 41.21 3.56 4.22 2.47 3.02 40.09	44.09 1.22 1.33 2.24 1.98 43.73 2.71 1.27 2.16 3.09 44.36 1.90 2.62 1.44 1.89 43.84
Clean Dataset	49.44 0.00 0.23 0.32 0.00 47.96 2.04 0.00 0.12 1.54 48.06 0.28 0.47 0.00 0.21 49.32	49.7 0.00 0.26 0.04 0.00 47.68 2.31 0.00 0.42 2.06 47.16 0.36 0.53 0.00 0.27 49.2

Table 2: Comparison of confusion matrices

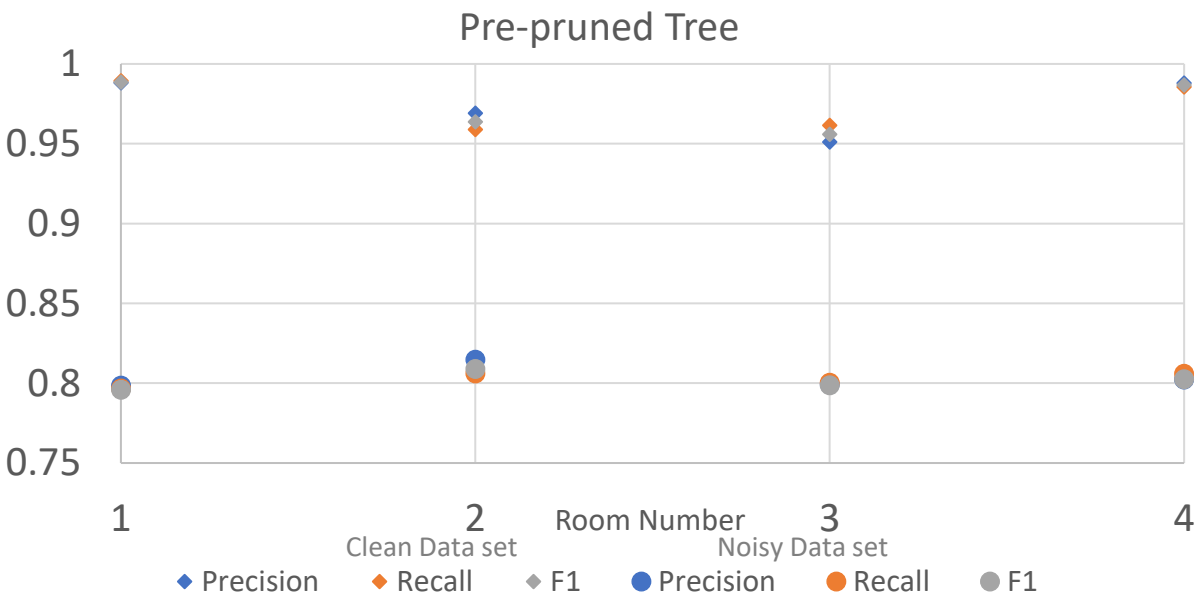

Figure 3: Average metrics for the pre-pruned trees

Next the average precision, recall and F1 values were calculated for both pre-pruned and pruned tree after cross validation. For pre-pruned trees, the average metrics are shown in Figure 3. Within a range of random fluctuation, all rooms have similar precisions and recalls. Room 3 marginally underperforms compared to the other rooms; however, it is still not highly confused. Theoretically, since the dataset is balanced, the precisions and recalls should be equal. However, small discrepancies arose due to the way the datasets were split. In reality, the model is less susceptible to the precision and recall trade-off as there were no rooms that were preferred over the other. The high F1 score for both datasets shows that both precision and recall are high.

