CO395 - Machine Learning Coursework 1

Padmanaba Srinivasan
01191525

Keerthanen Ravichandran
01195170

Harrison Ankers
01211208

Tze Hei Tang
01221240

February 11, 2019

Contents

1 Construction of the Decision Tree

1.1 Brief overview of building the decision tree

The following describes algorithm that builds the decision tree given a dataset.

1. Begin by creating a root node, then split the dataset at the point which provides the maximum information gain.
2. Recursively build subtrees using the subsets, which form left and right branches on the root node.
3. When no further splitting is possible, form leaf nodes which contain room values.

1.2 How we selected the best attribute in each node

The attribute with the maximum information gain is used as the splitting point to form subsets. The information gain is calculated as follows:

$$G(q) = H(dataset) - (\frac{|subsetA|}{|dataset|}H(subsetA) + \frac{|subsetB|}{dataset}H(subsetB)) \tag{1}$$

Where H(p) is the entropy of the data set p, denoted by:

$$H(dataset) = -\sum_{k}^{N} d_k log_2(d_k) \tag{2}$$

where d_k denotes the k^{th} element in data set D with N elements

Hence, from the above expressions, a threshold value that separates the original data set into data set A and B is selected such that the gain expression is maximised.

In order to achieve the above, we iterate through a sorted version of the data set that corresponds to the current node, and we try to separate the data set into A and B by setting the threshold to $\frac{d_i+d_{i+1}}{2}$.

Now, the entropy of data set A and B, and the corresponding information gain are computed according to equations (1) and (2). The threshold that gives the highest information gain is selected, and the data set is split into subsets A and B, where:

$$A_i < threshold$$

$$B_j \geq threshold$$

The dataset A now corresponds to the left child node, and data set B to the right child node. This process is repeated recursively.

Eventually, a leaf has to be created at the end of a branch; there are three conditions in which a leaf is created.

1. When either A or B is an empty set, the modal room value original data set is used as the final value of the leaf.

2. When the all of the attribute values in the original dataset is the same, the modal room number is used as the value of the leaf.
3. When all of the room values in a dataset are the same, further splitting cannot occur so a leaf node is created with the room number.

2 10 Fold Cross Validation

2.1 Why Cross Validation?

Cross Validation is a technique employed to produce and evaluate the best possible model when the number of samples is low. The dataset is separated into k folds with one fold used as a test set, another as a validation set and the remaining $k-2$ as training data. Repeating this with all possible combinations of training and validation sets will produce $k(k-1)$ tests overall. Evaluation of each model can be performed on the confusion matrix it yields when tested.

In this project, $k = 10$ and 90 trees are trained.

2.2 Cross Validation Implementation

Given the entire dataset and k=10, the size of each fold can be calculated. Following this the dataset is shuffled once and then a fold removed to form the test set. Of the remaining $k-1$ folds, another fold forms the validation set with the residual data making up the training data. This is implemented by having two loops, with the outer loops separating out the test set and the inner loop generating the training and validation set. As a result, every fold gets the opportunity to be test set validation set once each.

After a tree is trained on the training data, its' error rate on test set is found and the confusion matrix analysed (to calculate recall and precision rates as well as F1 values) before the tree is pruned and the new confusion matrix analysed again. Having two sets of confusion matrices both before and after pruning is a useful tool in comparing the effectiveness of pruning.

This procedure is repeated to train 90 trees and test and evaluate 180 trees overall.

3 Questions

3.1 Pruning

3.1.1 Pruning Implementation

Pruning is implemented via the *Pruning* class in *lib* which is initialised with a validation set and once built, the root of the tree is passed in for automatic Pruning.

