

Package ‘HEMDAG’

October 8, 2019

Title Hierarchical Ensemble Methods for Directed Acyclic Graphs

Version 2.5.9

Author Marco Notaro [aut, cre] (<<https://orcid.org/0000-0003-4309-2200>>),
Alessandro Petrini [ctb] (<<https://orcid.org/0000-0002-0587-1484>>),
Giorgio Valentini [aut] (<<https://orcid.org/0000-0002-5694-3919>>)

Maintainer Marco Notaro <marco.notaro@unimi.it>

Description An implementation of several Hierarchical Ensemble Methods (HEMs) for Directed Acyclic Graphs (DAGs). 'HEMDAG' package: 1) reconciles flat predictions with the topology of the ontology; 2) can enhance the predictions of virtually any flat learning methods by taking into account the hierarchical relationships between ontology classes; 3) provides biologically meaningful predictions that always obey the true-path-rule, the biological and logical rule that governs the internal coherence of biomedical ontologies; 4) is specifically designed for exploiting the hierarchical relationships of DAG-structured taxonomies, such as the Human Phenotype Ontology (HPO) or the Gene Ontology (GO), but can be safely applied to tree-structured taxonomies as well (as FunCat), since trees are DAGs; 5) scales nicely both in terms of the complexity of the taxonomy and in the cardinality of the examples; 6) provides several utility functions to process and analyze graphs; 7) provides several performance metrics to evaluate HEMs algorithms. (Marco Notaro, Max Schubach, Peter N. Robinson and Giorgio Valentini (2017) <[doi:10.1186/s12859-017-1854-y](https://doi.org/10.1186/s12859-017-1854-y)>).

URL <https://hemdag-tutorials.readthedocs.io>

<https://github.com/marconotaro/HEMDAG>

<https://anaconda.org/bioconda/r-hemdag>

BugReports <https://github.com/marconotaro/HEMDAG/issues>

Depends R (>= 2.10)

License GPL (>= 3)

Encoding UTF-8

Repository CRAN

LazyLoad true

NeedsCompilation yes

Imports graph, RBGL, precrec, preprocessCore, methods, plyr, foreach,
iterators, doParallel, parallel

Suggests Rgraphviz

RoxygenNote 6.1.1

R topics documented:

HEMDAG-package	3
adj.upper.tri	4
ancestors	5
AUPRC	6
AUROC	7
build.consistent.graph	9
check.annotation.matrix.integrity	9
check.DAG.integrity	10
children	11
compute.flipped.graph	12
constraints.matrix	12
create.stratified.fold.df	13
descendants	14
distances.from.leaves	15
do.edges.from.HPO.obo	15
Do.flat.scores.normalization	16
Do.full.annotation.matrix	17
Do.GPAV	18
Do.GPAV.holdout	21
Do.heuristic.methods	24
Do.heuristic.methods.holdout	27
Do.HTD	30
Do.HTD.holdout	33
do.subgraph	35
do.submatrix	36
do.unstratified.cv.data	37
example.datasets	37
find.best.f	38
find.leaves	40
FMM	40
full.annotation.matrix	42
GPAV	43
GPAV.over.examples	44
GPAV.parallel	45
graph.levels	46
Heuristic-Methods	46
hierarchical.checkers	47
HTD-DAG	48
lexicographical.topological.sort	49
Multilabel.F.measure	50

normalize.max	51
parents	52
PXR	53
read.graph	54
read.undirected.graph	55
root.node	56
scores.normalization	56
specific.annotation.list	57
specific.annotation.matrix	58
stratified.cross.validation	58
TPR-DAG-cross-validation	60
TPR-DAG-holdout	64
TPR-DAG-variants	68
transitive.closure.annotations	71
tupla.matrix	72
weighted.adjacency.matrix	73
write.graph	74
Index	75

HEMDAG-package	<i>HEMDAG: Hierarchical Ensemble Methods for Directed Acyclic Graphs</i>
----------------	--

Description

The HEMDAG package:

- provides an implementation of several Hierarchical Ensemble Methods for DAGs;
- reconciles flat predictions with the topology of the ontology;
- can enhance predictions of virtually any flat learning methods by taking into account the hierarchical relationships between ontology classes;
- provides biologically meaningful predictions that obey the true-path-rule, the biological and logical rule that governs the internal coherence of biomedical ontologies;
- is specifically designed for exploiting the hierarchical relationships of DAG-structured taxonomies, such as the Human Phenotype Ontology (HPO) or the Gene Ontology (GO), but can be safely applied to tree-structured taxonomies as well (as FunCat), since trees are DAGs;
- scales nicely both in terms of the complexity of the taxonomy and in the cardinality of the examples;
- provides several utility functions to process and analyze graphs;
- provides several performance metrics to evaluate HEMs algorithms;

A comprehensive tutorial showing how to apply HEMDAG to real case studies in computational biology and precision medicine is available at <https://hemdag-tutorials.readthedocs.io>.

Details

The HEMDAG package implements the following Hierarchical Ensemble Methods for DAGs:

1. **HTD-DAG**: Hierarchical Top Down ([HTD-DAG](#));
2. **GPAV**: Generalized Pool-Adjacent Violators, *Burdakov et al.* ([GPAV](#));
3. **TPR-DAG**: True-Path Rule ([TPR-DAG-variants](#));
4. **DESCENS**: Descendants Ensemble Classifier ([TPR-DAG-variants](#));
5. **ISO-TPR**: Isotonic-True-Path Rule ([TPR-DAG-variants](#));
6. **MAX, AND, OR**: Heuristic Methods, *Obozinski et al.* ([Heuristic-Methods](#));

Author(s)

Marco Notaro¹ (<https://orcid.org/0000-0003-4309-2200>);
 Alessandro Petrini¹ (<https://orcid.org/0000-0002-0587-1484>);
 Giorgio Valentini¹ (<https://orcid.org/0000-0002-5694-3919>);

Maintainer: *Marco Notaro* <marco.notaro@unimi.it>

¹ **AnacletoLab**, Computational Biology and Bioinformatics Laboratory, Computer Science Department, University of Milan, Italy

References

Marco Notaro, Max Schubach, Peter N. Robinson and Giorgio Valentini, *Prediction of Human Phenotype Ontology terms by means of Hierarchical Ensemble methods*, BMC Bioinformatics 2017, 18(1):449, doi:[10.1186/s12859-017-1854-y](https://doi.org/10.1186/s12859-017-1854-y)

adj.upper.tri

Binary Upper Triangular Adjacency Matrix

Description

This function returns a binary square upper triangular matrix where rows and columns correspond to the nodes' name of the graph g.

Usage

```
adj.upper.tri(g)
```

Arguments

g a graph of class graphNEL representing the hierarchy of the class.

Details

The nodes of the matrix are topologically sorted (by using the `tsort` function of the **RBGL** package). Let's denote with `adj` our adjacency matrix. Then `adj` represents a partial order data set in which the class `j` dominates the class `i`. In other words, `adj[i,j]=1` means that `j` dominates `i`; `adj[i,j]=0` means that there is no edge between the class `i` and the class `j`. Moreover the nodes of `adj` are ordered such that `adj[i,j]=1` implies $i < j$, i.e. `adj` is upper triangular.

Value

an adjacency matrix which is square, logical and upper triangular.

Examples

```
data(graph);
adj <- adj.upper.tri(g);
```

ancestors	<i>Build ancestors</i>
-----------	------------------------

Description

Compute the ancestors for each node of a graph.

Usage

```
build.ancestors(g)

build.ancestors.per.level(g, levels)

build.ancestors.bottom.up(g, levels)
```

Arguments

<code>g</code>	a graph of class <code>graphNEL</code> . It represents the hierarchy of the classes.
<code>levels</code>	a list of character vectors. Each component represents a graph level and the elements of any component correspond to nodes. The level 0 coincides with the root node.

Value

`build.ancestors` returns a named list of vectors. Each component corresponds to a node x of the graph and its vector is the set of its ancestors including also x .

`build.ancestors.per.level` returns a named list of vectors. Each component corresponds to a node x of the graph and its vector is the set of its ancestors including also x . The nodes are ordered from root (included) to leaves.

`build.ancestors.bottom.up` a named list of vectors. Each component corresponds to a node x of the graph and its vector is the set of its ancestors including also x . The nodes are ordered from leaves to root (included).

See Also

[graph.levels](#)

Examples

```
data(graph);
root <- root.node(g);
anc <- build.ancestors(g);
lev <- graph.levels(g, root=root);
anc.tod <-build.ancestors.per.level(g,lev);
anc.bup <- build.ancestors.bottom.up(g,lev);
```

AUPRC	<i>AUPRC measures</i>
-------	-----------------------

Description

Function to compute Area under the Precision Recall Curve (AUPRC) through **precrec** package.

Usage

```
AUPRC.single.class(labels, scores, folds = NULL, seed = NULL)

AUPRC.single.over.classes(target, predicted, folds = NULL, seed = NULL)
```

Arguments

labels	vector of the true labels (0 negative, 1 positive examples).
scores	a numeric vector of the values of the predicted labels (scores).
folds	number of folds on which computing the AUPRC. If folds=NULL (def.), the AUPRC is computed one-shot, otherwise the AUPRC is computed averaged across folds.
seed	initialization seed for the random generator to create folds. Set seed only if folds≠NULL. If seed=NULL and folds≠NULL, the AUPRC averaged across folds is computed without seed initialization.
target	matrix with the target multilabel: rows correspond to examples and columns to classes. $target[i, j] = 1$ if example i belongs to class j , $target[i, j] = 0$ otherwise.
predicted	a numeric matrix with predicted values (scores): rows correspond to examples and columns to classes.

Details

The AUPRC (for a single class or for a set of classes) is computed either one-shot or averaged across stratified folds.

`AUPRC.single.class` computes the AUPRC just for a given class.

`AUPRC.single.over.classes` computes the AUPRC for a set of classes, returning also the averaged values across the classes.

For all those classes having zero annotations, the AUPRC is set to 0. These classes are discarded in the computing of the AUPRC averaged across classes, both when the AUPRC is computed one-shot or averaged across stratified folds.

Names of rows and columns of labels and predicted matrix must be provided in the same order, otherwise a stop message is returned.

Value

`AUPRC.single.class` returns a numeric value corresponding to the AUPRC for the considered class; `AUPRC.single.over.classes` returns a list with two elements:

- 1. `average`: the average AUPRC across classes;
- 2. `per.class`: a named vector with AUPRC for each class. Names correspond to classes.

Examples

```
data(labels);
data(scores);
data(graph);
root <- root.node(g);
L <- L[,-which(colnames(L)==root)];
S <- S[,-which(colnames(S)==root)];
PRC.single.class <- AUPRC.single.class(L[,3], S[,3], folds=5, seed=23);
PRC.over.classes <- AUPRC.single.over.classes(L, S, folds=5, seed=23);
```

AUROC	<i>AUROC measures</i>
-------	-----------------------

Description

Function to compute the Area under the ROC Curve through **precrec** package.

Usage

```
AUROC.single.class(labels, scores, folds = NULL, seed = NULL)

AUROC.single.over.classes(target, predicted, folds = NULL, seed = NULL)
```

Arguments

labels	vector of the true labels (0 negative, 1 positive examples).
scores	a numeric vector of the values of the predicted labels (scores).
folds	number of folds on which computing the AUROC. If folds=NULL (def.), the AUROC is computed one-shot, otherwise the AUROC is computed averaged across folds.
seed	initialization seed for the random generator to create folds. Set seed only if folds≠NULL. If seed=NULL and folds≠NULL, the AUROC averaged across folds is computed without seed initialization.
target	annotation matrix: rows correspond to examples and columns to classes. $target[i, j] = 1$ if example i belongs to class j , $target[i, j] = 0$ otherwise.
predicted	a numeric matrix with predicted values (scores): rows correspond to examples and columns to classes.

Details

The AUROC (for a single class or for a set of classes) is computed either one-shot or averaged across stratified folds.

`AUROC.single.class` computes the AUROC just for a given class.

`AUROC.single.over.classes` computes the AUROC for a set of classes, including their average values across all the classes.

For all those classes having zero annotations, the AUROC is set to 0.5. These classes are included in the computing of the AUROC averaged across classes, both when the AUROC is computed one-shot or averaged across stratified folds.

The AUROC is set to 0.5 to all those classes having zero annotations. Names of rows and columns of labels and predicted must be provided in the same order, otherwise a stop message is returned.

Value

`AUROC.single.class` returns a numeric value corresponding to the AUROC for the considered class; `AUPR.single.over.classes` returns a list with two elements:

1. average: the average AUROC across classes;
2. per.class: a named vector with AUROC for each class. Names correspond to classes.

Examples

```
data(labels);
data(scores);
data(graph);
root <- root.node(g);
L <- L[, -which(colnames(L)==root)];
S <- S[, -which(colnames(S)==root)];
AUC.single.class <- AUROC.single.class(L[,3], S[,3], folds=5, seed=23);
AUC.over.classes <- AUROC.single.over.classes(L, S, folds=5, seed=23);
```

build.consistent.graph

Build Consistent Graph

Description

build a graph in which all nodes are reachable from root.

Usage

```
build.consistent.graph(g = g, root = "00")
```

Arguments

g	an object of class graphNEL.
root	name of the class that is on the top-level of the hierarchy (def. root="00").

Details

all nodes not accessible from the root (if any) are removed from the graph and printed on stdout.

Value

a graph (as an object of class graphNEL) in which all nodes are accessible from root.

Examples

```
data(graph);
root <- root.node(g);
G <- graph::addNode(c("X", "Y", "Z"), g);
G <- graph::addEdge(c("X", "Y", "Z"), c("HP:0011844", "HP:0009810", "HP:0012385"), G);
G <- build.consistent.graph(G, root=root);
```

check.annotation.matrix.integrity

Annotation matrix checker

Description

This function assess the integrity of an annotation table in which a transitive closure of annotations was performed.

Usage

```
check.annotation.matrix.integrity(anc, ann.spec, ann)
```

Arguments

anc	list of the ancestors of the ontology.
ann.spec	the annotation matrix of the most specific annotations (0/1): rows are genes and columns are terms.
ann	the full annotation matrix (0/1), that is the matrix in which the transitive closure of the annotation was performed. Rows are examples and columns are classes.

Value

If the transitive closure of the annotations is well performed "OK" is returned, otherwise a message error is printed on the stdout.

See Also

[build.ancestors](#), [transitive.closure.annotations](#), [full.annotation.matrix](#)

Examples

```
data(graph);
data(labels);
anc <- build.ancestors(g);
tca <- transitive.closure.annotations(L, anc);
check.annotation.matrix.integrity(anc, L, tca);
```

check.DAG.integrity *DAG checker*

Description

This function assess the integrity of a DAG.

Usage

```
check.DAG.integrity(g, root = "00")
```

Arguments

g	a graph of class graphNEL. It represents the hierarchy of the classes.
root	name of the class that is on the top-level of the hierarchy (def. root="00").

Value

If all the nodes are accessible from the root "DAG is OK" is printed, otherwise a message error and the list of the not accessible nodes is printed on the stdout.

Examples

```
data(graph);
root <- root.node(g);
check.DAG.integrity(g, root=root);
```

children

Build children

Description

Compute the children for each node of a graph.

Usage

```
build.children(g)

get.children.top.down(g, levels)

get.children.bottom.up(g, levels)
```

Arguments

<code>g</code>	a graph of class <code>graphNEL</code> . It represents the hierarchy of the classes.
<code>levels</code>	a list of character vectors. Each component represents a graph level and the elements of any component correspond to nodes. The level 0 coincides with the root node.

Value

`build.children` returns a named list of vectors. Each component corresponds to a node x of the graph and its vector is the set of its children.

`get.children.top.down` returns a named list of character vectors. Each component corresponds to a node x of the graph (i.e. parent node) and its vector is the set of its children. The nodes are ordered from root (included) to leaves.

`get.children.bottom.up` returns a named list of character vectors. Each component corresponds to a node x of the graph (i.e. parent node) and its vector is the set of its children. The nodes are ordered from leaves (included) to root.

See Also

[graph.levels](#)

Examples

```
data(graph);
root <- root.node(g);
children <- build.children(g);
lev <- graph.levels(g, root=root);
children.tod <- get.children.top.down(g,lev);
children.bup <- get.children.bottom.up(g,lev);
```

```
compute.flipped.graph Flip Graph
```

Description

Compute a directed graph with edges in the opposite direction.

Usage

```
compute.flipped.graph(g)
```

Arguments

`g` a graphNEL directed graph

Value

a graph (as an object of class graphNEL) with edges in the opposite direction w.r.t. `g`.

