Imperial College
London

COURSEWORK 1: DECISION TREE

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Intro to Machine Learning

Authors:
Christoph Renschler
Moin Bukhari
Nayan Kad
Nikita Belenkov

Date: November 6, 2020

1 Introduction

Machine learning has a plethora of applications in the real world and one avenue that we delve into in this report concerns device location detection using WiFi signal strengths.

For this project we were given datasets containing signal strengths from 7 different WIFI emitters and were tasked to locate the position of a mobile phone within 1 out of 4 rooms in the building. We were provided with 2 datasets; one clean and one noisy, each containing 2000 examples. These were used to train and evaluate our algorithm. We used a splitting function to form a decision tree and then used pruning to optimise sections of the tree that were redundant for classification.

2 Implementation

2.1 Building the decision tree

The decision tree is made by splitting the dataset at the optimal split point that provides the greatest information gain. Nodes are referenced by their parent node via the left and right attributes, and the entire tree can be accessed via the root node.

The nodes are split into decision nodes and leaf nodes. The decision node states the attribute (WIFI receiver) used to split and the value (signal strength) at which the split takes place. The leaf node always has an attribute of 7 and just contains the predicted room number as its leaf variable.

```
class Node:
        self.attribute # Attribute used for decision
        self.value # Optimal value for decision
        self.left # Left children node
        self.right # Right children node
        self.leaf # Indicates predicted value at a leaf node
                 # (0 if this is a decision node)
        self.dataset # Training subset associated with a node
```

Having a single class for both decision and leaf nodes, as well as storing the data subset associated with every node, allows for a much more efficient pruning implementation later on. However, this only makes sense for models trained on considerably small training sets.

The decision tree is built recursively by calling *decision_tree_learning*, which returns the root, depth and number of leaf nodes of the tree. Since there is no early stopping or pruning implemented at this stage, the returned tree is heavily overfitted.

2.1.1 Optimal split point

In order to construct a decision tree we had to iterate through our data to decide the which points to split at in order to classify our decisions. This was done by writing the *find_split* function which takes in a dataset as an argument and returns a pair containing the attribute and value to split on. The method for calculating the split point relies on knowing the information entropy of the complete dataset and of all the values for each attribute and choosing the attribute-value pair that results in the highest information gain. The mathematical formulae for this is as follows:

$$Gain(S_{all}, S_{left}, S_{right}) = H(S_{all}) - Remainder(S_{left}, S_{right})$$

where

$$H(dataset) = -\Sigma_{k=1}^{k=K} p_k * log_2(p_k),$$

$$Remainder(S_{left}, S_{right}) = \frac{|S_{left}|}{|S_{left}|+|S_{right}|}H(S_{left}) + \frac{|S_{right}|}{|S_{left}|+|S_{right}|}H(S_{right})$$

This method must be carried out on a sorted list as after choosing the necessary split point we separate the data into 2 sub-lists; one containing values lower than the split point and the other containing the values equal to and higher than the split point.

2.2 Evaluation

2.2.1 Initial Performance Assessment

To assess the performance of our newly generated decision tree we implemented a function *predict* which takes as argument a decision tree and signal samples to make predictions with. The function traverses the tree recursively using a comparison of the signal values and the decision values assigned on the tree to decide what room to predict for each set of signal strengths.

The function *evaluate* takes as argument a dataset and decision tree and allows us to identify when the decision tree makes a correct prediction and when it makes a misprediction. It compares the predicted value to the actual value and keeps a count of how many correct predictions have been made.

2.2.2 Cross Validation

The relatively scarce data makes cross-validation an appropriate technique to tune and test our decision tree. The *crossValidate* method performs n-fold cross validation, taking in the entire dataset and the desired number of folds. We first shuffle the whole dataset, then split it into 10 equal parts. For simple testing, we train 10 trees on 9 different parts and use the last 1 for testing, therefore allowing 200 items for testing. The accuracy is calculated with our *evaluate* function and averaged over the 10 trees. In order to validate the performance effect of pruning on the decision tree, we reserve one out of the 10 folds for validation. We decided to do so since the given dataset was too large to run an second internal cross-validation over all folds used for training on our local machines.

2.3 Pruning

To further improve the performance of our decision tree, we pass our tree through a process of pruning. We have identified and tested 2 distinct ways that we can prune a tree.

The first way is starting from the leaf node and traversing up the tree by pruning every 2 nodes below. After every recursive prune, we check if the partially pruned tree performs better than the previous tree. If so, we go ahead and perform the alteration, otherwise we stop pruning at that point.

The second way is pruning any node by simply looking at the training data subset associated with that node and taking the most common label as the label for the new leaf node. While this requires less processing we believe that it also leads to higher accuracy since you do not keep recursively averaging the labels. Comparing this to popular electoral systems, each state gives all its electoral votes to one candidate based on the majority of votes in that state. The president is then elected if they win the majority of electoral votes. This could lead to a president being elected that the majority of citizens actually did not vote for.

