DD2421 Machine learning Lab-1 Report

Markus Brewitz
Raahitya Botta

Assignment 0: Each one of the datasets has properties which makes them hard to learn. Motivate which of the three problems is most difficult for a decision tree algorithm to learn.

MONK-3 has the most noise, which would make it more difficult for a machine learning algorithm to learn. The decision tree might overfit due to the noise which might result in reduced accuracy on the test data. It also has the smallest training set.

MONK-2 has the condition that ai = 1 for exactly two i ∈ {1,2,...,6}, which can be difficult for a decision tree to identify the combination of where this condition applies.

Out of the two, MONK-3 is probably the harder to learn due to the noise, though.

Assignment 1: The file `dtree.py` defines a function `entropy` which calculates the entropy of a dataset. Import this file along with the monks datasets and use it to calculate the entropy of the *training* datasets.

Dataset	**Entropy**
MONK-1	1.0
MONK-2	0.957117428264771
MONK-3	0.9998061328047111

> **Assignment 2:** Explain entropy for a uniform distribution and a non-uniform distribution, present some example distributions with high and low entropy.

Entropy can be known as a measure of unpredictability or randomness.

$$\text{Entropy} = \sum_i -p_i \log_2 p_i$$

In a uniform distribution, each outcome is equally likely. This would mean that a dataset that is uniform that is the same size as a non-uniform dataset would have a higher entropy, since there is no way of predicting the more likely outcome.

In non-uniform distribution, the outcomes have different likelihood. The entropy of such distribution can vary depending on the probabilities spread in the outcomes.

High entropy: Uniform distribution, for example a fair six-sided dice:

Example: rolling a die

$p_1 = \frac{1}{6}; \quad p_2 = \frac{1}{6}; \ldots \quad p_6 = \frac{1}{6}$

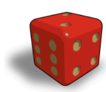
$$\begin{aligned}\text{Entropy} &= \sum_i -p_i \log_2 p_i = \\ &= 6 \times (-\frac{1}{6} \log_2 \frac{1}{6}) = \\ &= -\log_2 \frac{1}{6} = \log_2 6 \approx 2.58\end{aligned}$$

The result of a die-roll has 2.58 bit of information

Low entropy: A non-uniform distribution, for example a loaded die:

Example: rolling a fake die

$p_1 = 0.1; \ldots \quad p_5 = 0.1; \quad p_6 = 0.5$



$$\begin{aligned}\text{Entropy} &= \sum_i -p_i \log_2 p_i = \\ &= -5 \cdot 0.1 \log_2 0.1 - 0.5 \log_2 0.5 = \\ &\approx 2.16\end{aligned}$$

A real die is more unpredictable (2.58 bit) than a fake (2.16 bit)

Another example would be a weighted coin with $p_{head} = 0.05$ and $p_{tail} = 0.95$, giving us:

$$Entropy = -0.05 * log(0.05) - 0.95 * log(0.95) = 0.08621407475$$

Assignment 3: Use the function `averageGain` (defined in `dtree.py`) to calculate the expected information gain corresponding to each of the six attributes. Note that the attributes are represented as instances of the class Attribute (defined in `monkdata.py`) which you can access via `m.attributes[0]`, ..., `m.attributes[5]`. Based on the results, which attribute should be used for splitting the examples at the root node?

Dataset	a1	a2	a3	a4	a5	a6
MONK-1	0.07527255560831925	0.0058384299629092 86,	0.0047075666172972 1	0.0263116965076822 8	**0.28703074971578435**	0.0007578557158638421
MONK-2	0.0037561773775118823	0.0024584986660830532	0.0010561477158920196	0.0156642472926438 18	**0.01727717693791797**	0.006247622236881467
MONK-3	0.007120868396071844	**0.29373617350838865**	0.0008311140445336207	0.0028918172886543 97	0.2559117246197275 5	0.007077026074097326

Attribute a_5 is on average the attribute with the best information gain, and it's the best one for MONK-1 and MONK-2, so that would be the best one to use for splitting at the root node. For dataset MONK-3, attribute a_2 is slightly better, so that could be used.

