Imperial College
London

COURSEWORK 1

DECISION TREES

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Introduction to Machine Learning

Authors:

1. Dhruva Gowda Storz *(01807283)*
2. Alexander F. Spies *(01055106)*
3. Alex Gaskell *(01813313)*
4. Se Hyung Park *(01602366)*

Date: November 4, 2019

Table of Contents

List of Figures

List of Tables

1 Introduction

We report on the efficacy of a decision tree created to predict which of four rooms an individual is in based on the strength of seven WiFi signal readings. To gain an accurate measure of model performance, we implemented 10-fold cross validation and averaged our results to gain global classification accuracy, precision, recall and F1 score. Trees were also pruned to mitigate effects of overfitting, and were trained using both a clean and noisy dataset. Our results show that trees trained on the clean dataset perform with greater accuracy and gain little from the pruning procedure. Inversely, trees trained on the noisy dataset perform considerably worse, but experience a significant increase in performance after pruning.

Figures of trees before and after pruning as well as tables of results are provided in A & B respectively.

2 Implementation

2.1 Decision Tree Learning

Our decision tree algorithm is a recursive function which stores data as a nested dictionary. Our algorithm closely follows that in Algorithm 1 in the coursework specification so we will not repeat it here for the sake of brevity. The entropy quantifies the total amount of uncertainty in the system, therefore we implement a find_split function which determines the feature and value to split on based on maximising the entropy gain (hence it is a greedy algorithm); this operates as follows:

Algorithm 1: Determining splitting feature and feature value

```
Result: Get feature & value for splitting to deliver max entropy gain
for feature in Features do
    sort data by feature;
    for unique value in feature do
        Split data into $subset_A, subset_B$ using this value;
        for $subset_A, subset_B$ do
            Compute entropy for sub dataset (using Equation 1);
        end
        Compute entropy gain from splitting on feature (using Equation 2);
        Return value & entropy gain for max entropy gain;
    end
    Return feature & value for max entropy gain from splitting on each feature;
end
```

Where the quantities allowing us to gauge the information contained within a set of labels are the entropy, H, and gain, defined respectively as:

$$H(data) = \sum_{l=0}^{L} \frac{|data\, with\, label\, l|}{|data|} \log_2 \frac{|data\, with\, label\, l|}{|data|} \quad \text{and} \tag{1}$$

$$Gain = H(dataset) - \left(\frac{|subset_A|}{|dataset|} H(subset_A) + \frac{|subset_B|}{|dataset|} H(subset_B) \right). \tag{2}$$

Our decision_tree_learning function therefore constructs the tree by calling find_split initially on the whole dataset, then splitting on the feature and value returned, and recursively repeating this process until exhaustion. When the sub dataset from a split contains only a single label, then a leaf node is returned (with this label), otherwise a dictionary is returned and the recursion continues.

2.2 K-fold Cross Validation

Our evaluation was performed using 10-fold cross validation in order to determine a robust measure of our algorithm's performance; accounting for the potential of overfitting to yield deceptively high accuracies, and for statistical fluctuations originating from shuffling the data to result in easier test sets. Algorithm 2 details our implementation of k-fold cross validation:

Algorithm 2: K-fold cross validation

```
Result: Confusion matrices for K best pruned trees
Shuffle data;
Split data into K folds, each with (data_length / K) rows;
confusion_matrices = [];
for test_fold in folds do
    test_data = test_fold;
    folds_inner = folds except test_fold;
    confusion_matrices_inner = [];
    for val_fold in folds_inner do
        data_validation = fold_inner;
        data_train = folds_inner except val_fold;
        tree ← create tree on training data;
        pruned_tree ← prune tree using validation data;
        confusion_matrix ← evaluate pruned_tree using test_data;
        Append confusion_matrix to confusion_matrices_inner
    end
    Append the confusion matrix for the tree with the highest classification accuracy in Confusion_matrices_inner
      to Confusion_matrices;
end
```

As shown, our 10 fold cross validation implementation trains 9 trees on an inner loop, prunes each of these using the validation data and tests their performance against test data, saving the best performing tree. This process is then repeated 10 times, once for each outer loop. The result is a list of 10 confusion matrices representing the best-performing tree from its inner loop. We are careful to ensure that the data is split into 10 folds at the beginning of cross validation and remains that way, ensuring that there is never any overlap between train, test and validation sets. The details as to how these matrices were averaged to compute the global average for our algorithm are discussed in subsection 2.3.

2.3 Computing Average Results

To derive meaningful conclusions from the results of running 10-fold cross validation, we average over the metrics calculated for each tested tree in every fold. However, in order to account for potential bias due to imbalances in the features for a given test set, we normalized the confusion matrix by row, i.e., by feature. This procedure resulted in the calculation of a single unbiased confusion matrix for the entire cross validation procedure, from which the recall, precision and F1 were then derived, as discussed in section 3.

3 Evaluation

3.1 Metrics

The classification accuracy alone provides only a crude measure of model performance. As such, we introduce other useful metrics which can be calculated from the confusion matrix. Recall is defined as

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}, \tag{3}$$

capturing how well our model labels the readings from a room compared to the true number of readings from that room.

Precision is defined as

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}, \tag{4}$$

and provides a measure of how likely it is that a reading was actually from a given room after the model predicted it was from that room.

The F1 measure is defined as the harmonic mean of precision and recall, taking the form

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{5}$$

F1 acts as a single metric for performance of the model, combining the precision and recall metrics, taking into account any possible precision-recall trade-off.

