anRichment tutorial

Peter Langfelder

June 3, 2018

Abstract

This document illustrates the use and functionality of the R package `anRichment`.

Contents

1 Introduction

What is anRichment?

The packages `anRichment` and `anRichmentMethods` are add-on package for the R statististical language and environment (www.r-project.org) that aim to do the following:

- Make it simple to calculate enrichment of user-supplied classes (for example, modules) of genes in a collection of previously known reference gene sets. A future aim, at present not yet supported, is to also implement concordance calculations for continuous properties (rather than the binary present/absent membership in reference gene lists).
- Provide easy access to multiple gene sets: standard reference gene sets such as GO, KEGG, Reactome etc., and gene sets that were collected by various people including Jeremy A. Miller, Jim Wang, Mike Palazzolo, William Yang and members of his lab, as well as Rancho Biosciences and that were found useful in multiple research projects.
- Make it simple to add user-defined reference gene sets to the collection.
- Provide functionality for gene set annotation as well as tagging with keywords, and selecting sub-collections based on keywords
- Provide utility functions that simplify working with Bioconductor organism annotation packages, mapping of Entrez identifiers between organisms, creating of functional annotation labels for user gene classes (modules) and other tasks commonly encountered when performing functional enrichment analysis.

Package `anRichment` contains data and accessor functions to load collections of gene sets while `anRichmentMethods` implements methods. One only needs to load `anRichment` ; `anRichmentMethods` will be loaded automatically. We anticipate that `anRichmentMethods` will be updated more often; splitting off the data will make the more frequent download and installation of methods faster.

History

Jeremy Miller collected multiple brain- and blood-related gene sets from the literature and wrote the function `userListEnrichment` [1] that could calculate enrichment statistics for user-supplied classes in both his gene sets as well as user-supplied reference gene sets. Meanwhile, Peter Langfelder became fed-up with having to manually submit modules for enrichment analysis in DAVID and wrote the function `GOenrichmentAnalysis` in the `WGCNA` package to calculate enrichment of user gene classes in GO terms. Soon it became obvious that one would always be using both functions and it made sense to merge them. With it came the realization that one should have access not just to gene sets but also to basic meta-information such as a short description of what the gene set represents, source of the information etc, and that it would be great to be able to group reference gene sets into groups (or, more generally, tag them with keywords) to be able to efficiently sort and select sub-collections of the reference collection.

Caution: unstable ground!

Packages `anRichmentMethods` and `anRichment` should still be considered experimental and everything, including the package names, may change in the future.

2 Installation of anRichment

To run an analysis, one needs to install R, and, within R, install packages `anRichment`, `anRichmentMethods`, and their dependencies. We now provide an experimental script to do this automatically. To run it, start R and type

```
source(paste0("https://labs.genetics.ucla.edu/horvath/htdocs/CoexpressionNetwork",
              "GeneAnnotation/installAnRichment.R"));
installAnRichment();
```

This script will automatically install all dependecies as well as the `anRichmentMethods` and `anRichment` packages. The script tries to avoid installing packages that are already installed.
Should the automatic installation script fail, the user can try manual, step-by-step installation. The dependencies can be installed from within R by typing

```
source("http://bioconductor.org/biocLite.R");
biocLite(c("AnnotationDBI", "GO.db", "org.Hs.eg.db", "org.Mm.eg.db", "XML", "WGCNA",
           "TxDb.Hsapiens.UCSC.hg19.knownGene", "TxDb.Mmusculus.UCSC.mm10.knownGene"));
```

This installs the minimal requirements plus organism annotation packages for human and mouse. Users wishing to analyze gene sets corresponding to other organisms should also install the appropriate Bioconductor packages. For example, for rat (Rattus Norvegicus), one would execute (in addition to the above)

```
source("http://bioconductor.org/biocLite.R");
biocLite(c("org.Rn.eg.db"));
```

Finally, download the `anRichmentMethods` and `anRichment` packages from this web site[1], save them on your disk (noting the folder where they were saved), then type in R

```
install.packages("path/to/anRichmentMethods", repos = NULL, type = "source")
install.packages("path/to/anRichment", repos = NULL, type = "source")
```

where you replace the `path/to/anRichmentMethods` and `path/to/anRichment` above with the actual paths (full folder names) and the file names of the saved files.

3 Example analysis

This section contains annotated R code illustrating various aspects of the package functionality. We will calculate the enrichment of modules determined by Oldham et al. [2] in the human cortex in the gene lists compiled by Jeremy Miller, as well as in GO terms.
This analysis uses example data available from our web site[2]. To reproduce our example analysis, the user should download the data, save them into a folder created specifically for this analysis, and start a new R session in that folder.

3.1 Setting up the R session and loading data

We start by changing the working directory to the folder with the example data and we load the necessary packages.

```
options(stringsAsFactors = FALSE);
library("anRichment");
```

[1] http://labs.genetics.ucla.edu/horvath/htdocs/CoexpressionNetwork/GeneAnnotation/
[2] http://labs.genetics.ucla.edu/horvath/htdocs/CoexpressionNetwork/GeneAnnotation/

Change the path in the following call to the folder created for this analysis.
Remember to change `path/to/analysis` with actual path to the analysis folder. We next load the module assignments generated by Oldham et al. and convert gene symbols to Entrez identifiers. The conversion is carried out by the function `convert2entrez`.

```
# Read in the module assignment data for Mike Oldham's modules
data = read.csv(file = bzfile("Data/MO_FxOrg_ColorVectors.csv.bz2"), sep = ",", header = TRUE)
# We will only keep the CTX modules
symbol.0 = data$CTX_Gene;
moduleColor = data$CTX_Module;
table(moduleColor)

## moduleColor
##          black           blue          brown darkolivegreen          green
##            188           1037            868             28            502
##    greenyellow           grey       honeydew   midnightblue         orange
##             14            158             60             25            129
##      palegreen           pink     powderblue         purple            red
##             16             98             24             30            316
##         salmon            tan         tomato      turquoise         yellow
##             30             51             23           1115            837

# Some gene symbols have the form "XYZ /// ABC". Keep only the first symbol of all such multi-symbols.
split = strsplit(symbol.0, split = " /// ", fixed = TRUE);
symbol = sapply(split, function(x) x[1]);
# Convert symbols to Entrez IDs
entrez = convert2entrez(organism = "human", symbol = symbol);
# How many conversions were successful?
table(is.finite(entrez))

##
## FALSE  TRUE
##   912 17719
```

At this point we have the gene Entrez identifiers in the vector `entrez` and the corresponding module labels (colors) in `moduleColor`.

3.2 GO enrichment analysis

Enrichment analysis within `anRichment` is carried out by the function `enrichmentAnalysis`. This function has two principal inputs: the *classes* (for example, network modules) whose enrichment is to be studied, and a *collection* of reference gene sets (for example, GO terms). The reference collection has to be created before calling `enrichmentAnalysis`. Several reference collections are available internally. Here we use the GO collection, accessible by calling the function `buildGOcollection`.

```
GOcollection = buildGOcollection(organism = "human")
```

```
## 'select()' returned 1:many mapping between keys and columns
## 'select()' returned 1:1 mapping between keys and columns
```

The next call evaluates the enrichment of the gene modules in the collection of GO terms.

```
GOenrichment = enrichmentAnalysis(
     classLabels = moduleColor, identifiers = entrez,
     refCollection = GOcollection,
     useBackground = "given",
     threshold = 1e-4,
     thresholdType = "Bonferroni",
     getOverlapEntrez = TRUE,
     getOverlapSymbols = TRUE,
     ignoreLabels = "grey");

##  enrichmentAnalysis: preparing data..
##   ..working on label set 1 ..

collectGarbage();
```

The returned object is a list with several components:

```
names(GOenrichment)

##  [1] "enrichmentIsValid"        "enrichmentTable"
##  [3] "pValues"                  "Bonferroni"
##  [5] "FDR"                      "countsInDataSet"
##  [7] "effectiveBackgroundSize"  "dataSetDetails"
##  [9] "identifierIsInCollection" "effectiveClassLabels"
```

