Package 'HEMDAG'

September 21, 2018

Title Hierarchical Ensemble Methods for Directed Acyclic Graphs

Version 2.2.5

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

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

Description An implementation of Hierarchical Ensemble Methods for Directed Acyclic Graphs (DAGs). The 'HEMDAG' package can be used to enhance the predictions of virtually any flat learning methods, 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>).

URL https://hemdag-tutorials.readthedocs.io,
https://github.com/marconotaro/HEMDAG

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

Depends R (>= 2.10)

License GPL (>= 3)

Encoding UTF-8

Repository CRAN

LazyLoad true

NeedsCompilation yes

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

Suggests Rgraphviz

RoxygenNote 6.0.1

Date/Publication 2018-09-21 16:00:17 UTC

R topics documented:

`HEMDAG-package` *HEMDAG: Hierarchical Ensemble Methods for Directed Acyclic Graphs*

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 methods, 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. A comprehensive tutorial on the 'HEMDAG' package, is available at https://hemdag-tutorials.readthedocs.io.

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. **GPAV**: Generalized Pool-Adjacent Violators, *Burdakov et al.* (`GPAV`);
3. **TPR-DAG**: True-Path Rule (`TPR-DAG-variants`);
4. **DESCENS**: Descendants Ensemble Classifier (`TPR-DAG-variants`);
5. **ISO-TPR**: Isotonic-True-Path Rule (`TPR-DAG-variants`);
6. **MAX**, **AND**, **OR**: Heuristic Methods, *Obozinski et al.* (`Heuristic-Methods`);

Author(s)

Marco Notaro[1] (https://orcid.org/0000-0003-4309-2200);
Alessandro Petrini[1] (https://orcid.org/0000-0002-0587-1484);
Giorgio Valentini[1] (https://orcid.org/0000-0002-5694-3919);

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

[1] AnacletoLab, DI, Dipartimento di Informatica, Universita' degli Studi di Milano

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

---

`adj.upper.tri` *Binary Upper Triangular Adjacency Matrix*

Description

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

Usage

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

Arguments

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

Details

The nodes of the matrix are topologically sorted. Let's denote with adj our adjacency matrix. Then adj represents a partial order data set in which the class j dominates the class i. In other words, adj[i,j]=1 means that j dominates i; adj[i,j]=0 means that there is no edge between the class i and the class j. Moreover the nodes of adj are enumerated so that adj[i,j]=1 implies $i < j$, i.e. adj is upper triangular.

Value

an adjacency matrix which is square, logical and upper triangular

Examples

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

ancestors *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

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.ancestos 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 *AUPRC measures*

Description

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

Usage

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

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

Arguments

`labels`	vector of the true labels (0 negative, 1 positive examples)
`scores`	a numeric vector of the values of the predicted labels (scores)
`folds`	number of folds on which computing the AUPRC. If `folds=NULL` (`def.`), the AUPRC is computed one-shot, otherwise the AUPRC is computed averaged across folds.
`seed`	initialization seed for the random generator to create folds. Set `seed` only if `folds`$\neq$`NULL`. If `seed=NULL` and `folds`$\neq$`NULL`, the AUPRC averaged across folds is computed without seed initialization.
`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.

Details

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

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

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

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

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

Value

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

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

Examples

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

AUROC *AUROC measures*

Description

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

Usage

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

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

Arguments

labels	vector of the true labels (0 negative, 1 positive examples)
scores	a numeric vector of the values of the predicted labels (scores)
folds	number of folds on which computing the AUROC. If folds=NULL (def.), the AUROC is computed one-shot, otherwise the AUROC is computed averaged across folds.
seed	initialization seed for the random generator to create folds. Set seed only if folds≠NULL. If seed=NULL and folds≠NULL, the AUROC averaged across folds is computed without seed initialization.
target	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.

Details

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

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

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

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

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

Value

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

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

Examples

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

---

check.annotation.matrix.integrity

Annotation matrix checker

Description

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

Usage

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

Arguments

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

Description

This function assess the integrity of a DAG

Usage

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

Arguments

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

Value

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

Examples

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

`children` *Build children*

Description

Compute the children for each node of a graph

Usage

```
build.children(g)

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

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

Arguments

`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 *Constraints Matrix*

Description

This function returns a matrix with two columns and as many rows as there are edges. The entries of the first columns are the index of the node the edge cames 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 graphNELL. It represents the hierarchy of the classes

Value

a constraints matrix w.r.t the graph g

Examples

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

---

create.stratified.fold.df

DataFrame for Stratified Cross Validation

Description

Create a data frame for stratified cross-validation

Usage

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

Arguments

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

Details

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

Value

a data frame with three columns:

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

Examples

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

`descendants` *Build descendants*

Description

Compute the descendants for each node of a graph

Usage