3.2 Results

For the clean dataset, before pruning, the average classification rate was 97.1%, with precision of 99.1% and 98.6% for rooms 1 and 4 respectively. The trained trees had relatively more difficulty classifying rooms 2 and 3, however, misclassifying between them to similar degrees (with precision rates of 95.9% and 94.9% respectively, and with similar recall rates). The trained decision trees were still generally a good representation of the test set. The full results are shown on Table 2 and Table 3.

Table 1 shows the classification accuracies which resulted from the cross-validation procedure for both datasets, with and without pruning. We see that for the noisy dataset the average classification rate was 80.1% before pruning, significantly lower than for the clean dataset. This was predominantly due to additional random noise in the dataset, especially with the recall rates for each class falling to similar levels (around 80%). In addition, significant misclassification was no longer exclusive to rooms 2 and 3, but spread more evenly amongst all classes. The full results for the noisy dataset are shown on Table 6 and Table 7. Interestingly, pruning increases the classification rate to 88% for the noisy dataset; see subsection 4.2 for a full discussion (complete results seen in Table 8 and Table 9).

	Unpruned	**Pruned**
Clean	0.971	0.969
Noisy	0.801	0.880

Table 1: Classification rates for datasets before and after pruning

4 Questions

4.1 Noisy-Clean Datasets Question

As highlighted in the previous section, there is an observable difference in the performance when using the clean and noisy datasets (see Figure 2). The difference in performance can be partly explained from observing how the two datasets differ, as despite having similar mean values for each attribute per class, the variances for attribute-label pairings were greater for the noisy dataset; in particular, the distributions were observed to have tails which were barely present in the clean dataset. Additionally, the decision trees trained on the noisy dataset were both more dense (more nodes) and deeper on average (so long as the depth parameter was unconstrained), resulting from overfitting to noise in the noisy dataset, and thus detrimenting the tree's performance (see subsection 4.2 for elaboration).

As per Table 1, whilst pruning improves the performance significantly when using the noisy dataset, increasing the average classification rate to 88.0%, the noise still prevents the trees from replicating the performance achieved on the clean dataset. This behaviour results from the instability of naïve decision trees as a learning algorithm, particularly their susceptibility to overfitting. This instability is evidenced by the fact that the relatively subtle differences between the clean and noisy datasets results in drastically different trees, even in spite of pruning. Note that it is possible to overcome the limitations caused by overfitting more reliably by using the random forest algorithm, whereby one trains an ensemble of uncorrelated trees and takes the most common class output (the mode) as the prediction of the ensemble.

4.2 Pruning Question

Our implementation of pruning is recursive and in-place[1]. Figure 1 shows how one step of the recursive algorithm takes place. In essence, the algorithm steps upwards from the bottom left of the tree through to the bottom right, attempting to replace nodes which have two leaf nodes as their children (referred to as *base parents*) for increased classification accuracy. This bottom up recursion is crucial, as it ensures that multiple base parents in a branch can be efficiently replaced with leaf nodes, if deemed beneficial. It is worth noting that the decision to prune based on classification accuracy stems from the nature of the problem which we are attempting to solve. Specifically, as the problem we are solving is the localization of an individual in their home, we are more interested in the average accuracy over long periods of time, especially given that in practice the sampling rate of the wifi signal strenghts would high, and reasonable interpolation would allow us to perform the task of tracking the individual. This would of course not be the approach utilized in a setting such as medical diagnosis, where we specifically aim to minimize the number of false negatives, as flagging something erroneously may be less dertimental than overlooking it[2].

[1]We refer here to post-pruning, as opposed to pre-pruning in which the emphasis is on restricting parameters such as the maximum depth or number of nodes of the tree, before training

[2]Especially as most current AI based medical systems would refer your results to a doctor for further inspection.

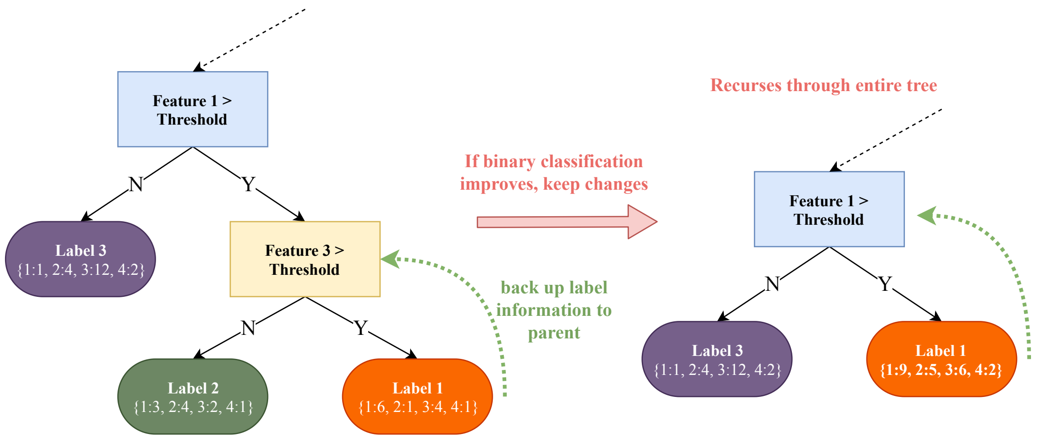

Feature 1 > Threshold
N
Y
Label 3 {1:1, 2:4, 3:12, 4:2}
Feature 3 > Threshold
N
Y
Label 2 {1:3, 2:4, 3:2, 4:1}
Label 1 {1:6, 2:1, 3:4, 4:1}
back up label information to parent
If binary classification improves, keep changes
Recurses through entire tree
Feature 1 > Threshold
N
Y
Label 3 {1:1, 2:4, 3:12, 4:2}
Label 1 {1:9, 2:5, 3:6, 4:2}

Figure 1: A pictorial depiction of a single step in the recursive pruning algorithm, where pruning the node which splits about "Feature 3" improved the classification accuracy, and thus the recursion continued upward. The leaves contain a record of the frequency of labels which were present in their split after training; this is merged after a successful pruning step, and allows the labelling of the resulting leaf node.