Examples

```
data(graph);
g.flipped <- compute.flipped.graph(g);
```

```
constraints.matrix Constraints Matrix
```

Description

This function returns a matrix with two columns and as many rows as there are edges. The entries of the first columns are the index of the node the edge comes from (i.e. children nodes), the entries of the second columns indicate the index of node the edge is to (i.e. parents nodes). Referring to a DAG this matrix defines a partial order.

Usage

```
constraints.matrix(g)
```

Arguments

`g` a graph of class `graphNEL`. It represents the hierarchy of the classes.

Value

a constraints matrix w.r.t the graph `g`.

Examples

```
data(graph);  
m <- constraints.matrix(g);
```

```
create.stratified.fold.df
```

DataFrame for Stratified Cross Validation

Description

Create a data frame for stratified cross-validation.

Usage

```
create.stratified.fold.df(labels, scores, folds = 5, seed = 23)
```

Arguments

<code>labels</code>	vector of the true labels (0 negative, 1 positive).
<code>scores</code>	a numeric vector of the values of the predicted labels.
<code>folds</code>	number of folds of the cross validation (def. <code>folds=5</code>).
<code>seed</code>	initialization seed for the random generator to create folds (def. <code>seed=23</code>). If <code>seed=NULL</code> , the stratified folds are generated without seed initialization.

Details

the folds are *stratified*, i.e. contain the same amount of positive and negative examples.

Value

a data frame with three columns:

- `scores`: contains the predicted scores;
- `labels`: contains the labels as `pos` or `neg`;
- `folds`: contains the index of the fold in which the example falls. The index can range from 1 to the number of folds.

Examples

```
data(labels);
data(scores);
df <- create.stratified.fold.df(L[,3], S[,3], folds=5, seed=23);
```

descendants

Build descendants

Description

Compute the descendants for each node of a graph.

Usage

```
build.descendants(g)

build.descendants.per.level(g, levels)

build.descendants.bottom.up(g, levels)
```

Arguments

<code>g</code>	a graph of class <code>graphNEL</code> . It represents the hierarchy of the classes.
<code>levels</code>	a list of character vectors. Each component represents a graph level and the elements of any component correspond to nodes. The level 0 coincides with the root node.

Value

`build.descendants` returns a named list of vectors. Each component corresponds to a node x of the graph, and its vector is the set of its descendants including also x .

`build.descendants.per.level` returns a named list of vectors. Each component corresponds to a node x of the graph and its vector is the set of its descendants including also x . The nodes are ordered from root (included) to leaves.

`build.descendants.bottom.up` returns a named list of vectors. Each component corresponds to a node x of the graph and its vector is the set of its descendants including also x . The nodes are ordered from leaves to root (included).

See Also

[graph.levels](#)

Examples

```
data(graph);
root <- root.node(g);
desc <- build.descendants(g);
lev <- graph.levels(g, root=root);
desc.tod <- build.descendants.per.level(g,lev);
desc.bup <- build.descendants.bottom.up(g,lev);
```

distances.from.leaves *Distances from leaves*

Description

This function returns the minimum distance of each node from one of the leaves of the graph.

Usage

```
distances.from.leaves(g)
```

Arguments

g a graph of class graphNEL. It represents the hierarchy of the classes.

Value

a named vector. The names are the names of the nodes of the graph *g*, and their values represent the distance from the leaves. A value equal to 0 is assigned to the leaves, 1 to nodes with distance 1 from a leaf and so on.

Examples

```
data(graph);
dist.leaves <- distances.from.leaves(g);
```

do.edges.from.HPO.obo *Parse an HPO OBO file*

Description

Read an HPO OBO file (**HPO**) and write the edges of the DAG on a plain text file. The format of the file is a sequence of rows and each row corresponds to an edge represented through a pair of vertices separated by blanks.

Usage

```
do.edges.from.HPO.obo(obofile = "hp.obo", file = "edge.file")
```

Arguments

obofile	an HPO OBO file. The extension of the obofile can be or plain format (".txt") or compressed (".gz").
file	name of the file of the edges to be written. The extension of the file can be or plain format (".txt") or compressed (".gz").

Details

a faster and more flexible parser to handle *obo* file can be found [here](#).

Value

a text file representing the edges in the format: source destination (i.e. one row for each edge).

Examples

```
## Not run:
hpobo <- "http://purl.obolibrary.org/obo/hp.obo";
do.edges.from.HPO.obo(obofile=hpobo, file="hp.edge");
## End(Not run)
```

Do.flat.scores.normalization

Flat scores normalization

Description

High level functions to normalize a flat scores matrix w.r.t. max normalization (MaxNorm) or quantile normalization (Qnorm).

Usage

```
Do.flat.scores.normalization(norm.type = "MaxNorm",
  flat.file = flat.file, flat.dir = flat.dir,
  flat.norm.dir = flat.norm.dir)
```

Arguments

norm.type	can be one of the following two values: <ul style="list-style-type: none"> • MaxNorm (def.): each score is divided w.r.t. the max of each class; • Qnorm: a quantile normalization is applied. Library preprocessCore is used.
flat.file	name of the flat scores matrix (without rda extension).
flat.dir	relative path to folder where flat scores matrix is stored .
flat.norm.dir	the directory where the normalized flat scores matrix must be stored.

Details

To apply the quantile normalization the **preprocessCore** package must be installed.

Value

the matrix of the scores flat normalized w.r.t. MaxNorm or Qnorm in flat.norm.dir.

Examples

```
data(scores);
tmpdir <- paste0(tmpdir(), "/");
save(S, file=paste0(tmpdir, "scores.rda"));
flat.dir <- flat.norm.dir <- tmpdir;
norm.types <- c("MaxNorm", "Qnorm");
for(norm.type in norm.types){
  Do.flat.scores.normalization(norm.type=norm.type, flat.file="scores",
    flat.dir=flat.dir, flat.norm.dir=flat.norm.dir);
}
```

Do.full.annotation.matrix

Do full annotation matrix

Description

High-level function to obtain a full annotation matrix, that is a matrix in which the transitive closure of annotations was performed, respect to a given weighted adjacency matrix.

Usage

```
Do.full.annotation.matrix(anc.file.name = anc.file.name,
  anc.dir = anc.dir, net.file = net.file, net.dir = net.dir,
  ann.file.name = ann.file.name, ann.dir = ann.dir,
  output.name = output.name, output.dir = output.dir)
```

Arguments

anc.file.name	name of the file containing the list for each node the list of all its ancestor (without rda extension).
anc.dir	relative path to directory where the ancestor file is stored.
net.file	name of the file containing the weighted adjacency matrix of the graph (without rda extension).
net.dir	relative path to directory where the weighted adjacency matrix is stored.
ann.file.name	name of the file containing the matrix of the most specific annotations (without rda extension).

<code>ann.dir</code>	relative path to directory where the matrix of the most specific annotation is stored.
<code>output.name</code>	name of the output file without rda extension.
<code>output.dir</code>	relative path to directory where the output file must be stored.

Value

a full annotation matrix T , that is a matrix in which the transitive closure of annotations was performed. Rows correspond to genes of the input weighted adjacency matrix and columns to terms. $T[i, j] = 1$ means that gene i is annotated for the term j , $T[i, j] = 0$ means that gene i is not annotated for the term j .

See Also

[full.annotation.matrix](#)

Examples

```
data(graph);
data(labels);
data(wadj);
anc <- build.ancestors(g);
tmpdir <- paste0(tempdir(), "/");
save(g, file=paste0(tmpdir, "graph.rda"));
save(L, file=paste0(tmpdir, "labels.rda"));
save(W, file=paste0(tmpdir, "wadj.rda"));
save(anc, file=paste0(tmpdir, "ancestors.rda"));
anc.dir <- net.dir <- ann.dir <- output.dir <- tmpdir;
anc.file.name <- "ancestors";
net.file <- "wadj";
ann.file.name <- "labels";
output.name <- "full.ann.matrix";
Do.full.annotation.matrix(anc.file.name=anc.file.name, anc.dir=anc.dir, net.file=net.file,
  net.dir=net.dir, ann.file.name=ann.file.name, ann.dir=ann.dir, output.name=output.name,
  output.dir=output.dir);
```

Description

High level function to correct the computed scores in a hierarchy according to the GPAV algorithm.

Usage

```
Do.GPAV(norm = TRUE, norm.type = NULL, W = NULL, parallel = FALSE,
        ncores = 1, folds = 5, seed = 23, n.round = 3,
        f.criterion = "F", recall.levels = seq(from = 0.1, to = 1, by = 0.1),
        compute.performance = FALSE, flat.file = flat.file,
        ann.file = ann.file, dag.file = dag.file, flat.dir = flat.dir,
        ann.dir = ann.dir, dag.dir = dag.dir,
        hierScore.dir = hierScore.dir, perf.dir = perf.dir)
```

Arguments

norm	boolean value: <ul style="list-style-type: none"> • TRUE (def.): the flat scores matrix has been already normalized in according to a normalization method; • FALSE: the flat scores matrix has not been normalized yet. See the parameter <code>norm.type</code> for which normalization can be applied;
norm.type	can be one of the following three values: <ol style="list-style-type: none"> 1. NULL (def.): set <code>norm.type</code> to NULL if and only if the parameter <code>norm</code> is set to TRUE; 2. MaxNorm: each score is divided for the maximum of each class; 3. Qnorm: quantile normalization. preprocessCore package is used;
W	vector of weight relative to a single example. If the vector <code>W</code> is not specified (def. <code>W=NULL</code>), <code>W</code> is a unitary vector of the same length of the columns' number of the flat scores matrix (root node included).
parallel	boolean value: <ul style="list-style-type: none"> • TRUE: execute the parallel implementation of GPAV (GPAV.parallel); • FALSE (def.): execute the sequential implementation of GPAV (GPAV.over.examples);
ncores	number of cores to use for parallel execution (def. 8). Set the parameter <code>ncores</code> to 1 if the parameter <code>parallel</code> is set to FALSE, otherwise set the desired number of cores.
folds	number of folds of the cross validation on which computing the performance metrics averaged across folds (def. 5). If <code>folds=NULL</code> , the performance metrics are computed one-shot, otherwise the performance metrics are averaged across folds. If <code>compute.performance</code> is set to FALSE, <code>folds</code> is automatically set to NULL.
seed	initialization seed for the random generator to create folds (def. 23). If NULL folds are generated without seed initialization. The parameter <code>seed</code> controls both the parameter <code>kk</code> and the parameter <code>folds</code> . If <code>compute.performance</code> is set to FALSE and <code>bottomup</code> is set to <code>threshold.free</code> , then <code>seed</code> is automatically set to NULL.
n.round	number of rounding digits to be applied to the hierarchical scores matrix (def. 3). It is used for choosing the best threshold on the basis of the best F-measure. If <code>compute.performance</code> is set to FALSE and <code>bottomup</code> is set to <code>threshold.free</code> , then <code>n.round</code> is automatically set to NULL.

<code>f.criterion</code>	<p>character. Type of F-measure to be used to select the best F-measure. Two possibilities:</p> <ol style="list-style-type: none"> 1. F (def.): corresponds to the harmonic mean between the average precision and recall; 2. avF: corresponds to the per-example F-score averaged across all the examples; <p>If <code>compute.performance</code> is set to FALSE and <code>bottomup</code> is set to <code>threshold.free</code>, then <code>f.criterion</code> is automatically set to NULL.</p>
<code>recall.levels</code>	a vector with the desired recall levels (def: from:0.1, to:0.9, by:0.1) to compute the Precision at fixed Recall level (PXR). If <code>compute.performance=FALSE</code> the parameter <code>recall.levels</code> is automatically set to NULL.
<code>compute.performance</code>	<p>boolean value: should the flat and hierarchical performance (AUPRC, AUROC, PXR, multilabel F-score) be returned?</p> <ul style="list-style-type: none"> • FALSE (def.): performance are not computed and just the hierarchical scores matrix is returned; • TRUE: both performance and hierarchical scores matrix are returned;
<code>flat.file</code>	name of the file containing the flat scores matrix to be normalized or already normalized (without rda extension).
<code>ann.file</code>	name of the file containing the label matrix of the examples (without rda extension).
<code>dag.file</code>	name of the file containing the graph that represents the hierarchy of the classes (without rda extension).
<code>flat.dir</code>	relative path where flat scores matrix is stored.
<code>ann.dir</code>	relative path where annotation matrix is stored.
<code>dag.dir</code>	relative path where graph is stored.
<code>hierScore.dir</code>	relative path where the hierarchical scores matrix must be stored.
<code>perf.dir</code>	relative path where the performance measures must be stored. If <code>compute.performance=FALSE</code> the functions automatically sets <code>perf.dir</code> to NULL.

Details

The function checks if the number of classes between the flat scores matrix and the annotations matrix mismatched. If so, the number of terms of the annotations matrix is shrunk to the number of terms of the flat scores matrix and the corresponding subgraph is computed as well. N.B.: it is supposed that all the nodes of the subgraph are accessible from the root.

Value

Two rda files stored in the respective output directories:

1. Hierarchical Scores Results: a matrix with examples on rows and classes on columns representing the computed hierarchical scores for each example and for each considered class. It is stored in the `hierScore.dir` directory.
2. Performance Measures: *flat* and *hierarchical* performance results:

- (a) AUPRC results computed through `AUPRC.single.over.classes` ([AUPRC](#));
- (b) AUROC results computed through `AUROC.single.over.classes` ([AUROC](#));
- (c) PXR results computed through `precision.at.given.recall.levels.over.classes` ([PXR](#));
- (d) FMM results computed through `compute.Fmeasure.multilabel` ([FMM](#));

It is stored in the `perf.dir` directory.

See Also

[GPAV](#)

Examples

```
data(graph);
data(scores);
data(labels);
tmpdir <- paste0(tmpdir(), "/");
save(g, file=paste0(tmpdir, "graph.rda"));
save(L, file=paste0(tmpdir, "labels.rda"));
save(S, file=paste0(tmpdir, "scores.rda"));
dag.dir <- flat.dir <- ann.dir <- tmpdir;
hierScore.dir <- perf.dir <- tmpdir;
recall.levels <- seq(from=0.25, to=1, by=0.25);
dag.file <- "graph";
flat.file <- "scores";
ann.file <- "labels";
Do.GPAV(norm=FALSE, norm.type= "MaxNorm", W=NULL, parallel=FALSE, ncores=1, folds=NULL,
seed=23, n.round=3, f.criterion="F", recall.levels=recall.levels, compute.performance=TRUE,
flat.file=flat.file, ann.file=ann.file, dag.file=dag.file, flat.dir=flat.dir, ann.dir=ann.dir,
dag.dir=dag.dir, hierScore.dir=hierScore.dir, perf.dir=perf.dir);
```

Do.GPAV.holdout

GPAV holdout

Description

High level function to correct the computed scores in a hierarchy according to the GPAV algorithm applying a classical holdout procedure.

