COMP97101(CO395) Machine Learning
The Imperial College of Science, Technology and Medicine

Coursework 1

Ludwig Jonsson CID: 01520034
Daryl Lim CID: 01336845
Raymond Zhu CID: 01559652
Kazuya Kai-Olowu CID: 01498848

1-11-2020

Contents

1 Implementation Details

1.1 Building the Decision Tree

The decision tree was implemented by creating a Tree class object containing Node class objects. Each node contains the variables: 'room', 'value', 'attribute', 'left', 'right', and 'size'. The tree contained the root: Node of the tree. The decision tree is build through recursive calls of the decisionTreeLearning() function which creates new nodes for the 'left' and 'right' branches. The split attribute and value used to create the new subsets of the data set is found using the findSplit() function (See Appendix, figure 10), which greedily finds the best split value by calculating the information gain for each split on every attribute, then finding which split gave the highest information gain and returning corresponding row and attribute.

When splitting the set in decisionTreeLearning, we sorted the attribute (ascending) returned from findSplit() and then split it on the row also returned by findSplit(). The value of this Node with sub-nodes containing the split dataset has its attribute set to the one used before to split and also the value of that attribute's row we split on.
The recursive termination condition is the case that a leaf node is created, which occurs in the case that all the labels in the current subset of the data set are the same, thus there is no more information gain possible from splitting. In this instance, the node is assign the label in the 'room' field, whereas in all other cases this field is empty (containing None).

1.2 Making Predictions

To make a prediction, the tree is traversed starting at the root node, stored in the 'root' variable in the tree class. At each node, the node's split value and attribute is used to determine whether to go to the left or right node; if the attribute's value of the sample being predicted is less than the split value, the left node is entered, otherwise the right node is entered. This repeats until a node with a 'room' label value that is not None is reached, and that 'room' label it self is the prediction.

1.3 K-Fold Cross Validation

The evaluation of a decision tree was done by performing 10-fold cross validation on the data set. Initially, the data set is shuffled; then for every iteration in the 10-fold cross validation process, a subsequent equally sized partition of the data set is used as the test set. Additionally, for each specific iteration, a subsequent equally sized partition of the data set, not equal to that used in the test set, is used for the validation set for each fold. The training set is then simply the remaining sample unused in both the test set and validation set, which is used to then build the decision tree. With the decision tree, a confusion matrix can be constructed, which is then used to calculate its average recall, precision, F measure, and classification rates.

1.4 Pruning

The pruning of the decision tree was done after each decision tree is constructed. The pruning process involved a post order traversal of the decision tree, where in the case that a node is connected to two leaf nodes (leaf nodes are nodes with a 'room' value not equal to None), the node takes the 'room' value from the child node with the larger size (number of unique labels); the pruning stands if this results in a higher information gain, otherwise it remains untouched.
If a new leaf Node is created, its size is set to be the same as the left Node it took its room value from.
See Appendix, figure 9, for more information.

2 Evaluation Metrics

2.1 Confusion Matrix

The confusion matrix stores the all the classifications made by the decision tree model. The rows in the matrix indicate what label (room number) our model predicts while the columns indicate the true label of the sample.

