

# Package ‘HieRanFor’

May 27, 2015

**Type** Package

**Title** Hierarchical RandomForest

**Version** 1.0

**Date** 2015-05-14

**Author** Yoni Gavish <gavishyoni@gmail.com>

**Maintainer** Yoni Gavish <gavishyoni@gmail.com>

**Depends** randomForest, ggplot2, reshape

**Suggests** e1071, doBy

**Imports** caret, treemap, pdfCluster

**Description** Runs randomForest as the local classification algorithm for each parent node along a pre-defined hierarchical tree like class structure. Contains a predict and plot functions and calculation of various flat and hierarchical performance measures. Further contains predict and performance function for new data and performance assessment for flat classification.

**License** GPL-2

**LazyData** true

## R topics documented:

HieRanFor-package . . . . .	2
GetMultPropVotes . . . . .	5
HieFMeasure . . . . .	6
ImportanceHie . . . . .	9
JoinLevels . . . . .	10
OliveOilHie . . . . .	11
PerformanceFlatRF . . . . .	13
PerformanceHRF . . . . .	15
PerformanceNewHRF . . . . .	20
plot.HRF . . . . .	22
PlotImportanceHie . . . . .	25
predict.HRF . . . . .	27
PredictNewHRF . . . . .	29
RandomHRF . . . . .	31
RunHRF . . . . .	33
tuneRF2 . . . . .	37
<b>Index</b>	<b>39</b>

---

HieRanFor-package	<i>HieRanFor: A package for running a hierarchical randomForest analysis.</i>
-------------------	---

---

## Description

Runs randomForest as the local classification algorithm for each parent node along a pre-defined hierarchical tree like class structure. Contains a predict and plot functions and calculation of various flat and hierarchical performance measures. Further contains predict and performance function for new data and performance assessment for flat classification.

## Details

Package:	HieRanFor
Type:	Package
Version:	1.0
Date:	2015-01-27
License:	GPL-2

## Credit

This package was created as part of deliverable D3.1 of WP3 of the project:

**EU-BON: Building the European Biodiversity Observation Network –**

a 7th Framework Programme funded by the European Union under Contract No. 308454.

## Main functions

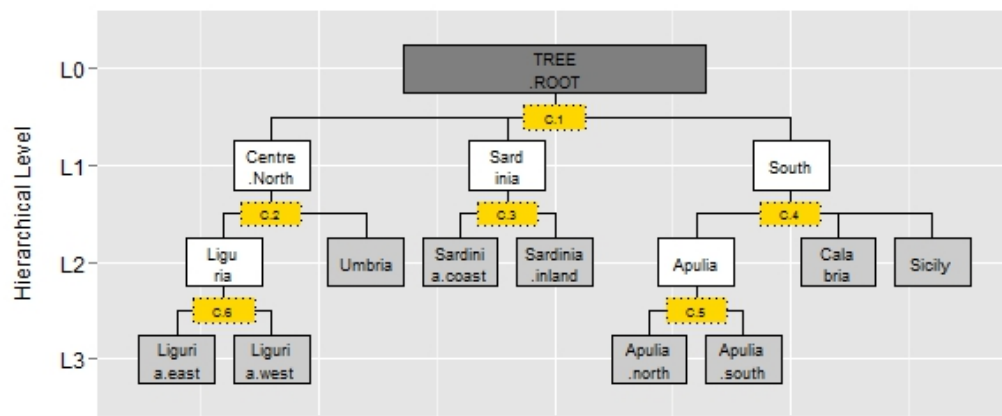
- [RunHRF](#) -  
Running a hierarchical randomForest analysis.
- [predict.HRF](#) -  
Extracting the proportion of votes for all local classifiers.
- [plot.HRF](#) -  
Plotting the class hierarchy structure.
- [ImportanceHie](#) -  
Variable importance values at each local classifier.
- [PerformanceHRF](#) -  
Assessing performance and accuracy.
- [PerformanceFlatRF](#) -  
Running a "HRF" object in a flat classifier and assessing performance.
- [PredictNewHRF](#) -  
Predicting crisp class for each case of new.data.
- [PerformanceNewHRF](#) -  
Assessing performance and accuracy of new.data.

- [HieFMeasure](#) - Information on the hierarchical performance measures.

## Definitions

- *class hierarchy* - A tree structure containing the tree root and all the internal nodes and terminal nodes. In HieRanFor, the class structure should be a directed tree with directions pointing from nodes closer to the tree root to those further down the hierarchy. Furthermore, each node may have only one parent node. See the figure below for an example of the class hierarchy of the '[OliveOilHie](#)' data-set.
- *hierarchical level*- An integer specifying the distance from the tree root. Each hierarchical level contains a set of nodes within the same distance from the tree root. Nodes in the same hierarchical level cannot be linked directly to one another in a tree like class hierarchy. e.g., 'L0', 'L1', 'L2' and 'L3' in the figure.
- *node* - A class in the class hierarchy. e.g., 'Umbria' or 'Aquila' in the figure.
- *tree root* - The node at the lowest hierarchical level of the class hierarchy. The only node that has no parent node. All other nodes are descended from the tree root node. e.g., 'TREE.ROOT' in the figure.
- *internal node* - A node that has at least one children node. e.g., 'Aquila'.
- *terminal node* - A node that has no children nodes. e.g., 'Umbria'.
- *parent node* - The linked node one level closer to the tree root from a focal node. In a tree like class structure, each internal or terminal node have a single parent node. e.g., for the focal node 'Calabria', the parent node is 'South'.
- *children node* - A linked node one level below a focal node in the class structure. All internal nodes have at least one children node. Terminal nodes have no child nodes. e.g., for the parent node 'South' the children nodes are 'Aquila', 'Calabria' and 'Sicily'.
- *sibling nodes* - All nodes that have the same parent node. e.g., 'Aquila', 'Calabria' and 'Sicily'.
- *tree depth* - The number of hierarchical levels in the class hierarchy where the tree root is considered as level 0. The tree depth of the figure below is 3.
- *path* - A linked sequence of nodes starting from the tree root, moving uni-directionally and ending at a terminal node. e.g., 'Tree.ROOT' → 'Sardinia' → 'Sardinia.inland'.
- *flat classification* - A classification in which all terminal nodes are considered to be in hierarchical level 1 (i.e., children nodes of the tree root). A flat classification will classify all the terminal nodes in a single local classifier, ignoring the class hierarchy.
- *hierarchical classification* - A classification in which at least one terminal node is from hierarchical level  $> 1$  (i.e., there is at least one internal node). In the figure below, 6 local classifiers are required to run a single hierarchical classification.

- *local classifier* -  
A single flat classification within an hierarchical classification. A local classifier is a randomForest algorithm that classifies all the children nodes of a given parent node. e.g., the local classifier 'C.6' classifies 'Liguria.east' and 'Liguria.west' -> the children nodes of 'Liguria'.
- *case* -  
A single data point in the training data or new data.
- *vote* -  
The output of a single classification tree in a single local classifier for a single case. See [randomForest](#) for more details.



### Author(s)

Written and maintained by:  
Yoni Gavish <[gavishyoni@gmail.com](mailto:gavishyoni@gmail.com)>

For reporting bugs, please use 'HieRanFor-bug:' in the subject line. For any communication regarding this package, please include 'HieRanFor' in the subject line.

### References

1. **Breiman L.** (2001) Random forests. *Machine Learning* 45:5-32.
2. **Kiritchenko S., Matwin S., and Famili F.** (2005) Functional annotation of genes using hierarchical text categorization. In: *Proc. of the ACL Workshop on Linking Biological Literature, Ontologies and Databases: Mining Biological Semantics*.

### See Also

[randomForest](#) for details on the randomForest algorithm.

---

GetMultPropVotes	<i>For each case, the multiplicative proportion of votes</i>
------------------	--

---

### Description

This function takes as input the proportion of votes that each case received in each local classifier (See: [predict.HRF](#)) and returns the multiplication of votes, along each path down the class hierarchy.

### Usage

```
GetMultPropVotes(prop.vote, unique.path, all.levels = FALSE, ...)
```

### Arguments

<code>prop.vote</code>	Data frame, the proportion of votes that each case received in each local classifier. The output of <a href="#">predict.HRF</a> .
<code>unique.path</code>	Data frame, one of the output data frames of <code>RunHRF</code> . Contains information on each path from the <code>tree.root</code> to each of the terminal nodes.
<code>all.levels</code>	Logical, if TRUE, a data frame with the predicted probabilities is returned for each level of the class hierarchy. If FALSE (default), multiplicative proportion of votes are evaluated only for the entire class hierarchy.
<code>...</code>	Optional parameters to be passed to low level functions.

### Details

The function prints an error message for cases whose sum over all terminal nodes for a given hierarchy level is different from 1 (up to 10 digits accuracy). Setting `all.levels = TRUE` enables exploration of proportion of votes, crisp classes and accuracy at specific tree depth.

### Value

The function returns a list with a single data frame if `all.levels=FALSE` or a list with length equaling the class hierarchy tree depth if `all.levels=TRUE`.

Each object within the list contains a data frame with the following columns:

- `'train.or.test'` -  
Whether the case was from the training data-set or from new.data.
- `'case.ID'` -  
The case.ID of the case.
- `other columns` -  
A column for each terminal node for the given level, containing the multiplicative proportion of votes. Values over all nodes should sum to 1.

### Author(s)

Yoni Gavish <[gavishyoni@gmail.com](mailto:gavishyoni@gmail.com)>

### See Also

[RunHRF](#) for running HRF, [predict.HRF](#) for predicting the proportion of votes, [PerformanceHRF](#) for various performance measures.