Pruning involves firstly testing the validation set on the tree and obtaining an error rate. Following this the $prunify$ function is called which recursively finds nodes which is the parent of two leaves directly. At this stage there are two possible paths: firstly to make the node a leaf or secondly to keep the decision node as is. The obtains the leaf values from both child nodes and calculates the error rate if the decision node were a leaf with each value. If error rate improves then the best improvement

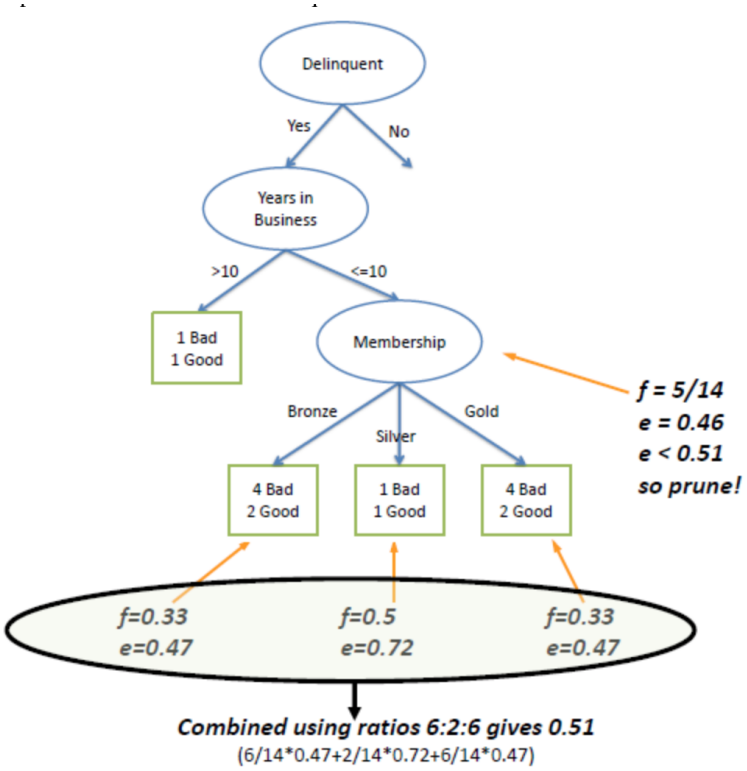

Figure 1: An illustration of pruning with Error-Estimation.

is chosen, the decision node converted into a leaf with this value and the best error rate updated. If pruning does not provide an improvement then the decision node is kept as is.

Another method of pruning is to use Error-Estimation (EE) [1], commonly used with the C4.5 algorithm. This method takes a more statistical approach and is similar in structure to the standard algorithm. With EE the dataset itself ́travelsd́own the tree, splitting on each decision node. When a set reaches a leaf, the leaf calculates the actual error rate a well as the estimated error rate with the formula in figure 3. Both these error rates are returned. Following this the decision node calculates the weighted sum of the error rate of its' leaves to find the node error, and the weighted sum of the estimated error to find the subtree error. Pruning occurs if the node error is larger than the subtree error. This repeats recursively úpt́he tree. An illustration of this is in figure 1. EE is implemented within the Node class and is called upon as a member function, passing in the validation set as a parameter.

$$e = \frac{f + \frac{z^2}{2N} + z\sqrt{\frac{f}{N} + \frac{f^2}{N} + \frac{z^2}{4N^2}}}{1 + \frac{z^2}{N}} \tag{3}$$

Testing both methods of pruning with the noisy dataset showed clear advantages to EE; from a high level the standard method for pruning improved noisy dataset classification rate by around 1% overall, whereas pruning using EE has a more significant effect, typically improving classification rate by 2-5%.

Looking at the number of nodes after pruning with each method yields some insights. Figure 2 shows

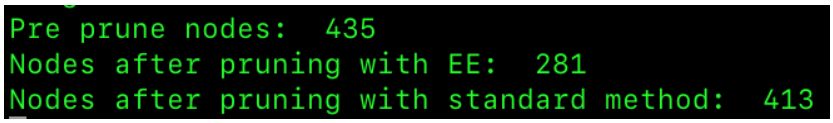

Figure 2: Comparing the number of nodes with different pruning methods.