We thus decided to use a *smartPrune* method, which tries pruning the entire subtree by adopting the majority label of its training subset. It then compares the accuracy of the tree before and after being pruned using a validation set and performs or does not perform the prune accordingly.

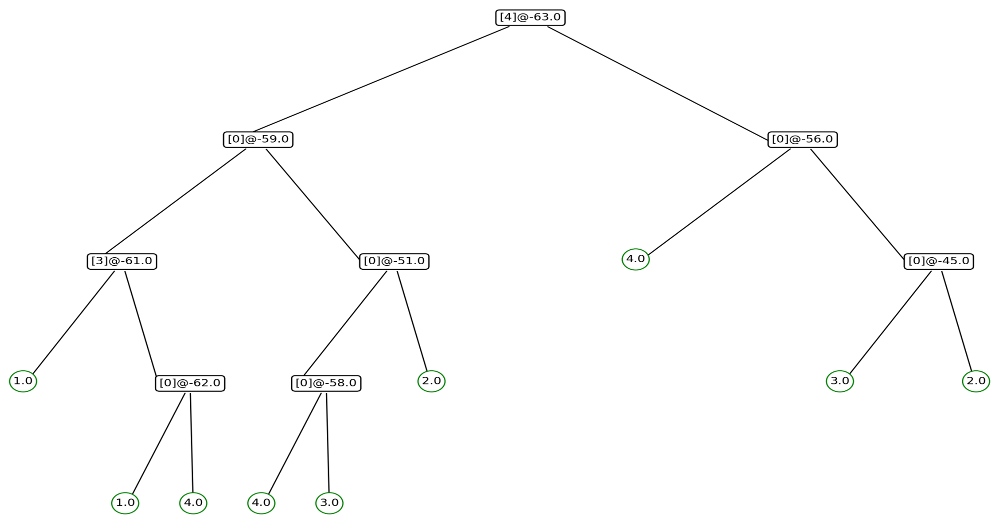

Figure 1: Unpruned decision tree generated with 100 samples of noisy data with a 1:1 training:validation data split

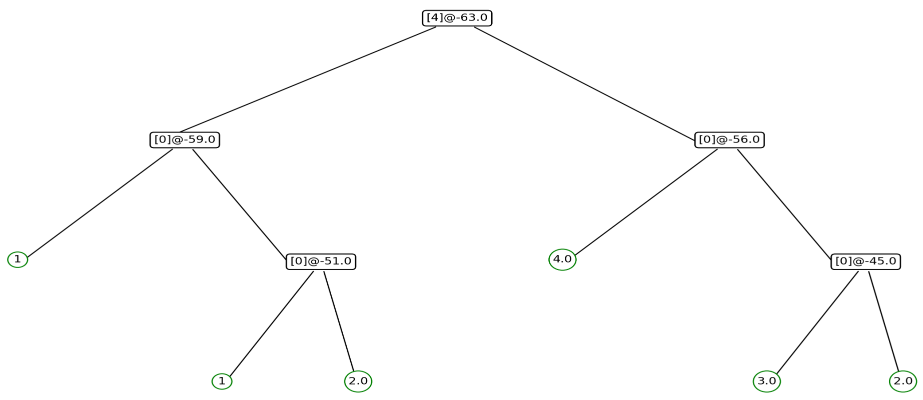

Figure 2: Decision tree after pruning

3 Results

3.1 Understanding the problem case

Since we did not implement any early stops, we expected the unpruned decision tree to heavily overfit on the training set, and thus the model having a relatively low accuracy. However, the accuracy of our decision tree when trained on the clean dataset was staggeringly high. By varying the training set size, we could confirm that the high accuracy is reasonable, given that the accuracy increases rapidly when using only very small training sets. While an accuracy of 25 percent is no better than random, which is the accuracy we get if we train on 1 data point as expected, a training size of 100 already leads to nearly 78% (noisy) and 92% (clean) accuracy. We found this very surprising and it must speak to the quality of the clean data and that the noisy data is not so noisy after all.

