Decision Tree Coursework
— CO395 Intro to Machine Learning —

Ryan Ong, Konstantinos Ntolkeras, Abhilash Dubbaka, Aleix Cambray Roma

10th February, 2019

1 Summary of Implementation

For this coursework, we were given two datasets, clean and noisy, and we had to perform decision tree analyses on these. The first part of the implementation was retrieving the data for the clean and noisy data and loading them as array objects. Following this, we had a cross-validation function, which would create decision trees, evaluate them and give out metrics on accuracy, the confusion matrix, recall, precision and F1 scores. These steps are explained in further detail below. To perform the cross-validation (CV), we began with randomly splitting the dataset into 10 equal segments to perform a 10 fold CV. Then, the following steps are performed:

Step 1. We take one segment as the test and the remaining 9 segments as the training + validation set.

Step 2. We first use this entire training+validation set to create a decision tree using the decision tree learning algorithm, which is further explained below.

Step 3. We then evaluate this tree on the test set and find out the accuracy, the confusion matrix as well as the recall, precision, F1 scores for each class. We store these metrics in corresponding arrays.

Step 4. Following this, we take the training + validation set and split it up based on the same split segment we had earlier and this time, we set aside one segment as the validation set and the other 8 are taken as the training set.

Step 5. For the training set, we train the tree using the decision tree learning algorithm and then we prune based on the validation set. The process of pruning is explained in a later section of this report.

Step 6. Using this pruned tree, we evaluate against the test set, similar to **Step 3** and save the corresponding metrics.

Step 7. We then move the validation set to take another segment of the 9 segments we had in **Step 4** and the training set to take the remaining 8 segments. Then **Steps 5** and **6** are repeated. This process repeats until the validation set has been each of the 9 segments. This would result in 9 set of evaluation metrics, which we store in corresponding arrays.

Step 8. Once **Step 7** is completed, we do a similar process, where we move the test set to another segment of the original 10 and make the training and validation sets the remaining 9 segments. Then, **Steps 2** to **7** are repeated until the validation set has been each of the 10 segments.

The end result of this CV process is one set of arrays for the different metrics for the 10 unpruned trees, and another set of arrays for the different metrics for the 90 pruned trees. Using the pruned trees, we calculate the average accuracy of all trees, average precision, recall and F1 score per class of all trees. We do the similar process with all the 90 pruned trees to get average metrics.

For the **decision tree learning process**, we first check if the dataset is a leaf set i.e. the room allocations for each sample in that set are the same. If it is, we create a leaf node. Otherwise, we split the data based on the best attribute, which gives us the most information gain. In essence, we take this dataset and order it in ascending order by one column, then we work our way down the samples (rows) comparing the room allocation in the last column against the previous sample (row) room allocation. If they are different, we consider the value in the ordered column on the previous row as a possible split point. Any rows with the same attribute value or less as the previous row in the ordered column are taken as the left dataset and everything else is taken as the right dataset. Then we calculate the information gain as per the below formula.

$$G(q) = H(\boldsymbol{dataset}) - \left(\frac{|\boldsymbol{subsetA}|}{|\boldsymbol{dataset}|} H(\boldsymbol{subsetA}) + \frac{|\boldsymbol{subsetB}|}{|\boldsymbol{dataset}|} H(\boldsymbol{subsetB}) \right)$$

where $|dataset| = |subsetA| + |subsetB|$. The entropy is calculated as $H(V) = -\sum_k P(v_k) log_2(P(v_k))$.

We do the same process for all the other possible split points in the column and then order the data by the next column and consider all the split point and so on, until we considered all the possible columns excluding the last column, which has the room allocations. Then, we compare all the information gains calculated and split the data based on the split point that gives the highest information gain. Then, we repeat this process on the two separate datasets until all the samples are split into leaf nodes. This is essentially following the ID3 algorithm. If there were instances where all the attributes were exactly the same but the room allocations were different in a dataset, then we took the room allocation with the most occurences in that dataset and assumed that as the leaf node value for all those samples.

For the **evaluation process** between the test data and the trained tree, we take the test data without the room allocation column and using our trained tree, we predict which room each sample will be in and then compare to the actual room allocations to work out the accuracy and the other metrics.

