

# Package ‘HEMDAG’

October 16, 2017

**Title** Hierarchical Ensemble Methods for Directed Acyclic Graphs

**Version** 1.1.1

**Author** Marco Notaro [aut, cre] and Giorgio Valentini [aut]  
(AnacletoLab, Dipartimento di Informatica, Universita' degli Studi di Milano)

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

**Description** An implementation of Hierarchical Ensemble Methods for Directed Acyclic Graphs (DAGs). The 'HEMDAG' package can be used to enhance the predictions of virtually any flat learning method, by taking into account the hierarchical nature of the classes of a bio-ontology. 'HEMDAG' 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 it can be also safely applied to tree-structured taxonomies (as FunCat), since trees are DAGs. 'HEMDAG' scale nicely both in terms of the complexity of the taxonomy and in the cardinality of the examples. (Marco Notaro, Max Schubach, Peter N. Robinson and Giorgio Valentini (2017) <doi:10.1186/s12859-017-1854-y>).

**Depends** R (>= 2.10)

**License** GPL (>= 3)

**Encoding** UTF-8

**Repository** CRAN

**LazyLoad** true

**NeedsCompilation** no

**Imports** graph,  
RBGL,  
PerfMeas,  
precrec,  
preprocessCore,  
methods

**Suggests** Rgraphviz

**RoxygenNote** 6.0.1

## R topics documented:

HEMDAG-package . . . . . 3

ancestors	4
AUPRC.single.class	5
AUPRC.single.over.classes	5
AUROC.single.class	6
AUROC.single.over.classes	7
check.annotation.matrix.integrity	8
check.DAG.integrity	9
children	9
compute.flipped.graph	10
constraints.matrix	11
descendants	11
DESCENS	12
DESCENS.CV	14
DESCENS.HOLDOUT	17
distances.from.leaves	20
Do.descens.threshold.free	21
Do.descens.threshold.free.holdout	23
do.edges.from.HPO.obo	25
Do.FLAT.scores.normalization	26
Do.full.annotation.matrix	27
Do.heuristic.methods	28
Do.heuristic.methods.holdout	30
Do.HTD	33
Do.HTD.holdout	35
do.subgraph	37
do.submatrix	38
Do.tpr.threshold.free	38
Do.tpr.threshold.free.holdout	40
do.unstratified.cv.data	43
example.datasets	43
find.best.f	44
find.leaves	46
full.annotation.matrix	46
graph.levels	47
Heuristic-Methods	48
hierarchical.checkers	49
HTD-DAG	50
Multilabel.F.measure	51
normalize.max	52
parents	53
read.graph	54
read.undirected.graph	54
root.node	55
specific.annotation.list	56
specific.annotation.matrix	56
stratified.cross.validation	57
TPR-DAG	58
TPR.DAG.CV	60

<i>HEMDAG-package</i>	3
TPR.DAG.HOLDOUT . . . . .	63
transitive.closure.annotations . . . . .	66
tupla.matrix . . . . .	67
weighted.adjacency.matrix . . . . .	68
write.graph . . . . .	69
<b>Index</b>	<b>70</b>

---

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. HEMDAG can be used to enhance the predictions of virtually any flat learning method, by taking into account the hierarchical nature of the classes of a bio-ontology. HEMDAG 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 it can be also safely applied to tree-structured taxonomies (as FunCat), since trees are DAGs. HEMDAG scale nicely both in terms of the complexity of the taxonomy and in the cardinality of the examples.

### Details

The HEMDAG package provides many utility functions to handle graph data structures and implements several Hierarchical Ensemble Methods for DAGs:

1. **HTD-DAG**: Hierarchical Top Down ([HTD-DAG](#));
2. **TPR-DAG**: True-Path Rule ([TPR-DAG](#));
3. **DESCENS**: Descendants Ensemble Classifier ([DESCENS](#));
4. **MAX, AND, OR**: Heuristic Methods, *Obozinski et al.* ([Heuristic-Methods](#));

### Author(s)

*Marco Notaro* and *Giorgio Valentini*, [AnacletoLab](#), DI, Dipartimento di Informatica, Universita' degli Studi di Milano  
Maintainer: *Marco Notaro* <[marco.notaro@unimi.it](mailto:marco.notaro@unimi.it)>

### 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)

ancestors

*Build ancestors***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.single.class	<i>AUPRC single class</i>
--------------------	---------------------------

---

**Description**

High-level function to compute the Area under the Precision Recall Curve (AUPRC) just for a single class through **precrec** package

**Usage**

```
AUPRC.single.class(target, pred)
```

**Arguments**

target	vector of the true labels (0 negative, 1 positive examples)
pred	numeric vector of the values of the predicted labels (scores)

**Value**

a numeric value corresponding to the AUPRC for the considered class

**See Also**

[AUPRC.single.over.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 <- AUPRC.single.class(L[,3],S[,3]);
```

---

AUPRC.single.over.classes	<i>AUPRC over classes</i>
---------------------------	---------------------------

---

**Description**

High-level function to compute the Area under the Precision Recall Curve (AUPRC) across a set of classes through **precrec** package

**Usage**

```
AUPRC.single.over.classes(target, pred)
```

**Arguments**

target	matrix with the target multilabels: rows correspond to examples and columns to classes. $target[i, j] = 1$ if example $i$ belongs to class $j$ , $target[i, j] = 0$ otherwise.
pred	a numeric matrix with predicted values (scores): rows correspond to examples and columns to classes.

**Value**

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

**See Also**

[AUPRC.single.class](#)

**Examples**

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

---

AUROC.single.class	<i>AUROC single class</i>
--------------------	---------------------------

---

**Description**

High-level function to compute the Area under the ROC Curve (AUPRC) just for a single class through **precrec** package

**Usage**

```
AUROC.single.class(target, pred)
```

**Arguments**

target	vector of the true labels (0 negative, 1 positive examples)
pred	numeric vector of the values of the predicted labels (scores)

**Value**

a numeric value corresponding to the AUROC for the considered class

**See Also**

[AUROC.single.over.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 <- AUROC.single.class(L[,3],S[,3]);
```

---

AUROC.single.over.classes

*AUROC over classes*

---

**Description**

High-level function to compute the Area under the ROC Curve (AUROC) for a set of classes through **precrec** package

**Usage**

```
AUROC.single.over.classes(target, pred)
```

**Arguments**

target	matrix with the target multilabels: rows correspond to examples and columns to classes. $target[i, j] = 1$ if example $i$ belongs to class $j$ , $target[i, j] = 0$ otherwise.
pred	a numeric matrix with predicted values (scores): rows correspond to examples and columns to classes.

**Value**

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

**See Also**

[AUROC.single.class](#)

## Examples

```
data(labels);
data(scores);
data(graph);
root <- root.node(g);
L <- L[,~which(colnames(L)==root)];
S <- S[,~which(colnames(S)==root)];
AUC <- AUROC.single.over.classes(L,S);
```

---

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, hpo.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.
hpo.ann	the full annotations 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	<i>DAG checker</i>
---------------------	--------------------

---

**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:"00")

**Value**

If there are nodes not accessible from the root "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	<i>Build children</i>
----------	-----------------------

---

**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**

g	a graph of class graphNEL. It represents the hierarchy of the classes
levels	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	<i>Constraints Matrix</i>
--------------------	---------------------------

---

**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);
```

---

descendants	<i>Build descendants</i>
-------------	--------------------------

---

**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**

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

`levels` 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);
```

---

DESCENS

---

*DESCENS variants*


---

**Description**

The novelty of DESCENS with respect to TPR-DAG algorithm consists in considering the contribution of all the descendants of each node instead of only that of its children, since with the TPR-DAG algorithm the contribution of the descendants of a given node decays exponentially with their distance from the node itself, reducing the impact of the predictions made at the most specific levels of the ontology. On the contrary DESCENS predictions are more influenced by the information embedded in the most specific terms of the taxonomy (e.g. leaf nodes), thus putting more emphasis on the terms that most characterize the gene under study.

**Usage**

```
descens.threshold(S, g, root = "00", t = 0.5)

descens.threshold.free(S, g, root = "00")

descens.weighted.threshold.free(S, g, root = "00", w = 0.5)

descens.weighted.threshold(S, g, root = "00", t = 0.5, w = 0.5)

descens.tau(S, g, tau = 0.5, 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 on the top-level of the hierarchy (def. root="00")
t	threshold for the choice of positive children (def. t=0.5)
w	weight to balance between the contribution of the node $i$ and that of its positive children
tau	weight $\tau$ to balance the contribution between the positives children of a node $i$ and that of its positives descendants excluding the positives children

### Details

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

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

where  $\Delta_i$  are the positive descendants of  $i$ . Different strategies to select the positive descendants  $\Delta_i$  can be applied:

1. **Threshold-Free** strategy: as positive descendants we choose those nodes that achieve a score higher than that of their ancestor node  $i$ :

$$\Delta_i := \{j \in \text{descendants}(i) \mid \bar{y}_j > \hat{y}_i\}$$

2. **Threshold** strategy: the positive descendants 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:

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

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:

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

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

The weighted DESCENS variants can be simply designed by adding a weight  $w \in [0, 1]$  to balance the contribution between the prediction of the classifier associated with the node  $i$  and that of its positive descendants:

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

The DESCENS- $\tau$  variants balances the contribution between the positives children of a node  $i$  and that of its positives descendants excluding the 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  $\phi_i$  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  $\tau$  we can balance the contribution of  $\phi_i$  and  $\delta_i$  positive nodes.

### Value

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

### Examples