Usage

```
Do.GPAV.holdout(norm = TRUE, norm.type = NULL, W = NULL,
parallel = FALSE, ncores = 1, folds = 5, seed = 23,
n.round = 3, f.criterion = "F", recall.levels = seq(from = 0.1, to
= 1, by = 0.1), compute.performance = FALSE, flat.file = flat.file,
ann.file = ann.file, dag.file = dag.file,
ind.test.set = ind.test.set, ind.dir = ind.dir,
flat.dir = flat.dir, ann.dir = ann.dir, dag.dir = dag.dir,
hierScore.dir = hierScore.dir, perf.dir = perf.dir)
```

Arguments

norm	boolean value: <ul style="list-style-type: none"> • TRUE (def.): the flat scores matrix has been already normalized in according to a normalization method; • FALSE: the flat scores matrix has not been normalized yet. See the parameter norm for which normalization can be applied;
norm.type	can be one of the following three values: <ol style="list-style-type: none"> 1. NULL (def.): set norm.type to NULL if and only if the parameter norm is set to TRUE; 2. MaxNorm: each score is divided for the maximum of each class; 3. Qnorm: quantile normalization. preprocessCore package is used;
W	vector of weight relative to a single example. If the vector W is not specified (def. W=NULL), W is a unitary vector of the same length of the columns' number of the flat scores matrix (root node included).
parallel	boolean value: <ul style="list-style-type: none"> • TRUE: execute the parallel implementation of GPAV (GPAV.parallel); • FALSE (def.): execute the sequential implementation of GPAV (GPAV.over.examples);
ncores	number of cores to use for parallel execution (def. 8). Set the parameter ncores to 1 if the parameter parallel is set to FALSE, otherwise set the desired number of cores.
folds	number of folds of the cross validation on which computing the performance metrics averaged across folds (def. 5). If folds=NULL, the performance metrics are computed one-shot, otherwise the performance metrics are averaged across folds. If compute.performance is set to FALSE, folds is automatically set to NULL.
seed	initialization seed for the random generator to create folds (def. 23). If NULL folds are generated without seed initialization. The parameter seed controls both the parameter kk and the parameter folds. If compute.performance is set to FALSE and bottomup is set to threshold.free, then seed is automatically set to NULL.
n.round	number of rounding digits to be applied to the hierarchical scores matrix (def. 3). It is used for choosing the best threshold on the basis of the best F-measure. If compute.performance is set to FALSE and bottomup is set to threshold.free, then n.round is automatically set to NULL.
f.criterion	character. Type of F-measure to be used to select the best F-measure. Two possibilities: <ol style="list-style-type: none"> 1. F (def.): corresponds to the harmonic mean between the average precision and recall; 2. avF: corresponds to the per-example F-score averaged across all the examples; <p>If compute.performance is set to FALSE and bottomup is set to threshold.free, then f.criterion is automatically set to NULL.</p>

<code>recall.levels</code>	a vector with the desired recall levels (def: from:0.1, to:0.9, by:0.1) to compute the Precision at fixed Recall level (PXR). If <code>compute.performance=FALSE</code> the parameter <code>recall.levels</code> is automatically set to NULL.
<code>compute.performance</code>	boolean value: should the flat and hierarchical performance (AUPRC, AUROC, PXR, multilabel F-score) be returned? <ul style="list-style-type: none"> • FALSE (def.): performance are not computed and just the hierarchical scores matrix is returned; • TRUE: both performance and hierarchical scores matrix are returned;
<code>flat.file</code>	name of the file containing the flat scores matrix to be normalized or already normalized (without rda extension).
<code>ann.file</code>	name of the file containing the label matrix of the examples (without rda extension).
<code>dag.file</code>	name of the file containing the graph that represents the hierarchy of the classes (without rda extension).
<code>ind.test.set</code>	name of the file containing a vector of integer numbers corresponding to the indices of the elements (rows) of scores matrix to be used in the test set.
<code>ind.dir</code>	relative path to folder where <code>ind.test.set</code> is stored.
<code>flat.dir</code>	relative path where flat scores matrix is stored.
<code>ann.dir</code>	relative path where annotation matrix is stored.
<code>dag.dir</code>	relative path where graph is stored.
<code>hierScore.dir</code>	relative path where the hierarchical scores matrix must be stored.
<code>perf.dir</code>	relative path where the performance measures must be stored. If <code>compute.performance=FALSE</code> , the parameter <code>perf.dir</code> is automatically set to NULL.

Details

The function checks if the number of classes between the flat scores matrix and the annotations matrix mismatched. If so, the number of terms of the annotations matrix is shrunk to the number of terms of the flat scores matrix and the corresponding subgraph is computed as well. N.B.: it is supposed that all the nodes of the subgraph are accessible from the root.

Value

Two rda files stored in the respective output directories:

1. Hierarchical Scores Results: a matrix with examples on rows and classes on columns representing the computed hierarchical scores for each example and for each considered class. It is stored in the `hierScore.dir` directory;
2. Performance Measures: *flat* and *hierarchical* performance results:
 - (a) AUPRC results computed through `AUPRC.single.over.classes` (AUPRC);
 - (b) AUROC results computed through `AUROC.single.over.classes` (AUROC);
 - (c) PXR results computed through `precision.at.given.recall.levels.over.classes` (PXR);
 - (d) FMM results computed through `compute.Fmeasure.multilabel` (FMM);

It is stored in the `perf.dir` directory.

See Also[GPAV](#)**Examples**

```

data(graph);
data(scores);
data(labels);
data(test.index);
tmpdir <- paste0(tmpdir(),"/");
save(g, file=paste0(tmpdir,"graph.rda"));
save(L, file=paste0(tmpdir,"labels.rda"));
save(S, file=paste0(tmpdir,"scores.rda"));
save(test.index, file=paste0(tmpdir,"test.index.rda"));
ind.dir <- dag.dir <- flat.dir <- ann.dir <- tmpdir;
hierScore.dir <- perf.dir <- tmpdir;
ind.test.set <- "test.index";
recall.levels <- seq(from=0.25, to=1, by=0.25);
dag.file <- "graph";
flat.file <- "scores";
ann.file <- "labels";
Do.GPAV.holdout(norm=FALSE, norm.type="MaxNorm", W=NULL, parallel=FALSE, ncores=1,
n.round=3, f.criterion="F", folds=NULL, seed=23, recall.levels=recall.levels,
compute.performance=TRUE, flat.file=flat.file, ann.file=ann.file, dag.file=dag.file,
ind.test.set=ind.test.set, ind.dir=ind.dir, flat.dir=flat.dir, ann.dir=ann.dir,
dag.dir=dag.dir, hierScore.dir=hierScore.dir, perf.dir=perf.dir);

```

Do.heuristic.methods *Do Heuristic Methods*

Description

High level function to compute the hierarchical heuristic methods MAX, AND, OR (Heuristic Methods MAX, AND, OR (*Obozinski et al., Genome Biology, 2008*)).

Usage

```

Do.heuristic.methods(heuristic.fun = "AND", norm = TRUE,
norm.type = NULL, folds = 5, seed = 23, n.round = 3,
f.criterion = "F", recall.levels = seq(from = 0.1, to = 1, by = 0.1),
compute.performance = FALSE, flat.file = flat.file,
ann.file = ann.file, dag.file = dag.file, flat.dir = flat.dir,
ann.dir = ann.dir, dag.dir = dag.dir,
hierScore.dir = hierScore.dir, perf.dir = perf.dir)

```


Arguments

heuristic.fun	<p>can be one of the following three values:</p> <ol style="list-style-type: none"> 1. "MAX": run the heuristic method MAX; 2. "AND": run the heuristic method AND; 3. "OR": run the heuristic method OR;
norm	<p>boolean value:</p> <ul style="list-style-type: none"> • TRUE (def.): the flat scores matrix has been already normalized in according to a normalization method; • FALSE: the flat scores matrix has not been normalized yet. See the parameter norm.type for which normalization can be applied;
norm.type	<p>can be one of the following three values:</p> <ol style="list-style-type: none"> 1. NULL (def.): set norm.type to NULL if and only if the parameter norm is set to TRUE; 2. MaxNorm: each score is divided for the maximum of each class; 3. Qnorm: quantile normalization. preprocessCore package is used;
folds	<p>number of folds of the cross validation on which computing the performance metrics averaged across folds (def. 5). If folds=NULL, the performance metrics are computed one-shot, otherwise the performance metrics are averaged across folds. If compute.performance is set to FALSE, folds is automatically set to NULL.</p>
seed	<p>initialization seed for the random generator to create folds (def. 23). If NULL folds are generated without seed initialization. The parameter seed controls both the parameter kk and the parameter folds. If compute.performance is set to FALSE and bottomup is set to threshold.free, then seed is automatically set to NULL.</p>
n.round	<p>number of rounding digits to be applied to the hierarchical scores matrix (def. 3). It is used for choosing the best threshold on the basis of the best F-measure. If compute.performance is set to FALSE and bottomup is set to threshold.free, then n.round is automatically set to NULL.</p>
f.criterion	<p>character. Type of F-measure to be used to select the best F-measure. Two possibilities:</p> <ol style="list-style-type: none"> 1. F (def.): corresponds to the harmonic mean between the average precision and recall; 2. avF: corresponds to the per-example F-score averaged across all the examples; <p>If compute.performance is set to FALSE and bottomup is set to threshold.free, then f.criterion is automatically set to NULL.</p>
recall.levels	<p>a vector with the desired recall levels (def: from:0.1, to:0.9, by:0.1) to compute the Precision at fixed Recall level (PXR). If compute.performance=FALSE the parameter recall.levels is automatically set to NULL.</p>
compute.performance	<p>boolean value: should the flat and hierarchical performance (AUPRC, AUROC, PXR, multilabel F-score) be returned?</p>

	<ul style="list-style-type: none"> • FALSE (def.): performance are not computed and just the hierarchical scores matrix is returned; • TRUE: both performance and hierarchical scores matrix are returned;
flat.file	name of the file containing the flat scores matrix to be normalized or already normalized (without rda extension).
ann.file	name of the file containing the label matrix of the examples (without rda extension).
dag.file	name of the file containing the graph that represents the hierarchy of the classes (without rda extension).
flat.dir	relative path where flat scores matrix is stored.
ann.dir	relative path where annotation matrix is stored.
dag.dir	relative path where graph is stored.
hierScore.dir	relative path where the hierarchical scores matrix must be stored.
perf.dir	relative path where the performance measures must be stored. If compute.performance=FALSE the functions automatically sets perf.dir to NULL.

Details

The function checks if the number of classes between the flat scores matrix and the annotations matrix mismatched. If so, the number of terms of the annotations matrix is shrunk to the number of terms of the flat scores matrix and the corresponding subgraph is computed as well. N.B.: it is supposed that all the nodes of the subgraph are accessible from the root.

We excluded the predictions of the root node in computing all the performances, since it is a *dummy* node added to the ontology for practical reasons (e.g. some graph-based software may require a single root node to work). However, the root node scores are stored in the hierarchical scores matrix.

Value

Two rda files stored in the respective output directories:

1. Hierarchical Scores Results: a matrix with examples on rows and classes on columns representing the computed hierarchical scores for each example and for each considered class. It is stored in the hierScore.dir directory;
2. Performance Measures: *flat* and *hierarchical* performance results:
 - (a) AUPRC results computed through AUPRC.single.over.classes ([AUPRC](#));
 - (b) AUROC results computed through AUROC.single.over.classes ([AUROC](#));
 - (c) PXR results computed through precision.at.given.recall.levels.over.classes ([PXR](#));
 - (d) FMM results computed through compute.Fmeasure.multilabel ([FMM](#));

It is stored in the perf.dir directory.

See Also

[Heuristic-Methods](#)

Examples

```
data(graph);
data(scores);
data(labels);
tmpdir <- paste0(tempdir(),"/");
save(g, file=paste0(tmpdir,"graph.rda"));
save(L, file=paste0(tmpdir,"labels.rda"));
save(S, file=paste0(tmpdir,"scores.rda"));
dag.dir <- flat.dir <- ann.dir <- tmpdir;
hierScore.dir <- perf.dir <- tmpdir;
recall.levels <- seq(from=0.25, to=1, by=0.25);
dag.file <- "graph";
flat.file <- "scores";
ann.file <- "labels";
Do.heuristic.methods(heuristic.fun="AND", norm=FALSE, norm.type="MaxNorm",
folds=NULL, seed=23, n.round=3, f.criterion = "F", recall.levels=recall.levels,
compute.performance=TRUE, flat.file=flat.file, ann.file=ann.file,
dag.file=dag.file, flat.dir=flat.dir, ann.dir=ann.dir, dag.dir=dag.dir,
hierScore.dir=hierScore.dir, perf.dir=perf.dir);
```

Do.heuristic.methods.holdout

Do Heuristic Methods holdout

Description

High level function to compute the hierarchical heuristic methods MAX, AND, OR (Heuristic Methods MAX, AND, OR (Obozinski et al., *Genome Biology*, 2008) applying a classical hold-out procedure.

Usage

```
Do.heuristic.methods.holdout(heuristic.fun = "AND", norm = TRUE,
norm.type = NULL, folds = 5, seed = 23, n.round = 3,
f.criterion = "F", recall.levels = seq(from = 0.1, to = 1, by = 0.1),
compute.performance = FALSE, flat.file = flat.file,
ann.file = ann.file, dag.file = dag.file,
ind.test.set = ind.test.set, ind.dir = ind.dir,
flat.dir = flat.dir, ann.dir = ann.dir, dag.dir = dag.dir,
hierScore.dir = hierScore.dir, perf.dir = perf.dir)
```

Arguments

heuristic.fun can be one of the following three values:

1. "MAX": run the heuristic method MAX;
2. "AND": run the heuristic method AND;
3. "OR": run the heuristic method OR;

norm	<p>boolean value:</p> <ul style="list-style-type: none"> • TRUE (def.): the flat scores matrix has been already normalized in according to a normalization method; • FALSE: the flat scores matrix has not been normalized yet. See the parameter norm for which normalization can be applied;
norm.type	<p>can be one of the following three values:</p> <ol style="list-style-type: none"> 1. NONE (def.): set norm.type to NONE if and only if the parameter norm is set to TRUE; 2. MaxNorm: each score is divided for the maximum of each class; 3. Qnorm: quantile normalization. preprocessCore package is used;
folds	<p>number of folds of the cross validation on which computing the performance metrics averaged across folds (def. 5). If folds=NULL, the performance metrics are computed one-shot, otherwise the performance metrics are averaged across folds. If compute.performance is set to FALSE, folds is automatically set to NULL.</p>
seed	<p>initialization seed for the random generator to create folds (def. 23). If NULL folds are generated without seed initialization. The parameter seed controls both the parameter kk and the parameter folds. If compute.performance is set to FALSE and bottomup is set to threshold.free, then seed is automatically set to NULL.</p>
n.round	<p>number of rounding digits to be applied to the hierarchical scores matrix (def. 3). It is used for choosing the best threshold on the basis of the best F-measure. If compute.performance is set to FALSE and bottomup is set to threshold.free, then n.round is automatically set to NULL.</p>
f.criterion	<p>character. Type of F-measure to be used to select the best F-measure. Two possibilities:</p> <ol style="list-style-type: none"> 1. F (def.): corresponds to the harmonic mean between the average precision and recall; 2. avF: corresponds to the per-example F-score averaged across all the examples; <p>If compute.performance is set to FALSE and bottomup is set to threshold.free, then f.criterion is automatically set to NULL.</p>
recall.levels	<p>a vector with the desired recall levels (def: from:0.1, to:0.9, by:0.1) to compute the Precision at fixed Recall level (PXR). If compute.performance=FALSE the parameter recall.levels is automatically set to NULL</p>
compute.performance	<p>boolean value: should the flat and hierarchical performance (AUPRC, AUROC, PXR, multilabel F-score) be returned?</p> <ul style="list-style-type: none"> • FALSE (def.): performance are not computed and just the hierarchical scores matrix is returned; • TRUE: both performance and hierarchical scores matrix are returned;
flat.file	<p>name of the file containing the flat scores matrix to be normalized or already normalized (without rda extension).</p>

<code>ann.file</code>	name of the file containing the label matrix of the examples (without rda extension).
<code>dag.file</code>	name of the file containing the graph that represents the hierarchy of the classes (without rda extension).
<code>ind.test.set</code>	name of the file containing a vector of integer numbers corresponding to the indices of the elements (rows) of scores matrix to be used in the test set.
<code>ind.dir</code>	relative path to folder where <code>ind.test.set</code> is stored.
<code>flat.dir</code>	relative path where flat scores matrix is stored.
<code>ann.dir</code>	relative path where annotation matrix is stored.
<code>dag.dir</code>	relative path where graph is stored.
<code>hierScore.dir</code>	relative path where the hierarchical scores matrix must be stored.
<code>perf.dir</code>	relative path where the performance measures must be stored. If <code>compute.performance=FALSE</code> , the parameter <code>perf.dir</code> is automatically set to <code>NULL</code> .

Details

The function checks if the number of classes between the flat scores matrix and the annotations matrix mismatched. If so, the number of terms of the annotations matrix is shrunk to the number of terms of the flat scores matrix and the corresponding subgraph is computed as well. N.B.: it is supposed that all the nodes of the subgraph are accessible from the root.

We excluded the predictions of the root node in computing all the performances, since it is a *dummy* node added to the ontology for practical reasons (e.g. some graph-based software may require a single root node to work). However, the root node scores are stored in the hierarchical scores matrix.

Value

Two rda files stored in the respective output directories:

1. Hierarchical Scores Results: a matrix with examples on rows and classes on columns representing the computed hierarchical scores for each example and for each considered class. It is stored in the `hierScore.dir` directory;
2. Performance Measures: *flat* and *hierarchical* performance results:
 - (a) AUPRC results computed through `AUPRC.single.over.classes` ([AUPRC](#));
 - (b) AUROC results computed through `AUROC.single.over.classes` ([AUROC](#));
 - (c) PXR results computed through `precision.at.given.recall.levels.over.classes` ([PXR](#));
 - (d) FMM results computed through `compute.Fmeasure.multilabel` ([FMM](#));

It is stored in the `perf.dir` directory.

