HEMDAG Documentation

Marco Notaro, Max Schubach, Giorgio Valentini

Nov 22, 2019

Installation Getting Started

HEMDAG package:

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

CHAPTER 1

Quickstart

This short How-To guides you from downloading the HEMDAG, load it into your R environment and make a first computation.

1.1 Installation

Please goto the *Installation* section and use the *Installation via Conda* option to install HEMDAG.

1.2 Load HEMDAG library

Start R in your console using

```
$ R
```

then load the library by using

```
library("HEMDAG")
```

1.3 Your first classification

We will use the DESCENS algorithm to do some predictions on a DAG (Human Phenotype Ontology).

In contrast to the *vanilla* TPR-DAG version, DESCENS takes into account the contribution of all the descendants of each node instead of only that of its children. So DESCENS predictions are more influenced by the information embedded in the most specific terms of the taxonomy (e.g. leaf nodes), thus putting more emphasis on the terms that most characterize the gene under study.

```
# load a ontology DAG stored in g
data(graph);
# load scores for genes to HPO stored in S
data(scores);
# load labels in L (genes annotated to HPO terms)
data(labels);
# Set the root in your DAG
root <- root.node(g);
# Run DESCENS Threshold free
S.descensTF  <- TPR.DAG(S, g, root, positive="descendants", bottomup="threshold.free",
↪ topdown="HTD");
# Run DESCENS with threshold
S.descensT   <- TPR.DAG(S, g, root, positive="descendants", bottomup="threshold",␣
↪topdown="HTD", t=0.5);
# Run weighted DESCENS with threshold free
S.descensW   <- TPR.DAG(S, g, root, positive="descendants", bottomup="weighted.
↪threshold.free", topdown="HTD", w=0.5);
# Run weighted DESCENS with threshold
S.descensWT  <- TPR.DAG(S, g, root, positive="descendants", bottomup="weighted.
↪threshold", topdown="HTD", t=0.5, w=05);
# Run DESCENS TAU
S.descensTAU <- TPR.DAG(S, g, root, positive="descendants", bottomup="tau", topdown=
↪"HTD", t=0.5);
```

CHAPTER 2

Installation

HEMDAG is available on CRAN as well as through Bioconda and also from source code. You can use one of the following ways for installing HEMDAG.

2.1 Installation via Conda

Note: This is the recommended way of installing for normal users.

This is the recommended way to install HEMDAG because it will enables you to switch software versions easily. And in addition R with all needed dependencies will be installed.

First, you have to install the Miniconda Python3 distribution. See here for installation instructions. Make sure to . . .

- Install the *Python 3* version of Miniconda.
- Answer yes to the question whether conda shall be put into your PATH.

Then, you can install HEMDAG with

```
$ conda install -c bioconda -c conda-forge r-hemdag
```

from the Bioconda channel.

2.2 Global Installation

You can directly install the library via R by issuing

```
$ R -e "install.packages('HEMDAG',repos = 'http://cran.us.r-project.org')"
```

in your terminal. But be sure to install R properly before that command.

Alternatively, you can install the `HEMDAG` library by typing in the `R` environment:

```
install.packages("HEMDAG");
```

Another possibility to install the development version of HEMDAG is by using the devtools package:

```
library(devtools);
install_github("marconotaro/HEMDAG");
```

Before running the above commands be sure to have correctly installed the devtools package (link)

2.3 Installing from Source

This section describes how to build HEMDAG from scratch.

2.3.1 Prequisites

For building HEMDAG, you will need the following dependencies

- R (2.10)
- **R-libraries:**
  - PerfMeas
  - rbgl (bioconductor)
  - graph (bioconductor)
  - precrec
  - preprocessCore (bioconductor)
  - plyr
  - foreach
  - doParallel

2.3.2 Package from CRAN

On a linux environment, download the package source from the CRAN repo and save it in the folder pippo. Then type:

```
R CMD INSTALL pippo/HEMDAG_2.6.0.tar.gz
```

2.3.3 Direct Git Checkout

Note: You only need to install from source if you want to develop HEMDAG yourself.

In this tutorial, we will download the HEMDAG sources and build them in ~/HEMDAG:

```
~ $ cd ~
~ $ git clone https://github.com/marconotaro/HEMDAG.git HEMDAG
```

2.3.4 Building

You can build HEMDAG by using:

```
R CMD build HEMDAG
```

This will generate the file `HEMDAG_2.5.9.tar.gz` and just install the package via:

```
R CMD INSTALL HEMDAG_2.5.9.tar.gz
```

CHAPTER 3

Usage of HEMDAG

For a detailed description of available functions in the HEMDAG package please goto the HEMDAG package on CRAN and have a look to the *Reference manual*.

CHAPTER 4

Tutorial

The hierarchical ensemble methods proposed in HEMDAG package can be run by using any ontology listed in OBO foundry (link). In this tutorial we perform experiments by using the Human Phenotype Ontology (HPO, link) and the Gene Ontology (GO, link).

Note: The experiments run on this tutorial were executed by using the HEMDAG version 2.6.0, the R version 3.6.1 and on a machine having Ubuntu 16.04 as operative system.

4.1 Hierarchical Prediction of HPO terms

Here we show a step-by-step application of HEMDAG to the hierarchical prediction of associations between human gene and abnormal phenotype. To this end we will use the small pre-built dataset available in the HEMDAG library. Nevertheless, you can perform the examples shown below by using the full dataset available at the following link.

Note: By using the full dataset the running time of the parametric ensemble variants is quite higher due to the tuning of the hyper-parameters...

Reminder. To load the HEMDAG library in the R environment, just type:

```
library(HEMDAG);
```

and to load an rda file in the R environment just type:

```
load("file_name.rda");
```

4.2 Loading the Flat Scores Matrix

In their more general form, the hierarchical ensemble methods adopt a two-step learning strategy: the first step consists in the flat learning of the ontology terms, while the second step *reconciles* the flat predictions by considering the topology of the ontology. Hence, the first *ingredient* that we need is the flat scores matrix. For the sake of simplicity, in the examples shown below we make use of the pre-built dataset available in the `HEMDAG` library. To load the flat scores matrix, open the `R` environment and type:

```
data(scores);
```

with the above command we loaded the flat scores matrix `S`, that is a named `100 X 23` matrix. Rows correspond to genes (`Entrez GeneID`) and columns to `HPO` terms/classes. The scores representing the likelihood that a given gene belongs to a given class: the higher the value, the higher the likelihood that a gene belongs to a given class. This flat scores matrix was obtained by running the `RANKS` package (link).

4.3 Loading the DAG

In order to know the hierarchical structure of the `HPO` terms, we must load the graph:

```
data(graph);
```

with the above command we loaded the graph `g`, 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). Nodes of the graph `g` must correspond to classes of the flat scores matrix `S`.

4.3.1 Optional step: plotting the graph `g`

Note: To plot the graph you need to install before the *Rgraphviz* package. Yo can install this library for example by conda (`conda install -c bioconda bioconductor-rgraphviz`) or by Bioconductor (link).

If you want to visualize the *ancestors view* of the term `HP:0100490`, just type:

```
library(Rgraphviz);
plot(g);
```

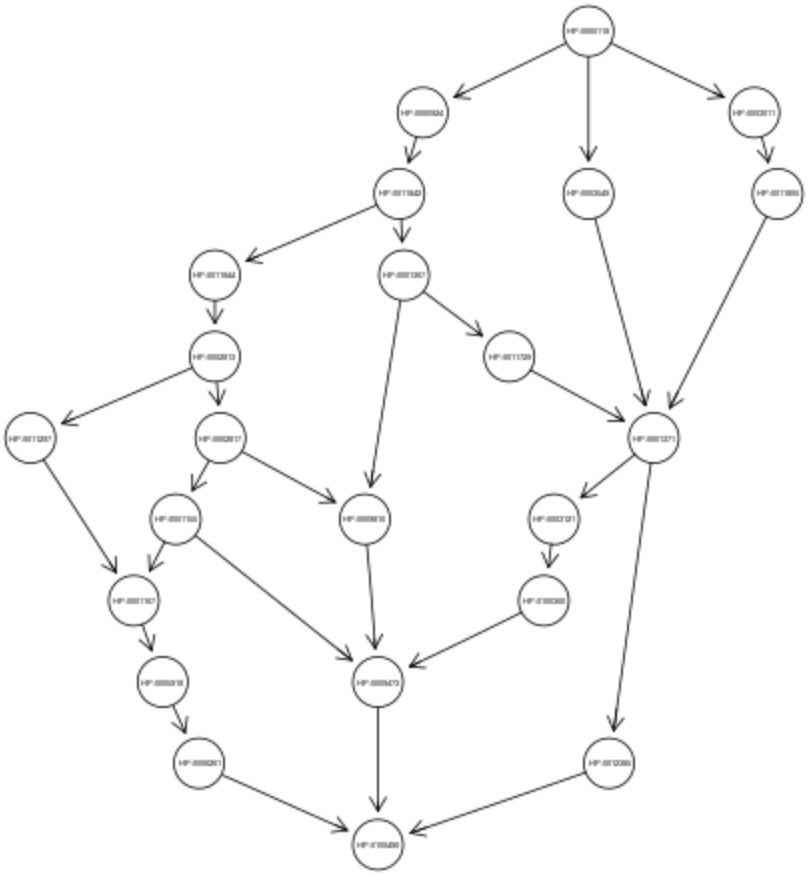
4.4 Scores Normalization

If the flat classifier used as base learner in `HEMDAG` library returns a score and not a probability, we must normalize the scores of the flat matrix to make the flat scores comparable with the hierarchical ones. `HEMDAG` allows to normalize the flat scores according to two different procedures:

1. **MaxNorm**: Normalization in the sense of the maximum: the score of each class is normalized by dividing the score values for the maximum score of that class:

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

2. **Qnorm**: Quantile normalization: quantile normalization of the `preprocessCore` package is used:



```
library(preprocessCore);
qnrom <- normalize.quantiles(S);
```

Be sure to install the `preprocessCore` package before running the above command. Yo can install it by conda (`conda install -c bioconda bioconductor-preprocesscore`) or by Bioconductor (link)

For the examples shown below, we normalize the flat scores matrix by applying the **MaxNorm**:

```
S <- normalize.max(S);
```

4.5 Running Hierarchical Ensemble Methods

First of all, we need to find the root node (i.e. node that is at the top-level of the hierarchy) of the `HPO` graph `g`. To do that just type:

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

in this way we store in the variable `root` the root node of the graph `g`.