The enrichment results are summarized in the component `enrichmentTable`:

```
names(GOenrichment$enrichmentTable);

##  [1] "class"                           "rank"
##  [3] "dataSetID"                       "dataSetName"
##  [5] "inGroups"                        "pValue"
##  [7] "Bonferroni"                      "FDR"
##  [9] "nCommonGenes"                    "fracOfEffectiveClassSize"
## [11] "expectedFracOfEffectiveClassSize" "enrichmentRatio"
## [13] "classSize"                       "effectiveClassSize"
## [15] "fracOfEffectiveSetSize"          "effectiveSetSize"
## [17] "shortDataSetName"                "overlapGenes"
```

The last column contains concatenated lists of overlap genes and would be difficult to display; hence we shorten the last column for display purposes and display the first few rows:

```
table.display = GOenrichment$enrichmentTable;
table.display$overlapGenes = shortenStrings(table.display$overlapGenes, maxLength = 70,
                                            split = "|");
head(table.display);

##   class rank  dataSetID                        dataSetName inGroups       pValue
## 1 black    1 GO:0097159   organic cyclic compound binding GO|GO.MF 0.0003119227
## 2 black    2 GO:1901363     heterocyclic compound binding GO|GO.MF 0.0004553479
## 3 black    3 GO:0051531  NFAT protein import into nucleus GO|GO.BP 0.0006728918
## 4 black    4 GO:0072178        nephric duct morphogenesis GO|GO.BP 0.0006728918
## 5 black    5 GO:0048813            dendrite morphogenesis GO|GO.BP 0.0008205382
## 6 black    6 GO:1904646 cellular response to amyloid-beta GO|GO.BP 0.0018047113
##   Bonferroni       FDR nCommonGenes fracOfEffectiveClassSize
```

```
## 1          1 0.1567894           52               0.54166667
## 2          1 0.2099545           51               0.53125000
## 3          1 0.2792653            2               0.02083333
## 4          1 0.2792653            2               0.02083333
## 5          1 0.3121124            7               0.07291667
## 6          1 0.5363013            3               0.03125000
##   expectedFracOfEffectiveClassSize enrichmentRatio classSize effectiveClassSize
## 1                     0.3671917436        1.475160       103                96
## 2                     0.3623030961        1.466314       103                96
## 3                     0.0005431831       38.354167       103                96
## 4                     0.0005431831       38.354167       103                96
## 5                     0.0162954916        4.474653       103                96
## 6                     0.0027159153       11.506250       103                96
##   fracOfEffectiveSetSize effectiveSetSize                  shortDataSetName
## 1             0.03846154             1352   organic cyclic compound binding
## 2             0.03823088             1334     heterocyclic compound binding
## 3             1.00000000                2  NFAT protein import into nucleus
## 4             1.00000000                2        nephric duct morphogenesis
## 5             0.11666667               60            dendrite morphogenesis
## 6             0.30000000               10 cellular response to amyloid-beta
##                                          overlapGenes
## 1                          (More than 50 overlapping genes)
## 2                          (More than 50 overlapping genes)
## 3                               2932 (GSK3B)|5295 (PIK3R1)
## 4                                1948 (EFNB2)|2043 (EPHA4)
## 5 2043 (EPHA4)|5911 (RAP2A)|9693 (RAPGEF2)|26037 (SIPA1L1)...
## 6                     2043 (EPHA4)|2932 (GSK3B)|3480 (IGF1R)
```

In this example the overlap genes are reported as Entrez IDs and gene symbols; one can also request reporting of Entrez only (set argument `getOverlapSymbols` to `FALSE` in the call to `enrichmentAnalysis`), symbols only (set argument `getOverlapEntrez` to `FALSE`), and one can vary the maximum number of reported genes in the output component `enrichmentTable`. The full table enrichment table has hundreds of rows and we save it as a plain-text, comma-seprated value (CSV) file that can be opened in any spreadsheet software.

```
write.csv(GOenrichment$enrichmentTable, file = "Results/GOenrichment-enrichmentTable.csv",
          row.names = FALSE);
```

3.3 Getting details of highest-enriched data sets

One is often interested not just in the enrichment p-values, but also in other information about the highest-enriched data sets, such as overlapping genes. This information is provided in the component `dataSetDetails` of the output of `enrichmentAnalysis`. The component `dataSetDetails` is a list with one component per input class, ordered in the same order as they are in `enrichmentTable`. The classes are also indicated by the names of the components of `dataSetDetails`:

```
names(GOenrichment$dataSetDetails)
```

```
##  [1] "black"          "blue"           "brown"          "darkolivegreen"
##  [5] "green"          "greenyellow"    "honeydew"       "midnightblue"
##  [9] "orange"         "palegreen"      "pink"           "powderblue"
## [13] "purple"         "red"            "salmon"         "tan"
## [17] "tomato"         "turquoise"      "yellow"
```

Each component corresponding to a class is in turn a list; each component corresponds to a gene set. The order of the gene sets is again the same as in `enrichmentTable`, and `names` of the list equal the gene set IDs. For each gene set, the list contains the following components:

```
names(GOenrichment$dataSetDetails[[1]][[1]])

## [1] "dataSetID"           "dataSetName"         "dataSetDescription"
## [4] "dataSetGroups"       "enrichmentP"         "commonGeneEntrez"
## [7] "commonGenePositions"
```

The components give the gene set ID, gene set name, gene set description, groups, enrichment p-value, and, perhaps most importantly, Entrez IDs of genes that are in the overlap of the class and the gene set, and the positions of these genes in the vector of input `classLabels`.
As an example, the `head(GOenrichment$enrichmentTable)` call above indicated that the third highest-enriched gene set in the black module is GO term "cell" and that the black module and the term cell share 29 genes. To find the Entrez identifiers of the common genes, one would execute

```
GOenrichment$dataSetDetails$black[[3]]$commonGeneEntrez

## [1] 2932 5295
```

or

```
GOenrichment$dataSetDetails[[1]][[3]]$commonGeneEntrez
```

since the black module is the first one reported in `enrichmentTable`. Another way of getting the Entrez identifiers of the common genes is in the column `overlapGenes` of the table `GOenrichment$enrichmentTable`. Here the Entrez identifiers are separated by the character `|`. Whether the column `overlapGenes` is included at all and what the separator for the Entrez codes can be controlled by arguments `getOverlapGenes` and `geneSeparator` to the function `enrichmentAnalysis`.

3.4 Groups of gene sets within a collection

Gene sets within a collection can be organized into *groups*. Each gene set may belong to zero, one or several groups (in this sense, groups can be thought of as tags or keywords attached to each gene set). Each group is identified by its name, and also carries a description and a reference. For example, the GO collection carries these groups:

```
knownGroups(GOcollection)

## [1] "GO"    "GO.BP" "GO.CC" "GO.MF"
```

In addition to the all-encompassing GO group, there are groups corresponding to the GO ontologies Biological Process (GO.BP), Cellular Component (GO.CC) and Molecular Function (GO.MF). Collections can be subset using the function `subsetCollection` so that only gene sets within given groups are retained. For example, one could define a collection of GO BP gene sets as

```
GO.BPcollection = subsetCollection(GOcollection, tags = "GO.BP")
```

Help for `subsetCollection` contains more information about how to select the union or intersection of groups, as well as selecting gene sets that *do not* belong to a group.

3.5 Getting the genes in selected gene sets

It is often informative to obtain all genes within a gene set, or a selection of gene sets. The function `geneLists` can be used to collect genes in a list in which each component is a vector of gene Entrez IDs in one gene set. The function can be used to get gene lists of all gene sets in a collection, or gene lists in selected gene sets.