## Examples

```

set.seed(354)
# create a random training dataset
random.hRF <- RandomHRF(num.term.nodes = 20, tree.depth = 4)
train.data <- random.hRF$train.data
# run HRF and predict
hie.RF.random <- RunHRF(train.data = train.data,
                        case.ID    = "case.ID",
                        hie.levels = c(2:(random.hRF$call$tree.depth + 1)),
                        mtry="tuneRF2")
prop.votes.lRF.train <- predict(hie.RF.random)$prop.vote.train

# multiply path only until the deepest level of the class hierarchy
multi.prop.votes.full <- GetMultPropVotes(
  prop.vote    = prop.votes.lRF.train,
  unique.path  = hie.RF.random$hier.struc$unique.path,
  all.levels   = FALSE)
multi.prop.votes.L4 <- multi.prop.votes.full[[1]]

#####
data(OliveOilHie)

hie.RF.00 <- RunHRF(train.data = OliveOilHie,
                    case.ID    = "case.ID",
                    hie.levels = c(2:4),
                    internal.end.path = TRUE,
                    mtry= "tuneRF2")

prop.votes.00 <- predict(hie.RF.00)$prop.vote.train
mult.prop.00 <- GetMultPropVotes(prop.vote    = prop.votes.00,
                                unique.path  = hie.RF.00$hier.struc$unique.path,
                                all.levels   = TRUE)

plot(hie.RF.00)
multi.prop.votes.L1 <- mult.prop.00[["prop.multiplicative.votes.L1"]]
names(multi.prop.votes.L1)[3:ncol(multi.prop.votes.L1)]

multi.prop.votes.L2 <- mult.prop.00[["prop.multiplicative.votes.L2"]]
names(multi.prop.votes.L2)[3:ncol(multi.prop.votes.L2)]

# note that terminal nodes from level 2 appears in L3 as well.
multi.prop.votes.L3 <- mult.prop.00[["prop.multiplicative.votes.L3"]]
names(multi.prop.votes.L3)[3:ncol(multi.prop.votes.L3)]

```

---

HieFMeasure

*Hierarchical precision, hierarchical recall and hierarchical F measure*


---

## Description

This function estimate 3 measures of accuracy that accounts for the hierarchical structure of the class hierarchy.

## Usage

```
HieFMeasure(conf.matr, unique.path, beta.h.F = 1, by.node = FALSE, ...)
```

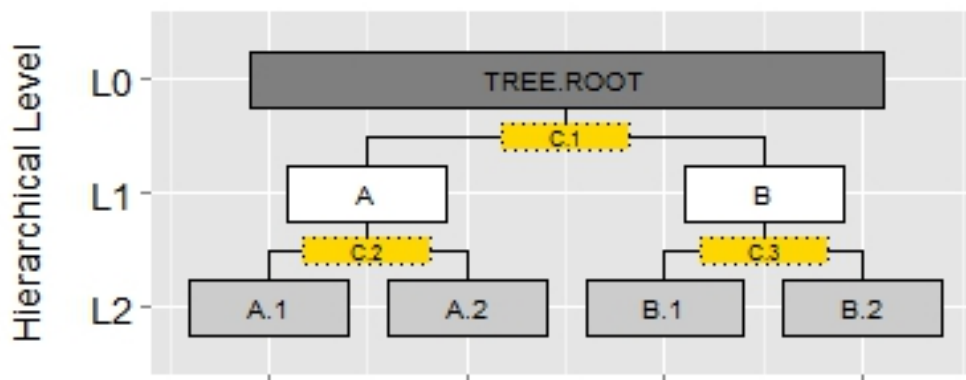
## Arguments

<code>conf.matr</code>	Object of class "confusionMatrix", as generated by <code>confusionMatrix</code> function of the 'caret' package.
<code>unique.path</code>	Data frame, the unique.path data frame from RunHRF. Can be found in an object of class "HRF" at <code>\$hier.struc\$unique.path</code> .
<code>beta.h.F</code>	Numeric in the range $\text{beta.h.F} \geq 0$ . Controls the relative weight of Hierarchical precision and hierarchical recall in the estimation of hierarchical F measure. Values close to 0 will give higher weight to Hierarchical precision. Equal weights are for $\text{beta.h.F} = 1$ , while values above 1 will give higher weight to hierarchical recall. See details in <i>Kiritchenko et al. (2005)</i> .
<code>by.node</code>	Logical, if TRUE, the hierarchical precision, recall and F measure are estimated for each class as well.
<code>...</code>	Optional parameters to be passed to low level functions.

## Details

To create a confusion matrix in with the caret package, the levels of the observed and predicted vectors should be identical. Therefore, we advise using `JoinLevels`. *Kiritchenko et al. (2005)* estimated the hierarchical precision, recall and F measure by summing over all cases. Here, we estimate the same values directly from the confusion matrix. In addition, *Kiritchenko et al. (2005)* do not provide a class specific measure of hierarchical accuracy. Here we developed one such measure.

In the hierarchical measures of accuracy, not all misclassifications carry the same weight. For example, in the simple hierarchy below, correct classification of a case from class *A.1* will contribute most to the overall accuracy (2 in this case). However, misclassification of the *A.1* case to class *A.2* will still contribute to the overall accuracy since *A.1* and *A.2* share one node in their path (*A*). On the other hand, misclassification of the *A.1* case to class *B.2* will not contribute anything to the overall accuracy, since the two terminal nodes only meet at the tree root of the class hierarchy.



## Value

a data frame with the name of the measure in the first column and the value in the second.

### hP, hR and hF

The hierarchical precision ( $hP$ ), recall ( $hR$ ) and F measure ( $hF$ ) are estimated from a confusion matrix (with  $j$  being the observed class and  $k$  being the predicted class) using the formulas given below. Similarly, the hierarchical precision for predicted class  $k$  ( $hP[k]$ ), the hierarchical recall for observed class  $j$  ( $hR[j]$ ), and the hierarchical F measure for class  $j=k$  ( $hF[j=k]$ ) can also be estimated from the confusion matrix. The by .node specific measures converge to the regular flat precision and recall (respectively) by setting  $S[j,k] = 1$  if  $j = k$  and  $S[j,k] = 0$  if  $j$  differs from  $k$ .

$hP = \sum_j \sum_k (S_{j,k} \cdot F_{j,k}) / \sum_k (S_{k,k} \cdot F_{+,k})$ $hR = \sum_j \sum_k (S_{j,k} \cdot F_{j,k}) / \sum_j (S_{j,j} \cdot F_{j,+})$ $hF = [(\beta^2 + 1) \cdot hP \cdot hR] / (\beta^2 \cdot hP + hR)$ $hP_k = \sum_j (S_{j,k} \cdot F_{j,k}) / (S_{k,k} \cdot F_{+,k})$ $hR_j = \sum_k (S_{j,k} \cdot F_{j,k}) / (S_{j,j} \cdot F_{j,+})$ $hF_{j=k} = [(\beta^2 + 1) \cdot hP_k \cdot hR_j] / (\beta^2 \cdot hP_k + hR_j)$	<p><b>j</b> An observed class</p> <p><b>k</b> A predicted class</p> <p><math>S_{j,k}</math> The length of the path from the first common ancestor of classes <math>j</math> and <math>k</math> to the root. For <math>S_{j,j}</math> and <math>S_{k,k}</math> - the depth of class <math>j</math> or <math>k</math>, respectively</p> <p><math>F_{j,k}</math> The number of cases observed in class <math>j</math> that were classified as class <math>k</math></p> <p><math>F_{+,k}</math> The total number of cases classified as class <math>k</math></p> <p><math>F_{j,+}</math> The total number of cases observed as class <math>j</math></p> <p><math>\beta</math> The relative weight of precision and recall (beta.h.F as defined above)</p>
---	---

### Author(s)

Yoni Gavish <gavishyoni@gmail.com>

### References

Kiritchenko, S., S. Matwin, and F. Famili. 2005. Functional annotation of genes using hierarchical text categorization. In: *Proc. of the ACL Workshop on Linking Biological Literature, Ontologies and Databases: Mining Biological Semantics*.

### Examples

```
data(OliveOilHie)
hie.RF.00 <- RunHRF(train.data      = OliveOilHie,
                    case.ID         = "case.ID",
                    hie.levels      = c(2:4),
                    mtry            = "tuneRF2",
                    internal.end.path = TRUE)

# extract the crisp class using PerformanceHRF
perf.hRF.00 <- PerformanceHRF(hie.RF      = hie.RF.00,
                              per.index   = c("hie.F.measure"),
                              crisp.rule  = c("stepwise.majority"))
crisp.00 <- perf.hRF.00$crisp.case.class

# Join factor levels of the observed and predicted
joined.levels <- JoinLevels(vector.1 = crisp.00$stepwise.majority.rule,
                           vector.2 = crisp.00$obs.term.node)

# create the confusionMatrix object
library(caret)
conf.matr <- confusionMatrix(data      = joined.levels$vector.1,
                             reference = joined.levels$vector.2,
                             dnn       = c("Prediction", "Observed"))
```



```
# the HieFMeasure
hie.F.value <- HieFMeasure(conf.matr = conf.matr,
                           unique.path = hie.RF.00$hier.struc$unique.path,
                           beta.h.F = 1,
                           by.node = TRUE)

# compare to the output of PerformanceHRF
hie.performance <- perf.hRF.00$hie.performance
```

---

ImportanceHie	<i>The importance value of each explanatory variable in each local classifier</i>
---------------	---

---

## Description

This function extracts the importance value of each variable for each local classifier. Requires setting `importance = TRUE` when running [RunHRF](#). Relies on the function [importance](#) in the `randomForest` package.

## Usage

```
ImportanceHie(hie.RF, format.out = c("col.4.out"), scale.imp = FALSE, ...)
```

## Arguments

<code>hie.RF</code>	Object of class "HRF" - the output of <code>RunHRF</code> .
<code>format.out</code>	The format of the output. Can be either "col.4.out" (default), "table.out" or both. See Value for details.
<code>scale.imp</code>	Logical, if TRUE, variable importance is scaled by the standard error. Default is FALSE. See <a href="#">importance</a> in the <code>randomForest</code> package for details.
<code>...</code>	Optional parameters to be passed to low level functions.

## Details

Returns the mean decrease in accuracy (type=1 in `randomForest::importance`).

## Value

A data frame or list containing the importance of each explanatory variable in each local classifier. Output structure depends on the settings of `format.out`:

1. `format.out = "col.4.out"` -  
Data frame with four columns:
  - 'classifier.ID' -The name of the local classifier.
  - 'par.level' -The name of the parent node in the local classifier.
  - 'expl.var' -The name of the explanatory variable.
  - 'mean.dec.accu' -the mean decrease in accuracy.

2. `format.out = "table.out"` -  
Data frame, with the first column containing the name of the local classifier and the rest of the columns the names and variable importance of each explanatory variable.
3. `format.out = c("col.4.out", "table.out")` -  
A list containing the two data frames, under 'var.imp.4.col' and 'var.imp.table', respectively.

