DD2421 Machine Learning - Lab 1: Decision Trees Python version

Örjan Ekeberg
Updated 2017 by Martin Hjelm & Nils Bore
Finished 2025 by Yiyi Miao (yiyim@kth.se)

September 10, 2025

1 Preparations

In this lab you will use a set of predefined Python functions to build and manipulate decision trees. In order to run this, you need to have Python installed. We also use the Qt graphics library, PyQt, for plotting. PyQt comes in different versions 4 or 5. For Mac users version 5 is recommended.

For different Windows versions consult the Internet. For Mac use Homebrew to install Python and PyQt5. For Ubuntu Python and PyQt are available in the debian package repository. Python and PyQt are also installed on the Unix computers in the computer halls.

Note: It is possible to do the lab without using the plotting functions found in PyQt, but then you will not be able to see the generated decision trees.

2 MONK datasets

This lab uses the artificial MONK dataset from the UC Irvine repository. The MONK's problems are a collection of three binary classification problems MONK-1, MONK-2 and MONK-3 over a six-attribute discrete domain. The attributes $a_1, a_2, a_3, a_4, a_5, a_6$ may take the following values:

$$\begin{array}{lll} a_1 \in \{1,2,3\} & a_2 \in \{1,2,3\} & a_3 \in \{1,2\} \\ a_4 \in \{1,2,3\} & a_5 \in \{1,2,3,4\} & a_6 \in \{1,2\} \end{array}$$

Consequently, there are 432 possible combinations of attribute values. The true concepts underlying each MONK's problem are given by table 1.

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.

Table 1: True concepts behind the MONK datasets

MONK-1	$(a_1 = a_2) \vee (a_5 = 1)$
MONK-2	$a_i = 1$ for exacly two $i \in \{1, 2, \ldots, 6\}$
MONK-3	$(a_5 = 1 \wedge a_4 = 1) \vee (a_5 \neq 4 \wedge a_2 \neq 3)$

MONK-3 has 5% additional noise (misclassification) in the training set.

Table 2: Characteristics of the three MONK datasets

Name	# train	# test	# attributes	# classes
MONK-1	124	432	6	2
MONK-2	169	432	6	2
MONK-3	122	432	6	2

Answer: Among the three MONK datasets, **MONK-2 is the most difficult** for a decision tree algorithm to learn.
The target concept for MONK-2 is exactly two attributes are equal to 1. This is a parity-like rule, which cannot be captured by a single attribute test. In other words, we have to take all 6 attributes into consideration. Decision trees such as ID3 rely on maximizing information gain locally, but in MONK-2 all single attributes have very low information gain at the root (in Assignment 3). The learner must grow a large, deep tree to represent all combinations of attribute values, making it highly prone to overfitting.
In contrast:

- **MONK-1** has a simpler logic: it is true when either $(a_1 = a_2)$ or $(a_5 = 1)$. This can be learned with relatively few splits.
- **MONK-3** has added noise (5% mislabels), but its underlying rule still contains clear attributes (involving a_5, a_4, a_2), which should be easier than MONK-2, but harder than MONK-1 due to the noise.

MONK-1 and MONK-3 both concern 3 attributes out of 6 while MONK-2 needs all the attributes.

The data consists of three separate datasets MONK-1, MONK-2 and MONK-3. Each dataset is further divided into a training and test set, where the first one is used for learning the decision tree, and the second one to evaluate its classification accuracy (see table 2).

The datasets are available in the file `monkdata.py`. In particular, six variables are defined which contain the datasets: `monk1`, `monk1test`, `monk2`, `monk2test`, `monk3` and `monk3test`. Each dataset is a sequence (more precisely, a tuple) of instances of the class `Sample`, defined in the same file.

You can access the data in your own Python scripts by importing the `monkdata.py` file as a module like this:

```
import monkdata as m
```

This makes the variable `m` a shorthand for the module so that you can access the datasets by writing `m.monk1`, etc.

3 Entropy

In order to decide on which attribute to split, decision tree learning algorithms such as ID3 and C4.5 use a statistical property called information gain. It measures how well a particular attribute distinguishes among different target classifications. Information gain is measured in terms of the expected reduction in the entropy or impurity of the data. The entropy of an arbitrary collection of examples is measured by

$$\text{Entropy}(S) = -\sum_i p_i \log_2 p_i \tag{1}$$

in which p_i denotes the proportion of examples of class i in S. The monk dataset is a binary classification problem (class 0 or 1) and therefore equation (1) simplifies to

$$\text{Entropy}(S) = -p_0 \log_2 p_0 - p_1 \log_2 p_1 \tag{2}$$

where p_0 and $p_1 = 1 - p_0$ are the proportions of examples belonging to class 0 and 1.

