CO395 - Introduction to Machine Learning
Decision Trees

Amin-Nejad Ali, Gan Ding Han, Liu Lingjun, Yoon Goonhu
{aa5118, dhg18, ll5018, gy18}@imperial.ac.uk

Course: CO395, Imperial College London

9th February, 2019

1 Introduction

A Decision tree is a predictive model consisting of nodes, links and leaves, where each node represents a feature, each link represents a rule and each leaf represents an outcome. It can be used for both classification and regression related tasks creating a tree for the entire data which produces a single outcome at every leaf. During training, data is divided into smaller and smaller subsets by ensuring the information gain is maximised from a certain feature value until all target variables are under one category.

1.1 Problem Overview

There are 4 rooms in the flat and 7 WiFi signal whose strengths vary based on where users are. Each column in the given data shows 7 values for its WiFi signal strength and the corresponding room label. In terms of datasets, two are provided (clean dataset and noisy dataset). Our task is to create, train, test and evaluate a decision tree, using these two different datasets and appropriate techniques such as pruning and visualisations such as confusion matrices. The decision tree needs to predict the room in which users are, based on WiFi signal values, without over-fitting or under-fitting.

1.2 Why Decision Tree is used

There are a few reasons why a decision tree is appropriate for this task.
In short, this is a supervised learning classification task where the algorithm is required to predict a discrete value, not a continuous value, without the necessity of any distributional assumptions about the data. A decision tree is particularly powerful and suited for this sort of problem as it assigns a data point to one of only a few possible classes. Moreover, the final output of a decision tree could be easily visualised and interpreted as decision rules, which makes humans understand the whole process relatively easily. Thus, decision trees can be more informative and tailored for solving this problem than other methods.

2 Implementation Details

The entire process of performing k-fold cross validation for training and evaluating the decision tree could be modularised and split into 4 distinct components: training, evaluating, pruning and cross validating. These 4 components were each implemented separately in their own Python file, making API calls to one another where necessary. The following sub-sections will therefore briefly summarise each of these 4 components.

2.1 Training

The class *Node* is used to store data corresponding to each node (and leaf) in the decision tree. If the node is non-leaf, then the variables *splitAttribute* and *splitValue* would contain data while *terminalValue* would be None. The reverse is true for leaves.

The main function *decision_tree_learning* trains the decision tree using the supplied dataset as prescribed in the specification file. The function first finds the split in the dataset that results in the maximum information gain and stores information on this split in a *Node*. Specifically, the split comprises the attribute and its value, and the function *find_split* was implemented as follows:

```
01: for each attribute in the dataset:
02:     sort the data in ascending order according to that attribute
03:     for each row in the dataset:
04:         if the labels in this row and the previous row are different then:
05:             sLeft ← dataset[0:row, -1]
06:             sRight → dataset[row:, -1]
07:             calculate the information gain
08:             update the maximum information gain accordingly
09: return the splitAttribute, splitValue
```

The function then calls itself recursively to do the same in order to produce the left and right nodes. At the end, *decision_tree_learning* returns the root node of the trained decision tree.

2.2 Evaluation

The evaluation function takes the root node of a trained decision tree and the test dataset, and produces a set of metrics based on how well the tree performed on the test dataset. The following are the metrics generated by the *evaluate* function.

1. The confusion matrix is a 4x4 Numpy array (prediction x ground truth) representing the predicted and true labels of the decision tree.
2. The label dictionary (*labelDict*) keeps track of the recall, precision and F1 values for each and every possible label in the test dataset. The keys of the dictionary are the names of the labels, while the values are dictionaries representing the aforementioned metrics for that specific label.
3. Accuracy is a floating point value representing the average classification rate.