```
build.descendants(g)

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

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

Arguments

`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);
```

---

distances.from.leaves *Distances from leaves*

Description

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

Usage

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

Arguments

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

Value

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

Examples

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

---

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

Description

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

Usage

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

Arguments

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

Value

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

Examples

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

Do.flat.scores.normalization

Flat scores normalization

Description

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

Usage

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

Arguments

norm.type	can be one of the following two values: • MaxNorm (def.): each score is divided w.r.t. the max of each class; • Qnorm: a quantile normalization is applied. Library preprocessCore is used.
flat.file	name of the flat scores matrix (without rda extension)
flat.dir	relative path to folder where flat scores matrix is stored
flat.norm.dir	the directory where the normalized flat scores matrix must be stored

Details

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

Value

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

Examples

```
data(scores);
if(!dir.exists("data")){
dir.create("data");
}
if(!dir.exists("results")){
dir.create("results");
}
save(S,file="data/scores.rda");
flat.dir <- "data/";
flat.norm.dir <- "results/";
flat.file <- "scores";
norm.types <- c("MaxNorm","Qnorm");
for(norm.type in norm.types){
Do.flat.scores.normalization(norm.type=norm.type, flat.file=flat.file,
flat.dir=flat.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 adiacency 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 adjiacency matrix of the graph (without `rda` extension)
`net.dir`	relative path to directory where the weighted adjiacency 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 adjiacency 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.GPAV` *GPAV – High Level Function*

Description

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

Usage

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

Arguments

norm	boolean value: • TRUE (def.): the flat scores matrix has been already normalized in according to a normalization method; • FALSE: the flat scores matrix has not been normalized yet. See the parameter norm.type for which normalization can be applied.
norm.type	can be one of the following three values: 1. NULL (def.): set norm.type to NULL if and only if the parameter norm is set to TRUE; 2. MaxNorm: each score is divided for the maximum of each class; 3. Qnorm: quantile normalization. **preprocessCore** package is used.
W	vector of weight relative to a single example. If the vector W is not specified (def. W=NULL), W is a unitary vector of the same length of the columns' number of the flat scores matrix (root node included)
parallel	boolean value: • TRUE: execute the parallel implementation of GPAV (GPAV.parallel); • FALSE (def.): execute the sequential implementation of GPAV (GPAV.over.examples);
ncores	number of cores to use for parallel execution (def. 8). Set the parameter ncores to 1 if the parameter parallel is set to FALSE, otherwise set the desired number of cores
folds	number of folds of the cross validation on which computing the performance metrics averaged across folds (def. 5). If folds=NULL, the performance metrics are computed one-shot, otherwise the performance metrics are averaged across folds.
seed	initialization seed for the random generator to create folds (def. 23). If NULL folds are generated without seed initialization.
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: 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

`recall.levels`	a vector with the desired recall levels (`def: from:0.1, to:0.9, by:0.1`) to compute the the Precision at fixed Recall level (PXR)
`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
`hierScore.dir`	relative path where the hierarchical scores matrix must be stored
`perf.dir`	relative path where all the performance measures must be stored

Details

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

Value

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

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

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

See Also

GPAV

Examples

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

Do.GPAV.holdout *GPAV holdout*

Description

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

Usage

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

Arguments

norm	boolean value: • TRUE (def.): the flat scores matrix has been already normalized in according to a normalization method; • FALSE: the flat scores matrix has not been normalized yet. See the parameter norm for which normalization can be applied.
norm.type	can be one of the following three values: 1. NULL (def.): set norm.type to NULL if and only if the parameter norm is set to TRUE;

2. `MaxNorm`: each score is divided for the maximum of each class;
3. `Qnorm`: quantile normalization. **preprocessCore** package is used.

`W` vector of weight relative to a single example. If the vector `W` is not specified (`def. W=NULL`), `W` is a unitary vector of the same length of the columns' number of the flat scores matrix (root node included)

`parallel` boolean value:

- `TRUE`: execute the parallel implementation of GPAV (`GPAV.parallel`);
- `FALSE` (`def.`): execute the sequential implementation of GPAV (`GPAV.over.examples`);

`ncores` number of cores to use for parallel execution (`def. 8`). Set the parameter `ncores` to 1 if the parameter `parallel` is set to `FALSE`, otherwise set the desired number of cores

`folds` number of folds of the cross validation on which computing the performance metrics averaged across folds (`def. 5`). If `folds=NULL`, the performance metrics are computed one-shot, otherwise the performance metrics are averaged across folds.

`seed` initialization seed for the random generator to create folds (`def. 23`). If `NULL` folds are generated without seed initialization.