It is worth noting that this pruning algorithm is only guaranteed to find the optimal tree (for a given evaluation set) because decision trees are deterministic, and as splits on one side of the tree are not coupled to those on another. Indeed, if this were not the case, one would have a combinatorial blow up as every base node replacement would have to be evaluated in the context of all other possible base node replacements[3].

Figures 3 & 4 show examples of pruned and unpruned trees for the case of the clean data set, respectively. Similarly, Figures 5 & 6 show non-pruned and pruned trees for the noisy data set. The latter pair of trees is considerably more interesting as it clearly shows that the effect of pruning is considerably more dramatic for the noisy dataset, owing to the large prevalence of splitting nodes which resulted from overfitting to noise. This effect is illustrated very clearly in Figure 2b, which shows that the trees overfit significantly to noise, resulting in greater depth and correspondingly more nodes. Consequently, we see that pruning is dramatically beneficial in the case of the noisy dataset, leading to a $\sim 8\%$ increase in classification accuracy on average, and dramatically reducing the number of nodes in a typical tree.

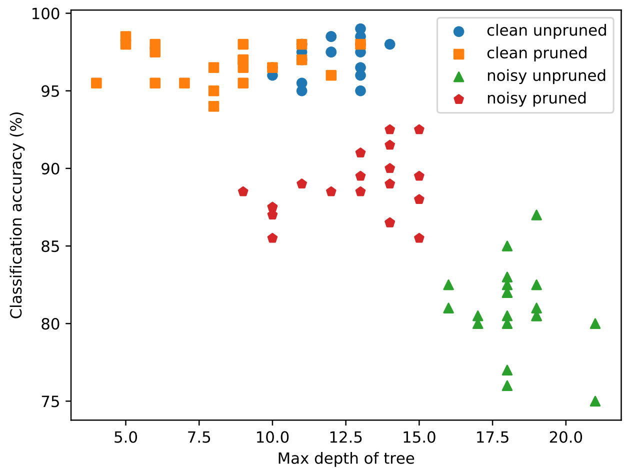
It is also worth noting that whilst Table 1 suggests that pruning worsens the accuracy of the trees trained on the clean dataset. The standard deviation in these results is of order $\sim 1\%$, such that little can be confidently read into such deviations.

As shown by these figures, the effect of pruning is to improve the generalizability of our tree by removing splits that were the result of training to noise; instead focusing on the broader, but more reliable, correlations in the features.

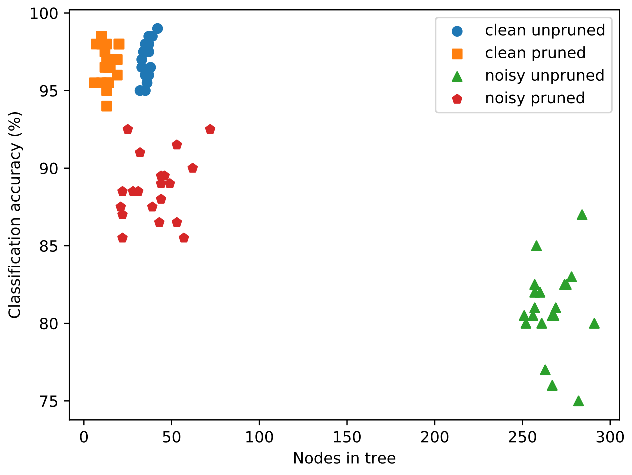
4.3 Depth Question

As alluded to in previous sections, the depth of the tree varies significantly for the two datasets, especially after the application of pruning. To illustrate this, Figure 2 shows how the depth of trees for both the clean and noisy datasets varied over 20 independent training runs, and how the application of pruning affected both the depth of the trees (Figure 2a), as well as the number of nodes which they consisted of (Figure 2b). Indeed, from these figures it is clear that trees trained on the noisy dataset have more nodes and are deeper than the trees trained on the clean dataset.

[3]An issue which one faces in highly non-linear optimization problems.



(a) Classification accuracy against the maximum depth of the trees.



(b) Classification accuracy against the number of nodes in the trees.

Figure 2: The effects of pruning on max tree depth and the number of nodes in the tree (both excluding leaves), and how these correlate with classification accuracy. In both plots, 20 trees were trained then tested before and after pruning.

Furthermore, pruning has a greater impact on both the max depth of the tree and the classification accuracy when used on trees trained on noisy data compared with those trained on clean data, as discussed in subsection 4.2.

From Figure 2a we may be tempted to infer a negative correlation between the max depth and classification accuracy, however, this only holds for the case of the noisy dataset. Indeed, in the context of noisy data such a relationship is to be expected as pruning helps to reduce meaningless overfitting. Conversely, for a pristine dataset, the tree should only extract meaningful relationships, and thus reach a depth similar to the optimal one.

Additionally, we considered the impact of imposing a ”maximum_depth” hyperparameter on the model, capping the maximum depth of the tree. We observed that for shallow trees (maximum_depth of 2 or below), increasing the maximum_depth improved the classification rate. This is expected as this threshold prevents the tree from learning the underlying distribution of the data, so model performance suffers. However, increasing this threshold further (using the noisy dataset), overfitting starts to take over so the classification accuracy falls again, and in the case of the clean dataset there are no meaningful splits left to make. Therefore, we conclude that the relationship between maximum depth and classification accuracy begins positive, for all cases, but plateaus and turns negative in some cases as max depth rises due to overfitting.