```
data(graph);
data(scores);
data(labels);
root <- root.node(g);
S.descensTF <- descens.threshold.free(S,g,root);
S.descensT <- descens.threshold(S,g,root,t=0.5);
S.descensW <- descens.weighted.threshold.free(S,g,root,w=0.5);
S.descensWT <- descens.weighted.threshold(S,g,root,w=0.5, t=0.5);
S.descensTAU <- descens.tau(S,g,root, tau=0.5);
```

---

DESCENS.CV

*DESCENS cross-validation*

---

### Description

High level function to correct the computed scores in a hierarchy according to one of the DESCENS variants performing cross-validation experiments

### Usage

```
Do.descens.threshold.cv(threshold = seq(from = 0.1, to = 0.9, by = 0.1),
  kk = 5, seed = NULL, norm = TRUE, norm.type = "NONE",
  flat.file = flat.file, ann.file = ann.file, dag.file = dag.file,
  flat.dir = flat.dir, ann.dir = ann.dir, dag.dir = dag.dir,
  flat.norm.dir = NULL, n.round = 3, f.criterion = "F",
  hierScore.dir = hierScore.dir, perf.dir = perf.dir)

Do.descens.weighted.threshold.free.cv(weight = seq(from = 0.1, to = 0.9, by =
  0.1), kk = 5, seed = NULL, norm = TRUE, norm.type = "NONE",
  flat.file = flat.file, ann.file = ann.file, dag.file = dag.file,
  flat.dir = flat.dir, ann.dir = ann.dir, dag.dir = dag.dir,
```

```
flat.norm.dir = NULL, n.round = 3, f.criterion = "F",
hierScore.dir = hierScore.dir, perf.dir = perf.dir)
```

```
Do.descens.tau.cv(tau = seq(from = 0.1, to = 0.9, by = 0.1), kk = 5,
  seed = NULL, norm = TRUE, norm.type = "NONE", n.round = 3,
  f.criterion = "F", flat.file = flat.file, ann.file = ann.file,
  dag.file = dag.file, flat.dir = flat.dir, ann.dir = ann.dir,
  dag.dir = dag.dir, flat.norm.dir = NULL, hierScore.dir = hierScore.dir,
  perf.dir = perf.dir)
```

```
Do.descens.weighted.threshold.cv(threshold = seq(from = 0.1, to = 0.9, by =
  0.1), weight = seq(from = 0.1, to = 1, by = 0.1), kk = 5, seed = NULL,
  norm = TRUE, norm.type = "NONE", flat.file = flat.file,
  ann.file = ann.file, dag.file = dag.file, flat.dir = flat.dir,
  ann.dir = ann.dir, dag.dir = dag.dir, flat.norm.dir = NULL,
  n.round = 3, f.criterion = "F", 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 increasing.
kk	number of folds of the cross validation (def: kk=5);
seed	initialization seed for the random generator to create folds. If NULL (def.) no initialization is performed
norm	boolean value: <ul style="list-style-type: none"> <li>• TRUE (def.): the flat scores matrix has been already normalized in according to a normalization method;</li> <li>• FALSE: the flat scores matrix has not been normalized yet. See the parameter norm.type for which normalization can be applied.</li> </ul>
norm.type	can be one of the following three values: <ol style="list-style-type: none"> <li>1. NONE (def.): set norm.type to NONE if and only if the parameter norm is set to TRUE;</li> <li>2. MaxNorm: each score is divided w.r.t. the max of each class;</li> <li>3. Qnorm: quantile normalization. <b>preprocessCore</b> package is used.</li> </ol>
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 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

<code>flat.norm.dir</code>	relative path where flat normalized scores matrix must be stored. Use this parameter if and only if <code>norm</code> is set to <code>FALSE</code> , otherwise set <code>flat.norm.dir</code> to <code>NULL</code> (def.)
<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
<code>f.criterion</code>	character. Type of F-measure to be used to select the best F-measure. Two possibilities: <ol style="list-style-type: none"> <li>1. <code>F</code> (def.): corresponds to the harmonic mean between the average precision and recall</li> <li>2. <code>avF</code>: corresponds to the per-example F-score averaged across all the examples</li> </ol>
<code>hierScore.dir</code>	relative path where the hierarchical scores matrix must be stored
<code>perf.dir</code>	relative path where the term-centric and protein-centric measures must be stored
<code>weight</code>	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 weight is, but obviously the execution time will be increasing.
<code>tau</code>	range of tau values to be tested in order to find the best tau (def: from:0.1, to:0.9, by:0.1). The denser the range is, the higher the probability to find the best weight is, but obviously the execution time will be increasing.

## Details

These high level functions perform a classical cross-validation procedure to find the best threshold maximizing on F-score measure.

## Value

Five rda files stored in the respective output directories:

1. `hierarchical scores matrix`: a matrix with examples on rows and classes on columns representing the computed hierarchical scores for each example and for each considered class. This file is stored in `hierScore.dir` directory.
2. `FMM (F-Measure Multilabel) results`: F-score computed by `find.best.f` function. Both *flat* and *hierarchical* results are reported. This file is stored in `perf.dir` directory.
3. `PRC (area under Precision-Recall Curve) results`: PRC computed by **precrec** package. Both *flat* and *hierarchical* results are reported. This file is stored in `perf.dir` directory.
4. `AUC (Area Under ROC Curve) results`: AUC computed by **precrec** package. Both *flat* and *hierarchical* results are reported. This file is stored in `perf.dir` directory.
5. `PXR (Precision at fixed Recall levels) average and per classes`: PXR computed by **PerfMeas** package. It is stored in `perf.dir` directory.

## See Also

[DESCENS](#)



**Examples**

```

data(graph);
data(scores);
data(labels);
if (!dir.exists("data")){
  dir.create("data");
}
if (!dir.exists("results")){
  dir.create("results");
}
save(g,file="data/graph.rda");
save(L,file="data/labels.rda");
save(S,file="data/scores.rda");
dag.dir <- flat.dir <- flat.norm.dir <- ann.dir <- "data/";
hierScore.dir <- perf.dir <- "results/";
dag.file <- "graph";
flat.file <- "scores";
ann.file <- "labels";
threshold <- seq(from=0.1, to=0.5, by=0.2);
Do.descens.threshold.cv(threshold=threshold, kk=5, seed=23, norm=FALSE, norm.type= "MaxNorm",
  flat.file=flat.file, ann.file=ann.file, dag.file=dag.file, flat.dir=flat.dir, ann.dir=ann.dir,
  dag.dir=dag.dir, flat.norm.dir=flat.norm.dir, n.round=3, f.criterion = "F",
  hierScore.dir=hierScore.dir, perf.dir=perf.dir);

```

DESCENS.HOLDOUT

*DESCENS holdout***Description**

High level function to correct the computed scores in a hierarchy according to one of the DESCENS variants performing a classical holdout procedure.

**Usage**

```

Do.descens.threshold.holdout(threshold = seq(from = 0.1, to = 0.9, by = 0.1),
  kk = 5, seed = NULL, norm = TRUE, norm.type = "NONE",
  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, flat.norm.dir = NULL,
  n.round = 3, f.criterion = "F", hierScore.dir = hierScore.dir,
  perf.dir = perf.dir)

```

```

Do.descens.weighted.threshold.free.holdout(weight = seq(from = 0.1, to = 1, by
  = 0.1), kk = 5, seed = NULL, norm = TRUE, norm.type = "NONE",
  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, flat.norm.dir = NULL,
  n.round = 3, f.criterion = "F", hierScore.dir = hierScore.dir,

```

```
perf.dir = perf.dir)
```

```
Do.descens.tau.holdout(tau = seq(from = 0.1, to = 0.9, by = 0.1), kk = 5,
  seed = NULL, norm = TRUE, norm.type = "NONE", 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, flat.norm.dir = NULL, n.round = 3,
  f.criterion = "F", hierScore.dir = hierScore.dir, perf.dir = perf.dir)
```

```
Do.descens.weighted.threshold.holdout(threshold = seq(from = 0.1, to = 0.9, by
  = 0.1), weight = seq(from = 0.1, to = 1, by = 0.1), kk = 5, seed = NULL,
  norm = TRUE, norm.type = "NONE", 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, flat.norm.dir = NULL, n.round = 3,
  f.criterion = "F", 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 increasing.
kk	number of folds of the cross validation (def: kk=5);
seed	initialization seed for the random generator to create folds. If NULL (def.) no initialization is performed
norm	boolean value: <ul style="list-style-type: none"> <li>• TRUE (def.): the flat scores matrix has been already normalized in according to a normalization method;</li> <li>• FALSE: the flat scores matrix has not been normalized yet. See the parameter norm.type for which normalization can be applied.</li> </ul>
norm.type	can be one of the following three values: <ol style="list-style-type: none"> <li>1. NONE (def.): set norm.type to NONE if and only if the parameter norm is set to TRUE;</li> <li>2. MaxNorm: each score is divided w.r.t. the max of each class;</li> <li>3. Qnorm: quantile normalization. <b>preprocessCore</b> package is used.</li> </ol>
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 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)
ind.test.set	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
ind.dir	relative path to folder where ind.test.set is stored
flat.dir	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>flat.norm.dir</code>	relative path where flat normalized scores matrix must be stored. Use this parameter if and only if <code>norm</code> is set to <code>FALSE</code> , otherwise set <code>flat.norm.dir</code> to <code>NULL</code> (def.)
<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
<code>f.criterion</code>	character. Type of F-measure to be used to select the best F-measure. Two possibilities: <ol style="list-style-type: none"> <li>1. <code>F</code> (def.): corresponds to the harmonic mean between the average precision and recall</li> <li>2. <code>avF</code>: corresponds to the per-example F-score averaged across all the examples</li> </ol>
<code>hierScore.dir</code>	relative path where the hierarchical scores matrix must be stored
<code>perf.dir</code>	relative path where the term-centric and protein-centric measures must be stored
<code>weight</code>	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 weight is, but obviously the execution time will be increasing.
<code>tau</code>	range of tau values to be tested in order to find the best tau (def: from:0.1, to:0.9, by:0.1). The denser the range is, the higher the probability to find the best weight is, but obviously the execution time will be increasing.

## Details

These high level functions perform a classical holdout procedure to find the best threshold maximizing on F.score measure.

## Value

Five rda files stored in the respective output directories:

1. `hierarchical scores matrix`: a matrix with examples on rows and classes on columns representing the computed hierarchical scores for each example and for each considered class. This file is stored in `hierScore.dir` directory.
2. `FMM (F-Measure Multilabel) results`: F-score computed by `find.best.f` function. Both *flat* and *hierarchical* results are reported. This file is stored in `perf.dir` directory.
3. `PRC (area under Precision-Recall Curve) results`: PRC computed by **precrec** package. Both *flat* and *hierarchical* results are reported. This file is stored in `perf.dir` directory.
4. `AUC (Area Under ROC Curve) results`: AUC computed by **precrec** package. Both *flat* and *hierarchical* results are reported. This file is stored in `perf.dir` directory.
5. `PXR (Precision at fixed Recall levels) average and per classes`: PXR computed by **PerfMeas** package. It is stored in `perf.dir` directory.

## See Also

[DESCENS](#)

**Examples**