Now, we are ready to run any ensemble algorithms implemented in the `HEMDAG` package. Depending on which ensemble variant you want to call, you must execute one of the command listed below:

4.5.1 HTD-DAG: Hierarchical Top Down for DAG

```
S.htd <- htd(S,g,root);
```

4.5.2 GPAV-DAG: Generalized Pool-Adjacent-Violators for DAG

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

4.5.3 TPR-DAG: True Path Rule for DAG

`TPR-DAG` is a family of algorithms according to the bottom-up approach adopted for the choice of the *positive* **children**. In the top-down step (that guarantees coherent predictions with the ontology ones) `TPR-DAG` strategy uses the `HTD-DAG` algorithm.

```
S.tprTF    <- TPR.DAG(S, g, root, positive="children", bottomup="threshold.free",␣
↪topdown="HTD");
S.tprT     <- TPR.DAG(S, g, root, positive="children", bottomup="threshold", topdown=
↪"HTD", t=0.5);
S.tprW     <- TPR.DAG(S, g, root, positive="children", bottomup="weighted.threshold.
↪free", topdown="HTD", w=0.5);
S.tprWT    <- TPR.DAG(S, g, root, positive="children", bottomup="weighted.threshold",␣
↪topdown="HTD", t=0.5, w=0.5);
```

4.5.4 ISO-TPR: Isotonic Regression for DAG

`TPR-DAG` is a family of algorithms according to the bottom-up approach adopted for the choice of the *positive* **children**. To make scores consistent with the ontology predictions `ISO-TPR` employs in the top-down step the `GPAV-DAG` algorithm.

```
S.ISOtprTF <- TPR.DAG(S, g, root, positive="children", bottomup="threshold.free",␣
↪topdown="GPAV");
S.ISOtprT  <- TPR.DAG(S, g, root, positive="children", bottomup="threshold", topdown=
↪"GPAV", t=0.5);
S.ISOtprW  <- TPR.DAG(S, g, root, positive="children", bottomup="weighted.threshold.
↪free", topdown="GPAV", w=0.5);
S.ISOtprWT <- TPR.DAG(S, g, root, positive="children", bottomup="weighted.threshold",␣
↪topdown="GPAV", t=0.5, w=0.5);
```

4.5.5 DESCENS: Descendants Ensemble Classifier

`DESCENS` is a family of algorithms according to the bottom-up approach adopted for the choice of the *positive* **descendants**. In the top-down step `DESCENS` uses the `HTD-DAG` algorithm.

```
S.descensTF      <- TPR.DAG(S, g, root, positive="descendants", bottomup="threshold.
↪free", topdown="HTD");
S.descensT       <- TPR.DAG(S, g, root, positive="descendants", bottomup="threshold",␣
↪topdown="HTD", t=0.5);
S.descensW       <- TPR.DAG(S, g, root, positive="descendants", bottomup="weighted.
↪threshold.free", topdown="HTD", w=0.5);
S.descensWT      <- TPR.DAG(S, g, root, positive="descendants", bottomup="weighted.
↪threshold", topdown="HTD", t=0.5, w=05);
S.descensTAU     <- TPR.DAG(S, g, root, positive="descendants", bottomup="tau",␣
↪topdown="HTD", t=0.5);
S.ISOdescensTF   <- TPR.DAG(S, g, root, positive="descendants", bottomup="threshold.
↪free", topdown="GPAV");
S.ISOdescensT    <- TPR.DAG(S, g, root, positive="descendants", bottomup="threshold",␣
↪topdown="GPAV", t=0.5);
S.ISOdescensW    <- TPR.DAG(S, g, root, positive="descendants", bottomup="weighted.
↪threshold.free", topdown="GPAV", w=0.5);
S.ISOdescensWT   <- TPR.DAG(S, g, root, positive="descendants", bottomup="weighted.
↪threshold", topdown="HTD", t=0.5, w=05);
S.ISOdescensTAU <- TPR.DAG(S, g, root, positive="descendants", bottomup="tau",␣
↪topdown="GPAV", t=0.5);
```

4.5.6 ISO-DESCENS: Isotonic Regression with Descendants Ensemble Classifier

`ISO-DESCENS` is a family of algorithms according to the bottom-up approach adopted for the choice of the *positive* **descendants**. For the top-down step `ISO-DESCENS` employs the `GPAV-DAG` algorithm.

```
S.ISOdescensTF  <- TPR.DAG(S, g, root, positive="descendants", bottomup="threshold.
↪free", topdown="GPAV");
S.ISOdescensT   <- TPR.DAG(S, g, root, positive="descendants", bottomup="threshold",␣
↪topdown="GPAV", t=0.5);
S.ISOdescensW   <- TPR.DAG(S, g, root, positive="descendants", bottomup="weighted.
↪threshold.free", topdown="GPAV", w=0.5);
S.ISOdescensWT  <- TPR.DAG(S, g, root, positive="descendants", bottomup="weighted.
↪threshold", topdown="HTD", t=0.5, w=05);
```

(continues on next page)

(continued from previous page)

```
S.ISOdescensTAU <- TPR.DAG(S, g, root, positive="descendants", bottomup="tau",␣
↪topdown="GPAV", t=0.5);
```

4.5.7 Obozinski Heuristic Methods

```
S.max <- heuristic.max(S,g,root);
S.and <- heuristic.and(S,g,root);
S.or  <- heuristic.or(S,g,root);
```

4.5.8 Hierarchical Constraints Check

The predictions returned by our ensemble methods always obey to the **True Path Rule**: positive instance for a class implies positive instance for all the ancestors of that class. To check this fact we can apply the function `check.hierarchy`:

```
check.hierarchy(S,g,root)$Status
[1] "NOTOK"

check.hierarchy(S.htd,g,root)$Status
[1] "OK"
```

Obviously, all the ensemble variants hold this property, for instance:

```
check.hierarchy(S.tprTF,g,root)$Status
[1] "OK"

check.hierarchy(S.descensW,g,root)$Status
[1] "OK"
```

4.5.9 Performance Evaluation

To know the ensemble methods behavior, the `HEMDAG` library, by using `precrec` package, provides several performance metrics:

- `AUROC`: area under the `ROC` curve;
- `AUPRC`: area under the precision-recall curve;
- `F-max`: maximum hierarchical `F-score` [Jiang2016];
- `PXR` : precision at different recall levels;

Note: `HEMDAG` allows to compute all the aforementioned performance metrics either **one-shot** or **averaged** across `k` fold. Depending on the size of your dataset, the metrics `F-max` and `PXR` could take a while to finish. Please refer to `HEMDAG` reference manual for further information about what these functions receive in input and return in output.

Loading the Annotation Matrix

To compare the hierarchical ensemble methods against the flat approach, we need of the annotation matrix:

```
data(labels);
```

with the above command we loaded the annotations table `L`, that is a named `100 X 23` matrix. Rows correspond to genes (`Entrez GeneID`) and columns to `HPO` terms/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`.

Flat vs Hierarchical

Before computing performance metrics we must remove the root node from the annotation matrix, the flat scores matrix and the hierarchical scores matrix. It does not make sense at all to take into account the predictions of the root node, since it is a *dummy* node added to the ontology for practical reasons (e.g. some graph-based software may require a single root node to work). In `R` this can be accomplished in one line of code.

```
## remove root node from annotation matrix
if(root %in% colnames(L))
    L <- L[,-which(colnames(L)==root)];

## remove root node from flat scores matrix
if(root %in% colnames(S))
    S <- S[,-which(colnames(S)==root)];

## remove root node from hierarchical scores matrix (eg S.htd)
if(root %in% colnames(S.htd))
    S.htd <- S.htd[,-which(colnames(S.htd)==root)];
```

Now we can compare the flat approach `RANKS` versus e.g. `HTD-DAG` by averaging the performance across `3` folds:

```
## FLAT
PRC.flat <- AUPRC.single.over.classes(L, S, folds=3, seed=1);
AUC.flat <- AUROC.single.over.classes(L, S, folds=3, seed=1);
PXR.flat <- precision.at.given.recall.levels.over.classes(L, S, recall.
↪levels=seq(from=0.1, to=1, by=0.1), folds=3, seed=1);
FMM.flat <- compute.Fmeasure.multilabel(L, S, n.round=3, f.criterion="F",␣
↪verbose=FALSE, b.per.example=TRUE, folds=3, seed=1);

## HIERARCHICAL
PRC.hier <- AUPRC.single.over.classes(L, S.htd, folds=3, seed=1);
AUC.hier <- AUROC.single.over.classes(L, S.htd, folds=3, seed=1);
PXR.hier <- precision.at.given.recall.levels.over.classes(L, S.htd, recall.
↪levels=seq(from=0.1, to=1, by=0.1), folds=3, seed=1);
FMM.hier <- compute.Fmeasure.multilabel(L, S.htd, n.round=3, f.criterion="F",␣
↪verbose=FALSE, b.per.example=TRUE, folds=3, seed=1);
```

By looking at the results we can see that `HTD-DAG` outperforms the flat classifier `RANKS`:

```
## AUC performance: flat vs hierarchical
 AUC.flat$average
 [1] 0.8263
 AUC.hier$average
 [1] 0.8312

 ## PRC performance: flat vs hierarchical
 PRC.flat$average
 [1] 0.4373
 PRC.hier$average
 [1] 0.4827
```

(continues on next page)

(continued from previous page)

```
## F-score performance: flat vs hierarchical
FMM.flat$average
   P      R      S      F      avF    A      T
0.7071 0.6443 0.6853 0.6743 0.5768 0.7314 0.7020
FMM.hier$average
   P      R      S      F      avF    A      T
0.5087 0.9394 0.4430 0.6600 0.5922 0.6570 0.4457

## Precision at different recall levels: flat vs hierarchical
PXR.flat$avgPXR
   0.1    0.2    0.3    0.4    0.5    0.6    0.7    0.8    0.9    1
0.5872 0.5872 0.5872 0.5715 0.5715 0.4487 0.4361 0.4361 0.4361 0.4361
PXR.hier$avgPXR
   0.1    0.2    0.3    0.4    0.5    0.6    0.7    0.8    0.9    1
0.6465 0.6465 0.6465 0.6227 0.6227 0.4996 0.4897 0.4897 0.4897 0.4897
```

Note: HTD-DAG is the simplest ensemble approach among those available. HTD-DAG strategy makes flat scores consistent with the hierarchy by propagating from to top to the bottom of the hierarchy the negative predictions. Hence, in the worst case might happen that the predictions at leaves nodes are all negatives. Other ensemble variants (such as GPAV-DAG and TPR-DAG and its variants) lead to better improvements.

4.6 Running Experiments with the Hierarchical Ensemble Methods

The HEMDAG library provides also high-level functions for batch experiments, where input and output data must be stored in compressed rda files. In this way we can run experiment with different ensemble variants by properly changing the arguments of high-level functions implemented in HEMDAG:

1. **Do.HTD**: high-level function to run experiments with HTD-DAG algorithm;
2. **Do.GPAV**: high-level function to run experiments with GPAV-DAG algorithm;
3. **Do.TPR.DAG**: high-level function to run experiments with all TPR-DAG variants;
4. **Do.HTD.holdout**: high-level function to run hold-out experiment with HTD-DAG algorithm;
5. **Do.GPAV.holdout**: high-level function to run hold-out experiment with GPAV-DAG algorithm;
6. **Do.TPR.DAG.holdout**: high-level function to run hold-out experiment with all TPR-DAG variants;

The normalization can be applied on-the-fly within the ensemble high-level function or can be pre-computed through the function Do.flat.scores.normalization. Please have a look to the reference manual for further details on this function.

4.6.1 Cross-Validated Experiments

Here we perform several experiments by using the high-level functions, which provide an user-friendly interface to facilitate the execution of hierarchical ensemble methods.