Appendices

A Decision Tree Pruning

A.1 Clean Dataset

Here we demonstrate the effect of pruning on a decision tree trained on 1800 samples of the clean data set, and pruned on the remaining 200. For the sake of clarity in the visualization, the tree depth was capped at 5.[4]

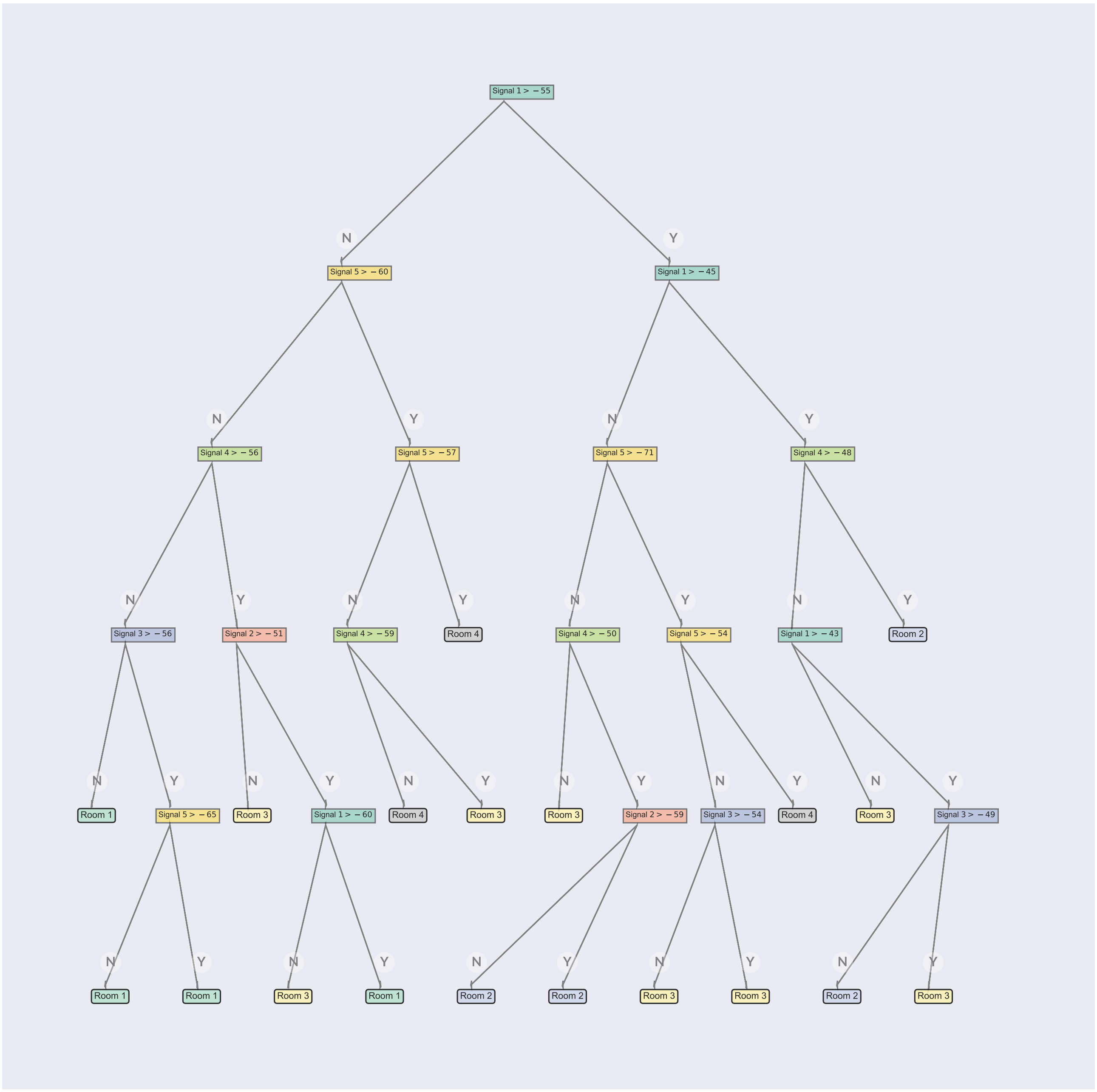


Figure 3: Resulting unpruned tree for the clean dataset

[4]These visualizations were using the Visualizer class which we implemented.