`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:

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

`recall.levels` a vector with the desired recall levels (`def: from:0.1, to:0.9, by:0.1`) to compute the the Precision at fixed Recall level (PXR)

`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

`hierScore.dir` relative path where the hierarchical scores matrix must be stored

`perf.dir` relative path where all the performance measures must be stored

Details

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

Value

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

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

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

See Also

GPAV

Examples

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

`Do.heuristic.methods` *Do Heuristic Methods*

Description

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

Usage

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

Arguments

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

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

`norm` boolean value:

- `TRUE` (def.): the flat scores matrix has been already normalized in according to a normalization method;
- `FALSE`: the flat scores matrix has not been normalized yet. See the parameter `norm.type` for which normalization can be applied.

`norm.type` can be one of the following three values:

1. `NULL` (def.): set `norm.type` to `NULL` if and only if the parameter `norm` is set to `TRUE`;
2. `MaxNorm`: each score is divided for the maximum of each class;
3. `Qnorm`: quantile normalization. **preprocessCore** package is used.

`folds` number of folds of the cross validation on which computing the performance metrics averaged across folds (`def. 5`). If `folds=NULL`, the performance metrics are computed one-shot, otherwise the performance metrics are averaged across folds.

`seed` initialization seed for the random generator to create folds (`def. 23`). If `NULL` folds are generated without seed initialization.

`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:

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

recall.levels	a vector with the desired recall levels (def: from:0.1, to:0.9, by:0.1) to compute the the Precision at fixed Recall level (PXR)
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
hierScore.dir	relative path where the hierarchical scores matrix must be stored
perf.dir	relative path where all the performance measures must be stored

Details

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

Value

Two rda files stored in the respective output directories:

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

It is stored in the perf.dir directory.

See Also

Heuristic-Methods

Examples

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

`Do.heuristic.methods.holdout`

Do Heuristic Methods holdout

Description

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

Usage

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

Arguments

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

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

`norm` boolean value:

- TRUE (def.): the flat scores matrix has been already normalized in according to a normalization method;
- FALSE: the flat scores matrix has not been normalized yet. See the parameter norm for which normalization can be applied.

norm.type — can be one of the following three values:

1. NONE (def.): set norm.type to NONE if and only if the parameter norm is set to TRUE;
2. MaxNorm: each score is divided for the maximum of each class;
3. Qnorm: quantile normalization. **preprocessCore** package is used.

folds — number of folds of the cross validation on which computing the performance metrics averaged across folds (def. 5). If folds=NULL, the performance metrics are computed one-shot, otherwise the performance metrics are averaged across folds.

seed — initialization seed for the random generator to create folds (def. 23). If NULL folds are generated without seed initialization.

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:

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

recall.levels — a vector with the desired recall levels (def: from:0.1, to:0.9, by:0.1) to compute the the Precision at fixed Recall level (PXR)

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

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

Details

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

Value

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

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

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

Examples

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

Do.HTD *HTD-DAG vanilla*

Description

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

Usage

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

Arguments

norm	boolean value: • TRUE (def.): the flat scores matrix has been already normalized in according to a normalization method; • FALSE: the flat scores matrix has not been normalized yet. See the parameter norm.type for which normalization can be applied.
norm.type	can be one of the following three values: 1. NULL (def.): set norm.type to NULL if and only if the parameter norm is set to TRUE; 2. MaxNorm: each score is divided for the maximum of each class; 3. Qnorm: quantile normalization. **preprocessCore** package is used.
folds	number of folds of the cross validation on which computing the performance metrics averaged across folds (def. 5). If folds=NULL, the performance metrics are computed one-shot, otherwise the performance metrics are averaged across folds.
seed	initialization seed for the random generator to create folds (def. 23). If NULL folds are generated without seed initialization.
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: 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
recall.levels	a vector with the desired recall levels (def: from:0.1, to:0.9, by:0.1) to compute the the Precision at fixed Recall level (PXR)

`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
`hierScore.dir`	relative path where the hierarchical scores matrix must be stored
`perf.dir`	relative path where all the performance measures must be stored

Details

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

Value

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

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

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

See Also

HTD-DAG

Examples

```
data(graph);
data(scores);
data(labels);
tmpdir <- paste0(tempdir(),"/");
save(g, file=paste0(tmpdir,"graph.rda"));
save(L, file=paste0(tmpdir,"labels.rda"));
```

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

`Do.HTD.holdout` *HTD-DAG holdout*

Description

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

Usage

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

Arguments

`norm`	boolean value: • `TRUE` (def.): the flat scores matrix has been already normalized in according to a normalization method; • `FALSE`: the flat scores matrix has not been normalized yet. See the parameter `norm` for which normalization can be applied.
`norm.type`	can be one of the following three values: 1. `NULL` (def.): set `norm.type` to `NULL` if and only if the parameter `norm` is set to `TRUE`; 2. `MaxNorm`: each score is divided for the maximum of each class; 3. `Qnorm`: quantile normalization. **preprocessCore** package is used.
`folds`	number of folds of the cross validation on which computing the performance metrics averaged across folds (`def. 5`). If `folds=NULL`, the performance metrics are computed one-shot, otherwise the performance metrics are averaged across folds.
`seed`	initialization seed for the random generator to create folds (`def. 23`). If `NULL` folds are generated without seed initialization.