Examples

```
data(graph);
data(scores);
data(labels);
```

```

data(test.index);
tmpdir <- paste0(tmpdir(),"/");
save(g, file=paste0(tmpdir,"graph.rda"));
save(L, file=paste0(tmpdir,"labels.rda"));
save(S, file=paste0(tmpdir,"scores.rda"));
save(test.index, file=paste0(tmpdir,"test.index.rda"));
ind.dir <- dag.dir <- flat.dir <- ann.dir <- tmpdir;
hierScore.dir <- perf.dir <- tmpdir;
recall.levels <- seq(from=0.25, to=1, by=0.25);
ind.test.set <- "test.index";
dag.file <- "graph";
flat.file <- "scores";
ann.file <- "labels";
Do.heuristic.methods.holdout(heuristic.fun="MAX", norm=FALSE, norm.type="MaxNorm",
folds=NULL, seed=23, n.round=3, f.criterion ="F", recall.levels=recall.levels,
compute.performance=TRUE, flat.file=flat.file, ann.file=ann.file, dag.file=dag.file,
ind.test.set=ind.test.set, ind.dir=ind.dir, flat.dir=flat.dir, ann.dir=ann.dir,
dag.dir=dag.dir, hierScore.dir=hierScore.dir, perf.dir=perf.dir);

```

Do.HTD

HTD-DAG vanilla

Description

High level function to correct the computed scores in a hierarchy according to the HTD-DAG algorithm.

Usage

```

Do.HTD(norm = TRUE, norm.type = NULL, folds = 5, seed = 23,
n.round = 3, f.criterion = "F", recall.levels = seq(from = 0.1, to
= 1, by = 0.1), compute.performance = FALSE, flat.file = flat.file,
ann.file = ann.file, dag.file = dag.file, flat.dir = flat.dir,
ann.dir = ann.dir, dag.dir = dag.dir,
hierScore.dir = hierScore.dir, perf.dir = perf.dir)

```

Arguments

- | | |
|-----------|--|
| norm | boolean value: <ul style="list-style-type: none"> • TRUE (def.): the flat scores matrix has been already normalized in according to a normalization method; • FALSE: the flat scores matrix has not been normalized yet. See the parameter norm.type for which normalization can be applied; |
| norm.type | can be one of the following three values: <ol style="list-style-type: none"> 1. NULL (def.): set norm.type to NULL if and only if the parameter norm is set to TRUE; 2. MaxNorm: each score is divided for the maximum of each class; |

	3. Qnorm: quantile normalization. preprocessCore package is used;
fold	number of folds of the cross validation on which computing the performance metrics averaged across folds (def. 5). If folds=NULL, the performance metrics are computed one-shot, otherwise the performance metrics are averaged across folds. If compute.performance is set to FALSE, folds is automatically set to NULL.
seed	initialization seed for the random generator to create folds (def. 23). If NULL folds are generated without seed initialization. The parameter seed controls both the parameter kk and the parameter folds. If compute.performance is set to FALSE and bottomup is set to threshold.free, then seed is automatically set to NULL.
n.round	number of rounding digits to be applied to the hierarchical scores matrix (def. 3). It is used for choosing the best threshold on the basis of the best F-measure. If compute.performance is set to FALSE and bottomup is set to threshold.free, then n.round is automatically set to NULL.
f.criterion	character. Type of F-measure to be used to select the best F-measure. Two possibilities: <ol style="list-style-type: none"> 1. F (def.): corresponds to the harmonic mean between the average precision and recall; 2. avF: corresponds to the per-example F-score averaged across all the examples; If compute.performance is set to FALSE and bottomup is set to threshold.free, then f.criterion is automatically set to NULL.
recall.levels	a vector with the desired recall levels (def: from:0.1, to:0.9, by:0.1) to compute the Precision at fixed Recall level (PXR). If compute.performance=FALSE then recall.levels is automatically set to NULL.
compute.performance	boolean value: should the flat and hierarchical performance (AUPRC, AUROC, PXR, multilabel F-score) be returned? <ul style="list-style-type: none"> • FALSE: performance are not computed and just the hierarchical scores matrix is returned; • TRUE (def.): both performance and hierarchical scores matrix are returned;
flat.file	name of the file containing the flat scores matrix to be normalized or already normalized (without rda extension).
ann.file	name of the file containing the label matrix of the examples (without rda extension).
dag.file	name of the file containing the graph that represents the hierarchy of the classes (without rda extension).
flat.dir	relative path where flat scores matrix is stored.
ann.dir	relative path where annotation matrix is stored.
dag.dir	relative path where graph is stored.
hierScore.dir	relative path where the hierarchical scores matrix must be stored.
perf.dir	relative path where the performance measures must be stored. If compute.performance=FALSE, perf.dir is automatically set to NULL.

Details

The function checks if the number of classes between the flat scores matrix and the annotations matrix mismatched. If so, the number of terms of the annotations matrix is shrunk to the number of terms of the flat scores matrix and the corresponding subgraph is computed as well. N.B.: it is supposed that all the nodes of the subgraph are accessible from the root.

We excluded the predictions of the root node in computing all the performances, since it is a *dummy* node added to the ontology for practical reasons (e.g. some graph-based software may require a single root node to work). However, the root node scores are stored in the hierarchical scores matrix.

Value

Two rda files stored in the respective output directories:

1. Hierarchical Scores Results: a matrix with examples on rows and classes on columns representing the computed hierarchical scores for each example and for each considered class. It is stored in the `hierScore.dir` directory;
2. Performance Measures: *flat* and *hierarchical* performance results:
 - (a) AUPRC results computed through `AUPRC.single.over.classes` ([AUPRC](#));
 - (b) AUROC results computed through `AUROC.single.over.classes` ([AUROC](#));
 - (c) PXR results computed through `precision.at.given.recall.levels.over.classes` ([PXR](#));
 - (d) FMM results computed through `compute.Fmeasure.multilabel` ([FMM](#));

It is stored in the `perf.dir` directory.

See Also

[HTD-DAG](#)

Examples

```
data(graph);
data(scores);
data(labels);
tmpdir <- paste0(tmpdir(),"/");
save(g, file=paste0(tmpdir,"graph.rda"));
save(L, file=paste0(tmpdir,"labels.rda"));
save(S, file=paste0(tmpdir,"scores.rda"));
dag.dir <- flat.dir <- ann.dir <- tmpdir;
hierScore.dir <- perf.dir <- tmpdir;
recall.levels <- seq(from=0.2, to=1, by=0.4);
dag.file <- "graph";
flat.file <- "scores";
ann.file <- "labels";
Do.HTD(norm=FALSE, norm.type="MaxNorm", folds=NULL, seed=23, n.round=3, f.criterion="F",
recall.levels=recall.levels, compute.performance=TRUE, flat.file=flat.file, ann.file=ann.file,
dag.file=dag.file, flat.dir=flat.dir, ann.dir=ann.dir, dag.dir=dag.dir,
hierScore.dir=hierScore.dir, perf.dir=perf.dir);
```


Do.HTD.holdout

*HTD-DAG holdout***Description**

High level function to correct the computed scores in a hierarchy according to the HTD-DAG algorithm applying a classical holdout procedure.

Usage

```
Do.HTD.holdout(norm = TRUE, norm.type = NULL, folds = 5, seed = 23,
  n.round = 3, f.criterion = "F", recall.levels = seq(from = 0.1, to
    = 1, by = 0.1), compute.performance = FALSE, flat.file = flat.file,
  ann.file = ann.file, dag.file = dag.file,
  ind.test.set = ind.test.set, ind.dir = ind.dir,
  flat.dir = flat.dir, ann.dir = ann.dir, dag.dir = dag.dir,
  hierScore.dir = hierScore.dir, perf.dir = perf.dir)
```

Arguments

norm	boolean value: <ul style="list-style-type: none"> • TRUE (def.): the flat scores matrix has been already normalized in according to a normalization method; • FALSE: the flat scores matrix has not been normalized yet. See the parameter norm for which normalization can be applied;
norm.type	can be one of the following three values: <ol style="list-style-type: none"> 1. NULL (def.): set norm.type to NULL if and only if the parameter norm is set to TRUE; 2. MaxNorm: each score is divided for the maximum of each class; 3. Qnorm: quantile normalization. preprocessCore package is used;
folds	number of folds of the cross validation on which computing the performance metrics averaged across folds (def. 5). If folds=NULL, the performance metrics are computed one-shot, otherwise the performance metrics are averaged across folds. If compute.performance is set to FALSE, folds is automatically set to NULL.
seed	initialization seed for the random generator to create folds (def. 23). If NULL folds are generated without seed initialization. The parameter seed controls both the parameter kk and the parameter folds. If compute.performance is set to FALSE and bottomup is set to threshold.free, then seed is automatically set to NULL.
n.round	number of rounding digits to be applied to the hierarchical scores matrix (def. 3). It is used for choosing the best threshold on the basis of the best F-measure. If compute.performance is set to FALSE and bottomup is set to threshold.free, then n.round is automatically set to NULL.

<code>f.criterion</code>	<p>character. Type of F-measure to be used to select the best F-measure. Two possibilities:</p> <ol style="list-style-type: none"> 1. <code>F (def.)</code>: corresponds to the harmonic mean between the average precision and recall; 2. <code>avF</code>: corresponds to the per-example F-score averaged across all the examples; <p>If <code>compute.performance</code> is set to <code>FALSE</code> and <code>bottomup</code> is set to <code>threshold.free</code>, then <code>f.criterion</code> is automatically set to <code>NULL</code>.</p>
<code>recall.levels</code>	a vector with the desired recall levels (def: from:0.1, to:0.9, by:0.1) to compute the Precision at fixed Recall level (PXR). If <code>compute.performance=FALSE</code> then <code>recall.levels</code> is automatically set to <code>NULL</code> .
<code>compute.performance</code>	<p>boolean value: should the flat and hierarchical performance (AUPRC, AUROC, PXR, multilabel F-score) be returned?</p> <ul style="list-style-type: none"> • <code>FALSE</code>: performance are not computed and just the hierarchical scores matrix is returned; • <code>TRUE (def.)</code>: both performance and hierarchical scores matrix are returned;
<code>flat.file</code>	name of the file containing the flat scores matrix to be normalized or already normalized (without rda extension).
<code>ann.file</code>	name of the file containing the label matrix of the examples (without rda extension).
<code>dag.file</code>	name of the file containing the graph that represents the hierarchy of the classes (without rda extension).
<code>ind.test.set</code>	name of the file containing a vector of integer numbers corresponding to the indices of the elements (rows) of scores matrix to be used in the test set .
<code>ind.dir</code>	relative path to folder where <code>ind.test.set</code> is stored.
<code>flat.dir</code>	relative path where flat scores matrix is stored.
<code>ann.dir</code>	relative path where annotation matrix is stored.
<code>dag.dir</code>	relative path where graph is stored.
<code>hierScore.dir</code>	relative path where the hierarchical scores matrix must be stored.
<code>perf.dir</code>	relative path where the performance measures must be stored. If <code>compute.performance=FALSE</code> , <code>perf.dir</code> is automatically set to <code>NULL</code> .

Details

The function checks if the number of classes between the flat scores matrix and the annotations matrix mismatched. If so, the number of terms of the annotations matrix is shrunk to the number of terms of the flat scores matrix and the corresponding subgraph is computed as well. N.B.: it is supposed that all the nodes of the subgraph are accessible from the root.

We excluded the predictions of the root node in computing all the performances, since it is a *dummy* node added to the ontology for practical reasons (e.g. some graph-based software may require a single root node to work). However, the root node scores are stored in the hierarchical scores matrix.

Value

Two rda files stored in the respective output directories:

1. Hierarchical Scores Results: a matrix with examples on rows and classes on columns representing the computed hierarchical scores for each example and for each considered class. It is stored in the `hierScore.dir` directory;
2. Performance Measures: *flat* and *hierarchical* performace results:
 - (a) AUPRC results computed though `AUPRC.single.over.classes` ([AUPRC](#));
 - (b) AUROC results computed through `AUROC.single.over.classes` ([AUROC](#));
 - (c) PXR results computed though `precision.at.given.recall.levels.over.classes` ([PXR](#));
 - (d) FMM results computed though `compute.Fmeasure.multilabel` ([FMM](#));

It is stored in the `perf.dir` directory.

See Also

[HTD-DAG](#)

Examples

```
data(graph);
data(scores);
data(labels);
data(test.index);
tmpdir <- paste0(tmpdir(), "/");
save(g, file=paste0(tmpdir, "graph.rda"));
save(L, file=paste0(tmpdir, "labels.rda"));
save(S, file=paste0(tmpdir, "scores.rda"));
save(test.index, file=paste0(tmpdir, "test.index.rda"));
ind.dir <- dag.dir <- flat.dir <- ann.dir <- tmpdir;
hierScore.dir <- perf.dir <- tmpdir;
recall.levels <- seq(from=0.2, to=1, by=0.4);
ind.test.set <- "test.index";
dag.file <- "graph";
flat.file <- "scores";
ann.file <- "labels";
Do.HTD.holdout(norm=FALSE, norm.type="MaxNorm", n.round=3, f.criterion="F", folds=5, seed=23,
recall.levels=recall.levels, compute.performance=TRUE, flat.file=flat.file, ann.file=ann.file,
dag.file=dag.file, ind.test.set=ind.test.set, ind.dir=ind.dir, flat.dir=flat.dir, ann.dir=ann.dir,
dag.dir=dag.dir, hierScore.dir=hierScore.dir, perf.dir=perf.dir);
```

do.subgraph

Build subgraph

Description

This function returns a subgraph with only the supplied nodes and any edges between them.

Usage

```
do.subgraph(nd, g, edgemode = "directed")
```

Arguments

nd a vector with the nodes for which the subgraph must be built.
 g a graph of class graphNEL. It represents the hierarchy of the classes.
 edgemode can be "directed" or "undirected".

Value

a subgraph with only the supplied nodes.

Examples

```
data(graph);
anc <- build.ancestors(g);
nd <- anc[["HP:0001371"]];
subg <- do.subgraph(nd, g, edgemode="directed");
```

do.submatrix

Build submatrix

Description

Terms having less than n annotations are pruned. Terms having exactly n annotations are discarded as well.

Usage

```
do.submatrix(ann, n)
```

Arguments

ann the annotation matrix (0/1). Rows are examples and columns are classes.
 n integer number of annotations to be pruned.

Value

Matrix of annotations having only those terms with more than n annotations.

Examples

```
data(labels);
subm <- do.submatrix(L,5);
```

do.unstratified.cv.data

Unstratified Cross Validation

Description

This function splits a dataset in k -fold in an unstratified way, i.e. a fold does not contain an equal amount of positive and negative examples. This function is used to perform k -fold cross-validation experiments in a hierarchical correction contest where splitting dataset in a stratified way is not needed.

Usage

```
do.unstratified.cv.data(S, kk = 5, seed = NULL)
```

Arguments

<code>S</code>	matrix of the flat scores. It must be a named matrix, where rows are example (e.g. genes) and columns are classes/terms (e.g. GO terms).
<code>kk</code>	number of folds in which to split the dataset (def. $k=5$).
<code>seed</code>	seed for the random generator. If NULL (def.) no initialization is performed.

Value

a list with $k = kk$ components (folds). Each component of the list is a character vector contains the index of the examples, i.e. the index of the rows of the matrix `S`.

Examples

```
data(scores);
foldIndex <- do.unstratified.cv.data(S, kk=5, seed=23);
```

example.datasets

Small real example datasets

Description

Collection of real sub-datasets used in the examples of the **HEMDAG** package

Usage

```
data(graph)
data(labels)
data(scores)
data(wadj)
data(test.index)
```

Details

The DAG `g` contained in `graph` data is an object of class `graphNEL`. The graph `g` has 23 nodes and 30 edges and represents the "ancestors view" of the HPO term *Camptodactyly of finger* ("HP:0100490").

The matrix `L` contained in the `labels` data is a 100 X 23 matrix, whose rows correspond to genes (*Entrez GeneID*) and columns to HPO classes. $L[i, j] = 1$ means that the gene i belong to class j , $L[i, j] = 0$ means that the gene i does not belong to class j . The classes of the matrix `L` correspond to the nodes of the graph `g`.

The matrix `S` contained in the `scores` data is a named 100 X 23 flat scores matrix, representing the likelihood that a given gene belongs to a given class: higher the value higher the likelihood. The classes of the matrix `S` correspond to the nodes of the graph `g`.

The matrix `W` contained in the `wadj` data is a named 100 X 100 symmetric weighted adjacency matrix, whose rows and columns correspond to genes. The genes names (*Entrez GeneID*) of the adjacency matrix `W` correspond to the genes names of the flat scores matrix `S` and to genes names of the target multilabel matrix `L`.

The vector of integer numbers `test.index` contained in the `test.index` data refers to the index of the examples of the scores matrix `S` to be used in the test set. It is useful only in holdout experiments.

Note

Some examples of full data sets for the prediction of HPO terms are available at the following [link](#). Note that the processing of the full datasets should be done similarly to the processing of the small data examples provided directly in this package. Please read the README clicking the link above to know more details about the available full datasets.

<code>find.best.f</code>	<i>Best hierarchical F-score</i>
--------------------------	----------------------------------

Description

Function to select the best hierarchical F-score by choosing an appropriate threshold in the scores.