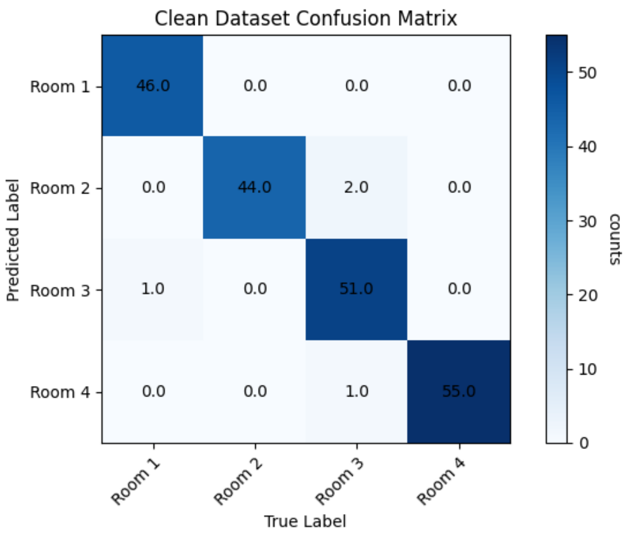


Figure 1: Confusion Matrix generated from evaluating the clean dataset

In the confusion matrix for the clean data set (Figure 1), we can see how well our model performs when at classifying each room. The diagonal terms in the matrix are the correctly classified samples, while all counts outside the diagonal are falsely classified. For a perfectly performing model with 100% accuracy, we will expect the confusion matrix to be a diagonal matrix.
For instance, samples taken from room 1 are correctly classified 46 out of 47 times (97.87%), with type II errors (false negative) only occurring once out of the 47 samples (2.13%) from the test set.

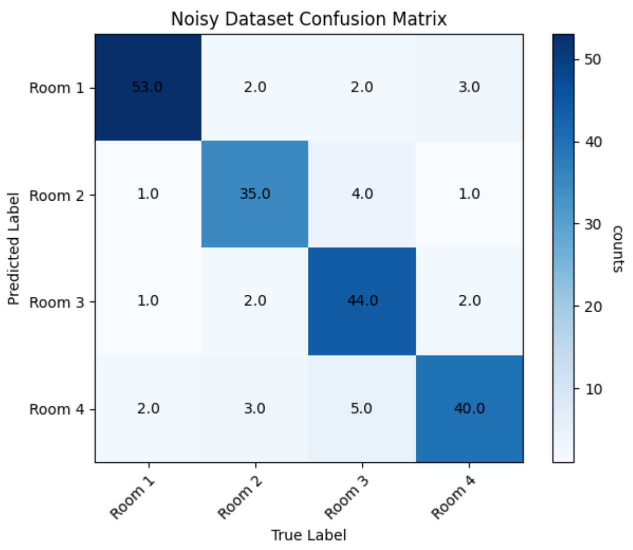


Figure 2: Confusion Matrix generated from evaluating the noisy dataset

As for the noisy data set (Figure 2), we can clearly see that the model is performing worse than it did on the clean data set as there are more counts outside the diagonal in the confusion matrix.

There are multiple ways of quantifying the performance of a model using the confusion matrix, as it will be discussed in the next section.

2.2 Recall/Precision/F-measure/Classification rate

Recall is the percentage of correct positively predicted samples in all of the positively classified samples for a given label. In layman's terms, we can express this as out of all the samples which our model predicts is taken from Room X, what percentage of which is actually from Room X? This is equivalent to the diagonal term as a percentage of the sum of the entire row in the confusion matrix.

Since there are 4 rooms, the recall of the model will be expressed as a 4x1 matrix. Below is the recall of the model, calculated using confusion matrix generated from evaluating the **clean** data set.

$$\begin{bmatrix} \frac{46}{46} & \frac{44}{44+2} & \frac{51}{51+1} & \frac{55}{55+1} \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 0.960 & 0.980 & 0.982 \end{bmatrix}$$

Precision is also an evaluation metric that can be used to measure the performance of a model. The difference, however, is that precision measures the percentage of correct positively classified samples from the true positive samples. In simpler wording, it shows the percentage of all Room X samples in the test set that our model correctly predicts is coming from Room X. This is equivalent to the diagonal term as a percentage of the sum of the entire column in the confusion matrix.