4 Collections available within anRichment

4.1 Internal collection

This is the oldest[3] collection within `anRichment`, compiled by Jeremy Miller and others. The collection can be accessed using the function `internalCollection`.

```
internalColl = internalCollection(organism = "human");
```

The internal collection is tagged with multiple groups at several different levels:

```
knownGroups(internalColl, sortBy = "size")

##  [1] "JAM"
##  [2] "Brain region marker enriched gene sets"
##  [3] "Brain region markers"
##  [4] "BrainRegionMarkers"
##  [5] "BrainRegionMarkers.HBA"
##  [6] "BrainRegionMarkers.HBA.localMarker(top200)"
##  [7] "ImmunePathways"
##  [8] "BrainLists"
##  [9] "BrainRegionMarkers.HBA.globalMarker(top200)"
## [10] "BrainLists.Blalock_AD"
## [11] "BrainLists.DiseaseGenes"
## [12] "BloodAtlases"
## [13] "BloodAtlases.Whitney"
## [14] "BrainLists.JAXdiseaseGene"
## [15] "BrainLists.MO"
## [16] "BrainLists.Lu_Aging"
## [17] "Cell type marker enriched gene sets"
## [18] "BrainLists.CA1vsCA3"
## [19] "BrainLists.MitochondrialType"
## [20] "BrainLists.MO.2+_26Mar08"
## [21] "BrainLists.MO.Sugino"
## [22] "BloodAtlases.Gnatenko2"
## [23] "BloodAtlases.Kabanova"
## [24] "BrainLists.Voineagu"
## [25] "Cell type markers"
## [26] "StemCellLists"
## [27] "StemCellLists.Lee"
```

Functions `dataSetNames` and `dataSetIDs` can be used to query set names within groups or convert gene set IDs to names and vice-versa.

```
dataSetNames(internalColl, groups = "PWLists.Yang")

## character(0)

dataSetNames(internalColl, groups = "BrainLists.MO")

## [1] "Alzgene_2+_26Mar08__MO"
## [2] "GABAergicNeuronsInMouseCortex_Sugino__MO"
## [3] "GlutamatergicNeuronsInMouseCortex_Sugino__MO"
## [4] "SZGene_2+_26Mar08__MO"
```

[3]The 'internal' collection was orginally the only one internally stored, hence the non-descriptive name

```
dataSetNames(internalColl, groups = "Nonexistent group")

## character(0)
```

Functions `dataSetIDs` and `dataSetNames` can be used to retrieve the data set IDs and names within a collection, possibly restricted to certain groups.

```
ids = dataSetIDs(internalColl, groups = "BrainLists.MO")
dataSetNames(internalColl, dataSets = ids)

## [1] "Alzgene_2+_26Mar08__MO"
## [2] "GABAergicNeuronsInMouseCortex_Sugino__MO"
## [3] "GlutamatergicNeuronsInMouseCortex_Sugino__MO"
## [4] "SZGene_2+_26Mar08__MO"
```

4.2 NCBI BioSystems collection

This collection contains pathways from the KEGG, REACTOME, BIOCYC and Lipid Maps components of NCBI BioSystems (https://www.ncbi.nlm.nih.gov/biosystems). The collection can be accessed as

```
biosysCollection = BioSystemsCollection("human")
```

The gene sets are tagged by the component in which they belong. NCBI BioSystems provide these gene sets for multiple organisms, so the accessor function does not perform any between-organisms conversion; should that be necessary, one can use the function `convertCollectionToOrganism`. Note that this collection is updated only occasionally.

4.3 Genomic position collection

Each gene set in this collection contains genes within a certain interval centered on a defined base-pair in the genome. The collection is built using the function `genomicPositionCollection`, for example as

```
genomicPosCollection = genomicPositionCollection(
  organism = "human",
  spacings = 5e6,
  overlapFactor = 2)

## Working on chromosome chr1
## Working on chromosome chr1_gl000191_random
## Working on chromosome chr1_gl000192_random
## Working on chromosome chr10
## Working on chromosome chr11
## Working on chromosome chr12
## Working on chromosome chr13
## Working on chromosome chr14
## Working on chromosome chr15
## Working on chromosome chr16
## Working on chromosome chr17
## Working on chromosome chr17_ctg5_hap1
## Working on chromosome chr17_gl000205_random
## Working on chromosome chr18
## Working on chromosome chr19
## Working on chromosome chr19_gl000209_random
## Working on chromosome chr2
```

```
## Working on chromosome chr20
## Working on chromosome chr21
## Working on chromosome chr22
## Working on chromosome chr3
## Working on chromosome chr4
## Working on chromosome chr4_ctg9_hap1
## Working on chromosome chr4_gl000193_random
## Working on chromosome chr4_gl000194_random
## Working on chromosome chr5
## Working on chromosome chr6
## Working on chromosome chr6_apd_hap1
## Working on chromosome chr6_cox_hap2
## Working on chromosome chr6_dbb_hap3
## Working on chromosome chr6_mann_hap4
## Working on chromosome chr6_mcf_hap5
## Working on chromosome chr6_qbl_hap6
## Working on chromosome chr6_ssto_hap7
## Working on chromosome chr7
## Working on chromosome chr7_gl000195_random
## Working on chromosome chr8
## Working on chromosome chr9
## Working on chromosome chrUn_gl000211
## Working on chromosome chrUn_gl000212
## Working on chromosome chrUn_gl000213
## Working on chromosome chrUn_gl000218
## Working on chromosome chrUn_gl000219
## Working on chromosome chrUn_gl000220
## Working on chromosome chrUn_gl000222
## Working on chromosome chrUn_gl000223
## Working on chromosome chrUn_gl000228
## Working on chromosome chrX
## Working on chromosome chrY
```

This will create a collection with interval size 5 MB, with starts of consecutive intervals shifted by $5/2 = 2.5$ MB. This collection is presently only available for human and mouse genomes; we hope to add additional organisms in the near future.

4.4 HDSigDB collection

The HD signatures database (HDSigDB) contains gene sets from a large number of published studies that relate (some directly, some more remotely) to research on Huntington's disease. The collection can be accessed as

```
HDSigCollection = HDSigDBCollection(organism = "human")
```

This collection contains some of the gene sets that were originally part of the internal collection and may, in the future, contain sets that are now included in one of the literature collections. HDSigDB is extensively documented at HDinHD (registration required) and is maintained by Rancho Biosciences for CHDI.

4.5 HD Target DB collection

The Huntington's disease target database (HD Target DB) contains gene sets related to HD that were curated by Mike Palazzolo and Jim Wang from various sources (including textbooks). The gene sets are described in Ref [3]. The collection can be retrieved using function `HDTargetDBCollection`.

4.6 Miller AIBS collection

The Miller AIBS collection contains gene sets collected by Jeremy A. Miller at AIBS. It contains variuos cell type and brain region marker sets, gene sets collected from expression studies of developing brain, as well as a collection of transcription factor (TF) targets from the original ChEA study. The collection can be retrieved using function `MillerAIBSCollection`.

4.7 Yang literature collection

The Yang literature collection contains gene sets collected by X. William Yang and members of his research group. Included are some cell type and brain region marker sets, sets from various functional studies and others. The collection can be retrieved using function `YangLiteratureCollection`.

4.8 Molecular Signatures Database

The Molecular Signatures Database (MSigDB, link) is distributed by Broad Institute together with the GSEA software. Since the authors do not allow re-distribution of the database itself, package `anRichment` provides a function to convert MSigDB from the format provided by Broad to an `anRichment` collection. To use the function (and MSigDB), download MSigDB in the XML form from the MSigDB link above (you must login/register). The function `MSigDBCollection` can then be used to build the corresponding `anRichment` collection

```
msdbColl = MSigDBCollection(file = "path/to/msigdb.xlm", organism = "human")
```

This function builds the collection and converts it to the target organism specified in the argument `organism`; one can also build the collection without any organism conversions using function `buildMSigDBCollection`.