`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: 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
`recall.levels`	a vector with the desired recall levels (`def: from:0.1, to:0.9, by:0.1`) to compute the the Precision at fixed Recall level (PXR)
`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
`hierScore.dir`	relative path where the hierarchical scores matrix must be stored
`perf.dir`	relative path where all the performance measures must be stored

Details

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

Value

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

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

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

See Also

HTD-DAG

Examples

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

---

do.subgraph *Build subgraph*

Description

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

Usage

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

Arguments

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

Value

a subgraph with only the supplied nodes

Examples

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

`do.submatrix` *Build submatrix*

Description

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

Usage

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

Arguments

ann	the 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.unstratified.cv.data`
Unstratified Cross Validation

Description

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

Usage

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

Arguments

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 index of the examples, i.e. the index of the rows of the matrix S

Examples

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

---

example.datasets *Small real example datasets*

Description

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

Usage

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

Details

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

The matrix L contained in the labels data is a 100 X 23 matrix, whose rows 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` *Best hierarchical F-score*

Description

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

Usage

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

Arguments

`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 continuous predicted values (scores): rows correspond to examples and columns to classes
`n.round`	number of rounding digits to be applied to predicted (`default=3`)
`f.criterion`	character. Type of F-measure to be used to select the best F-score. There are two possibilities: 1. `F` (def.) corresponds to the harmonic mean between the average precision and recall; 2. `avF` corresponds to the per-example F-score averaged across all the examples.
`verbose`	boolean. If `TRUE` (def.) the number of iterations are printed on stdout
`b.per.example`	boolean. • `TRUE`: results are returned for each example; • `FALSE`: only the average results are returned

Details

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

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

Value

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

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

Examples

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

---

find.leaves *Leaves*

Description

Find the leaves of a directed graph

Usage

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

Arguments

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

Value

a vector with the names of the leaves of g

Examples

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

FMM *Compute Multilabel F-measure*

Description

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

Usage

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

Arguments

target	matrix with the target 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
n.round	number of rounding digits to be applied to predicted (default=3)
f.criterion	character. Type of F-measure to be used to select the best F-score. There are two possibilities: 1. F (def.) corresponds to the harmonic mean between the average precision and recall; 2. avF corresponds to the per-example F-score averaged across all the examples.
verbose	boolean. If TRUE (def.) the number of iterations are printed on stdout
b.per.example	boolean. • TRUE: results are returned for each example; • FALSE: only the average results are returned

folds — number of folds on which computing the AUROC. If folds=NULL (def.), the AUROC is computed one-shot, otherwise the AUROC is computed averaged across folds.

seed — initialization seed for the random generator to create folds. Set seed only if folds≠NULL. If seed=NULL and folds≠NULL, the AUROC averaged across folds is computed without seed initialization.

Details

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

Value

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

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

Examples

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

full.annotation.matrix *Full 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 adjiacency matrix and columns to terms. $T[i, j] = 1$ means that gene i is annotated for the term j, $T[i, j] = 0$ means that gene i is not annotated for the term j.

See Also

weighted.adjacency.matrix, build.ancestors, specific.annotation.matrix, transitive.closure.annotations

Examples

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

GPAV *Generalized Pool-Adjacent Violators*

Description

Implementation of GPAV (Generalized Pool-Adjacent Violators) algorithm (*Burdavok et al., Journal of Computational Mathematics, 2006* – *link*) 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. GPAV algorithm treats nodes sequentially (in accordance with the topological order of adj) and merges some adjacent nodes into sets which are called blocks. The fitted values for the nodes of the same block are equal to the weighted average value of their response values.

Usage

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

Arguments

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

Details

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

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

Value

a list of 3 elements:

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

See Also

adj.upper.tri

Examples

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

GPAV.over.examples *GPAV Over Examples*

Description

Function to compute GPAV across all the examples

Usage

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

Arguments

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

Value

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

See Also

GPAV.parallel

Examples

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

GPAV.parallel *GPAV Over Examples – Parallel Implementation*

Description

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

Usage

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

Arguments

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

Value

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

Examples

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

graph.levels *Build Graph Levels*

Description

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

Usage

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

Arguments

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

Value

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

Examples

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

HEMDAG-defunct *Defunct functions in package* **HEMDAG**.