```

data(graph);
data(scores);
data(labels);
data(test.index);
if (!dir.exists("data")){
  dir.create("data");
}
if (!dir.exists("results")){
  dir.create("results");
}
save(g,file="data/graph.rda");
save(L,file="data/labels.rda");
save(S,file="data/scores.rda");
save(test.index, file="data/test.index.rda");
ind.dir <- dag.dir <- flat.dir <- flat.norm.dir <- ann.dir <- "data/";
hierScore.dir <- perf.dir <- "results/";
ind.test.set <- "test.index";
dag.file <- "graph";
flat.file <- "scores";
ann.file <- "labels";
threshold <- seq(from=0.1, to=0.3, by=0.1);
Do.descens.threshold.holdout(threshold=threshold, kk=5, seed=23, norm=FALSE, norm.type="MaxNorm",
  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, flat.norm.dir=flat.norm.dir,
  n.round=3, f.criterion="F", hierScore.dir=hierScore.dir, perf.dir=perf.dir);

```

---

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.descens.threshold.free

*DESCENS vanilla*


---

## Description

High level function to correct the computed scores in a hierarchy according to the DESCENS vanilla algorithm

## Usage

```
Do.descens.threshold.free(norm = TRUE, norm.type = "NONE",
  flat.file = flat.file, ann.file = ann.file, dag.file = dag.file,
  flat.dir = flat.dir, ann.dir = ann.dir, dag.dir = dag.dir,
  flat.norm.dir = NULL, n.round = 3, f.criterion = "F",
  hierScore.dir = hierScore.dir, perf.dir = perf.dir)
```

## Arguments

norm	boolean value: <ul style="list-style-type: none"> <li>• TRUE (def.): the flat scores matrix has been already normalized in according to a normalization method;</li> <li>• FALSE: the flat scores matrix has not been normalized yet. See the parameter norm.type for which normalization can be applied.</li> </ul>
norm.type	can be one of the following three values: <ol style="list-style-type: none"> <li>1. NONE (def.): set norm.type to NONE if and only if the parameter norm is set to TRUE;</li> <li>2. MaxNorm: each score is divided w.r.t. the max of each class;</li> <li>3. Qnorm: quantile normalization. <b>preprocessCore</b> package is used.</li> </ol>
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 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

<code>flat.norm.dir</code>	relative path where flat normalized scores matrix must be stored. Use this parameter if and only if <code>norm</code> is set to <code>FALSE</code> , otherwise set <code>flat.norm.dir</code> to <code>NULL</code> (def.)
<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
<code>f.criterion</code>	character. Type of F-measure to be used to select the best F-measure. Two possibilities: <ol style="list-style-type: none"> <li>1. <code>F</code> (def.): corresponds to the harmonic mean between the average precision and recall</li> <li>2. <code>avF</code>: corresponds to the per-example F-score averaged across all the examples</li> </ol>
<code>hierScore.dir</code>	relative path where the hierarchical scores matrix must be stored
<code>perf.dir</code>	relative path where the term-centric and protein-centric measures must be stored

### Value

Five rda files stored in the respective output directories:

1. `hierarchical_scores_matrix`: a matrix with examples on rows and classes on columns representing the computed hierarchical scores for each example and for each considered class. This file is stored in `hierScore.dir` directory.
2. `FMM` (F-Measure Multilabel) results: F-score computed by `find.best.f` function. Both *flat* and *hierarchical* results are reported. This file is stored in `perf.dir` directory.
3. `PRC` (area under Precision-Recall Curve) results: PRC computed by **precrec** package. Both *flat* and *hierarchical* results are reported. This file is stored in `perf.dir` directory.
4. `AUC` (Area Under ROC Curve) results: AUC computed by **precrec** package. Both *flat* and *hierarchical* results are reported. This file is stored in `perf.dir` directory.
5. `PXR` (Precision at fixed Recall levels) average and per classes: PXR computed by **PerfMeas** package. It is stored in `perf.dir` directory.

### See Also

[DESCENS](#)

### Examples

```
data(graph);
data(scores);
data(labels);
if (!dir.exists("data")){
  dir.create("data");
}
if (!dir.exists("results")){
  dir.create("results");
}
save(g,file="data/graph.rda");
save(L,file="data/labels.rda");
save(S,file="data/scores.rda");
```

```

dag.dir <- flat.dir <- flat.norm.dir <- ann.dir <- "data/";
hierScore.dir <- perf.dir <- "results/";
dag.file <- "graph";
flat.file <- "scores";
ann.file <- "labels";
Do.descens.threshold.free(norm=FALSE, norm.type= "MaxNorm", flat.file=flat.file,
ann.file=ann.file, dag.file=dag.file, flat.dir=flat.dir, ann.dir=ann.dir,
dag.dir=dag.dir, flat.norm.dir=flat.norm.dir, n.round=3, f.criterion ="F",
hierScore.dir=hierScore.dir, perf.dir=perf.dir);

```

---

Do.descens.threshold.free.holdout

*DESCENS vanilla holdout*


---

## Description

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

## Usage

```

Do.descens.threshold.free.holdout(norm = TRUE, norm.type = "NONE",
  flat.file = flat.file, ann.file = ann.file, dag.file = dag.file,
  flat.dir = flat.dir, ann.dir = ann.dir, dag.dir = dag.dir,
  flat.norm.dir = NULL, ind.test.set = ind.test.set, ind.dir = ind.dir,
  n.round = 3, f.criterion = "F", hierScore.dir = hierScore.dir,
  perf.dir = perf.dir)

```

## Arguments

norm	boolean value: <ul style="list-style-type: none"> <li>TRUE (def.): the flat scores matrix has been already normalized in according to a normalization method;</li> <li>FALSE: the flat scores matrix has not been normalized yet. See the parameter norm for which normalization can be applied.</li> </ul>
norm.type	can be one of the following three values: <ol style="list-style-type: none"> <li>NONE (def.): set norm.type to NONE if and only if the parameter norm is set to TRUE;</li> <li>MaxNorm: each score is divided w.r.t. the max of each class;</li> <li>Qnorm: quantile normalization. <b>preprocessCore</b> package is used.</li> </ol>
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 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)

<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>flat.norm.dir</code>	relative path where flat normalized scores matrix must be stored. Use this parameter if and only if <code>norm</code> is set to <code>FALSE</code> , otherwise set <code>flat.norm.dir</code> to <code>NULL</code> (def.)
<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>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
<code>f.criterion</code>	character. Type of F-measure to be used to select the best F-measure. Two possibilities: <ol style="list-style-type: none"> <li>1. F (def.): corresponds to the harmonic mean between the average precision and recall</li> <li>2. avF: corresponds to the per-example F-score averaged across all the examples</li> </ol>
<code>hierScore.dir</code>	relative path where the hierarchical scores matrix must be stored
<code>perf.dir</code>	relative path where the term-centric and protein-centric measures must be stored

## Value

Five rda files stored in the respective output directories:

1. `hierarchical scores matrix`: a matrix with examples on rows and classes on columns representing the computed hierarchical scores for each example and for each considered class. This file is stored in `hierScore.dir` directory.
2. `FMM (F-Measure Multilabel) results`: F-score computed by `find.best.f` function. Both *flat* and *hierarchical* results are reported. This file is stored in `perf.dir` directory.
3. `PRC (area under Precision-Recall Curve) results`: PRC computed by **precrec** package. Both *flat* and *hierarchical* results are reported. This file is stored in `perf.dir` directory.
4. `AUC (Area Under ROC Curve) results`: AUC computed by **precrec** package. Both *flat* and *hierarchical* results are reported. This file is stored in `perf.dir` directory.
5. `PXR (Precision at fixed Recall levels) average and per classes`: PXR computed by **PerfMeas** package. It is stored in `perf.dir` directory.

## See Also

[DESCENS](#)

## Examples

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



