AI2 Module 3: Assignment (Part A)

(Amended by David Talbot, d.r.talbot@sms.ed.ac.uk)*
Division of Informatics

January 2005

This document describes Part A of the assignment for AI2 Module 3. The assignment must be submitted electronically as follows. You should combine together whatever files you develop for the different parts of the exercise into a single file and submit that file. The file should be in a format that can be loaded directly into Prolog (i.e. non-program text should be in comments). The file should be submitted electronically by **5pm, Friday, 28th January 2005**. Late penalties will be applied in the standard way to submissions. To submit the file use: `submit ai2 ai2bh B1 FILE` where `FILE` is the name of your file. All the files required for this part of the assignment can be found at:

`http://www.inf.ed.ac.uk/teaching/courses/ai2/module3/assessed/index.html`

If you have any problem, queries or are just stuck please feel free to email me at `d.r.talbot@sms.ed.ac.uk` or, from the 11th January 2005, find me in Rm.9, 3rd floor of 2 Buccleuch Place *during my office hours*; Tuesday 2-4 and Thursday 11-12.

1 Learning Decision Trees

For this assignment you should implement a version of the Decision Tree Learning Algorithm (see figure 18.5 of Russell&Norvig reprinted on slide 2-21). The work is divided into smaller parts so as to make the task easier. The various sub-parts can be done independently. Note that your code should accompanied by comments explaining it. Part of the marks for the exercise will be given to good documentation.

There are two data sets associated with this assignment. One describes the restaurant domain of tutorial 2 with the same example set. The other describes data collected on animals at a zoo. Given the information collected on each animal the problem is to determine whether it is a mammal (yes) or not (no).

*Written by Paul Crook based on material by Roni Khardon

The data files are `rest.pl` and `zoo.pl` respectively. You may find the restaurant example easier to use while debugging your programs but should submit results as instructed below. Two other files are provided in the same directory: `decision.pl` and `tests.pl`. The contents of `decision.pl` are explained below. The file `tests.pl` includes code for tests that you have to run after completing each part of the program. Instructions on submission of tests are given below.

You should use SICStus Prolog on the dai machines to run this code. To run SICStus Prolog type `sicstus` at UNIX command prompt, you should then be presented with the Prolog prompt `| ?-`. Once in Prolog you will need to load (consult) the `tests.pl` file, `decision.pl` file, the appropriate data file, either `rest.pl` or `zoo.pl`, as well as your code, before you can run the tests. To consult a file type `[FILE].` — including the full stop — at the Prolog prompt, where FILE is the file name without the `.pl` part, e.g. `[tests].` .

NB ensure that you have consulted the correct data set for each test, the test routines do not check this. If you have to run tests first on the restaurant data set and then on the zoo data set, it is probably worth halting the Prolog interpreter (by typing `halt.`) and restarting it to ensure that the previous data file has been cleaned out.

You should assume that the training and test data can be manipulated *only* via the following interface predicates. These predicates are provided in the data files. Each training or test data item has a unique name and a set of attributes and values.

- `attributes(?AttrList)` - succeeds with `AttrList` the list of attributes used in the data.
- `classes(-ClassesList)` - succeeds with `ClassList` the list of possible classification results.
- `values(+Attr,-ValueList)` - succeeds if the list of possible values for attribute `Attr` is `ValueList`.
- `training_data(-NameList)` - succeeds with `NameList` the list of names of all training data items.
- `test_data(-NameList)` - succeeds with `NameList` the list of names of all test data items.
- `attr_value(+Attr,+Name,?Value)` - succeeds if for the data item with name `Name` the value of attribute `Attr` is `Value`.
- `class(+Name,-Class)` - succeeds if the data item with name `Name` is classified with class `Class`.

Several additional predicates are provided in the file `decision.pl` You should study these as they might help you in the implementation. A brief description is:

- `all_have_class(+Names,?Class)` - succeeds if the data items whose names form the list `Names` are all in class `Class`.
- `majority_value(+Names,-Class)` - succeeds if `Class` is the class of the majority of the data items whose names form the list `Names`.

`member(?X,?L)` - the standard predicate.
`delete(+Item,+List,-DelList)` - succeeds with `DelList` including all elements in List but `Item`.
`val_split(+Attr,+Value,+Examples,-SExamples)` - succeeds if `Examples` is a list of example names, `SExamples` is the subset of `Examples` for which the attribute `Attr` has value `Value`.
`entropy(Examples,I)` - succeeds if `Examples` is a list of example names, and `I` is the value of the entropy in `Examples` using the formula on slide 2-33.
`number(+Goal,-Num)` - succeeds if the number of solutions of goal `Goal` is `Num`.