**Author(s)**

Yoni Gavish <gavishyoni@gmail.com>

**See Also**

[PlotImportanceHie](#) for a ggplot2 based plotting of hierarchical importance values.

**Examples**

```
data(OliveOilHie)
hie.RF.00 <- RunHRRF(train.data      = OliveOilHie,
                    case.ID         = "case.ID",
                    hie.levels      = c(2:4),
                    mtry             = "tuneRF2",
                    internal.end.path = TRUE)

impor.hie.RF.00 <- ImportanceHie(hie.RF      = hie.RF.00,
                                format.out = c("col.4.out"))

plot(hie.RF.00)
PlotImportanceHie(input.data = impor.hie.RF.00,
                  X.data      = 2,
                  Y.data      = 3,
                  imp.data     = 4,
                  plot.type   = "Tile",
                  X.Title     = c("Parent node Name"),
                  Y.Title     = c("Explanatory variable"),
                  imp.title    = c("Mean \n Decrease \n in \n Accuracy"))

# table format
impor.hie.RF.00 <- ImportanceHie(hie.RF      = hie.RF.00,
                                format.out = c("table.out"))

# both output formats
impor.hie.RF.00 <- ImportanceHie(hie.RF      = hie.RF.00,
                                format.out = c("col.4.out", "table.out"))

import.col.4.for <- impor.hie.RF.00$imp.var.4.col
import.table.for <- impor.hie.RF.00$imp.val.table
```

## Description

This function takes two vectors, combines their factor levels, and return a list with the same vectors, only with identical and full factor levels. The order of the factors is first all the factors of vector.1 then the added factor levels of vector.2. As `randomForest` cannot predict if any input variables of a new data contains factor levels that were not included in the training data, we recommend running this function on each categorical input variable of the training and new data before running the hierarchical randomForest with [RunHRF](#).

## Usage

```
JoinLevels(vector.1, vector.2, ...)
```

## Arguments

<code>vector.1</code>	The first vector.
<code>vector.2</code>	The second vector.
<code>...</code>	Optional parameters to be passed to low level functions.

## Value

A list with two vectors, arranged according to the input order

## Author(s)

Yoni Gavish <[gavishyoni@gmail.com](mailto:gavishyoni@gmail.com)>

## Examples

```
# create two vectors
vec.1 <- as.factor(rep(c("a", "b", "c"), 10))
vec.2 <- as.factor(rep(c("g", "a", "f", "b"), 20))

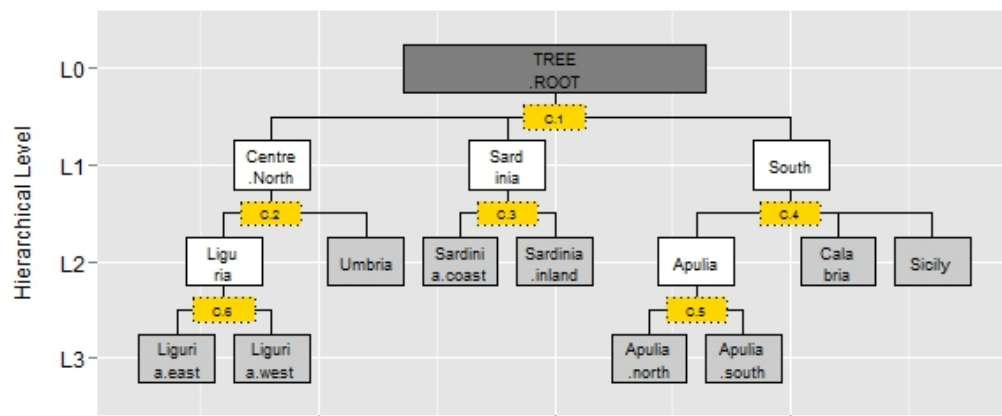
# run the function
new.Vec <- JoinLevels(vec.1, vec.2)

# Re-assign to the original vectors
vec.1 <- new.Vec[[1]]
vec.2 <- new.Vec[[2]]

# note the levels
levels(vec.1)
levels(vec.2)
```

## Description

The original data-set ([oliveoil](#)) contains information on eight chemical measurements on different specimen of olive oil produced in various regions in Italy (northern Apulia, southern Apulia, Calabria, Sicily, inland Sardinia and coast Sardinia, eastern and western Liguria, Umbria) and further classifiable into three macro-areas: Centre-North, South, Sardinia". Here the hierarchy is slightly modified to include three hierarchical levels by adding a level 2 class for the Liguria and Apulia regions. The level 3 classes for these two regions are further separated to east/west and north/south, for Liguria and Apulia, respectively. Other terminal nodes are in level 2, resulting with a class hierarchy with terminal nodes at different levels. The default of `END.PATH` is used for `end.path.name`. See the figure below for the class hierarchy.



## Usage

```
data(OliveOilHie)
```

## Format

A data frame with 572 rows and 12 variables:

- `case.ID` -  
A unique `case.ID` value for each case
- `L1`, `L2`, `L3` -  
the three levels of the class hierarchy
- Other columns -  
The eight chemical measurements

## Source

see details in [oliveoil](#) in the package [pdfCluster](#)

---

PerformanceFlatRF	<i>Runs a flat classification on the same data as hie.RF and assess performance.</i>
-------------------	--

---

## Description

This function takes as input an object of class HRF, identifies the observed terminal node for each case of the original training data, and runs a regular flat classification on all terminal nodes. The function then applies two different methods for selecting a single terminal node (given by `crisp.rule` and described in [PerformanceHRF](#)). Finally, the performance is explored in relation to the observed class using various performance measures (given by `per.index`)

## Usage

```
PerformanceFlatRF(hie.RF, mtry = hie.RF$call$mtry,
  ntree = hie.RF$call$ntree, importance = hie.RF$call$importance,
  proximity = hie.RF$call$proximity, keep.forest = hie.RF$call$keep.forest,
  keep.inbag = hie.RF$call$keep.inbag, per.index = c("flat.measures",
    "hie.F.measure"), crisp.rule = c("multiplicative.majority",
    "multiplicative.permutation"), perm.num = 500, by.node = TRUE,
  div.logical = TRUE, div.print = 25, beta.h.F = 1, ...)
```

## Arguments

<code>hie.RF</code>	Object of class "HRF" - the output of <code>RunHRF</code> .
<code>mtry</code>	Number of variables randomly sampled as candidates at each split. Default is to use the same method used in <code>hie.RF</code> .
<code>ntree</code>	Number of trees to grow in the flat classifier. See <a href="#">randomForest</a> for additional details. Default is the same <code>ntree</code> used in each local classifier of <code>hie.RF</code> .
<code>importance</code>	Logical, if TRUE, importance of variables will be assessed. See <a href="#">randomForest</a> for additional details. Default is the same as <code>hie.RF</code> .
<code>proximity</code>	Logical, If TRUE, proximity will be calculated. See <a href="#">randomForest</a> for additional details. Default is the same as <code>hie.RF</code> .
<code>keep.forest</code>	Logical, if TRUE (recommended) the forest will be retained. Default is the same as <code>hie.RF</code> .
<code>keep.inbag</code>	Logical, if TRUE an $n$ by <code>ntree</code> matrix be returned that keeps track of which cases are out-of-bag in which trees. $n$ being the number of cases in the training set. Required for votes extraction. Default is the same as <code>hie.RF</code> .
<code>per.index</code>	The performance and accuracy indices to compute. See details in <a href="#">PerformanceHRF</a> .
<code>crisp.rule</code>	The method of selecting a single crisp class from the proportion of votes. See details in <a href="#">PerformanceHRF</a> .
<code>perm.num</code>	Integer, number of random permutations if 'multiplicative.permutation' is applied. See details in <a href="#">PerformanceHRF</a> .
<code>by.node</code>	Logical, if TRUE, performance measures are estimated for each terminal node as well.
<code>div.logical</code>	Logical, if TRUE progress when 'multiplicative.permutation' is applied will be printed every <code>div.print</code> permutations.

<code>div.print</code>	See above.
<code>beta.h.F</code>	Numeric in the range <code>beta.h.F ==&gt; 0</code> . Controls weights in the hierarchical F measure index. See <a href="#">HieFMeasure</a> for details.
<code>...</code>	Optional parameters to be passed to low level functions.

### Details

When running a flat classification, a single randomForest algorithm is used to distinguish between all terminal nodes. As no information on the proportion of votes along the class hierarchy is produced, the "stepwise.majority" option for `crisp.rule` cannot be calculated.

This function allows comparison of the performance of the hierarchical and flat randomForest. In addition, the proportion of votes produced by this function (can be extracted from the randomForest object) is comparable to the proportion of votes returned by [PerformanceHRF](#) in the 'multiplicative.prop' data frame.

### Value

List with the following components:

- 'flat.RF' -  
An object of class randomForest for the flat classification.
- 'crisp.case.class' -  
Data frame containing the crisp class for each case based on all options defined by `crisp.rule`. The observed class (terminal node) is given at the last column under `obs.term.node`.
- 'hie.performance' -  
Data frame summarizing all the performance measures, starting with the overall performance indices and followed by all the `by.node` measures. See details in [PerformanceHRF](#).
- 'nodes.measures.columns' -  
Optional, if `by.node=TRUE`, data frame with three columns including the name of the terminal node, the performance index and the name of the column in `hie.performance` that holds the output.
- 'call' -  
The call to function `PerformanceFlatHRF`.

### Author(s)

Yoni Gavish <[gavishyoni@gmail.com](mailto:gavishyoni@gmail.com)>

### See Also

[RunHRF](#) for running a hierarchical randomForest analysis, [PerformanceHRF](#) for performance analysis, [HieFMeasure](#) for additional information on the hierarchical performance measures.

