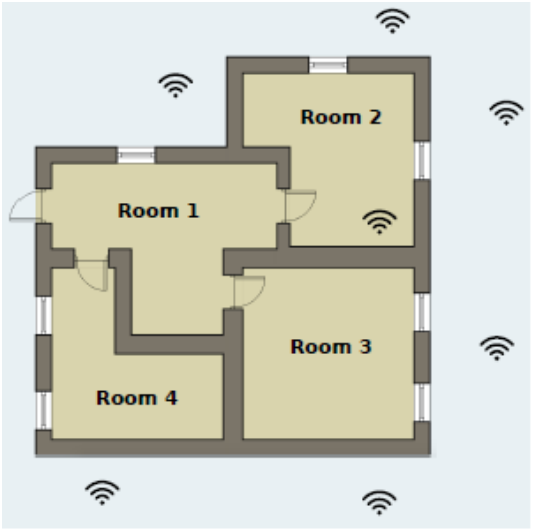
# Introduction to Machine Learning: Coursework 1
# Decision Tree Learning

Hanna Behnke, Daria Galkina, Doruk Taneli, Alexander Reichenbach

November 16, 2020




## Contents

# 1 Implementation Details

## 1.1 Implementation of the Tree and Learning Methods

The decision_tree_learning() function serves the purpose of training the tree model and is based on the pseudo-code given in the coursework specification. It takes a matrix containing the data-set and a depth variable as arguments. For the structure of the tree, we followed the recommended approach and used a dictionary for each node. We did not include a 'leaf' attribute in the node dictionary to save space; as we can find out if a node is a leaf in constant time, checking if the node has right and left children. The structure of the decision tree is presented in Figure 1.

Inside the decision_tree_learning() function, the find_split() function is called. It takes the provided data-set and returns a dictionary that contains the information about the best split:

- **The attribute** used in to choose the split value
- **The value** based on which the split was made
- **The left child data** that was selected if the attribute value was smaller than the split value
- **The right child data** that was selected if the attribute value was equal to or greater than the split value.

```
"""
Decision Tree Model Structure:
The root node will be passed to functions as the "model"

Non-leaf node structure:
  {'attribute': attribute for split,
   'value':     value of the attribute to split,
   'left':      left branch from this node,
   'right':     right branch from this node}

Leaf node structure:
  {'attribute': attribute for split,
   'value':     value of the attribute to split,
   'left':      left branch from this node,
   'right':     right branch from this node,
   'label':     predicted label for data that reaches this node}

"""
```

Figure 1: Decision Tree Structure

In order to maximise the information gain with each decision in the tree, the find_split() function loops through all unique attribute values and evaluates possible splits based on each of them. If the information gain improves with the provided split, the values of the dictionary are overwritten with the current ones, i.e. the given node is updated. The loop continues until all unique values for all attributes have been tried out. The final dictionary (i.e. node) containing the best split information is then returned to the decision_tree_learning() function.

The decision_tree_learning() is a recursive function. It carries on creating new nodes, based on the find_split() function, adding one to the depth, until the final node is pure (i.e. a leaf), containing only data for one of the classes.

### 1.2 Implementation of the Evaluation Methods

To evaluate the performance of our decision tree (DT) algorithm, we implemented k-fold cross-validation. The function takes three parameters as input:

- The data-set on which the DT algorithm is evaluated
- The number of k-folds that should be created
- A boolean value that determines whether pruning should be applied or not.

First, the cross-validation function calls the function k-fold, which splits the data-set into k subsets of near-equal size (the size might differ by one row, if $len(dataset) \mod k \neq 0$). The function iterates through training and evaluation k times. Each time, another of the k subsets is used as test set while the other k-1 subsets are used for training. The resulting k models are stored in an array and returned as output of the function for the purpose of exploring the structure of the individual trees.

In order to quantify the average performance of the training algorithm, a 4x4 confusion matrix is created per iteration and stored in an array. After k iterations, the average confusion matrix is calculated by adding up the k matrices and dividing all elements by k. Alternatively, the numpy.mean() method would have been suitable to calculate the average confusion matrix, too. We deliberately decided to store and return the average confusion matrix only as it forms the basis for the calculation of every other performance metric. However, we added a function *pretty_evaluate()* which allows the cross-validation method to print all metrics (Accuracy, Precision, Recall and F1) right away.