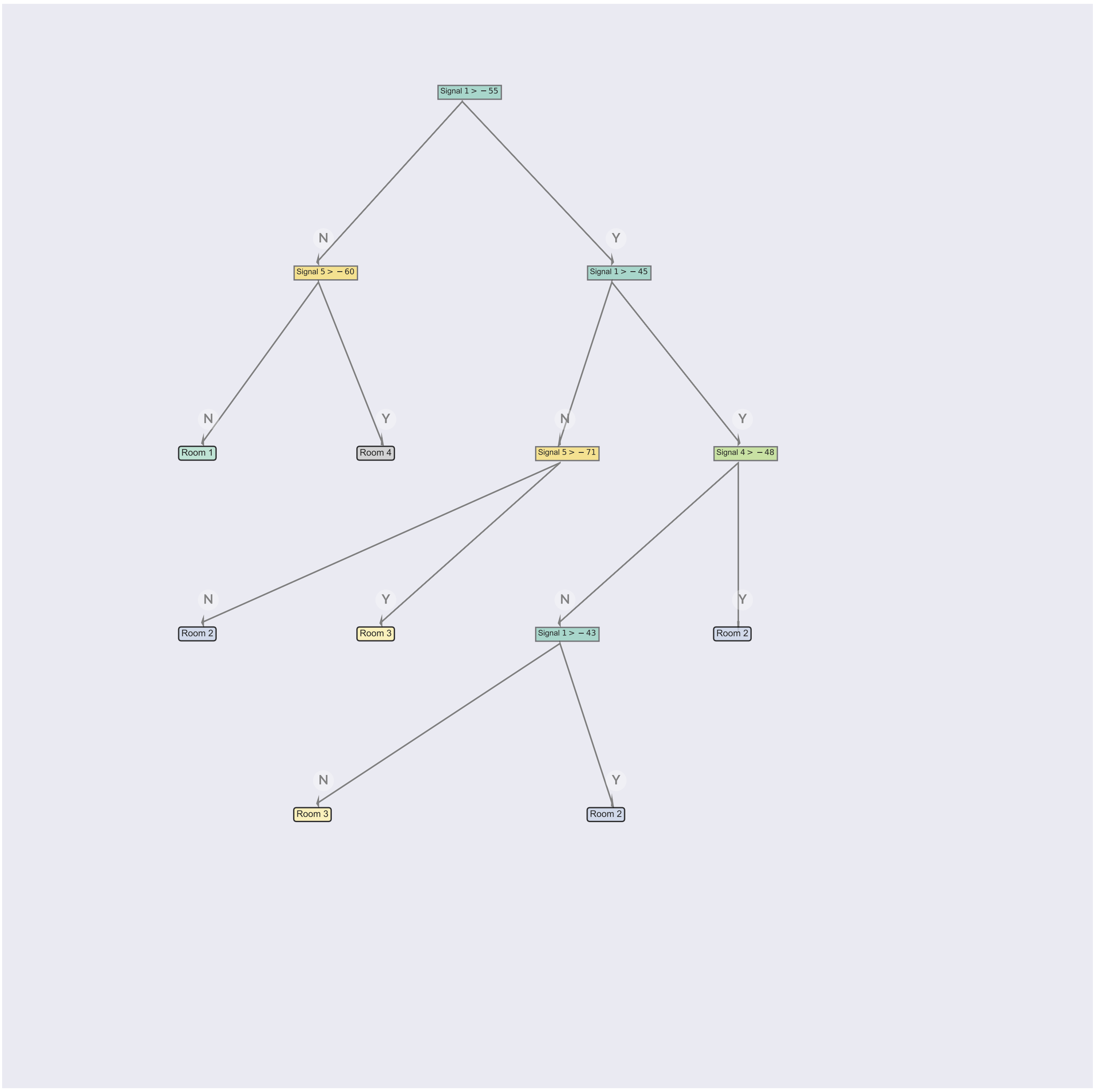


Figure 4: Resulting pruned tree for the clean dataset

A.2 Noisy Dataset

Here we demonstrate the effect of pruning on a decision tree trained on 1800 samples of the noisy data set, and pruned on the remaining 200. For the sake of clarity in the visualization, the tree depth was capped at 5.[5]