### Examples

```
# analyse the OliveOilHie dataset
data(OliveOilHie)
hie.RF.00 <- RunHRF(train.data      = OliveOilHie,
                    case.ID        = "case.ID",
                    hie.levels     = c(2:4),
                    mtry           = "tuneRF2",
                    internal.end.path = TRUE)
```

```

# run and assess performance as flat classification
flat.RF.00 <- PerformanceFlatRF(hie.RF      = hie.RF.00,
                               per.index   = c("flat.measures",
                                                "hie.F.measure"),
                               crisp.rule  = c("multiplicative.majority",
                                                "multiplicative.permutation"),
                               perm.num    = 10,
                               div.print   = 2)

# extract values
names(flat.RF.00)
flat.RF.00.RF      <- flat.RF.00$flat.RF # object of class randomForest
votes.flat        <- flat.RF.00$flat.RF$votes
flat.RF.00.crisp   <- flat.RF.00$crisp.case.class
hie.perf.flat     <- flat.RF.00$hie.performance
flat.RF.00.nodes.meas <- flat.RF.00$nodes.measures.columns
flat.RF.00.call    <- flat.RF.00$call

# compare the hie.perf.flat to the HRF analysis
# Performance of the hierarchical randomForest
perf.hRF.00 <- PerformanceHRF(hie.RF      = hie.RF.00,
                              crisp.rule  = c("multiplicative.majority",
                                                "multiplicative.permutation"),
                              perm.num    = 10,
                              div.print   = 2)

hie.perf.HRF     <- perf.hRF.00$hie.performance

# Despite the overall high performance, HRF is slightly better...
comp.perf <- data.frame(model = c("Flat", "HRF"))
join.perf <- rbind(hie.perf.flat[1, c("Accuracy", "Kappa", "h.F.measure")],
                  hie.perf.HRF[1, c("Accuracy", "Kappa", "h.F.measure")])
comp.perf <- cbind(data.frame(model = c("Flat", "HRF")),
                  join.perf)
comp.perf

```

---

PerformanceHRF

*Flat and hierarchical performance measures.*


---

## Description

This function predicts the proportion of votes for each case in each local classifier and then estimates various flat and hierarchical performance measures. The performance measures are based on translating the proportion of votes to a crisp class (selecting a single class). The function offers three different method for translating the soft proportion of votes to a crisp class.

## Usage

```

PerformanceHRF(hie.RF, per.index = c("flat.measures", "hie.F.measure"),
               crisp.rule = c("stepwise.majority", "multiplicative.majority",
                              "multiplicative.permutation"), perm.num = 500, by.node = TRUE,
               div.logical = TRUE, div.print = 25, beta.h.F = 1, ...)

```

## Arguments

<code>hie.RF</code>	Object of class "HRF" - the output of RunHRF.
<code>per.index</code>	The performance and accuracy indices to compute. See details below.
<code>crisp.rule</code>	The method of selecting a single crisp class from the proportion of votes. See details below.
<code>perm.num</code>	Integer, number of random permutations if 'multiplicative.permutation' is applied.
<code>by.node</code>	Logical, if TRUE, performances indices will be estimated for each terminal node as well as for the overall confusion matrix.
<code>div.logical</code>	Logical, if TRUE progress when 'multiplicative.permutation' is applied will be printed every <code>div.print</code> permutations.
<code>div.print</code>	See above.
<code>beta.h.F</code>	Numeric in the range $\beta.h.F \geq 0$ . Controls weights in the hierarchical F measure index. See <a href="#">HieFMeasure</a> for details.
<code>...</code>	Optional parameters to be passed to low level functions.

## Details

For a given flat classification, the output of the randomForest algorithm for each case is the proportion of out-of-bag (OOB) votes that each class (node) received. In the hierarchical version of randomForest, the proportion of OOB votes are returned separately for each local classifier (see [predict.HRF](#)). In flat classification, the proportions of votes are translated into a crisp classification by applying the majority rule, i.e., by selecting the class with the highest proportion of votes. In the hierarchical version of randomForest, there are 3 different methods to select one crisp class for a given case.

### The "stepwise.majority" method:

In each local classifier, the flat majority rule is applied. Then starting with the `tree.root`, a case is classified down the tree until a terminal node is reached. This method gives high emphasis to local classifiers close to the `tree.root`, since a case can only be classified to sibling classes of those selected using the majority rule in classes closer to the `tree.root`.

If the blue values in the figure below are the proportion of votes for a given case in a given local classifier, then the case would be classified to class A in local classifier C.1, and then to class A.2 in local classifier C.2.

### The "multiplicative.majority" method:

First multiply the proportion of votes along each path from the `tree.root` to any of the terminal nodes (see: [GetMultPropVotes](#)). The result is a vector of probabilities for each terminal node, similar to that returned from flat classification. Next, apply the majority rule and select the terminal node that received the highest multiplicative proportion of votes. This method give less emphasis to local classifiers near the root of the class hierarchy, but may depend on the number of siblings nodes.

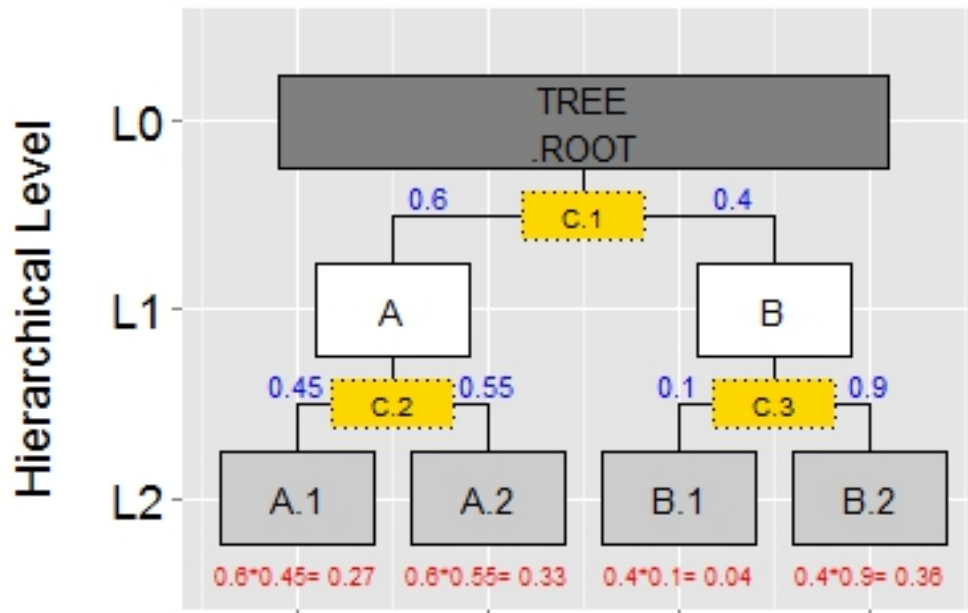
If the red values in the figure below are the multiplication of votes along each path to all 4 terminal nodes for a given case, then the case would be classified to class B.2

### The "multiplicative.permutation" method:

Similar to `multiplicative.majority`, but instead of applying a majority rule, the method randomly select a terminal node based on the multiplicative probabilities. The user defines the number of permutations and accuracy is assessed separately for each permutation. When most cases are classified to the same class in various classification trees (i.e., when the proportion of votes for one category is close to 1), the mean accuracy over all permutation should be similar to that achieved under the "multiplicative.majority" method.



In the example in the figure below, each permutation will choose a class as random draw with probabilities equal to the values in red.



The two options of `per.index`:

- 'flat.measures' - Accuracy measures that do not consider the hierarchical structure of the classes. In this case, the accuracy measures returned are those returned by `confusionMatrix` of the caret package.
- 'hie.F.measure' - Accuracy measures that account for the hierarchical structure of the classes. See details in `HieFMeasure` and reference within.

## Value

A list with the following components:

- 'raw.vote' - Data frame containing for each case, the proportion of votes for each node in each local classifier (the output of `predict.HRF`).
- 'crisp.case.class' - Data frame containing the crisp class for each case based on all options defined by `crisp.rule`. The observed class (terminal node) is given at the last column under `obs.term.node`.
- 'hie.performance' - Data frame summarizing all the performance measures, starting with the overall performance indices and followed by all the `by.node` measures. See details below.
- 'multiplicative.prop' - Optional data frame with the multiplicative proportion of votes (the output of `GetMultPropVotes`).

Will be returned if "multiplicative.majority" or "multiplicative.permutation" were selected.

- 'nodes.measures.columns' -  
Optional, if by.node=TRUE, data frame with three columns including the name of the terminal node, the performance index and the name of the column in hie.performance that holds the output.
- 'call' -  
The call to function PerformanceHRF.

### The hie.performance data frame

The hie.performance data frame contains the following overall performance measures:

- 'crisp.rule' -  
The crisp rule used to create the crisp classification.
- 'Accuracy' -  
The overall accuracy.
- 'Kappa' -  
The unweighted Kappa statistic.
- 'AccuracyLower' -  
The lower bound of the 95 confidence interval around the overall error based on [binom.test](#).
- 'AccuracyUpper' -  
The upper bound of the 95 confidence interval.
- 'AccuracyNull' -  
The expected null accuracy.
- 'AccuracyPValue' -  
One side test to see if accuracy is better than "no information rate".
- 'AccuracyNull' -  
A p-value from McNemar's test using [mcnemar.test](#) (may return NA).
- 'h.precision' -  
The hierarchical precision from [HieFMeasure](#).
- 'h.recall' -  
The hierarchical recall from [HieFMeasure](#).
- 'h.F.measure' -  
The hierarchical F measure from [HieFMeasure](#).

These columns are followed by all the by.node results, with the name of each column follows the structure: 'nodename;measure' (see nodes.measures.columns above).

### Author(s)

Yoni Gavish <[gavishyoni@gmail.com](mailto:gavishyoni@gmail.com)>

### See Also

[RunHRF](#) for running a hierarchical randomForest analysis, [PerformanceNewHRF](#) for performance analysis on new.data with observed terminal node, [PerformanceFlatRF](#) for hierarchical and flat performance analyses for a flat randomForest, [HieFMeasure](#) for additional information on the hierarchical performance measures.