Usage

```
find.best.f(target, predicted, n.round = 3, f.criterion = "F",
  verbose = TRUE, b.per.example = FALSE)
```

Arguments

<code>target</code>	matrix with the target multilabel: rows correspond to examples and columns to classes. $target[i, j] = 1$ if example i belongs to class j , $target[i, j] = 0$ otherwise.
<code>predicted</code>	a numeric matrix with continuous predicted values (scores): rows correspond to examples and columns to classes.
<code>n.round</code>	number of rounding digits to be applied to predicted (default=3).

f.criterion	character. Type of F-measure to be used to select the best F-score. There are two possibilities: <ol style="list-style-type: none"> 1. F (def.) corresponds to the harmonic mean between the average precision and recall; 2. avF corresponds to the per-example F-score averaged across all the examples;
verbose	boolean. If TRUE (def.) the number of iterations are printed on stdout.
b.per.example	boolean. <ul style="list-style-type: none"> • TRUE: results are returned for each example; • FALSE: only the average results are returned;

Details

All the examples having no positive annotations are discarded. The predicted scores matrix (predicted) is rounded according to parameter `n.round` and all the values of predicted are divided by `max(predicted)`. Then all the thresholds corresponding to all the different values included in predicted are attempted, and the threshold leading to the maximum F-measure is selected.

Names of rows and columns of target and predicted matrix must be provided in the same order, otherwise a stop message is returned.

Value

Two different outputs respect to the input parameter `b.per.example`:

- `b.per.example==FALSE`: a list with a single element average. A named vector with 7 elements relative to the best result in terms of the F-measure: Precision (P), Recall (R), Specificity (S), F-measure (F), av.F-measure (av.F), Accuracy (A) and the best selected Threshold (T). F is the F-measure computed as the harmonic mean between the average precision and recall; av.F is the F-measure computed as the average across examples and T is the best selected threshold;
- `b.per.example==TRUE`: a list with two elements:
 1. average: a named vector with 7 elements relative to the best result in terms of the F-measure: Precision (P), Recall (R), Specificity (S), F-measure (F), av.F-measure (av.F), Accuracy (A) and the best selected Threshold (T);
 2. per.example: a named matrix with the Precision (P), Recall (R), Specificity (S), Accuracy (A), F-measure (F), av.F-measure (av.F) and the best selected Threshold (T) for each example. Row names correspond to examples, column names correspond respectively to Precision (P), Recall (R), Specificity (S), Accuracy (A), F-measure (F), av.F-measure (av.F) and the best selected Threshold (T);

Examples

```
data(graph);
data(labels);
data(scores);
root <- root.node(g);
L <- L[, -which(colnames(L)==root)];
S <- S[, -which(colnames(S)==root)];
FMM <- find.best.f(L, S, n.round=3, f.criterion="F", verbose=TRUE, b.per.example=TRUE);
```

find.leaves

Leaves

Description

Find the leaves of a directed graph.

Usage

```
find.leaves(g)
```

Arguments

`g` a graph of class `graphNEL`. It represents the hierarchy of the classes.

Value

a vector with the names of the leaves of `g`.

Examples

```
data(graph);
leaves <- find.leaves(g);
```

FMM

Compute F-measure Multilabel

Description

Function to compute the best hierarchical F-score either one-shot or averaged across folds

Usage

```
compute.Fmeasure.multilabel(target, predicted, n.round = 3,
  f.criterion = "F", verbose = TRUE, b.per.example = FALSE,
  folds = NULL, seed = NULL)
```

Arguments

`target` matrix with the target multilabel: rows correspond to examples and columns to classes. $target[i, j] = 1$ if example i belongs to class j , $target[i, j] = 0$ otherwise.

`predicted` a numeric matrix with predicted values (scores): rows correspond to examples and columns to classes.

`n.round` number of rounding digits to be applied to predicted (default=3).

<code>f.criterion</code>	character. Type of F-measure to be used to select the best F-score. There are two possibilities: <ol style="list-style-type: none"> 1. F (def.) corresponds to the harmonic mean between the average precision and recall; 2. avF corresponds to the per-example F-score averaged across all the examples;
<code>verbose</code>	boolean. If TRUE (def.) the number of iterations are printed on stdout.
<code>b.per.example</code>	boolean. <ul style="list-style-type: none"> • TRUE: results are returned for each example; • FALSE: only the average results are returned;
<code>folds</code>	number of folds on which computing the AUROC. If <code>folds=NULL</code> (def.), the AUROC is computed one-shot, otherwise the AUROC is computed averaged across folds.
<code>seed</code>	initialization seed for the random generator to create folds. Set seed only if <code>folds≠NULL</code> . If <code>seed=NULL</code> and <code>folds≠NULL</code> , the AUROC averaged across folds is computed without seed initialization.

Details

Names of rows and columns of target and predicted matrix must be provided in the same order, otherwise a stop message is returned.

Value

Two different outputs respect to the input parameter `b.per.example`:

- `b.per.example==FALSE`: a list with a single element average. A named vector with 7 elements relative to the best result in terms of the F-measure: Precision (P), Recall (R), Specificity (S), F-measure (F), av.F-measure (av.F), Accuracy (A) and the best selected Threshold (T). F is the F-measure computed as the harmonic mean between the average precision and recall; av.F is the F-measure computed as the average across examples and T is the best selected threshold;
- `b.per.example==TRUE`: a list with two elements:
 1. average: a named vector with 7 elements relative to the best result in terms of the F-measure: Precision (P), Recall (R), Specificity (S), F-measure (F), av.F-measure (av.F), Accuracy (A) and the best selected Threshold (T);
 2. per.example: a named matrix with the Precision (P), Recall (R), Specificity (S), Accuracy (A), F-measure (F), av.F-measure (av.F) and the best selected Threshold (T) for each example. Row names correspond to examples, column names correspond respectively to Precision (P), Recall (R), Specificity (S), Accuracy (A), F-measure (F), av.F-measure (av.F) and the best selected Threshold (T);

Examples

```
data(graph);
data(labels);
data(scores);
root <- root.node(g);
```

```
L <- L[, -which(colnames(L) == "root")];
S <- S[, -which(colnames(S) == "root")];
FMM <- compute.Fmeasure.multilabel(L, S, n.round=3, f.criterion="F", verbose=TRUE,
b.per.example=TRUE, folds=5, seed=23);
```

full.annotation.matrix

Full annotation matrix

Description

Construct a full annotations table using ancestors and the most specific annotations table w.r.t. a given weighted adjacency matrix (wadj). The rows of the full annotation matrix correspond to all the examples of the given weighted adjacency matrix and the columns to the class/terms. The transitive closure of the annotations is performed.

Usage

```
full.annotation.matrix(W, anc, ann.spec)
```

Arguments

W	symmetric adjacency weighted matrix of the graph.
anc	list of the ancestors of the ontology.
ann.spec	the annotation matrix of the most specific annotations (0/1): rows are genes and columns are classes.

Details

The examples present in the annotation matrix (ann.spec) but not in the adjacency weighted matrix (W) are purged.

Value

a full annotation table T, that is a matrix in which the transitive closure of annotations was performed. Rows correspond to genes of the weighted adjacency matrix and columns to terms. $T[i, j] = 1$ means that gene i is annotated for the term j , $T[i, j] = 0$ means that gene i is not annotated for the term j .

See Also

[weighted.adjacency.matrix](#), [build.ancestors](#),
[specific.annotation.matrix](#), [transitive.closure.annotations](#)

Examples

```
data(wadj);
data(graph);
data(labels);
anc <- build.ancestors(g);
full.ann <- full.annotation.matrix(W, anc, L);
```

GPAV

Generalized Pool-Adjacent Violators

Description

Implementation of GPAV (Generalized Pool-Adjacent Violators) algorithm. (Burdakov et al., In: Di Pillo G, Roma M, editors. *An $O(n^2)$ Algorithm for Isotonic Regression*. Boston, MA: Springer US; 2006. p. 25–33. Available from: https://doi.org/10.1007/0-387-30065-1_3)

Usage

```
GPAV(Y, W = NULL, adj)
```

Arguments

- | | |
|-----|---|
| Y | vector of scores relative to a single example. Y must be a numeric named vector, where names correspond to classes' names, i.e. nodes of the graph g (root node included). |
| W | vector of weight relative to a single example. If the vector W is not specified (def. W=NULL), it is assumed that W is a unitary vector of the same length of the vector Y. |
| adj | adjacency matrix of the graph which must be sparse, logical and upper triangular. Number of columns of adj must be equal to the length of Y and W. |

Details

Given the constraints adjacency matrix of the graph, a vector of scores $\hat{y} \in R^n$ and a vector of strictly positive weights $w \in R^n$, the GPAV algorithm returns a vector \bar{y} which is as close as possible, in the least-squares sense, to the response vector \hat{y} and whose components are partially ordered in accordance with the constraints matrix adj. In other words, GPAV solves the following problem:

$$\bar{y} = \begin{cases} \min \sum_{i \in N} (\hat{y}_i - \bar{y}_i)^2 \\ \forall i, \quad j \in par(i) \Rightarrow \bar{y}_j \geq \bar{y}_i \end{cases}$$

Value

a list of 3 elements:

- `YFit`: a named vector with the scores of the classes corrected according to the GPAV algorithm. NOTE: the classes of `YFit` are topologically sorted, that is are in the same order of those of `adj`.
- `blocks`: list of vectors, containing the partitioning of nodes (represented with an integer number) into blocks;
- `W`: vector of weights.

See Also

[adj.upper.tri](#)

Examples

```
data(graph);
data(scores);
Y <- S[3,];
adj <- adj.upper.tri(g);
S.GPAV <- GPAV(Y,W=NULL,adj);
```

GPAV.over.examples *GPAV Over Examples*

Description

Function to compute GPAV across all the examples.

Usage

```
GPAV.over.examples(S, g, W = NULL)
```

Arguments

<code>S</code>	a named flat scores matrix with examples on rows and classes on columns (root node included).
<code>g</code>	a graph of class <code>graphNEL</code> . It represents the hierarchy of the classes.
<code>W</code>	vector of weight relative to a single example. If the vector <code>W</code> is not specified (def. <code>W=NULL</code>), it is assumed that <code>W</code> is a unitary vector of the same length of the columns' number of the matrix <code>S</code> (root node included).

Value

a named matrix with the scores of the classes corrected according to the GPAV algorithm.

See Also

[GPAV.parallel](#)

Examples

```
data(graph);
data(scores);
S.GPAV <- GPAV.over.examples(S,W=NULL,g);
```

GPAV.parallel

*GPAV Over Examples – Parallel Implementation***Description**

Function to compute GPAV across all the examples (parallel implementation).

Usage

```
GPAV.parallel(S, g, W = NULL, ncores = 8)
```

Arguments

S	a named flat scores matrix with examples on rows and classes on columns (root node included).
g	a graph of class graphNEL. It represents the hierarchy of the classes.
W	vector of weight relative to a single example. If the vector W is not specified (def. W=NULL), it is assumed that W is a unitary vector of the same length of the columns' number of the matrix S (root node included).
ncores	number of cores to use for parallel execution (def. 8).

Value

a named matrix with the scores of the classes corrected according to the GPAV algorithm.

Examples

```
data(graph);
data(scores);
if (Sys.info()['sysname']!="Windows"){
  S.GPAV <- GPAV.parallel(S,W=NULL,g,ncores=2);
}
```

graph.levels

Build Graph Levels

Description

This function groups a set of nodes in according to their maximum depth in the graph. It first inverts the weights of the graph and then applies the Bellman Ford algorithm to find the shortest path, achieving in this way the longest path.

Usage

```
graph.levels(g, root = "00")
```

Arguments

g	an object of class graphNEL
root	name of the root node (def. root="00")

Value

a list of the nodes grouped w.r.t. the distance from the root: the first element of the list corresponds to the root node (level 0), the second to nodes at maximum distance 1 (level 1), the third to the node at maximum distance 3 (level 2) and so on.

Examples

```
data(graph);
root <- root.node(g);
lev <- graph.levels(g, root=root);
```

Heuristic-Methods

Obozinski Heuristic Methods

Description

Implementation of the Heuristic Methods MAX, AND, OR (*Obozinski et al., Genome Biology, 2008, doi:10.1186/gb-2008-9-s1-s6*).

Usage

```
heuristic.max(S, g, root = "00")

heuristic.and(S, g, root = "00")

heuristic.or(S, g, root = "00")
```

Arguments

S	a named flat scores matrix with examples on rows and classes on columns.
g	a graph of class graphNEL. It represents the hierarchy of the classes.
root	name of the class that it is the top-level (root) of the hierarchy (def:00).

Details

Heuristic Methods:

1. **MAX**: reports the largest logist regression (LR) value of self and all descendants: $p_i = \max_{j \in \text{descendants}(i)} \hat{p}_j$;
2. **AND**: reports the product of LR values of all ancestors and self. This is equivalent to computing the probability that all ancestral terms are "on" assuming that, conditional on the data, all predictions are independent: $p_i = \prod_{j \in \text{ancestors}(i)} \hat{p}_j$;
3. **OR**: computes the probability that at least one of the descendant terms is "on" assuming again that, conditional on the data, all predictions are independent: $1 - p_i = \prod_{j \in \text{descendants}(i)} (1 - \hat{p}_j)$;

Value

a matrix with the scores of the classes corrected according to the chosen heuristic algorithm.

Examples

```
data(graph);
data(scores);
data(labels);
root <- root.node(g);
S.max <- heuristic.max(S,g,root);
S.and <- heuristic.and(S,g,root);
S.or <- heuristic.or(S,g,root);
```

hierarchical.checkers *Hierarchical constraints checker*

Description

Check if the true path rule is violated or not. In other words this function checks if the score of a parent or an ancestor node is always larger or equal than that of its children or descendants nodes.

Usage

```
check.hierarchy.single.sample(y.hier, g, root = "00")

check.hierarchy(S.hier, g, root = "00")
```

Arguments

<code>y.hier</code>	vector of scores relative to a single example. This must be a named numeric vector.
<code>g</code>	a graph of class <code>graphNEL</code> . It represents the hierarchy of the classes.
<code>root</code>	name of the class that is on the top-level of the hierarchy (def. <code>root=""00"</code>).
<code>S.hier</code>	the matrix with the scores of the classes corrected in according to hierarchy. This must be a named matrix: rows are examples and columns are classes.

Value

return a list of 3 elements:

- `Status`:
 - OK if none hierarchical constraints have been broken;
 - NOTOK if there is at least one hierarchical constraint broken;
- `hierarchy_constraints_broken`:
 - TRUE: example did not respect the hierarchical constraints;
 - FALSE: example broke the hierarchical constraints;
- `hierarchy_constraints_satisfied`: how many terms satisfied the hierarchical constraint;

Examples

```
data(graph);
data(scores);
root <- root.node(g);
S.hier <- htd(S,g,root);
S.hier.single.example <- S.hier[sample(ncol(S.hier),1),];
check.hierarchy.single.sample(S.hier.single.example, g, root=root);
check.hierarchy(S.hier, g, root);
```

HTD-DAG

HTD-DAG

Description

Implementation of a top-down procedure to correct the scores of the hierarchy according to the constraints that the score of a node cannot be greater than a score of its parents.

Usage

```
htd(S, g, root = ""00")
```

Arguments

<code>S</code>	a named flat scores matrix with examples on rows and classes on columns.
<code>g</code>	a graph of class <code>graphNEL</code> . It represents the hierarchy of the classes.
<code>root</code>	name of the class that it is the top-level (root) of the hierarchy (def: <code>00</code>).

Details

The HTD-DAG algorithm modifies the flat scores according to the hierarchy of a DAG through a unique run across the nodes of the graph. For a given example $x \in X$, the flat predictions $f(x) = \hat{y}$ are hierarchically corrected to \bar{y} , by per-level visiting the nodes of the DAG from top to bottom according to the following simple rule:

$$\bar{y}_i := \begin{cases} \hat{y}_i & \text{if } i \in \text{root}(G) \\ \min_{j \in \text{par}(i)} \bar{y}_j & \text{if } \min_{j \in \text{par}(i)} \bar{y}_j < \hat{y}_i \\ \hat{y}_i & \text{otherwise} \end{cases}$$

The node levels correspond to their maximum path length from the root.

Value

a matrix with the scores of the classes corrected according to the HTD-DAG algorithm.

See Also

[graph.levels](#), [hierarchical.checkers](#)

Examples

```
data(graph);
data(scores);
root <- root.node(g);
S.htd <- htd(S,g,root);
```

lexicographical.topological.sort

Lexicographical Topological Sorting

Description

Nodes of a graph are sorted according to a lexicographical topological ordering.

Usage

```
lexicographical.topological.sort(g)
```

Arguments

`g` an object of class `graphNEL`

Details

A topological sorting is a linear ordering of the nodes such that given an edge from u to v , the node u comes before node v in the ordering. Topological sorting is not possible if the graph g is not a DAG. To implement the topological sorting algorithm we applied the Kahn's algorithm.