```

data(test.index);
if (!dir.exists("data")){
  dir.create("data");
}
if (!dir.exists("results")){
  dir.create("results");
}
save(g,file="data/graph.rda");
save(L,file="data/labels.rda");
save(S,file="data/scores.rda");
save(test.index, file="data/test.index.rda");
ind.dir <- dag.dir <- flat.dir <- flat.norm.dir <- ann.dir <- "data/";
hierScore.dir <- perf.dir <- "results/";
ind.test.set <- "test.index";
dag.file <- "graph";
flat.file <- "scores";
ann.file <- "labels";
Do.descens.threshold.free.holdout(norm=FALSE, norm.type= "MaxNorm", 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, flat.norm.dir=flat.norm.dir,
n.round=3, f.criterion ="F", hierScore.dir=hierScore.dir, perf.dir=perf.dir);

```

---

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(file = "hp.obo", output.file = "edge.file")
```

## Arguments

file	an HPO OBO file
output.file	name of the file of the edges to be written

## 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(file=hpobo, output.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,
  dag.file = dag.file, flat.dir = flat.dir, dag.dir = dag.dir,
  flat.norm.dir = flat.norm.dir)
```

## Arguments

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

## Details

To apply the quantile normalization the **preprocessCore** library is used.

## Value

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

## Examples

```
data(scores);
data(graph);
if (!dir.exists("data")){
  dir.create("data");
}
if (!dir.exists("results")){
  dir.create("results");
}
save(S, file="data/scores.rda");
```

```

save(g,file="data/graph.rda");
flat.dir <- dag.dir <- "data/";
flat.norm.dir <- "results/";
flat.file <- "scores";
dag.file <- "graph";
norm.types <- c("MaxNorm","Qnorm");
for(norm.type in norm.types){
Do.FLAT.scores.normalization(norm.type=norm.type, flat.file=flat.file,
dag.file=dag.file, flat.dir=flat.dir, dag.dir=dag.dir,
flat.norm.dir=flat.norm.dir);
}

```

---

Do.full.annotation.matrix

*Do full annotations 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 containg 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)
ann.dir	relative path to directory where the matrix of the most specific annotation is stored
output.name	name of the output file without rda extension (without rda extension)
output.dir	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);
if (!dir.exists("data")){
  dir.create("data");
}
if (!dir.exists("results")){
  dir.create("results");
}
anc <- build.ancestors(g);
save(anc,file="data/ancestors.rda");
save(g,file="data/graph.rda");
save(L,file="data/labels.rda");
save(W,file="data/wadj.rda");
anc.dir <- net.dir <- ann.dir <- "data/";
output.dir <- "results/";
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);
```

---

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 = heuristic.fun, norm = TRUE,
  norm.type = "NONE", flat.file = flat.file, ann.file = ann.file,
  dag.file = dag.file, flat.dir = flat.dir, ann.dir = ann.dir,
  dag.dir = dag.dir, flat.norm.dir = NULL, n.round = 3,
  f.criterion = "F", hierScore.dir = hierScore.dir, perf.dir = perf.dir)
```

**Arguments**

<code>heuristic.fun</code>	can be one of the following three values: <ol style="list-style-type: none"> <li>1. <code>heuristicMAX</code>: run the MAX heuristic method;</li> <li>2. <code>heuristicAND</code>: run the AND heuristic method;</li> <li>3. <code>heuristicOR</code>: run the OR heuristic method;</li> </ol>
<code>norm</code>	boolean value: <ul style="list-style-type: none"> <li>• <code>TRUE</code> (def.): the flat scores matrix has been already normalized in according to a normalization method;</li> <li>• <code>FALSE</code>: the flat scores matrix has not been normalized yet. See the parameter <code>norm.type</code> for which normalization can be applied.</li> </ul>
<code>norm.type</code>	can be one of the following three values: <ol style="list-style-type: none"> <li>1. <code>heuristicMAX</code>: run the heuristic method MAX;</li> <li>2. <code>heuristicAND</code>: run the heuristic method AND;</li> <li>3. <code>heuristicOR</code>: run the heuristic method OR;</li> </ol>
<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 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>flat.norm.dir</code>	relative path where flat normalized scores matrix must be stored. Use this parameter if and only if <code>norm</code> is set to <code>FALSE</code> , otherwise set <code>flat.norm.dir</code> to <code>NULL</code> (def.)
<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
<code>f.criterion</code>	character. Type of F-measure to be used to select the best F-measure. Two possibilities: <ol style="list-style-type: none"> <li>1. <code>F</code> (def.): corresponds to the harmonic mean between the average precision and recall</li> <li>2. <code>avF</code>: corresponds to the per-example F-score averaged across all the examples</li> </ol>
<code>hierScore.dir</code>	relative path where the hierarchical scores matrix must be stored
<code>perf.dir</code>	relative path where the term-centric and protein-centric measures must be stored

**Value**

Five rda files stored in the respective output directories:

1. `hierarchical scores matrix`: a matrix with examples on rows and classes on columns representing the computed hierarchical scores for each example and for each considered class. This file is stored in `hierScore.dir` directory.
2. `FMM (F-Measure Multilabel) results`: F-score computed by `find.best.f` function. Both *flat* and *hierarchical* results are reported. This file is stored in `perf.dir` directory.
3. `PRC (area under Precision-Recall Curve) results`: PRC computed by **precrec** package. Both *flat* and *hierarchical* results are reported. This file is stored in `perf.dir` directory.
4. `AUC (Area Under ROC Curve) results`: AUC computed by **precrec** package. Both *flat* and *hierarchical* results are reported. This file is stored in `perf.dir` directory.
5. `PXR (Precision at fixed Recall levels) average and per classes`: PXR computed by **PerfMeas** package. It is stored in `perf.dir` directory.

### See Also

[Heuristic-Methods](#)

### Examples

```
data(graph);
data(scores);
data(labels);
if (!dir.exists("data")){
  dir.create("data");
}
if (!dir.exists("results")){
  dir.create("results");
}
save(g,file="data/graph.rda");
save(L,file="data/labels.rda");
save(S,file="data/scores.rda");
dag.dir <- flat.dir <- flat.norm.dir <- ann.dir <- "data/";
hierScore.dir <- perf.dir <- "results/";
dag.file <- "graph";
flat.file <- "scores";
ann.file <- "labels";
Do.heuristic.methods(heuristic.fun=heuristicAND, norm=FALSE,
norm.type="MaxNorm", flat.file=flat.file, ann.file=ann.file, dag.file=dag.file,
flat.dir=flat.dir, ann.dir=ann.dir, dag.dir=dag.dir, flat.norm.dir=flat.norm.dir,
n.round=3, f.criterion="F", 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 = heuristic.fun, norm = TRUE,
  norm.type = "NONE", 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,
  flat.norm.dir = NULL, n.round = 3, f.criterion = "F",
  hierScore.dir = hierScore.dir, perf.dir = perf.dir)
```

**Arguments**

<code>heuristic.fun</code>	can be one of the following three values: <ol style="list-style-type: none"> <li>1. <code>heuristicMAX</code>: run the heuristic method MAX;</li> <li>2. <code>heuristicAND</code>: run the heuristic method AND;</li> <li>3. <code>heuristicOR</code>: run the heuristic method OR;</li> </ol>
<code>norm</code>	boolean value: <ul style="list-style-type: none"> <li>• TRUE (def.): the flat scores matrix has been already normalized in according to a normalization method;</li> <li>• FALSE: the flat scores matrix has not been normalized yet. See the parameter <code>norm</code> for which normalization can be applied.</li> </ul>
<code>norm.type</code>	can be one of the following three values: <ol style="list-style-type: none"> <li>1. NONE (def.): set <code>norm.type</code> to NONE if and only if the parameter <code>norm</code> is set to TRUE;</li> <li>2. <code>MaxNorm</code>: each score is divided w.r.t. the max of each class;</li> <li>3. <code>Qnorm</code>: quantile normalization. <b>preprocessCore</b> package is used.</li> </ol>
<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 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>flat.norm.dir</code>	relative path where flat normalized scores matrix must be stored. Use this parameter if and only if <code>norm</code> is set to FALSE, otherwise set <code>flat.norm.dir</code> to NULL (def.)
<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
<code>f.criterion</code>	character. Type of F-measure to be used to select the best F-measure. Two possibilities:

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
- hierScore.dir    relative path where the hierarchical scores matrix must be stored
- perf.dir        relative path where the term-centric and protein-centric measures must be stored

## Value

Five rda files stored in the respective output directories:

1. hierarchical scores matrix: a matrix with examples on rows and classes on columns representing the computed hierarchical scores for each example and for each considered class. This file is stored in hierScore.dir directory.
2. FMM (F-Measure Multilabel) results: F-score computed by find.best.f function. Both *flat* and *hierarchical* results are reported. This file is stored in perf.dir directory.
3. PRC (area under Precision-Recall Curve) results: PRC computed by **precrec** package. Both *flat* and *hierarchical* results are reported. This file is stored in perf.dir directory.
4. AUC (Area Under ROC Curve) results: AUC computed by **precrec** package. Both *flat* and *hierarchical* results are reported. This file is stored in perf.dir directory.
5. PXR (Precision at fixed Recall levels) average and per classes: PXR computed by **PerfMeas** package. It is stored in perf.dir directory.

## Examples

