DD2431 Machine Learning
Lab 1: Decision Trees
Python version

Örjan Ekeberg

September 10, 2015

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 for plotting. It is possible to do the lab without using the plotting functions, 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. Each one of the datasets has properties which makes them hard to learn. Can you guess which of the three problems is most difficult for a decision tree algorithm to learn?

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

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 (misclassifications) 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

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.

Dataset	Entropy
MONK-1	
MONK-2	
MONK-3	

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 2: 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]`.

Information Gain

Dataset	a_1	a_2	a_3	a_4	a_5	a_6
MONK-1						
MONK-2						
MONK-3						

Based on the results, which attribute should be used for splitting the examples at the root node?

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?

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.

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.py` to draw a graphical representation.

Assignment 3: 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.

	E_{train}	E_{test}
MONK-1		
MONK-2		
MONK-3		

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 classifiction performance on the validation dataset. You should stop pruning when all the pruned trees perform worse than the current candidate.

Assignment 4: 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` $\in \{0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$.