5 Example analysis, continued

5.1 Combining collections

Collections can be combined together using the function `mergeCollections`,

```
combinedCollection = mergeCollections(
   GOcollection,
   internalColl,
   biosysCollection,
   HDSigCollection);
knownGroups(combinedCollection, sortBy = "size")
```

```
##  [1] "GO"
##  [2] "GO.BP"
##  [3] "GO.MF"
##  [4] "GO.CC"
##  [5] "REACTOME"
##  [6] "HDSigDB"
##  [7] "KEGG"
##  [8] "High-throughput association analysis"
##  [9] "BIOCYC"
## [10] "Differential expression induced by Htt CAG length expansion"
## [11] "JAM"
## [12] "Brain region marker enriched gene sets"
## [13] "JA Miller"
## [14] "Pathway Interaction Database"
## [15] "Brain region markers"
```

```
## [16] "Striatum"
## [17] "BrainRegionMarkers.HBA"
## [18] "mRNA WGCNA"
## [19] "Brain"
## [20] "Cortex"
## [21] "BrainRegionMarkers"
## [22] "BrainRegionMarkers.HBA.localMarker(top200)"
## [23] "Cerebellum"
## [24] "BrainRegionMarkers.HBA.localMarker(FC>2)"
## [25] "Cell type marker enriched gene sets"
## [26] "Hippocampus"
## [27] "Liver"
## [28] "ImmunePathways"
## [29] "Cell type markers"
## [30] "Neuron"
## [31] "BrainLists"
## [32] "BrainRegionMarkers.HBA.globalMarker(top200)"
## [33] "BrainLists.CTX"
## [34] "Protein WGCNA"
## [35] "BrainLists.MO"
## [36] "BrainLists.HumanMeta"
## [37] "BrainLists.Sugino/Winden"
## [38] "Microglia"
## [39] "BloodAtlases"
## [40] "BrainLists.Blalock_AD"
## [41] "BrainLists.CA1vsCA3"
## [42] "BrainLists.MO.Foster"
## [43] "BrainLists.MouseMeta"
## [44] "Microglia markers"
## [45] "BrainLists.DiseaseGenes"
## [46] "BrainLists.HumanChimp"
## [47] "StemCellLists"
## [48] "Astrocyte"
## [49] "BrainLists.ADvsCT_inCA1"
## [50] "BrainLists.Cahoy"
## [51] "Oligodendrocyte"
## [52] "Protein-protein interactions"
## [53] "StemCellLists.Lee"
## [54] "BloodAtlases.Blood(composite)"
## [55] "BloodAtlases.Whitney"
## [56] "BrainLists.ABA"
## [57] "BrainLists.EarlyAD"
## [58] "BrainLists.JAXdiseaseGene"
## [59] "BrainLists.Voineagu"
## [60] "Medium spiny neuron"
## [61] "Palazzolo-Wang"
## [62] "BrainLists.Cahoy.Definite"
## [63] "BrainLists.Cahoy.Probable"
## [64] "BrainLists.Lu_Aging"
## [65] "BrainLists.MicroglialMarkers.GSE1910"
## [66] "BrainLists.MicroglialMarkers.3treatments_Thomas"
## [67] "BrainLists.MicroglialMarkers.AitGhezala"
## [68] "BrainLists.MitochondrialType"
```

```
## [69] "BrainLists.MO.2+_26Mar08"
## [70] "BrainLists.MO.Sugino"
## [71] "PWLists.PMID_17500595_Kaltenbach_2007"
## [72] "StemCellLists.Cui"
## [73] "BloodAtlases.Gnatenko"
## [74] "BloodAtlases.Gnatenko2"
## [75] "BloodAtlases.Kabanova"
## [76] "BrainLists.Bayes"
## [77] "BrainLists.MicroglialMarkers.GSE772"
## [78] "BrainLists.MO.Bachoo"
## [79] "BrainLists.MO.Morciano"
## [80] "PWLists.PMID_22556411_Culver_2012"
## [81] "PWLists.PMID_22578497_Cajigas_2012"
```

One can then calculate the enrichment using the combined collection:

```
combinedEnrichment =  enrichmentAnalysis(classLabels = moduleColor, identifiers = entrez,
                                  refCollection = combinedCollection,
                                  useBackground = "given",
                                  threshold = 1e-4,
                                  thresholdType = "Bonferroni");

##  enrichmentAnalysis: preparing data..
##   ..working on label set 1 ..
```

The first few entries in the combined enrichment table (with last column left out again) are

```
head(combinedEnrichment$enrichmentTable[, -ncol(combinedEnrichment$enrichmentTable)])

##   class rank          dataSetID
## 1 black    1 HDSigDb.human.2756
## 2 black    2 HDSigDb.human.2923
## 3 black    3 HDSigDb.human.4308
## 4 black    4 HDSigDb.human.3103
## 5 black    5 HDSigDb.human.8921
## 6 black    6 HDSigDb.human.8109
##                                                                                 dataSetName
## 1                   Coexpressed gene module M11A from cerebral cortex (Oldham via Miller/HDSigDB)
## 2                        Coexpressed gene module M7 from human brain (Miller via Miller/HDSigDB)
## 3                     HD grade 3 markers in human frontal lobe BA4 (HG-U133A) (Hodges via HDSigDB)
## 4 Alzheimer disease up-regulated genes in CA1 area of hippocampus Liang (Liang via Miller/HDSigDB)
## 5             Up-regulated genes in cerebellum of 6 mon HD Q140 mice vs Q20 (Aaronson via HDSigDB)
## 6                                  Cortex RNA M34 module (darkmagenta) (Langfelder via HDSigDB)
##
## 1                                                          HDSigDB|JA Miller|Brain|BrainLists.CTX|mRNA
## 2                                                           HDSigDB|JA Miller|Brain|BrainLists.HumanMe
## 3                                                 HDSigDB|Differential expression induced by Htt CAG len
## 4                   HDSigDB|JA Miller|Brain|BrainLists.ADvsCT_inCA1|High-throughput association analysi
## 5 HDSigDB|High-throughput association analysis|Differential expression induced by Htt CAG length expansi
## 6                                                                                 HDSigDB|mRNA
##          pValue    Bonferroni           FDR nCommonGenes
## 1 8.055479e-182 4.258126e-176 4.731252e-177           93
## 2  1.725193e-28  9.119370e-23  1.139921e-24           19
## 3  7.936704e-21  4.195342e-15  3.178289e-17           40
```

```
## 4  1.585711e-18  8.382071e-13  5.514520e-15           42
## 5  1.230674e-11  6.505341e-06  2.733336e-08           37
## 6  1.777557e-09  9.396164e-04  3.132055e-06           26
##   fracOfEffectiveClassSize expectedFracOfEffectiveClassSize enrichmentRatio
## 1                0.9789474                       0.025753425       38.012318
## 2                0.2000000                       0.006027397       33.181818
## 3                0.4210526                       0.078082192        5.392428
## 4                0.4421053                       0.100273973        4.408973
## 5                0.3894737                       0.121369863        3.208982
## 6                0.2736842                       0.073972603        3.699805
##   classSize effectiveClassSize fracOfEffectiveSetSize effectiveSetSize
## 1       102                 95             0.98936170               94
## 2       102                 95             0.86363636               22
## 3       102                 95             0.14035088              285
## 4       102                 95             0.11475410              366
## 5       102                 95             0.08352144              443
## 6       102                 95             0.09629630              270
##                                                    shortDataSetName
## 1                    Coexpressed gene module M11A from cerebral cortex
## 2                         Coexpressed gene module M7 from human brain
## 3              HD grade 3 markers in human frontal lobe BA4 (HG-U133A)
## 4 Alzheimer disease up-regulated genes in CA1 area of hippocampus Liang
## 5        Up-regulated genes in cerebellum of 6 mon HD Q140 mice vs Q20
## 6                                  Cortex RNA M34 module (darkmagenta)
```

Combining collections affects not only the multiple testing corrections, it can also affect the uncorrected p-values for certain choices of the background (e.g., when the background set of genes is taken to be those genes that appear in both the reference collection and the `identifiers` supplied by the user).