```
data(graph);
data(scores);
data(labels);
data(test.index);
if (!dir.exists("data")){
  dir.create("data");
}
if (!dir.exists("results")){
  dir.create("results");
}
save(g,file="data/graph.rda");
save(L,file="data/labels.rda");
save(S,file="data/scores.rda");
save(test.index, file="data/test.index.rda");
ind.dir <- dag.dir <- flat.dir <- flat.norm.dir <- ann.dir <- "data/";
hierScore.dir <- perf.dir <- "results/";
ind.test.set <- "test.index";
dag.file <- "graph";
flat.file <- "scores";
ann.file <- "labels";
Do.heuristic.methods.holdout(heuristic.fun=heuristicAND, norm=FALSE,
norm.type="MaxNorm", 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, flat.norm.dir=flat.norm.dir, n.round=3, f.criterion="F",
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 = "NONE", flat.file = flat.file,
  ann.file = ann.file, dag.file = dag.file, flat.dir = flat.dir,
  ann.dir = ann.dir, dag.dir = dag.dir, flat.norm.dir = NULL,
  n.round = 3, f.criterion = "F", hierScore.dir = hierScore.dir,
  perf.dir = perf.dir)
```

**Arguments**

norm	boolean value: <ul style="list-style-type: none"> <li>• TRUE (def.): the flat scores matrix has been already normalized in according to a normalization method;</li> <li>• FALSE: the flat scores matrix has not been normalized yet. See the parameter norm.type for which normalization can be applied.</li> </ul>
norm.type	can be one of the following three values: <ol style="list-style-type: none"> <li>1. NONE (def.): set norm.type to NONE if and only if the parameter norm is set to TRUE;</li> <li>2. MaxNorm: each score is divided w.r.t. the max of each class;</li> <li>3. Qnorm: quantile normalization. <b>preprocessCore</b> package is used.</li> </ol>
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 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
flat.norm.dir	relative path where flat normalized scores matrix must be stored. Use this parameter if and only if norm is set to FALSE, otherwise set flat.norm.dir to NULL (def.)
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

<code>f.criterion</code>	character. Type of F-measure to be used to select the best F-measure. Two possibilities: <ol style="list-style-type: none"> <li>1. F (def.): corresponds to the harmonic mean between the average precision and recall</li> <li>2. avF: corresponds to the per-example F-score averaged across all the examples</li> </ol>
<code>hierScore.dir</code>	relative path where the hierarchical scores matrix must be stored
<code>perf.dir</code>	relative path where the term-centric and protein-centric measures must be stored

### Value

Five rda files stored in the respective output directories:

1. `hierarchical scores matrix`: a matrix with examples on rows and classes on columns representing the computed hierarchical scores for each example and for each considered class. This file is stored in `hierScore.dir` directory.
2. `FMM (F-Measure Multilabel) results`: F-score computed by `find.best.f` function. Both *flat* and *hierarchical* results are reported. This file is stored in `perf.dir` directory.
3. `PRC (area under Precision-Recall Curve) results`: PRC computed by **precrec** package. Both *flat* and *hierarchical* results are reported. This file is stored in `perf.dir` directory.
4. `AUC (Area Under ROC Curve) results`: AUC computed by **precrec** package. Both *flat* and *hierarchical* results are reported. This file is stored in `perf.dir` directory.
5. `PXR (Precision at fixed Recall levels) average and per classes`: PXR computed by **PerfMeas** package. It is stored in `perf.dir` directory.

### See Also

[HTD-DAG](#)

### Examples

```
data(graph);
data(scores);
data(labels);
if (!dir.exists("data")){
  dir.create("data");
}
if (!dir.exists("results")){
  dir.create("results");
}
save(g,file="data/graph.rda");
save(L,file="data/labels.rda");
save(S,file="data/scores.rda");
dag.dir <- flat.dir <- flat.norm.dir <- ann.dir <- "data/";
hierScore.dir <- perf.dir <- "results/";
dag.file <- "graph";
flat.file <- "scores";
ann.file <- "labels";
Do.HTD(norm=FALSE, norm.type= "MaxNorm", flat.file=flat.file, ann.file=ann.file,
```

```
dag.file=dag.file, flat.dir=flat.dir, ann.dir=ann.dir, dag.dir=dag.dir,
flat.norm.dir=flat.norm.dir, n.round=3, f.criterion ="F", 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 = "NONE", 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, flat.norm.dir = NULL, n.round = 3,
  f.criterion = "F", hierScore.dir = hierScore.dir, perf.dir = perf.dir)
```

## Arguments

norm	boolean value: <ul style="list-style-type: none"> <li>• TRUE (def.): the flat scores matrix has been already normalized in according to a normalization method;</li> <li>• FALSE: the flat scores matrix has not been normalized yet. See the parameter norm for which normalization can be applied.</li> </ul>
norm.type	can be one of the following three values: <ol style="list-style-type: none"> <li>1. NONE (def.): set norm.type to NONE if and only if the parameter norm is set to TRUE;</li> <li>2. MaxNorm: each score is divided w.r.t. the max of each class;</li> <li>3. Qnorm: quantile normalization. <b>preprocessCore</b> package is used.</li> </ol>
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 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)
ind.test.set	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
ind.dir	relative path to folder where ind.test.set is stored
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

<code>flat.norm.dir</code>	relative path where flat normalized scores matrix must be stored. Use this parameter if and only if <code>norm</code> is set to <code>FALSE</code> , otherwise set <code>flat.norm.dir</code> to <code>NULL</code> (def.)
<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
<code>f.criterion</code>	character. Type of F-measure to be used to select the best F-measure. Two possibilities: <ol style="list-style-type: none"> <li>1. <code>F</code> (def.): corresponds to the harmonic mean between the average precision and recall</li> <li>2. <code>avF</code>: corresponds to the per-example F-score averaged across all the examples</li> </ol>
<code>hierScore.dir</code>	relative path where the hierarchical scores matrix must be stored
<code>perf.dir</code>	relative path where the term-centric and protein-centric measures must be stored

### Value

Five rda files stored in the respective output directories:

1. `hierarchical_scores_matrix`: a matrix with examples on rows and classes on columns representing the computed hierarchical scores for each example and for each considered class. This file is stored in `hierScore.dir` directory.
2. `FMM` (F-Measure Multilabel) results: F-score computed by `find.best.f` function. Both *flat* and *hierarchical* results are reported. This file is stored in `perf.dir` directory.
3. `PRC` (area under Precision-Recall Curve) results: PRC computed by **precrec** package. Both *flat* and *hierarchical* results are reported. This file is stored in `perf.dir` directory.
4. `AUC` (Area Under ROC Curve) results: AUC computed by **precrec** package. Both *flat* and *hierarchical* results are reported. This file is stored in `perf.dir` directory.
5. `PXR` (Precision at fixed Recall levels) average and per classes: PXR computed by **PerfMeas** package. It is stored in `perf.dir` directory.

### See Also

[HTD-DAG](#)

### Examples

```
data(graph);
data(scores);
data(labels);
data(test.index);
if (!dir.exists("data")){
  dir.create("data");
}
if (!dir.exists("results")){
  dir.create("results");
}
save(g,file="data/graph.rda");
save(L,file="data/labels.rda");
```

```

save(S,file="data/scores.rda");
save(test.index, file="data/test.index.rda");
ind.dir <- dag.dir <- flat.dir <- flat.norm.dir <- ann.dir <- "data/";
hierScore.dir <- perf.dir <- "results/";
ind.test.set <- "test.index";
dag.file <- "graph";
flat.file <- "scores";
ann.file <- "labels";
Do.HTD.holdout(norm=FALSE, norm.type= "MaxNorm", 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, flat.norm.dir=flat.norm.dir, n.round=3, f.criterion ="F",
hierScore.dir=hierScore.dir, perf.dir=perf.dir);

```

---

do.subgraph	<i>Build subgraph</i>
-------------	-----------------------

---

## 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(hpo.ann, n)
```

### Arguments

hpo.ann	the annotations 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.tpr.threshold.free *TPR-DAG vanilla*


---

### Description

High level function to correct the computed scores in a hierarchy according to the TPR-DAG vanilla algorithm

### Usage

```
Do.tpr.threshold.free(norm = TRUE, norm.type = "NONE",
  flat.file = flat.file, ann.file = ann.file, dag.file = dag.file,
  flat.dir = flat.dir, ann.dir = ann.dir, dag.dir = dag.dir,
  flat.norm.dir = NULL, n.round = 3, f.criterion = "F",
  hierScore.dir = hierScore.dir, perf.dir = perf.dir)
```

**Arguments**

norm	boolean value: <ul style="list-style-type: none"> <li>• TRUE (def.): the flat scores matrix has been already normalized in according to a normalization method;</li> <li>• FALSE: the flat scores matrix has not been normalized yet. See the parameter norm.type for which normalization can be applied.</li> </ul>
norm.type	can be one of the following three values: <ol style="list-style-type: none"> <li>1. NONE (def.): set norm.type to NONE if and only if the parameter norm is set to TRUE;</li> <li>2. MaxNorm: each score is divided w.r.t. the max of each class;</li> <li>3. Qnorm: quantile normalization. <b>preprocessCore</b> package is used.</li> </ol>
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 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
flat.norm.dir	relative path where flat normalized scores matrix must be stored. Use this parameter if and only if norm is set to FALSE, otherwise set flat.norm.dir to NULL (def.)
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
f.criterion	character. Type of F-measure to be used to select the best F-measure. Two possibilities: <ol style="list-style-type: none"> <li>1. F (def.): corresponds to the harmonic mean between the average precision and recall</li> <li>2. avF: corresponds to the per-example F-score averaged across all the examples</li> </ol>
hierScore.dir	relative path where the hierarchical scores matrix must be stored
perf.dir	relative path where the term-centric and protein-centric measures must be stored

**Value**

Five rda files stored in the respective output directories:

1. hierarchical scores matrix: a matrix with examples on rows and classes on columns representing the computed hierarchical scores for each example and for each considered class. This file is stored in hierScore.dir directory.
2. FMM (F-Measure Multilabel) results: F-score computed by find.best.f function. Both *flat* and *hierarchical* results are reported. This file is stored in perf.dir directory.