2.3 Pruning

The *prune_tree* function takes as input the root node of the original decision tree, the original accuracy of that tree on the validation dataset, and also the validation dataset itself. The function then recursively traverses the tree to identify nodes connected to 2 leaves. If pruning such a node (turning it into one of its leaves) results in a better accuracy than the original accuracy, then that node is pruned and the original accuracy is updated to the improved accuracy. The pseudo-code is given in brief below.

```
01: originalAccuracy ← evaluate(root)
02: while True:
03:     nodesToPrune ← False
04:     root, didPrune ← traverse_tree(root, originalAccuracy)
05:     if not didPrune:
06:         break
```

```
01: def traverse_tree(node, originalAccuracy):
02:     if node is a leaf then:
```

```
        return
    else if node has 2 leaves and accuracyAfterPruning ≥ originalAccuracy  then:
        convert node to the leaf with greater accuracy
        originalAccuracy ← accuracyAfterPruning
        return
    else:
        traverse_tree(node.left, originalAccuracy)
        traverse_tree(node.right, originalAccuracy)
```

A few design considerations were taken into account when creating the algorithm for the pruning.

- The pruning was done top-down as opposed to bottom-up because of how the object *Node* was structured: each *Node* has a left and right branch pointing to another *Node* objects down the tree.
- If pruning a node would result in a new accuracy that is greater than or equal to (as opposed to strictly greater than) the original accuracy, we would prune that node. This is because we can allow the tree to generalise better without compromising on the test accuracy.

2.4 Cross Validation

In the cross validation phase, the dataset is firstly split into training, test and validation subsets. Upon analysing the skew within both noisy and clean datasets, it was decided to take a random sampling approach each time the algorithm is run as opposed to a stratified sampling approach whereby the data subsets are ensured to all have a similar distribution of room labels. This was due to the fact that each room label is evenly represented (25% each) and there are a large number of observations (2000) for both datasets.

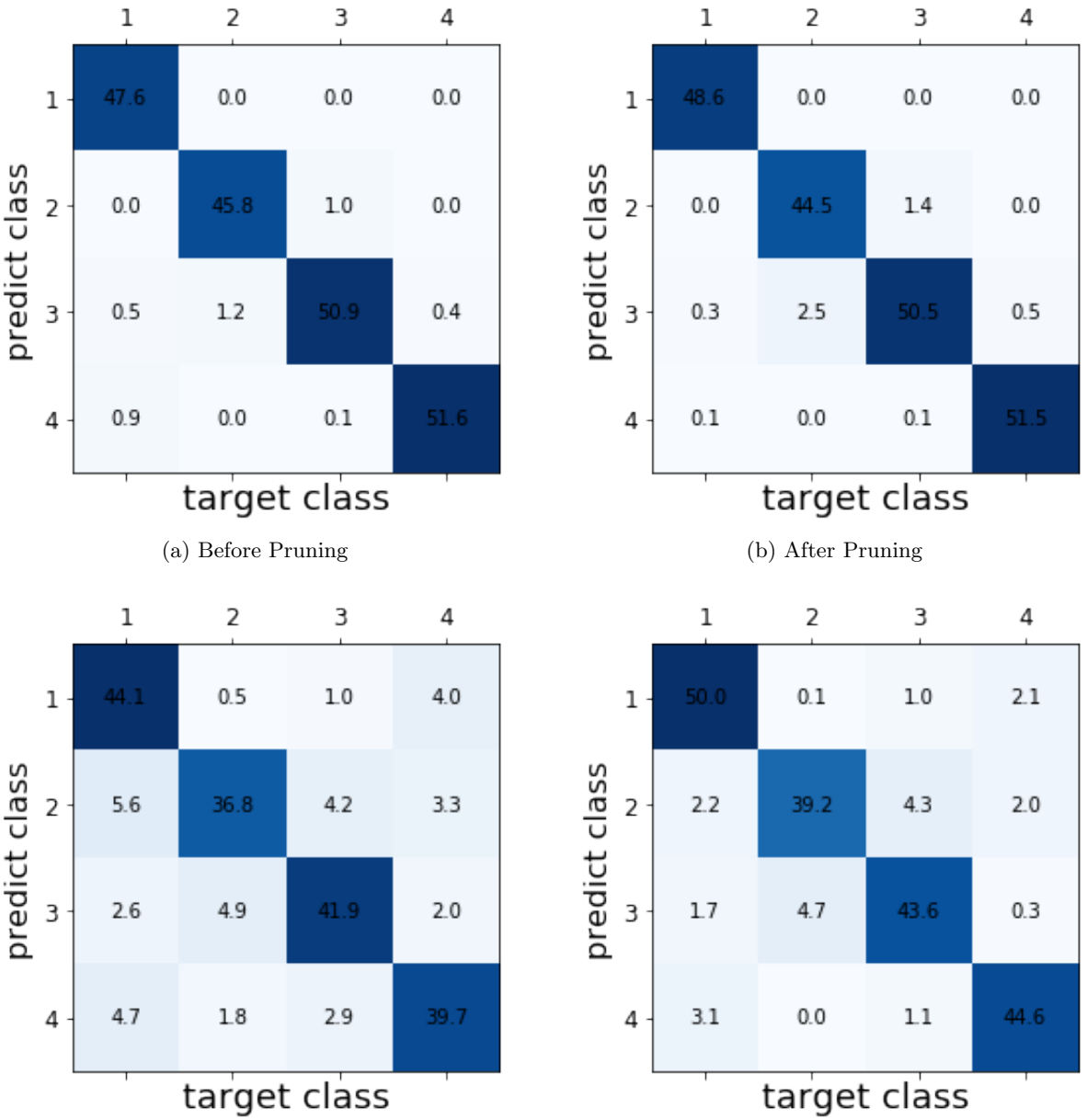
Once the datset has been split, the function trains, prunes (optional) and then evaluates the average of the metrics. The pseudo-code is given in brief below.

```
shuffle the dataset
if prune is True, then:
    set aside 10% of the dataset to be the test set, and the remainder for training and validation
for each iteration in kFold:
    from the remaining dataset, split it into training (90%) and validation (10%) sets
    train the tree using the training dataset
    if prune is True, then:
        prune the tree using the validation set
        metrics ← evaluate the pruned tree using the test set
    else:
        metrics ← evaluate the tree tree using the validation set
    metricsList ← metrics
calculate each metric average (accuracy, confusion matrix, label dictionary) in metricsList
```