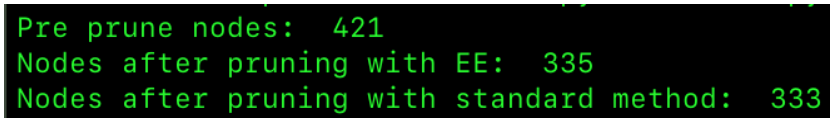

Figure 3: Pruning with EE followed by the standard method.

number of nodes before and after pruning with each method. The trees provided to both prune algorithms were trained on the same training set and so were identical.

Combining these methods, pruning with EE first followed by the standard method yields metrics in figure 3 and then reversing the order, pruning vice-verse yields figure 4. In both cases, the datasets (all derived from the noisy dataset) used were the same and so the initial decision tree to was the same. In this example using the standard pruning method after the aggressive EE pruning yields a slightly smaller tree. Over several different datasets, the second method yielded trees of the same size or slightly larger than trees produced by the first method. Correspondingly, the classification rates were the same or slightly larger for the first method overall.

Using combined pruning algorithms yielded slightly higher classification rates than using any one algorithm on its own and showed a clear inverse link between tree size and classification rate. With any pruning algorithm, trees derived from the clean dataset little change in the number of nodes or classification rate.

3.1.2 Pruning's effects on Classification Rate

Observing the classification rate for a number of folds on the clean dataset reveals little change in classification rate, with classification rate typically changing by less than 0.5% post pruning. Decreases in the classification rate are more common and can be expected as the purpose of pruning is to reduce overfitting. The clean dataset is so because it doesn't overfit much to begin with and whatever pruning occurs changes classification rate minimally, with classification rate hovering around 97% before and after pruning. This seems feasible given the observation that the size of the tree remains the same after pruning despite the methods.

The noisy dataset however, tells a very different story. Pre-pruning, the tree's accuracy hovers around 80-81%. Pruning with the standard method only improves classification rate from 1-2% boosting it up to 81-82%. Using only EE typically boosts classification rate to around 84% and combining methods yields the same improvement. The small difference in using only EE and combining methods

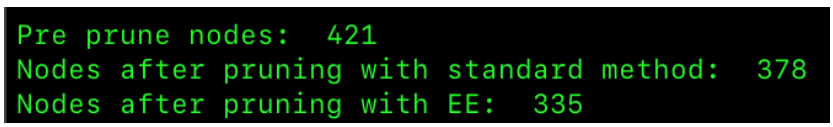

Figure 4: Pruning with the standard method followed by EE.

is explained by EE aggressively pruning most of the tree leaving the standard method to prune very little.

Overall, pruning has little effect on classification rate in the clean dataset which is evidenced by little to no reduction in the number of nodes in the tree, with there being around 60 nodes on average and pruning reducing this by around 20%. In some, rare, extreme cases the number of nodes decreased by 50%, although classification rate remained constant.

With the noisy dataset the number of nodes drops from around 400-450 to around 300-330, roughly a 25% reduction in tree size overall. This suggests rampant overfitting on the noisy dataset which is alleviated by pruning but even still is in stark contrast to training on clean data. Also, noticing the difference in sheer number of nodes in trees between the clean and noisy datasets despite training data being of same size, there seems to be a link between the noisiness of the data and the degree to which it is overfitted. This logically leads onto the question of whether data can be cleaned before it is used and whether an indicator of noisy data could be related to the size of the decision tree it produces.

3.2 Metrics in Clean and Noisy Datasets

3.2.1 Metrics

Various metrics can be obtained from the tests to assess the performance of the tree. Specifically performance can also be analysed by looking at metrics such as classification rate, recall, and F1 measure.

3.2.2 Performance

The classification rate displays the percentage of correctly identified elements as a proportion of the whole test set. The pre prune, clean and noisy datasets produced the confusion matrices in tables 1 and 2 respectively. The metrics derived from these matrices are in table 3.

