Decision Tree: nagdmc_entropy_tree

Purpose

nagdmc_entropy_tree computes an n-ary classification tree by using an entropy criterion.

Declaration

```
#include <nagdmc.h>
void nagdmc_entropy_tree(long rec1, long nvar, long nrec, long dblk, double data[],
                         long nxvar, long xvar[], long yvar, long ncat[],
                         long cat[], long mnc, long *iproot, int *info);
```

Parameters

1: **rec1** – `long` *Input*

On entry: the index in the data of the first data record used in the analysis.

Constraint: **rec1** ≥ 0.

2: **nvar** – `long` *Input*

On entry: the number of variables in the data.

Constraint: **nvar** > 1.

3: **nrec** – `long` *Input*

On entry: the number of consecutive records, beginning at **rec1**, used in the analysis.

Constraint: **nrec** > 1.

4: **dblk** – `long` *Input*

On entry: the total number of records in the data block.

Constraint: **dblk** $\geq$ **rec1** $+$ **nrec**.

5: **data**[**dblk** $*$ **nvar**] – `double` *Input*

On entry: the data values for the jth variable (for $j = 0, 1, \ldots,$ **nvar**-1) are stored in **data**[$i*$**nvar**$+j$], for $i = 0, 1, \ldots,$ **dblk** $- 1$.

6: **nxvar** – `long` *Input*

On entry: the number of independent variables. If **nxvar** $= 0$ then all variables in the data, excluding **yvar**, are treated as independent variables.

Constraint: $0 \leq$ **nxvar** $<$ **nvar**.

7: **xvar**[**nxvar**] – `long` *Input*

On entry: the indices indicating the position in **data** in which values of the independent variables are stored. If **nxvar** $= 0$ then **xvar** must be 0, and the indices of independent variables are given by $j = 0, 1, \ldots,$ **nvar** $- 1$; $j \neq$ **yvar**.

Constraints: if **nxvar** > 0, $0 \leq$ **xvar**[i] $<$ **nvar**, for $i = 0, 1, \ldots,$ **nxvar** $- 1$; otherwise **xvar** must be 0.

8: **yvar** – `long` *Input*

On entry: the index in **data** in which values of the dependent variable are stored.

Constraints: $0 \leq$ **yvar** $<$ **nvar**; if **nxvar** > 0, **yvar** $\neq$ **xvar**[i], for $i = 0, 1, \ldots,$ **nxvar** $- 1$.

9: **ncat**[**nvar**] – `long` *Input*

On entry: **ncat**[i] contains the number of categories in the ith variable, for $i = 0, 1, \ldots,$ **nvar** $- 1$. If the ith variable is continuous, **ncat**[i] must be set equal to zero.

Constraints: **ncat**[i] ≥ 0, for $i = 0, 1, \ldots,$ **nvar** $- 1$, ($i \neq$ **yvar**); **ncat**[**yvar**]> 1.

10: **cat**[d] – `long` *Input*

On entry: a list of category values for variables in the same variable order as **ncat**, where d is equal to the sum of the elements in **ncat**.

11: **mnc** – `long` *Input*

On entry: during the search for an optimal partition of data at a node each candidate partition must contain at least **mnc** data records.

Constraint: **mnc** $\leq$ **nrec**/2.

12: **iproot** – `long *` *Output*

On exit: **iproot** is an integer cast of the memory location pointing to the root node in the tree. This value is passed to the functions described in 'See Also'. Information on the detail of a decision tree can be found by using the value of **iproot**.

Detail of partitions in an entropy classification tree are available by using in a C program the code:

```
ENode *proot;
proot = (ENode *)iproot;
```

where `ENode` is a C structure with the following members:

`nchildren` – `long`

the number of child nodes belonging to the node;

`ndata` – `long`

the number of data records at this node;

`nclasses` – `long`

the number of categories on the dependent variable;

`ninclasses` – `long []`

`ninclasses`[k] gives the number of data records at the node in kth category of the dependent variable, for $k = 0, 1, \ldots,$ `nclasses` $- 1$;

`modal_class` – `long`

the modal category of the dependent variable over data records at the node;

`index` – `long`

the index in the data used to partition data records;

`discrete` – `long`

if `discrete` $= 1$, the variable described by `index` is categorical; otherwise the variable is continuous;

`value` – `double`

the value of the test if the variable used to partition data records is continuous; otherwise not referenced;

`children` – `ENode **`

if this node is not a leaf node, `children`[i] points to the ith child node, for $i = 0, 1, \ldots,$ `nchildren` $- 1$; otherwise `children` is set to 0;

`parent` – `ENode *`

if this node is not the root node, a pointer to the parent node; otherwise `parent` is set to 0.

A C source code example that accesses the information in this classification tree is given in 'Explanatory Code'.

13: **info** – `int *` *Output*

On exit: **info** gives information on the success of the function call:

0: the function successfully completed its task.

i; $i = 1, 2, \ldots, 4, 6, 7, \ldots, 9, 11$: the specification of the ith formal parameter was incorrect.

99: the function failed to allocate enough memory.

100: an internal error occurred during the execution of the function.

Notation

nrec the number of data records, n.
nxvar determines the number of independent variables, p.
ncat contains the number of category values on variables, c and c_j, for $j = 1, 2, \ldots, p$.
mnc the minimum number of data records at child nodes, s.