**Examples**

```

# create a random HRF dataset and run HRF analysis
set.seed(354)
random.hRF <- RandomHRF(num.term.nodes = 20, tree.depth = 4)
train.data <- random.hRF$train.data
hie.RF.random <- RunHRF(train.data = train.data,
                        case.ID = "case.ID",
                        hie.levels = c(2:(random.hRF$call$tree.depth + 1)))

# assess performance
perf.hRF.random <- PerformanceHRF(hie.RF = hie.RF.random,
                                perm.num = 20,
                                div.print = 5)

### Extract values ###
names(perf.hRF.random)
raw.vote.random <- perf.hRF.random$raw.vote
crisp.case.class.random <- perf.hRF.random$crisp.case.class
hie.performance.random <- perf.hRF.random$hie.performance
multiplicative.prop.random <- perf.hRF.random$multiplicative.prop
nodes.measures.columns.random <- perf.hRF.random$nodes.measures.columns
hie.perf.call.random <- perf.hRF.random$call

#### example with the OliveOilHie dataset
data(OliveOilHie)
hie.RF.00 <- RunHRF(train.data = OliveOilHie,
                    case.ID = "case.ID",
                    hie.levels = c(2:4),
                    mtry = "tuneRF2",
                    internal.end.path = TRUE)

perf.hRF.olive <- PerformanceHRF(hie.RF = hie.RF.00,
                                crisp.rule = c("multiplicative.majority"))

### Extract values ###
crisp.case.class.olive <- perf.hRF.olive$crisp.case.class
hie.performance.olive <- perf.hRF.olive$hie.performance

### plotting option ###
# create a confusion matrix
conf.matr.olive <- as.data.frame(table(crisp.case.class.olive$obs.term.node,
                                     crisp.case.class.olive$multiplicative.majority.rule))
# use the PlotImportanceHie to plot the confusion matrix
PlotImportanceHie(input.data = conf.matr.olive,
                  X.data = 1,
                  Y.data = 2,
                  imp.data = 3,
                  plot.type = "Tile",
                  X.Title = c("Observed"),
                  Y.Title = c("mMultiplicative majority rule"),
                  imp.title = c("frequency"),
                  low.col = "darkslategray4",
                  high.col = "red",
                  geom.tile.bor.col = "gray20")

```

---

PerformanceNewHRF	<i>Predict and assess performance of new.data, for which the 'true' class is known.</i>
-------------------	---

---

## Description

This function takes as input a new.data data frame with the same explanatory variables as those used in hie.RF. Next, the [predict.HRF](#) function is applied to extract the proportion of votes that each case in new.data received for each node in each local classifier of hie.RF. The function then applies up to three different methods for selecting a single terminal node (given by crisp.rule and described in [PerformanceHRF](#)). Finally, the performance is explored in relation to the observed class using various performance measures (given by per.index and described in [PerformanceHRF](#)).

## Usage

```
PerformanceNewHRF(hie.RF, new.data, new.data.case.id = 1,
  new.data.exp.var = NA, new.data.hie, crisp.rule = c("stepwise.majority",
    "multiplicative.majority", "multiplicative.permutation"), perm.num = 500,
  div.logical = TRUE, div.print = 25, per.index = c("flat.measures",
    "hie.F.measure"), by.node = TRUE, beta.h.F = 1, ...)
```

## Arguments

hie.RF	Object of class HRF - the output of RunHRF.
new.data	Data frame containing additional cases that were not a part of the original training set.
new.data.case.id	Integer, specifying the column number with the case.id in the new.data data frame. The case.id values should be unique and different from those in the training data.
new.data.exp.var	Vector of integers, specifying the columns of new.data that contains the same set of explanatory variables as used in the training of hie.RF. Default is all columns except new.data.case.id and new.data.hie.
new.data.hie	Vector of character or integers, containing the names or column numbers of the hierarchical levels in new.data. Order of columns should be from the tree.root to the terminal nodes. If a single column is provided, it should contain only terminal nodes.
crisp.rule	The method of selecting a single crisp class from the proportion of votes. See details in <a href="#">PerformanceHRF</a> .
perm.num	Integer, number of random permutations for each case if 'multiplicative.permutation' is applied. See details in <a href="#">PerformanceHRF</a> .
div.logical	Logical, if TRUE progress when 'multiplicative.permutation' is applied will be printed every div.print permutations
div.print	See above.
per.index	The performance and accuracy indices to compute. See details in <a href="#">PerformanceHRF</a> .
by.node	Logical, if TRUE performances indices will be estimated for each terminal node as well as for the overall confusion matrix.

`beta.h.F` Numeric in the range `beta.h.F >= 0`. Controls weights in the hierarchical F measure index. See [HieFMeasure](#) for details.

`...` Optional parameters to be passed to low level functions.

### Details

See details on the various `crisp.rule`, `per.index` and on the structure of the output data frames in [PerformanceHRF](#).

### Value

A list with the following components:

- `'raw.vote'` - Data frame containing for each case, the proportion of votes for each node in each local classifier (the output of [predict.HRF](#)).
- `'crisp.case.class'` - Data frame containing the crisp class for each case based on all options defined by `crisp.rule`. The observed class (terminal node) is given at the last column under `obs.term.node`.
- `'hie.performance'` - Data frame summarizing all the performance measures, starting with the overall performance indices and followed by all the `by.node` measures. See details in [PerformanceHRF](#).
- `'multiplicative.prop'` - Optional data frame with the multiplicative proportion of votes (the output of [GetMultPropVotes](#)). Will be returned if `"multiplicative.majority"` or `"multiplicative.permutation"` were selected.
- `'nodes.measures.columns'` - Optional, if `by.node=TRUE`, data frame with three columns including the name of the terminal node, the performance index and the name of the column in `hie.performance` that holds the output.
- `'call'` - The call to function `PerformanceNewHRF`.

### Author(s)

Yoni Gavish <[gavishyoni@gmail.com](mailto:gavishyoni@gmail.com)>

### See Also

[RunHRF](#) for running a hierarchical randomForest analysis, [PredictNewHRF](#) for predicting crisp class for each case of `new.data`, [PerformanceHRF](#) for performance analysis, [HieFMeasure](#) for additional information on the hierarchical performance measures.

### Examples

```
# create a random HRF dataset and RunHRF
set.seed(354)
random.hrf <- RandomHRF(num.term.nodes = 20,
                        tree.depth = 4,
                        new.data.observed = TRUE)
train.data <- random.hrf$train.data
new.data <- random.hrf$new.data
hie.RF.random <- RunHRF(train.data = train.data,
```

```

case.ID      = "case.ID",
hie.levels   = c(2:(random.hrf$call$tree.depth + 1)))

# assess performance for the new.data
perf.new.data <- PerformanceNewHRF(hie.RF          = hie.RF.random,
new.data      = new.data ,
new.data.case.id = 1,
new.data.hie   = c(2:(random.hrf$call$tree.depth + 1)),
crisp.rule     = c("stepwise.majority",
                  "multiplicative.majority",
                  "multiplicative.permutation"),
perm.num       = 10,
div.print      = 2,
per.index      = c("flat.measures", "hie.F.measure"),
by.node        = TRUE)

# extract the data
names(perf.new.data)
perf.new.votes <- perf.new.data$raw.votes
perf.new.crisp <- perf.new.data$crisp.case.class
perf.new.hie.perf <- perf.new.data$hie.performance
perf.new.mult.prop <- perf.new.data$multiplicative.prop
perf.new.nodes.meas <- perf.new.data$nodes.measures.columns
perf.new.call <- perf.new.data$call

```

---

plot.HRF

---

*Plot the Hierarchical class structure*


---

## Description

This function takes as input an object of class "HRF", and plots the structure of the class hierarchy.

## Usage

```

## S3 method for class 'HRF'
plot(x, rect.width = 0.8, rect.hieght = 0.5,
text.size = 7.5, text.angle = 0, split.text = 6, int.col = "white",
term.col = "gray80", root.col = "gray50",
Y.title = "Hierarchical Level\n", label.nodes = TRUE,
label.classifiers = TRUE, class.ID.rect.col = "black",
class.ID.rect.fill = "gold", class.ID.rect.linetype = "dotted",
class.ID.rect.hieght.scale = 0.25, class.ID.text.size.scale = 0.8,
class.ID.text.color = "black", ...)

```

## Arguments

x	Object of class "HRF" - the output of RunHRF.
rect.width	Numeric, the width of boxes for each node, at width=1, the boxes overlap.
rect.hieght	Numeric, the height of the boxes for each node, center to center distance between levels is 1.
text.size	Numeric, a scaling number for the size of text.
text.angle	Numeric, the angle of the text of the nodes.

<code>split.text</code>	Numeric, the number of characters in nodes names above which the text is broken to two lines.
<code>int.col</code>	Character, specifying the color of the fill of the internal nodes.
<code>term.col</code>	Character, specifying the color of the fill of the terminal nodes.
<code>root.col</code>	Character, specifying the color of the fill of the <code>tree.root</code> node.
<code>Y.title</code>	Character, title of the y axis
<code>label.nodes</code>	Logical, if TRUE (default), the nodes are labelled.
<code>label.classifiers</code>	Logical, if TRUE (default), the local classifiers are labelled.
<code>class.ID.rect.col</code>	Character, specifying the border color of the classifier.ID boxes.
<code>class.ID.rect.fill</code>	Character, specifying the fill color of the classifier.ID boxes.
<code>class.ID.rect.linetype</code>	Character, specifying the linetype of the classifier.ID boxes.
<code>class.ID.rect.hieght.scale</code>	Numeric, the scaling of the classifier.ID box relative to the nodes boxes.
<code>class.ID.text.size.scale</code>	Numeric, the scaling of the classifier.ID text relative to the nodes text
<code>class.ID.text.color</code>	Character, specifying the color of the text for the classifier.ID.
<code>...</code>	Optional parameters to be passed to the low level functions.

### Details

Based on [ggplot2](#), the function plots the class hierarchy along with the local classifiers.

### Value

The function returns a plot. If the function is saved to a new object, that the plot can be accessed and further edited using `$plot` on the new object (see examples for details).

### Author(s)

Yoni Gavish <[gavishyoni@gmail.com](mailto:gavishyoni@gmail.com)>

### See Also

[RunHRF](#) for running a hierarchical randomForest, [PerformanceHRF](#) for assessing the performance and accuracy of the HRF.