3.2 Metrics

A confusion matrix – the average of 10 models trained and tested using cross-validation – shows the performance of the unpruned and optimally pruned model for both the clena and noisy datasets. We use the following metrics to evaluate the performance of our classification model. Given that occurences of false positive and false negative predictions are equally amiss, the F1 measure is a good fit for this problem.

$$\text{Precision} = \frac{TP}{TP+FP} \qquad \text{Recall} = \frac{TP}{TP+FN}$$

$$TP = \text{True positive} \quad TN = \text{True negative} \quad FP = \text{False positive} \quad FN = \text{False negative}$$

$$F_1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = \frac{\text{TP}}{\text{TP} + \frac{1}{2}(\text{FP} + \text{FN})}$$

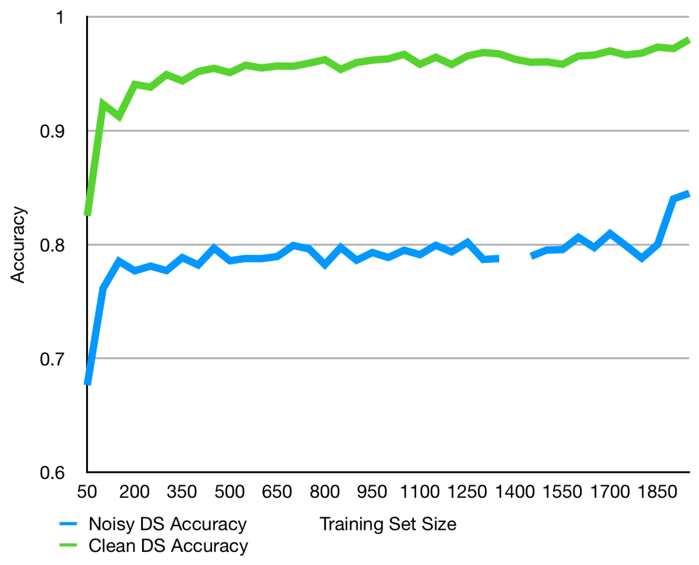

Figure 3: Change in accuracy with increasing size of the training set (averaged over 5 runs

3.3 Clean data

As discussed above, the accuracy of the unpruned decision tree was staggeringly high even before pruning. Below are the average precision, recall and F1 rates for each class as well as the macro-average rates of the unpruned tree trained on the clean dataset.

Actual Room \ Predicted Room	1	2	3	4
1	49.0	0	0.3	0.7
2	0.0	47.8	2.2	0.0
3	0.2	2.3	47.1	0.4
4	0.3	0.0	0.4	49.3

Table 1: Unpruned, clean dataset trained, 10 fold cross validated confusion matrix

	Actual Room				
	1	2	3	4	Macro
Precision Rate	0.990	0.954	0.954	0.938	0.957
Recall Rate	0.980	0.955	0.964	0.980	0.962
F1 Measure	0.984	0.954	0.957	0.958	0.960
CR					0.966

Table 2: Evaluation metrics for the unpruned, clean dataset.

Since the accuracy was already considerably high, pruning the tree merely increased accuracy by 1.1% from 96.6% to 97.7%. Interestingly, the precision and recall rates for all classes have improved in this example, although this technically is not always the case with pruning.

Actual Room \ Predicted Room	1	2	3	4
1	49.5	0.0	0.2	0.3
2	0.0	48.1	1.9	0.0
3	0.6	1.0	48.2	0.2
4	0.3	0.0	0.2	49.5

Table 3: Pruned, clean dataset trained, 10 fold cross validated confusion matrix

	Room				
	1	2	3	4	Macro
Precision Rate	0.988	0.980	0.954	0.990	0.978
Recall Rate	0.990	0.962	0.976	0.990	0.979
F1 Measure	0.989	0.971	0.965	0.990	0.978
CR					0.977

Table 4: Evaluation metrics for the pruned, clean dataset.

3.4 Noisy data

Actual Room \ Predicted Room	1	2	3	4
1	38.8	3.2	2.8	4.2
2	3.3	39.5	4	2.9
3	3.3	3.8	41.4	3
4	4.4	2.1	4	39.3

Table 5: Unpruned, noisy dataset trained, 10 fold cross validated confusion matrix

Here we can see the results that we get from training our tree on the noisy data before pruning. You can see that we get 79.5% average classification rate, but all the other metrics are the same, which we found a bit odd.

	Room				
	1	2	3	4	Average
Precision Rate	0.792	0.795	0.804	0.789	0.795
Recall Rate	0.779	0.812	0.793	0.796	0.795
F1 Measure	0.785	0.804	0.798	0.792	0.795
CR	0	0	0	0	0.795

Table 6: Evaluation metrics for the unpruned, noisy dataset.

After pruning we can see that the average classification rate has increased to 89.4%, which shows that the decision tree was heavily overfitted and that the pruning operation successfully reduced the classification error. The other metrics have also increased by are still equal to each other. Overall we have gotten some very good results from training the tree on both noisy and clean datasets.

Predicted Room / Actual Room	1	2	3	4
1	44.8	1.2	0.8	2.2
2	1.9	45.3	1.6	0.9
3	2.1	2.7	44.7	2
4	2.2	1.3	2.4	43.9

Table 7: Pruned, noisy dataset trained, 10 fold cross validated confusion matrix

	Room				
	1	2	3	4	Average
Precision Rate	0.914	0.911	0.868	0.881	0.894
Recall Rate	0.878	0.897	0.903	0.896	0.894
F1 Measure	0.896	0.903	0.885	0.888	0.893
CR	0	0	0	0	0.894

Table 8: Evaluation metrics for the pruned, noisy dataset.