Description

Let x_i denote the values of p independent variables and y_i the value of the dependent variable for the ith data record at a node A in a classification tree, for $i = 1, 2, \ldots, n$. The jth independent variable can be continuous or categorical and its ith value is denoted by x_{ij}, for $j = 1, 2, \ldots, p$. If the jth independent variable is categorical it takes c_j category values, for $j = 1, 2, \ldots, p$. The dependent variable is a categorical variable with c category values.

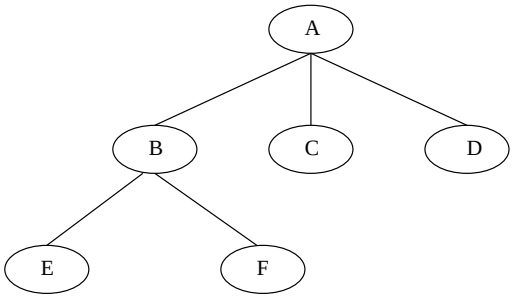

Figure 1: *Graphical representation of an n-ary tree showing parent nodes connected by lines to their child nodes. The root node, node A, is associated with all data records and is the only node not to have a parent node. Nodes C, D, E and F do not have child nodes and are known as leaf nodes. Node B is neither the root node nor a leaf node and is known as an internal node. Given a positive value for the scalar s, a partition of $m \geq s$ data records at a parent node into records at child nodes is based on the outcome of a test at the parent node. In this example the test at node A is on a categorical variable with three category values, and the test at node B is either a test on a continuous variable or on a categorical variable with two category values.*

Consider the case of partitioning m data records in a set X at a parent node such that each data record in X is sent to a single child node (see Figure 1). If m is less than a user-supplied value s, a partition of data is not computed; otherwise a data partition is defined by computing a univariate test on an independent variable. Two kinds of test are available. For a discrete variable j with r unique values $u_1, u_2, \ldots, u_r$, a criterion is evaluated for a partition of X into:

$$\{X_1, X_2, \ldots, X_k, \ldots, X_r\},$$

where X_k contains those members of X that satisfy:

$$x_{ij} = u_k, \quad i = 1, 2, \ldots, m; \; k = 1, 2, \ldots, r.$$

For a continuous variable j a criterion is evaluated for each possible binary partition of X into X_1 and X_2, where X_1 contains those members of X that satisfy:

$$x_{ij} \leq t_l, \quad i = 1, 2, \ldots, m.$$

and X_2 contains those members of X that satisfy:

$$x_{ij} > t_l, \quad i = 1, 2, \ldots, m,$$

and t_l is found by sorting the data values on variable j and computing the mid-point of neighbouring ordered values, for $l = 1, 2, \ldots, m - 1$. The partition which optimises a criterion is retained.

The process of computing partitions continues recursively for each node created until the number of data records at a node is less than s or the data records at a node are all the same.

The criterion used to determine partitions is the gain ratio which is based on values of entropy. In particular, the entropy of X is:

$$E(X) = -\sum_{i=1}^{c} \frac{u_i}{m} \log_2 \left(\frac{u_i}{m}\right),$$

where u_i is the number of data records in X that belong to the ith category. Now consider a partition of X into disjoint and exhaustive sets $X_1, X_2, \ldots, X_r$; the weighted average of entropy of each partition is given by,

$$I(X_1, X_2, \ldots, X_r) = \frac{1}{m} \sum_{i=1}^{r} |X_i| E(X_i),$$

and the gain ratio R by:

$$R = \frac{E(X) - I(X_1, X_2, \ldots, X_r)}{S(X_1, X_2, \ldots, X_r)},$$

where $S(\cdot)$ is the split information:

$$S(X_1, X_2, \ldots, X_r) = \frac{1}{m} \sum_{i=1}^{r} |X_i| \log_2 \left(\frac{|X_i|}{m}\right).$$

References and Further Reading

Quinlan J R (1987) Simplifying decision trees *Int. J. Man-Machine Studies* **27** 221–234

Quinlan J R (1993) *C4.5: Programs for Machine Learning* San Mateo: Morgan Kaufmann

Explanatory Code

The following C function prints the memory locations of nodes in a tree and its parent node. The type (leaf or internal) of each node is printed along with the detail of the partition at that node. If the function is called with **iproot** as its argument, the entire tree is printed.

```
#include <stdio.h>

void step_through(long node) {
    long i;
    ENode *lnode;

    lnode = (ENode *)node;

    if (lnode == 0)
        return;

    printf("\n Node %8p"
           "\n Parent %8p"
           "\n No. data at node %8li"
           "\n Modal class %li",
           lnode,lnode->parent,lnode->ndata,lnode->modal_class);

    printf("\n Class distribution at node:");

    for (i=0; i<lnode->nclasses; ++i)
        printf(" %li",lnode->ninclasses[i]);

    if (lnode->index > 0) {
        printf("\n Partition on index %li",lnode->index);

        if (lnode->discrete == 0)
            printf("; continuous test value: %8.4f",lnode->children[0]->value);

        printf("\n");

        for (i=0; i<lnode->nchildren; ++i)
            step_through((long)(lnode->children[i]));
    }
    else
        printf("\n Leaf node\n");
}
```

See Also

nagdmc_free_entropy_tree	returns to the operating system memory used by an entropy tree.
nagdmc_load_entropy_tree	loads an entropy tree from a file.
nagdmc_predict_entropy_tree	computes predictions given an entropy tree.
nagdmc_prune_entropy_tree	prunes an entropy tree using pessimistic error pruning.
nagdmc_save_entropy_tree	writes an entropy tree to a file.
entropy_tree_ex.c	the example calling program.