3 Evaluation

3.1 Confusion Matrices

A confusion matrix is a table that describes the performance of a typical classification model on a set of test samples. In our model, the test samples are set to be 10% of the whole dataset (noisy or clean) after shuffling. For the full tree prior to pruning, the confusion matrix is calculated using the test data, which is presented in Figure 1 (a) and (c) for the clean and noisy datasets respectively. These trees are then pruned on the validation data, before the test data is utilised again to calculate the confusion matrices post-pruning as presented in Figure 1 (b) and (d), again for the clean and noisy



(a) Before Pruning

(b) After Pruning

(c) Before Pruning

(d) After Pruning

Figure 1: Average confusion matrix for cross validation classification results on: a,b) clean dataset, and c,d) noisy dataset

datasets respectively. In order to obtain the average confusion matrix, we are using the mean results from all the confusion matrices produced during 10-fold cross validation.

It is also worth mentioning that, for a model being deployed in production, the trade-off between training on more samples and pruning for a less complex tree should be taken into account. This is due to the fact that a decision tree which will not be pruned does not require separate validation data to be taken out of the whole dataset to assess the pruning procedure meaning that the model can be trained with more samples, potentially improving accuracy.

Based on the confusion matrices presented above, the average statistical measures of *recall*, *precision* and *F1* are calculated for each individual label.

3.2 Dataset comparison

As demonstrated in Table 2, for the noisy dataset, the recall, precision and F1 score are all significantly improved post-pruning. However, as Table 1 shows, these metrics are worsened for the clean dataset.