If pruning is set to *True*, nested cross-validation is performed in order to evaluate how much pruning increases the performance of the tree model on unseen test data. The inner loop excludes the one fold of test data from the training process and therefore iterates k-1 times, using one fold as validation set and the remaining k-2 folds for training. Each of the pruned trees is then evaluated on the unseen test set. This results in an overall number of k*(k-1) confusion matrices to compute the average on. To allow for direct comparison, the function trains and returns results for the unpruned tree, too.

## 2 Evaluation Results

In the following, the performance of our decision tree learning algorithm is evaluated on both the clean and noisy data-sets.

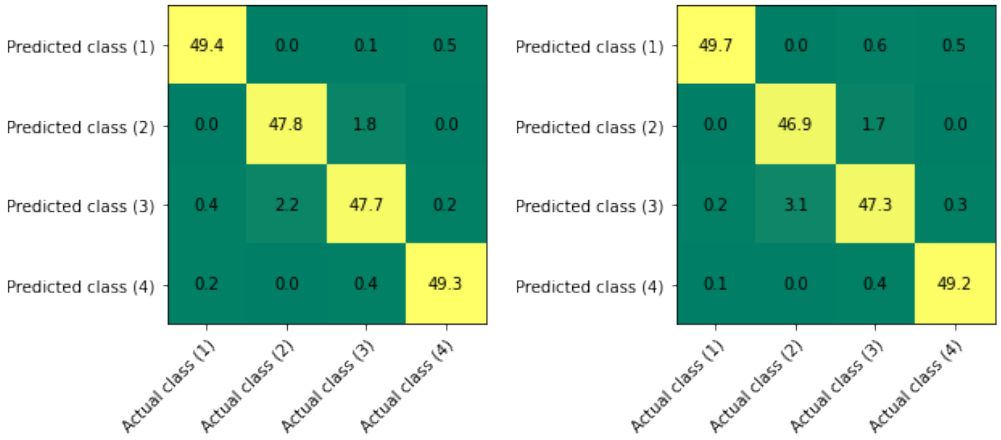
| Class | Precision | Recall | F1 |
|---|---|---|---|
| 1 | 98.8% | 98.8% | 98.8% |
| 2 | 96.37% | 95.6% | 95.98% |
| 3 | 94.46% | 95.4% | 94.93% |
| 4 | 98.8% | 98.6% | 98.7% |

(a) Unpruned Tree

| Class | Precision | Recall | F1 |
|---|---|---|---|
| 1 | 97.79% | 99.38% | 98.58% |
| 2 | 96.54% | 93.76% | 95.13% |
| 3 | 92.99% | 94.62% | 93.8% |
| 4 | 98.95% | 98.47% | 98.71% |

(b) Pruned Tree

Table 1: Algorithm performance on clean data-set.




(a) Unpruned Tree (b) Pruned Tree

Figure 2: Confusion matrices for clean data-set results (per class).

## 2.1 Evaluation on the clean Data-Set

The tree model trained on the clean data-set performs very well with a high **average classification rate/ accuracy of 97.1%** without pruning. In other words and as visible in the confusion matrix (Figure 2a), this means averaged over 10-fold cross validation 48.5 out of 50 processed WiFi signal mixes per room were classified correctly (i.e. as true positives). The classifier at hand performs especially well for rooms 1 and 4, where it has F1-scores $\geq 98.7\%$ (see Table 1a). This implies that the values for both precision and recall are extraordinarily high for these two classes. For instance, if the model predicts a mix of WiFi signals to correlate to room 4, it is 98.8% likely that the signals were actually collected from room 4 ($\rightarrow$ precision). Similarly, on average 98.6% of all signals collected from room 4 actually receive the right label from the decision tree classifier ($\rightarrow$ recall). For rooms 2 and 3, the same scores turn out slightly lower as on average roughly two instances corresponding to each room are misclassified to the other (see confusion matrix, Figure 2a and Table 1a). This is most likely related to the signal emitter right on the wall between the two rooms. However, still none of their F1-scores goes below 94.93%.