In order to achieve the highest possible accuracy, we attempted combining the clean and noisy dataset and training our tree on that joined dataset. However, we faced issues of having entries with identical attributes but with different labels. Dealing with such a condition would result in a less efficient $decision_tree_learningfunctiontobuildthetree,wheref$

3.5 Comparison

While pruning had a positive effect on models trained on both datasets, increasing the bias and decreasing the variance in both cases, it had a considerably stronger effect on the decision tree trained on noisy data. This can be explained by the lower quality data, which causes the model to overfit much more easily.

4 Questions

4.1 Is there any difference in the performance when using the clean and noisy datasets? If yes/no explain why. Discuss the differences in the overall performance and per class.

Yes, there is a noticeable performance difference. The noisy dataset consistently performs worse than the clean one usually by about 10% as you can see in Figure 3. The noisy data-set yield a lower accuracy due the to the inherent presence of error, when the data is collected or generated it will contain some element of human or instrument error, therefore it logically performs lower than the ideal clean dataset. The drawback of the clean dataset that it is much harder to acquire in the real world. In the case of the data we have been given it is of very good quality and if we look back on Figure 3, you can see that with only a 100 data point training set one can achieve good accuracy results. The difference is slightly lower if we look at pruned trees which were trained on clean vs noisy datasets, which comes out with noisy being about 8% higher accuracy.

4.2 Briefly explain how you implemented your pruning function and details the in influence of the pruning on the accuracy of your tree on both datasets.

In order to prevent overfitting our tree to the training data we must attempt to prune our decision tree by methodically removing nodes from our trained decision tree and comparing the performance of the pruned tree against the unpruned tree on an unseen dataset.

To implement our pruning function perfectlyPruned we used a bottom-up approach whereby we recursively move down in the tree to begin at the first node which is a parent of 2 leaf nodes and assess whether there will be an improvement if we prune this node. To do this we calculate the accuracy of the tree as if we had made the prune on the current node by running the tree on a test dataset and compare that to the accuracy of the tree prior to the prune in question. The outcome of this determines whether the prune is kept or whether the node and it's 2 leaves are restored.

If the prune is not kept then we can infer that no parent of this node should be pruned as this cannot improve the accuracy of the decision tree. This inference is realised in the smartPrune function which is checked to decide whether any prunes at a higher level in the decision tree are viable. We break out of the current recursive loop and check the next parent node which has 2 leaves and this process repeats for all nodes in the tree. The value assigned to a node is decided by comparing the size of the associated datasets of the leaves and using the value of the dataset with the highest quantity of tests.

We also used a cross validation approach to pruning in our function prunedCrossValidate_confusion. We did this by splitting our dataset into k folds and using k-1 folds to train and 1 fold to test. This configuration is rotated so that every fold is used as a validation set once. This is the best method to use when we have a limited dataset for training and validating and ensures we get the maximum usage out of the dataset.

As mentioned in the previous section, the performance difference of the decision tree on the noisy dataset increases after pruning by approximately 8%. This is calculated by comparing the data acquired when running the unpruned tree (Table 6) and the pruned tree (Table 8). Overfitting makes the decision tree sensitive to fluctuations in the training data and this will lead to incorrect inferences being made about unseen data - and subsequently more mispredictions. Pruning enables the tree to make more general inferences about the data and the overall relationship of the attibutes and labels to provide predictions with a higher level of accuracy and less sensitivity.

4.3 Comment on the maximal depth of the tree that you generated for both datasets and before and after pruning. What can you tell about the relationship between maximal depth and prediction accuracy?

The unpruned tree generated on the clean dataset had a depth of 10 and consisted of approximately 52 leaf nodes. While pruning decreased the number of leaf nodes by 14 on average, it did not affect the tree depth. Since the dataset is less noisy, the decision tree did not overfit quite as much. Training the decision tree on the noisy dataset lead to a much stronger overfit. Therefore was able to increase accuracy much more, while reducing the amount of leaf nodes from approximately 300 to 200 and reducing the depth by 1. In conclusion, the highest accuracy is being achieved with the most perfect fitting decision tree. Decreasing the depth of a overfitted tree can greatly increase accuracy while reducing the depth. Pruning the tree by too much, however, will lead to less and less accurate models. However, in some cases, simpler models might be preferable even though they are slightly less accurate, for example when the power consumption of a model plays a major role. Optimizing our decision tree algorithm for such an case could be achived by introducing the depth reduction of a tree as a weight in the decision to accept or reject a prune. This could be added to the smartPrune function.