Decision Tree: nagdmc_gini_tree

Purpose

nagdmc_gini_tree classifies data by using a binary tree computed using the Gini index criterion.

Declaration

```
#include <nagdmc.h>

void nagdmc_gini_tree(long rec1, long nvar, long nrec, long dblk, double data[],
                      long nxvar, long xvar[], long yvar, long ncat[], long bcat[],
                      double prior[], long mns, long mnc, double alpha,
                      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: $\mathbf{rec1} \geq 0$.

2: **nvar** – `long` *Input*

On entry: the number of variables in the data.

Constraint: $\mathbf{nvar} > 1$.

3: **nrec** – `long` *Input*

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

Constraint: $\mathbf{nrec} > 1$.

4: **dblk** – `long` *Input*

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

Constraint: $\mathbf{dblk} \geq \mathbf{rec1} + \mathbf{nrec}$.

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

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

6: **nxvar** – `long` *Input*

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

Constraint: $0 \leq \mathbf{nxvar} < \mathbf{nvar}$.

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

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

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

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

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

Constraints: $\mathbf{ncat}[i] \geq 0$, for $i = 0, 1, \ldots, \mathbf{nvar} - 1, (i \neq \mathbf{yvar})$; $\mathbf{ncat}[\mathbf{yvar}] > 1$.

10: **bcat**[**nvar**] – `long` *Input*

On entry: **bcat**[i] contains the base level value for the **ncat**[i] categories on the ith variable. If **ncat**[i] > 0, for $i = 0, 1, \ldots,$ **nvar** $- 1$, the categorical values on the ith variable are given by **bcat**[i] $+ j$, for $j = 0, 1, \ldots,$ **ncat**[i] $- 1$; otherwise **bcat**[i] is not referenced. If the base level for each categorical variable is zero, **bcat** can be 0.

11: **prior**[c] – `double` *Input*

On entry: **prior**[i] contains the prior weight on the ith category value on the dependent variable, for $i = 0, 1, \ldots, c - 1$, where $c =$ **ncat**[**yvar**]. If **prior** is not 0, an equal weighting is put on each of the category values on the dependent variable.

Constraint: if **prior** is not 0, the elements in **prior** must sum equal to 1.0.

12: **mns** – `long` *Input*

On entry: if the number of data records at a node is greater than or equal to **mns**, a partition of data is attempted; otherwise a leaf node is forced.

Constraint: $1 <$ **mns** $<$ **nrec**.

13: **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: $1 \leq$ **mnc** $\leq$ **mns**$/2$.

14: **alpha** – `double` *Input*

On entry: if the decrease in misclassification rate due to partitioning data at a parent node into its child nodes is less than **alpha**, the parent node is forced to be a leaf node.

Constraint: $0.0 \leq$ **alpha** < 1.0.

15: **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 a binary classification tree are available by using in a C program the code:

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

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

`type` – `int`

if this node is a leaf, `type` is set to one; otherwise `type` is set to 0;

`ndata` – `long`

the number of data records at this node;

`nig` – `long []`

`nig`[k] gives the number of data records at the node in category **bcat**[**yvar**]$+k$ of the dependent variable, for $k = 0, 1, \ldots,$ **ncat**[**yvar**] $- 1$;

`yval` – `long`

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

`parent` – `CTNode *`

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

If `type` = 1, the remaining structure members are set equal to dummy values; otherwise the following information is available:

`svar` – `long`

the index in the data of the variable on which records are partitioned;

`ncats` – `long`

if independent variable **svar** is categorical, the number of categories on variable j^*; otherwise zero;

`sval – double`

if `ncats` $= 0$, `sval` gives the scalar value of the test on variable `svar`; otherwise `sval` is not referenced;

`lr – char []`

if `ncats` $= 0$, `lr` is not referenced; otherwise it is an array of `ncats` elements, the value of `lr[i]` determines the direction in the binary tree taken by data records at the node with category **bcat**[`svar`] $+ i$ on variable `svar`, for $i = 0, 1, \ldots,$ `ncats` $- 1$. The possible values for `lr`[i] are:

- `'l'` data records at the node with category value **bcat**[`svar`] $+ i$ on `svar` are sent to the left child node;
- `'r'` data records at the node with category value **bcat**[`svar`] $+ i$ on `svar` are sent to the right child node.
- `'a'` the ith category on `svar` is absent at this node.

`giv – double`

the Gini index criterion value;

`improve – double`

the improvement in Gini index value obtained by partitioning data records at this node;

`lchild – CTNode *`

a pointer to left child node;

`rchild – CTNode *`

a pointer to right child node.

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

16: **info** – `int *` *Output*

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

- 0: the function successfully completed its task.