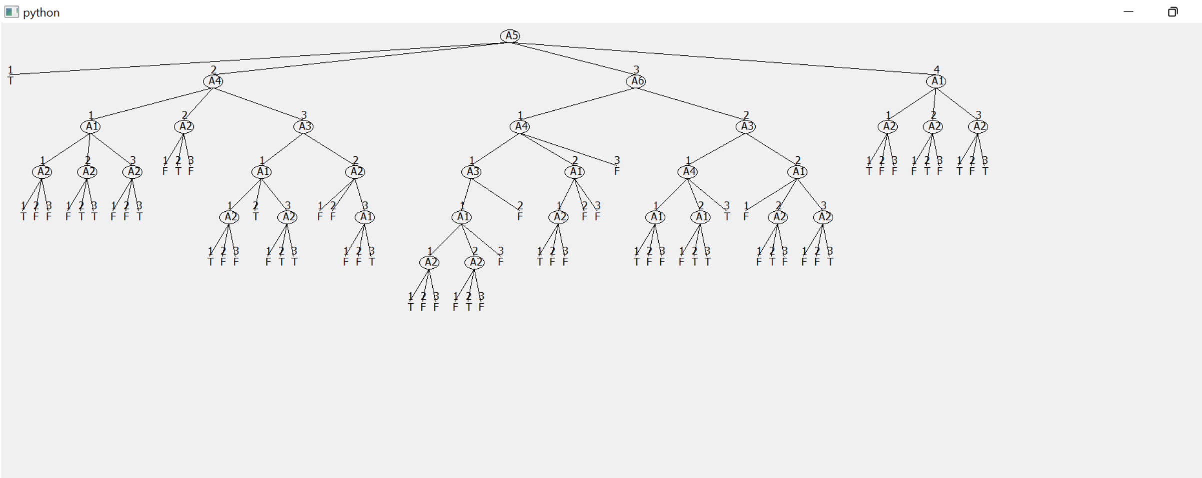
Assignment 4: For splitting we choose the attribute that maximizes the information gain, Eq.3. Looking at Eq.3 how does the entropy of the subsets, S_k, look like when the information gain is maximized? How can we motivate using the information gain as a heuristic for picking an attribute for splitting? Think about reduction in entropy after the split and what the entropy implies.

Maximising the information gain minimises the entropy, and since lower entropy implies that the S_k is better than S at representing the underlying pattern, it functions as a good heuristic for choosing an attribute for splitting. In general, maximizing information gain when splitting the attributes leads to lower entropy meaning the data is split in a way that is most beneficiary for classification and efficient decision-making.

5 Building Decision Trees

The attribute with the highest information gain for the monk1 dataset is attribute 5. This subset includes the values [1, 2, 3, 4]. If we create subsets from this and determine the information gain, we get the following:

Value	A1	A2	A3	A4	A5	A6
Subset 1	0	0	0	0	0	0
Subset 2	0.040216841609413634	0.015063475072186083	0.015063475072186083	**0.04889220262952931**	0.0	0.025807284723902146
Subset 3	0.03305510013455182	0.0021971835391009 22	0.017982293842278896	0.01912275517747053	0.0	**0.04510853782483648**
Subset 4	**0.20629074641530198**	0.033898395077640586	0.025906145434984817	0.07593290844153944	0.0	0.0033239629631565126




```
PS D:\Machine Learning\dectrees Lab1\dectrees\python> & C:/Users/sri/AppData/Local/Programs/Python/Python310/python.e
Learning/dectrees Lab1/dectrees/python/5.3.py"
Root: A5
  When A5 = 1:
    Split by: A1
      Class = True
  When A5 = 2:
    Split by: A4
      Class = False
  When A5 = 3:
    Split by: A6
      Class = False
  When A5 = 4:
    Split by: A1
      Class = True
PS D:\Machine Learning\dectrees Lab1\dectrees\python> 
```

Assignment 5: Build the full decision trees for all three Monk datasets using **buildTree**. Then, use the function **check** to measure the performance of the decision tree on both the training and test datasets.
For example to built a tree for **monk1** and compute the performance on the test data you could use

```
import monkdata as m
import dtree as d

t=d.buildTree(m.monk1, m.attributes);
print(d.check(t, m.monk1test))
```

Compute the train and test set errors for the three Monk datasets for the full trees. Were your assumptions about the datasets correct? Explain the results you get for the training and test datasets.

	E_{train} error	E_{test} error
MONK-1	0.0	0.17129629629629628
MONK-2	0.0	0.30787037037037035
MONK-3	0.0	0.05555555555555558