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.

Answer:Using the provided function for entropy: dtree.entropy(dataset). Results are rounded to 4 figures.

Dataset	Entropy
MONK-1	1.0000
MONK-2	0.9571
MONK-3	0.9998

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

Answer:
Entropy is a measure of the amount of uncertainty, which qualifies the expected amount of information needed to describe the state of the variable.
For a **uniform distribution**, all outcomes are equally likely. This corresponds to the case of maximum uncertainty, and therefore entropy is maximized. Specifically, it equals to $log_2(N)$, where N is the sample size. For example, tossing a fair coin ($p(\text{heads}) = 0.5$, $p(\text{tails}) = 0.5$) has an entropy of $H = -0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1$ bit.
For a **non-uniform distribution**, some outcomes are more likely than others. This reduces uncertainty, and thus the entropy is lower.
In general:

- **High entropy example:** a fair coin (both $p = 1/2$ for head and tail) with entropy $\log_2 2 = 1$ bit.
- **Low entropy example:** a very biased coin, e.g. $p(\text{heads}) = 0.99$, $p(\text{tails}) = 0.01$, where the entropy is only ≈ 0.08 bits. An extreme case is a **certain event**, such as a biased coin that always comes up heads ($p(\text{heads}) = 1, p(\text{tails}) = 0$), which has entropy $H = 0$.

Thus, entropy is maximized when the distribution is uniform (maximum uncertainty), and minimized when the distribution is highly skewed or deterministic (minimum uncertainty).

4 Information Gain

The information gain measures the expected reduction in impurity caused by partitioning the examples according to an attribute. It thereby indicates the effectiveness of an attribute in classifying the training data. The information gain of an attribute A, relative to a collection of examples S is defined as

$$\text{Gain}(S, A) = \text{Entropy}(S) - \sum_{k \in \text{values}(A)} \frac{|S_k|}{|S|} \text{Entropy}(S_k) \tag{3}$$

where S_k is the subset of examples in S for the attribute A has the value k.

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?

Answer:

Information Gain

Dataset	a_1	a_2	a_3	a_4	a_5	a_6
`MONK-1`	0.0753	0.0058	0.0047	0.0263	**0.2870**	0.0008
`MONK-2`	0.0376	0.0025	0.0011	0.0029	**0.0173**	0.0062
`MONK-3`	0.0071	**0.2937**	0.0008	0.0029	0.2559	0.0071

Choose the attribute of the most information gain as follows:

- For MONK-1 use a_5 with avgerage information gain of 0.2870
- For MONK-2 use a_5 with avgerage information gain of 0.0173
- For MONK-3 use a_2 with avgerage information gain of 0.2937

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.

Answer:

$$\text{Gain}(S, A) = \text{Entropy}(S) - \sum_{k \in \text{values}(A)} \frac{|S_k|}{|S|} \text{Entropy}(S_k)$$

According to Eq. 3, maximizing the information gain means that the weighted entropy of the subsets S_k is minimized. Entropy measures the impurity or uncertainty of a dataset: low entropy corresponds to purer subsets (dominated by one class), while high entropy indicates mixed classes.
Therefore, using information gain as a heuristic ensures that we select the attribute which most reduces the overall uncertainty about the class labels after splitting. In practice, this means the chosen attribute is the most effective at separating the data into more homogeneous groups, which directly improves the classification power of the decision tree.

5 Building Decision Trees

Split the `monk1` data into subsets according to the selected attribute using the function `select` (again, defined in `dtree.py`) and compute the information gains for the nodes on the next level of the tree. Which attributes should be tested for these nodes?

Subset	A1	A2	A3	A4	A5	A6
A5=1	0	0	0	0	0	0
A5=2	0.0402	0.0151	0.0373	**0.0489**	0	0.0258
A5=3	0.0331	0.0022	0.0180	0.0191	0	**0.0451**
A5=4	0.**2063**	0.0339	0.0259	0.0759	0	0.0033

Table 3: Information gain values for different subsets of A_5.

For the `monk1` data draw the decision tree up to the first two levels and assign the majority class of the subsets that resulted from the two splits to the leaf nodes. You can use the predefined function `mostCommon` (in `dtree.py`) to obtain the majority class for a dataset.