5.2 Creating enrichment labels for classes

An unsupervised network analysis such as WGCNA typically labels modules by arbitrary labels such as numbers 1, 2, ... or colors "turquoise", "blue", ... It is often desirable to create biologically informative labels based on what genes the modules contain; such labels can be created from highest-enriched gene sets. `anRichmentMethods` provides the function `enrichmentLabels` for this purpose. The function takes as input the enrichment table returned by `enrichmentAnalysis` in the `enrichmentTable` component, and lets the user specify the various columns needed to put together the enrichment label. Here we create the enrichment labels for the cortical modules.

```
eLabels = enrichmentLabels(
   combinedEnrichment$enrichmentTable,
   focusOnGroups = c("all", "GO", "Cell type markers", "Brain region markers", "HDSigDB"),
   groupShortNames = c("all", "GO", "CT", "BR", "HD"),
   minSize = 0.05,
   numericClassLabels = FALSE);
```

The function returns a data frame with information about highest enriched terms in each of the groups specified in `focusOnGroups` ("all" is a special keyword meaning all terms irrespective of what group they belong to). The component `enrichmentLabel` contain one-line, text-formatted summaries that can be used as labels in various steps of the presentation of network analysis results. In this case the highest-enriched term is invariably the module itself, since these modules are also contained as gene sets in the internal collection.

5.3 Restricting enrichment calculations to given groups

The enrichment calculation can be restricted to groups given by the user. As an example, we evaluate the enrichment of the cortex modules in blood atlas gene sets. Because the blood sets comprise a relatively small number of genes,

we also instruct the function to use the `"given"` gens, that is all genes given in `identifiers`, as background.

```
bloodAtlasEnrichment =  enrichmentAnalysis(classLabels = moduleColor, identifiers = entrez,
                                refCollection = combinedCollection,
                                useGroups = c("BloodAtlases", "ImmunePathways"),
                                useBackground = "given",
                                threshold = 5e-2,
                                nBestDataSets = 3,
                                thresholdType = "Bonferroni");
```

```
##  enrichmentAnalysis: preparing data..
##   ..working on label set 1 ..
```

```
head(bloodAtlasEnrichment$enrichmentTable[, -16])
```

```
##   class rank  dataSetID                                                dataSetName
## 1 black    1 JAM:002948                                       MTor__ImmunePathway
## 2 black    2 JAM:003034 Reticulocytes_genesCorrelatedAcrossIndividuals__Whitney
## 3 black    3 JAM:002902                                     IL-5 __ImmunePathway
## 4  blue    1 JAM:002932                                      MAPK__ImmunePathway
## 5  blue    2 JAM:003077                               TCR signaling__ImmunePathway
## 6  blue    3 JAM:002895                                    IL-1R1__ImmunePathway
##                                inGroups      pValue Bonferroni       FDR
## 1                    JAM|ImmunePathways 0.049782508          1 0.7699695
## 2 JAM|BloodAtlases|BloodAtlases.Whitney 0.070766864          1 0.9073142
## 3                    JAM|ImmunePathways 0.086073414          1 0.9984516
## 4                    JAM|ImmunePathways 0.001816587          1 0.0602069
## 5                    JAM|ImmunePathways 0.003543222          1 0.1002472
## 6                    JAM|ImmunePathways 0.012138756          1 0.2995948
##   nCommonGenes fracOfEffectiveClassSize expectedFracOfEffectiveClassSize
## 1            2               0.02105263                       0.003835616
## 2            2               0.02105263                       0.004657534
## 3            2               0.02105263                       0.005205479
## 4           19               0.03247863                       0.016438356
## 5           11               0.01880342                       0.007945205
## 6            8               0.01367521                       0.005753425
##   enrichmentRatio classSize effectiveClassSize fracOfEffectiveSetSize
## 1        5.488722       102                 95              0.1428571
## 2        4.520124       102                 95              0.1176471
## 3        4.044321       102                 95              0.1052632
## 4        1.975783       655                585              0.3166667
## 5        2.366637       655                585              0.3793103
## 6        2.376882       655                585              0.3809524
##                                shortDataSetName
## 1                                          MTor
## 2 Reticulocytes_genesCorrelatedAcrossIndividuals
## 3                                          IL-5
## 4                                          MAPK
## 5                                TCR signaling
## 6                                       IL-1R1
##                                                                                  overlapGenes
## 1                                                                                   2932|5295
## 2                                                                                   150|10098
## 3                                                                                   2932|5295
## 4 1398|1399|2260|5604|6885|5594|1432|5599|5601|8550|4763|51701|5058|5322|5534|5894|5921|6197|7043
```

```
## 5                                          1398|1399|3708|5604|6885|5594|1432|5599|5601|5058|5894
## 6                                                   5604|6885|5594|1432|5599|5601|7334|7335
```

Instead of specifying the groups as input to `enrichmentAnalysis`, one can first subset the reference collection (i.e., restrict it to selected groups or by other criteria), then use the new reference collection in a call to `enrichmentAnalysis`. See Section 7.1 for more details.

5.4 Specifying active vs. inactive gene lists rather than class labels

Instead of giving labels and gene Entrez identifiers, one can also directly specify a list of "active" genes, and the corresponding background. This is equivalent to supplying, as identifiers, the union of the background and active lists, and labels that are 1 for active genes, and 0 for background.
It is not necessary to remove the "active" identifiers from the list of inactive ones; active identifiers are automatically removed from the inactive vector by the `enrichmentAnalysis` function. As an example, we re-calculate the enrichment of the "blue" module in GO terms.

```
active = entrez[moduleColor=="blue"];
all = entrez;
GOenrichment.blue = enrichmentAnalysis(active = active, inactive = all,
                                       refCollection = GOcollection,
                                       useBackground = "intersection",
                                       threshold = 1e-4,
                                       thresholdType = "Bonferroni");

##  enrichmentAnalysis: preparing data..
##   ..working on label set 1 ..

head(GOenrichment.blue$enrichmentTable[, -16])

##      class rank  dataSetID                            dataSetName inGroups
## 1 active.1    1 GO:0032991                 macromolecular complex GO|GO.CC
## 2 active.1    2 GO:0005737                               cytoplasm GO|GO.CC
## 3 active.1    3 GO:0044424                      intracellular part GO|GO.CC
## 4 active.1    4 GO:0005622                           intracellular GO|GO.CC
## 5 active.1    5 GO:0005829                                 cytosol GO|GO.CC
## 6 active.1    6 GO:0051649 establishment of localization in cell GO|GO.BP
##         pValue   Bonferroni          FDR nCommonGenes fracOfEffectiveClassSize
## 1 6.360479e-20 1.418514e-15 1.418514e-15          403                0.4642857
## 2 1.267772e-18 2.827384e-14 8.007750e-15          706                0.8133641
## 3 1.431176e-18 3.191809e-14 8.007750e-15          802                0.9239631
## 4 1.436239e-18 3.203100e-14 8.007750e-15          812                0.9354839
## 5 5.377528e-18 1.199296e-13 2.398592e-14          399                0.4596774
## 6 2.368443e-16 5.282101e-12 8.803501e-13          216                0.2488479
##   expectedFracOfEffectiveClassSize enrichmentRatio classSize effectiveClassSize
## 1                        0.3218703        1.442462       871                868
## 2                        0.6862005        1.185315       871                868
## 3                        0.8244583        1.120691       871                868
## 4                        0.8405123        1.112992       871                868
## 5                        0.3249408        1.414650       871                868
## 6                        0.1477323        1.684452       871                868
##   fracOfEffectiveSetSize                    shortDataSetName
## 1             0.10983919                 macromolecular complex
## 2             0.09025825                               cytoplasm
## 3             0.08533731                    intracellular part
```

```
## 4             0.08475107                          intracellular
## 5             0.10772138                                cytosol
## 6             0.12826603 establishment of localization in cell
##                       overlapGenes
## 1 (More than 50 overlapping genes)
## 2 (More than 50 overlapping genes)
## 3 (More than 50 overlapping genes)
## 4 (More than 50 overlapping genes)
## 5 (More than 50 overlapping genes)
## 6 (More than 50 overlapping genes)
```

5.5 Legacy enrichment calculations: function `userListEnrichment`

For users who wish to run old code that relied on the `userListEnrichment` function from the `WGCNA` R package, package `anRichment` provides a replacement function also called `userListEnrichment`. The replacement function takes all of the arguments that the original took, plus additional arguments allowing the user to specify organism and use gene Entrez identifiers instead of gene symbols. The replacement function produces output that is very similar to the original, but users should be aware that the reported overlaps and p-values may be slightly different because the conversion between the gene symbols used in the old function and the Entrez identifiers used in the new one does not cover all genes.

