DD2431 Machine Learning
Lab 1: Decision Trees

Frank Hoffmann
modified by Örjan Ekeberg

September 12, 2010

1 Preparations

Before you start this lab you should be familiar with programming in MATLAB. In particular you should understand how MATLAB performs matrix computations, simple graphics, simple programming constructs, functions and the help system in MATLAB.

In this lab you will use some predefined functions for building decision trees and analyzing their performance, but you will also have to program some computations yourself.

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. 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

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

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 directory `/info/ml10/labs/dectrees/` as `monks-1.train`, `monks-1.test`, `monks-2.train`, `monks-2.test`, `monks-3.train` and `monks-3.test`. Each row contains one instance. The first column contains the target class (0 or 1), and the next six columns the attributes. You can ignore the last column containing the string $data_i$ which simply enumerates the instances.

For the sake of convenience you find a MATLAB script `readmonks.m` that loads the datasets into MATLAB.

```
>> addpath('/info/ml10/labs/dectrees');
>> readmonks
>> who
```

The data sets are loaded into MATLAB variables called `monks_1_train`, `monks_1_test` etc. Notice, that for compatibility reasons the class is stored in the last column and the first six columns contain the attributes.

3 Decision Trees

In order to decide on which attribute to split, the 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: Write a function `ent(data)` that computes the entropy of a collection of examples passed as the parameter `data`. Hint: MATLAB has a build in function, `log2`, for computing the base 2 logarithm. Use your function to compute the entropy of the three MONK *training* data sets.

Dataset	Entropy
MONK-1	
MONK-2	
MONK-3	

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. For the purpose of extracting the subset of examples use the two following MATLAB functions provided in the course directory.
`values(data,i)` returns a vector of unique values for the i-th attribute in data.
`subset(data,i,j)` returns the subset of examples in data for which attribute i has value j.

Assignment 2: Write a function `gain(data)` that computes the information gain of all the attributes of a collection of examples. Use this function to compute the information gain of the six attributes in the three MONK training data sets.

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?

Split the data into subsets according to the selected attribute 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 *monks_1_train* 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. Use the predefined function `majority_class(data)` to obtain the majority class for a set of instances.

Now compare your results with that of a predefined routine for ID3. Use the function `T=build_tree(data)` to build the decision tree and the function `disp_tree(T)` to display the tree. Notice, that in the display a *no* corresponds to class 0, *yes* to class 1. If you set the global variable `max_depth=2`, you can limit the depth of the decision tree built. Note that in order to set the `max_depth` variable you have to declare it `global` and rerun `build_tree()` after changing the value. Do not forget to set this parameter back to its original value of 10 afterwards.

Assignment 3: Build the full decision trees for all three Monk datasets using the some predefined routines for ID3.

`calculate_error(T,data)`
computes the classification error over some data.

For example to built a tree for monks_1, display it and compute the test set error for monks_1

```
>> T=build_tree(monks_1_train);
>> disp_tree(T);
```

```
>> error=calculate_error(T,monks_1_test);
```

Compute the train and test set errors for the three Monk datasets for the unpruned trees.

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

The idea of reduced error pruning is to consider each node in the tree as a candidate for pruning. A node is removed if the resulting pruned tree performs no worse than the original tree over a validation set not used during training. In that case the subtree rooted at that node is replaced by a leaf node, to which the majority classification of examples in that node is assigned.

`T=prune_tree(T,data)`
prunes a tree previously generated with `build_tree` using the examples in `data`.

For the purpose of pruning, we have to partition our original training data into a training set for building the tree and a validation set for pruning tree. Notice, that using the test set for validation would be cheating as we then are no longer able to use the test set for estimating the true error of our pruned decision tree. Therefore, we randomly partition the original training set into training and validation set.

```
>> [n,m]=size(monks_1_train);
>> p=randperm(n);
>> frac=0.7;
>> monks_1_train_new=monks_1_train(p(1:floor(n*frac)),:);
>> monks_1_prune=monks_1_train(p(floor(n*frac)+1:n),:);
>> T1=build_tree(monks_1_train_new);
>> T1p=prune_tree(T1,monks_1_prune);
```

Where the fraction `frac` of instances is used for building the tree and the remaining examples for pruning it.

Assignment 4: Evaluate the effect of pruning on the test error for the monk_1 and monk_3 datasets, in particular determine the optimal partition into training and pruning by optimizing the parameter `frac`. Plot the classification error on the test sets as a function of the parameter `frac` $\in \{0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$.