Data Preparation

For the following experiments we store the input data (i.e. the flat scores matrix `S`, the graph `g` and the annotation table `L`) in the directory `data` and the output data (i.e. the hierarchical scores matrix and the performances) in the folder `results`:

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

if(!dir.exists("data"))
    dir.create("data");

if(!dir.exists("results"))
    dir.create("results");

# store data
save(g,file="data/graph.rda");
save(L,file="data/labels.rda");
save(S,file="data/scores.rda");
```

HTD-DAG Experiments

Here we perform exactly the same experiment that we did above, but using this time the high-level `Do.HTD` to compute the `HTD-DAG` algorithm:

```
Do.HTD( norm=FALSE, norm.type="MaxNorm", folds=3, seed=1, n.round=3, f.criterion="F",
        recall.levels=seq(from=0.1, to=1, by=0.1), flat.file="scores", ann.file=
↪"labels",
        dag.file="graph", flat.dir="data/", ann.dir="data/", dag.dir="data/",
        hierScore.dir="results/", perf.dir="results/", compute.performance=TRUE);
```

Obviously the results returned by `Do.HTD` are identical to those obtained by the step-by-step experiment performed above:

```
load("results/PerfMeas.MaxNorm.scores.hierScores.HTD.rda");

## AUC performance: flat vs hierarchical
AUC.flat$average
[1] 0.8263
AUC.hier$average
[1] 0.8312

## PRC performance: flat vs hierarchical
PRC.flat$average
[1] 0.4373
PRC.hier$average
[1] 0.4827

## F-score performance: flat vs hierarchical
FMM.flat$average
   P      R      S      F      avF    A      T
0.7071 0.6443 0.6853 0.6743 0.5768 0.7314 0.7020
FMM.hier$average
   P      R      S      F      avF    A      T
```

(continues on next page)

(continued from previous page)

```
0.5087 0.9394 0.4430 0.6600 0.5922 0.6570 0.4457

## Precision at different recall levels: flat vs hierarchical
PXR.flat$avgPXR
   0.1    0.2    0.3    0.4    0.5    0.6    0.7    0.8    0.9    1
0.5872 0.5872 0.5872 0.5715 0.5715 0.4487 0.4361 0.4361 0.4361 0.4361
PXR.hier$avgPXR
   0.1    0.2    0.3    0.4    0.5    0.6    0.7    0.8    0.9    1
0.6465 0.6465 0.6465 0.6227 0.6227 0.4996 0.4897 0.4897 0.4897 0.4897
```

Note: All the high-level functions running ensemble-based algorithms automatically remove the root node from the annotation matrix, the flat and the hierarchical scores matrix before computing performance metrics.

GPAV-DAG Experiments

Burdakov et al. in [Burdakov06] proposed an approximate algorithm, named GPAV, to solve the *isotonic regression* (IR) or *monotonic regression* (MR) problem in its general case (i.e. partial order of the constraints). GPAV algorithm combines both low computational complexity (estimated to be $\mathcal{O}(|V|^2)$) and high accuracy.

To run experiments with GPAV-DAG we must type, for instance:

```
Do.GPAV( norm=FALSE, norm.type= "MaxNorm", W=NULL, parallel=TRUE, ncores=7,
         folds=3, seed=1, n.round=3, f.criterion ="F", flat.file="scores",
         recall.levels=seq(from=0.1, to=1, by=0.1), ann.file="labels",
         dag.file="graph", flat.dir="data/", ann.dir="data/", dag.dir="data/",
         hierScore.dir="results/", perf.dir="results/", compute.performance=TRUE);
```

By loading the GPAV-DAG performance results we can see that this ensemble variant outperforms the flat classifier RANKS:

```
load("results/PerfMeas.MaxNorm.scores.hierScores.GPAV.rda");

## AUC performance: flat vs hierarchical
AUC.flat$average
[1] 0.8263
AUC.hier$average
[1] 0.8438

## PRC performance: flat vs hierarchical
PRC.flat$average
[1] 0.4373
PRC.hier$average
[1] 0.5201

## F-score performance: flat vs hierarchical
FMM.flat$average
     P      R      S      F    avF      A      T
0.7071 0.6443 0.6853 0.6743 0.5768 0.7314 0.7020
FMM.hier$average
     P      R      S      F    avF      A      T
0.5893 0.8581 0.5311 0.6988 0.6009 0.6983 0.5360

## Precision at different recall levels: flat vs hierarchical
```

(continues on next page)

(continued from previous page)

```
PXR.flat$avgPXR
   0.1    0.2    0.3    0.4    0.5    0.6    0.7    0.8    0.9      1
0.5872 0.5872 0.5872 0.5715 0.5715 0.4487 0.4361 0.4361 0.4361 0.4361
PXR.hier$avgPXR
   0.1    0.2    0.3    0.4    0.5    0.6    0.7    0.8    0.9      1
0.6976 0.6976 0.6976 0.6835 0.6835 0.5214 0.5005 0.5005 0.5005 0.5005
```

TPR-DAG and ISO-TPR experiments

TPR-DAG is a family of algorithms in according to the chosen bottom-up and top-down approach. There are both parametric and non-parametric variants. To change variant is sufficient to modify the argument of the following parameters of the Do.TPR.DAG high-level function:

- threshold;
- weight;
- positive;
- topdown;
- bottomup;

Please refer to the reference manual for further details about these parameters.

By replacing GPAV-DAG with HTD-DAG in the top-down step of TPR-DAG (variable topdown), we design the ISO-TPR algorithm. The most important feature of ISO-TPR is that it maintains the hierarchical constraints by construction and selects the closest solution (in the least square sense) to the bottom-up predictions that obey the *true path rule*, the logical and biological rule that govern the bio-ontology, such as HPO and GO.

Below we perform several experiments by playing with different TPR-DAG and ISO-TPR ensemble variants. In all the experiments, the performances were averaged across 3 folds.

Note: In Do.TPR-DAG high-level function the parameter kk refers to the number of folds of the cross validation on which tuning the parameters of the *parametric* variants of the hierarchical ensemble algorithms, whereas the parameter folds refers to number of folds of the cross validation on which computing the performance metrics averaged across folds. For the non-parametric variants (i.e. if bottomup = threshold.free), Do.TPR-DAG automatically set to zero the parameters kk and folds.

---

1. ISOtprT: flat scores matrix normalized by MaxNorm, *positive* **children** selection (normalizing the threshold on AUPRC (PRC) across 5 folds, parameter kk=5) and by applying GPAV-DAG strategy in the top-down step

```
Do.TPR.DAG( threshold=seq(0.1,0.9,0.1), weight=0, kk=5, folds=3, seed=1, norm=FALSE,
            norm.type="MaxNorm", positive="children", bottomup="threshold", topdown=
↪"GPAV",
            n.round=3, f.criterion="F", metric="PRC", recall.levels=seq(from=0.1,␣
↪to=1, by=0.1),
            flat.file="scores", ann.file="labels", dag.file="graph", flat.dir="data/",
            ann.dir="data/", dag.dir="data/", hierScore.dir="results/",
            perf.dir="results/", compute.performance=TRUE);
```

2. ISOdescensTF: flat scores matrix normalized by MaxNorm, *positive* **descendants** selection (without threshold) and by applying GPAV-DAG strategy in the top-down step

```
Do.TPR.DAG( threshold=0, weight=0, kk=NULL, folds=3, seed=73, norm=FALSE, norm.type=
↪"MaxNorm",
            positive="descendants", bottomup="threshold.free", topdown="GPAV", n.
↪round=3,
            f.criterion="F", metric=NULL, recall.levels=seq(from=0.1, to=1, by=0.1),
            flat.file="scores", ann.file="labels", dag.file="graph", flat.dir="data/",
            ann.dir="data/", dag.dir="data/", hierScore.dir="results/",
            perf.dir="results/", compute.performance=TRUE);
```

3. `ISOdescensTAU`: flat scores matrix normalized by `Qnorm`, *positive* **descendants** selection (maximizing the threshold on the `F-score` (`FMAX`) across `5` folds, parameter `kk=5`)and by applying `GPAV-DAG` strategy in the top-down step

```
Do.TPR.DAG( threshold=seq(0.1,0.9,0.1), weight=0, kk=5, folds=3, seed=1, norm=FALSE,
            norm.type="Qnorm", positive="descendants", bottomup="tau", topdown="GPAV",
            n.round=3, f.criterion="F", metric="FMAX", flat.file="scores",
            ann.file="labels", recall.levels=seq(from=0.1, to=1, by=0.1),
            dag.file="graph", flat.dir="data/", ann.dir="data/", dag.dir="data/",
            hierScore.dir="results/", perf.dir="results/", compute.performance=TRUE);
```

4. `tprT`: flat scores matrix normalized by `Qnorm`, *positive* **children** selection (maximizing the threshold on the `F-score` (`FMAX`) across `5` folds, parameter `kk=5`) and by applying `HTD-DAG` strategy in the top-down step

```
Do.TPR.DAG( threshold=seq(0.1,0.9,0.1), weight=0, kk=5, folds=3, seed=1, norm=FALSE,
            norm.type="Qnorm", positive="children", bottomup="threshold", topdown="HTD
↪",
            n.round=3, f.criterion="F", metric="FMAX", recall.levels=seq(from=0.1,␣
↪to=1, by=0.1),
            flat.file="scores", ann.file="labels", dag.file="graph", flat.dir="data/",
            ann.dir="data/", dag.dir="data/", hierScore.dir="results/",
            perf.dir="results/", compute.performance=TRUE);
```

5. `descensW`: flat scores matrix normalized by `MaxNorm`,*positive* **descendants** selection (maximizing the weight on the `F-score` (`FMAX`) across `5` folds, parameter `kk=5`) and by applying `HTD-DAG` strategy in the top-down step

```
Do.TPR.DAG( threshold=0, weight=seq(0.1,0.9,0.1), kk=5, folds=3, seed=1, norm=FALSE,
            norm.type="MaxNorm", positive="descendants", bottomup="weighted.threshold.
↪free",
            topdown="GPAV", n.round=3, f.criterion="F", metric="FMAX", flat.file=
↪"scores",
            recall.levels=seq(from=0.1, to=1, by=0.1), ann.file="labels", dag.file=
↪"graph",
            flat.dir="data/", ann.dir="data/", dag.dir="data/", hierScore.dir=
↪"results/",
            perf.dir="results/", compute.performance=TRUE);
```

6. `descensTF`: flat scores matrix normalized by `Qnorm`, *positive* **descendants** selection (without threshold) and by applying `HTD-DAG` strategy in the top-down step

```
Do.TPR.DAG( threshold=0, weight=0, kk=NULL, folds=3, seed=1, norm=FALSE,
            norm.type="Qnorm", positive="descendants", bottomup="threshold.free",
            topdown="HTD", n.round=3, f.criterion="F", metric=NULL, flat.file="scores
↪",
            recall.levels=seq(from=0.1, to=1, by=0.1), ann.file="labels", dag.file=
↪"graph",
            flat.dir="data/", ann.dir="data/", dag.dir="data/", hierScore.dir=
↪"results/",
```

(continues on next page)

(continued from previous page)

```
            perf.dir="results/", compute.performance=TRUE);
```

For instance, by loading the results of the `ISOtprT`, we can see that also this variant improves upon `RANKS` performances:

```
load("results/PerfMeas.MaxNorm.scores.hierScores.ISOtprT.rda");

## AUC performance: flat vs hierarchical
AUC.flat$average
[1] 0.8263
AUC.hier$average
[1] 0.8446

## PRC performance: flat vs hierarchical
PRC.flat$average
[1] 0.4373
PRC.hier$average
[1] 0.5485

## F-score performance: flat vs hierarchical
FMM.flat$average
   P      R      S      F      avF    A      T
0.7071 0.6443 0.6853 0.6743 0.5768 0.7314 0.7020
FMM.hier$average
   P      R      S      F      avF    A      T
0.6007 0.8747 0.5261 0.7122 0.6224 0.7025 0.5827