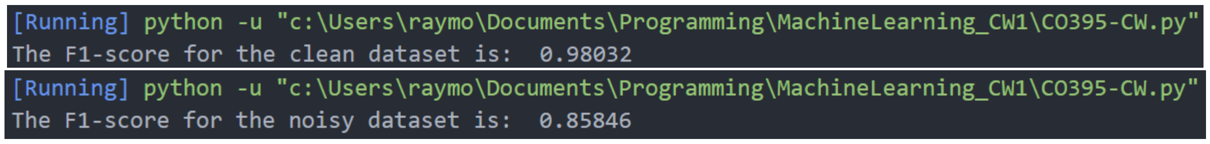
The precision will also be structured in a 4x1 matrix, as the precision for each room is different. Below is the precision of the model when performing on the **noisy** data set.

$$\begin{bmatrix} \frac{53}{53+1+1+2} & \frac{35}{2+35+2+3} & \frac{44}{2+4+44+5} & \frac{40}{3+1+2+40} \end{bmatrix} \longrightarrow \begin{bmatrix} 0.930 & 0.833 & 0.800 & 0.869 \end{bmatrix}$$

The F-measure or F1-score is another metric that is used to evaluate the performance of the model. The metric is a function of both the recall and the precision from above, and is finds a balanced representation of the two. The equation of the F1-score is as below.

$$F_1 = 2 \cdot \frac{\textbf{precision} \cdot \textbf{recall}}{\textbf{precision} + \textbf{recall}}$$

To evaluate the average F1, the precision and recall values used will be the mean of the 4 elements in each matrix. The detailed calculation will not be shown here but below are the screenshots for the F1-score outputted by the code.



```
[Running] python -u "c:\Users\raymo\Documents\Programming\MachineLearning_CW1\CO395-CW.py"
The F1-score for the clean dataset is:  0.98032
[Running] python -u "c:\Users\raymo\Documents\Programming\MachineLearning_CW1\CO395-CW.py"
The F1-score for the noisy dataset is:  0.85846
```

Figure 3: F1 scores for noisy and clean dataset

Finally, the classification rate is the percentage of correct predictions from the test set (200 samples). For our model, classification rate for the clean data set was **98%** and for the noisy data was **86%**. These values are close to the F1-Scores, which is a sign that our data are balanced in terms of samples from each class.

2.3 Analysis of the Cross Validation Evaluation

Cross validation evaluation is a technique used to find the best possible decision tree using different partitions (folds) within the data set. The fold number chosen was 10, which means that the data set will be divided into 10 fold after being initially randomly shuffled. The data set is then processed through two nested loops, with the outer one choosing which fold to be used as the test set, and the inner loop deciding the fold used for validation set. The remaining 8 folds (1600 samples) will be used to train a tree in each iteration of the inner loop, and that tree will be evaluated with the validation set, recording the performance in terms of the classification rate.

For each iteration in the outer loop, the best performing tree will be evaluated with the test set, and the performance is also stored. Finally, the output confusion matrix will be an average of the confusion matrices stored from the outer loops.

The benefit of this evaluation technique is that it uses 100% of the data set for both training and testing, but at different times. Hence it can be said that no data was wasted, while ensuring that the model was never evaluated with seen data. Additionally, rotating the test set around and using an average of the output metrics will reduce the variance of the performance compared to the output from a model where 10% of the data is purely used for the test set.

3 Pruning

3.1 Algorithm and Implementation

The pruning function was implemented as a method for the decision tree class, which was called inside the cross validation function right after generating the tree using the training set. The method calls an inner function that performs a postorder binary tree traversal to ensure that each child node is visited before processing the parent node. This allows the function to prune in a bottom-up fashion in a single recursive traversal. The pseudocode for the postorder can be found in Figure 10 in the Appendix.

The general idea is to find a node that satisfies the condition to be pruned, which is when both left and right children are leaf nodes (nodes with non-null *room* attribute). This enables the function to modify in-place the current node's attributes to one of the children nodes' attributes to become a leaf node, allowing for simple reversion of the pruning. The current node then takes on the attributes of the child that has the larger proportion of samples, i.e. larger *size* attribute. The current node's *size* attribute is not updated to size to the sum of left and right children *size* attributes because the minority samples in fact negatively influence the accuracy of the pruned tree. The confusion matrices of both unpruned and pruned versions of the tree are obtained by a function call to evaluate using the validation set, from which the accuracies can be derived. The accuracies are then compared, and if the unpruned tree has a higher accuracy than the pruned tree then the pruning is reverted, otherwise the both children are set to null to fully prune the node. Due to the postorder nature of the traversal, these changes are propagated upwards throughout the tree.