***Output using* mostCommon:**

```
--- A5 == 1:
True
--- A5 == 2:
A4 == 1:    A4 == 2:    A4 == 3:
False       False       False
--- A5 == 3:
A6 == 1:    A6 == 2:
```

```
False       False
--- A5 == 4:
A1 == 1:    A1 == 2:    A1 == 3:
False       False       True
```

Now compare your results with that of a predefined routine for ID3. Use the function `buildTree(data, m.attributes)` to build the decision tree. If you pass a third, optional, parameter to `buildTree`, you can limit the depth of the generated tree.

You can use `print` to print the resulting tree in text form, or use the function `drawTree` from the file `drawtree_qt4.py` or `drawtree_qt5.py`, depending on your PyQt version, to draw a graphical representation.

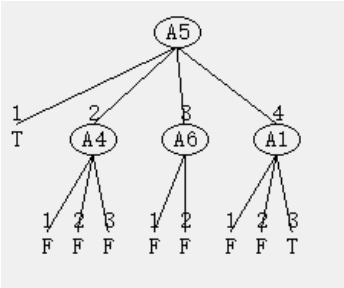


Figure 1: 2-Layer Decision Tree for MONK-1

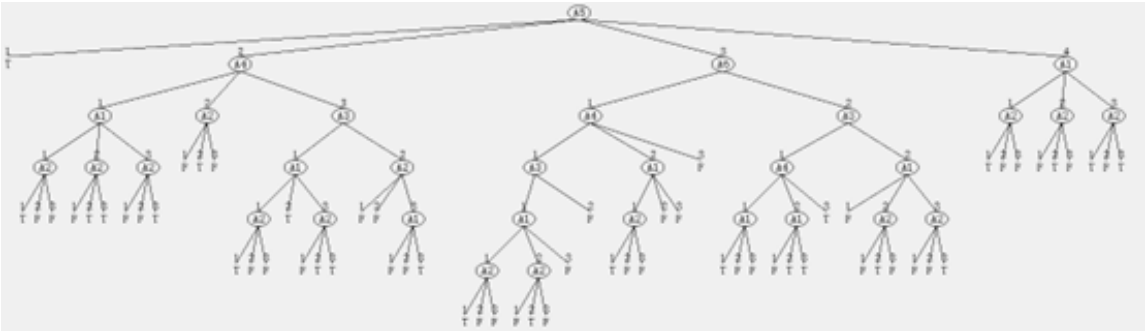


Figure 2: 6-Layer Decision Tree for MONK-1

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.

Answer:

	E_{train}	E_{test}
MONK-1	0	0.1713
MONK-2	0	0.3079
MONK-3	0	0.0556

For all three datasets, the training error is zero. This is expected, because the full decision trees keep splitting until every training sample is correctly classified (overfitting the training data).

- MONK-2: Training error is zero, but test error is the highest ($\approx 31\%$), showing severe overfitting. This matches our assumption that MONK-2 is the hardest dataset for the decision tree.
- MONK-1: Test error of 17% is moderate, because the greedy decision tree algorithm grows unnecessary branches and slightly overfits.
- MONK-3: However, MONK-3 actually shows the smallest error of ($\approx 5\%$, close to the noise level) on testing data, which shows the initial assumptions were not correct. The reason might be that the best attribute for MONK-3 a_2 provides the most information gain. And to compare the figures of all the decision trees, we can see that MONK-3 has the smallest decision tree with the least leaf nodes.

6 Pruning

The idea of reduced error pruning is to consider each node in the tree as a candidate for removal. A node is removed if the resulting pruned tree performs at least as well as the original tree over a separate validation dataset, i.e. a dataset not used during training. When a node is removed, the subtree rooted at that node is replaced by a leaf node, to which the majority classification of examples in that node is assigned.

For the purpose of pruning, we have to split our original training data into one training set for building the tree and one validation set for pruning. Notice, that using the test set for validation would be cheating because we would then no longer be able to use the test set for independently estimating the true error of our pruned decision tree. Instead, we will randomly partition the original training set into training and validation set. This can be done by defining a function which randomly reorders the data samples and returns the first and second parts separately:

```
import random

def partition(data, fraction):
    ldata = list(data)
    random.shuffle(ldata)
    breakPoint = int(len(ldata) * fraction)
    return ldata[:breakPoint], ldata[breakPoint:]

monk1train, monk1val = partition(m.monk1, 0.6)
```

In the file `dtree.py` there is a utility function `allPruned` which returns a sequence of all possible ways a given tree can be pruned.