For the pruned tree, the performance metrics are very similar, but on average slightly lower. Here, the model has an average classification rate/ accuracy of 96.6% with the lowest F1-score being that of room 3 at 93.8%. For a more detailed description of pruning and its effect on the model performance please refer to Section 4.2.

## 2.2 Evaluation on the noisy Data-Set

The added noise in the second data-set means that it is harder to predict in which room a WIFI-signal was received. Due to this, the performance metrics calculated using cross-validation are slightly worse than the results for the clean data-set. On average, 79.95% of the WIFI-signals are classified correctly. Pruning proves to be very effective, lifting the average classification rate to 86.45%. A comprehensive analysis of the effects of pruning can be found in section 4.2.

Similarly to the clean data-set, the class with the worst performance metrics is room three. On average, nearly 11 signals received in room three are falsely classified as room one, two or

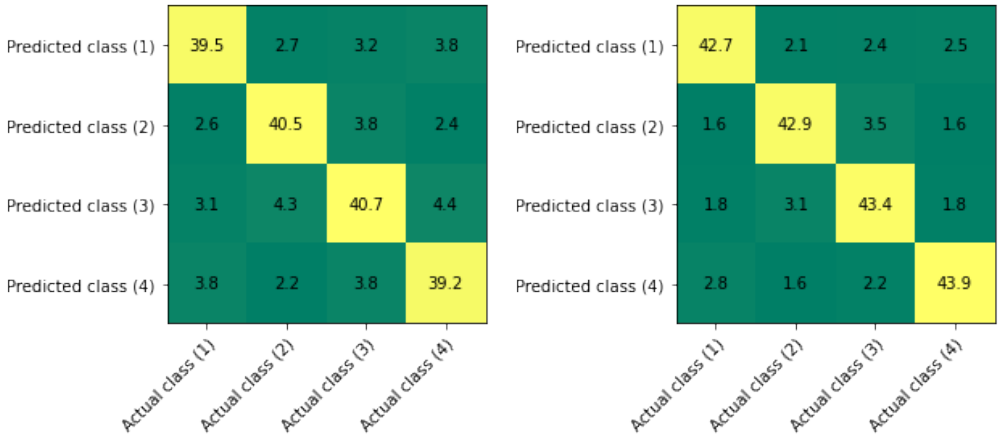
| Class | Precision | Recall | F1 |
|---|---|---|---|
| 1 | 80.28% | 80.61% | 80.45% |
| 2 | 82.15% | 81.49% | 81.82% |
| 3 | 77.52% | 79.03% | 78.27% |
| 4 | 80.0% | 78.71% | 79.35% |

(a) Unpruned Tree

| Class | Precision | Recall | F1 |
|---|---|---|---|
| 1 | 85.8% | 87.14% | 86.47% |
| 2 | 86.51% | 86.32% | 86.41% |
| 3 | 86.52% | 84.31% | 85.4% |
| 4 | 86.96% | 88.11% | 87.53% |

(b) Pruned Tree

Table 2: Algorithm performance on noisy data-set.




(a) Unpruned Tree

(b) Pruned Tree

Figure 3: Confusion matrices for noisy data-set results (per class).

four (Figure 3a), leading to a recall of 79.03% (Table 1a). The precision is even lower (77.52%): Approximately 12 signals were incorrectly classified as being received in room 3. The F1 score, which combines precision and recall equally weighted in one metric, shows the lowest performance, too. The most likely explanation for the overall performance decrease compared to the clean data-set is that the model is overfitting: The noise is misinterpreted as a signal and the trained trees do not generalize well on the unseen test data. This reasoning is supported by the observation that pruning drastically increases all of the performance metrics.

Taking a closer look at the average scores for the pruned trees, one can see that the variance in the precision per class is clearly reduced. However, the F1 score for room three is still the lowest (85.4%, Table 2b). A possible explanation is that one of the WIFI emitters is located in room two, right next to the wall that separates room three and room two (see Figure 1 in the coursework specification). The trained models interpret a signal received by this emitter as a strong indicator for room two, even though the person receiving the signal could physically stand behind the wall in room three. Therefore, the confusion matrix entry for predicted class 2 and actual class 3 is the highest (3.5 predictions on average, Figure 3b) and the recall for class three is comparatively low (84.31%, Table 2b). Following the same reasoning, it is also explainable why room four achieves the best precision, recall and F1 scores after pruning: Only one WIFI emitter is located in close proximity to the room and strong signals received from it are a good indicator that the person is located in the room four (see Figure 1 in the coursework specification).