Description

The functions listed below are defunct. The alternative function is supplied. Help page for defunct functions is available by typing help("HEMDAG-defunct")

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, root = "00", t = 0.5)

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 the positive descendants (def. t=0.5); whereas in the descens.tau variant the parameter t balances the contribution between the positives children of a node i and that of its positives descendants excluding its positives children
w	weight to balance between the contribution of the node i and that of its positive descendants

Value

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

descens.threshold

For descens.threshold, use TPR.DAG.

descens.threshold.free

For descens.threshold.free, use TPR.DAG.

descens.weighted.threshold.free

For descens.weighted.threshold.free, use TPR.DAG.

descens.weighted.threshold

For descens.weighted.threshold, use TPR.DAG.

descens.tau

For descens.tau, use TPR.DAG.

tpr.threshold

For tpr.threshold, use TPR.DAG.

tpr.threshold.free

For tpr.threshold.free, use TPR.DAG.

tpr.weighted.threshold.free

For tpr.weighted.threshold.free, use TPR.DAG.

`tpr.weighted.threshold`

For `tpr.weighted.threshold`, use `TPR.DAG`.

`Heuristic-Methods` *Obozinski Heuristic Methods*

Description

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

Usage

```
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: $p_i = max_{j \in descendants(i)} \hat{p_j}$;
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: $p_i = \prod_{j \in ancestors(i)} \hat{p_j}$;
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: $1 - p_i = \prod_{j \in descendants(i)} (1 - \hat{p_j})$;

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 cheker*

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 colums are classes

Value

return a list of 3 elements:

- Status:
  - OK if none hierarchical constraints have bee broken;
  - NOTOK if there is at least one hierarchical constraints broken;
- Hierarchy_Constraints_Broken:
  - TRUE: example did not respect the hierarchical constraints;
  - FALSE: example broke the hierarchical constraints;
- Hierarchy_costraints_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:

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

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 and multilabel classification

Usage

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

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

Arguments

`target`	matrix with the target 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 discrete predicted values: rows correspond to examples and columns to classes. $predicted[i,j] = 1$ if example i is predicted belonging to class j, $target[i,j] = 0$ otherwise
`b.per.example`	boolean. • `TRUE`: results are returned for each example; • `FALSE`: only the average results are returned

Details

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

Value

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

- `b.per.example==FALSE`: a list with a single element average. A named vector with average precision (P), recall (R), specificity (S), F-measure (F), average F-measure (avF) and Accuracy (A) across examples. F is the F-measure computed as the harmonic mean between the average precision and recall; av.F is the F-measure computed as the average across examples.

- b.per.example==FALSE: 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

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

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 isthe set of its parents. The nodes are ordered from leaves to root (excluded).

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

See Also

graph.levels

Examples

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

PXR *Precision-Recall Measure*

Description

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

Usage

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

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

Arguments

`labels`	vector of the true labels (0 negative, 1 positive examples)
`scores`	a numeric vector of the values of the predicted labels (scores)
`target`	matrix with the target 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
`folds`	number of folds on which computing the PXR. If `folds=NULL` (`def.`), the PXR is computed one-shot, otherwise the PXR is computed averaged across folds.
`seed`	initialization seed for the random generator to create folds. Set `seed` only if `folds`$\neq$`NULL`. If `seed=NULL` and `folds`$\neq$`NULL`, the PXR averaged across folds is computed without seed initialization.
`recall.levels`	a vector with the desired recall levels (`def: from:0.1`, `to:0.9`, `by:0.1`)

Details

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

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

Value

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

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

Examples

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

---

read.graph *Read a directed graph from a file*

Description

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

Usage

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

Arguments

`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. The extension of the file can be or plain format (".txt") or compressed (".gz").

Value

an object of class `graphNEL`

Examples

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

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

Description

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

Usage

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

Arguments

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

Value

a graph of class graphNEL

Examples

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

root.node *Root node*

Description

Find the root node of a directed graph

Usage

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

Arguments

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

Value

name of the root node

Examples

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

`scores.normalization` *Scores Normalization Function*

Description

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

Usage

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

Arguments

`norm.type`	can be one of the following two values: • MaxNorm (`def.`): each score is divided w.r.t. the max of each class; • Qnorm: a quantile normalization is applied. Library preprocessCore is used.
`S`	a named flat scores matrix with examples on rows and classes on columns

Details

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

Value

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

Examples

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

specific.annotation.list *Specific annotations list*

Description

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

Usage

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

Arguments

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

Value

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

See Also

specific.annotation.matrix

Examples

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

specific.annotation.matrix *HPO specific annotations matrix*

Description

Construct the labels matrix of the most specific HPO terms

Usage

```
specific.annotation.matrix(file = "gene2pheno.txt.gz", 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 repositoty 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.gz", package="HEMDAG");
spec.ann <- specific.annotation.matrix(file=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. kk=5)
seed	seed of the random generator (def. seed=NULL). If is set to NULL no initialization is performed
labels	labels matrix. Rows are genes and columns are classes. Let's denote M the labels matrix. If $M[i,j] = 1$, means that the gene i is annotated with the class j, otherwise $M[i,j] = 0$.