## Precision at different recall levels: flat vs hierarchical
PXR.flat$avgPXR
   0.1    0.2    0.3    0.4    0.5    0.6    0.7    0.8    0.9    1
0.5872 0.5872 0.5872 0.5715 0.5715 0.4487 0.4361 0.4361 0.4361 0.4361
PXR.hier$avgPXR
   0.1    0.2    0.3    0.4    0.5    0.6    0.7    0.8    0.9    1
0.7043 0.7043 0.7043 0.6876 0.6876 0.5401 0.5219 0.5219 0.5219 0.5219
```

Obozinski Heuristic Methods experiments

`HEMDAG` implements also three heuristics ensemble methods (`AND`, `MAX`, `OR`) proposed in [Obozinski08]. Experiments with these variants can be performed exactly in the same way as done above. Please see the high-level function `Do.heuristic.methods` in the reference manual to further details about how to run experiments with the Obozinski's heuristic ensemble-variants.

4.6.2 Hold-out Experiments

`HEMDAG` library allows to do also classical hold-out experiments. Respect to the cross-validated experiments performed above, we only need to load the indices of the examples to be used in the test set:

```
data(test.index);
save(test.index, file="data/test.index.rda");
```

Now we can perform hold-out experiments. In all the experiments shown below, the performances were computed one-shot (`folds=NULL`). We store the results in the directory `results_ho`:

```
if(!dir.exists("results_ho"))
    dir.create("results_ho");
```

HTD-DAG Experiments: Hold-out Version

```
Do.HTD.holdout( norm=FALSE, norm.type="MaxNorm", n.round=3, f.criterion ="F",␣
↪folds=NULL,
                seed=NULL, recall.levels=seq(from=0.1, to=1, by=0.1), flat.file=
↪"scores",
                ann.file="labels", dag.file="graph", flat.dir="data/", ann.dir="data/
↪",
                dag.dir="data/", ind.test.set="test.index", ind.dir="data/",
                hierScore.dir="results_ho/", perf.dir="results_ho/",
                compute.performance=TRUE);
```

By looking at the performances we can see that HTD-DAG outperforms RANKS:

```
load("results_ho/PerfMeas.MaxNorm.scores.hierScores.HTD.rda");

## AUC performance: flat vs hierarchical
AUC.flat$average
[1] 0.8621
AUC.hier$average
[1] 0.8997

## PRC performance: flat vs hierarchical
PRC.flat$average
[1] 0.2789
PRC.hier$average
[1] 0.4504

## F-score performance: flat vs hierarchical
FMM.flat$average
   P      R      S      F      avF    A      T
0.5952 0.8182 0.4190 0.6891 0.6404 0.7424 0.3770
FMM.hier$average
   P      R      S      F      avF    A      T
0.5589 0.9444 0.2824 0.7023 0.6506 0.6818 0.3590

## Precision at different recall levels: flat vs hierarchical
PXR.flat$avgPXR
   0.1    0.2    0.3    0.4    0.5    0.6    0.7    0.8    0.9    1
0.4424 0.4424 0.4424 0.4379 0.4379 0.3708 0.3621 0.3621 0.3621 0.3621
PXR.hier$avgPXR
   0.1    0.2    0.3    0.4    0.5    0.6    0.7    0.8    0.9    1
0.6629 0.6629 0.6629 0.6174 0.6174 0.4698 0.4547 0.4547 0.4547 0.4547
```

GPAV-DAG Experiments: Hold-out Version

```
Do.GPAV.holdout( norm=FALSE, norm.type="MaxNorm", n.round=3, f.criterion ="F",␣
↪folds=NULL,
                 seed=NULL, recall.levels=seq(from=0.1, to=1, by=0.1), flat.file=
↪"scores",
                 ann.file="labels", dag.file="graph", flat.dir="data/", ann.dir="data/
↪",
```

(continues on next page)

(continued from previous page)

```
                dag.dir="data/", ind.test.set="test.index", ind.dir="data/",
                hierScore.dir="results_ho/", perf.dir="results_ho/",
                compute.performance=TRUE);
```

By looking at the performances we can see that GPAV-DAG outperforms the flat classifier RANKS:

```
load("results_ho/PerfMeas.MaxNorm.scores.hierScores.GPAV.rda");

## AUC performance: flat vs hierarchical
AUC.flat$average
[1] 0.8621
AUC.hier$average
[1] 0.8925

## PRC performance: flat vs hierarchical
PRC.flat$average
[1] 0.2789
PRC.hier$average
[1] 0.3427

## F-score performance: flat vs hierarchical
FMM.flat$average
   P      R      S      F      avF    A      T
0.5952 0.8182 0.4190 0.6891 0.6404 0.7424 0.3770
FMM.hier$average
   P      R      S      F      avF    A      T
0.6952 0.8889 0.4606 0.7802 0.7239 0.8030 0.4370

## Precision at different recall levels: flat vs hierarchical
PXR.flat$avgPXR
   0.1    0.2    0.3    0.4    0.5    0.6    0.7    0.8    0.9    1
0.4424 0.4424 0.4424 0.4379 0.4379 0.3708 0.3621 0.3621 0.3621 0.3621
PXR.hier$avgPXR
   0.1    0.2    0.3    0.4    0.5    0.6    0.7    0.8    0.9    1
0.5341 0.5341 0.5341 0.5250 0.5250 0.4333 0.4273 0.4273 0.4273 0.4273
```

TPR-DAG and ISO-TPR Experiments: Hold-out Version

Note: Similarly as done in the Do.TPR-DAG also in the hold-out version of the high-level function (Do.TPR-DAG.holdout), the parameter kk refers to the number of folds of the cross validation on which tuning the parameters of the *parametric* variants of the hierarchical ensemble algorithms, whereas the parameter folds refers to number of folds of the cross validation on which computing the performance metrics averaged across folds. For the non-parametric variants (i.e. if bottomup = threshold.free), Do.TPR-DAG.holdout automatically set to zero the parameters kk and folds.

1. descensT: flat scores matrix normalized by MaxNorm, *positive* **descendants** selection (maximizing the threshold on the AUPRC (PRC) across 5 folds, parameters kk=5) and by applying HTD-DAG strategy in the top-down step

```
Do.TPR.DAG.holdout( threshold=seq(0.1,0.9,0.1), weight=0, kk=5, folds=NULL, seed=1,␣
↪norm=FALSE,
                    norm.type="MaxNorm", positive="descendants", bottomup="threshold",
                    topdown="HTD", recall.levels=seq(from=0.1, to=1, by=0.1), n.
↪round=3,
```

(continues on next page)

(continued from previous page)

```
                    f.criterion="F", metric="PRC", flat.file="scores", ann.file=
↪"labels",
                    dag.file="graph", flat.dir="data/", ann.dir="data/", dag.dir=
↪"data/",
                    ind.test.set="test.index", ind.dir="data/", hierScore.dir=
↪"results_ho/",
                    perf.dir="results_ho/", compute.performance=TRUE);
```

2. `ISOdescensT`: flat scores matrix normalized by `MaxNorm`, *positive* **descendants** selection (maximizing the threshold on the `AUPRC` – `PRC` across `5` folds – `kk=5`) and by applying `GPAV-DAG` strategy in the top-down step

```
Do.TPR.DAG.holdout( threshold=seq(0.1,0.9,0.1), weight=0, kk=5, folds=NULL, seed=1,␣
↪norm=FALSE,
                    norm.type="MaxNorm", positive="descendants", topdown="GPAV",
                    bottomup="threshold", n.round=3, recall.levels=seq(from=0.1, to=1,
↪ by=0.1),
                    f.criterion="F", metric="FMAX", flat.file="scores", ann.file=
↪"labels",
                    dag.file="graph", flat.dir="data/", ann.dir="data/", dag.dir=
↪"data/",
                    ind.test.set="test.index", ind.dir="data/", hierScore.dir=
↪"results_ho/",
                    perf.dir="results_ho/", compute.performance=TRUE);
```

For instance, by loading the results of the `descensT` variant, we can see that this ensemble variant improves upon `RANKS` performances:

```
load("results_ho/PerfMeas.MaxNorm.scores.hierScores.descensT.rda");

## AUC performance: flat vs hierarchical
AUC.flat$average
[1] 0.8621
AUC.hier$average
[1] 0.8789

## PRC performance: flat vs hierarchical
PRC.flat$average
[1] 0.2789
PRC.hier$average
[1] 0.5482

## F-score performance: flat vs hierarchical
FMM.flat$average
   P      R      S      F      avF    A      T
0.5952 0.8182 0.4190 0.6891 0.6404 0.7424 0.3770
FMM.hier$average
   P      R      S      F      avF    A      T
0.7481 0.8889 0.5532 0.8125 0.7809 0.8788 0.5510

## Precision at different recall levels: flat vs hierarchical
PXR.flat$avgPXR
   0.1    0.2    0.3    0.4    0.5    0.6    0.7    0.8    0.9    1
0.4424 0.4424 0.4424 0.4379 0.4379 0.3708 0.3621 0.3621 0.3621 0.3621
PXR.hier$avgPXR
   0.1    0.2    0.3    0.4    0.5    0.6    0.7    0.8    0.9    1
0.7538 0.7538 0.7538 0.6932 0.6932 0.4851 0.4796 0.4796 0.4796 0.4796
```

Obozinski Heuristic Methods experiments: Hold-out Version

Hold-out experiments with the three Obozinski heuristic variants can be performed exactly in the same way as done above. Please see the high-level function `Do.heuristic.methods.holdout` in the reference manual to further details about how to run experiments with the Obozinski's heuristic ensemble-variants.

4.7 Hierarchical Prediction of GO terms

Let us show now a step-by-step application of `HEMDAG` to the hierarchical prediction of protein function by using the model organism `DROME` (*D. melanogaster*).

Note: For the sake of space here we show experiments with the ensemble-based hierarchical learning algorithms `GPAV` and `ISO-TPR`. However, any other ensemble-based variants executed for the `HPO`-term prediction and more in general listed in the `HEMDAG` library can be also applied for the `GO`-term prediction.

4.8 Data Description

The data used in the experiments shown below can be downloaded at the following link.

1. `7227_DROME_GO_MF_DAG_STRING_v10.5_20DEC17.rda`: object of class `graphNEL` that represents the hierarchy of terms of the `GO` subontology *Molecular Function* (`MF`). This `DAG` has `1736` nodes (`GO` terms) and `2295` edges (between-term relationships). From the `GO` obo file (December 2017 release) we extracted both the `is_a` and the `part_of` relationships, since it is safe grouping annotations by using both these `GO` relationships.

2. `7227_DROME_GO_MF_ANN_STRING_v10.5_20DEC17.rda`: annotation matrix in which the transitive closure of annotation was performed. Rows correspond to `STRING-ID` and columns to `GO` terms. If `T` represents the annotation table, `i` a protein and `j` a `GO` term, `T[i,j]=1` means that the protein `i` is annotated with the term `j`, `T[i,j]=0` means that protein `i` is not annotated with the term `j`. We downloaded the `GO` labels from the Gene Ontology Annotation (GOA) website (December 2017 release). We extracted just the experimentally supported annotations, i.e. the annotations that are directly supported by experimental evidences. The Experimental Evidence codes used to annotate the proteins are the following: (*i*) Inferred from Experiment (`EXP`); (*ii*) Inferred from Direct Assay (`IDA`); (*iii*) Inferred from Physical Interaction (`IPI`); (*iv*) Inferred from Mutant Phenotype (`IMP`); (*v*) Inferred from Genetic Interaction (`IGI`); (*vi*) Inferred from Expression Pattern (`IEP`). Annotation matrix size: `13702 X 1736`.

3. `Scores.7227.DROME.GO.MF.pearson.100.feature.LogitBoost.5fcv.rda`: flat scores matrix representing the probability (or a score) that a gene product `i` belong to a given functional class `j`. The higher the value, the higher the probability that a protein belongs to a given `GO` terms. This flat scores matrix was obtained by running the caret (Classification And REgression Training) *R* package (link). As flat classifier we used the `LogitBoost` setting the number of boosting iterations to `10` (i.e., we used the default parameter setting). The protein-protein interaction network used to create this flat scores matrix was downloaded from the STRING website (version 10.5). We evaluated the generalization performance of the `LogitBoost` classifier, cross-validating the model on the fourth-fifths of the data (training set) and evaluating the performance on the remaining one-fifths (test data). More precisely we created stratified-folds, that is folds containing the same amount of positives and negatives examples (i.e., proteins). In addition, to reduce the empirical temporal complexity, during the training phase, we selected from the `STRING` network the first `100` top-ranked features by using the classical Pearson's correlation coefficient. The supervised feature selection method was cross-validated in an unbiased way, since we chose the

top-ranked features during the training phase and then we used the selected features in the test phase. However this entails to repeat the feature selection 'on the fly' in each training fold of the cross-validation, with a consequent selection of diverse top-ranked features in every training set. Finally, in order to avoid the prediction of GO terms having too few annotations for a reliable assessment, we considered only those classes having 10 or more annotations, obtaining so a flat scores matrix having 13702 rows (STRING-ID) and 327 columns (GO terms). It is worth noting that by adopting a stratified 5-fold cross-validation and taking into account only those GO terms having more than 10 annotations, we guaranteed to have at least 2 positive instances in each training fold of the cross-validation.

4.9 Running Experiments with the Hierarchical Ensemble Methods

Let us start to play with the ensemble-based hierarchical learning algorithms GPAV and ISO-TPR to predict the protein function of the model organism DROME.

4.10 Loading the Data

We load the input data (i.e. the flat scores matrix S, the graph g and the annotation table ann) and we store them in the directory data. The output data (i.e. the hierarchical scores matrix and the performances) will be store in the folder results:

```
# load input data
load(url("https://raw.githubusercontent.com/marconotaro/HEMDAG/master/docs/data/7227_
↪DROME_GO_MF_DAG_STRING_v10.5_20DEC17.rda"));
load(url("https://raw.githubusercontent.com/marconotaro/HEMDAG/master/docs/data/7227_
↪DROME_GO_MF_ANN_STRING_v10.5_20DEC17.rda"));
load(url("https://raw.githubusercontent.com/marconotaro/HEMDAG/master/docs/data/
↪Scores.7227.DROME.GO.MF.pearson.100.feature.LogitBoost.5fcv.rda"));

if(!dir.exists("data"))
    dir.create("data");

if(!dir.exists("results"))
    dir.create("results");

# store data
save(g,file="data/7227_DROME_GO_MF_DAG_STRING_v10.5_20DEC17.rda");
save(ann,file="data/7227_DROME_GO_MF_ANN_STRING_v10.5_20DEC17.rda");
save(S,file="data/Scores.7227.DROME.GO.MF.pearson.100.feature.LogitBoost.5fcv.rda");
```

4.11 Cross-Validated Experiments

In the same way we carried-out the experiments shown in section *Cross-Validated Experiments* for the prediction of human gene-abnormal phenotype associations, below we perform the experiments for the prediction of functions of DROME proteins by using the Gene Ontology annotations as protein labels. In all the experiments shown below the flat scores matrix was normalized in the sense of the maximum, i.e. the score of each GO term was normalized by dividing the score values for the maximum score of that class (variable `norm.type = MaxNorm`).

Note: All the high-level functions in the HEMDAG library check if the number of classes between the flat scores matrix and the annotation matrix mismatched. If that happen, the number of terms of the annotation matrix is shrunk to the

number of terms of the flat scores matrix and the corresponding subgraph is computed as well. It is assumed that all the nodes of the subgraph are accessible from the root.

First of all, we need to load the `HEMDAG` library and set the path of input files and the directories where to store the results:

```
# loading library
library(HEMDAG);

# setting variables
dag.dir <- flat.dir <- ann.dir <- "data/";
hierScore.dir <- perf.dir <- "results/";
dag.file <- "7227_DROME_GO_MF_DAG_STRING_v10.5_20DEC17";
ann.file <- "7227_DROME_GO_MF_ANN_STRING_v10.5_20DEC17";
flat.file <- "Scores.7227.DROME.GO.MF.pearson.100.feature.LogitBoost.5fcv";
```

4.11.1 GPAV Experiments

Now we can run the `GPAV` high-level function:

```
Do.GPAV( norm=FALSE, norm.type= "MaxNorm", W=NULL, parallel=TRUE, ncores=7,␣
↪folds=NULL,
         seed=NULL, 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, compute.performance=TRUE);
```

By looking at the results it easy to see that the learning algorithm `GPAV` outperforms the flat classifier `LogitBoost`:

```
load("results/PerfMeas.MaxNorm.Scores.7227.DROME.GO.MF.pearson.100.feature.LogitBoost.
↪5fcv.hierScores.GPAV.rda");

## AUC performance: flat vs hierarchical
AUC.flat$average
[1] 0.8211
AUC.hier$average
[1] 0.8552

## PRC performance: flat vs hierarchical
PRC.flat$average
[1] 0.1995
PRC.hier$average
[1] 0.2352

## F-score performance: flat vs hierarchical
FMM.flat$average
   P      R      S      F      avF    A      T
0.4255 0.5515 0.9781 0.4803 0.4055 0.9684 0.1190
FMM.hier$average
   P      R      S      F      avF    A      T
0.4837 0.5582 0.9830 0.5183 0.4398 0.9735 0.1080

## Precision at different recall levels: flat vs hierarchical
PXR.flat$avgPXR
```

(continues on next page)

(continued from previous page)

```
   0.1    0.2    0.3    0.4    0.5    0.6    0.7    0.8    0.9    1
0.4053 0.3349 0.2795 0.2304 0.1839 0.1349 0.0911 0.0597 0.0314 0.0105
PXR.hier$avgPXR
   0.1    0.2    0.3    0.4    0.5    0.6    0.7    0.8    0.9    1
0.4687 0.3896 0.3356 0.2924 0.2352 0.1778 0.1223 0.0797 0.0401 0.0119
```

4.11.2 ISO-TPR Experiments

Here we run some `ISO-TPR` variants:

1. `ISOtprTF`: *positive* **children** selection (without threshold)

```
Do.TPR.DAG( threshold=0, weight=0, kk=NULL, folds=NULL, seed=NULL, norm=FALSE,
            norm.type="MaxNorm", positive="children", bottomup="threshold.free",␣
↪topdown="GPAV",
            W=NULL, parallel=TRUE, ncores=7, n.round=3, f.criterion="F", metric=NULL,
            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, compute.performance=TRUE);
```

By looking at the results we can see that our ensemble-based algorithm `ISOtprTF` outperforms the flat classifier `LogitBoost`:

```
load("results/PerfMeas.MaxNorm.Scores.7227.DROME.GO.MF.pearson.100.feature.LogitBoost.
↪5fcv.hierScores.ISOtprTF.rda");

## AUC performance: flat vs hierarchical
AUC.flat$average
[1] 0.8211
(AUC.hier$average
[1] 0.8544

## PRC performance: flat vs hierarchical
PRC.flat$average
[1] 0.1995
PRC.hier$average
[1] 0.2397

## F-score performance: flat vs hierarchical
FMM.flat$average
   P      R      S      F      avF    A      T
0.4255 0.5515 0.9781 0.4803 0.4055 0.9684 0.1190
FMM.hier$average
   P      R      S      F      avF    A      T
0.4820 0.5652 0.9822 0.5203 0.4413 0.9729 0.1200

## Precision at different recall levels: flat vs hierarchical
PXR.flat$avgPXR
   0.1    0.2    0.3    0.4    0.5    0.6    0.7    0.8    0.9    1
0.4053 0.3349 0.2795 0.2304 0.1839 0.1349 0.0911 0.0597 0.0314 0.0105
PXR.hier$avgPXR
   0.1    0.2    0.3    0.4    0.5    0.6    0.7    0.8    0.9    1
0.4764 0.4025 0.3444 0.2938 0.2383 0.1773 0.1226 0.0794 0.0402 0.0119
```

2. `ISOdescensTF`: *positive* **descendants** selection (without threshold)

```
Do.TPR.DAG( threshold=0, weight=0, kk=NULL, folds=NULL, seed=NULL, norm=FALSE,
            norm.type="MaxNorm", positive="descendants", bottomup="threshold.free",
            topdown="GPAV", W=NULL, parallel=TRUE, ncores=7, n.round=3, f.criterion="F
↪",
            metric=NULL, 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, compute.performance=TRUE);
```

By looking at the results we can see that our ensemble-based algorithm `ISOdescensTF` outperforms the flat classifier `LogitBoost`:

```
load("results/PerfMeas.MaxNorm.Scores.7227.DROME.GO.MF.pearson.100.feature.LogitBoost.
↪5fcv.hierScores.ISOdescensTF.rda");

## AUC performance: flat vs hierarchical
AUC.flat$average
[1] 0.8211
AUC.hier$average
[1] 0.8549

## PRC performance: flat vs hierarchical
PRC.flat$average
[1] 0.1995
PRC.hier$average
[1] 0.2449

## F-score performance: flat vs hierarchical
FMM.flat$average
   P      R      S      F      avF    A      T
0.4255 0.5515 0.9781 0.4803 0.4055 0.9684 0.1190
FMM.hier$average
   P      R      S      F      avF    A      T
0.4798 0.5683 0.9817 0.5203 0.4406 0.9725 0.1200

## Precision at different recall levels: flat vs hierarchical
PXR.flat$avgPXR
   0.1    0.2    0.3    0.4    0.5    0.6    0.7    0.8    0.9    1
0.4053 0.3349 0.2795 0.2304 0.1839 0.1349 0.0911 0.0597 0.0314 0.0105
PXR.hier$avgPXR
   0.1    0.2    0.3    0.4    0.5    0.6    0.7    0.8    0.9    1
0.5023 0.4109 0.3528 0.3027 0.2427 0.1785 0.1226 0.0781 0.0400 0.0118
```

4.12 Hold-out Experiments

For the sake of the space we do not show the hold-out experiments for the `GO` term prediction for the model organism `DROME`, since they can be executed exactly in the same way of the hold-out experiments performed in section *Hold-out Experiments* for the prediction of human gene-`HPO` term associations. All that you need to do is properly set the input files name.

CHAPTER 5

Frequently Asked Questions

5.1 Where are the questions?

Right now, there are no frequently asked questions. Please contact the authors if you have questions.

CHAPTER 6

Cite HEMDAG

If you use this software package please cite our BMC Bioinformatics article:

```
M. Notaro, M. Schubach, P. N. Robinson, and G Valentini.
Prediction of Human Phenotype Ontology terms by means of hierarchical ensemble␣
↪methods.
BMC Bioinformatics, 18(1):449, 2017.
```

CHAPTER 7

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

7.1 Types of Contributions

7.1.1 Report Bugs

Report bugs at https://github.com/marconotaro/HEMDAG/issues

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

7.1.2 Fix Bugs

Look through the Github issues for bugs. If you want to start working on a bug then please write short message on the issue tracker to prevent duplicate work.

7.1.3 Implement Features

Look through the Github issues for features. If you want to start working on an issue then please write short message on the issue tracker to prevent duplicate work.

7.1.4 Write Documentation

HEMDAG could always use more documentation, whether as part of the official HEMDAG docs, in docstrings, or even on the web in blog posts, articles, and such.

HEMDAG uses Sphinx for the user manual (that you are currently reading). See *doc_guidelines* on how the documentation reStructuredText is used. See *doc_setup* on creating a local setup for building the documentation.

7.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/marconotaro/HEMDAG/issues

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

7.2 Documentation Guidelines

For the documentation, please adhere to the following guidelines:

- Put each sentence on its own line, this makes tracking changes through Git SCM easier.
- Provide hyperlink targets, at least for the first two section levels.
- Use the section structure from below.