- i; $i = 1, 2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 14$: 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 records, p.
nxvar	the number of variables, m.
ncat	the number of categories on variables, c_y and c_j, for $j = 1, 2, \ldots, m$.
bcat	the base level categories, b_y and b_j, for $j = 1, 2, \ldots, m$.
mns	the minimum number of records, s, required for a partition to be attempted.
mnc	the minimum number of records, t, at each child.
alpha	the pruning constant, α.

Description

Let x_i denote the values of m independent variables and y_i the value of the dependent variable for the ith data record at a node A, for $i = 1, 2, \ldots, p$. The jth independent variable can be continuous or categorical and its ith value is denoted by x_{ij}, for $j = 1, 2, \ldots, m$. If the jth independent variable is categorical it takes the c_j consecutive values $b_j, b_j + 1, \ldots, b_j + c_j - 1$, for a base level value b_j. The dependent variable is a categorical variable with c_y consecutive values $b_y, b_y + 1, \ldots, b_y + c_y - 1$, for a base level value b_y. Furthermore, let o denote the modal category and l_k be the number of records that belong to the kth category, for $k = 1, 2, \ldots, c_y$, over the values of the dependent variable at node A.

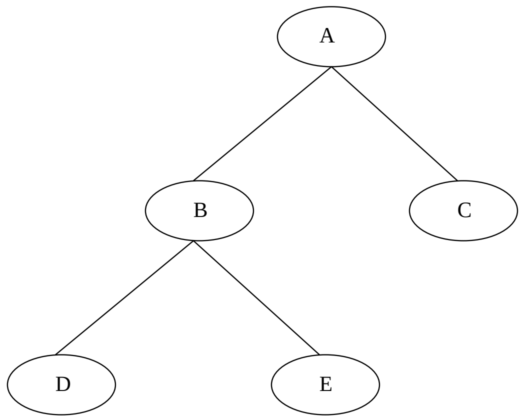

Figure 1: *Graphical representation of a binary tree showing parent nodes connected by lines to their child nodes. The root node, node A, is associated with all data records and is the only node not to have a parent node. Nodes C, D and E do not have child nodes and are known as leaf nodes. Node B is neither the root node nor a leaf node and is known as an internal node. Given positive values for the scalars s and* $t \leq s/2$*, a partition of* $p \geq s$ *data records at a parent node into* $q \geq t$ *records at one child node and* $r \geq t$ *records at the other child node is based on the outcome of a test at the parent node.*

Consider the case of partitioning p data records at a parent node A into child nodes B and C such that each record at node A is sent to either node B or node C (see Figure 1). Let s be the minimum number of data records at a parent node required to partition data. If $p < s$, 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. Firstly, a test on a continuous independent variable j sends the ith data record at the parent node to the left child node if $x_{ij} \leq u$ and otherwise to the right child node, for a value u that minimises a criterion and sends at least t data records to left and right child nodes. Secondly, a test on a categorical independent variable j sends the ith data record at the parent node to the child node determined by the binary partition of category values that minimises a criterion and sends at least t data records to left and right child nodes. In both cases, the criterion most often used in a binary classification tree is based on the Gini index of impurity.

The test chosen at parent node A is the univariate test which partitions $p \geq s$ records at a node A into $q \geq t$ records at child node B and $r \geq t$ records at child node C and minimises the sum over the child nodes, g, of the Gini index of impurity:

$$g = 2 - \frac{1}{p}\left[q\sum_{k=1}^{c_y}(b_k)^2 + r\sum_{k=1}^{c_y}(c_k)^2\right],$$

where b_k is the probability that data at node B belongs to the kth category of the dependent variable i.e.,

$$b_k = \frac{w_k l_k}{\sum_{i=1}^{c_y} w_i l_i}, \quad k = 1, 2, \ldots, c_y,$$

where l_k is the number of records at the node that belong to the kth category of the dependent variable, and the kth weight $w_k = c_y \pi_k$ for the prior probability π_k with,

$$\left.\begin{array}{l} \pi_k \geq 0 \\ \sum_{k=1}^{c_y} \pi_k = 1 \end{array}\right\}, \quad k = 1, 2, \ldots, c_y,$$

with the probability c_k defined in a similar fashion using data records at node C, for $k = 1, 2, \ldots, c_y$.

Suppose that a data partition on independent variable $1 \leq j^* \leq m$ gives the minimum value g^* of Gini index of impurity over child nodes. The improvement, z, in the Gini index of impurity is computed by subtracting g^* from the Gini index of impurity value over node A, i.e.,

$$z = 1 - \sum_{k=1}^{c_y} (a_k)^2 - g^*,$$

where the probability a_k is defined similarly to b_k.

Once a partition of data at a parent node into left and right child nodes has been found, the process continues recursively by considering partitions of data records at child nodes. When the recursive computation process terminates, nodes are removed from a binary tree if the decrease in the rate of misclassification of the dependent variable between a child node and its parent node is less than the value of a user-supplied scalar, α. This removal of nodes is known as pruning.

References and Further Reading

Brieman L. Friedman J. Olshen R. and Stone C. (1984) *Classification and Regression Trees* Belmont Calif.

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. The test value for the Gini index of impurity is printed along with the improvement in the Gini index value obtained by partitioning data records at the node. Finally, the modal category of the dependent variable is printed followed by the number of data records at the node. If the function is called with **iproot** as its second argument, the entire tree is printed.