3. PRC (area under Precision-Recall Curve) results: PRC computed by **precrec** package. Both *flat* and *hierarchical* results are reported. This file is stored in `perf.dir` directory.
4. AUC (Area Under ROC Curve) results: AUC computed by **precrec** package. Both *flat* and *hierarchical* results are reported. This file is stored in `perf.dir` directory.
5. PXR (Precision at fixed Recall levels) average and per classes: PXR computed by **PerfMeas** package. It is stored in `perf.dir` directory.

## See Also

[TPR-DAG](#)

## Examples

```
data(graph);
data(scores);
data(labels);
if (!dir.exists("data")){
  dir.create("data");
}
if (!dir.exists("results")){
  dir.create("results");
}
save(g,file="data/graph.rda");
save(L,file="data/labels.rda");
save(S,file="data/scores.rda");
dag.dir <- flat.dir <- flat.norm.dir <- ann.dir <- "data/";
hierScore.dir <- perf.dir <- "results/";
dag.file <- "graph";
flat.file <- "scores";
ann.file <- "labels";
Do.tpr.threshold.free(norm=FALSE, norm.type= "MaxNorm", flat.file=flat.file,
ann.file=ann.file, dag.file=dag.file, flat.dir=flat.dir, ann.dir=ann.dir,
dag.dir=dag.dir, flat.norm.dir=flat.norm.dir, n.round=3, f.criterion ="F",
hierScore.dir=hierScore.dir, perf.dir=perf.dir);
```

---

Do.tpr.threshold.free.holdout

*TPR-DAG vanilla holdout*

---

## Description

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



## Usage

```
Do.tpr.threshold.free.holdout(norm = TRUE, norm.type = "NONE",
  flat.file = flat.file, ann.file = ann.file, dag.file = dag.file,
  flat.dir = flat.dir, ann.dir = ann.dir, dag.dir = dag.dir,
  flat.norm.dir = NULL, ind.test.set = ind.test.set, ind.dir = ind.dir,
  n.round = 3, f.criterion = "F", hierScore.dir = hierScore.dir,
  perf.dir = perf.dir)
```

## Arguments

norm	boolean value: <ul style="list-style-type: none"> <li>• TRUE (def.): the flat scores matrix has been already normalized in according to a normalization method;</li> <li>• FALSE: the flat scores matrix has not been normalized yet. See the parameter norm for which normalization can be applied.</li> </ul>
norm.type	can be one of the following three values: <ol style="list-style-type: none"> <li>1. NONE (def.): set norm.type to NONE if and only if the parameter norm is set to TRUE;</li> <li>2. MaxNorm: each score is divided w.r.t. the max of each class;</li> <li>3. Qnorm: quantile normalization. <b>preprocessCore</b> package is used.</li> </ol>
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 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
flat.norm.dir	relative path where flat normalized scores matrix must be stored. Use this parameter if and only if norm is set to FALSE, otherwise set flat.norm.dir to NULL (def.)
ind.test.set	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
ind.dir	relative path to folder where ind.test.set is stored
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
f.criterion	character. Type of F-measure to be used to select the best F-measure. Two possibilities: <ol style="list-style-type: none"> <li>1. F (def.): corresponds to the harmonic mean between the average precision and recall</li> <li>2. avF: corresponds to the per-example F-score averaged across all the examples</li> </ol>
hierScore.dir	relative path where the hierarchical scores matrix must be stored
perf.dir	relative path where the term-centric and protein-centric measures must be stored

## Value

Five rda files stored in the respective output directories:

1. `hierarchical scores matrix`: a matrix with examples on rows and classes on columns representing the computed hierarchical scores for each example and for each considered class. This file is stored in `hierScore.dir` directory.
2. FMM (F-Measure Multilabel) results: F-score computed by `find.best.f` function. Both *flat* and *hierarchical* results are reported. This file is stored in `perf.dir` directory.
3. PRC (area under Precision-Recall Curve) results: PRC computed by **precrec** package. Both *flat* and *hierarchical* results are reported. This file is stored in `perf.dir` directory.
4. AUC (Area Under ROC Curve) results: AUC computed by **precrec** package. Both *flat* and *hierarchical* results are reported. This file is stored in `perf.dir` directory.
5. PXR (Precision at fixed Recall levels) average and per classes: PXR computed by **PerfMeas** package. It is stored in `perf.dir` directory.

## See Also

[TPR-DAG](#)

## Examples

```
data(graph);
data(scores);
data(labels);
data(test.index);
if (!dir.exists("data")){
  dir.create("data");
}
if (!dir.exists("results")){
  dir.create("results");
}
save(g,file="data/graph.rda");
save(L,file="data/labels.rda");
save(S,file="data/scores.rda");
save(test.index, file="data/test.index.rda");
ind.dir <- dag.dir <- flat.dir <- flat.norm.dir <- ann.dir <- "data/";
hierScore.dir <- perf.dir <- "results/";
ind.test.set <- "test.index";
dag.file <- "graph";
flat.file <- "scores";
ann.file <- "labels";
Do.tpr.threshold.free.holdout(norm=FALSE, norm.type= "MaxNorm", 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, flat.norm.dir=flat.norm.dir,
n.round=3, f.criterion ="F", hierScore.dir=hierScore.dir, perf.dir=perf.dir);
```

---

do.unstratified.cv.data

*Unstratified cross-validation*


---

## Description

This function splits a dataset in k-fold in an unstratified way (that is a fold may not have 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

S	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. HPO terms)
kk	number of folds in which to split the dataset (def. k=5)
seed	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 names of the examples.

## Examples

```
data(scores);
```

---

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 correspondes 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.

---

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

## Arguments

<code>target</code>	matrix with the target multilabels: 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>pred</code>	a numeric matrix with predicted values (scores): rows correspond to examples and columns to classes
<code>n.round</code>	number of rounding digits to be applied to <code>pred</code> (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"> <li>1. F (def.) corresponds to the harmonic mean between the average precision and recall;</li> <li>2. avF corresponds to the per-example F-score averaged across all the examples.</li> </ol>
verbose	boolean. If TRUE (def.) the number of iterations are printed on stdout
b.per.example	boolean. <ul style="list-style-type: none"> <li>• TRUE: results are returned for each example;</li> <li>• FALSE: only the average results are returned</li> </ul>

### Details

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

### 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);
```

---

full.annotation.matrix

*Full annotations 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 annotations 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);
```

---

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**

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

**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](https://doi.org/10.1186/gb-2008-9-s1-s6))

**Usage**

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

heuristicAND(S, g, root = "00")

heuristicOR(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.heuristicMAX <- heuristicMAX(S,g,root);
S.heuristicAND <- heuristicAND(S,g,root);
S.heuristicOR <- heuristicOR(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

y.hier	vector of scores relative to a single example. This must be a named numeric vector
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)
S.hier	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

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

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);
```

---

Multilabel.F.measure    *Multilabel F-measure*


---

**Description**

Method for computing Precision, Recall, Specificity, Accuracy and F-measure for multiclass 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 multilabels: 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
b.per.example	boolean. <ul style="list-style-type: none"> <li>• TRUE: results are returned for each example;</li> <li>• FALSE: only the average results are returned</li> </ul>

**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

<code>g</code>	a graph of class <code>graphNEL</code> . It represents the hierarchy of the classes
<code>root</code>	name of the root node (def. <code>root="00"</code> )
<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

`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);

```

---

read.graph	<i>Read a directed graph from a file</i>
------------	------------------------------------------

---

**Description**

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

**Usage**

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

**Arguments**

file	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
------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Value**

an object of class graphNEL

**Examples**

```

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

```

---

read.undirected.graph	<i>Read an undirected graph from a file</i>
-----------------------	---------------------------------------------

---

**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")
```

**Arguments**

file                      name of the file to be read

**Value**

a graph of class graphNEL

**Examples**

```
edges <- system.file("extdata/edges.txt" ,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);
```

```
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
```

*HPO specific annotations matrix*

---

### Description

Construct the labels matrix of the most specific HPO terms

### Usage

```
specific.annotation.matrix(file = "gene2pheno.txt", genename = "TRUE")
```



**Arguments**

- file** text file representing the most specific associations gene-HPO term (def: "gene2pheno.txt"). The file must be written as sequence of rows. Each row represents a gene and all its associations with abnormal phenotype tab separated, *e.g.*: *gene\_1* <tab> *phen1* <tab> ... *phen\_N*. See **Details** section to know more information about how to obtain this file.
- genename** boolean value:
- TRUE (def.): the names of genes are *gene symbol* (i.e. characters);
  - FALSE: the names of gene are entrez *gene ID* (i.e. integer numbers);

**Details**

The input plain text file representing the most specific associations gene-HPO term can be obtained by forking the GitHub repository [HPOparser](#), a collection of Perl subroutines to parse the HPO OBO file and the HPO annotations 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", package="HEMDAG");
spec.ann <- specific.annotation.matrix(gene2pheno, genename=TRUE);
```