```
.. heading_1:

=========
Heading 1
=========


.. heading_2:

---------
Heading 2
---------


.. heading_3:

Heading 3
=========


.. heading_4:

Heading 4
---------


.. heading_5:
```

(continues on next page)

(continued from previous page)

```
Heading 5
~~~~~~~~~


.. heading_6:

Heading 6
:::::::::
```

7.3 Documentation Setup

For building the documentation, you have to install the Python program Sphinx. We use conda for that, see *Installation via Conda*

Use the following steps for installing Sphinx and the dependencies for building the HEMDAG documentation:

```
$ cd HEMDAG/docs
$ conda create --name sphinx --file environment.yml
$ source activate sphinx
```

Use the following for building the documentation. If you are not in the sphinx environment (e.g. you uses `source deactivate sphinx`) please activate the virtual environment using `source activate sphinx` Afterwards, you can always use `make html` for building.

```
(sphinx) $ cd HEMDAG/docs
(sphinx) $ make html  # rebuild for changed files only
(sphinx) $ make clean && make html  # force rebuild
```

7.4 Get Started!

Ready to contribute?

1. Fork the *HEMDAG* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/HEMDAG.git
   ```

3. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally.

4. When you're done making your changes, make sure that the build runs through.

   ```
   $ cd docs && make clean && make html
   ```

5. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

7.5 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated.
3. Describe your changes in the `CHANGELOG` file.

CHAPTER 8

Authors

in alphabetical order

- Marco Notaro
- Max Schubach
- Giorgio Valentini

CHAPTER 9

History

```
#### HEMDAG 2.6.0

###### CHANGES
- fixed NAMESPACE notes in CRAN checks;
- added link to the GitHub repository ``obogaf::parser``;
- adjusted link to read the docs;

#### HEMDAG 2.5.9

###### NEW FEATURES
- added ``build.consistent.graph``;

###### CHANGES
- added some warning checks in functions that compute performance metrics;
- improved some graph utility functions;
- improved manual;
- improved tutorial on read the docs -- [link](https://hemdag.readthedocs.io);
- namespace made clearer;
- fixed minor bugs;
- removed defunct functions;

#### HEMDAG 2.4.8

###### CHANGES
- fixed a minor bug in ``Do.GPAV.holdout``;
- improved package description;

#### HEMDAG 2.4.7

###### NEW FEATURES
- fixed degenerate case in ``precision.at.all.recall.levels.single.class`` (labels␣
↪are all negatives/positives);
- fixed degenerate case in ``precision.at.given.recall.levels.over.classes`` (labels␣
↪in a fold are all negatives/positives);
```

(continues on next page)

(continued from previous page)

```
- fixed degenerate case in ``do.stratified.cv.data.single.class`` (sampling of the␣
↪labels with just one positive/negative);
- added input variable ``compute.performance`` to the following high level functions:
    - ``Do.TPR.DAG`` and ``Do.TPR.DAG.holdout``;
    - ``Do.HTD`` and ``Do.HTD.holdout``;
    - ``Do.GPAV`` and ``Do.GPAV.holdout``;
    - ``Do.heuristic.methods`` and ``Do.heuristic.methods.holdout``;

###### CHANGES
- improved manual;

#### HEMDAG 2.3.6

###### NEW FEATURES
- added ``lexicographical.topological.sort``;

###### CHANGES
- fixed minor bugs;
- improved manual;

#### HEMDAG 2.2.5

###### NEW FEATURES
- precision-recall performance computed through ``precrec`` package:
    - added ``precision.at.all.recall.levels.single.class``;
    - ``PXR.at.multiple.recall.levels.over.classes`` substituted with ``precision.at.
↪given.recall.levels.over.classes``;
- improved IO functions: the extension of the input or output file can be or plain␣
↪text (``.txt``) or compressed (``.gz``);

###### CHANGES
- fixed minor bugs;
- improved manual;

#### HEMDAG 2.2.4

###### CHANGES
- fixed ``CRAN`` Package Check Results: removed unneeded header and define from␣
↪``GPAV C++`` source code

#### HEMDAG 2.2.3

###### NEW FEATURES
- Added ``GPAV`` algorithm (Burdakov et al., *Journal of Computational Mathematics*,␣
↪2006 -- [link](https://doi.org/10.1007/0-387-30065-1_3));
- Embedded ``GPAV`` algorithm in the top-down step of the functions ``TPR.DAG``, ``Do.
↪TPR.DAG`` and ``Do.TPR.DAG.holdout``;
- Some functions have been defunct. To know the defunct functions just typing in the␣
↪R environment: ``help("HEMDAG-defunct")``;

###### CHANGES
- improved manual;

###### AUTHOR
- Added **Alessandro Petrini** as author for his contribution in writing the ``C++``␣
↪code of ``GPAV`` algorithm;
```

(continues on next page)

(continued from previous page)

```
#### HEMDAG 2.1.3

###### CHANGES
- various fixes from 2.1.2

#### HEMDAG 2.1.2

###### NEW FEATURES
- Improved performance metrics:
    - added ``compute.Fmeasure.multilabel``;
    - added ``PXR.at.multiple.recall.levels.over.classes``;
    - all the performance metrics (``AUPRC``, ``AUROC``, ``FMM``, ``PXR``) can be␣
↪computed either **one-shot** or averaged **across folds**;

- Improved the high-level hierarchical ensemble functions:
    - embedded the new performance metric functions;
    - added the parameter ``metric``: maximization by ``FMAX`` or ``PRC`` (see manual␣
↪for further details);
    - added some checkers (warning/stop messages) to make the library more user-
↪friendly;

###### CHANGES
- improved manual;

#### HEMDAG 2.0.1

###### CHANGES
- fixed bug in ``do.stratified.cv.data.single.class``;

#### HEMDAG 2.0.0

###### NEW FEATURES
- Added ``TPR-DAG``: function gathering several hierarchical ensemble variants;
- Added ``Do.TPR.DAG``: high-level function to run ``TPR-DAG`` **cross-validated**␣
↪experiments;
- Added ``Do.TPR.DAG.holdout``: high-level functions to run ``TPR-DAG`` **holdout**␣
↪experiments;

- The following ``TPR-DAG`` and ``DESCENS`` high-level functions were removed:
    - Do.tpr.threshold.free;
    - Do.tpr.threshold.cv;
    - Do.tpr.weighted.threshold.free.cv;
    - Do.tpr.weighted.threshold.cv;
    - Do.descens.threshold.free;
    - Do.descens.threshold.cv;
    - Do.descens.weighted.threshold.free.cv;
    - Do.descens.tau.cv;
    - Do.descens.weighted.threshold.cv;
    - Do.tpr.threshold.free.holdout;
    - Do.tpr.threshold.holdout;
    - Do.tpr.weighted.threshold.free.holdout;
    - Do.tpr.weighted.threshold.holdout;
    - Do.descens.threshold.free.holdout;
    - Do.descens.threshold.holdout;
    - Do.descens.weighted.threshold.free.holdout;
    - Do.descens.tau.holdout;
    - Do.descens.weighted.threshold.holdout;
```

(continues on next page)

(continued from previous page)

```
> NOTE: all the removed functions can be run opportunely setting the input parameters␣
↪of the new high-level function ``Do.TPR.DAG`` (for **cross-validated** experiments)␣
↪and ``Do.TPR.DAG.holdout`` (for **hold-out** experiments);

###### CHANGES
- improved manual;

#### HEMDAG 1.1.1

###### NEW FEATURES
- Added ``DESCENS`` algorithm;
- Added Heuristic Methods ``MAX``, ``AND``, ``OR`` (Obozinski et al., Genome Biology,␣
↪2008 -- [link](https://genomebiology.biomedcentral.com/articles/10.1186/gb-2008-9-
↪s1-s6));
- Added ``tupla.matrix`` function;

###### CHANGES
- improved manual;
- Added link to the GitHub repository ``HPOparser`` (note: from version ``2.6.0``␣
↪``HPOparser`` was changed in ``obogaf::parser``);
- Added ``CITATION`` file;

#### HEMDAG 1.0.0

###### PACKAGE GENESIS
```

CHAPTER 10

HEMDAG License

HEMDAG is licensed under the GNU GPLv3 license:

```
GNU GENERAL PUBLIC LICENSE
                       Version 3, 29 June 2007

 Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                            Preamble

  The GNU General Public License is a free, copyleft license for
software and other kinds of works.

  The licenses for most software and other practical works are designed
to take away your freedom to share and change the works.  By contrast,
the GNU General Public License is intended to guarantee your freedom to
share and change all versions of a program--to make sure it remains free
software for all its users.  We, the Free Software Foundation, use the
GNU General Public License for most of our software; it applies also to
any other work released this way by its authors.  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
them if you wish), that you receive source code or can get it if you
want it, that you can change the software or use pieces of it in new
free programs, and that you know you can do these things.

  To protect your rights, we need to prevent others from denying you
these rights or asking you to surrender the rights.  Therefore, you have
certain responsibilities if you distribute copies of the software, or if
you modify it: responsibilities to respect the freedom of others.
```

(continues on next page)

(continued from previous page)

```
  For example, if you distribute copies of such a program, whether
gratis or for a fee, you must pass on to the recipients the same
freedoms that you received.  You must make sure that they, too, receive
or can get the source code.  And you must show them these terms so they
know their rights.

  Developers that use the GNU GPL protect your rights with two steps:
(1) assert copyright on the software, and (2) offer you this License
giving you legal permission to copy, distribute and/or modify it.

  For the developers' and authors' protection, the GPL clearly explains
that there is no warranty for this free software.  For both users' and
authors' sake, the GPL requires that modified versions be marked as
changed, so that their problems will not be attributed erroneously to
authors of previous versions.

  Some devices are designed to deny users access to install or run
modified versions of the software inside them, although the manufacturer
can do so.  This is fundamentally incompatible with the aim of
protecting users' freedom to change the software.  The systematic
pattern of such abuse occurs in the area of products for individuals to
use, which is precisely where it is most unacceptable.  Therefore, we
have designed this version of the GPL to prohibit the practice for those
products.  If such problems arise substantially in other domains, we
stand ready to extend this provision to those domains in future versions
of the GPL, as needed to protect the freedom of users.

  Finally, every program is threatened constantly by software patents.
States should not allow patents to restrict development and use of
software on general-purpose computers, but in those that do, we wish to
avoid the special danger that patents applied to a free program could
make it effectively proprietary.  To prevent this, the GPL assures that
patents cannot be used to render the program non-free.