Value

a vector in which the nodes of the graph *g* are sorted according to a lexicographical topological order.

Examples

```
data(graph);
T <- lexicographical.topological.sort(g);
```

Multilabel.F.measure *Multilabel F-measure*

Description

Method for computing Precision, Recall, Specificity, Accuracy and F-measure for multiclass and multilabel classification.

Usage

```
F.measure.multilabel(target, predicted, b.per.example = FALSE)

## S4 method for signature 'matrix,matrix'
F.measure.multilabel(target, predicted,
  b.per.example = FALSE)
```

Arguments

target	matrix with the target multilabel: rows correspond to examples and columns to classes. $target[i, j] = 1$ if example i belongs to class j , $target[i, j] = 0$ otherwise.
predicted	a numeric matrix with discrete predicted values: rows correspond to examples and columns to classes. $predicted[i, j] = 1$ if example i is predicted belonging to class j , $target[i, j] = 0$ otherwise.
b.per.example	boolean. <ul style="list-style-type: none"> • TRUE: results are returned for each example; • FALSE: only the average results are returned;

Details

Names of rows and columns of target and predicted matrix must be provided in the same order, otherwise a stop message is returned.

Value

Two different outputs respect to the input parameter `b.per.example`:

- `b.per.example==FALSE`: a list with a single element average. A named vector with average precision (P), recall (R), specificity (S), F-measure (F), average F-measure (avF) and Accuracy (A) across examples. F is the F-measure computed as the harmonic mean between the average precision and recall; av.F is the F-measure computed as the average across examples;
- `b.per.example==TRUE`: a list with two elements:
 1. average: a named vector with average precision (P), recall (R), specificity (S), F-measure (F), average F-measure (avF) and Accuracy (A) across examples;
 2. per.example: a named matrix with the Precision (P), Recall (R), Specificity (S), Accuracy (A), F-measure (F) and av.F-measure (av.F) for each example. Row names correspond to examples, column names correspond respectively to Precision (P), Recall (R), Specificity (S), Accuracy (A), F-measure (F) and av.F-measure (av.F);

Examples

```
data(labels);
data(scores);
data(graph);
root <- root.node(g);
L <- L[, -which(colnames(L)==root)];
S <- S[, -which(colnames(S)==root)];
S[S>0.7] <- 1;
S[S<0.7] <- 0;
FMM <- F.measure.multilabel(L,S);
```

normalize.max

Max normalization

Description

Function to normalize the scores of a flat scores matrix per class.

Usage

```
normalize.max(S)
```

Arguments

`S` matrix with the raw non normalized scores. Rows are examples and columns are classes.

Details

The scores of each class are normalized by dividing the score values for the maximum score of that class. If the max score of a class is zero, no normalization is needed, otherwise NaN value will be printed as results of 0 out of 0 division.

Value

A score matrix with the same dimensions of *S*, but with scores max/normalized separately for each class.

Examples

```
data(scores);
maxnorm <- normalize.max(S);
```

parents

Build parents

Description

Compute the parents for each node of a graph.

Usage

```
get.parents(g, root = "00")

get.parents.top.down(g, levels, root = "00")

get.parents.bottom.up(g, levels, root = "00")

get.parents.topological.sorting(g, root = "00")
```

Arguments

<i>g</i>	a graph of class <i>graphNEL</i> . It represents the hierarchy of the classes.
<i>root</i>	name of the root node (def. <i>root</i> ="00").
<i>levels</i>	a list of character vectors. Each component represents a graph level and the elements of any. component correspond to nodes. The level 0 coincides with the root node.

Value

get.parents returns a named list of character vectors. Each component corresponds to a node *x* of the graph (i.e. child node) and its vector is the set of its parents (the root node is not included).

get.parents.top.down returns a named list of character vectors. Each component corresponds to a node *x* of the graph (i.e. child node) and its vector is the set of its parents. The nodes order follows the levels of the graph from root (excluded) to leaves.

get.parents.bottom.up returns a named list of character vectors. Each component corresponds to a node *x* of the graph (i.e. child node) and its vector is the set of its parents. The nodes are ordered from leaves to root (excluded).

get.parents.topological.sorting a named list of character vectors. Each component corresponds to a node *x* of the graph (i.e. child node) and its vector is the set of its parents. The nodes are ordered according to a topological sorting, i.e. parents node come before children node.

See Also[graph.levels](#)**Examples**

```

data(graph);
root <- root.node(g)
parents <- get.parents(g, root=root);
lev <- graph.levels(g, root=root);
parents.tod <- get.parents.top.down(g, lev, root=root);
parents.bup <- get.parents.bottom.up(g, lev, root=root);
parents.tsort <- get.parents.topological.sorting(g, root=root);

```

PXR

*Precision-Recall Measure***Description**

Functions to compute the Precision-Recall (PXR) values through **precrec** package.

Usage

```

precision.at.all.recall.levels.single.class(labels, scores)

precision.at.given.recall.levels.over.classes(target, predicted,
  folds = NULL, seed = NULL, recall.levels = seq(from = 0.1, to = 1,
  by = 0.1))

```

Arguments

labels	vector of the true labels (0 negative, 1 positive examples).
scores	a numeric vector of the values of the predicted labels (scores).
target	matrix with the target multilabel: rows correspond to examples and columns to classes. $target[i, j] = 1$ if example i belongs to class j , $target[i, j] = 0$ otherwise.
predicted	a numeric matrix with predicted values (scores): rows correspond to examples and columns to classes.
folds	number of folds on which computing the PXR. If folds=NULL (def.), the PXR is computed one-shot, otherwise the PXR is computed averaged across folds.
seed	initialization seed for the random generator to create folds. Set seed only if folds≠NULL. If seed=NULL and folds≠NULL, the PXR averaged across folds is computed without seed initialization.
recall.levels	a vector with the desired recall levels (def: from:0.1, to:0.9, by:0.1).

Details

`precision.at.all.recall.levels.single.class` computes the precision at all recall levels just for a single class.

`precision.at.given.recall.levels.over.classes` computes the precision at fixed recall levels over classes.

Value

`precision.at.all.recall.levels.single.class` returns a two-columns matrix, representing a pair of precision and recall values. The first column is the precision, the second the recall; `precision.at.given.recall.levels.over.classes` returns a list with two elements:

1. `avgPXR`: a vector with the average precision at different recall levels across classes;
2. `PXR`: a matrix with the precision at different recall levels: rows are classes, columns precision at different recall levels;

Examples

```
data(labels);
data(scores);
data(graph);
root <- root.node(g);
L <- L[, -which(colnames(L)==root)];
S <- S[, -which(colnames(S)==root)];
labels <- L[,1];
scores <- S[,1];
rec.levels <- seq(from=0.25, to=1, by=0.25);
PXR.single <- precision.at.all.recall.levels.single.class(labels, scores);
PXR <- precision.at.given.recall.levels.over.classes(L, S, folds=5, seed=23,
  recall.levels=rec.levels);
```

`read.graph`

Read a directed graph from a file

Description

A directed graph is read from a file and a `graphNEL` object is built.

Usage

```
read.graph(file = "graph.txt.gz")
```

Arguments

<code>file</code>	name of the file to be read. The format of the file is a sequence of rows and each row corresponds to an edge represented through a pair of vertices separated by blanks. The extension of the file can be or plain format (".txt") or compressed (".gz").
-------------------	--

Value

an object of class graphNEL.

Examples

```
ed <- system.file("extdata/graph.edges.txt.gz", package= "HEMDAG");  
g <- read.graph(file=ed);
```

read.undirected.graph *Read an undirected graph from a file*

Description

The graph is read from a file and a graphNEL object is built. The format of the input file is a sequence of rows. Each row corresponds to an edge represented through a pair of vertices separated by blanks, and the weight of the edge.

Usage

```
read.undirected.graph(file = "graph.txt.gz")
```

Arguments

file	name of the file to be read. The extension of the file can be or plain format (".txt") or compressed (".gz").
------	---

Value

a graph of class graphNEL.

Examples

```
edges <- system.file("extdata/edges.txt.gz", package="HEMDAG");  
g <- read.undirected.graph(file=edges);
```

root.node	<i>Root node</i>
-----------	------------------

Description

Find the root node of a directed graph.

Usage

```
root.node(g)
```

Arguments

g a graph of class graphNEL. It represents the hierarchy of the classes.

Value

name of the root node.

Examples

```
data(graph);
root <- root.node(g);
```

scores.normalization	<i>Scores Normalization Function</i>
----------------------	--------------------------------------

Description

Functions to normalize a flat scores matrix w.r.t. max normalization (MaxNorm) or quantile normalization (Qnorm)

Usage

```
scores.normalization(norm.type = "MaxNorm", S)
```

Arguments

norm.type can be one of the following two values:

- MaxNorm (def.): each score is divided w.r.t. the max of each class;
- Qnorm: a quantile normalization is applied. Library preprocessCore is used;

S a named flat scores matrix with examples on rows and classes on columns.

Details

To apply the quantile normalization the **preprocessCore** package must be installed.

Value

the matrix of the scores flat normalized w.r.t. MaxNorm or Qnorm.

Examples

```
data(scores);
norm.types <- c("MaxNorm", "Qnorm");
for(norm.type in norm.types){
  scores.normalization(norm.type=norm.type, S=S)
}
```

`specific.annotation.list`*Specific annotations list*

Description

Construct a list of the most specific annotations starting from the table of the most specific annotations.

Usage

```
specific.annotation.list(ann)
```

Arguments

ann	annotation matrix (0/1). Rows are examples and columns are most specific terms. It must be a named matrix.
-----	--

Value

a named list, where the names of each component correspond to an examples (genes) and the elements of each component are the most specific classes associated to that genes.

See Also

[specific.annotation.matrix](#)

Examples

```
data(labels);
spec.list <- specific.annotation.list(L);
```

```
specific.annotation.matrix
```

specific annotation matrix

Description

Construct the labels matrix of the most specific OBO terms (as GO or HPO).

Usage

```
specific.annotation.matrix(file = "gene2pheno.txt.gz")
```

Arguments

file	text file representing the most specific associations gene-OBO. The file must be written as sequence of rows. Each row represents a gene/protein and all its associations with an ontology term pipe separated, <i>e.g.</i> : <i>gene1 obo1 ... oboN</i> . The input example file used here (def: "gene2pheno.txt") shows the gene and all its associations with an HPO terms.
------	--

Details

The input plain text file representing the most specific associations gene-OBO term can be obtained by cloning the GitHub repository [obogaf-parser](#), a perl5 module specifically designed to handle HPO and GO obo file and their gene annotation file (gaf file).

Value

the annotation matrix of the most specific annotations (0/1): rows are genes and columns are HPO terms. Let's denote M the labels matrix. If $M[i, j] = 1$, means that the gene i is annotated with the class j , otherwise $M[i, j] = 0$.

Examples

```
gene2pheno <- system.file("extdata/gene2pheno.txt.gz", package="HEMDAG");
spec.ann <- specific.annotation.matrix(file=gene2pheno);
```

```
stratified.cross.validation
```

Stratified Cross Validation

Description

Generate data for the stratified cross-validation.

Usage

```
do.stratified.cv.data.single.class(examples, positives, kk = 5,
  seed = NULL)

do.stratified.cv.data.over.classes(labels, examples, kk = 5,
  seed = NULL)
```

Arguments

examples	indices or names of the examples. Can be either a vector of integers or a vector of names.
positives	vector of integers or vector of names. The indices (or names) refer to the indices (or names) of 'positive' examples.
kk	number of folds (def. kk=5).
seed	seed of the random generator (def. seed=NULL). If is set to NULL no initialization is performed.
labels	labels matrix. Rows are genes and columns are classes. Let's denote M the labels matrix. If $M[i, j] = 1$, means that the gene i is annotated with the class j , otherwise $M[i, j] = 0$.

Details

the folds are *stratified*, i.e. contain the same amount of positive and negative examples.

Value

do.stratified.cv.data.single.class returns a list with 2 two component:

- fold.non.positives: a list with k components. Each component is a vector with the indices (or names) of the non-positive elements. Indices (or names) refer to row numbers (or names) of a data matrix;
- fold.positives: a list with k components. Each component is a vector with the indices (or names) of the positive elements. Indices (or names) refer to row numbers (or names) of a data matrix;

do.stratified.cv.data.over.classes returns a list with n components, where n is the number of classes of the labels matrix. Each component n is in turn a list with k elements, where k is the number of folds. Each fold contains an equal amount of positives and negatives examples.

Examples

```
data(labels);
examples.index <- 1:nrow(L);
examples.name <- rownames(L);
positives <- which(L[,3]==1);
x <- do.stratified.cv.data.single.class(examples.index, positives, kk=5, seed=23);
y <- do.stratified.cv.data.single.class(examples.name, positives, kk=5, seed=23);
z <- do.stratified.cv.data.over.classes(L, examples.index, kk=5, seed=23);
k <- do.stratified.cv.data.over.classes(L, examples.name, kk=5, seed=23);
```

TPR-DAG-cross-validation

TPR-DAG cross-validation experiments

Description

High level function to correct the computed scores in a hierarchy according to the chosen ensemble algorithm.

Usage

```
Do.TPR.DAG(threshold = seq(from = 0.1, to = 0.9, by = 0.1),
  weight = seq(from = 0.1, to = 0.9, by = 0.1), kk = 5, folds = 5,
  seed = 23, norm = TRUE, norm.type = NULL, positive = "children",
  bottomup = "threshold.free", topdown = "HTD", W = NULL,
  parallel = FALSE, ncores = 1, n.round = 3, f.criterion = "F",
  metric = NULL, recall.levels = seq(from = 0.1, to = 1, by = 0.1),
  compute.performance = FALSE, flat.file = flat.file,
  ann.file = ann.file, dag.file = dag.file, flat.dir = flat.dir,
  ann.dir = ann.dir, dag.dir = dag.dir,
  hierScore.dir = hierScore.dir, perf.dir = perf.dir)
```

Arguments

threshold	range of threshold values to be tested in order to find the best threshold (def: from:0.1, to:0.9, by:0.1). The denser the range is, the higher the probability to find the best threshold is, but obviously the execution time will be higher. Set this parameter only for the <i>thresholded</i> variants; for the <i>threshold-free</i> variants, the parameter threshold is automatically set to zero.
weight	range of weight values to be tested in order to find the best weight (def: from:0.1, to:0.9, by:0.1). The denser the range is, the higher the probability to find the best threshold is, but obviously the execution time will be higher. Set this parameter only for the <i>weighted</i> variants; for the <i>weight-free</i> variants, the parameter weight is automatically set to zero.
kk	number of folds of the cross validation (def: kk=5) on which tuning the parameters threshold, weight and tau of the parametric variants of the hierarchical ensemble algorithms. For the non-parametric variants (i.e. if bottomup = threshold.free), the parameter kk is automatically set to zero.
folds	number of folds of the cross validation on which computing the performance metrics averaged across folds (def: 5). If folds=NULL, the performance metrics are computed one-shot, otherwise the performance metrics are averaged across folds. If compute.performance is set to FALSE, folds is automatically set to NULL.
seed	initialization seed for the random generator to create folds (def: 23). If NULL folds are generated without seed initialization. The parameter seed controls both the parameter kk and the parameter folds. If compute.performance is set

	to FALSE and bottomup is set to threshold.free, then seed is automatically set to NULL.
norm	<p>boolean value: should the flat scores matrix be normalized?</p> <ul style="list-style-type: none"> • TRUE (def.): the flat scores matrix has been already normalized in according to a normalization method; • FALSE: the flat scores matrix has not been normalized yet. See the parameter norm.type to set the on the fly normalization method to apply among those possible;
norm.type	<p>can be one of the following three values:</p> <ol style="list-style-type: none"> 1. NULL (def.): set norm.type to NULL if and only if the parameter norm is set to TRUE; 2. MaxNorm: each score is divided for the maximum of each class; 3. Qnorm: quantile normalization. preprocessCore package is used;
positive	<p>choice of the <i>positive</i> nodes to be considered in the bottom-up strategy. Can be one of the following values:</p> <ul style="list-style-type: none"> • children (def.): for each node are considered its positive children; • descendants: for each node are considered its positive descendants;
bottomup	<p>strategy to enhance the flat predictions by propagating the positive predictions from leaves to root. It can be one of the following values:</p> <ul style="list-style-type: none"> • threshold.free (def.): positive nodes are selected on the basis of the threshold.free strategy (def.); • threshold: positive nodes are selected on the basis of the threshold strategy; • weighted.threshold.free: positive nodes are selected on the basis of the weighted.threshold.free strategy; • weighted.threshold: positive nodes are selected on the basis of the weighted.threshold strategy; • tau: positive nodes are selected on the basis of the tau strategy; NOTE: tau is only a DESCENS variants. If you use tau strategy you must set the parameter positive=descendants;
topdown	<p>strategy to make the scores hierarchy-consistent. It can be one of the following values:</p> <ul style="list-style-type: none"> • HTD (def.): HTD-DAG strategy is applied (HTD-DAG); • GPAV: GPAV strategy is applied (GPAV);
W	<p>vector of weight relative to a single example. If the vector W is not specified (by def. W=NULL), W is a unitary vector of the same length of the columns' number of the flat scores matrix (root node included). Set W only if topdown=GPAV.</p>
parallel	<p>boolean value:</p> <ul style="list-style-type: none"> • TRUE: execute the parallel implementation of GPAV (GPAV.parallel); • FALSE (def.): execute the sequential implementation of GPAV (GPAV.over.examples); <p>Use parallel if and only if topdown=GPAV; otherwise set parallel=FALSE.</p>

