Introduction to ML - Decision Tree Coursework

Ho Yin Kong, Jin Xian Yap, Emily Haw and Dhivya Ravindran

October - November 2020

1 Implementation Details

1.1 Building the model

Our decision tree learning model works by first generating decision trees from a given training dataset. This works by recursively splitting the sorted dataset in order to achieve the greatest information gain per tree branch split. With this recursive approach, all the data in the dataset ends up being represented by a leaf stating the data point's label, reached by following decisions down the tree. These decisions relate to whether that data point's certain attribute is below or above or equal to a certain value. Once the tree is generated, it can either be directly evaluated against a testing dataset separate from the training dataset or use a validation dataset to prune the tree to produce a simpler module. Details of the specifics of the implementations follow.

Our split function works by finding the split that obtains the highest information gain. This is achieved by iterating through each attribute for the dataset. For each attribute, we sort the dataset by the attribute, and consider information gain from splitting on the attribute at the midpoint value between two adjacent data points with differing labels. This allows us to find the attribute and value split that provides the highest information gain. Information gain is calculated using the same formula as given in the slides.

Once the split point is found, the node for the set is created, and its attribute and value is set for later comparisons' use. The splitting algorithm is then applied to the two split sub datasets, and this process repeats recursively. Once the dataset becomes of size 1, a leaf node is created, storing the data point's label. With this, the entire decision is generated.

1.2 Using the model to classify an entry

Using the decision tree to predict a set of attribute's room classification is easily done by applying traversing through the tree with a given data point's attributes, taking left or right subtrees depending on the booleans evaluated at each node. Once a leaf node is reached, the datapoint can be given the corresponding label.

1.3 Evaluation

The confusion matrix of the whole dataset is calculated by taking the sums of the confusion matrix of each individual fold. The confusion matrix that gets added to the sum of the matrices comes from the test dataset that is evaluated against the tree with the higher accuracy after pruning is completed. Average recall rates, precision rates and F1 scores for each of the classes are also calculated based on this confusion matrix.

1.4 Cross-validation and Pruning

Our cross validation function splits the dataset into 3 parts: training data, validation data and test data. For each iteration, we allocate one fold to the test data and one to the validation data, using the remaining 8 folds to train the model. The accuracy of the pre-pruned model is noted in each iteration and summed. After noting the accuracy of the pre-pruned tree, we then prune the tree using the validation dataset. We then note the accuracy of the pruned tree after pruning has completed. This is also summed. After all the

iterations have completed, we use the sums of the pre-pruned and post-pruned tree accuracies to find the average accuracy of our pre-pruned and post-pruned models.

The pruning function called from cross validation takes the root node of the tree, a node to examine and the validation dataset. This function is called recursively from the bottom-up. If a node is a parent of 2 leaf nodes, the leaf nodes are simply deleted as each node already keeps track of the majority label. If the pruned node has a better accuracy on the validation set then the unpruned node, we keep it. If not, we revert the node back to its previous state. This function is called recursively until all nodes that are parents of leafs in the tree have been considered.

2 Results from Evaluation

2.1 Test Accuracy from each Iteration

i-th fold	0	1	2	3	4	5	6	7	8	9	Average
Unpruned	0.975	0.970	0.965	0.955	1.000	0.975	0.970	0.970	0.975	0.985	0.974
Pruned	0.960	0.960	0.960	0.970	0.990	0.970	0.950	0.960	0.965	0.975	0.968

Table 1: Clean Dataset

i-th fold	0	1	2	3	4	5	6	7	8	9	Average
Unpruned	0.830	0.820	0.795	0.775	0.830	0.750	0.820	0.760	0.795	0.825	0.800
Pruned	0.905	0.880	0.885	0.880	0.865	0.855	0.875	0.890	0.875	0.880	0.879

Table 2: Noisy Data Set

2.2 Evaluation of Clean Dataset

	Class 1 Predicted	Class 2 Predicted	Class 3 Predicted	Class 4 Predicted
Class 1 Actual	497	0	2	1
Class 2 Actual	0	478	22	0
Class 3 Actual	3	14	480	3
Class 4 Actual	4	0	1	495

Table 3: Average Confusion Matrix after pruning of Clean Dataset

	Average Recall Rate (%)	Average Precision (%)	Average F1-measure (%)
Class 1	99.38	98.56	98.97
Class 2	95.75	97.25	96.49
Class 3	95.82	95.06	95.44
Class 4	99.08	99.31	99.20

Table 4: Average Evaluation Metrics after pruning for the Clean Dataset

The average classification rate of our model was 96.8%, In addition, all classes had a relatively high recall rate between 95% to 99%, indicating that our model has a low percentage of false negatives across all 4 classes. Thus, there is a low probability that our model incorrectly classifies a data point belonging to a certain class to another class.