2 Results of Evaluation

2.1 Confusion matrix

Table 1: Average Confusion Matrix (Before Pruning) - Clean dataset

Class	1-predicted	2-predicted	3-predicted	4-predicted
1-actual	48.7	0.0	0.6	0.7
2-actual	0.0	48.9	1.1	0.0
3-actual	0.2	2.4	47.1	0.3
4-actual	0.3	0.0	0.5	49.2

Table 2: Average Confusion Matrix (Before Pruning) - Noisy dataset

Class	1-predicted	2-predicted	3-predicted	4-predicted
1-actual	38.4	2.5	3.6	4.5
2-actual	2.6	39.4	4.8	2.9
3-actual	2.2	4.6	41.7	3.0
4-actual	4.4	3.0	3.7	38.7

2.2 Evaluation metrics (average)

Table 3: Classification report (Before Pruning) - Clean dataset

Class	Avg Precision	Avg Recall	Avg F1 Score
1	0.990	0.975	0.982
2	0.953	0.979	0.966
3	0.955	0.944	0.949
4	0.978	0.985	0.981

Table 4: Classification report (Before Pruning) - Noisy dataset

Class	Avg Precision	Avg Recall	Avg F1 Score
1	0.808	0.783	0.794
2	0.797	0.793	0.792
3	0.778	0.809	0.791
4	0.787	0.778	0.781

In this classification task, we have 4 classes (room 1, 2, 3 and 4). The average classification rate for clean vs noisy dataset is 96.95% and 79.10% respectively. In order to have a deeper understanding of how our models are performing, we will look at the confusion matrix and evaluation metrics.

The confusion matrices above show us a summary of prediction results made by our decision tree algorithm. It shows the number of correct and incorrect predictions, broken down per class, hence allowing us to identify the types of errors (which class is commonly misclassified) our decision tree algorithm are making. The diagonal entry (coloured blue) of the matrix shows us the correct predictions (true positives and true negatives) whereas the non-diagonal entry shows us the incorrect predictions (false positives and false negatives).

Given that we have a balanced test set of 50 sample per class, for the clean dataset, the confusion matrix shows us that, on average, our algorithm has correctly predicted each class at least 47 out

of the 50 samples of each class. This is captured by the high average recall metric in table 3. For the noisy dataset, the confusion matrix shows us that, on average, our algorithm has only correctly predicted each class at most 42 out of the 50 sample. Table 4 shows a lower average recall rate.

$$recall = \frac{True\_Positive}{True\_Positive + False\_Negative}$$

The confusion matrix also shows us the precision of our prediction results, where given a class prediction, how many are true positive (correct classification) and how many are false positive (incorrect classification). For example, in table 1, we can see that, on average, for all the room 1 predictions made by our algorithm on the test set, 48.7 of them are actually room 1 (true positive), 0.2 are actually room 3 (false positive) and 0.3 are actually room 4 (false positive). This shows that our algorithm has a high precision rate when it comes to predicting room 1. This is captured by the high average precision metric in table 3. Table 4 shows a lower average precision rate across class for noisy dataset.

$$precision = \frac{True\_Positive}{True\_Positive + False\_Positive}$$

Using just high precision or recall rate alone doesn't give us a clear picture of how well the algorithm is performing. In a situation where we only have 2 classes, the algorithm will have a high recall rate with low precision if it just predicts all samples with one of the classes. A high precision with low recall rate means that the algorithm missed out on a lot of the true positive predictions which isn't ideal either. This is why it is often better to evaluate our algorithms based on F1-score which is computed as follows:

$$F1\_score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Based on table 3 (clean dataset), F1_scores are very similar across classes, ranging from 0.949 - 0.982, with class room 1 having the highest average F1_score and class room 3 having the lowest average F1_score. This means that, given the clean dataset, our algorithm is best at predicting class room 1 and worst at predicting class room 3. Based on table 4 (noisy dataset), F1_scores are also very similar across classes. However, it ranges significantly lower from 0.781 - 0.794, with class room 1 having the highest average F1_score and class room 4 having the lowest average F1_score. Therefore, given the noisy dataset, our algorithm is best at predicting class room 1 and worst at predicting class room 4.