Recall rate indicates how many of the actual positive results are classified as positive by the model which for the clean data is very high. Similarly, precision rate - which is a measure of the correct positive predictions as a proportion of total positive predictions - too is high. Correspondingly, the F1 rate too is high. These indicate a high level confidence that the model will correctly classify inputs. The fact the F1 is so close to 1 for both pre prune and post prune indicates that this tree is close of perfect precision and recall.

Post pruning, for the model trained on clean data the recall and precision rates change very little indicating that this method of pruning cannot improve prediction accuracy much further.

The metrics for the noisy dataset however, tell a different story. Both precision rate and recall rate are lower indicating a more spread out confusion matrix where fewer positive predictions and correct and fewer positive values are classified as being positive. Post pruning the metrics increase by around 4.5% all round, with the increase in F1 showing that the model is getting closer to perfect precision and recall.

3.2.3 Pruning performance

Pruning optimises the tree via pruning off leaves that cause overfitting on a validation set, following which the pruned model is tested on the test data once more. It is expected that model performs better on the test data after pruning.

	Room 1 (Actual)	**Room 2 (Actual)**	**Room 3 (Actual)**	**Room 4 (Actual)**
Room 1 (Predicted)	49.4	0	0.2	0.5
Room 2 (Predicted)	0	47.6	1.6	0
Room 3 (Predicted)	0.4	2.4	48.1	0.2
Room 4 (Predicted)	0.2	0	0.1	49.3

Table 1: Confusion Matrix (clean data pre-prune)

	Room 1 (Actual)	**Room 2 (Actual)**	**Room 3 (Actual)**	**Room 4 (Actual)**
Room 1 (Predicted)	38.7	3.3	2.8	3.5
Room 2 (Predicted)	3.7	40	3.9	2.1
Room 3 (Predicted)	3.2	3.5	41.5	3.8
Room 4 (Predicted)	3.4	2.9	3.3	40.4

Table 2: Confusion Matrix (noisy data pre-prune)

When tested with clean data the classification rate was 97.1%. Furthermore, when tested with noisy data the rate again lowered to 84.85%. The clean data classification matrix only slightly increased, though, the noisy data had a larger increase. Both clean and noisy data produced the confusion matrices in tables 1 and 2 respectively. Post prune tables are in tables 4 and 5. The clean data produced an average recall of 97.12% and the noisy produced a recall of 84.86%. The clean data showed almost no improvement, however, the noisy data had a more dramatic increase. The clean data produced an average F1 of 97.11% and the noisy produced a F1 of 84.86%. The F1 metric showed a slight decrease in performance for the clean data, however, the noisy data improved again.

On the clean dataset, pre pruning classification rate was 97.2% and fell slightly to 97.1% post pruning. This can be attributed to the data being pruned on the validation set, which by extension reduces overfitting on the test data and attempts to fit the model more to the validation set. Over several repetitions there was no consistent trend in post prune classification rate, sometimes being lower and sometimes higher.

The noisy dataset however, yields some important insights into the effectiveness of pruning. Pre prune classification rate stood at 80.3% which increased to 84.85% post prune. This is a significant improvement and again over several repetitions pruning improved classification rate by 3-4%.

The confusion matrices also hold other important information. In the clean data pre prune matrix (table 1) notice that Room 2 and Room 3 are much more likely to be confused for the each other and observing their metrics in table 6 confirm this with both these rooms having slightly lower precision and recall rates. The situation doesn't change much with these two rooms still being confused for one another post pruning in table 7. This ambiguity could be solved by using training Random Forests and taking the modal result when testing on a sample.