  The precise terms and conditions for copying, distribution and
modification follow.

                       TERMS AND CONDITIONS

  0. Definitions.

  "This License" refers to version 3 of the GNU General Public License.

  "Copyright" also means copyright-like laws that apply to other kinds of
works, such as semiconductor masks.

  "The Program" refers to any copyrightable work licensed under this
License.  Each licensee is addressed as "you".  "Licensees" and
"recipients" may be individuals or organizations.

  To "modify" a work means to copy from or adapt all or part of the work
in a fashion requiring copyright permission, other than the making of an
exact copy.  The resulting work is called a "modified version" of the
earlier work or a work "based on" the earlier work.

  A "covered work" means either the unmodified Program or a work based
```

(continues on next page)

(continued from previous page)

```
on the Program.

  To "propagate" a work means to do anything with it that, without
permission, would make you directly or secondarily liable for
infringement under applicable copyright law, except executing it on a
computer or modifying a private copy.  Propagation includes copying,
distribution (with or without modification), making available to the
public, and in some countries other activities as well.

  To "convey" a work means any kind of propagation that enables other
parties to make or receive copies.  Mere interaction with a user through
a computer network, with no transfer of a copy, is not conveying.

  An interactive user interface displays "Appropriate Legal Notices"
to the extent that it includes a convenient and prominently visible
feature that (1) displays an appropriate copyright notice, and (2)
tells the user that there is no warranty for the work (except to the
extent that warranties are provided), that licensees may convey the
work under this License, and how to view a copy of this License.  If
the interface presents a list of user commands or options, such as a
menu, a prominent item in the list meets this criterion.

  1. Source Code.

  The "source code" for a work means the preferred form of the work
for making modifications to it.  "Object code" means any non-source
form of a work.

  A "Standard Interface" means an interface that either is an official
standard defined by a recognized standards body, or, in the case of
interfaces specified for a particular programming language, one that
is widely used among developers working in that language.

  The "System Libraries" of an executable work include anything, other
than the work as a whole, that (a) is included in the normal form of
packaging a Major Component, but which is not part of that Major
Component, and (b) serves only to enable use of the work with that
Major Component, or to implement a Standard Interface for which an
implementation is available to the public in source code form.  A
"Major Component", in this context, means a major essential component
(kernel, window system, and so on) of the specific operating system
(if any) on which the executable work runs, or a compiler used to
produce the work, or an object code interpreter used to run it.

  The "Corresponding Source" for a work in object code form means all
the source code needed to generate, install, and (for an executable
work) run the object code and to modify the work, including scripts to
control those activities.  However, it does not include the work's
System Libraries, or general-purpose tools or generally available free
programs which are used unmodified in performing those activities but
which are not part of the work.  For example, Corresponding Source
includes interface definition files associated with source files for
the work, and the source code for shared libraries and dynamically
linked subprograms that the work is specifically designed to require,
such as by intimate data communication or control flow between those
subprograms and other parts of the work.
```

(continues on next page)

(continued from previous page)

```
  The Corresponding Source need not include anything that users
can regenerate automatically from other parts of the Corresponding
Source.

  The Corresponding Source for a work in source code form is that
same work.

  2. Basic Permissions.

  All rights granted under this License are granted for the term of
copyright on the Program, and are irrevocable provided the stated
conditions are met.  This License explicitly affirms your unlimited
permission to run the unmodified Program.  The output from running a
covered work is covered by this License only if the output, given its
content, constitutes a covered work.  This License acknowledges your
rights of fair use or other equivalent, as provided by copyright law.

  You may make, run and propagate covered works that you do not
convey, without conditions so long as your license otherwise remains
in force.  You may convey covered works to others for the sole purpose
of having them make modifications exclusively for you, or provide you
with facilities for running those works, provided that you comply with
the terms of this License in conveying all material for which you do
not control copyright.  Those thus making or running the covered works
for you must do so exclusively on your behalf, under your direction
and control, on terms that prohibit them from making any copies of
your copyrighted material outside their relationship with you.

  Conveying under any other circumstances is permitted solely under
the conditions stated below.  Sublicensing is not allowed; section 10
makes it unnecessary.

  3. Protecting Users' Legal Rights From Anti-Circumvention Law.

  No covered work shall be deemed part of an effective technological
measure under any applicable law fulfilling obligations under article
11 of the WIPO copyright treaty adopted on 20 December 1996, or
similar laws prohibiting or restricting circumvention of such
measures.

  When you convey a covered work, you waive any legal power to forbid
circumvention of technological measures to the extent such circumvention
is effected by exercising rights under this License with respect to
the covered work, and you disclaim any intention to limit operation or
modification of the work as a means of enforcing, against the work's
users, your or third parties' legal rights to forbid circumvention of
technological measures.

  4. Conveying Verbatim Copies.

  You may convey verbatim copies of the Program's source code as you
receive it, in any medium, provided that you conspicuously and
appropriately publish on each copy an appropriate copyright notice;
keep intact all notices stating that this License and any
non-permissive terms added in accord with section 7 apply to the code;
keep intact all notices of the absence of any warranty; and give all
recipients a copy of this License along with the Program.
```

(continues on next page)

(continued from previous page)

```
  You may charge any price or no price for each copy that you convey,
and you may offer support or warranty protection for a fee.

  5. Conveying Modified Source Versions.

  You may convey a work based on the Program, or the modifications to
produce it from the Program, in the form of source code under the
terms of section 4, provided that you also meet all of these conditions:

    a) The work must carry prominent notices stating that you modified
    it, and giving a relevant date.

    b) The work must carry prominent notices stating that it is
    released under this License and any conditions added under section
    7.  This requirement modifies the requirement in section 4 to
    "keep intact all notices".

    c) You must license the entire work, as a whole, under this
    License to anyone who comes into possession of a copy.  This
    License will therefore apply, along with any applicable section 7
    additional terms, to the whole of the work, and all its parts,
    regardless of how they are packaged.  This License gives no
    permission to license the work in any other way, but it does not
    invalidate such permission if you have separately received it.

    d) If the work has interactive user interfaces, each must display
    Appropriate Legal Notices; however, if the Program has interactive
    interfaces that do not display Appropriate Legal Notices, your
    work need not make them do so.

  A compilation of a covered work with other separate and independent
works, which are not by their nature extensions of the covered work,
and which are not combined with it such as to form a larger program,
in or on a volume of a storage or distribution medium, is called an
"aggregate" if the compilation and its resulting copyright are not
used to limit the access or legal rights of the compilation's users
beyond what the individual works permit.  Inclusion of a covered work
in an aggregate does not cause this License to apply to the other
parts of the aggregate.

  6. Conveying Non-Source Forms.

  You may convey a covered work in object code form under the terms
of sections 4 and 5, provided that you also convey the
machine-readable Corresponding Source under the terms of this License,
in one of these ways:

    a) Convey the object code in, or embodied in, a physical product
    (including a physical distribution medium), accompanied by the
    Corresponding Source fixed on a durable physical medium
    customarily used for software interchange.

    b) Convey the object code in, or embodied in, a physical product
    (including a physical distribution medium), accompanied by a
    written offer, valid for at least three years and valid for as
    long as you offer spare parts or customer support for that product
```

(continues on next page)

(continued from previous page)

```
    model, to give anyone who possesses the object code either (1) a
    copy of the Corresponding Source for all the software in the
    product that is covered by this License, on a durable physical
    medium customarily used for software interchange, for a price no
    more than your reasonable cost of physically performing this
    conveying of source, or (2) access to copy the
    Corresponding Source from a network server at no charge.

    c) Convey individual copies of the object code with a copy of the
    written offer to provide the Corresponding Source.  This
    alternative is allowed only occasionally and noncommercially, and
    only if you received the object code with such an offer, in accord
    with subsection 6b.

    d) Convey the object code by offering access from a designated
    place (gratis or for a charge), and offer equivalent access to the
    Corresponding Source in the same way through the same place at no
    further charge.  You need not require recipients to copy the
    Corresponding Source along with the object code.  If the place to
    copy the object code is a network server, the Corresponding Source
    may be on a different server (operated by you or a third party)
    that supports equivalent copying facilities, provided you maintain
    clear directions next to the object code saying where to find the
    Corresponding Source.  Regardless of what server hosts the
    Corresponding Source, you remain obligated to ensure that it is
    available for as long as needed to satisfy these requirements.

    e) Convey the object code using peer-to-peer transmission, provided
    you inform other peers where the object code and Corresponding
    Source of the work are being offered to the general public at no
    charge under subsection 6d.

  A separable portion of the object code, whose source code is excluded
from the Corresponding Source as a System Library, need not be
included in conveying the object code work.

  A "User Product" is either (1) a "consumer product", which means any
tangible personal property which is normally used for personal, family,
or household purposes, or (2) anything designed or sold for incorporation
into a dwelling.  In determining whether a product is a consumer product,
doubtful cases shall be resolved in favor of coverage.  For a particular
product received by a particular user, "normally used" refers to a
typical or common use of that class of product, regardless of the status
of the particular user or of the way in which the particular user
actually uses, or expects or is expected to use, the product.  A product
is a consumer product regardless of whether the product has substantial
commercial, industrial or non-consumer uses, unless such uses represent
the only significant mode of use of the product.

  "Installation Information" for a User Product means any methods,
procedures, authorization keys, or other information required to install
and execute modified versions of a covered work in that User Product from
a modified version of its Corresponding Source.  The information must
suffice to ensure that the continued functioning of the modified object
code is in no case prevented or interfered with solely because
modification has been made.
```

(continues on next page)

(continued from previous page)

```
  If you convey an object code work under this section in, or with, or
specifically for use in, a User Product, and the conveying occurs as
part of a transaction in which the right of possession and use of the
User Product is transferred to the recipient in perpetuity or for a
fixed term (regardless of how the transaction is characterized), the
Corresponding Source conveyed under this section must be accompanied
by the Installation Information.  But this requirement does not apply
if neither you nor any third party retains the ability to install
modified object code on the User Product (for example, the work has
been installed in ROM).

  The requirement to provide Installation Information does not include a
requirement to continue to provide support service, warranty, or updates
for a work that has been modified or installed by the recipient, or for
the User Product in which it has been modified or installed.  Access to a
network may be denied when the modification itself materially and
adversely affects the operation of the network or violates the rules and
protocols for communication across the network.

  Corresponding Source conveyed, and Installation Information provided,
in accord with this section must be in a format that is publicly
documented (and with an implementation available to the public in
source code form), and must require no special password or key for
unpacking, reading or copying.

  7. Additional Terms.

  "Additional permissions" are terms that supplement the terms of this
License by making exceptions from one or more of its conditions.
Additional permissions that are applicable to the entire Program shall
be treated as though they were included in this License, to the extent
that they are valid under applicable law.  If additional permissions
apply only to part of the Program, that part may be used separately
under those permissions, but the entire Program remains governed by
this License without regard to the additional permissions.

  When you convey a copy of a covered work, you may at your option
remove any additional permissions from that copy, or from any part of
it.  (Additional permissions may be written to require their own
removal in certain cases when you modify the work.)  You may place
additional permissions on material, added by you to a covered work,
for which you have or can give appropriate copyright permission.