3 Diagrams

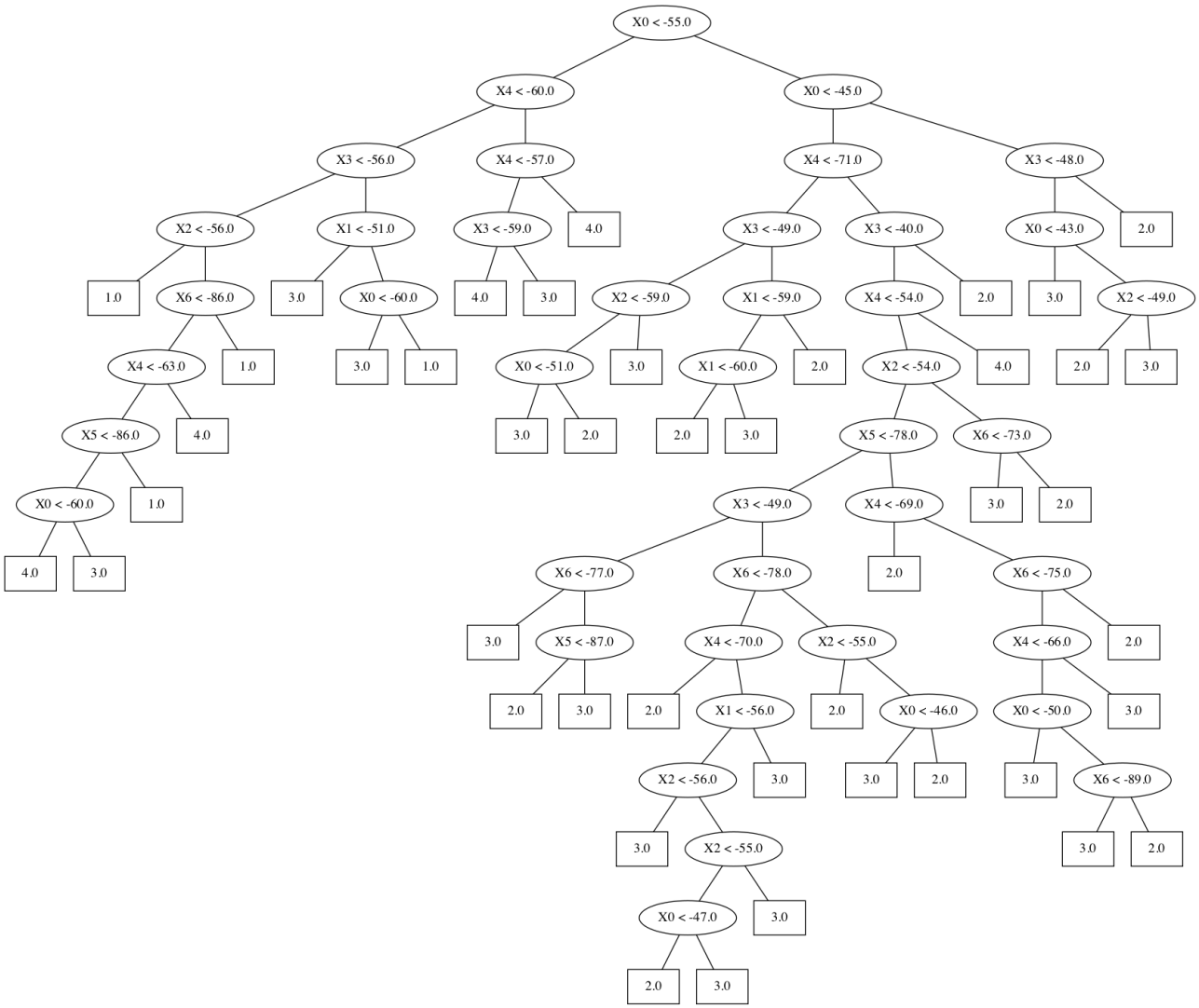

Figure 1: Tree trained on clean dataset (Unpruned)

4 Pruning and Evaluation

4.1 Implementation of pruning function

Pruning was implemented as a recursive function which follows the pseudo-code in Algorithm 1. This function works by visiting each node recursively. At each node, it checks whether its two children nodes are both leaf nodes or not. If both children nodes are leaf nodes, then the current node is a potential pruning node. Therefore, at this point, a decision must be made as to whether to prune or not, and if so, how to prune. To make this decision, we must evaluate the performance of the whole tree on a validation set for each of the three possible scenarios:

- Leave as is
- Substitute by left child
- Substitute by right child

The decision on which action to take is based on validation performance and therefore this is an instance of reduced error pruning. However, the way this evaluation is performed is not by evaluating the whole tree with each of the possible substitutions of the current node on the whole validation dataset, this would be costly and difficult. Instead, each time a new node is visited, the dataset is split, so at each recursive call, there is information about the relevant validation dataset resulting from performing all the splits up until that specific node. This way we can just evaluate each of the options as a sub-tree on the relevant validation set. Examples of the three candidate sub-trees are shown in Figure 2.

The overall output of this recursive function will return a tree which has undergone one level of pruning. However, it is entirely possible that on this resulting tree, pruning could also be applied and therefore, we prune the tree until convergence. The way we check for convergence is to introduce a counter *changes* for the amount of nodes pruned. The pruning function is then called within a loop until *changes* is zero. Zero changes will mean there are no more nodes in which pruning would result in reduced error in the validation set.