Details

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

Value

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

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

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

Examples

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

TPR-DAG-cross-validation

TPR-DAG cross-validation experiments

Description

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

Usage

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

Arguments

threshold	range of threshold values to be tested in order to find the best threshold (def: from:0.1, to:0.9, by:0.1). The denser the range is, the higher the probability to find the best threshold is, but obviously the execution time will be higher. Use this parameter only for the *thresholded* variants; for the *threshold-free* variant the functions automatically sets threshold to zero
weight	range of weight values to be tested in order to find the best weight (def: from:0.1, to:0.9, by:0.1). The denser the range is, the higher the probability to find the best threshold is, but obviously the execution time will be higher. Use this parameter only for the *weighted* variants; for the other variants the function automatically sets weight to zero
kk	number of folds of the cross validation (def: kk=5) on which tuning the parameters threshold, weight and tau of the parametric variants of the hierarchical ensemble algorithms. For the non-parametric variants (i.e. if bottomup = threshold.free) the function automatically sets kk=NULL if the input kk≠NULL
folds	number of folds of the cross validation on which computing the performance metrics averaged across folds (def. 5). If folds=NULL, the performance metrics are computed one-shot, otherwise the performance metrics are averaged across folds.
seed	initialization seed for the random generator to create folds (def. 23). If NULL folds are generated without seed initialization. The parameter seed controls both the parameter kk and the parameter folds.
norm	boolean value: should the flat scores matrix be normalized?

- TRUE (def.): the flat scores matrix has been already normalized in according to a normalization method;
- FALSE: the flat scores matrix has not been normalized yet. See the parameter norm.type to set the on the fly normalization method to apply among those possible.

norm.type — can be one of the following three values:

1. NULL (def.): set norm.type to NULL if and only if the parameter norm is set to TRUE;
2. MaxNorm: each score is divided for the maximum of each class;
3. Qnorm: quantile normalization. **preprocessCore** package is used.

positive — choice of the *positive* nodes to be considered in the bottom-up strategy. Can be one of the following values:

- children (def.): for each node are considered its positive children;
- descendants: for each node are considered its positive descendants;

bottomup — strategy to enhance the flat predictions by propagating the positive predictions from leaves to root. It can be one of the following values:

- threshold.free (def.): positive nodes are selected on the basis of the threshold.free strategy (def.);
- threshold: positive nodes are selected on the basis of the threshold strategy;
- weighted.threshold.free: positive nodes are selected on the basis of the weighted.threshold.free strategy;
- weighted.threshold: positive nodes are selected on the basis of the weighted.threshold strategy;
- tau: positive nodes are selected on the basis of the tau strategy. NOTE: tau is only a DESCENS variants. If you use tau strategy you must set the parameter positive=descendants;

topdown — strategy to make the scores hierarchy-consistent. It can be one of the following values:

- HTD (def.): HTD-DAG strategy is applied (HTD-DAG);
- GPAV: GPAV strategy is applied (GPAV).

W — vector of weight relative to a single example. If the vector W is not specified (by def. W=NULL), W is a unitary vector of the same length of the columns' number of the flat scores matrix (root node included). Set W only if topdown=GPAV.

parallel — boolean value:

- TRUE: execute the parallel implementation of GPAV (GPAV.parallel);
- FALSE (def.): execute the sequential implementation of GPAV (GPAV.over.examples).

Use parallel if and only if topdown=GPAV; otherwise set parallel=FALSE.

ncores — number of cores to use for parallel execution (def. 8). Set ncores=1 if parallel=FALSE, otherwise set ncores to the desired number of cores. Use ncores only if topdown=GPAV; otherwise set parallel=1.

recall.levels — a vector with the desired recall levels (def: from:0.1, to:0.9, by:0.1) to compute the the Precision at fixed Recall level (PXR)

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: 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
metric	a string character specifying the performance metric on which to maximize the parametric ensemble variant. It can be one of the following values: 1. PRC: the parametric ensemble variant is maximized on the basis of AUPRC (AUPRC); 2. FMAX: the parametric ensemble variant is maximized on the basis of Fmax (Multilabel.F.measure; 3. NULL: on the threshold.free variant none parameter optimization is needed, since the variant is non-parametric. So, if bottomup=threshold.free set metric=NULL (def.).
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
hierScore.dir	relative path where the hierarchical scores matrix must be stored
perf.dir	relative path where the performance measures must be stored

Details

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

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

Value

Two rda files stored in the respective output directories:

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

2. Performance Measures: *flat* and *hierarchical* performance results:
   (a) AUPRC results computed though AUPRC.single.over.classes (AUPRC);
   (b) AUROC results computed through AUROC.single.over.classes (AUROC);
   (c) PXR results computed though precision.at.given.recall.levels.over.classes (PXR);
   (d) FMM results computed though compute.Fmeasure.multilabel (FMM);

It is stored in the perf.dir directory.