  Notwithstanding any other provision of this License, for material you
add to a covered work, you may (if authorized by the copyright holders of
that material) supplement the terms of this License with terms:

    a) Disclaiming warranty or limiting liability differently from the
    terms of sections 15 and 16 of this License; or

    b) Requiring preservation of specified reasonable legal notices or
    author attributions in that material or in the Appropriate Legal
    Notices displayed by works containing it; or

    c) Prohibiting misrepresentation of the origin of that material, or
    requiring that modified versions of such material be marked in
    reasonable ways as different from the original version; or
```

(continues on next page)

(continued from previous page)

```
    d) Limiting the use for publicity purposes of names of licensors or
    authors of the material; or

    e) Declining to grant rights under trademark law for use of some
    trade names, trademarks, or service marks; or

    f) Requiring indemnification of licensors and authors of that
    material by anyone who conveys the material (or modified versions of
    it) with contractual assumptions of liability to the recipient, for
    any liability that these contractual assumptions directly impose on
    those licensors and authors.

  All other non-permissive additional terms are considered "further
restrictions" within the meaning of section 10.  If the Program as you
received it, or any part of it, contains a notice stating that it is
governed by this License along with a term that is a further
restriction, you may remove that term.  If a license document contains
a further restriction but permits relicensing or conveying under this
License, you may add to a covered work material governed by the terms
of that license document, provided that the further restriction does
not survive such relicensing or conveying.

  If you add terms to a covered work in accord with this section, you
must place, in the relevant source files, a statement of the
additional terms that apply to those files, or a notice indicating
where to find the applicable terms.

  Additional terms, permissive or non-permissive, may be stated in the
form of a separately written license, or stated as exceptions;
the above requirements apply either way.

  8. Termination.

  You may not propagate or modify a covered work except as expressly
provided under this License.  Any attempt otherwise to propagate or
modify it is void, and will automatically terminate your rights under
this License (including any patent licenses granted under the third
paragraph of section 11).

  However, if you cease all violation of this License, then your
license from a particular copyright holder is reinstated (a)
provisionally, unless and until the copyright holder explicitly and
finally terminates your license, and (b) permanently, if the copyright
holder fails to notify you of the violation by some reasonable means
prior to 60 days after the cessation.

  Moreover, your license from a particular copyright holder is
reinstated permanently if the copyright holder notifies you of the
violation by some reasonable means, this is the first time you have
received notice of violation of this License (for any work) from that
copyright holder, and you cure the violation prior to 30 days after
your receipt of the notice.

  Termination of your rights under this section does not terminate the
licenses of parties who have received copies or rights from you under
this License.  If your rights have been terminated and not permanently
```

(continues on next page)

(continued from previous page)

```
reinstated, you do not qualify to receive new licenses for the same
material under section 10.

  9. Acceptance Not Required for Having Copies.

  You are not required to accept this License in order to receive or
run a copy of the Program.  Ancillary propagation of a covered work
occurring solely as a consequence of using peer-to-peer transmission
to receive a copy likewise does not require acceptance.  However,
nothing other than this License grants you permission to propagate or
modify any covered work.  These actions infringe copyright if you do
not accept this License.  Therefore, by modifying or propagating a
covered work, you indicate your acceptance of this License to do so.

  10. Automatic Licensing of Downstream Recipients.

  Each time you convey a covered work, the recipient automatically
receives a license from the original licensors, to run, modify and
propagate that work, subject to this License.  You are not responsible
for enforcing compliance by third parties with this License.

  An "entity transaction" is a transaction transferring control of an
organization, or substantially all assets of one, or subdividing an
organization, or merging organizations.  If propagation of a covered
work results from an entity transaction, each party to that
transaction who receives a copy of the work also receives whatever
licenses to the work the party's predecessor in interest had or could
give under the previous paragraph, plus a right to possession of the
Corresponding Source of the work from the predecessor in interest, if
the predecessor has it or can get it with reasonable efforts.

  You may not impose any further restrictions on the exercise of the
rights granted or affirmed under this License.  For example, you may
not impose a license fee, royalty, or other charge for exercise of
rights granted under this License, and you may not initiate litigation
(including a cross-claim or counterclaim in a lawsuit) alleging that
any patent claim is infringed by making, using, selling, offering for
sale, or importing the Program or any portion of it.

  11. Patents.

  A "contributor" is a copyright holder who authorizes use under this
License of the Program or a work on which the Program is based.  The
work thus licensed is called the contributor's "contributor version".

  A contributor's "essential patent claims" are all patent claims
owned or controlled by the contributor, whether already acquired or
hereafter acquired, that would be infringed by some manner, permitted
by this License, of making, using, or selling its contributor version,
but do not include claims that would be infringed only as a
consequence of further modification of the contributor version.  For
purposes of this definition, "control" includes the right to grant
patent sublicenses in a manner consistent with the requirements of
this License.

  Each contributor grants you a non-exclusive, worldwide, royalty-free
patent license under the contributor's essential patent claims, to
```

(continues on next page)

(continued from previous page)

```
make, use, sell, offer for sale, import and otherwise run, modify and
propagate the contents of its contributor version.

  In the following three paragraphs, a "patent license" is any express
agreement or commitment, however denominated, not to enforce a patent
(such as an express permission to practice a patent or covenant not to
sue for patent infringement).  To "grant" such a patent license to a
party means to make such an agreement or commitment not to enforce a
patent against the party.

  If you convey a covered work, knowingly relying on a patent license,
and the Corresponding Source of the work is not available for anyone
to copy, free of charge and under the terms of this License, through a
publicly available network server or other readily accessible means,
then you must either (1) cause the Corresponding Source to be so
available, or (2) arrange to deprive yourself of the benefit of the
patent license for this particular work, or (3) arrange, in a manner
consistent with the requirements of this License, to extend the patent
license to downstream recipients.  "Knowingly relying" means you have
actual knowledge that, but for the patent license, your conveying the
covered work in a country, or your recipient's use of the covered work
in a country, would infringe one or more identifiable patents in that
country that you have reason to believe are valid.

  If, pursuant to or in connection with a single transaction or
arrangement, you convey, or propagate by procuring conveyance of, a
covered work, and grant a patent license to some of the parties
receiving the covered work authorizing them to use, propagate, modify
or convey a specific copy of the covered work, then the patent license
you grant is automatically extended to all recipients of the covered
work and works based on it.

  A patent license is "discriminatory" if it does not include within
the scope of its coverage, prohibits the exercise of, or is
conditioned on the non-exercise of one or more of the rights that are
specifically granted under this License.  You may not convey a covered
work if you are a party to an arrangement with a third party that is
in the business of distributing software, under which you make payment
to the third party based on the extent of your activity of conveying
the work, and under which the third party grants, to any of the
parties who would receive the covered work from you, a discriminatory
patent license (a) in connection with copies of the covered work
conveyed by you (or copies made from those copies), or (b) primarily
for and in connection with specific products or compilations that
contain the covered work, unless you entered into that arrangement,
or that patent license was granted, prior to 28 March 2007.

  Nothing in this License shall be construed as excluding or limiting
any implied license or other defenses to infringement that may
otherwise be available to you under applicable patent law.

  12. No Surrender of Others' Freedom.

  If conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not
excuse you from the conditions of this License.  If you cannot convey a
covered work so as to satisfy simultaneously your obligations under this
```

(continues on next page)

(continued from previous page)

```
License and any other pertinent obligations, then as a consequence you may
not convey it at all.  For example, if you agree to terms that obligate you
to collect a royalty for further conveying from those to whom you convey
the Program, the only way you could satisfy both those terms and this
License would be to refrain entirely from conveying the Program.

  13. Use with the GNU Affero General Public License.

  Notwithstanding any other provision of this License, you have
permission to link or combine any covered work with a work licensed
under version 3 of the GNU Affero General Public License into a single
combined work, and to convey the resulting work.  The terms of this
License will continue to apply to the part which is the covered work,
but the special requirements of the GNU Affero General Public License,
section 13, concerning interaction through a network will apply to the
combination as such.

  14. Revised Versions of this License.

  The Free Software Foundation may publish revised and/or new versions of
the GNU General Public License from time to time.  Such new versions will
be similar in spirit to the present version, but may differ in detail to
address new problems or concerns.

  Each version is given a distinguishing version number.  If the
Program specifies that a certain numbered version of the GNU General
Public License "or any later version" applies to it, you have the
option of following the terms and conditions either of that numbered
version or of any later version published by the Free Software
Foundation.  If the Program does not specify a version number of the
GNU General Public License, you may choose any version ever published
by the Free Software Foundation.

  If the Program specifies that a proxy can decide which future
versions of the GNU General Public License can be used, that proxy's
public statement of acceptance of a version permanently authorizes you
to choose that version for the Program.

  Later license versions may give you additional or different
permissions.  However, no additional obligations are imposed on any
author or copyright holder as a result of your choosing to follow a
later version.

  15. Disclaimer of Warranty.

  THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY
APPLICABLE LAW.  EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT
HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY
OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO,
THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE.  THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM
IS WITH YOU.  SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF
ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

  16. Limitation of Liability.

  IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
```

(continues on next page)

(continued from previous page)

```
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS
THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY
GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE
USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF
DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD
PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS),
EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF
SUCH DAMAGES.

  17. Interpretation of Sections 15 and 16.

  If the disclaimer of warranty and limitation of liability provided
above cannot be given local legal effect according to their terms,
reviewing courts shall apply local law that most closely approximates
an absolute waiver of all civil liability in connection with the
Program, unless a warranty or assumption of liability accompanies a
copy of the Program in return for a fee.

                     END OF TERMS AND CONDITIONS

            How to Apply These Terms to Your New Programs

  If you develop a new program, and you want it to be of the greatest
possible use to the public, the best way to achieve this is to make it
free software which everyone can redistribute and change under these terms.

  To do so, attach the following notices to the program.  It is safest
to attach them to the start of each source file to most effectively
state the exclusion of warranty; and each file should have at least
the "copyright" line and a pointer to where the full notice is found.

    <one line to give the program's name and a brief idea of what it does.>
    Copyright (C) <year>  <name of author>

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

  If the program does terminal interaction, make it output a short
notice like this when it starts in an interactive mode:

    <program>  Copyright (C) <year>  <name of author>
    This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.
```

(continues on next page)

(continued from previous page)

```
The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License.  Of course, your program's commands
might be different; for a GUI interface, you would use an "about box".

  You should also get your employer (if you work as a programmer) or school,
if any, to sign a "copyright disclaimer" for the program, if necessary.
For more information on this, and how to apply and follow the GNU GPL, see
<http://www.gnu.org/licenses/>.

  The GNU General Public License does not permit incorporating your program
into proprietary programs.  If your program is a subroutine library, you
may consider it more useful to permit linking proprietary applications with
the library.  If this is what you want to do, use the GNU Lesser General
Public License instead of this License.  But first, please read
<http://www.gnu.org/philosophy/why-not-lgpl.html>.
```

Bibliography

[Jiang2016] Y. Jiang et al., An expanded evaluation of protein function prediction methods shows an improvement in accuracy, Genome Biology, vol. 17, p. 184, 2016

[Burdakov06] O. Sysoev, A. Grimvall, and O. Burdakov, Data preordering in generalized pav algorithm for monotonic regression, Journal of Computational Mathematics, vol. 24, no. 6, pp. 771–790, 2006

[Obozinski08] Obozinski G, Lanckriet G, Grant C, M J, Noble WS. Consistent probabilistic output for protein function prediction. Genome Biology. 2008;9:135–142. doi:10.1186/gb-2008-9-s1-s6.