	Clean Data		Noisy Data	
	Pre-pruning	Post-pruning	Pre-pruning	Post-pruning
Avg. Precision	97.20%	97.10%	80.30%	84.85%
Avg. Recall	97.22%	97.12%	80.30%	84.86%
Avg. F1	97.20%	97.11%	80.30%	84.86%

Table 3: Metrics from confusion matrices

	Room 1 (Actual)	**Room 2 (Actual)**	**Room 3 (Actual)**	**Room 4 (Actual)**
Room 1 (Predicted)	49.5	0	0.2	0.5
Room 2 (Predicted)	0	47.7	2	0
Room 3 (Predicted)	0.5	23.0	47.7	0.2
Room 4 (Predicted)	0	0	0.1	49.3

Table 4: Confusion Matrix (clean data post-prune)

	Room 1 (Actual)	**Room 2 (Actual)**	**Room 3 (Actual)**	**Room 4 (Actual)**
Room 1 (Predicted)	42.6	2.9	2.6	2.5
Room 2 (Predicted)	1.6	41.3	3.9	1.8
Room 3 (Predicted)	2.5	3.6	42.6	2.3
Room 4 (Predicted)	2.3	1.9	2.4	43.2

Table 5: Confusion Matrix (noisy data post-prune)

Metrics	**Room 1**	**Room 2**	**Room 3**	**Room 4**
Recall rate	0.986	0.967	0.941	0.994
Precision rate	0.988	0.952	0.962	0.986
F1 rate	0.987	0.960	0.952	0.990

Table 6: Metrics for clean, pre-prune

Metrics	**Room 1**	**Room 2**	**Room 3**	**Room 4**
Recall rate	0.986	0.960	0.941	0.998
Precision rate	0.99	0.954	0.954	0.986
F1 rate	0.988	0.957	0.947	0.992

Table 7: Metrics for clean, post-prune

Comparing the metrics for the noisy dataset both pre prune and post prune (tables 8 and 9 respectively) show a significant increase across all classes, which is confirmed by observing the pre prune and post prune confusion matrices. Rooms 1 and 4 shows the largest gain in the metrics and recall and precision rates for all classes are roughly the same suggesting that all classes are impacted equally by the noisy data.

3.3 Depth and Prediction Accuracy

Looking at the metrics for the noisy dataset, both pre and post prune observe that for all classes the recall, precision and F1 measures increase. This immediately signifies an increase in prediction accuracy. Since pruning has removed nodes and reduced the depth of the tree an relationship exists between reduced depth and increasing prediction accuracy.

The clean dataset sheds further light on this; the prediction accuracy decreases slightly between after pruning despite depth staying the same. As a result a relationship could exist between the extent of pruning and improvements in prediction accuracy. In other words prediction accuracy will increase only if further pruning causes a reduction in depth.

Table 3.2.3 shows that the depth of the decision tree increases with noisy data. In the clean dataset pruning has little effect on depth and suggests that a clean dataset can be identified by little change in depth of the tree when pruned.

By contrast, trees trained on noisy data see a circa 20% reduction in depth. This further bolsters the theory that noisy datasets correlate with the extent of tree depth reduction. Observing the drop in tree depth and the corresponding increase in classification rate, in general shorter trees imply a higher classification rate as pruning reduces overfitting in the tree, thereby generalising the paths within the tree.

Metrics	**Room 1**	**Room 2**	**Room 3**	**Room 4**
Recall rate	0.801	0.805	0.798	0.808
Precision rate	0.790	0.805	0.805	0.811
F1 rate	0.795	0.804	0.802	0.810

Table 8: Metrics for noisy, pre-prune

Noisy post prune	**Room 1**	**Room 2**	**Room 3**	**Room 4**
Recall rate	0.842	0.850	0.835	0.867
Precision rate	0.870	0.831	0.827	0.867
F1 rate	0.855	0.840	0.83	0.867

Table 9: Metrics for noisy, post-prune

Datasets	**Depth**
Clean (Pre-prune)	8
Clean (Post-prune)	8
Noisy (Pre-prune)	16
Noisy (Post-prune)	13

Table 10: Average depths for trees trained on both datasets, pre and post prune.

4 Printing the tree

The *PrettyPrint* module uses the $forest$ package in LaTeX to plot a decision tree object. The trees are printed via a recursive function which is then compiled to a PDF.
Figures 5 and 6 are the trees trained on the clean and noisy datasets.
Drawings of both trees have also been included as separate PDF's for clarity.