---

```
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=5)
seed	seed of the random generator (def=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 $M[i, j] = 0$ .

**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 examples positives and negatives.

**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);
```

### Description

Different variants of the TPR-DAG algorithm are implemented. In their more general form the TPR-DAG algorithms adopt a two step learnig 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.threshold(S, g, root = "00", t = 0.5)

tpr.threshold.free(S, g, root = "00")

tpr.weighted.threshold.free(S, g, root = "00", w = 0.5)

tpr.weighted.threshold(S, g, root = "00", t = 0.5, w = 0.5)

```

### 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 on the top-level of the hierarchy (def. root="00")
t	threshold for the choice of positive children (def. t=0.5)
w	weight to balance between the contribution of the node $i$ and that of its positive children

### 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  $\phi_i$  are the positive children of  $i$ . Different strategies to select the positive children  $\phi_i$  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 ben 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  $\phi$ , 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  $w$ .

### Value

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

### Examples

```
data(graph);
data(scores);
data(labels);
root <- root.node(g);
S.tprTF <- tpr.threshold.free(S,g,root);
S.tprT <- tpr.threshold(S,g,root,t=0.5);
S.tprW <- tpr.weighted.threshold.free(S,g,root,w=0.5);
S.tprWT <- tpr.weighted.threshold(S,g,root,w=0.5, t=0.5);
```

---

TPR.DAG.CV

*TPR-DAG cross-validation*

---

### Description

High level function to correct the computed scores in a hierarchy according to one of the TPR-DAG variants performing cross-validation experiments

**Usage**

```
Do.tpr.threshold.cv(threshold = seq(from = 0.1, to = 0.9, by = 0.1), kk = 5,
  seed = NULL, norm = TRUE, norm.type = "NONE", flat.file = flat.file,
  ann.file = ann.file, dag.file = dag.file, flat.dir = flat.dir,
  ann.dir = ann.dir, dag.dir = dag.dir, flat.norm.dir = NULL,
  n.round = 3, f.criterion = "F", hierScore.dir = hierScore.dir,
  perf.dir = perf.dir)
```

```
Do.tpr.weighted.threshold.free.cv(weight = seq(from = 0.1, to = 1, by = 0.1),
  kk = 5, seed = NULL, norm = TRUE, norm.type = "NONE",
  flat.file = flat.file, ann.file = ann.file, dag.file = dag.file,
  flat.dir = flat.dir, ann.dir = ann.dir, dag.dir = dag.dir,
  flat.norm.dir = NULL, n.round = 3, f.criterion = "F",
  hierScore.dir = hierScore.dir, perf.dir = perf.dir)
```

```
Do.tpr.weighted.threshold.cv(threshold = seq(from = 0.1, to = 0.9, by = 0.1),
  weight = seq(from = 0.1, to = 1, by = 0.1), kk = 5, seed = NULL,
  norm = TRUE, norm.type = "NONE", flat.file = flat.file,
  ann.file = ann.file, dag.file = dag.file, flat.dir = flat.dir,
  ann.dir = ann.dir, dag.dir = dag.dir, flat.norm.dir = NULL,
  n.round = 3, f.criterion = "F", 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 increasing.
kk	number of folds of the cross validation (def: kk=5);
seed	initialization seed for the random generator to create folds. If NULL (def.) no initialization is performed
norm	boolean value: <ul style="list-style-type: none"> <li>• TRUE (def.): the flat scores matrix has been already normalized in according to a normalization method;</li> <li>• FALSE: the flat scores matrix has not been normalized yet. See the parameter norm.type for which normalization can be applied.</li> </ul>
norm.type	can be one of the following three values: <ol style="list-style-type: none"> <li>1. NONE (def.): set norm.type to NONE if and only if the parameter norm is set to TRUE;</li> <li>2. MaxNorm: each score is divided w.r.t. the max of each class;</li> <li>3. Qnorm: quantile normalization. <b>preprocessCore</b> package is used.</li> </ol>
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 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 <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>flat.norm.dir</code>	relative path where flat normalized scores matrix must be stored. Use this parameter if and only if <code>norm</code> is set to <code>FALSE</code> , otherwise set <code>flat.norm.dir</code> to <code>NULL</code> (def.)
<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
<code>f.criterion</code>	character. Type of F-measure to be used to select the best F-measure. Two possibilities: <ol style="list-style-type: none"> <li>1. <code>F</code> (def.): corresponds to the harmonic mean between the average precision and recall</li> <li>2. <code>avF</code>: corresponds to the per-example F-score averaged across all the examples</li> </ol>
<code>hierScore.dir</code>	relative path where the hierarchical scores matrix must be stored
<code>perf.dir</code>	relative path where the term-centric and protein-centric measures must be stored
<code>weight</code>	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 weight is, but obviously the execution time will be increasing.

## Details

These high level functions perform a classical cross-validation procedure to find the best threshold maximizing on F-score measure.

## Value

Five `rda` files stored in the respective output directories:

1. `hierarchical scores matrix`: a matrix with examples on rows and classes on columns representing the computed hierarchical scores for each example and for each considered class. This file is stored in `hierScore.dir` directory.
2. `FMM (F-Measure Multilabel) results`: F-score computed by `find.best.f` function. Both *flat* and *hierarchical* results are reported. This file is stored in `perf.dir` directory.
3. `PRC (area under Precision-Recall Curve) results`: PRC computed by **precrec** package. Both *flat* and *hierarchical* results are reported. This file is stored in `perf.dir` directory.
4. `AUC (Area Under ROC Curve) results`: AUC computed by **precrec** package. Both *flat* and *hierarchical* results are reported. This file is stored in `perf.dir` directory.
5. `PXR (Precision at fixed Recall levels) average and per classes`: PXR computed by **PerfMeas** package. It is stored in `perf.dir` directory.

## See Also

[TPR-DAG](#)

**Examples**

```

data(graph);
data(scores);
data(labels);
if (!dir.exists("data")){
  dir.create("data");
}
if (!dir.exists("results")){
  dir.create("results");
}
save(g,file="data/graph.rda");
save(L,file="data/labels.rda");
save(S,file="data/scores.rda");
dag.dir <- flat.dir <- flat.norm.dir <- ann.dir <- "data/";
hierScore.dir <- perf.dir <- "results/";
dag.file <- "graph";
flat.file <- "scores";
ann.file <- "labels";
threshold <- seq(from=0.1, to=0.5, by=0.1);
Do.tpr.threshold.cv(threshold=threshold, kk=5, seed=23, norm=FALSE, norm.type= "MaxNorm",
  flat.file=flat.file, ann.file=ann.file, dag.file=dag.file, flat.dir=flat.dir, ann.dir=ann.dir,
  dag.dir=dag.dir, flat.norm.dir=flat.norm.dir, n.round=3, f.criterion = "F",
  hierScore.dir=hierScore.dir, perf.dir=perf.dir);

```

TPR.DAG.HOLDOUT

*TPR-DAG holdout***Description**

High level function to correct the computed scores in a hierarchy according to one of the TPR-DAG variants performing a classical holdout procedure.

**Usage**

```

Do.tpr.threshold.holdout(threshold = seq(from = 0.1, to = 0.9, by = 0.1),
  kk = 5, seed = NULL, norm = TRUE, norm.type = "NONE",
  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, flat.norm.dir = NULL,
  n.round = 3, f.criterion = "F", hierScore.dir = hierScore.dir,
  perf.dir = perf.dir)

```

```

Do.tpr.weighted.threshold.free.holdout(weight = seq(from = 0.1, to = 1, by =
  0.1), kk = 5, seed = NULL, norm = TRUE, norm.type = "NONE",
  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, flat.norm.dir = NULL,
  n.round = 3, f.criterion = "F", hierScore.dir = hierScore.dir,