5.6 Choice of background for enrichment calculations

The background set of genes (the "universe") for enrichment calculations can be specified using the argument `useBackground`. The choices are:

- Intersection of the genes given in `identifiers` and in the reference collection (the default);
- Genes given in `identifiers`;
- All genes in the reference collection;
- All organism genes present in the appropriate organism database package from Bioconductor.

For large collections, for example the GO collection, that cover most of the organism's genes, it is prudent to restrict the background to all genes in the collection. Similarly, if `identifiers` cover most of the organism's genes in a reasonably unbiased manner (e.g., all genes on a microarray or in a whole-genome RNA-seq experiment, it makes sense to restrict the background to only genes present in `identifiers`. When both conditions are met, the "intersection" background is appropriate as it is less likely to lead to inflated p-values.

6 Additional collections not included in anRichment

Some users may be interested in additional collections beyond the stanadard internal collections and the dynamically generated collections such as GO, MSigDB genomic position. We aim to provide further collections in separate 'mostly-data" packages that contain an additional collection (or collections) and simple accessor functions.

6.1 The `HuntingtonDiseaseWGCNACollection` collection

This collection contains, as gene sets, the modules determined by Weighted Gene Co-expression Network Analysis (WGCNA) applied to various Huntington's disease (HD)-related data sets. Some analyses are plain WGCNA, some are consensus WGCNA across multiple data sets. Some of the data sets survey expression in human, and some in mouse data, so the collection contains gene sets corresponding to both organisms.
To retrieve the WGCNA HD collection with gene Entrez IDs converted to human, use

```
library(HuntingtonDiseaseWGCNACollection);
WGCNA.HD.coll = HuntingtonDiseaseWGCNACollection("human")
```

Specifying `NULL` instead of `"human"` will return the collection with gene sets left in their orginial organism; this collection may be useful for further subsetting but cannot be used for enrichment calculations before all gene sets are converted to a single organism.
Groups in the internal collection group together all human and all mouse analyses, and there is a group for each analysis that groups togther the modules in that analysis:

```
knownGroups(WGCNA.HD.coll, sortBy = "size")
```

```
##  [1] "WGCNA of HD data by Peter Langfelder"
##  [2] "WGCNA of mouse HD data by Peter Langfelder"
##  [3] "WGCNA of human HD data by Peter Langfelder"
##  [4] "Consensus WGCNA of Str, Ctx, Hip, Crb 6-, 10-month Allelic Series"
##  [5] "Consensus WGCNA of 2-, 6-, 10-month Allelic Series striatum"
##  [6] "WGCNA of Harvard Brain Tissue Resource Center - CB"
##  [7] "Consensus WGCNA of 2-, 6-, 10-month Allelic Series cerebellum"
##  [8] "Consensus WGCNA of 2-, 6-, 10-month Allelic Series cortex"
##  [9] "Consensus WGCNA of 2-, 6-, 10-month Allelic Series liver"
## [10] "WGCNA of Harvard Brain Tissue Resource Center - PrfC"
## [11] "Consensus WGCNA across human CN, CB, CTX"
## [12] "WGCNA of Harvard Brain Tissue Resource Center - VisC"
## [13] "Consensus WGCNA of 2-, 6-, 10-month Allelic Series hippocampus"
## [14] "Consensus WGCNA across human CN, CTX"
## [15] "Consensus WGCNA across mouse GEO striatum"
## [16] "WGCNA of Giles 2012, Q150 striatum"
## [17] "Consensus WGCNA across mouse GEO and Allelic series striatum"
## [18] "Consensus WGCNA across mouse R6/2, Q150, Allelic Series striatum"
## [19] "Consensus WGCNA of Hodges data on grade 0-2 samples only"
## [20] "Consensus WGCNA of human CN data"
## [21] "WGCNA of Becanovic (2010), YAC128 striatum (Affy data)"
## [22] "WGCNA of Giles2012, Q150 striatum, adjusted for age"
## [23] "WGCNA of HD iPSC cell cultures from HD iPSC Consortium"
## [24] "WGCNA of BACHD-dN17 Cortex"
## [25] "WGCNA of BACHD-dN17 Striatum"
## [26] "WGCNA of Kuhn (2007), R6/2 striatum"
```

6.2 Single cell RNA-Seq cell type collection and brain disease collection

These two collections were collected by Verge Genomics. They are accessed as

```
library(SCSBrainCellTypeCollection)
scbtCollection = SCSBrainCellTypeCollection("human");
library(BrainDiseaseCollection)
bdCollection = BrainDiseaseCollection("human");
```

7 User-defined gene sets and collections

This section describes methods for modifying (e.g., subsetting) existing collections, creating user-defined gene sets and collections, as well as exporting existing collections into plain text tables.

7.1 Subsetting collections

Collections can be subset (i.e., using the function `subsetCollection`. The criteria for inclusion in the subset collection can be specified as tags (these can match data set name or groups the data set belongs to) or dates. Search by tag can be either using an exact match or using search via regular expressions. Data sets can also be restricted by earliest or latest day. Finally, search can be inverted, that is, matching data sets will be *excluded* from the returned collection. As an example, we take the internal collection `internalColl` and create various subsets. First, we restrict the collection to brain lists (group "BrainLists"):

```
brainListColl = subsetCollection(internalColl, tags = "BrainLists");
nDataSets(brainListColl)

## [1] 34
```

There are 34 gene sets in this collection. To remove "BrainLists" from the internal collection, we would execute

```
noBrainListColl = subsetCollection(internalColl, tags = "BrainLists", invertSearch = TRUE);
nDataSets(noBrainListColl)

## [1] 166
```

We now have 166 gene sets left.

7.2 Adding user-defined gene sets and collections programmatically

An important capability of `anRichment` is the ability for the user to create custom gene sets, groups, and collections. We illustrate this procedure on a simple example of genes that are either in the blue or black module. Our approach here is a bit naive since in a real analysis one should be more careful about the probe to gene mapping. We start by selecting the blue and black genes and dropping missing Entrez identifiers,

```
moduleColorX = moduleColor;
moduleColorX[is.na(moduleColor)] = "grey";
bbGeneEntrez.0 = entrez[ moduleColorX %in% c("blue", "black") ];
# Some of the entrez codes are missing; we will drop them
bbGeneEntrez.1 = bbGeneEntrez.0[ is.finite(bbGeneEntrez.0) ];
# Multiple entrez codes are represented by several probes: keep only one copy of each.
bbGeneEntrez = unique(bbGeneEntrez.1);
```

A gene set contains the Entrez identifiers of the genes that belong to it, and, for every gene, an evidence code for the evidence that the gene belongs to the gene set, as well as source of that evidence (an article reference, web site, etc). Available evidence codes can be displayed by typing

```
knownEvidenceCodes()[, c(1:3)]

##    evidenceCode                             evidenceDescription
## 1           EXP                          Inferred from Experiment
## 2           IDA                        Inferred from Direct Assay
## 3           IPI                Inferred from Physical Interaction
## 4           IMP                    Inferred from Mutant Phenotype
## 5           IGI                 Inferred from Genetic Interaction
## 6           IEP                  Inferred from Expression Pattern
## 7           ISS Inferred from Sequence or Structural Similarity
## 8           ISO                  Inferred from Sequence Orthology
## 9           ISA                  Inferred from Sequence Alignment
## 10          ISM                      Inferred from Sequence Model
```

```
## 11          IGC                    Inferred from Genomic Context
## 12          IBA     Inferred from Biological aspect of Ancestor
## 13          IBD   Inferred from Biological aspect of Descendant
## 14          IKR                      Inferred from Key Residues
## 15          IMR                  Inferred from Missing Residues
## 16          IRD                  Inferred from Rapid Divergence
## 17          RCA   inferred from Reviewed Computational Analysis
## 18          TAS                      Traceable Author Statement
## 19          NAS                  Non-traceable Author Statement
## 20           IC                            Inferred by Curator
## 21           ND                   No biological Data available
## 22          IEA             Inferred from Electronic Annotation
## 23           NR                                  Not Recorded
## 24        other                                         other
##              evidenceType
## 1            Experimental
## 2            Experimental
## 3            Experimental
## 4            Experimental
## 5            Experimental
## 6            Experimental
## 7           Computational
## 8           Computational
## 9           Computational
## 10          Computational
## 11          Computational
## 12          Computational
## 13          Computational
## 14          Computational
## 15          Computational
## 16          Computational
## 17          Computational
## 18       Author statement
## 19       Author statement
## 20      Curator statement
## 21      Curator statement
## 22 Automatically assigned
## 23               Obsolete
## 24                  other
```