# 3 Tree Visualisation

Figures 4 and 5 show visualisations of the final two decision trees generated by our learning algorithm on each the clean and noisy data-sets. These models were trained based on the entire provided data-sets to maximise the amount of information they could learn from the data. However, due to this no evaluation metrics can be provided for these specific trees. As per our evaluation of the impact on pruning, the tree trained on the clean data-set (Figure 4) was not pruned, while the tree trained on the noisy data-set (Figure 5) was.

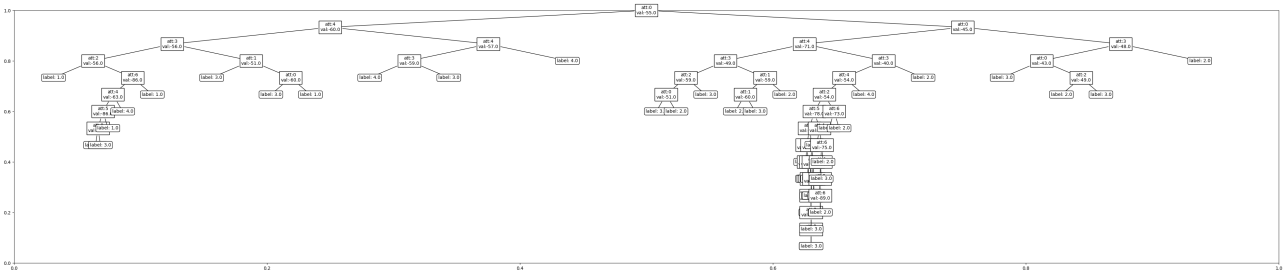


Figure 4: Model trained on the entire clean data-set (unpruned)

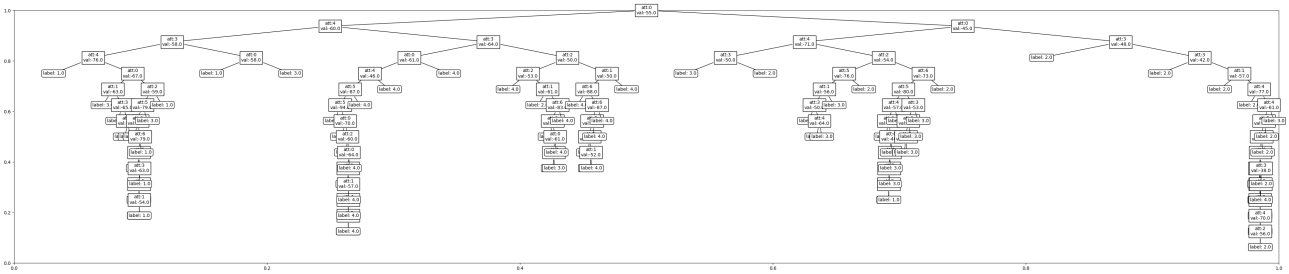


Figure 5: Model trained on the entire noisy data-set (pruned)

# 4 Answers to Questions

## 4.1 Noisy-Clean Data-Set Question

The clean data-set resulted in better performance across all classes, with all performance measurements being significantly higher regardless if the tree was pruned or not. The difference in performance at average classification rate between clean and noisy data-sets is 17.15% for unpruned tree and 10.15% for pruned.

For the clean data-set, the pruning did not improve the performance of the tree. Therefore, all classes had slightly higher results when using the unpruned model. The class with highest precision, recall and F1 score is class 1 with 98.8%, 98.8%, 98.8% respectively, followed by class 4 with 98.8%, 98.6%, 98.7% respectively.

For the noisy data-set pruning has improved the performance of the model. All classes obtained higher performance measurements, on average improving the results by 6.8%. In this case the best performing classes have swapped, with class 4 having the highest precision, recall and F1 score, 86.96%, 88.11%, 87.53% respectively, followed by class 1 with 85.8%, 87.14%, 86.47%. It is worth noting that the difference in performance between class 1 and next best performing class is much smaller than in the clean data-set.