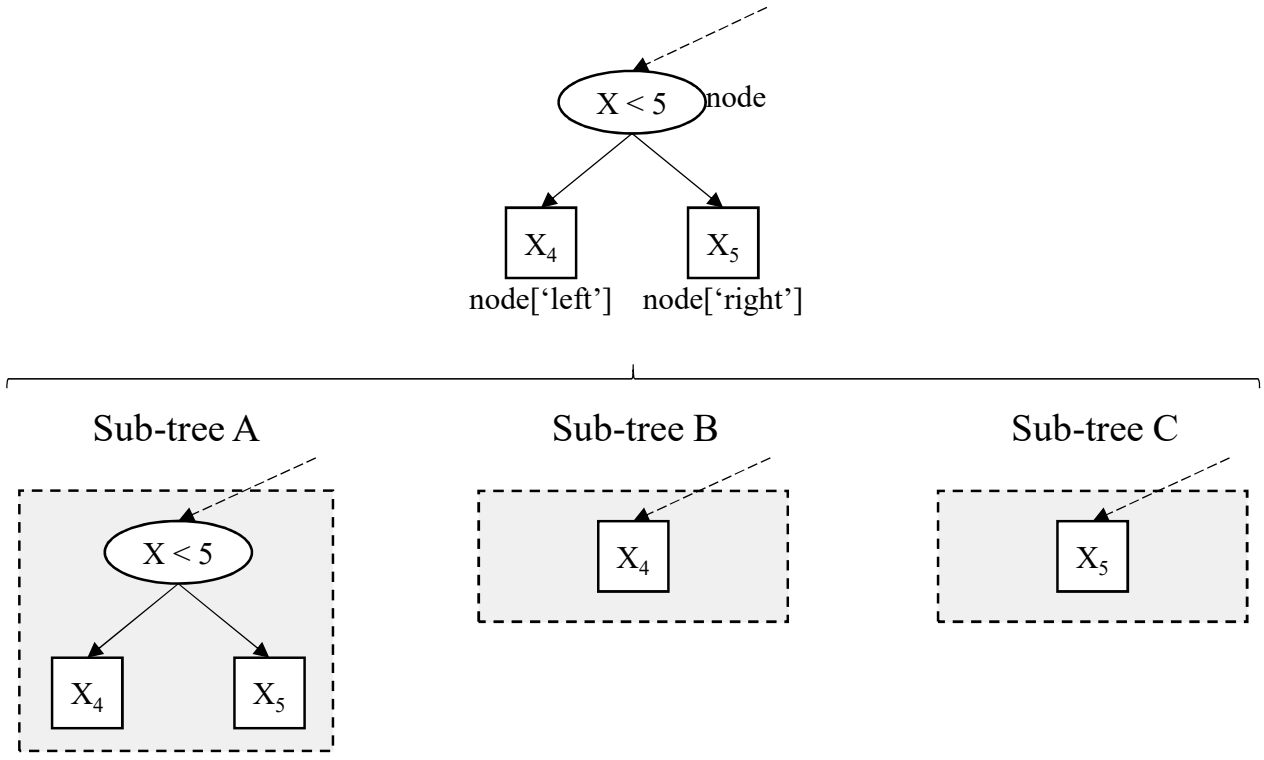

Figure 2: Example of a candidate node (above) and its respective potential sub-trees after pruning.

Algorithm 1 Decision Tree Pruning

```
procedure PruneTree(node, relevant_val_dataset, changes=0)
    if left child and right child are both leaf nodes then
        current_correct ← evaluate(relevant_val_dataset, node)
        left_correct ← evaluate(relevant_val_dataset, node['left'])
        right_correct ← evaluate(relevant_val_dataset, node['right'])
        if left_correct ≥ current_correct and left_correct ≥ right_correct then
            node ← node['leaf']
            changes++
        else if right_correct > current_correct and right_correct > left_correct then
            node ← node['right']
            changes++
    else
        node_split ← [node['attribute'], node['value']]
        l_dataset, r_dataset ← split_dataset(relevant_val_dataset, node_split)
        if node['left']['leaf']is 0 then
            node['left'], changes ← prune_tree(node['left'], l_dataset, changes)
        if node['right']['leaf']is 0 then
            node['right'], changes ← prune_tree(node['right'], r_dataset, changes)
    return node, changes
```

4.2 Evaluation - pruning on accuracy of tree on both datasets

4.2.1 Confusion matrix

Table 5: Average Confusion Matrix (After Pruning) - Clean dataset

Class	1-predicted	2-predicted	3-predicted	4-predicted
1-actual	49.2	0.0	0.3	0.4
2-actual	0.0	47.8	2.2	0.0
3-actual	0.2	3.0	46.6	0.3
4-actual	0.3	0.0	0.7	48.9