3.2 Evaluation of Pruning on Accuracy

To evaluate the effect of pruning on accuracy, 10-fold cross validation was performed with and without pruning for both clean and noisy data sets. The confusion matrices for each resultant tree were then used to calculate obtain the accuracy for each combination of clean-unpruned, clean-pruned, noisy-unpruned, and noisy-pruned. Let P be a boolean variable that represents whether pruning was performed during the cross validation.

Clean & !P	Clean & P	Noisy & !P	Noisy & P
0.975	0.980	0.830	0.860

Table 1: Experimental accuracies for each P and data set combination

The experimental data in Table 1 shows a 0.5% gain in accuracy from pruning for the clean data set, and a 3% gain in accuracy from pruning for the noisy data set. These performance increases from pruning are congruent with the notion of pruning reducing the depth of the tree, which increases the generality of the tree and prevents overfitting for the training data. This will be further discussed in section 4.

4 Comparison between Noisy and Clean Data

4.1 Difference in Performance

As we can see in Table 1, the clean dataset gave better performance in correctly classifying rooms. This could partly be attributed to the noisy dataset providing inaccurate data which will lead the the tree to taking the wrong path when traversing the tree. In essence, a *messy* value could, when creating the tree, make it so the node's value we compare against when trying to classifying a room, to be incorrect.

On average, before pruning, the model trained on the clean dataset had an accuracy of ∼17.5% greater than the same trained on the noisy dataset. After pruning, that difference went down to ∼14%.
Running the noisy dataset also took longer to run than its clean dataset counterpart. This can also partly be attributed to its larger size having to be generated.

As we can see from Tables 2 and 3, each class performed much better when the model was trained on the clean data. We can also see that room 4 had the largest difference between the different runs, and room 1 had the smallest. Room 4 was also the class, for which training on the clean dataset instead of the noisy dataset, decreased the amount of times our model falsely predicted it. Room 2 was the class which saw the highest improvement in being classified (i.e. not being a false negative).

Clean Dataset			
Room	Precision	Recall	F1
Class 1	1.0	0.979	0.989
Class 2	0.957	1.0	0.978
Class 3	0.981	0.944	0.962
Class 4	0.982	1.0	0.991

Noisy Dataset			
Room	Precision	Recall	F1
Class 1	0.883	0.93	0.906
Class 2	0.854	0.833	0.843
Class 3	0.898	0.8	0.846
Class 4	0.8	0.87	0.834

Table 2: Evalution metrics for each class trained on the two different datasets and then pruned. Note that many floating point values are rounded

Difference in %			
Room	Precision	Recall	F1
Class 1	13.3	5.3	9.2
Class 2	12.1	20	16
Class 3	9.2	18	13.7
Class 4	22.8	14.9	18.8

Table 3: Metrics for each class compared between training on clean dataset and noisy dataset

4.2 Difference in Size

There is a clear difference between the sizes of the trees created as well. Even though the clean dataset produced trees that were hard to see visualize in the terminal window. The noisy dataset produced trees that were impossible to fully fit on the screen whilst still being able to see the values of the nodes.

Please refer to the appendix for pictures of the generated trees. Please note that we did not include trees made from the noisy dataset. This was done simply because those trees could not accurately be shown without possibly a very large display.

5 Tree Depth

5.1 Maximal Depth of the Generated Trees

The maximum depth of the tree with the highest prediction accuracy was 11 for the clean data set, and 20 for the noisy data set. Pruning of the decision trees didn't necessarily affect the maximal depth for either data set, although it did for some occasions on the clean data set. Although pruning out excess leaves leads to an improvement of performance on the test set, it doesn't necessarily reduce the maximum depth of the tree as some of the leaf nodes at the maximal depth remain beneficial in maintaining the highest information gain.