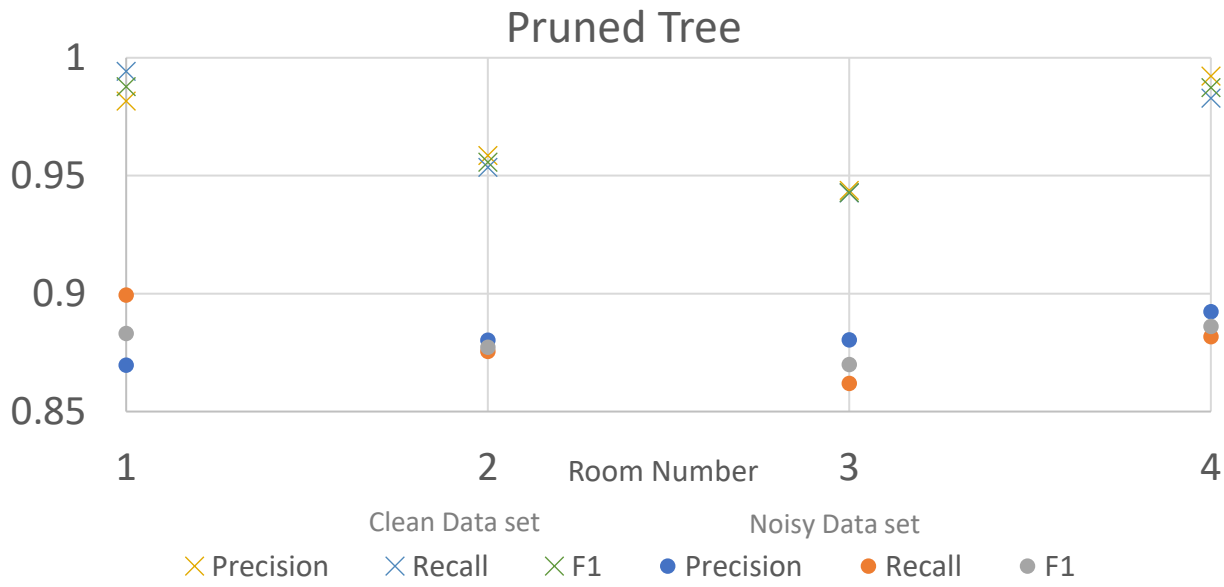

Figure 4: Average metrics for the pruned trees

Figure 4 represents the average metrics after pruning the trees. The number of nodes and leaves will decrease; the precision and recall can improve. Since the noisy dataset contains more noise and meaningless data, it is more difficult to achieve a higher precision and recall

compared to the clean dataset. The difference of precision and recall between the clean and noisy datasets can be seen by the large gap between the scatter points in both Figure 3 and Figure 4.