Besides, all classes also had a relatively high precision rate above 95%, indicating that there was a low probability that our model missed true positives. There is a low probability of data points being identified as false positives across all four classes.

Finally, the F1 score of all classes are relatively high, indicating a high performance of our model, since it indicates that our model has both high precision and recall rates for all classes. This also supports the high classification rate of our model.

2.3 Evaluation of Noisy Dataset

	Class 1 Predicted	Class 2 Predicted	Class 3 Predicted	Class 4 Predicted
Class 1 Actual	434	17	18	21
Class 2 Actual	18	434	32	13
Class 3 Actual	26	29	439	21
Class 4 Actual	25	13	23	437

Table 5: Average Confusion Matrix after pruning of Noisy Dataset

	Average Recall Rate (%)	Average Precision (%)	Average F1-measure (%)
Class 1	88.83	86.61	87.71
Class 2	87.11	87.80	87.46
Class 3	85.27	85.68	85.47
Class 4	87.95	89.05	88.50

Table 6: Average Evaluation Metrics for the Noisy Dataset

The average classification rate for the noisy dataset of our model was 87.9%, which was slightly lower as compared to that of the clean dataset. In addition, all classes still had a relatively high recall rate between 85.2% to 88.8%, despite having a lower value as compared to the clean dataset.

Despite this, our model still have a low percentage of false negatives across all 4 classes. However, there is a slightly higher probability that our model incorrectly classifies a datapoint belonging to a certain class to another class as compared to when the model is applied to the clean dataset. Besides, all classes also had a relatively high precision rate between 85% and 89%, indicating that there was a low probability that our model missed true positives. There is still a low probability of datapoints being identified as false positives across all four classes.

Finally, the F1 score of all classes are relatively high, indicating a high performance of our model, since it indicates that our model has both high precision and recall rates for all classes. This also supports the high classification rate of our model.