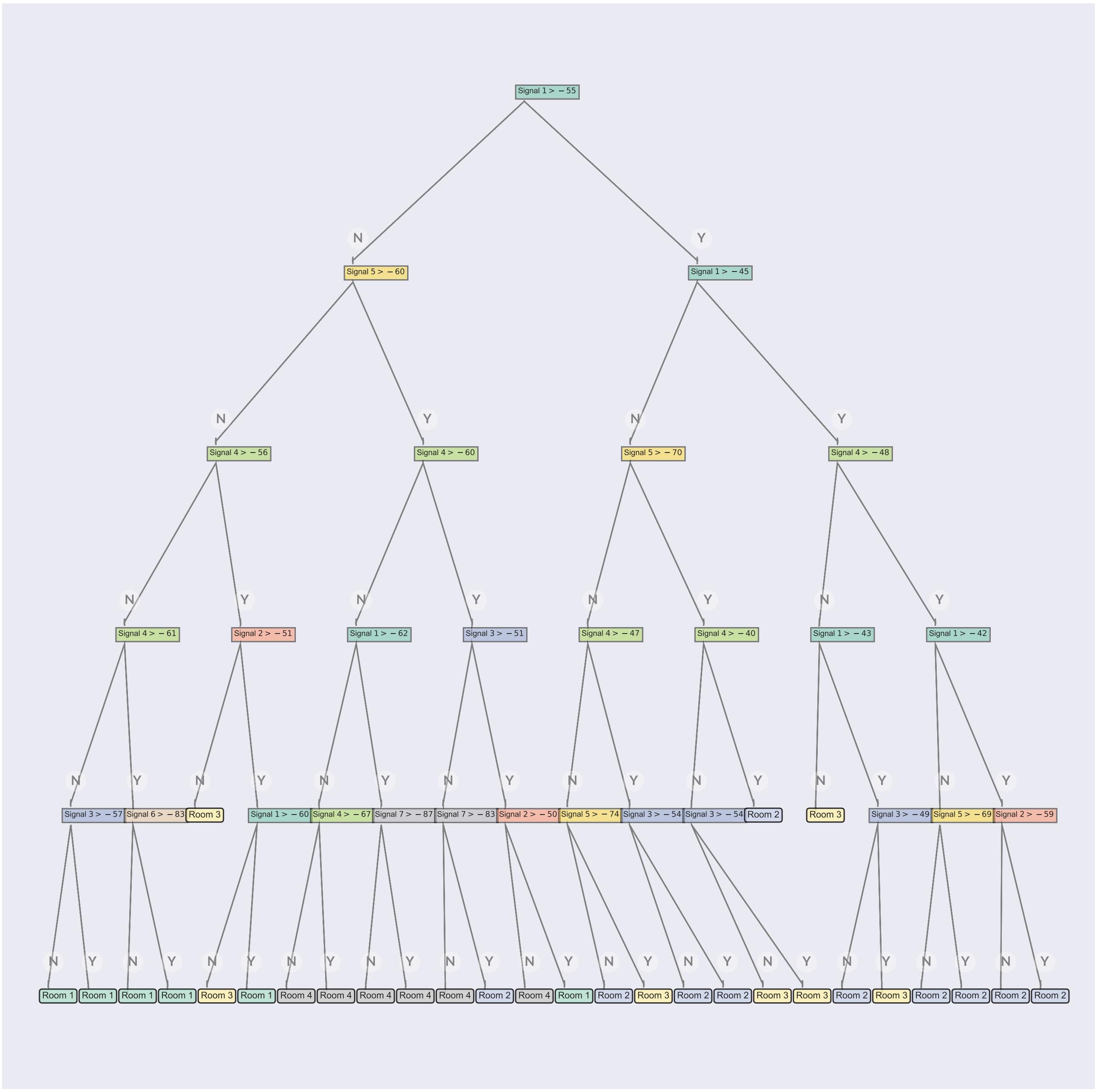


Figure 5: Resulting unpruned tree for the noisy dataset

[5]These visualizations are automatically generated by the Visualizer class which we implemented

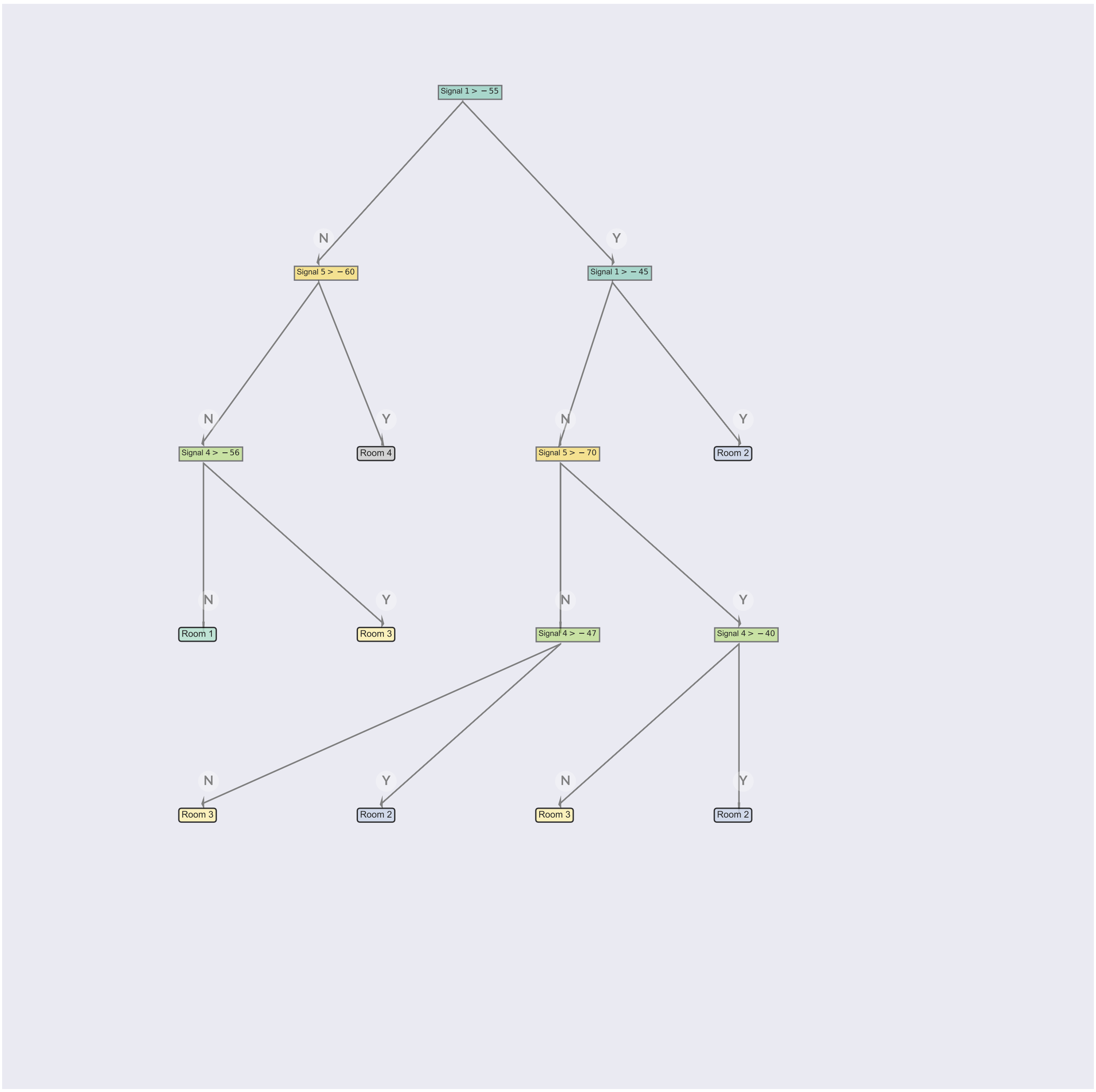


Figure 6: Resulting pruned tree for the noisy dataset

A.3 Entire Dataset

Here we demonstrate the effect of pruning on a decision tree trained on 3600 samples of the total data set, and pruned on the remaining 400. The tree depth was not capped.