5.2 Relationship Between Maximal Depth and Prediction Accuracy

As the maximum depth of the tree is increased, there is a downward trend in the classification error in the training set for both the clean and noisy datasets. Although not clearly shown in the figures 4 and 5, this trend would not be as clear for the test set if the maximum depth is set too high; this is as the decision tree may start to overfit the training set, resulting in a potential increase in the classification error on the test set.

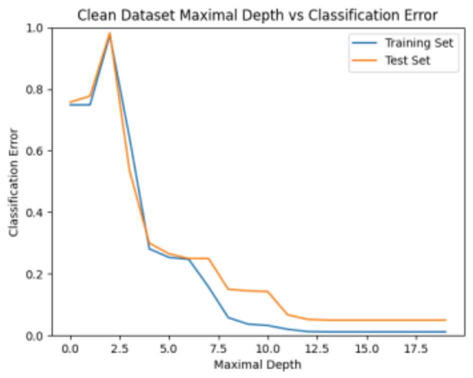


Figure 4: Clean Dataset

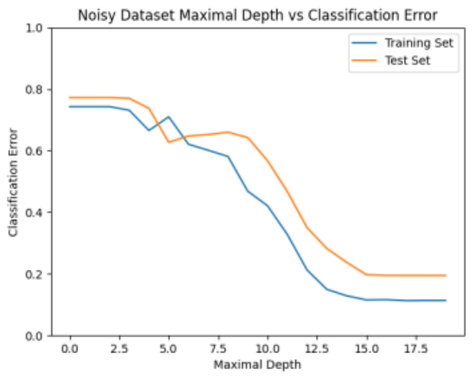
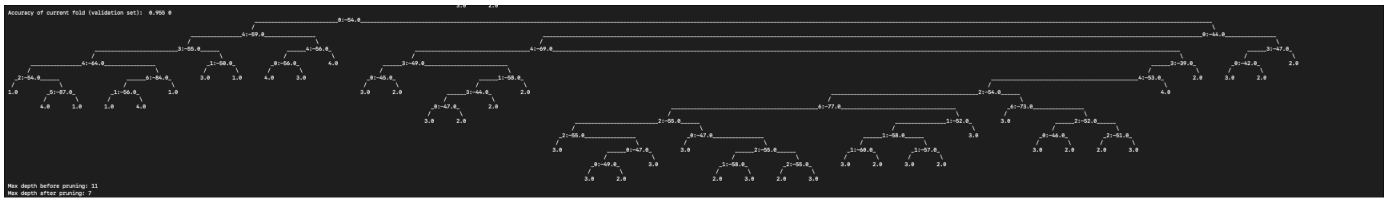