Write code which performs the complete pruning by repeatedly calling `allPruned` and picking the tree which gives the best classification performance on the validation dataset. You should stop pruning when all the pruned trees perform worse than the current candidate.

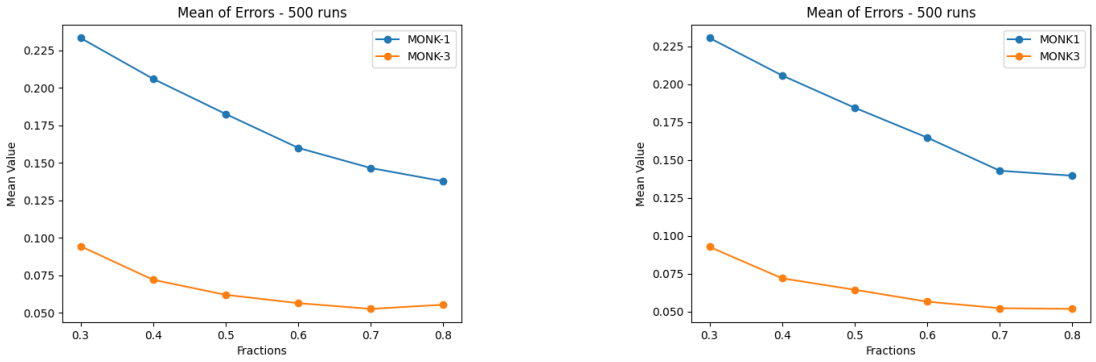
Assignment 6: Explain pruning from a bias variance trade-off perspective.

Answer:
Pruning prevents overfitting by deleting unnecessary nodes of the tree. By pruning the tree, the model complexity and the variance of the result will be reduced, which reflects the enhancement in terms of robustness and compatibility. However, the bias and variance are negative to each other: reducing the variance inevitably increases the bias.

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.

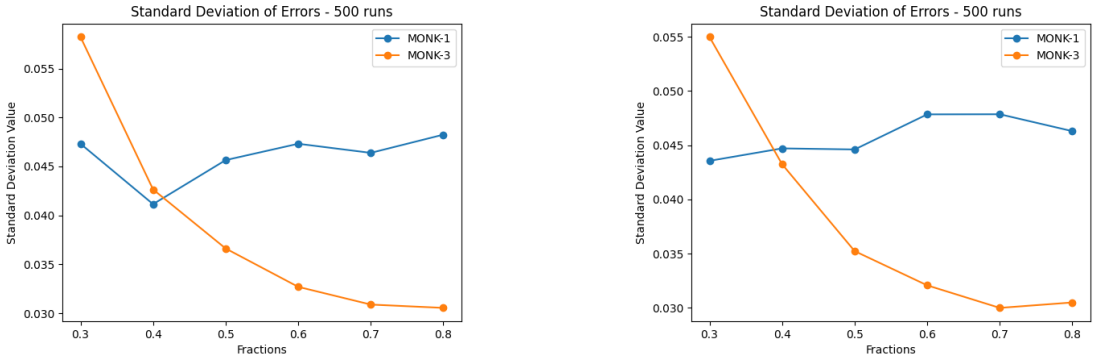
Answer: To start with, the results in assignment 7 is random due to the split of data.



(a) (b)

Figure 3: Mean of errors over 500 runs

In Figure 3(a), the mean error over 500 runs for MONK-1 is minimized at a partition of 0.8, while for MONK-3 the lowest mean error occurs at 0.7. However, it is worth noting that MONK-3 sometimes also achieves its minimum at 0.8, as shown in in Figure 3(b). Overall, the difference in mean error between partitions of 0.7 and 0.8 for MONK-3 is relatively small.



(a) (b)

Figure 4: Standard deviation values over 500 runs

Compared with the mean value of MONK-1, the pattern for the standard deviation of MONK-1 is less obvious. As shown in Figure 4(a), at small training fractions ($0.3 \sim 0.5$), the standard deviation might be relatively smaller. But overall, the standard deviation is stable to some extent.

For MONK-3, at small training fractions ($0.3 \sim 0.6$), the standard deviation is relatively high. As training fraction grows ($0.7 \sim 0.8$), the standard deviation becomes very low, meaning the model performance is stable.

So the optimal partition for MONK-1 and MONK-3 can be 0.8 for better performance in terms of the mean value and standard deviation.