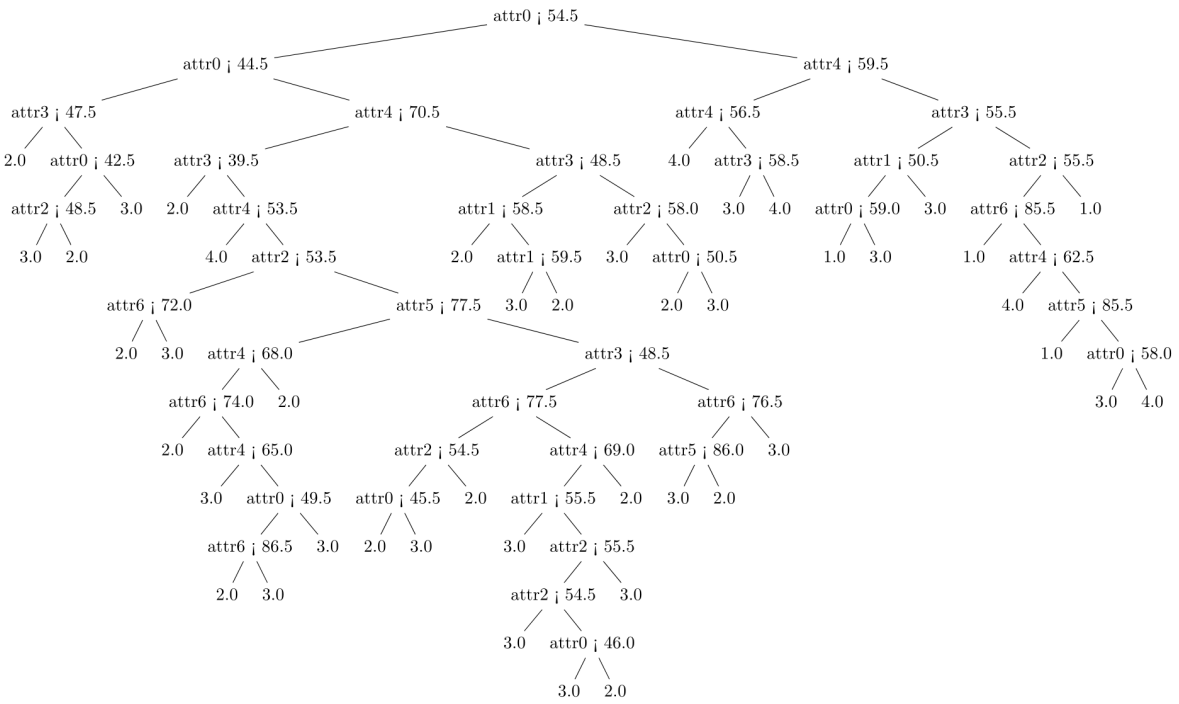

Figure 5: Clean decision tree.

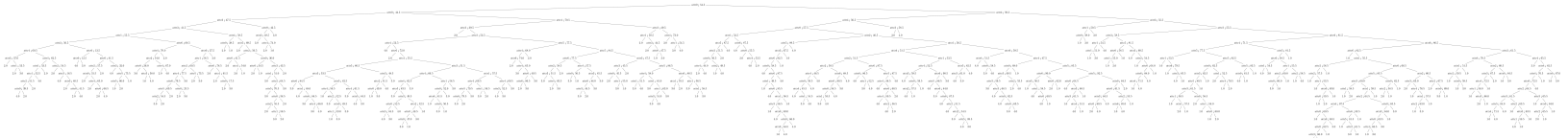

Figure 6: Noisy decision tree.

5 Bibliography

References

[1] Dr Saed Sayad, Overfitting. Accessed 25/01/2019
http://www.saedsayad.com/decision_tree_overfitting.htm