### Examples

```
# create random HRF data
set.seed(354)
random.hrf <- RandomHRF(num.term.nodes = 20, tree.depth = 4)
train.data <- random.hrf$train.data
# run HRF
hie.RF.random <- RunHRF(train.data = train.data,
                        case.ID = "case.ID",
                        hie.levels = c(2:(random.hrf$call$tree.depth + 1)))
```

```

# S3 method for plot
plot.hie.RF.random <- plot(hie.RF.random)

# further editing of the plot with ggplot2
plot.hRF.tree.ran <- plot.hie.RF.random$plot
class(plot.hRF.tree.ran)
plot.hRF.tree.ran <- plot.hRF.tree.ran +
  ggtitle("The Class Hierarchy, internal and
          terminal nodes and all local classifiers")
plot.hRF.tree.ran <- plot.hRF.tree.ran +
  theme(plot.title = element_text(lineheight=.8,
                                   face="bold",
                                   color="red"))

plot.hRF.tree.ran <- plot.hRF.tree.ran +
  theme(axis.title.y = element_text(size = 20,
                                     colour="blue"))

plot.hRF.tree.ran
# the plot uses the following coordinates:
# x runs from 1 to the number of terminal nodes + 1
# y runs from 1 to the number of levels in the class hierarchy + 1
plot.hRF.tree.ran <- plot.hRF.tree.ran +
  geom_text(data=NULL,
            aes(x = 1,
                y = random.hRF$call$tree.depth + 1,
                label = "Top-\nLeft"),
            size = 5,
            color = "magenta")

plot.hRF.tree.ran <- plot.hRF.tree.ran +
  geom_text(data=NULL,
            aes(x = 1,
                y = 1,
                label = "Bottom-\nLeft"),
            size = 5,
            color = "magenta")

plot.hRF.tree.ran <- plot.hRF.tree.ran +
  geom_text(data=NULL,
            aes(x = random.hRF$call$num.term.nodes + 1,
                y = random.hRF$call$tree.depth + 1,
                label = "Top-\nRight"),
            size = 5,
            color = "magenta")

plot.hRF.tree.ran <- plot.hRF.tree.ran +
  geom_text(data=NULL,
            aes(x = random.hRF$call$num.term.nodes + 1,
                y = 1,
                label = "Bottom-\nRight"),
            size = 5,
            color = "magenta")

plot.hRF.tree.ran

#####
#example with a the OliveOilHie data-set
data(OliveOilHie)
hie.RF.00 <- RunHRF(train.data = OliveOilHie,
                    case.ID = "case.ID",
                    hie.levels = c(2:4),

```



```

                                mtry          = "tuneRF2",
                                internal.end.path = TRUE,
                                ntree=20)

plot(x          = hie.RF.00,
     text.size  = 9,
     split.text = 10)

```

PlotImportanceHie

*Plotting function for the hierarchical importance***Description**

This function takes the 4 column list generated by ImportanceHie and plots the importance of each variable in each local classifier as either a bubble plot or a heat map.

**Usage**

```

PlotImportanceHie(input.data, X.data = 2, Y.data = 3, imp.data = 4,
  plot.type = "Tile", imp.title = colnames(input.data)[imp.data],
  X.Title = colnames(input.data)[X.data],
  Y.Title = colnames(input.data)[Y.data], low.col = "blue",
  high.col = "red", geom.tile.bor.col = "white", pos.col = "green",
  zero.col = "white", neg.col = "red", supp.warn = TRUE, ...)

```

**Arguments**

<code>input.data</code>	Data frame, data to be used for plotting.
<code>X.data</code>	The column number in <code>input.data</code> that contains the categories for the X axis.
<code>Y.data</code>	The column number in <code>input.data</code> that contains the categories for the Y axis.
<code>imp.data</code>	The column number in <code>input.data</code> that contains the variable importance values (or any other quantitative vector) that will scale the bubble size or tile color.
<code>plot.type</code>	The type of plot to produce. Can be either "Tile" (default) "Bubble".
<code>imp.title</code>	Character, title for the Importance legend.
<code>X.Title</code>	Character, title for the x axis.
<code>Y.Title</code>	Character, title for the y axis.
<code>low.col</code>	Character, if <code>plot.type</code> is set to "Tile", the color to use for the lowest importance value.
<code>high.col</code>	Character, if <code>plot.type</code> is set to "Tile", the color to use for the highest importance value.
<code>geom.tile.bor.col</code>	Character, if <code>plot.type</code> is set to "Tile", the color to use for the border of tiles. Set to NA for no tile borders.
<code>pos.col</code>	Character, if <code>plot.type</code> is set to "Bubble", the color to use for importance values > 0.
<code>zero.col</code>	Character, if <code>plot.type</code> is set to "Bubble", the color to use for importance values = 0.



```

                                supp.warn      = TRUE)

# further editing of the plot with ggplot2
impor.plot.tile.2 <- impor.plot.tile$plot +
  theme(axis.title.y = element_text(size = 20,
                                     colour = "red"))

impor.plot.tile.2 <- impor.plot.tile.2 +
  ggtitle("The mean decrease in accuracy of each
  explanatory variable in each local classifier")
impor.plot.tile.2

#Bubble
PlotImportanceHie(input.data = Importance.hie.RF,
                  X.data      = 2,
                  Y.data      = 3,
                  imp.data     = 4,
                  plot.type    = "Bubble")

#Bubble, black and white
PlotImportanceHie(input.data = Importance.hie.RF,
                  X.data      = 2,
                  Y.data      = 3,
                  imp.data     = 4,
                  plot.type    = "Bubble",
                  pos.col      = "black",
                  zero.col     = "gray50",
                  neg.col      = "white")

```

---

predict.HRF	<i>For all cases, the proportion of OOB votes that each class received in each local randomForest classifier.</i>
-------------	---

---

## Description

The function takes as input an object of class "HRF". For each case in the original training data or in new.data, the function extracts the proportion of OOB votes for each node (internal and terminal) in each local classifier.

## Usage

```

## S3 method for class 'HRF'
predict(object, train.predict = TRUE, new.data = NULL,
        new.data.case.ID = 1, new.data.exp.var = NULL, bind.train.new = FALSE,
        ...)

```

## Arguments

object	Object of class "HRF" - the output of RunHRF.
train.predict	Logical, if TRUE, returns for each case in the training data the proportion of OOB votes that each class received in each local randomForest classifier. If FALSE, only the votes for new.data are returned.
new.data	Optional data frame containing additional cases that were not a part of the original training set, for which the proportion of votes should be extracted.

<code>new.data.case.ID</code>	Integer, specifying the column number with the <code>case.ID</code> in the <code>new.data</code> data frame. The <code>case.ID</code> values should be unique and different from those in the training data.
<code>new.data.exp.var</code>	Vector of integers, specifying the columns of <code>new.data</code> that contains the same set of explanatory variables as used in the training of <code>hie.RF</code> . Before running the <code>RunHRF</code> , we recommend using the <code>link{JoinLevels}</code> function on each categorical variables to ensure the extraction of votes for <code>new.data</code> .
<code>bind.train.new</code>	Logical, if <code>TRUE</code> the cases in the training set and <code>new.data</code> will be returned as one output data frame (along with the two separate ones).
<code>...</code>	Optional parameters to be passed to the low level functions.

### Details

For the training data, only OOB votes are used in each local classifier.

Inherited from [randomForest](#), predictions for `new.data` cannot be made if the `new.data` contains factor levels (both for classes and for categorical explanatory variables) that were not represented in the training data. Before running `RunHRF` we recommend either sub-setting the training and new data from one general data frame or running the [JoinLevels](#) function on each categorical variable.

### Value

a list consisting of up to three of the following data frames:

- `'prop.vote.train'` -  
The proportion of OOB votes that each case from the training data-set received in each local classifier for each class.
- `'prop.vote.new'` -  
The proportion of OOB votes that each case from the `new.data` data-set received in each local classifier for each class.
- `'prop.vote.full'` -  
Bind of `prop.vote.train` and `prop.vote.new` if `bind.train.new` is `TRUE`.

### Author(s)

Yoni Gavish <gavishyoni@gmail.com>

### See Also

[RunHRF](#) for running a hierarchical randomForest analysis, [GetMultPropVotes](#) for estimating the multiplicative proportion of votes from the output of `predict.HRF`, [PerformanceHRF](#) for assessing performance and accuracy, [PredictNewHRF](#) for predicting crisp class for each case of `new.data`.

### Examples

```
set.seed(354)
random.hrf <- RandomHRF(num.term.nodes = 20, tree.depth = 4)
train.data <- random.hrf$train.data
new.data   <- random.hrf$new.data

# run HRF
hie.RF.random <- RunHRF(train.data = train.data,
```

```

case.ID      = "case.ID",
hie.levels = c(2:(random.hrf$call$tree.depth + 1)))

# predict only for the training data
Predict.HRF.train  <- predict(hie.RF.random)
prop.votes.lrf.train <- Predict.HRF.train$prop.vote.train

# predict only for new.data
Predict.HRF.new <- predict(object      = hie.RF.random,
                           train.predict = FALSE,
                           new.data     = new.data,
                           new.data.case.ID = 1,
                           new.data.exp.var = c(2:ncol(new.data)),
                           bind.train.new  = FALSE)
prop.votes.lrf.new <- Predict.HRF.new$prop.vote.new

# predict for training and new data + bind

Predict.HRF.both <- predict(object = hie.RF.random,
                           train.predict = TRUE,
                           new.data     = new.data,
                           new.data.case.ID = 1,
                           new.data.exp.var = c(2:ncol(new.data)),
                           bind.train.new  = TRUE)
attributes(Predict.HRF.both)
prop.votes.lrf.both <- Predict.HRF.both$prop.vote.full

# the prop.votes.lrf.both data frame contains
# one additional column: 'train.or.test'
# cases from the training data set are listed as train
# cases from the new.data data set are listed as test
names(prop.votes.lrf.both)[1]
levels(prop.votes.lrf.both$train.or.test)

```

---

PredictNewHRF

*Predict crisp class for new.data*


---

## Description

This function takes as input a new.data data frame with the same explanatory variables as those used in hie.RF. Next, the [predict.HRF](#) function is applied to extract the proportion of votes that each case in new.data received for each node in each local classifier. Finally, up to three methods of selecting a single terminal node (as given by [crisp.rule](#) and described in [PerformanceHRF](#)) are applied for each case.

## Usage

```

PredictNewHRF(hie.RF, new.data, new.data.case.ID = 1,
              new.data.exp.var = c(2:ncol(new.data)),
              crisp.rule = c("multiplicative.majority", "multiplicative.permutation",
                             "stepwise.majority"), perm.num = 500, div.logical = TRUE,
              div.print = 25, ...)