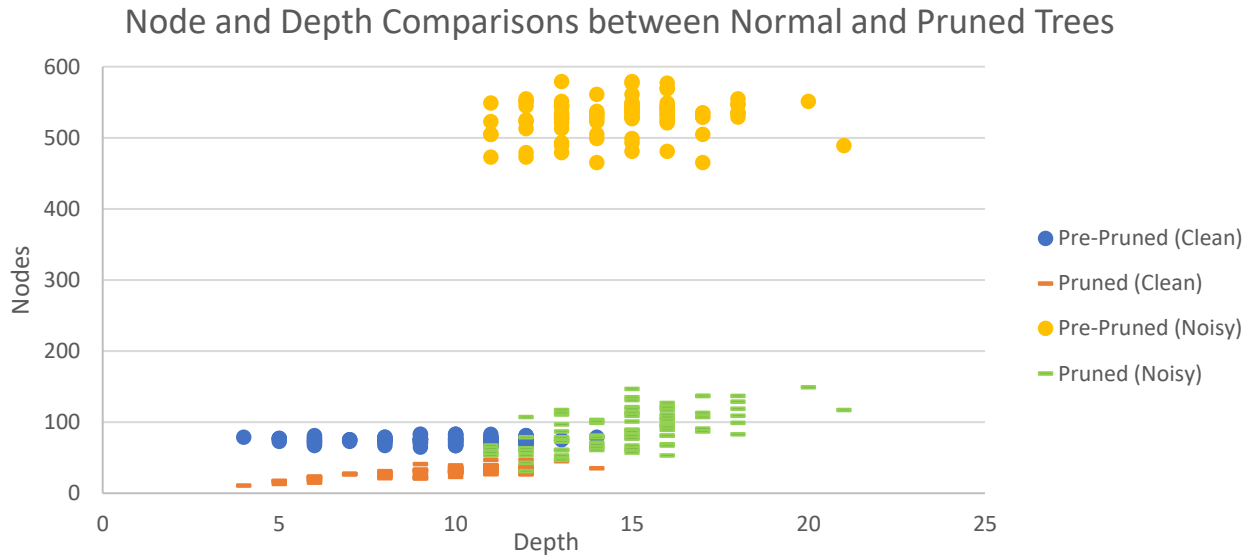

Figure 5: Comparison between average number of nodes against depth

Figure 5 has four different clusters that represent the number of nodes against the number of depth present on each tree fold. The blue and yellow dots represent the normal (unpruned) trees that are formed initially whereas the orange and green dashes represent the pruned trees.

The clean tree originally starts as a close cluster (blue dots) which is expected as the algorithm will need to make less decisions to sort the different points in the dataset. After pruning, the size of the tree decreases, but not by much, as there is a lower bound to the amount of decision nodes required to retain accuracy in the validation set.

Figure 5 also shows that the unpruned noisy tree is larger in comparison to the other trees as it tries to fit to the training data as best as it can which results in a deep tree with a large amount of decision nodes and leaves. After pruning, the points move closer to the origin as the number of nodes and leaves decreases significantly as pruning helps increase the accuracy while reducing the size of the tree.

	Pre-pruned tree	**Pruned tree**	**Percentage Change**
Noisy Dataset	529.6	88.64	-83.26%
Clean Dataset	74.47	29.36	-60.57%

Table 3: Comparison between the average number of nodes3

As expected, Table 3 shows that the percentage change seen on the noisy dataset is much larger than the one on the clean dataset due to the large number of nodes that need to be pruned on the noisy trees. The clean dataset, however, has a larger percentage change than that was anticipated as pruning should have only marginally reduced the tree, since noise wasn't present in the training data, and the validation data would have shared similar patterns to the training data.