See Also

TPR-DAG-variants

Examples

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

TPR-DAG-holdout *TPR-DAG holdout experiments*

Description

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

Usage

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

Arguments

`threshold`	range of threshold values to be tested in order to find the best threshold (`def: from:0.1, to:0.9, by:0.1`). The denser the range is, the higher the probability to find the best threshold is, but obviously the execution time will be higher. Use this parameter only for the *thresholded* variants; for the *threshold-free* variant the functions automatically sets `threshold` to zero
`weight`	range of weight values to be tested in order to find the best weight (`def: from:0.1, to:0.9, by:0.1`). The denser the range is, the higher the probability to find the best threshold is, but obviously the execution time will be higher. Use this parameter only for the *weighted* variants; for the other variants the function automatically sets `weight` to zero
`kk`	number of folds of the cross validation (`def: kk=5`) on which tuning the parameters `threshold`, `weight` and `tau` of the parametric variants of the hierarchical ensemble algorithms. For the non-parametric variants (i.e. if `bottomup = threshold.free`) the function automatically sets `kk=NULL` if the input `kk`$\neq$`NULL`
`folds`	number of folds of the cross validation on which computing the performance metrics averaged across folds (`def. 5`). If `folds=NULL`, the performance metrics are computed one-shot, otherwise the performance metrics are averaged across folds.
`seed`	initialization seed for the random generator to create folds (`def. 23`). If `NULL` folds are generated without seed initialization. The parameter `seed` controls both the parameter `kk` and the parameter `folds`.
`norm`	boolean value: should the flat scores matrix be normalized? • `TRUE` (`def.`): the flat scores matrix has been already normalized in according to a normalization method; • `FALSE`: the flat scores matrix has not been normalized yet. See the parameter `norm.type` to set the on the fly normalization method to apply among those possible.
`norm.type`	can be one of the following three values: 1. `NULL` (`def.`): set `norm.type` to `NULL` if and only if the parameter `norm` is set to `TRUE`; 2. `MaxNorm`: each score is divided for the maximum of each class;

3. Qnorm: quantile normalization. **preprocessCore** package is used.

positive — choice of the *positive* nodes to be considered in the bottom-up strategy. Can be one of the following values:

- children (def.): for each node are considered its positive children;
- descendants: for each node are considered its positive descendants;

bottomup — strategy to enhance the flat predictions by propagating the positive predictions from leaves to root. It can be one of the following values:

- threshold.free (def.): positive nodes are selected on the basis of the threshold.free strategy (def.);
- threshold: positive nodes are selected on the basis of the threshold strategy;
- weighted.threshold.free: positive nodes are selected on the basis of the weighted.threshold.free strategy;
- weighted.threshold: positive nodes are selected on the basis of the weighted.threshold strategy;
- tau: positive nodes are selected on the basis of the tau strategy. NOTE: tau is only a DESCENS variants. If you use tau strategy you must set the parameter positive=descendants;

topdown — strategy to make the scores hierarchy-consistent. It can be one of the following values:

- HTD (def.): HTD-DAG strategy is applied (HTD-DAG);
- GPAV: GPAV strategy is applied (GPAV).

W — vector of weight relative to a single example. If the vector W is not specified (by def. W=NULL), W is a unitary vector of the same length of the columns' number of the flat scores matrix (root node included). Set W only if topdown=GPAV.

parallel — boolean value:

- TRUE: execute the parallel implementation of GPAV (GPAV.parallel);
- FALSE (def.): execute the sequential implementation of GPAV (GPAV.over.examples).

Use parallel if and only if topdown=GPAV; otherwise set parallel=FALSE.

ncores — number of cores to use for parallel execution (def. 8). Set ncores=1 if parallel=FALSE, otherwise set ncores to the desired number of cores. Use ncores if and only if topdown=GPAV; otherwise set parallel=1.

recall.levels — a vector with the desired recall levels (def: from:0.1, to:0.9, by:0.1) to compute the the Precision at fixed Recall level (PXR)

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:

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

metric	a string character specifying the performance metric on which to maximize the parametric ensemble variant. It can be one of the following values: 1. PRC: the parametric ensemble variant is maximized on the basis of AUPRC (AUPRC); 2. FMAX: the parametric ensemble variant is maximized on the basis of Fmax (Multilabel.F.measure); 3. NULL: on the threshold.free variant none parameter optimization is needed, since the variant is non-parametric. So, if bottomup=threshold.free set metric=NULL (def.).
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
hierScore.dir	relative path where the hierarchical scores matrix must be stored
perf.dir	relative path where the performance measures must be stored