```

**Arguments**

<code>hie.RF</code>	Object of class "HRF" - the output of <code>RunHRF</code> .
<code>new.data</code>	Data frame containing additional cases that were not a part of the original training set.
<code>new.data.case.ID</code>	Integer, specifying the column number with the case.ID in the new.data data frame. The case.ID values should be unique and different from those in the training data.
<code>new.data.exp.var</code>	Vector of integers, specifying the columns of new.data that contains the same set of explanatory variables as used in the training of <code>hie.RF</code> .
<code>crisp.rule</code>	The method of selecting a single crisp class from the proportion of votes. See details in <a href="#">PerformanceHRF</a> .
<code>perm.num</code>	Integer, number of random permutations for each case if 'multiplicative.permutation' is applied.
<code>div.logical</code>	Logical, if TRUE progress when 'multiplicative.permutation' is applied will be printed every <code>div.print</code> permutations.
<code>div.print</code>	See above.
<code>...</code>	Optional parameters to be passed to low level functions.

**Details**

If the observed class of new.data are known, the function [PerformanceNewHRF](#) will perform all the steps in this function and will add the estimation of performance measures. Inherited from [randomForest](#), predictions for new.data cannot be made if the new.data contains factor levels (both for classes and for categorical explanatory variables) that were not represented in the training data. Before running `RunHRF` we recommend either sub-setting the training and new data from one general data frame or running the [JoinLevels](#) function on each categorical variable.

**Value**

A list with the following components:

- `'raw.vote'` - Data frame containing for each case, the proportion of votes for each node in each local classifier (the output of [predict.HRF](#)).
- `'crisp.case.class'` - Data frame containing the crisp class for each case based on all options defined by `crisp.rule`.
- `'multiplicative.prop'` - Optional data frame with the multiplicative proportion of votes (the output of [GetMultPropVotes](#)). Will be returned if "multiplicative.majority" or "multiplicative.permutation" were selected.
- `'call'` - The call to function `PredictNewHRF`.

**Author(s)**

Yoni Gavish <[gavishyoni@gmail.com](mailto:gavishyoni@gmail.com)>

**See Also**

[RunHRF](#) for running a hierarchical randomForest analysis, [PerformanceHRF](#) for performance analysis, [PerformanceNewHRF](#) for performance analysis on new. data with observed terminal node,

**Examples**

```
# create a random HRF dataset and RunHRF
set.seed(354)
random.hrf <- RandomHRF(num.term.nodes = 20, tree.depth = 4)
train.data <- random.hrf$train.data
new.data <- random.hrf$new.data
hie.RF.random <- RunHRF(train.data = train.data,
                        case.ID = "case.ID",
                        hie.levels = c(2:(random.hrf$call$tree.depth + 1)))

# predict for new.data
pred.new.hrf <- PredictNewHRF(hie.RF = hie.RF.random,
                             new.data = new.data,
                             crisp.rule = c("stepwise.majority",
                                             "multiplicative.majority",
                                             "multiplicative.permutation"),
                             perm.num = 10,
                             div.print = 2)

# extract values
names(pred.new.hrf)
pred.new.votes <- pred.new.hrf$raw.votes
pred.new.mult.prop <- pred.new.hrf$multiplicative.prop
pred.new.crisp.class <- pred.new.hrf$crisp.case.class
pred.new.call <- pred.new.hrf$call
```

---

RandomHRF	<i>Create random class hierarchy with random training and new.data cases.</i>
-----------	---

---

**Description**

This function uses the [random.hierarchical.data](#) function of the package "treemap" to create a tree hierarchy with a given number of terminal nodes and with a given number of hierarchical levels. Next, random quantitative and categorical explanatory variables are added to a user-defined number of training and new. data cases of each terminal node.

**Usage**

```
RandomHRF(num.term.nodes = 25, tree.depth = 5, cases.t.node.train = 15,
          cases.t.node.new = 25, new.data.observed = FALSE, numer.exp.var = 10,
          categ.exp.var = 10, case.ID = "case.ID", ...)
```

**Arguments**

`num. term. nodes` Integer, the number of terminal nodes at the lowest level.  
`tree. depth` Integer, the number of levels in the class hierarchy.

<code>cases.t.node.train</code>	Integer, the number of training cases to create for each terminal node in the training data.
<code>cases.t.node.new</code>	Integer, the number of training cases to create for each terminal node in new.data.
<code>new.data.observed</code>	Logical, if FALSE (default)- the new.data will not contain an observed class hierarchy for each case. If TRUE an observed class hierarchy will be added to each case of new.data.
<code>numer.exp.var</code>	Integer, number of numeric explanatory variables.
<code>categ.exp.var</code>	Integer, number of categorical explanatory variables.
<code>case.ID</code>	Character, the name of the column that will contain the case.IDs.
<code>...</code>	Optional parameters to be passed to low level functions.

### Details

The numeric explanatory variables are based on sampling with replacement from a vector of integers starting with 1 and ending with a random number between 1 and 50. The categorical explanatory variables are based on sampling with replacement of the first 10 letters. Future version may allow greater control on the creation of explanatory variables.

Setting `cases.t.node.train = 0` or `cases.t.node.new = 0` will result with NA for `train.data` or `new.data`, respectively.

### Value

A list consisting of:

- `'train.data'` -  
Data frame with the training data-set.
- `'new.data'` -  
Data frame with the new.data data-set.
- `'call'` -  
The call to RandomHRF.

### Note

Currently, the function can only return hierarchies for which all terminal nodes occur at the lowest level. Future versions may allow trimmed hierarchy trees.

### Author(s)

Yoni Gavish <gavishyoni@gmail.com>

### Examples

```
set.seed(354)
random.hrf <- RandomHRF(num.term.nodes = 20,
                        tree.depth      = 4)

train.data <- random.hrf$train.data
new.data   <- random.hrf$new.data
random.hrf$call$tree.depth
# example with new.data.observed =TRUE
```



```

set.seed(354)
random.hRF <- RandomHRF(num.term.nodes = 15,
                        tree.depth      = 6,
                        new.data.observed = TRUE)
train.data <- random.hRF$train.data
new.data   <- random.hRF$new.data

```

RunHRF

*Run the Hierarchical randomForest on the training data.*

## Description

The main function of the package that identifies the hierarchical class structure from the input training data and runs a randomForest algorithm as the local classifier at each internal node that has more than one child node. Returns an object of class "HRF" that contains all the local randomForest objects along with additional information on the hierarchical structure.

## Usage

```

RunHRF(train.data, case.ID, hie.levels, internal.end.path = FALSE,
        end.path.name = "END.PATH", root.include = FALSE,
        root.name = "TREE.ROOT", exp.var = NA, mtry = "tuneRF", ntree = 500,
        importance = TRUE, proximity = TRUE, keep.forest = TRUE,
        keep.inbag = TRUE, ...)

```

## Arguments

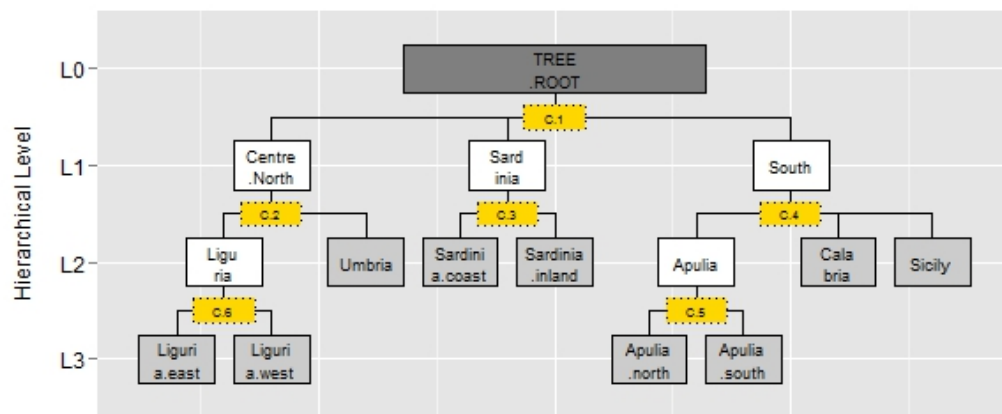
train.data	Data frame with the training data.
case.ID	A character or integer, specifying the name or column number used as case IDs in train.data. Case ID values must be unique.
hie.levels	A vector of characters or integers, containing the names or column numbers of the hierarchical levels in train.data. Order of columns in train.data should be from the tree root to the terminal nodes.
internal.end.path	Logical, FALSE (default) if all terminal nodes are in the lowest level of the class hierarchy. TRUE if some terminal nodes are not at the lowest hierarchy level.
end.path.name	Character - the factor value used in level $i + 1$ for terminal nodes ending in level $i$ . The only factor in the class hierarchy that can appear in more than one hierarchy level.
root.include	Logical, TRUE if root.name is included in hie.levels.
root.name	Character - name to use for the tree root.
exp.var	A vector of characters or integers, containing the names or column numbers in train.data with the explanatory variables. Default takes all columns other than case.ID and hie.levels.
mtry	Number of variables randomly sampled as candidates at each split. Note that the default is to use tuneRF function of randomForest package for each local classifier. Setting mtry to <a href="#">tuneRF2</a> will use a slightly different version of tuneRF.
ntree	Number of trees to grow in each local classifier. See ?randomForest for additional details.

importance	Logical, if TRUE importance of variables will be assessed and saved at each local classifier. See ?randomForest for additional details.
proximity	Logical, if TRUE, proximity will be calculated for each local classifier. See ?randomForest for additional details.
keep.forest	Logical, if TRUE (recommended) the forest of each local classifier will be retained. If FALSE, the predict and performance functions will return an error.
keep.inbag	Logical, if TRUE an $n$ by $n$ tree matrix be returned that keeps track of which cases are in-bag in which trees. $n$ being the number of cases in the training set of a local classifier.
...	Optional parameters to be passed to low level functions.

### Details

Implements Breiman's (2001) random forest algorithm based on the [randomForest](#) package for classification.

In hierarchical randomForest, the additional information on class hierarchy is used to train more than 1 local classifier. Each case of a certain terminal node is also used as a training case for any internal nodes in the path leading from the tree root to the terminal node. For example, in the "OliveOilHie" data-set (figure below), a training case for the terminal class *Apulia.north* is used in local classifier *C.5* to separate *Apulia.north* and *Apulia.south*. The same case also represent *Apulia* in local classifier *C.4* to separate *Apulia* from *Calabria* and *Sicily*. Finally, the same case represent *South* in local classifier *C.1*.