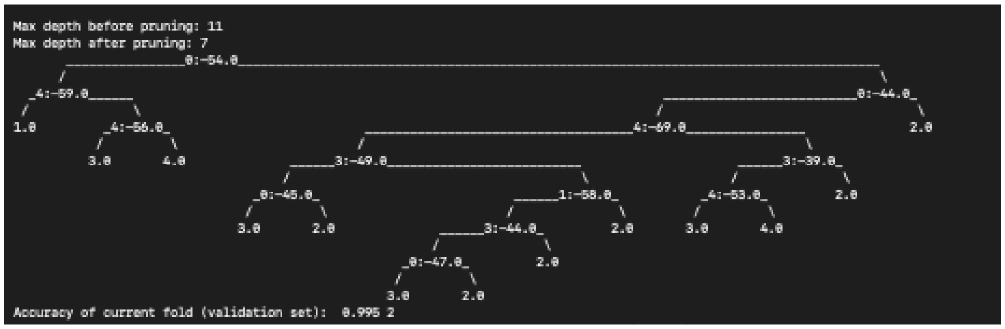
Figure 5: Noisy Dataset

6 Appendix



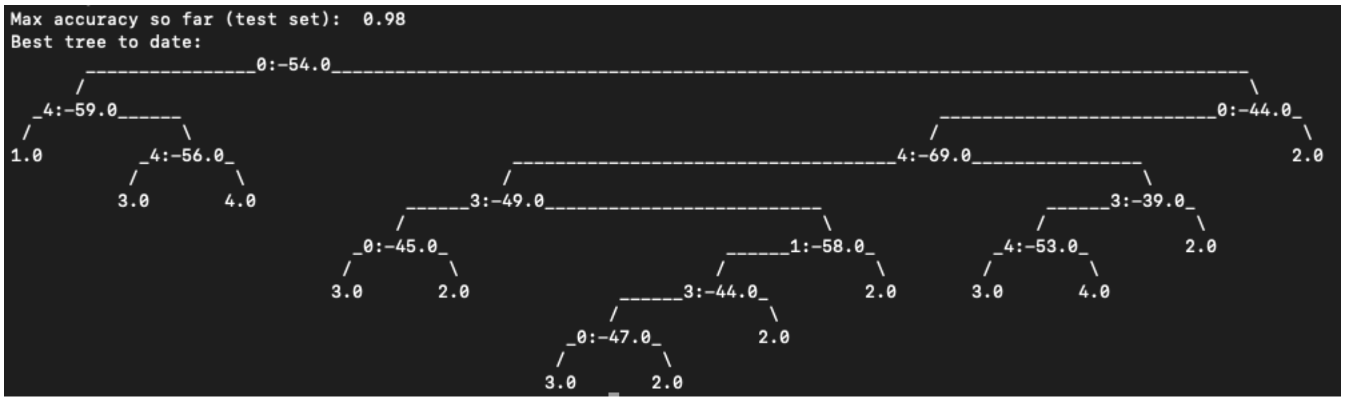
```
                                                                                       3.0      2.0
Accuracy of current fold (validation set):  0.955 0
                                     _____________0:-54.0_____________________________________________________________________________________________________________________________
                                    /                                                                                                                                             \
                          _________4:-59.0_________                                        ___________________________________________________________________________________________0:-44.0_________
                         /                         \                                      /                                                                                                         \
            ____________3:-55.0___              ____4:-56.0_                   __________4:-69.0______________________________________________________________________________                ____3:-47.0_
           /                      \            /            \                 /                                                                                               \              /            \
  ________4:-64.0_________       _1:-50.0_    _0:-56.0_     4.0        ______3:-49.0_______________                                                                    ____3:-39.0_      _0:-42.0_       2.0
 /                        \     /         \  /         \              /                            \                                                                  /            \    /         \
_2:-54.0____          ____6:-84.0_       3.0       1.0  4.0       3.0    _0:-45.0_              ______1:-58.0_                                  ___________________4:-53.0_        2.0  3.0      2.0
/            \        /            \                             /         \            /              \                                       /                          \
1.0      _5:-87.0_   _1:-56.0_      1.0                        3.0       2.0    ______3:-44.0_           2.0                    ____________2:-54.0____                   4.0
         /        \  /         \                                               /              \                                /                        \
       4.0      1.0 1.0       4.0                                         _0:-47.0_          2.0           ________________6:-77.0_______________       _6:-73.0_________
                                                                          /         \                      /                                        \      /                 \
                                                                        3.0       2.0        ___________2:-55.0____                     __________1:-52.0_  3.0       ______2:-52.0_____
                                                                                            /                        \                 /                   \           /                  \
                                                                                 _2:-55.0__________           _0:-47.0_________     ____1:-58.0____        3.0    _0:-46.0_          _2:-51.0_
                                                                                 /                  \        /                 \   /               \             /         \        /         \
                                                                               3.0         _____0:-47.0_   3.0        _____2:-55.0____ _1:-60.0_  _1:-57.0_     3.0       2.0      2.0       3.0
                                                                                          /             \            /                \ /        \  /        \
                                                                                     _0:-49.0_          3.0      _1:-58.0_     _2:-55.0_ 3.0     2.0 3.0      2.0
                                                                                     /         \                 /         \   /         \
                                                                                   3.0       2.0              2.0       3.0  2.0       3.0
Max depth before pruning: 11
Max depth after pruning: 7
```