Since this gene set was determined by analysis of expression data, so we will assign code "Inferred from Expression Pattern" (IEP). We now generate the gene set.

```
bbGeneSet = newGeneSet(
    geneEntrez = bbGeneEntrez,
    geneEvidence = "IEP",
    geneSource = paste0("Oldham MC et al, ",
                        "Functional organization of the transcriptome in human brain",
                        "Nature Neuroscience 11, 1271 - 1282 (2008)"),
    ID = "dummy000001",
    name = "bb_M00_CTX",
    description = "Blue or black genes from CTX network",
    source = paste0("Oldham MC et al, ",
                    "Functional organization of the transcriptome in human brain",
```

```
                    "Nature Neuroscience 11, 1271 - 1282 (2008)"),
    organism = "human",
    internalClassification = c("PL", "dummy"),
    groups = "PL",
    lastModified = "2011-11-01");
```

In addition to the gene information, a gene set also contains several pieces of meta-information: a unique identifier, a name (should also be unique but need not be), a short description, source (article reference etc), organism for which the gene set is defined, internal classification (a vector of keywords that are in principle arbitrary but should hopefully be organized in a hierarchical structure, from most general to most specific), names of groups the gene set belongs to, and date of last modification. The meta-information helps the user identify the meaning and source of gene sets and it is very important that as much information as possible be included.
We next create a group `"PL"` that is referenced in the gene set we just created.

```
PLgroup =  newGroup(name = "PL", description = "PL's experimental group of gene sets",
                                 source = "Personal imagination");
```

Finally, we create a collection that will hold the gene set and the group

```
PLcollection = newCollection(dataSets = list(bbGeneSet), groups = list(PLgroup));
```

Note that `newCollection` taks as arguments lists of gene sets and lists of groups.
Alternatively, one can also create an empty collection and add data sets and groups later

```
PLcollection = newCollection()
PLcollection = addToCollection(PLcollection, bbGeneSet, PLgroup)
```

The collection is now ready for enrichment calculations.

```
PLenrichment = enrichmentAnalysis(classLabels = moduleColor, identifiers = entrez,
                                  refCollection = PLcollection,
                                  useBackground = "given",
                                  threshold = 5e-2,
                                  nBestDataSets = 3,
                                  thresholdType = "Bonferroni");

##  enrichmentAnalysis: preparing data..
##   ..working on label set 1 ..

head(PLenrichment$enrichmentTable[, -16])

##            class rank   dataSetID dataSetName inGroups       pValue
## 1          black    1 dummy000001  bb_M00_CTX       PL 1.631779e-72
## 2           blue    1 dummy000001  bb_M00_CTX       PL 0.000000e+00
## 3          brown    1 dummy000001  bb_M00_CTX       PL 1.000000e+00
## 4 darkolivegreen    1 dummy000001  bb_M00_CTX       PL 1.000000e+00
## 5          green    1 dummy000001  bb_M00_CTX       PL 1.000000e+00
## 6    greenyellow    1 dummy000001  bb_M00_CTX       PL 1.000000e+00
##     Bonferroni          FDR nCommonGenes fracOfEffectiveClassSize
## 1 3.263557e-71 1.631779e-71           95                        1
## 2 0.000000e+00 0.000000e+00          585                        1
## 3 1.000000e+00 1.000000e+00            0                        0
## 4 1.000000e+00 1.000000e+00            0                        0
## 5 1.000000e+00 1.000000e+00            0                        0
```

```
## 6 1.000000e+00 1.000000e+00           0                        0
##   expectedFracOfEffectiveClassSize enrichmentRatio classSize effectiveClassSize
## 1                        0.1863014        5.367647       102                 95
## 2                        0.1863014        5.367647       655                585
## 3                        0.1863014        0.000000       704                600
## 4                        0.1863014        0.000000        23                 19
## 5                        0.1863014        0.000000       371                346
## 6                        0.1863014        0.000000         8                  8
##   fracOfEffectiveSetSize shortDataSetName                     overlapGenes
## 1              0.1397059       bb_M00_CTX (More than 50 overlapping genes)
## 2              0.8602941       bb_M00_CTX (More than 50 overlapping genes)
## 3              0.0000000       bb_M00_CTX
## 4              0.0000000       bb_M00_CTX
## 5              0.0000000       bb_M00_CTX
## 6              0.0000000       bb_M00_CTX
```

7.3 Importing and exporting collections to text tables

Creating many genes sets and groups programmatically can be tedious; it is often easier to prepare the requisite information in the form of text tables. Such information, stored in data frames, can be processed into a collection using the function `collectionFromDataFrames`. The inverse function, `collection2dataFrames`, turns a collection into a list consisting of data frames that store the information in plain tables.
Information stored in a collection can be turned into a set of 6 data frames:

1. A data frame containing meta-information about gene sets. In this data frame, each row corresponds to a gene set and columns give the set ID, name, description, source, organism, internal classification and groups the gene set belongs to. Since internal classification and groups can have multiple entries, entries are concatenated together using a defined separator (the default separator is — but it can be changed by the user). An additional optional column can provide a short name that is suitable for diplay where space is at a premium.

2. A data frame containing the gene content of each gene set. Each row corresponds to a gene, and the columns give the name or ID of the gene set the gene belongs to, gene Entrez, evidence code, and source. A gene can be listed multiple times since it can belong to different gene sets, or it can be included multiple times in a single gene set with different evidence codes (or even different sources).

3. A data frame containing meta-information about *gene properties*. A gene property is a numeric value recorded for each gene; the meta-information contains the same columns as that for gene sets, plus an additional column referencing an appropriate column in the gene property weight data frame described below. While gene properties can be at present defined, the package does not contain functions to relate gene properties to user-supplied gene classes.

4. A data frame containing the actual gene properties. In this data frame, each row corresponds to a gene and each column to a gene property. Each gene can be present only once.

5. A data frame containing gene weights corresponding to each property. This optional information allows one to weigh genes differently in (yet to be implemented) calculations with continuous properties. Multiple properties can share the same weight vector (e.g., equal weights), thus saving storage needed for the weight data frame.

6. A data frame containing information about groups. Each row corresponds to a group; columns give the group name, description, and source.

As an example, we create a small collection of cell type marker gene sets and export it into data frames.

```
cellTypeColl = subsetCollection(internalColl, tags = "Cell type markers");
cellTypeDF = collection2dataFrames(cellTypeColl);
```

The object `cellTypeDF` is a list of 7 data frames:

```
names(cellTypeDF)

## [1] "geneSetInfo"         "dataPropertyInfo"    "geneSetContent"
## [4] "genePropertyContent" "genePropertyWeights" "evidenceCodes"
## [7] "groupInfo"
```

The 7 data frames contain the 6 listed above, plus a data frame listing the evidence codes and their meaning. Since this data frame is generated internally, it is not needed when converting the data frames to a collection.
One could now save the individual data frames into files, turn them into databases etc. Here we show the first few entries of the `geneSetInfo` that contains meta-information about gene sets. To make the output easier to read, we use the `WGCNA` function `shortenStrings` that retains only a relatively short initial part of each entry.

```
head(shortenStrings(cellTypeDF$geneSetInfo))

##           ID                   name    shortName            description
## 1 JAM:003031 RedBloodCell__Kabanova RedBloodCell Red blood cell markers
##                   source organism       internalClassification
## 1 Kabanova S, et al. ...    human JAM|BloodAtlases|BloodAtl...
##                         groups lastModified alternateNames externalDB
## 1 JAM|BloodAtlases|BloodAtl...   2011-04-19
##   externalAccession webLink
## 1
```

We encourage the reader to replace the `geneSetInfo` component with other components in the code above to see the first few rows of each data frame.
The reverse of `collection2dataFrames` is `collectionFromDataFrames`, that is, this function creates a collection from a series of data frames. The function takes as input 6 data frames as described above, and it allows the user to specify the actual column to be used for each required type of information. This means the function is quite flexible in the sense that the input data frames do not have to have fixed column names or a fixed column order.
The defaults of the function are set such that the `collectionFromDataFrames` can process the output of codecollection2dataFrames with a minimum of arguments needed to be specified explicitly. In out example, the function can be called as

```
cellTypeColl.2 = collectionFromDataFrames(
  geneSetInfoDF = cellTypeDF$geneSetInfo,
  geneSetContentDF = cellTypeDF$geneSetContent,
  groupDF = cellTypeDF$groupInfo);
```

The collections `cellTypeColl` and `cellTypeColl.2` are now equivalent.