Table 1: Average statistics for individual labels on clean dataset

Before Pruning (acc. 97.95%)

Label	Recall	Precision	F1
1	0.9714	1.0	0.9854
2	0.9745	0.9789	0.9766
3	0.9788	0.9604	0.9695
4	0.9923	0.9811	0.9866

After Pruning (acc. 97.55%)

Label	Recall	Precision	F1
1	0.9918	1.0	0.9959
2	0.9468	0.9703	0.9579
3	0.9712	0.9402	0.9549
4	0.9904	0.9962	0.9932

Table 2: Average statistics for individual labels on noisy dataset

Before Pruning (acc. 81.25%)

Label	Recall	Precision	F1
1	0.7737	0.8903	0.8269
2	0.8364	0.7408	0.7805
3	0.8380	0.8175	0.8266
4	0.8102	0.8105	0.8092

After Pruning (acc. 88.70%)

Label	Recall	Precision	F1
1	0.8772	0.9401	0.9072
2	0.8909	0.8226	0.8551
3	0.8320	0.8675	0.8692
4	0.9102	0.9140	0.9121

We reason that for the noisy dataset, through performing pruning, we are reducing the size and the complexity of the decision tree thus providing a defence against *overfitting* to the training data and improving the predictive ability of decision tree on the whole. However, with the clean dataset, since there is minimal noise and the data is a true reflection of reality, pruning actually results in a slight loss of some important node-information, making the model *underfit* the data (i.e. lose critical information from the training data and instead overfit the validation data).

Recall is formally defined as the proportion of relevant instances that have been retrieved over the total number of relevant instances. As shown in Table 2, for the noisy dataset, **room 4** gets the highest recall after pruning, in which most of the WiFi signals are correctly classified by our decision tree. For the clean dataset, this is again **room 4** prior to pruning but recall actually falls slightly leaving **room 1** with the highest recall.

Precision is formally defined as the proportion of relevant retrieved instances over the total retrieved instances. In the noisy dataset, the precision for classifying **room 1** is the highest for both a full and pruned tree. For the clean dataset, this is precision is again highest for **room 1** which receives a perfect score of 1.0 both before and after pruning i.e. there are no false positives.

The F1 score is an indicator of the balance between recall and precision, which also shows improvement on noisy dataset after pruning in Table 2. However, as demonstrated in Table 1, the F1 score declines if a clean tree is pruned as it underfits the data.

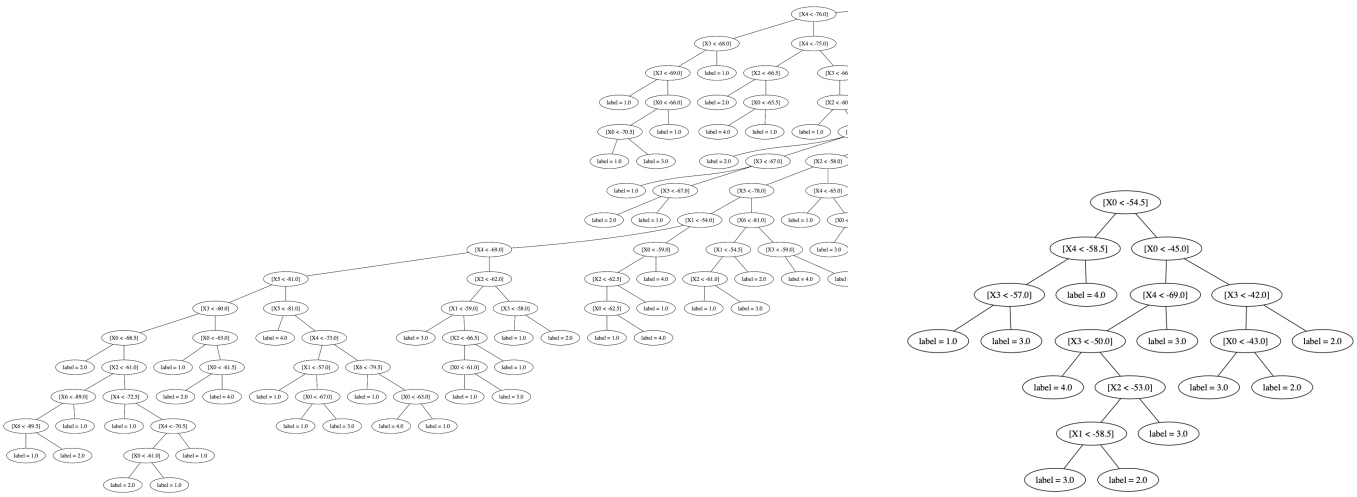
3.3 Pruning comparison

Pruning is a common strategy for reducing the tree size to avoid overfitting. In order to demonstrate the tree complexity reduction after pruning, the decision tree is visualised in Figure 2.