Details

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

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

Value

Two rda files stored in the respective output directories:

1. Hierarchical Scores Results: a matrix with examples on rows and classes on columns representing the computed hierarchical scores for each example and for each considered class. It is stored in the hierScore.dir directory.
2. Performance Measures: *flat* and *hierarchical* performace results:
   (a) AUPRC results computed though AUPRC.single.over.classes (AUPRC);
   (b) AUROC results computed through AUROC.single.over.classes (AUROC);

(c) PXR results computed though `precision.at.given.recall.levels.over.classes` (PXR);
(d) FMM results computed though `compute.Fmeasure.multilabel` (FMM);

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

See Also

TPR-DAG-variants

Examples

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

TPR-DAG-variants *TPR-DAG Ensemble Variants*

Description

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

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

Usage

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

Arguments

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")
positive	choice of the *positive* nodes to be considered in the bottom-up strategy. Can be one of the following values: • children (def.): for each node are considered its positive children; • descendants: for each node are considered its positive descendants;
bottomup	strategy to enhance the flat predictions by propagating the positive predictions from leaves to root. It can be one of the following values: • threshold.free (def.): positive nodes are selected on the basis of the threshold.free strategy (def.); • threshold: positive nodes are selected on the basis of the threshold strategy; • weighted.threshold.free: positive nodes are selected on the basis of the weighted.threshold.free strategy; • weighted.threshold: positive nodes are selected on the basis of the weighted.threshold strategy; • tau: positive nodes are selected on the basis of the tau strategy. NOTE: tau is only a DESCENS variants. If you use tau strategy you must set the parameter positive=descendants;
topdown	strategy to make the scores hierarchy-consistent. It can be one of the following values: • HTD (def.): HTD-DAG strategy is applied (HTD-DAG); • GPAV: GPAV strategy is applied (GPAV).
t	threshold for the choice of positive nodes (def. t=0). Set t only for the variants that requiring a threshold for the selection of the positive nodes, otherwise set t to zero
w	weight to balance between the contribution of the node i and that of its positive nodes. Set w only for the *weighted* variants, otherwise set w to zero
W	vector of weight relative to a single example. If the vector W is not specified (by def. W=NULL), W is a unitary vector of the same length of the columns' number of the flat scores matrix (root node included). Set W only if topdown=GPAV.
parallel	boolean value: • TRUE: execute the parallel implementation of GPAV (GPAV.parallel); • FALSE (def.): execute the sequential implementation of GPAV (GPAV.over.examples). Use parallel if and only if topdown=GPAV; otherwise set parallel=FALSE.

ncores — number of cores to use for parallel execution (def. 8). Set ncores=1 if parallel=FALSE, otherwise set ncores to the desired number of cores. Use ncores if and only if topdown=GPAV; otherwise set parallel=1.

Details

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

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

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

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

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

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:

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

   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:

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

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

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

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

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

The contribution of the descendants of a given node decays exponentially with their distance from the node itself. To enhance the contribution of the most specific nodes to the overall decision of the

ensemble we designed a novel variant that we named DESCENS. The novelty of DESCENS consists in strongly considering the contribution of all the descendants of each node instead of only that of its children. Therefore DESCENS predictions are more influenced by the information embedded in the leaves nodes, that are the classes containing the most informative and meaningful information from a biological and medical standpoint. For the choice of the "positive" descendants we use the same strategies adopted for the selection of the "positive" children shown above. Furthermore, we designed a variant specific only for DESCENS, that we named DESCENS-τ. The DESCENS-τ variants balances the contribution between the "positives" children of a node i and that of its "positives" descendants excluding its children by adding a weight $\tau \in [0, 1]$:

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

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

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

Value

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

See Also

GPAV, HTD-DAG

Examples

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

`transitive.closure.annotations`

Transitive closure of annotations

Description

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

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, output.file = "net.file.gz")
```

Arguments

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

Details

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

Value

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

Examples

```
## Not run:
data(wadj);
tupla.matrix(W, output.file="tupla.wadj.gz");
tupla.matrix(W, 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", 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. For instance: nodeX nodeY score. The file extension can be or plain format (".txt") or compressed (".gz").
nodename	boolean value: • TRUE (def.): the names of nodes are gene symbol (i.e. characters); • FALSE: the names of the nodes are entrez gene ID (i.e. integer numbers);

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.gz", package="HEMDAG");
W <- weighted.adjacency.matrix(file=edges, nodename=TRUE);
```

---

write.graph *Write a directed graph on file*

Description

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

Usage

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

Arguments

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

Value

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

Examples

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

Index