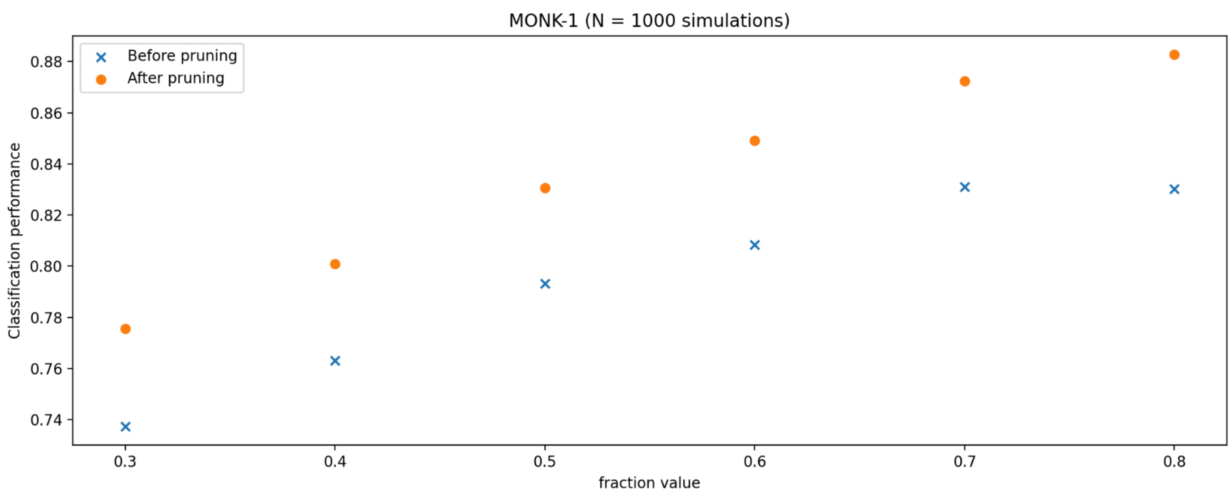
Assignment 6: Explain pruning from a bias variance trade-off perspective.

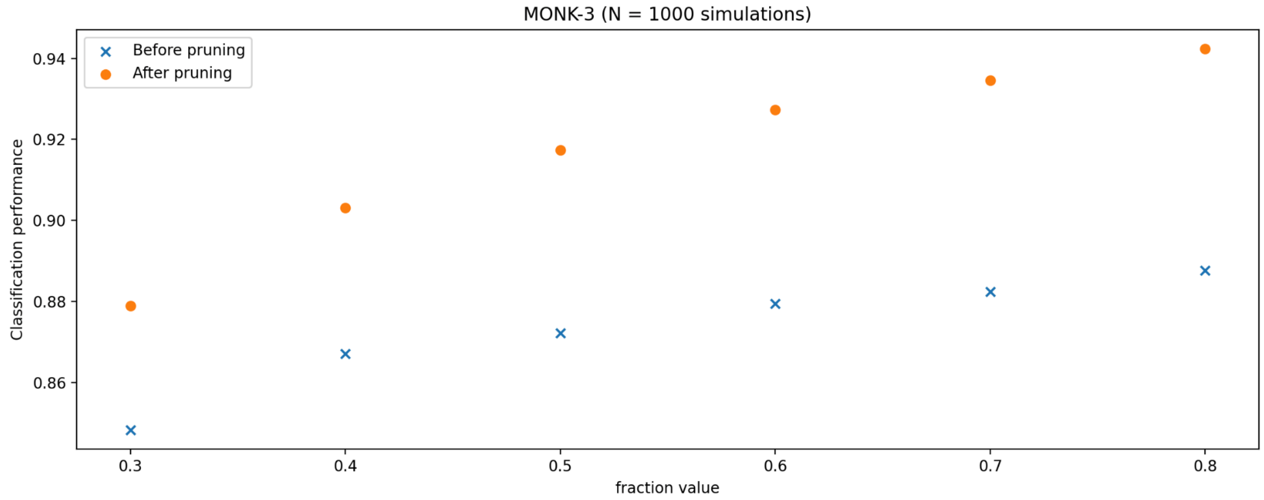
Bias refers to errors the learning algorithm has because of being overly simplistic, which might make it miss relevant relations in the dataset. In a decision tree, high bias might mean a shallow tree.

Variance refers to how sensitive the algorithm is to changes in the data set. High variance can come with more complexity, such as a deeper decision tree.
Lower bias often means a higher variance, and vice versa, and since we don't want the algorithm to have many errors or overfit on the data sets, finding a balance between the two is important. Through pruning the tree, we lower the variance, and by performing classification performance checks on the pruned trees, we can ensure that we only prune the tree so much so that we don't increase the bias and error too much.

Assignment 7: Evaluate the effect pruning has on the test error for the `monk1` and `monk3` datasets, in particular determine the optimal partition into training and pruning by optimizing the parameter `fraction`. Plot the classification error on the test sets as a function of the parameter `fraction` $\in \{0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$.

Note that the split of the data is random. We therefore need to compute the statistics over several runs of the split to be able to draw any conclusions. Reasonable statistics includes mean and a measure of the spread. Do remember to print axes labels, legends and data points as you will not pass without them.




MONK-3 (N = 1000 simulations)
Before pruning
After pruning
Classification performance
0.94
0.92
0.90
0.88
0.86
0.3
0.4
0.5
0.6
0.7
0.8
fraction value