Appendix

Tree depth: 10
Branches: 66
Leaves: 34

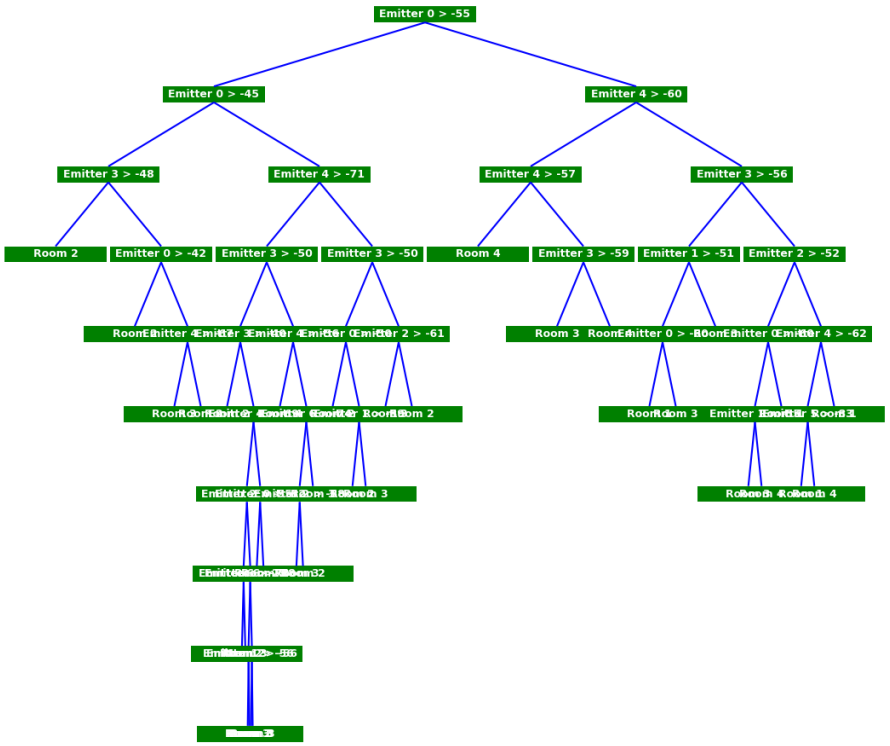

Figure 6: Example of clean dataset pre-pruned tree

Tree depth: 9
Branches: 20
Leaves: 11

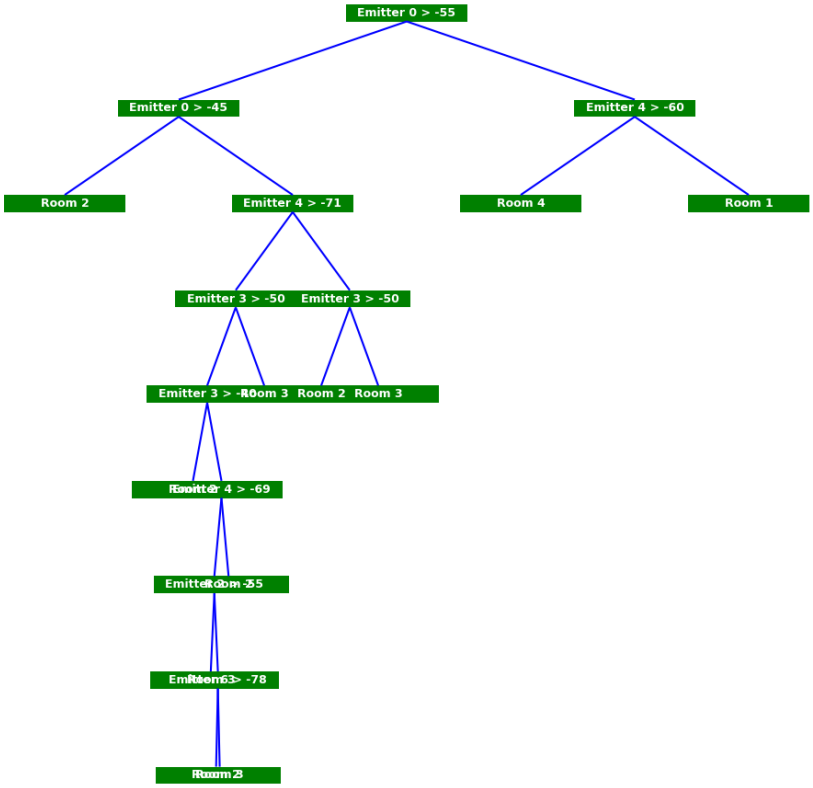

Figure 7: Example of clean dataset pruned tree