<code>ncores</code>	number of cores to use for parallel execution (def. 8). Set <code>ncores=1</code> if <code>parallel=FALSE</code> , otherwise set <code>ncores</code> to the desired number of cores. Use <code>ncores</code> only if <code>topdown=GPAV</code> ; otherwise set <code>parallel=1</code> .
<code>n.round</code>	number of rounding digits to be applied to the hierarchical scores matrix (def. 3). It is used for choosing the best threshold on the basis of the best F-measure. If <code>compute.performance</code> is set to <code>FALSE</code> and <code>bottomup</code> is set to <code>threshold.free</code> , then <code>n.round</code> is automatically set to <code>NULL</code> .
<code>f.criterion</code>	<p>character. Type of F-measure to be used to select the best F-measure. Two possibilities:</p> <ol style="list-style-type: none"> 1. <code>F</code> (def.): corresponds to the harmonic mean between the average precision and recall; 2. <code>avF</code>: corresponds to the per-example F-score averaged across all the examples; <p>If <code>compute.performance</code> is set to <code>FALSE</code> and <code>bottomup</code> is set to <code>threshold.free</code>, then <code>f.criterion</code> is automatically set to <code>NULL</code>.</p>
<code>metric</code>	<p>a string character specifying the performance metric on which to maximize the parametric ensemble variant. It can be one of the following values:</p> <ol style="list-style-type: none"> 1. <code>PRC</code>: the parametric ensemble variant is maximized on the basis of <code>AUPRC</code> (AUPRC); 2. <code>FMAX</code>: the parametric ensemble variant is maximized on the basis of <code>Fmax</code> (Multilabel.F.measure); 3. <code>NULL</code>: on the <code>threshold.free</code> variant none parameter optimization is needed, since the variant is non-parametric. So, if <code>bottomup=threshold.free</code> set <code>metric=NULL</code> (def.);
<code>recall.levels</code>	a vector with the desired recall levels (def: from:0.1, to:0.9, by:0.1) to compute the Precision at fixed Recall level (PXR). If <code>compute.performance=FALSE</code> the parameter <code>recall.levels</code> is automatically set to <code>NULL</code> .
<code>compute.performance</code>	<p>boolean value: should the flat and hierarchical performance (<code>AUPRC</code>, <code>AUROC</code>, <code>PXR</code>, <code>multilabel F-score</code>) be returned?</p> <ul style="list-style-type: none"> • <code>FALSE</code> (def.): performance are not computed and just the hierarchical scores matrix is returned; • <code>TRUE</code>: both performance and hierarchical scores matrix are returned;
<code>flat.file</code>	name of the file containing the flat scores matrix to be normalized or already normalized (without <code>rda</code> extension).
<code>ann.file</code>	name of the file containing the label matrix of the examples (without <code>rda</code> extension).
<code>dag.file</code>	name of the file containing the graph that represents the hierarchy of the classes (without <code>rda</code> extension).
<code>flat.dir</code>	relative path where flat scores matrix is stored.
<code>ann.dir</code>	relative path where annotation matrix is stored.
<code>dag.dir</code>	relative path where graph is stored.
<code>hierScore.dir</code>	relative path where the hierarchical scores matrix must be stored.
<code>perf.dir</code>	relative path where the performance measures must be stored. If <code>compute.performance=FALSE</code> , the parameter <code>perf.dir</code> is automatically set to <code>NULL</code> .

Details

The parametric hierarchical ensemble variants are cross-validated by maximizing in according to the metric chosen in the parameter `metric`, that is F-measure ([Multilabel.F.measure](#)) or AUPRC ([AUPRC](#)).

The function checks if the number of classes between the flat scores matrix and the annotations matrix mismatched. If so, the number of terms of the annotations matrix is shrunk to the number of terms of the flat scores matrix and the corresponding subgraph is computed as well. N.B.: it is supposed that all the nodes of the subgraph are accessible from the root.

We excluded the predictions of the root node in computing all the performances, since it is a *dummy* node added to the ontology for practical reasons (e.g. some graph-based software may require a single root node to work). However, the root node scores are stored in the hierarchical scores matrix.

Value

Two rda files stored in the respective output directories:

1. Hierarchical Scores Results: a matrix with examples on rows and classes on columns representing the computed hierarchical scores for each example and for each considered class. It is stored in the `hierScore.dir` directory;
2. Performance Measures: *flat* and *hierarchical* performance results:
 - (a) AUPRC results computed through `AUPRC.single.over.classes` ([AUPRC](#));
 - (b) AUROC results computed through `AUROC.single.over.classes` ([AUROC](#));
 - (c) PXR results computed through `precision.at.given.recall.levels.over.classes` ([PXR](#));
 - (d) FMM results computed through `compute.Fmeasure.multilabel` ([FMM](#));

It is stored in the `perf.dir` directory.

See Also

[TPR-DAG-variants](#)

Examples

```
data(graph);
data(scores);
data(labels);
tmpdir <- paste0(tmpdir(),"/");
save(g, file=paste0(tmpdir,"graph.rda"));
save(L, file=paste0(tmpdir,"labels.rda"));
save(S, file=paste0(tmpdir,"scores.rda"));
dag.dir <- flat.dir <- ann.dir <- tmpdir;
hierScore.dir <- perf.dir <- tmpdir;
dag.file <- "graph";
flat.file <- "scores";
ann.file <- "labels";
threshold <- weight <- 0;
norm.type <- "MaxNorm";
```

```

positive <- "children";
bottomup <- "threshold.free";
topdown <- "HTD";
recall.levels <- seq(from=0.25, to=1, by=0.25);
Do.TPR.DAG(threshold=threshold, weight=weight, kk=NULL, folds=NULL, seed=NULL,
norm=FALSE, norm.type=norm.type, positive=positive, bottomup=bottomup, topdown=topdown,
W=NULL, compute.performance=TRUE, parallel=FALSE, ncores=1, n.round=3, f.criterion="F",
metric=NULL, recall.levels=recall.levels, flat.file=flat.file, ann.file=ann.file,
dag.file=dag.file, flat.dir=flat.dir, ann.dir=ann.dir, dag.dir=dag.dir,
hierScore.dir=hierScore.dir, perf.dir=perf.dir);

```

TPR-DAG-holdout

TPR-DAG holdout experiments

Description

High level function to correct the computed scores in a hierarchy according to the chosen ensemble algorithm.

Usage

```

Do.TPR.DAG.holdout(threshold = seq(from = 0.1, to = 0.9, by = 0.1),
  weight = seq(from = 0.1, to = 1, by = 0.1), kk = 5, folds = 5,
  seed = 23, norm = TRUE, norm.type = NULL, positive = "children",
  bottomup = "threshold.free", topdown = "HTD", W = NULL,
  parallel = FALSE, ncores = 1, recall.levels = seq(from = 0.1, to =
  1, by = 0.1), n.round = 3, f.criterion = "F", metric = NULL,
  compute.performance = FALSE, flat.file = flat.file,
  ann.file = ann.file, dag.file = dag.file,
  ind.test.set = ind.test.set, ind.dir = ind.dir,
  flat.dir = flat.dir, ann.dir = ann.dir, dag.dir = dag.dir,
  hierScore.dir = hierScore.dir, perf.dir = perf.dir)

```

Arguments

threshold	range of threshold values to be tested in order to find the best threshold (def: from:0.1, to:0.9, by:0.1). The denser the range is, the higher the probability to find the best threshold is, but obviously the execution time will be higher. Set this parameter only for the <i>thresholded</i> variants; for the <i>threshold-free</i> variants, the parameter threshold is automatically set to zero.
weight	range of weight values to be tested in order to find the best weight (def: from:0.1, to:0.9, by:0.1). The denser the range is, the higher the probability to find the best threshold is, but obviously the execution time will be higher. Set this parameter only for the <i>weighted</i> variants; for the <i>weight-free</i> variants, the parameter weight is automatically set to zero.

kk	number of folds of the cross validation (def: kk=5) on which tuning the parameters threshold, weight and tau of the parametric variants of the hierarchical ensemble algorithms. For the non-parametric variants (i.e. if bottomup = threshold.free), the parameter kk is automatically set to zero.
folds	number of folds of the cross validation on which computing the performance metrics averaged across folds (def: 5). If folds=NULL, the performance metrics are computed one-shot, otherwise the performance metrics are averaged across folds. If compute.performance is set to FALSE, folds is automatically set to NULL.
seed	initialization seed for the random generator to create folds (def: 23). If NULL folds are generated without seed initialization. The parameter seed controls both the parameter kk and the parameter folds. If compute.performance is set to FALSE and bottomup is set to threshold.free, then seed is automatically set to NULL.
norm	boolean value: should the flat scores matrix be normalized? <ul style="list-style-type: none"> • TRUE (def.): the flat scores matrix has been already normalized in according to a normalization method; • FALSE: the flat scores matrix has not been normalized yet. See the parameter norm.type to set the on the fly normalization method to apply among those possible;
norm.type	can be one of the following three values: <ol style="list-style-type: none"> 1. NULL (def.): set norm.type to NULL if and only if the parameter norm is set to TRUE; 2. MaxNorm: each score is divided for the maximum of each class; 3. Qnorm: quantile normalization. preprocessCore package is used;
positive	choice of the <i>positive</i> nodes to be considered in the bottom-up strategy. Can be one of the following values: <ul style="list-style-type: none"> • children (def.): for each node are considered its positive children; • descendants: for each node are considered its positive descendants;
bottomup	strategy to enhance the flat predictions by propagating the positive predictions from leaves to root. It can be one of the following values: <ul style="list-style-type: none"> • threshold.free (def.): positive nodes are selected on the basis of the threshold.free strategy (def.); • threshold: positive nodes are selected on the basis of the threshold strategy; • weighted.threshold.free: positive nodes are selected on the basis of the weighted.threshold.free strategy; • weighted.threshold: positive nodes are selected on the basis of the weighted.threshold strategy; • tau: positive nodes are selected on the basis of the tau strategy; NOTE: tau is only a DESCENS variants. If you use tau strategy you must set the parameter positive=descendants;
topdown	strategy to make the scores hierarchy-consistent. It can be one of the following values:

	<ul style="list-style-type: none"> • HTD (def.): HTD-DAG strategy is applied (HTD-DAG); • GPAV: GPAV strategy is applied (GPAV);
<code>W</code>	vector of weight relative to a single example. If the vector <code>W</code> is not specified (by def. <code>W=NULL</code>), <code>W</code> is a unitary vector of the same length of the columns' number of the flat scores matrix (root node included). Set <code>W</code> only if <code>topdown=GPAV</code> .
<code>parallel</code>	<p>boolean value:</p> <ul style="list-style-type: none"> • TRUE: execute the parallel implementation of GPAV (GPAV.parallel); • FALSE (def.): execute the sequential implementation of GPAV (GPAV.over.examples); <p>Use <code>parallel</code> if and only if <code>topdown=GPAV</code>; otherwise set <code>parallel=FALSE</code>.</p>
<code>ncores</code>	number of cores to use for parallel execution (def. 8). Set <code>ncores=1</code> if <code>parallel=FALSE</code> , otherwise set <code>ncores</code> to the desired number of cores. Use <code>ncores</code> if and only if <code>topdown=GPAV</code> ; otherwise set <code>parallel=1</code> .
<code>recall.levels</code>	a vector with the desired recall levels (def: from:0.1, to:0.9, by:0.1) to compute the Precision at fixed Recall level (PXR). If <code>compute.performance=FALSE</code> then <code>recall.levels</code> is automatically set to <code>NULL</code> .
<code>n.round</code>	number of rounding digits to be applied to the hierarchical scores matrix (def. 3). It is used for choosing the best threshold on the basis of the best F-measure. If <code>compute.performance</code> is set to <code>FALSE</code> and <code>bottomup</code> is set to <code>threshold.free</code> , then <code>n.round</code> is automatically set to <code>NULL</code> .
<code>f.criterion</code>	<p>character. Type of F-measure to be used to select the best F-measure. Two possibilities:</p> <ol style="list-style-type: none"> 1. F (def.): corresponds to the harmonic mean between the average precision and recall; 2. avF: corresponds to the per-example F-score averaged across all the examples; <p>If <code>compute.performance</code> is set to <code>FALSE</code> and <code>bottomup</code> is set to <code>threshold.free</code>, then <code>f.criterion</code> is automatically set to <code>NULL</code>.</p>
<code>metric</code>	<p>a string character specifying the performance metric on which to maximize the parametric ensemble variant. It can be one of the following values:</p> <ol style="list-style-type: none"> 1. PRC: the parametric ensemble variant is maximized on the basis of AUPRC (AUPRC); 2. FMAX: the parametric ensemble variant is maximized on the basis of Fmax (Multilabel.F.measure); 3. NULL: on the <code>threshold.free</code> variant none parameter optimization is needed, since the variant is non-parametric. So, if <code>bottomup=threshold.free</code> set <code>metric=NULL</code> (def.);
<code>compute.performance</code>	<p>boolean value: should the flat and hierarchical performance (AUPRC, AUROC, PXR, multilabel F-score) be returned?</p> <ul style="list-style-type: none"> • FALSE: performance are not computed and just the hierarchical scores matrix is returned; • TRUE (def.): both performance and hierarchical scores matrix are returned;
<code>flat.file</code>	name of the file containing the flat scores matrix to be normalized or already normalized (without rda extension).

<code>ann.file</code>	name of the file containing the label matrix of the examples (without rda extension).
<code>dag.file</code>	name of the file containing the graph that represents the hierarchy of the classes (without rda extension).
<code>ind.test.set</code>	name of the file containing a vector of integer numbers corresponding to the indices of the elements (rows) of scores matrix to be used in the test set.
<code>ind.dir</code>	relative path to folder where <code>ind.test.set</code> is stored.
<code>flat.dir</code>	relative path where flat scores matrix is stored.
<code>ann.dir</code>	relative path where annotation matrix is stored.
<code>dag.dir</code>	relative path where graph is stored.
<code>hierScore.dir</code>	relative path where the hierarchical scores matrix must be stored.
<code>perf.dir</code>	relative path where the performance measures must be stored. If <code>compute.performance=FALSE</code> , the parameter <code>perf.dir</code> is automatically set to <code>NULL</code> .

Details

The parametric hierarchical ensemble variants are cross-validated by maximizing in according to the metric chosen in the parameter `metric`, that is F-measure ([Multilabel.F.measure](#)) or AUPRC ([AUPRC](#)).

The function checks if the number of classes between the flat scores matrix and the annotations matrix mismatched. If so, the number of terms of the annotations matrix is shrunk to the number of terms of the flat scores matrix and the corresponding subgraph is computed as well. N.B.: it is supposed that all the nodes of the subgraph are accessible from the root.

We excluded the predictions of the root node in computing all the performances, since it is a *dummy* node added to the ontology for practical reasons (e.g. some graph-based software may require a single root node to work). However, the root node scores are stored in the hierarchical scores matrix.

Value

Two rda files stored in the respective output directories:

1. Hierarchical Scores Results: a matrix with examples on rows and classes on columns representing the computed hierarchical scores for each example and for each considered class. It is stored in the `hierScore.dir` directory;
2. Performance Measures: *flat* and *hierarchical* performance results:
 - (a) AUPRC results computed through `AUPRC.single.over.classes` ([AUPRC](#));
 - (b) AUROC results computed through `AUROC.single.over.classes` ([AUROC](#));
 - (c) PXR results computed through `precision.at.given.recall.levels.over.classes` ([PXR](#));
 - (d) FMM results computed through `compute.Fmeasure.multilabel` ([FMM](#));

It is stored in the `perf.dir` directory.