```

```
perf.dir = perf.dir)
```

```
Do.tpr.weighted.threshold.holdout(threshold = seq(from = 0.1, to = 0.9, by =
  0.1), weight = seq(from = 0.1, to = 1, by = 0.1), kk = 5, seed = NULL,
  norm = TRUE, norm.type = "NONE", 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, flat.norm.dir = NULL, n.round = 3,
  f.criterion = "F", 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 increasing.
kk	number of folds of the cross validation (def: kk=5);
seed	initialization seed for the random generator to create folds. If NULL (def.) no initialization is performed
norm	boolean value: <ul style="list-style-type: none"> <li>• TRUE (def.): the flat scores matrix has been already normalized in according to a normalization method;</li> <li>• FALSE: the flat scores matrix has not been normalized yet. See the parameter norm.type for which normalization can be applied.</li> </ul>
norm.type	can be one of the following three values: <ol style="list-style-type: none"> <li>1. NONE (def.): set norm.type to NONE if and only if the parameter norm is set to TRUE;</li> <li>2. MaxNorm: each score is divided w.r.t. the max of each class;</li> <li>3. Qnorm: quantile normalization. <b>preprocessCore</b> package is used.</li> </ol>
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 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)
ind.test.set	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
ind.dir	relative path to folder where ind.test.set is stored
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
flat.norm.dir	relative path where flat normalized scores matrix must be stored. Use this parameter if and only if norm is set to FALSE, otherwise set flat.norm.dir to NULL (def.)



<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
<code>f.criterion</code>	character. Type of F-measure to be used to select the best F-measure. Two possibilities: <ol style="list-style-type: none"> <li>1. F (def.): corresponds to the harmonic mean between the average precision and recall</li> <li>2. avF: corresponds to the per-example F-score averaged across all the examples</li> </ol>
<code>hierScore.dir</code>	relative path where the hierarchical scores matrix must be stored
<code>perf.dir</code>	relative path where the term-centric and protein-centric measures must be stored
<code>weight</code>	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 weight is, but obviously the execution time will be increasing.

### Details

These high level functions perform a classical holdout procedure to find the best threshold maximizing on F.score measure.

### Value

Five rda files stored in the respective output directories:

1. `hierarchical_scores_matrix`: a matrix with examples on rows and classes on columns representing the computed hierarchical scores for each example and for each considered class. This file is stored in `hierScore.dir` directory.
2. FMM (F-Measure Multilabel) results: F-score computed by `find.best.f` function. Both *flat* and *hierarchical* results are reported. This file is stored in `perf.dir` directory.
3. PRC (area under Precision-Recall Curve) results: PRC computed by **precrec** package. Both *flat* and *hierarchical* results are reported. This file is stored in `perf.dir` directory.
4. AUC (Area Under ROC Curve) results: AUC computed by **precrec** package. Both *flat* and *hierarchical* results are reported. This file is stored in `perf.dir` directory.
5. PXR (Precision at fixed Recall levels) average and per classes: PXR computed by **PerfMeas** package. It is stored in `perf.dir` directory.

### See Also

[TPR-DAG](#)

### Examples

```
data(graph);
data(scores);
data(labels);
data(test.index);
if (!dir.exists("data")){
  dir.create("data");
```

```

}
if (!dir.exists("results")){
  dir.create("results");
}
save(g,file="data/graph.rda");
save(L,file="data/labels.rda");
save(S,file="data/scores.rda");
save(test.index, file="data/test.index.rda");
ind.dir <- dag.dir <- flat.dir <- flat.norm.dir <- ann.dir <- "data/";
hierScore.dir <- perf.dir <- "results/";
ind.test.set <- "test.index";
dag.file <- "graph";
flat.file <- "scores";
ann.file <- "labels";
threshold <- seq(from=0.1, to=0.3, by=0.1);
Do.tpr.threshold.holdout(threshold=threshold, kk=5, seed=23, norm=FALSE, norm.type="MaxNorm",
  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, flat.norm.dir=flat.norm.dir,
  n.round=3, f.criterion="F", hierScore.dir=hierScore.dir, perf.dir=perf.dir);

```

---

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 HPO 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 HPO terms.
anc	list of the ancestors of the ontology.

## Value

an annotation table T: rows correspond to genes and columns to HPO 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

Trasform 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, compressed = TRUE, output.file = "net.file.gz")
```

## Arguments

m	a weighthd adjacency matrix of the graph. Rows and columns are examples. It must be a square named matrix.
compressed	boolean value: <ul style="list-style-type: none"> <li>• TRUE (def.): the output file will be a gz compressed format;</li> <li>• FALSE: the output file will be a plain text format;</li> </ul>
output.file	name of the file of the to be written

## 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

if compressed=TRUE the weighted adjacency matrix as tupla is stored in a compressed gz, otherwise (compressed=FALSE) it is stored in a plain text file.

## Examples

```
## Not run:
data(wadj);
tupla.matrix(W,compressed=TRUE, output.file="tupla.wadj.gz");
tupla.matrix(W,compressed=FALSE, output.file="tupla.wadj.txt");
## End(Not run)
```

---

`weighted.adjacency.matrix`*Weighted Adjacency Matrix*

---

## Description

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

## Usage

```
weighted.adjacency.matrix(file = "edges.txt", compressed = TRUE,  
  nodename = TRUE)
```

## Arguments

- |            |                                                                                                                                                                                                                                                               |
|------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| file       | name of the plain text file to be read (def. edges). 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.<br>For instance: nodeX nodeY score |
| compressed | boolean value: <ul style="list-style-type: none"><li>• TRUE (def.): the input file must be in a .gz compressed format;</li><li>• FALSE: the input file must be in a plain text format;</li></ul>                                                              |
| nodename   | boolean value: <ul style="list-style-type: none"><li>• TRUE (def.): the names of nodes are gene symbol (i.e. characters);</li><li>• FALSE: the names of the nodes are entrez gene ID (i.e. integer numbers);</li></ul>                                        |

## Details

The input parameter nodename sorts the row names of the wadj matrix in increasing order if they are integer number or in alphabetic order if they are characters.

## Value

a named symmetric weighted adjacency matrix of the graph

## Examples

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

---

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")
```

**Arguments**

g	a graph of class graphNEL
file	name of the file to be written

**Value**

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

**Examples**

```
## Not run:  
data(graph);  
write.graph(g, file="graph.edges.txt");  
## End(Not run)
```

# Index

## \*Topic **package**

- HEMDAG-package, 3
- ancestors, 4
- AUPRC.single.class, 5, 6
- AUPRC.single.over.classes, 5, 5
- AUROC.single.class, 6, 7
- AUROC.single.over.classes, 7, 7
- build.ancestors, 8, 47, 66
- build.ancestors(ancestors), 4
- build.children(children), 9
- build.descendants(descendants), 11
- check.annotation.matrix.integrity, 8
- check.DAG.integrity, 9
- check.hierarchy
  - (hierarchical.checkers), 49
- children, 9
- compute.flipped.graph, 10
- constraints.matrix, 11
- descendants, 11
- DESCENS, 3, 12, 16, 19, 22, 24
- DESCENS.CV, 14
- DESCENS.HOLDOUT, 17
- descens.tau(DESCENS), 12
- descens.threshold(DESCENS), 12
- descens.weighted.threshold(DESCENS), 12
- distances.from.leaves, 20
- Do.descens.tau.cv(DESCENS.CV), 14
- Do.descens.tau.holdout
  - (DESCENS.HOLDOUT), 17
- Do.descens.threshold.cv(DESCENS.CV), 14
- Do.descens.threshold.free, 21
- Do.descens.threshold.free.holdout, 23
- Do.descens.threshold.holdout
  - (DESCENS.HOLDOUT), 17
- Do.descens.weighted.threshold.cv
  - (DESCENS.CV), 14
- Do.descens.weighted.threshold.free.cv
  - (DESCENS.CV), 14
- Do.descens.weighted.threshold.free.holdout
  - (DESCENS.HOLDOUT), 17
- Do.descens.weighted.threshold.holdout
  - (DESCENS.HOLDOUT), 17
- do.edges.from.HPO.obo, 25
- Do.FLAT.scores.normalization, 26
- Do.full.annotation.matrix, 27
- Do.heuristic.methods, 28
- Do.heuristic.methods.holdout, 30
- Do.HTD, 33
- Do.HTD.holdout, 35
- do.stratified.cv.data.over.classes
  - (stratified.cross.validation), 57
- do.stratified.cv.data.single.class
  - (stratified.cross.validation), 57
- do.subgraph, 37
- do.submatrix, 38
- Do.tpr.threshold.cv(TPR.DAG.CV), 60
- Do.tpr.threshold.free, 38
- Do.tpr.threshold.free.holdout, 40
- Do.tpr.threshold.holdout
  - (TPR.DAG.HOLDOUT), 63
- Do.tpr.weighted.threshold.cv
  - (TPR.DAG.CV), 60
- Do.tpr.weighted.threshold.free.cv
  - (TPR.DAG.CV), 60
- Do.tpr.weighted.threshold.free.holdout
  - (TPR.DAG.HOLDOUT), 63
- Do.tpr.weighted.threshold.holdout
  - (TPR.DAG.HOLDOUT), 63
- do.unstratified.cv.data, 43
- example.datasets, 43
- F.measure.multilabel
  - (Multilabel.F.measure), 51

F.measure.multilabel,matrix,matrix-method  
    (Multilabel.F.measure), 51  
find.best.f, 44  
find.leaves, 46  
full.annotation.matrix, 8, 28, 46  
  
g (example.datasets), 43  
get.children.bottom.up (children), 9  
get.children.top.down (children), 9  
get.parents (parents), 53  
graph.levels, 4, 10, 12, 47, 50, 53  
  
HEMDAG (HEMDAG-package), 3  
HEMDAG-package, 3  
Heuristic-Methods, 48  
heuristicAND (Heuristic-Methods), 48  
heuristicMAX (Heuristic-Methods), 48  
heuristicOR (Heuristic-Methods), 48  
hierarchical.checkers, 49, 50  
htd (HTD-DAG), 50  
HTD-DAG, 50  
  
L (example.datasets), 43  
  
Multilabel.F.measure, 51  
  
normalize.max, 52  
  
parents, 53  
  
read.graph, 54  
read.undirected.graph, 54  
root.node, 55  
  
S (example.datasets), 43  
specific.annotation.list, 56  
specific.annotation.matrix, 47, 56, 56,  
    66  
stratified.cross.validation, 57  
  
test.index (example.datasets), 43  
TPR-DAG, 58  
TPR.DAG.CV, 60  
TPR.DAG.HOLDOUT, 63  
tpr.threshold (TPR-DAG), 58  
tpr.weighted.threshold (TPR-DAG), 58  
transitive.closure.annotations, 8, 47,  
    66  
tupla.matrix, 67  
  
W (example.datasets), 43  
weighted.adjacency.matrix, 47, 68  
write.graph, 69