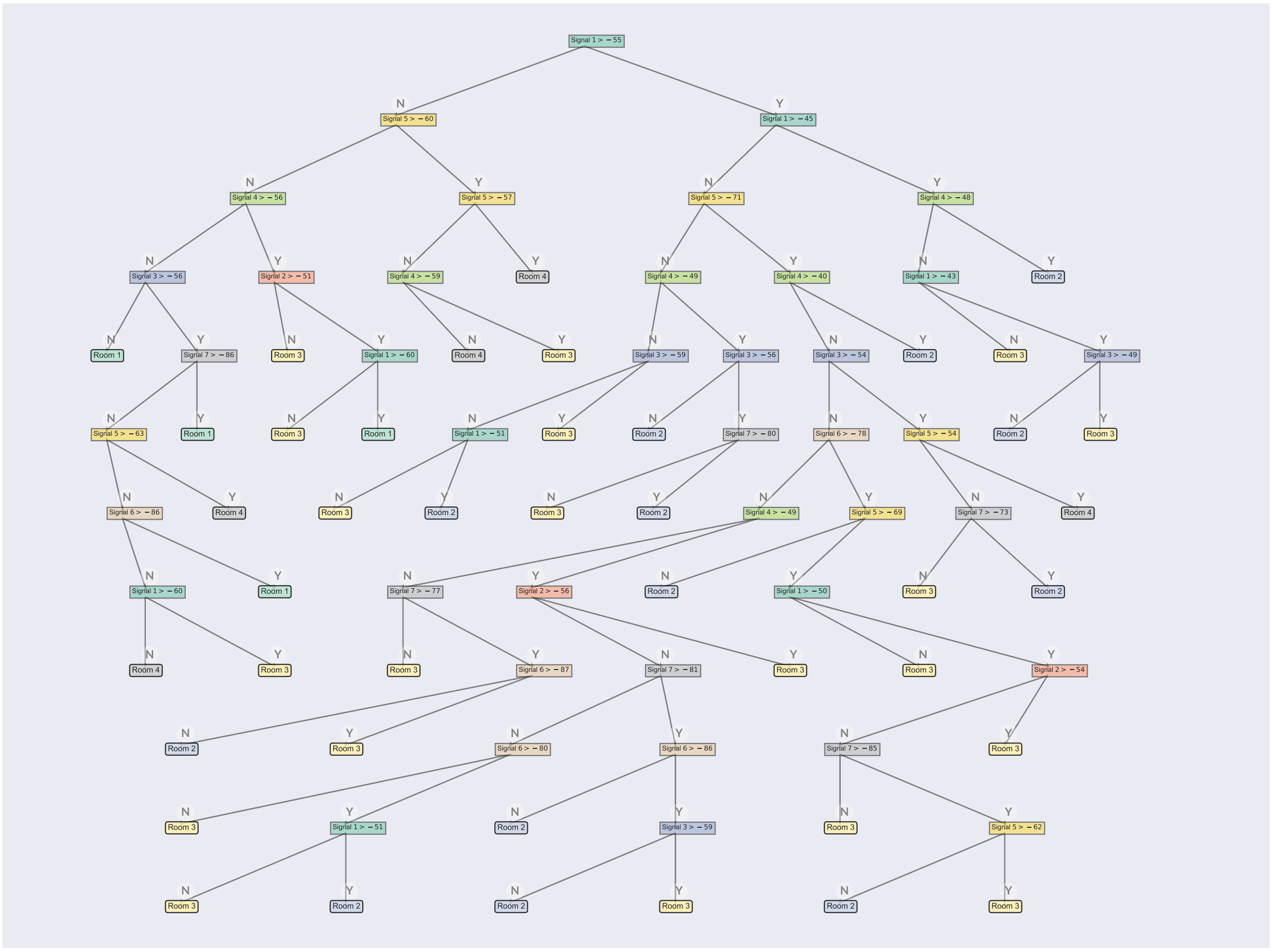


Figure 7: Resulting unpruned tree for the entire dataset

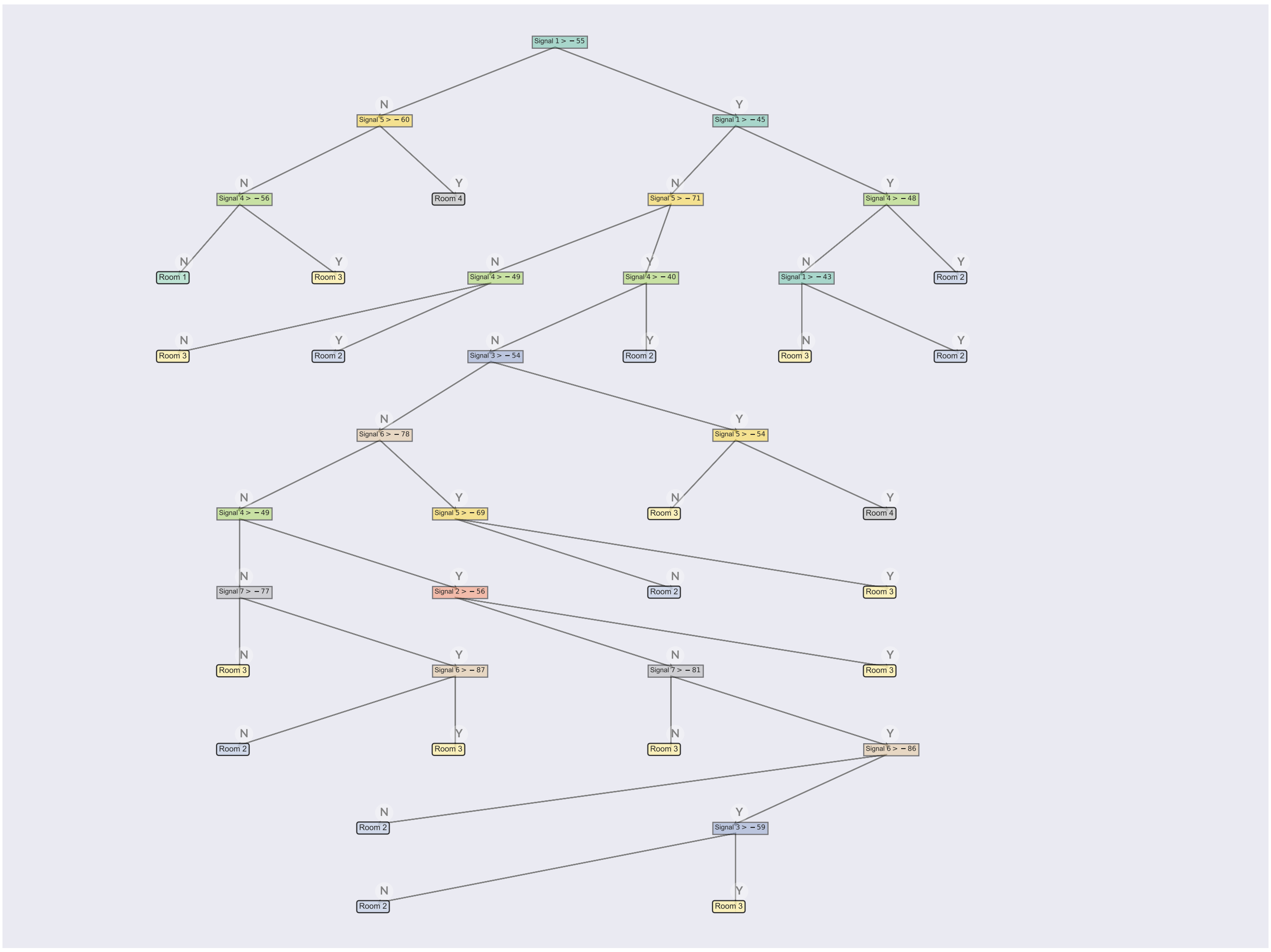


Figure 8: Resulting pruned tree for the entire dataset

B Tables

B.1 Unpruned Clean:

	Room 1 Predicted	**Room 2 Predicted**	**Room 3 Predicted**	**Room 4 predicted**
Room 1 Actual	0.986	0.000	0.006	0.008
Room 2 Actual	0.000	0.959	0.041	0.000
Room 3 Actual	0.001	0.044	0.949	0.006
Room 4 Actual	0.008	0.000	0.003	0.990

Table 2: Unpruned clean confusion matrix

	Room 1	**Room 2**	**Room 3**	**Room 4**
Recall	0.986	0.959	0.949	0.990
Precision	0.991	0.957	0.951	0.986
F1	0.988	0.958	0.950	0.988

Table 3: Average recall, precision and F1 for the unpruned clean dataset

B.2 Pruned Clean:

	Room 1 Predicted	**Room 2 Predicted**	**Room 3 Predicted**	**Room 4 predicted**
Room 1 Actual	0.992	0.000	0.007	0.002
Room 2 Actual	0.000	0.959	0.041	0.000
Room 3 Actual	0.007	0.047	0.939	0.007
Room 4 Actual	0.009	0.000	0.004	0.987

Table 4: Pruned clean confusion matrix

	Room 1	**Room 2**	**Room 3**	**Room 4**
Recall	0.992	0.959	0.939	0.987