3 Visualisation of the Decision Tree

Figure 1: Pruned and Unpruned Tree for Clean Dataset

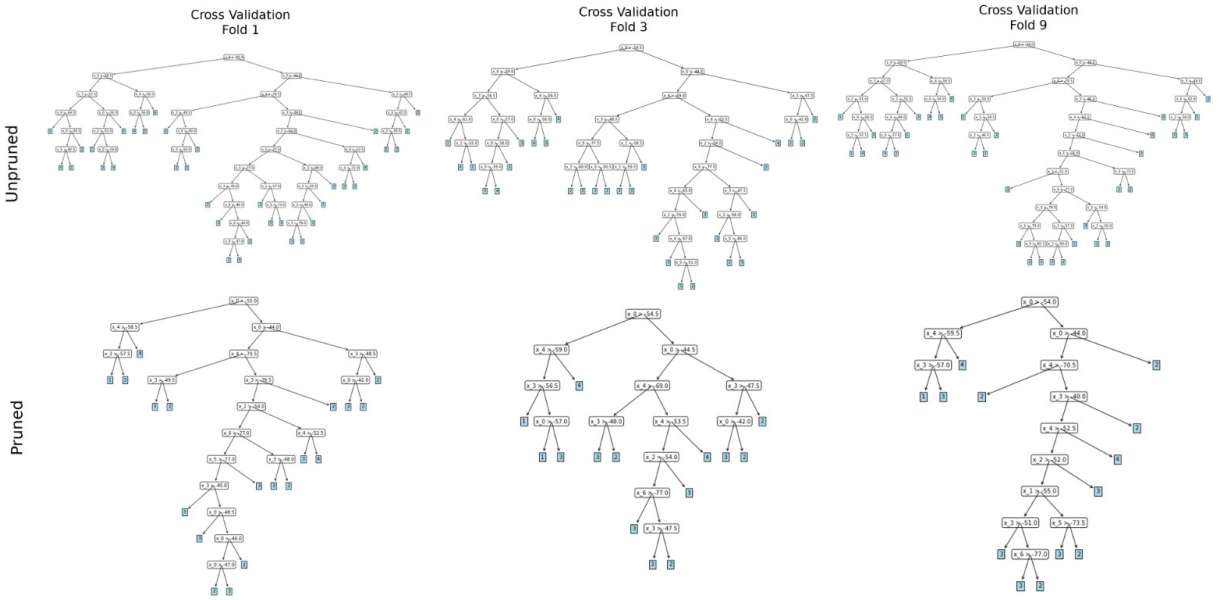
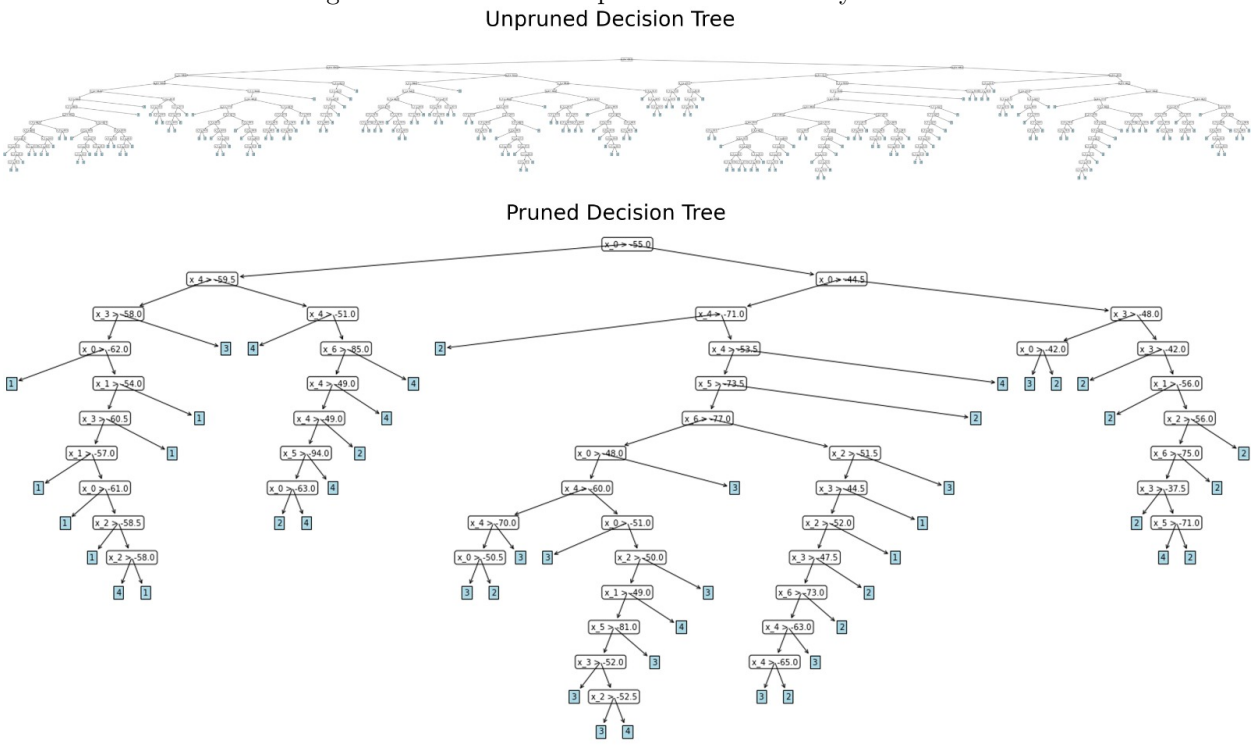


Figure 2: Pruned and Unpruned Tree for Noisy Dataset



3.1 Clean Dataset

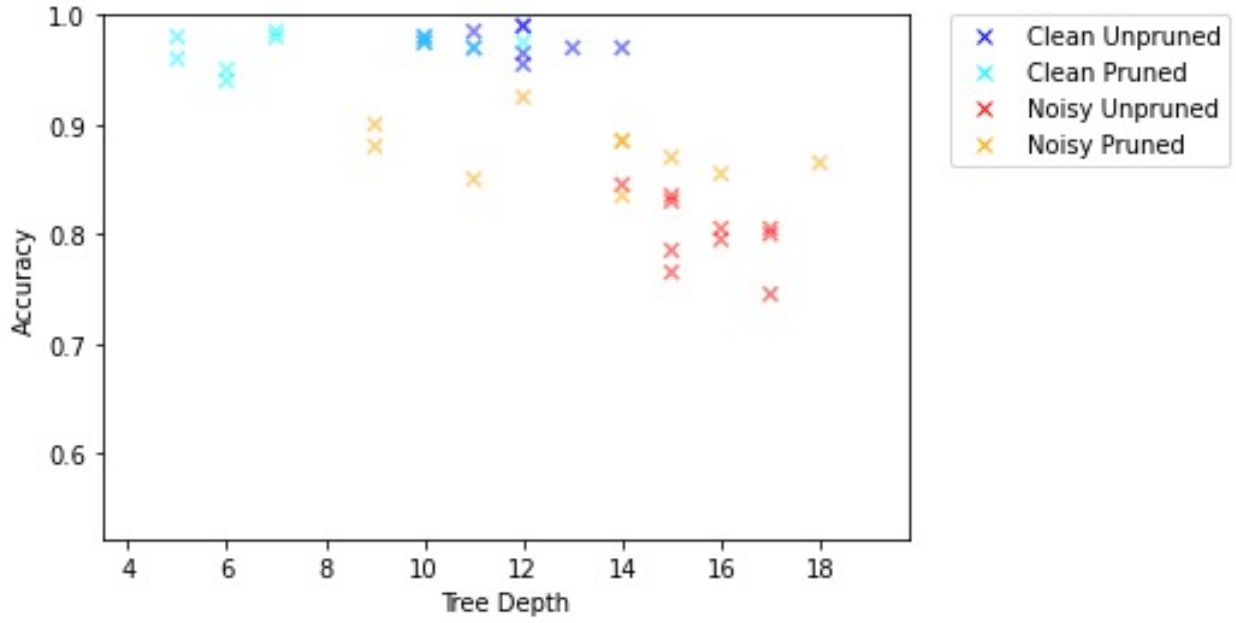
Evidently, pruning the trees generated from the 'clean' dataset (as seen in Figure 1) resulted in decision trees with generally fewer leaves, though the decrease in depth of the trees are not significant. Though the pruned decision trees produce higher accuracy on the validation dataset, they fail to improve on the test accuracy produced by the original decision trees.