Specifically, the average size of a full tree trained on the noisy dataset has a depth of 19 to 21, and with around 430 to 460 nodes. After pruning, the size is reduced to a depth of roughly 7 to 10, with 30 to 50 nodes. For the clean dataset, the full tree has a depth of 9 to 13 with around 60 to 70 nodes. After pruning, the depth reduces to roughly 5 to 9 with approximately 10 to 30 nodes. The clean tree requires fewer nodes and less depth due to the lack of noise in the dataset i.e. less complexity to model.

For the noisy dataset, irrelevant information can be pruned away after training when evaluating against the validation dataset. Therefore, the generalisation ability is enhanced and predictive ability is improved.

However, if pruning is performed on a tree trained with the clean dataset, some of the nodes containing important classification information will be mistakenly pruned causing an unnecessary



(a) Before Pruning (b) After Pruning

Figure 2: The visualisation of the decision tree on noisy dataset before and after pruning

reduction in tree complexity. This does still improve the generalisation of the model but at the cost of reduced accuracy on the unseen test data because it is clean. This is confirmed by testing the pruned clean dataset tree on the noisy dataset which actually results in a higher accuracy than using a tree pruned on the noisy dataset as depicted in Table 3.

Table 3: Average statistics for individual labels of clean tree on noisy dataset

After Pruning (acc. 89.14%)

Label	Recall	Precision	F1
1	0.8717	0.8875	0.8791
2	0.9108	0.8812	0.8957
3	0.8960	0.9043	0.9001
4	0.8881	0.8945	0.8910

3.4 Depth

As mentioned above, maximal tree depth for the noisy dataset goes is 20 before reducing to 10 after pruning. For the clean dataset, the corresponding numbers are 11 and 7 respectively.

The depth of a tree is essentially a proxy for its ability to model complex functions. The noise within the noisy dataset embodies this complexity and explains why a greater tree depth is required to achieve a similar level of accuracy as the clean dataset. Increasing tree depth increases the classification performance on the training set but at the risk of overfitting the data. Therefore generalisation performance may suffer if depth is increased too far and this is in fact exactly what is seen above with the pruning experiments on the noisy dataset.

Generally, the relationship between maximal tree depth and accuracy is positive until a certain point where it has encoded all the critical relationships between the data points. After this point, accuracy begins to plateau and then gradually fall as the tree begins to incorporate the inherent noise within the training data into its branches i.e. overfit the data. However, this is not what is seen with the clean dataset because there is no noise.

4 Conclusion

In summary, we have demonstrated how to model a decision tree and the effects of noise on the dataset as well as the pruning process. We have also demonstrated sometimes even a simple decision tree can achieve very high classification accuracy; neural networks are not always needed for all tasks if the relationship is simple enough.

In terms of results, we show that the highest accuracy is generally achieved for classifying rooms 1 and 4. Given that room 2 is the only room with a WiFi router inside (all other 6 routers are on the outside of the flat), we would have expected room 2 to have the highest accuracy and so this result surprises us.

Lastly, we would like to highlight some of the limitations of decision trees and propose the use of *random forests* to combat these limitations. Decision trees are much more prone to bias and variance in the dataset, hence the lower performance on the noisy dataset vs the clean dataset (90% accuracy vs 97%). Random forest is the term given to a large collection of decision trees whose results are aggregated into one final result. These decision trees are all created on randomly different samples of data and features enabling them to limit any bias and variance in the dataset. When all these trees are combined, their ability to limit overfitting without substantially increasing error due to bias is why they are such powerful and popular models.