See Also

[TPR-DAG-variants](#)

Examples

```

data(graph);
data(scores);
data(labels);
data(test.index);
tmpdir <- paste0(tmpdir(), "/");
save(g, file=paste0(tmpdir, "graph.rda"));
save(L, file=paste0(tmpdir, "labels.rda"));
save(S, file=paste0(tmpdir, "scores.rda"));
save(test.index, file=paste0(tmpdir, "test.index.rda"));
ind.dir <- dag.dir <- flat.dir <- ann.dir <- tmpdir;
hierScore.dir <- perf.dir <- tmpdir;
ind.test.set <- "test.index";
dag.file <- "graph";
flat.file <- "scores";
ann.file <- "labels";
threshold <- weight <- 0;
norm.type <- "MaxNorm";
positive <- "children";
bottomup <- "threshold.free";
topdown <- "HTD";
recall.levels <- seq(from=0.25, to=1, by=0.25);
Do.TPR.DAG.holdout(threshold=threshold, weight=weight, kk=NULL, folds=NULL, seed=NULL, norm=FALSE,
norm.type=norm.type, positive=positive, bottomup=bottomup, topdown=topdown, W=NULL,
compute.performance=TRUE, parallel=FALSE, ncores=1, recall.levels=recall.levels, n.round=3,
f.criterion="F", metric=NULL, flat.file=flat.file, ann.file=ann.file, dag.file=dag.file,
ind.test.set=ind.test.set, ind.dir=ind.dir, flat.dir=flat.dir, ann.dir=ann.dir, dag.dir=dag.dir,
hierScore.dir=hierScore.dir, perf.dir=perf.dir);

```

TPR-DAG-variants

TPR-DAG Ensemble Variants

Description

Function gathering the true-path-rule-based hierarchical learning ensemble algorithms and its variants. In their more general form the TPR-DAG algorithms adopt a two step learning strategy:

1. in the first step they compute a *per-level bottom-up* visit from the leaves to the root to propagate positive predictions across the hierarchy;
2. in the second step they compute a *per-level top-down* visit from the root to the leaves in order to assure the hierarchical consistency of the predictions;

Usage

```

TPR.DAG(S, g, root = "00", positive = "children",
bottomup = "threshold.free", topdown = "HTD", t = 0, w = 0,
W = NULL, parallel = FALSE, ncores = 1)

```

Arguments

<code>S</code>	a named flat scores matrix with examples on rows and classes on columns.
<code>g</code>	a graph of class graphNEL. It represents the hierarchy of the classes.
<code>root</code>	name of the class that it is on the top-level of the hierarchy (def. <code>root=""00"</code>).
<code>positive</code>	choice of the <i>positive</i> nodes to be considered in the bottom-up strategy. Can be one of the following values: <ul style="list-style-type: none"> • <code>children</code> (def.): for each node are considered its positive children; • <code>descendants</code>: for each node are considered its positive descendants;
<code>bottomup</code>	strategy to enhance the flat predictions by propagating the positive predictions from leaves to root. It can be one of the following values: <ul style="list-style-type: none"> • <code>threshold.free</code> (def.): positive nodes are selected on the basis of the <code>threshold.free</code> strategy (def.); • <code>threshold</code>: positive nodes are selected on the basis of the <code>threshold</code> strategy; • <code>weighted.threshold.free</code>: positive nodes are selected on the basis of the <code>weighted.threshold.free</code> strategy; • <code>weighted.threshold</code>: positive nodes are selected on the basis of the <code>weighted.threshold</code> strategy; • <code>tau</code>: positive nodes are selected on the basis of the <code>tau</code> strategy. NOTE: <code>tau</code> is only a DESCENS variants. If you use <code>tau</code> strategy you must set the parameter <code>positive=descendants</code>;
<code>topdown</code>	strategy to make the scores hierarchy-consistent. It can be one of the following values: <ul style="list-style-type: none"> • <code>HTD</code> (def.): HTD-DAG strategy is applied (HTD-DAG); • <code>GPAV</code>: GPAV strategy is applied (GPAV);
<code>t</code>	threshold for the choice of positive nodes (def. <code>t=0</code>). Set <code>t</code> only for the variants that requiring a threshold for the selection of the positive nodes, otherwise set <code>t</code> to zero.
<code>w</code>	weight to balance between the contribution of the node <i>i</i> and that of its positive nodes. Set <code>w</code> only for the <i>weighted</i> variants, otherwise set <code>w</code> to zero.
<code>W</code>	vector of weight relative to a single example. If the vector <code>W</code> is not specified (by def. <code>W=NULL</code>), <code>W</code> is a unitary vector of the same length of the columns' number of the flat scores matrix (root node included). Set <code>W</code> only if <code>topdown=GPAV</code> .
<code>parallel</code>	boolean value: <ul style="list-style-type: none"> • <code>TRUE</code>: execute the parallel implementation of GPAV (GPAV.parallel); • <code>FALSE</code> (def.): execute the sequential implementation of GPAV (GPAV.over.examples); Use <code>parallel</code> if and only if <code>topdown=GPAV</code> ; otherwise set <code>parallel=FALSE</code> .
<code>ncores</code>	number of cores to use for parallel execution (def. <code>8</code>). Set <code>ncores=1</code> if <code>parallel=FALSE</code> , otherwise set <code>ncores</code> to the desired number of cores. Use <code>ncores</code> if and only if <code>topdown=GPAV</code> ; otherwise set <code>parallel=1</code> .

Details

The *vanilla* TPR-DAG adopts a per-level bottom-up traversal of the DAG to correct the flat predictions \hat{y}_i :

$$\bar{y}_i := \frac{1}{1 + |\phi_i|} (\hat{y}_i + \sum_{j \in \phi_i} \bar{y}_j)$$

where ϕ_i are the positive children of i . Different strategies to select the positive children ϕ_i can be applied:

1. **Threshold-Free** strategy: the positive nodes are those children that can increment the score of the node i , that is those nodes that achieve a score higher than that of their parents:

$$\phi_i := \{j \in \text{child}(i) | \bar{y}_j > \hat{y}_i\}$$

2. **Threshold** strategy: the positive children are selected on the basis of a threshold that can be selected in two different ways:

- (a) for each node a constant threshold \bar{t} is a priori selected:

$$\phi_i := \{j \in \text{child}(i) | \bar{y}_j > \bar{t}\}$$

For instance if the predictions represent probabilities it could be meaningful to a priori select $\bar{t} = 0.5$.

- (b) the threshold is selected to maximize some performance metric \mathcal{M} estimated on the training data, as for instance the F-score or the AUPRC. In other words the threshold is selected to maximize some measure of accuracy of the predictions $\mathcal{M}(j, t)$ on the training data for the class j with respect to the threshold t . The corresponding set of positives $\forall i \in V$ is:

$$\phi_i := \{j \in \text{child}(i) | \bar{y}_j > t_j^*, t_j^* = \arg \max_t \mathcal{M}(j, t)\}$$

For instance t_j^* can be selected from a set of $t \in (0, 1)$ through internal cross-validation techniques.

The weighted TPR-DAG version can be designed by adding a weight $w \in [0, 1]$ to balance between the contribution of the node i and that of its positive children ϕ , through their convex combination:

$$\bar{y}_i := w \hat{y}_i + \frac{(1 - w)}{|\phi_i|} \sum_{j \in \phi_i} \bar{y}_j$$

If $w = 1$ no weight is attributed to the children and the TPR-DAG reduces to the HTD-DAG algorithm, since in this way only the prediction for node i is used in the bottom-up step of the algorithm. If $w = 0$ only the predictors associated to the children nodes vote to predict node i . In the intermediate cases we attribute more importance to the predictor for the node i or to its children depending on the values of w .

The contribution of the descendants of a given node decays exponentially with their distance from the node itself. To enhance the contribution of the most specific nodes to the overall decision of the ensemble we designed a novel variant that we named DESCENS. The novelty of DESCENS consists in strongly considering the contribution of all the descendants of each node instead of only that of its children. Therefore DESCENS predictions are more influenced by the information embedded in the leaves nodes, that are the classes containing the most informative and meaningful information

from a biological and medical standpoint. For the choice of the “positive” descendants we use the same strategies adopted for the selection of the “positive” children shown above. Furthermore, we designed a variant specific only for DESCENS, that we named DESCENS- τ . The DESCENS- τ variants balances the contribution between the “positives” children of a node i and that of its “positives” descendants excluding its children by adding a weight $\tau \in [0, 1]$:

$$\bar{y}_i := \frac{\tau}{1 + |\phi_i|}(\hat{y}_i + \sum_{j \in \phi_i} \bar{y}_j) + \frac{1 - \tau}{1 + |\delta_i|}(\hat{y}_i + \sum_{j \in \delta_i} \bar{y}_j)$$

where ϕ_i are the “positive” children of i and $\delta_i = \Delta_i \setminus \phi_i$ the descendants of i without its children. If $\tau = 1$ we consider only the contribution of the “positive” children of i ; if $\tau = 0$ only the descendants that are not children contribute to the score, while for intermediate values of τ we can balance the contribution of ϕ_i and δ_i positive nodes.

Simply by replacing the HTD ([HTD-DAG](#)) top-down step with the GPAV approach ([GPAV](#)) we can design the TPR-DAG variant ISO-TPR. The most important feature of ISO-TPR is that it maintains the hierarchical constraints by construction and selects the closest solution (in the least square sense) to the bottom-up predictions that obeys the true path rule. Obviously, any aforementioned strategy for the selection of “positive” children or descendants can be applied before executing the GPAV correction.

Value

a named matrix with the scores of the classes corrected according to the chosen algorithm.

See Also

[GPAV](#), [HTD-DAG](#)

Examples

```
data(graph);
data(scores);
data(labels);
root <- root.node(g);
S.hier <- TPR.DAG(S, g, root, positive="children", bottomup="threshold.free", topdown="HTD",
t=0, w=0, W=NULL, parallel=FALSE, ncores=1);
```

transitive.closure.annotations

Transitive closure of annotations

Description

Performs the transitive closure of the annotations using ancestors and the most specific annotation table. The annotations are propagated from bottom to top, enriching the most specific annotations table. The rows of the matrix correspond to the genes of the most specific annotation table and the columns to the OBO terms/classes.

Usage

```
transitive.closure.annotations(ann.spec, anc)
```

Arguments

- ann.spec the annotation matrix of the most specific annotations (0/1): rows are genes and columns are OBO terms.
- anc list of the ancestors of the ontology.

Value

an annotation table T: rows correspond to genes and columns to OBO terms. $T[i, j] = 1$ means that gene i is annotated for the term j , $T[i, j] = 0$ means that gene i is not annotated for the term j .

See Also

```
specific.annotation.matrix, build.ancestors
```

Examples

```
data(graph);
data(labels);
anc <- build.ancestors(g);
tca <- transitive.closure.annotations(L, anc);
```

tupla.matrix	<i>Tupla Matrix</i>
--------------	---------------------

Description

Transform a Weighted Adjacency Matrix (wadj matrix) of a graph in a tupla, i.e. as a sequences of rows separated by blank and the weight of the edges, e.g nodeX nodeY score.

Usage

```
tupla.matrix(m, output.file = "net.file.gz")
```

Arguments

- m a weighted adjacency matrix of the graph. Rows and columns are examples. It must be a square named matrix.
- output.file name of the file of the to be written. The extension of the file can be or plain format (".txt") or compressed (".gz").

Details

Only the *non-zero* interactions are kept, while the *zero* interactions are discarded. In other words in the output.file are reported only those nodes having a weight different from zero.

Value

the weighted adjacency matrix as `tupla` is stored in the `output.file`.

Examples

```
data(wadj);
tmpdir <- paste0(tempdir(), "/");
tupla.matrix(W, output.file=paste0(tmpdir, "graph.edges.txt.gz"));
tupla.matrix(W, output.file=paste0(tmpdir, "graph.edges.txt"));
```

`weighted.adjacency.matrix`

Weighted Adjacency Matrix

Description

Construct a Weighted Adjacency Matrix (`wadj` matrix) of a graph.

Usage

```
weighted.adjacency.matrix(file = "edges.txt")
```

Arguments

<code>file</code>	name of the plain text file to be read (def. <code>edges</code>). The format of the file is a sequence of rows. Each row corresponds to an edge represented through a pair of vertices separated by blanks and the weight of the edges. For instance: <code>nodeX nodeY score</code> . The file extension can be or plain format (<code>".txt"</code>) or compressed (<code>".gz"</code>).
-------------------	---

Value

a named symmetric weighted adjacency matrix of the graph.

Examples

```
edges <- system.file("extdata/edges.txt.gz", package="HEMDAG");
W <- weighted.adjacency.matrix(file=edges);
```

write.graph	<i>Write a directed graph on file</i>
-------------	---------------------------------------

Description

An object of class graphNEL is read and the graph is written on a plain text file as sequence of rows.

Usage

```
write.graph(g, file = "graph.txt.gz")
```

Arguments

g	a graph of class graphNEL.
file	name of the file to be written. The extension of the file can be or plain format (".txt") or compressed (".gz").

Value

a plain text file representing the graph. Each row corresponds to an edge represented through a pair of vertices separated by blanks.

Examples

```
data(graph);  
tmpdir <- paste0(tempdir(), "/");  
file <- paste0(tmpdir, "graph.edges.txt.gz");  
write.graph(g, file=file);
```

Index

*Topic **package**

- HEMDAG-package, 3
- adj.upper.tri, 4, 44
- ancestors, 5
- AUPRC, 6, 21, 23, 26, 29, 32, 35, 62, 63, 66, 67
- AUROC, 7, 21, 23, 26, 29, 32, 35, 63, 67
- build.ancestors, 10, 42, 72
- build.ancestors(ancestors), 5
- build.children(children), 11
- build.consistent.graph, 9
- build.descendants(descendants), 14
- check.annotation.matrix.integrity, 9
- check.DAG.integrity, 10
- check.hierarchy
 - (hierarchical.checkers), 47
- children, 11
- compute.flipped.graph, 12
- compute.Fmeasure.multilabel(FMM), 40
- constraints.matrix, 12
- create.stratified.fold.df, 13
- descendants, 14
- distances.from.leaves, 15
- do.edges.from.HPO.obo, 15
- Do.flat.scores.normalization, 16
- Do.full.annotation.matrix, 17
- Do.GPAV, 18
- Do.GPAV.holdout, 21
- Do.heuristic.methods, 24
- Do.heuristic.methods.holdout, 27
- Do.HTD, 30
- Do.HTD.holdout, 33
- do.stratified.cv.data.over.classes
 - (stratified.cross.validation), 58
- do.stratified.cv.data.single.class
 - (stratified.cross.validation), 58
- do.subgraph, 35
- do.submatrix, 36
- Do.TPR.DAG(TPR-DAG-cross-validation), 60
- Do.TPR.DAG.holdout(TPR-DAG-holdout), 64
- do.unstratified.cv.data, 37
- example.datasets, 37
- F.measure.multilabel
 - (Multilabel.F.measure), 50
- F.measure.multilabel,matrix,matrix-method
 - (Multilabel.F.measure), 50
- find.best.f, 38
- find.leaves, 40
- FMM, 21, 23, 26, 29, 32, 35, 40, 63, 67
- full.annotation.matrix, 10, 18, 42
- g(example.datasets), 37
- get.children.bottom.up(children), 11
- get.children.top.down(children), 11
- get.parents(parents), 52
- GPAV, 4, 21, 24, 43, 61, 66, 69, 71
- GPAV.over.examples, 19, 22, 44, 61, 66, 69
- GPAV.parallel, 19, 22, 44, 45, 61, 66, 69
- graph.levels, 6, 11, 14, 46, 49, 53
- HEMDAG(HEMDAG-package), 3
- HEMDAG-package, 3
- Heuristic-Methods, 46
- heuristic.and(Heuristic-Methods), 46
- heuristic.max(Heuristic-Methods), 46
- heuristic.or(Heuristic-Methods), 46
- hierarchical.checkers, 47, 49
- htd(HTD-DAG), 48
- HTD-DAG, 48
- L(example.datasets), 37
- lexicographical.topological.sort, 49
- Multilabel.F.measure, 50, 62, 63, 66, 67

`normalize.max`, 51

`parents`, 52

`precision.at.all.recall.levels.single.class`
(PXR), 53

`precision.at.given.recall.levels.over.classes`
(PXR), 53

PXR, 21, 23, 26, 29, 32, 35, 53, 63, 67

`read.graph`, 54

`read.undirected.graph`, 55

`root.node`, 56

`S` (example.datasets), 37

`scores.normalization`, 56

`specific.annotation.list`, 57

`specific.annotation.matrix`, 42, 57, 58,
72

`stratified.cross.validation`, 58

`test.index` (example.datasets), 37

TPR-DAG-cross-validation, 60

TPR-DAG-holdout, 64

TPR-DAG-variants, 68

TPR.DAG (TPR-DAG-variants), 68

`transitive.closure.annotations`, 10, 42,
71

`tupla.matrix`, 72

`W` (example.datasets), 37

`weighted.adjacency.matrix`, 42, 73

`write.graph`, 74