The reason for this data-set to have a worse performance in general is due to the noise within the data-set. The noise means that there is meaningless information (possibly due to error in measurement) within the data that we train our model on. The model can not interpret it correctly resulting in a poorer performance. The noise can contribute towards the model being over-fitted, performing very well on the training data but much poorer on the testing data. This suggests that we either do not have enough data or our algorithm is not good enough to separate the signal from the noise, resulting in the model memorising the noise instead of finding it.

With the use of pruning, reducing overfitting outliers in the training data-set, and cross-validation, the effect of the noise on the model has been reduced. However, in the future, it is also possible to try and identify potential noise instances and remove them from training data.

### 4.2 Pruning Question

The pruning method *prune(tree_node, data-set)* takes the tree node dictionary (as shown in Figure 1) as an input and recursively prunes/ removes any nodes which are only connected to leaves and whose removal does not reduce the accuracy for the given node. The accuracy is determined based on the data-set the prune method is passed, i.e. the validation data-set. The data-points from this set are distributed throughout the (sub)tree at hand and in sum treated as the weight of a leaf. If a node is decided to be pruned, the class label of the leaf with the higher weight becomes the class label of the newly created leaf. Pruning was executed using k-fold-cross validation as described in Section 1.2.

The effects of pruning differ significantly between the two data-sets. Looking at the clean data-set, the act of pruning shows only very little impact. The average classification rate/ accuracy across all classes is slightly decreased from 97.1% for the original tree to 96.6% for the pruned tree. This shows that while it is a general condition for pruning that a leaf is only "cut" if this improves the tree's accuracy, the different distribution of data within the validation- and test-data-sets also allow pruning to slightly decrease the final accuracy. As can be observed from table 1, the individual values for precision, recall and F1-score per class are all very similar to each other for the clean data-set. Some values are slightly improved by pruning (e.g. recall for room 1 from 98.8% to 99.38%), while others are made slightly worse (e.g. recall for room 2 from 95.6% to 93.76%).

However, moving on to the noisy data-set, pruning shows more significant effects. For this data-set, pruning improves the overall classification rate/ accuracy from 79.95% for the original tree to 86.45% for the pruned tree. Looking at table 2, the values for precision, recall and the F1-score per class are each significantly improved as well (see Section 4.1). One specific observation that can be made from the performance metrics of the pruned model (Table 2b) is that pruning has almost harmonised the precision values for all classes to $\approx 86\%$, while these were initially scattered between 77.52% and 82.15%. In other words, a set of WiFi signals predicted to correspond to a

given room by the pruned tree model is approximately 86% likely to have actually originated from that room, while before pruning that same probability could have been as low as 77.5% (for room 3, see Table 2a). Overall, these results show that pruning has reduced the original tree's overfit to the noise in the training data significantly, which allows for better performance on the test data and presumably on any other unseen data.

For exemplary visualisations of individual trees created in the process of cross validation and pruning (as described in Section 2) please refer to Figures 6 and 7 (exemplary unpruned and pruned tree, respectively, trained on clean data) and Figures 8 and 9 (exemplary unpruned and pruned tree, respectively, trained on noisy data) in Appendix A.

### 4.3 Depth Question

We can associate depth with the complexity of the model. Having a deep tree means that we have a complex model. Using our knowledge from the evaluation classes, we comment on the relationship between depth and the prediction accuracy.

The decision tree learning algorithm keeps creating new nodes and increasing the depth until all samples in one split have the same label. This works fine in the clean data-set. However, when we have noise in the data, the decision tree model gets very deep and complex to be able to decide on every single point. This causes overfitting. As mentioned before, overfitting may do well in the training data-set, but it performs poorly on the unseen test data.

| Depth | Unpruned | Pruned |
|---|---|---|
| Clean | 13.5 | 8.3 |
| Noisy | 20.1 | 14.2 |

(a) Average decision tree depths

| Accuracy | Unpruned | Pruned |
|---|---|---|
| Clean | 97.1% | 96.6% |
| Noisy | 79.95% | 86.45% |

(b) Average decision tree accuracies

Table 3: Relationship between depth and prediction accuracy

Using pruning, we reduce the depth of the tree by deleting nodes that do not increase accuracy on training data. On the clean data-set, accuracy on test data suffers by 0.5% when we simplify the model by reducing the depth from the average of 13.5 to 8.3. This suggests that pruning makes the model a little simpler than optimal. Whereas on the noisy data-set, the prediction increases significantly by 6.5% when we decrease the depth from 20.1 to 14.2, which means we got much closer to the optimal complexity by reducing the depth. The depths and prediction accuracies of all four decision trees of can be seen in the table above.