Figure 6: Best tree found during k-fold tuning before pruning
Each non-leaf node is on the form [Attribute]:[Value to compare against] Each leaf node represents the room determined



```
Max depth before pruning: 11
Max depth after pruning: 7
    ______________0:-54.0_____________________________________________________________________________
   /                                                                                                  \
 _4:-59.0______                                                                 ______________________0:-44.0_
 /             \                                                               /                              \
1.0          _4:-56.0_                              ________________________________4:-69.0______________          2.0
             /         \                           /                                                     \
           3.0       4.0                ______3:-49.0______________________                          ______3:-39.0_
                                       /                                   \                        /              \
                                 _0:-45.0_                          ______1:-58.0_               _4:-53.0_          2.0
                                 /         \                       /              \              /         \
                               3.0       2.0             ______3:-44.0_           2.0          3.0       4.0
                                                        /              \
                                                   _0:-47.0_           2.0
                                                   /         \
                                                 3.0       2.0
Accuracy of current fold (validation set):  0.995 2
```

Figure 7: Best tree found during k-fold tuning post pruning
Each non-leaf node is on the form [Attribute]:[Value to compare against] Each leaf node represents the room determined



```
Max accuracy so far (test set):  0.98
Best tree to date:
      ________________0:-54.0______________________________________________________________________________________
     /                                                                                                             \
 _4:-59.0______                                                                          ____________________________0:-44.0_
 /             \                                                                        /                                     \
1.0          _4:-56.0_                                  ____________________________________4:-69.0________________            2.0
             /         \                               /                                                          \
           3.0       4.0                 ______3:-49.0__________________________                           ______3:-39.0_
                                        /                                       \                         /              \
                                  _0:-45.0_                              ______1:-58.0_                _4:-53.0_          2.0
                                  /         \                           /              \               /         \
                                3.0       2.0                ______3:-44.0_            2.0            3.0       4.0
                                                            /              \
                                                       _0:-47.0_           2.0
                                                       /         \
                                                     3.0       2.0
```

Figure 8: Best pruned tree found using using deterministic ***"randomization"*** function
Each non-leaf node is on the form [Attribute]:[Value to compare against] Each leaf node represents the room determined

```
function postOrderTraversal(current_node):
   If current_node is not null and current_node is not a leaf node then:
       postOrderTraversal(left_child)
       postOrderTraversal(right_child)
       If both left_child and right_child are leaf nodes then:
           Obtain unpruned_accuracy from confusion matrix of tree
           Save current_node room and size attributes
           If number of samples in left_child <= number of samples in
              right_child then:
               Update current_node attributes to right_child attributes and
                  make current_node a leaf node
           Else:
               Update current_node attributes to left_child attributes and
                  make current_node a leaf node
           Obtain pruned_accuracy from confusion matrix of tree
           If unpruned_accuracy > pruned_accuracy then:
               Revert changes using saved current_node attributes
           Else prune the left and right subtrees:
               Set left_child and right_child both to null
```

Figure 9: Algorithm for pruning tree

```
function findSplit(dataSet):
    finalIGValue := -inf
    finalRow := -inf
    finalCol := -inf
    FOR i IN 0 TO numberOfColumns(dataSet)-1:
        dataSet->sortByColumn[i](Ascending)
        FOR j IN 1 TO numberOfRows(dataSet):
            leftSubSet := dataSet[:j, :]
            rightSubSet := dataSet[j:, :]
            infoGain := getInformationGain(dataSet, leftSubSet, rightSubSet)
            IF infoGain > finalIGValue:
                finalIGValue := infoGain
                finalRow := j
                finalCol := i
    RETURN finalRow, finalCol
```

Figure 10: Algorithm for finding best split