Precision	0.984	0.953	0.949	0.991
F1	0.988	0.956	0.944	0.989

Table 5: Average recall, precision and F1 for the pruned clean dataset

B.3 Unpruned Noisy:

	Room 1 Predicted	Room 2 Predicted	Room 3 Predicted	Room 4 predicted
Room 1 Actual	0.791	0.061	0.066	0.082
Room 2 Actual	0.056	0.809	0.085	0.050
Room 3 Actual	0.056	0.073	0.800	0.072
Room 4 Actual	0.077	0.044	0.071	0.808

Table 6: Unpruned Noisy Confusion Matrix

	Room 1	Room 2	Room 3	Room 4
Recall	0.791	0.809	0.799	0.808
Precision	0.807	0.820	0.782	0.799
F1	0.799	0.815	0.790	0.803

Table 7: Average recall, precision and F1 for the unpruned noisy dataset

B.4 Pruned Noisy:

	Room 1 Predicted	Room 2 Predicted	Room 3 Predicted	Room 4 predicted
Room 1 Actual	0.902	0.023	0.032	0.043
Room 2 Actual	0.039	0.887	0.050	0.023
Room 3 Actual	0.044	0.066	0.853	0.036
Room 4 Actual	0.053	0.028	0.038	0.881

Table 8: Pruned noisy confusion matrix

	Room 1	Room 2	Room 3	Room 4
Recall	0.902	0.887	0.853	0.881
Precision	0.869	0.883	0.876	0.896
F1	0.885	0.885	0.865	0.888

Table 9: Average recall, precision and F1 for the pruned noisy dataset