To sum up, accuracy dropped a little on the clean dataset while reducing depth, suggesting that our pruning might be a little too agressive. Moreover, accuracy increased significantly in the noisy dataset while reducing the depth, indicating that our pruning function does a good job on preventing overfit.

# Appendix A Further Tree Visualisations

This section shows additional exemplary visualisations of trees created in the process of cross-validation, which contributed to the performance metrics presented in Section 2.

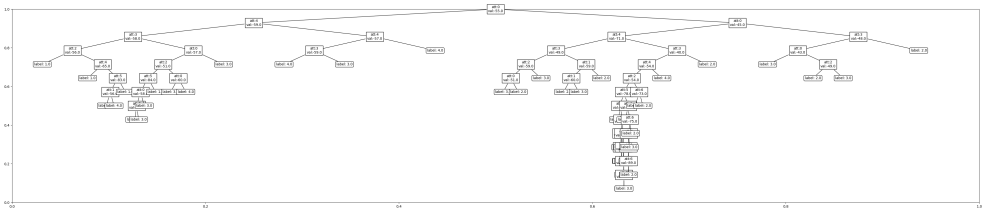


Figure 6: One of the trees from cross-validation (clean data, unpruned).

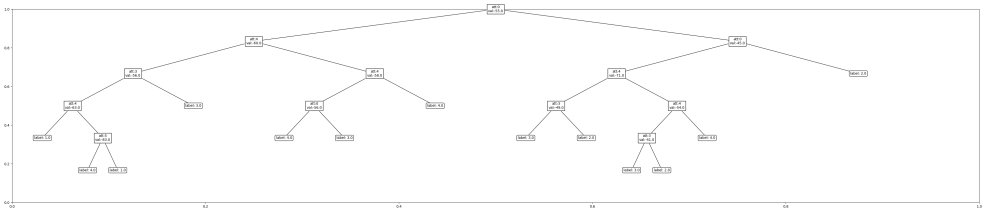


Figure 7: One of the trees from cross-validation (clean data, pruned).

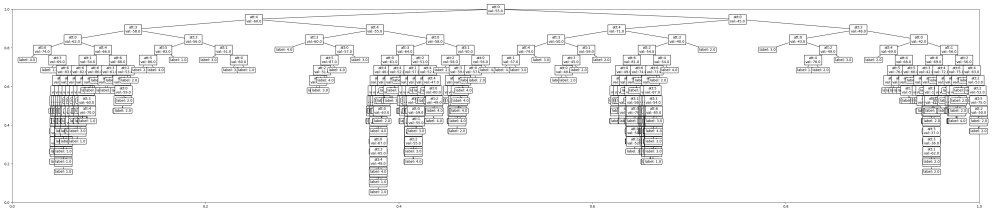


Figure 8: One of the trees from cross-validation (noisy data, unpruned).

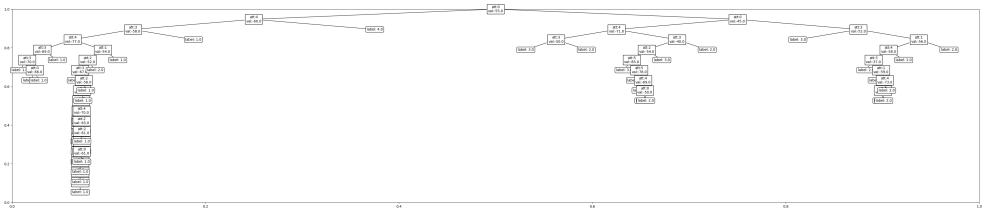


Figure 9: One of the trees from cross-validation (noisy data, pruned).