The operation of `number` is shown in the following example:

```
age(john,25).
age(mary,26).
age(fred,25).
age(sue,27).

?- number(age(X,25),N).
N = 2
```

2 Decision Tree Representation (15%)

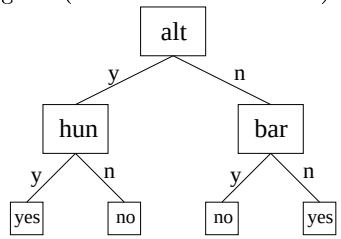
We represent decision trees using Prolog terms of the form:

`t(Attribute,ListOfValuesAndSubTrees)`

The `ListOfValuesAndSubTrees` will include each possible value for the attribute exactly once. Each element in the list has the form `Value-SubTree` or `Value-Leaf`. A leaf of the tree is represented by the labels `yes` or `no`. For example, the term:

```
t(alt, [
   y-t(hun, [y-yes, n-no ]),
   n-t(bar, [y-no,  n-yes])
      ]).
```

corresponds to the following tree (for the restaurant domain):



Write a predicate

`tree_size(+Tree,-Size)` - succeeds if `Size` is the number of leaves in `Tree`.

Here, as well as in other parts, you may assume that the input (here a tree) is in the right format. You may find using the cut (!) useful.

For this part (Part 2) you should submit the code plus the results of running `test1a`, `test1b` from the test file in the restaurant domain. Assuming you have consulted the `tests.pl` file, type `test1a.` at the Prolog prompt to run test1a.

Note: For recording the output when a test is run you may either cut and paste or alternatively use the unix `script` command. When you use `script filename`, the system saves everything appearing on the screen to the file `filename`. Start script before starting Prolog. You must end the command by pressing `Control-D` after halting Prolog. You can then cut and paste from the file.

3 Classifying Examples (25%)

Write three predicates as follows:

`evaluate(+Tree,+Ex,?Class)` - succeeds if `Class` is the classification given by `Tree` to the example named `Ex`. (`Class` is the leaf value found for the example `Ex` by following the branches of the given Tree; it is not necessarily the same as the example's classification in the data set. You can assume that `Ex` is the name of one of the examples in the currently load data set.)
`tree_correct(+Tree,+Ex)` - succeeds if the class given by the tree to the example is the correct class for the example `Ex`.
`testtree(+Tree)` - always succeeds. It tests the tree against all the test examples and as a side effect prints the percentage of examples classified correctly.

For this part (Part 3) you should submit the code plus the results of running `test2a`, `test2b`, `test2c`, `test2d`, `test2e` from the test file in the restaurant domain.

4 Learning Algorithm (25%)

In this part you should implement a simple version of the learning algorithm. You should write your program in a modular way so that it uses the following predicate in order to choose which attribute to split on at each stage.

`choose_attribute(+Mode,+Attrs,+Examples,-Best,-Rem)` - succeeds if, in order to classify the examples whose names form the list `Examples`, the best attribute to choose from the list `Attrs` is `Best`, and the remaining attributes form the list `Rem`. `Mode` is either `simple` or `gain`.

For this part (Part 4), we define a simple `choose_attribute/5` predicate to allow testing of the learning algorithm code. The `simple` version of `choose_attribute/5` just selects the first attribute out of the given list as the best. It can be defined by:

```
choose_attribute(simple,[First|Remaining],_,First,Remaining).
```

The top-level interface to your program should be the predicate:

`learntree(+Mode,-Tree)` - succeeds if the decision tree resulting from the learning algorithm applied to the training data is `Tree`. It should work correctly with any implementation of `choose_attribute` that takes `Mode` as a parameter.

For the learning algorithm itself you may find it useful to follow the structure of figure 18.5 of Russell&Norvig using 4 clauses for the algorithm, 3 dealing with the base cases and the final one with the recursive case.

Branches should be included even if no examples exist for a particular attribute value. You should of course never have any leaves that are the empty list, each leaf should contain the *classification* of the examples at that node. When there are no examples for a particular attribute value, then the leaf's value should be the classification of the majority of the examples at the parent node (unless there's no such majority, in which case take a random decision whether the class is yes or no).

For this part (Part 4) you should submit the code plus the results of running `test3r` in the restaurant domain and `test3z` in the zoo domain. *NB Ensure that you have consulted the appropriate data files for each test.*

5 Choosing Attribute Using Information Gain (25%)

In this part you should implement the information gain heuristic for choosing attributes. You may want to make use of the `entropy` predicate described above. Write three predicates as follows:

`remainder(+Attr,+Examples,-R)` - succeeds with `R` having the value of the *remainder*() formula (as described on Slide 2-34) for attribute `Attr` for the set of examples whose names form the list `Examples`.

`gain(+Attr,+Examples,-Gain)` - succeeds with `Gain` having the value of the *Gain*() formula as described on Slide 2-34. `Examples` and `Attr` are as in `remainder`.

`choose_attribute(gain,AttrList,Examples,Best,Remaining)` -
a version of `choose_attribute` that succeeds when Best is the attribute that should be chosen according to the information gain heuristic (as described on slide 2-34).

For this part (Part 5) you should submit the code plus the results of running `test4ra, test4rb, test4rc, test4rd, test4re` in the restaurant domain and `test4za, test4zb, test4zc, test4zd` in the zoo domain.

6 Performance (10%)

Write a paragraph in which you discuss the accuracy of the categorisation learnt by the decision tree. If you were faced with a problem where a decision tree has not performed well, discuss the possible reasons for this, including ways in which you might be able to improve the performance.

Include you answer to this part (Part 6) as commented lines at the end of the file that you submit.