### Value

An object of class "HRF" that contains the following:

- 'hier.struc' - See section below.
- 'train.data.ready' - Data frame containing the training data after restructuring to the usable format of RunHRF.
- 'case.ID' - Column number in train.data.ready containing the case IDs.
- 'path.name' - Column number in train.data.ready containing the path names.

- 'hie.levels' -  
Column number in train.data.ready containing the hierarchical information.
- 'exp.var' -  
Column number in train.data.ready containing the explanatory variables.
- 'all.local.RF' -  
The main large list with all the information on each local classifier. For each local classifier, the function returns a list consisting of three objects:
  - 'local.lRF.info' -  
The information on the local randomForest as found in the \$hier.struc\$lRF.info data frame.
  - 'local.data' -  
The Case ID's of all cases that were used as training data for the local randomForest.
  - 'local.RF' -  
an object of class ranomForest for the local classifier. See package randomForest for details.
- 'order.local.RF' -  
data frame containing the order in which local randomForests are stored in the all.local.RF list.
- 'call' -  
the call to function RunHRF.

~~~~~

### The 'hier.struc' list

The 'hier.struc' is a list with 3 data frames:

- \$lRF.info -  
Information on each local randomForest, containing the following columns:
    - 'classifier.ID' -  
The name of the local classifier, labelled as C.1, C.2...
    - 'par.level' -  
The level in the class hierarchy of the parent node of the local classifier.
    - 'par.name' -  
The name of the parent node in the local classifier.
    - 'par.clas.id' -  
The classifier.ID in which the parent node was classified.
    - 'num.sib.tot' -  
The number of sibling nodes in the local classifier.
    - 'num.sib.ter' -  
The number of terminal sibling nodes in the local classifier.
    - 'num.sib.int' -  
The number of internal sibling nodes in the local classifier.
- ~~~~~
- \$nodes.info -  
Information on each internal or terminal node in the class hierarchy, containing the following columns:
    - 'node.name' -  
The name of the node.

- 'node.level' -  
The level of the node in the class hierarchy.
- 'node.freq' -  
The number of times the node appears in the training data.
- 'node.par.lev' -  
The level in which the parent of the node resides.
- 'node.par.name' -  
The name of the node's parent node.
- 'term.int.node' -  
If the node is terminal or internal.
- 'clas.yes.no' -  
If the node is the parent node in a local classifier.
- 'classifier.ID' -  
If the node is the parent node in a local classifier, the name of the local classifier.
- 'classified.in' -  
The name of the local classifier in which the node is classified.
- 'lev.above.clas.in' -  
The number of levels above the node's level in which it is classified.

~~~~~

- \$unique.path -  
Information on all the unique paths from the tree root to each of the terminal nodes. Include two additional columns for root.name and end.path.name.

### Author(s)

Yoni Gavish <gavishyoni@gmail.com>

### References

1. **Breiman L.** (2001) Random forests. *Machine Learning* 45:5-32.

### See Also

[predict.HRF](#) for extracting the proportion of votes, [plot.HRF](#) for plotting the class structure, [ImportanceHie](#) for variable importance, [PerformanceHRF](#) for assessing performance and accuracy, [PerformanceFlatRF](#) for running an "HRF" object in a flat classifier and assessing performance.

### Examples

```
# create random HRF data
set.seed(354)
random.hrf <- RandomHRF(num.term.nodes = 20, tree.depth = 4)
train.data <- random.hrf$train.data

# Run the Hierarchial randomForest
hie.RF.random <- RunHRF(train.data = train.data,
                        case.ID    = "case.ID",
                        hie.levels = c(2:(random.hrf$call$tree.depth + 1)))

# S3 method for plot -> the class hierarchy
plot(hie.RF.random)

# extracting information
```

```

lRF.info      <- hie.RF.random$hier.struc$lRF.info
nodes.info    <- hie.RF.random$hier.struc$nodes.info
unique.path   <- hie.RF.random$hier.struc$unique.path
train.data.ready <- hie.RF.random$train.data.ready
case.ID       <- hie.RF.random$case.ID
path.name     <- hie.RF.random$path.name
hie.levels    <- hie.RF.random$hie.levels
exp.var       <- hie.RF.random$exp.var
all.local.RF  <- hie.RF.random$all.local.RF
order.local.RF <- hie.RF.random$order.local.RF
fun.call      <- hie.RF.random$call

# extracting the info for local classifier C.2
c.2.local.classifier <- all.local.RF[[order.local.RF[
  order.local.RF$classifier.ID == "C.2", 2]]]

# structure for each local classifier
# info on the local classifier
c.2.local.lRF.info <- c.2.local.classifier$local.lRF.info
# case.ID that were used to train the randomForest
c.2.local.case.ID <- c.2.local.classifier$local.data
# object of class randomForest
c.2.local.RF      <- c.2.local.classifier$local.RF
class(c.2.local.RF)

#####
# the OliveOilHie data-set contains terminal nodes at levels 2 and 3
# RunHRF Will return an error if internal.end.path is not set to TRUE

data(OliveOilHie)
# don't run - an error message is returned

# hie.RF.00 <- RunHRF(train.data      = OliveOilHie,
#                     case.ID        = "case.ID",
#                     hie.levels     = c(2:4),
#                     mtry           = "tuneRF2",
#                     internal.end.path = FALSE)

# no error message
hie.RF.00 <- RunHRF(train.data      = OliveOilHie,
                    case.ID        = "case.ID",
                    hie.levels     = c(2:4),
                    mtry           = "tuneRF2",
                    internal.end.path = TRUE)

plot(x = hie.RF.00, text.size = 9, split.text = 10)

```

tuneRF2

*the tuneRF function of randomForest after correcting for error relating to errorOld=0*

## Description

The [tuneRF](#) function of the randomForest package may return an error when errorOld reaches 0 when exploring mtry. This function returns the mtry reached before the error. In addition, the default option for "plot" in tuneRF is changes to FALSE.

## Usage

```
tuneRF2(x, y, mtryStart = if (is.factor(y)) floor(sqrt(ncol(x))) else
  floor(ncol(x)/3), ntreeTry = 50, stepFactor = 2, improve = 0.05,
  trace = TRUE, plot = FALSE, doBest = FALSE, ...)
```

## Arguments

x	See <a href="#">tuneRF</a> in the package randomForest for details.
y	See <a href="#">tuneRF</a> in the package randomForest for details.
mtryStart	See <a href="#">tuneRF</a> in the package randomForest for details.
ntreeTry	See <a href="#">tuneRF</a> in the package randomForest for details.
stepFactor	See <a href="#">tuneRF</a> in the package randomForest for details.
improve	See <a href="#">tuneRF</a> in the package randomForest for details.
trace	See <a href="#">tuneRF</a> in the package randomForest for details.
plot	See <a href="#">tuneRF</a> in the package randomForest for details.
doBest	See <a href="#">tuneRF</a> in the package randomForest for details.
...	Optional parameters to be passed to low level functions.

## Value

See [tuneRF](#) in the package randomForest for details.

## Acknowledgments and details

This function borrows the entire code from the function [tuneRF](#) from the package '[randomForest](#)'. The only change made here (other than a change of the default plot to FALSE) is for the line:

```
Improve <- 1 - errorCur/errorOld
```

From the original [tuneRF](#) code that returns NA for errorOld = 0. The NA result with an overall error of [tuneRF](#). This line was replaced with:

```
if(errorCur==0){Improve <- improve}
if(errorCur!=0){Improve <- 1 - errorCur/errorOld}
```

We recommend using [tuneRF2](#) only when [tuneRF](#) returns an error.

## Examples

```
data(OliveOilHie)
set.seed(250)
# note the "error in if (Improve > improve)..."
# further note the OOB error=0% and the 'Nan' in the printed info from tuneRF
# don't run
# hie.RF.OO <- RunHRF(train.data      = OliveOilHie,
#                    case.ID         = "case.ID",
#                    hie.levels      = c(2:4),
#                    mtry             = "tuneRF",
#                    internal.end.path = TRUE)

hie.RF.OO <- RunHRF(train.data      = OliveOilHie,
                    case.ID         = "case.ID",
                    hie.levels      = c(2:4),
                    mtry             = "tuneRF2",
                    internal.end.path = TRUE)
```

# Index

`binom.test`, [18](#)

`confusionMatrix`, [7](#), [17](#)

`GetMultPropVotes`, [5](#), [16](#), [17](#), [21](#), [28](#), [30](#)

`ggplot2`, [23](#), [26](#)

`HieFMeasure`, [3](#), [6](#), [14](#), [16–18](#), [21](#)

`HieRanFor`-package, [2](#)

`importance`, [9](#)

`ImportanceHie`, [2](#), [9](#), [26](#), [36](#)

`JoinLevels`, [7](#), [10](#), [28](#), [30](#)

`mcnemar.test`, [18](#)

`oliveoil`, [12](#)

`OliveOilHie`, [3](#), [11](#), [34](#)

`pdfCluster`, [12](#)

`PerformanceFlatRF`, [2](#), [13](#), [18](#), [36](#)

`PerformanceHRF`, [2](#), [5](#), [13](#), [14](#), [15](#), [20](#), [21](#), [23](#), [28–31](#), [36](#)

`PerformanceNewHRF`, [2](#), [18](#), [20](#), [30](#), [31](#)

`plot.HRF`, [2](#), [22](#), [36](#)

`PlotImportanceHie`, [10](#), [25](#)

`predict.HRF`, [2](#), [5](#), [16](#), [17](#), [20](#), [21](#), [27](#), [29](#), [30](#), [36](#)

`PredictNewHRF`, [2](#), [21](#), [28](#), [29](#)

`random.hierarchical.data`, [31](#)

`randomForest`, [4](#), [13](#), [28](#), [30](#), [34](#), [38](#)

`RandomHRF`, [31](#)

`RunHRF`, [2](#), [5](#), [9](#), [11](#), [14](#), [18](#), [21](#), [23](#), [28](#), [31](#), [33](#)

`tuneRF`, [37](#), [38](#)

`tuneRF2`, [33](#), [37](#)