Tree depth: 10
Branches: 236
Leaves: 119

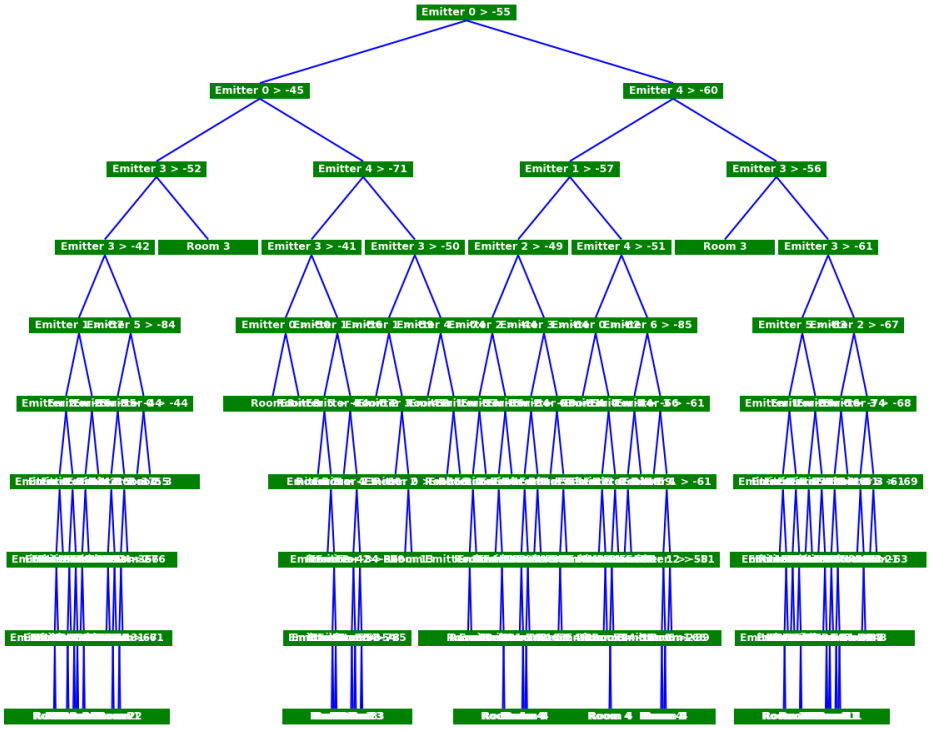

Figure 8: Example of noisy dataset pre-pruned tree

Tree depth: 10
Branches: 42
Leaves: 22

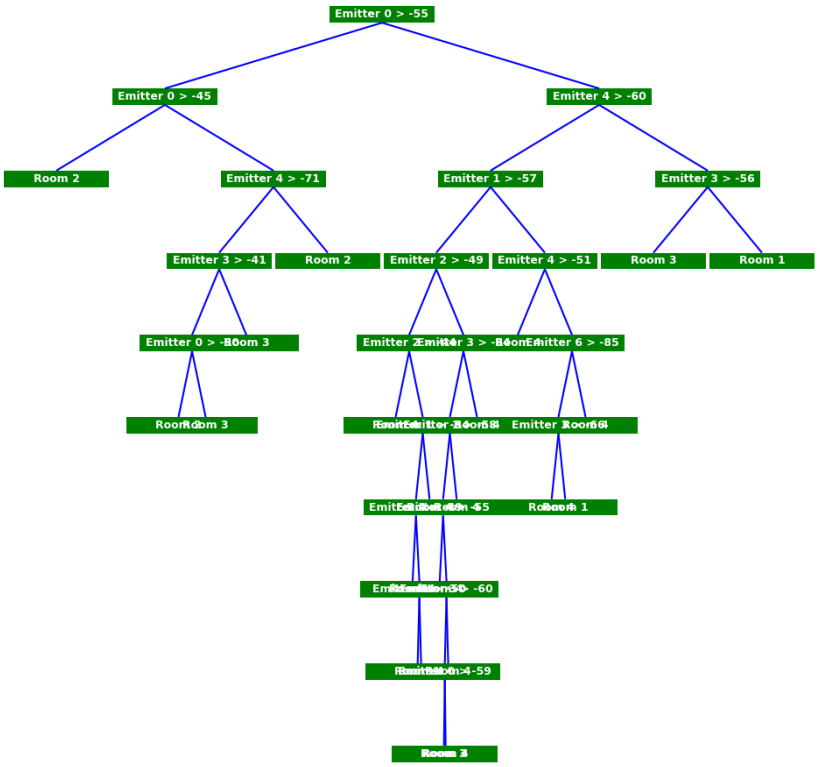

Figure 9: Example of noisy dataset pruned tree