Table 6: Average Confusion Matrix (After Pruning) - Noisy dataset

Class	1-predicted	2-predicted	3-predicted	4-predicted
1-actual	40.5	2.0	2.8	3.6
2-actual	2.4	41.8	3.5	2.0
3-actual	2.3	4.7	41.8	2.8
4-actual	3.2	2.1	2.3	42.1

4.2.2 Evaluation metrics (average)

Table 7: Classification report (After Pruning) - Clean dataset

Class	Avg Precision	Avg Recall	Avg F1 Score
1	0.990	0.985	0.988
2	0.943	0.958	0.949
3	0.935	0.933	0.933
4	0.983	0.980	0.981

Table 8: Classification report (After Pruning) - Noisy dataset

Class	Avg Precision	Avg Recall	Avg F1 Score
1	0.840	0.826	0.832
2	0.827	0.843	0.832
3	0.829	0.809	0.817
4	0.833	0.844	0.837

4.2.3 Performance before vs after pruning

Before pruning, the average classification rates are 96.95% and 79.10% for clean and noisy dataset respectively. After pruning, the average classification rate for the clean dataset remains similar at 96.28% whereas for the noisy dataset, the average classification rate has increased to 83.13%.

Through investigate deeper into the evaluation metrics for noisy dataset, we found that the average precision has increased across classes after pruning, with a new range of 0.827 - 0.840. Class 1 remains the highest average precision rate. The average recall rate after pruning has also increased across classes, with class 2 now having the highest average recall rate. F1_scores has also increased across classes afrter pruning, with both class 1 and 2 having the highest average F1_score.

Overall, we have seen a big improvement in our prediction results after pruning our tree that was trained on the noisy dataset.

5 Further Discussions

5.1 Difference in the overall performance and per class when using the clean and noisy datasets

As we can see from the produced metrics in section 2, the average accuracy for the clean data is 0.9695, while the equivalent one for the noisy dataset is 0.791. Therefore, we experience a big decline in one of the most important metrics of performance and the question that comes straight into mind is whether this difference is statistically significant. We also obtain the standard deviation of the two distributions which is essential to calculate the t-statistic and the corresponding p-value. Since we are facing different datasets, we run a two-sample T-test and we obtain a t-statistic of 17.6 and a corresponding infinitesimal p-value. Hence, we are confident enough that even in the 0.01 significance level we reject the null hypothesis (H0) that the average accuracy levels for the clean and noisy datasets are equal. The rest of performance metrics also decrease in the noisy dataset as compared to the clean one. Precision, recall and F1 scores are all significantly smaller in the noisy dataset for all classes.

After observing this difference in performance, we would like to explain the reason behind this finding. The presence of noise in the data turns classification problems into complicated problems for which accurate solutions are hard to obtain. Noise can cause problems such as creating a cluster of examples of one class within the domain of another class or deleting examples that might be critical for a class. Consequently, the boundaries of different classes might start overlapping making knowledge extraction more difficult and decreasing the accuracy of these models as compared to models trained on clean data. Hence, the performance of classifiers depends significantly on both the overall quality of the data and the robustness of the specific classifiers to noise.

5.2 Maximum depth on prediction accuracy

Table 9: Average maximum depth and accuracy by dataset (before and after pruning)

Dataset	**Before Pruning**		**After Pruning**	
	Average Maximum Depth	Average	Average Maximum Depth	Average
Clean	12.7	96.95%	9.1	96.28%
Noisy	19.1	79.10%	15.8	83.13%

As shown in Table 9, for both datasets we have that pruning reduces the depth of the trees on average by 3.6 and 3.3 levels for the clean and noisy datasets, respectively. The smaller depth for the clean dataset shows us that it is easier to categorise, whereas the noisy dataset produces much deeper trees and so the models are more complex. The clean dataset decision tree accuracy stayed roughly the same despite being pruned, whereas the noisy dataset decision tree accuracy increased by 4% after being pruned. This could be due to the decision tree being overfitted on the training data for the noisy dataset, so it would work well for the training data and validation set but not well on the unseen test data due to the noise in the data. This is due to the fact that we did not enforce any maximum depth limit so the tree grows unbounded and therefore, models the noise. However, we observe that the accuracy of the clean dataset has not changed much indicating that the tree was not overfitting the data before pruning. Therefore, the unpruned tree from the clean dataset would generalise well on unseen data.