```
#include <stdio.h>

void step_through(long bcat[], long node) {
    long  i, j;
    CTNode  *lnode;

    lnode = (CTNode *)node;

    if (lnode == 0) return;

    printf("\n Node   %8p"
           "\n Parent %8p"
           "\n type:  %8i"
           "\n svar:  %8li"
           "\n sval:  %8.4f"
           "\n giv:   %8.4f"
           "\n imp:   %8.4f"
           "\n yval:  %8li"
           "\n ndata: %8li",
           lnode,lnode->parent,lnode->type,lnode->svar,lnode->sval,
           lnode->giv,lnode->improve,lnode->yval,lnode->ndata);

    j = 0 + (bcat != 0 ? bcat[lnode->svar] : 0);

    if (lnode->ncats > 0) {
        printf("\n lr:         ");
        for (i = 0; i < lnode->ncats; ++i) {
            if (lnode->lr[i] != ABSENT)
                printf(" Category %li goes %c;",j+i,lnode->lr[i]);
        }
        printf("\b");
    }

    printf("\n");

    step_through(bcat,(long)(lnode->lchild));
    step_through(bcat,(long)(lnode->rchild));
}
```

See Also

nagdmc_free_gini_tree	returns memory containing a binary classification tree to the operating system.
nagdmc_load_gini_tree	loads a binary classification tree into memory.
nagdmc_save_gini_tree	saves a binary classification tree to a binary file.
nagdmc_predict_gini_tree	classifies new data using a binary classification tree.
gini_tree_ex.c	the example calling program.