3.2 Noisy Dataset

Pruning on decision trees trained using the noisy dataset (as seen in Figure 2) has decreased the number of leaves of the decision tree produced by a significant ratio. According to evaluation data, it also gives a greater test accuracy than the unpruned decision tree.

4 Answering the Questions in Part 2

Figure 3: Depth against Accuracy for Unpruned and Pruned Datasets



4.1 Noisy-Clean Datasets

As shown above in the evaluation metrics for both clean and noisy datasets, our final model performed better on the clean dataset as compared to the noisy dataset, giving a higher average accuracy rate of 96.8% as compared to 87.9%, resulting in a difference of 8.9% between both datasets. The model also performed better for each class in the clean dataset with F1 rates for all classes at above 95%. In contrast, our final model gave a lower F1 score for all classes, averaging at 87.3% across all 4 classes.

The drop in performance for the noisy dataset is due to the model being overfitted to random noise in the data that does not necessarily reflect the true underlying patterns in the dataset. When this same model is used to classify a separate test dataset, this overfitting on random noise on the training data causes the classification accuracy for the test data to drop.

Across both pruned and unpruned datasets for both clean and noisy data, the model after pruning had similar, if not better performance for both clean and noisy datasets. Without pruning, the training on the

clean dataset gave an average accuracy of 97.4% while that on the noisy dataset had an average accuracy of 80.0%.

4.2 Pruning

Our pruning function works recursively. Each call is given a reference to the root node of the decision tree, as well as access to the validation set. This validation set was chosen using Method 1 of cross-validation with hyperparameter tuning in the notes. This means that each validation set is the fold immediately after the testing set fold, for all the iterations of folding. Each node in the decision tree also has a reference to its majority label count, for its corresponding subtree.

Within the pruning function, if the function enters a leaf node, it returns. If the node is not a leaf, we know that there is both a left and right child by the way decision trees are constructed. The function recurses down and calls prune on both children. Once both prune functions have returned, we check whether both the left and right children are leaf nodes. If so, it means we can try to prune the current node.

The tree's accuracy is evaluated against the validation set, before it stashes the left and right leaves away and sets its label to be the majority label. Then, the pruned tree's accuracy is evaluated. If the new accuracy is not an improvement, the left and right children are set back to the stashes leaves. Pruning the noisy dataset led to an increase in accuracy of 7.9% whereas pruning the clean dataset led to a small negative accuracy difference, from 97.4% to 96.75%. We believe this may be caused by using a validation set that was specific to each fold in the outer cross-validation loop and perhaps the pruned tree for the clean dataset. In another experiment, we might explore using a nested cross-validation strategy instead, where we split the data into folds one tenth the original size, use 1 fold for testing and then split the remaining data into folds of one ninth its size, and hence producing 90 trees.

4.3 Depth

For the clean dataset, we had an average maximal depth of an unpruned tree of 11.1 across all folds. After pruning, we got an average maximal depth of 9.4 across all folds. For this dataset, we saw a drop of 0.65% in prediction accuracy. We attribute this drop in accuracy to a trade off resulting in a simpler model that is easier to understand and more robust. This is as the leaves of the tree were pruned resulting in a lower maximal depth and hence, a more general model.

For the noisy dataset, an unpruned decision tree had an average maximal depth of 17.7. After pruning, our decision tree had an average maximal depth of 12.5 across all folds. This is a significant drop of 5.2 folds. Prediction accuracy also rose by 7.9% after pruning. While, similarly to the clean dataset, accuracy would have been traded for simplicity; in this case, the decision tree would have been over fitted to a noisy dataset prior to pruning due to the randomness of the dataset. This is why the simplicity achieved through pruning the tree gives us a more general and accommodative model which achieves a higher accuracy.