7.4 Exporting gene set meta-information into a data frame

The package provides a separate function `geneSetInformation` for creating a data frame with information about gene sets. The functionality overlaps somewhat with `collection2dataFrames` but `geneSetInformation` also allows the user to export the information only about selected sets, and it also adds a column containing the total number of genes in each gene set. Together with the function `geneLists` (Section 3.5) these provide means for accessing gene set information.

7.5 Parent (super-) and child (sub-)groups

It is sometimes desirable to specify parent-child or super- and sub-group relationships between groups. For example, one may have a group named Cortex for gene sets that relate to the brain area cerebral cortex (e.g., markers of cortical cell types), and a group named Brain for all brain-related gene sets. Since all cortex-related gene sets are

also brain-related, it would be convenient to specify that group Cortex is a subgroup (or "child") of group Brain in that a gene set that belongs to group Cortex would automatically be also considered part of group Brain, without having to list Brain explicitly as one of the groups. Package **anRichmentMethods** provides functionality to specify such group relationships and automatically match all genes in a subgroup to a query that uses its supergroup as one of the tags.
When creating a group using function **newGroup**, one can specify the parents (super-)groups for the group. The relationships are transitive, that is, if one defines group B to be a parent of group A (B→A) and group C to be a parent of group B (C→B), group C is automatically considered a parent of A (C→A). This code illustrates creating the groups:

```
groupA = newGroup(name = "A", description = "A", parents = "B");
groupB = newGroup(name = "B", description = "B", parents = "C");
groupC = newGroup(name = "C", description = "C");

groupLst = list(groupA, groupB, groupC);
```

Implied groups (i.e., supergroups or parents plus self) of groups can be determined using function **impliedGroups**. The function can also return all subgroups (children). For example, to get all implied (super-) groups, one can use

```
impliedGroups(groupLst, get = "parents");

## $A
## [1] "A" "B" "C"
##
## $B
## [1] "B" "C"
##
## $C
## [1] "C"
```

The output is a list with one component per group, giving the names of the groups implied by the group. In the above, group A implies groups A (self), B (direct supergroup) and C (indirect supergroup). Thus, a gene set that belongs to group A will automatically match a query for gene sets in groups B and C, without having to explicitly list groups B and C when creating the gene set.
The reverse, i.e., subgroups or children of each group can be determined using

```
impliedGroups(groupLst, get = "children");

## $A
## [1] "A"
##
## $B
## [1] "B" "A"
##
## $C
## [1] "C" "B" "A"
```

See **help(impliedGroups)** for more ways of using the function.

8 Organisms for which data are available

Many functions in this package require the user to specify the organism to which entrez identifiers belong. Organism can be specified as a character string in one of 3 ways: the common name (for example, "human"), scientific name ("Homo sapiens") or the scientific shorthand ("Hs"). **anRichment** relies on Bioconductor annotation packages for

annotation of the organismal genomes; these annotation packages exist only for a handful of selected organisms. The ones currently supported by `anRichment` can be listed by calling the function `organismLabels`:

```
organismLabels()
```

```
##    commonName           scientificName shorthand
## 1       human             Homo sapiens        Hs
## 2       mouse             Mus musculus        Mm
## 3         rat        Rattus norvegicus        Rn
## 4     malaria    Plasmodium falciparum        Pf
## 5       yeast Saccharomyces cerevisiae        Sc
## 6         fly  Drosophila melanogaster        Dm
## 7      bovine               Bos taurus        Bt
## 8        worm   Caenorhabditis elegans        Ce
## 9      canine         Canis familiaris        Cf
## 10  zebrafish              Danio rerio        Dr
## 11    chicken            Gallus gallus        Gg
## 12   mosquito        Anopheles gambiae        Ag
## 13     monkey           Macaca mulatta       Mmu
## 14      chimp          Pan troglodytes        Pt
## 15        pig               Sus scrofa        Ss
```

8.1 Converting gene sets and collections between organisms

Each gene set carries information about which organism it corresponds to. Sometimes it is desired to calculate enrichment across organisms, that is, the user's genes and the reference collection are defined for different organisms. This necessitates mapping the gene Entrez identifiers between the organisms. Mapping of Entrez identifiers can be achieved using the function `mapEntrez`. This function uses homology information (from http://www.informatics.jax.org/homology.shtml) for mappings where this information is available; otherwise, Entrez identifiers are mapped by matching their corresponding gene symbols (which is less precise but better than nothing). For converting entire gene sets or collections, one can use the functions `convertGeneSetToOrganism` and `convertCollectionToOrganism`; these use `mapEntrez` to map the Entrez identifiers, and take care of changing the organism information in the gene set structure as needed.
The user should keep in mind that converting collections between organisms should not be viewed as a substitute for defining gene sets for each organism directly from experimental data. In particular, for the GO collection, we strongly recommend calling `buildGOcollection` for every organism necessary, rather than building the GO collection once and then converting it to other organisms using `convertCollectionToOrganism`. To remind users of the fact that gene sets were converted between organisms, functions `convertGeneSetToOrganism` and `convertCollectionToOrganism` can optionally add a suffix to the gene set names and an extra sentence to the description that specify the organism that the gene set(s) were converted from.

9 Internals

The package aims to provide infrastructure for working with gene sets (in which each gene can be absent or present, possibly with multiple types of evidence) and for gene properties, which assign a number and optionally a weight for each gene.

9.1 Gene sets and gene properties

Together, gene sets and gene properties are called, for lack of a better name (or imagination on the part of the author), data sets. Roughly speaking, a data set is a list that contains all of the meta-information that gene sets and gene properties carry and which is detailed below, plus a component `data` whose format and meaning differs between gene sets and gene properties. Each data set also contains `"PL-dataSet"` within its `class` vector; gene sets

further contain `"PL-geneSet"`, while gene properties contain either `"PL-numericProperty"` or `"PL-discreteProperty"`, depending on whether the information is to be treated as a continuous number or an ordinal (discrete) variable. The components contained in a gene set can be seen, for example, as

```
names(bbGeneSet)

##  [1] "data"                  "ID"                    "name"
##  [4] "shortName"             "description"           "source"
##  [7] "organism"              "internalClassification" "groups"
## [10] "lastModified"          "alternateNames"        "externalDB"
## [13] "externalAccession"     "webLink"               "weightIndex"
## [16] "type"
```

The component `data` is a data frame and contains the gene set content, i.e., the gene Entrez identifiers, evidence codes and sources for the individual genes. The rest are meta-data or internal data. The components `internalClassification` and `groups` are (possibly empty) vectors; `weightIndex` is not used for gene sets but is present for compatibility with gene properties; all other components are non-empty scalars. For gene sets, the component `type` is always `"PL-geneSet"`. Additionally, the attribute `class` is set as described above:

```
class(bbGeneSet)

## [1] "PL-geneSet" "PL-dataSet" "list"
```

9.2 Groups

Groups within `anRichment`allow the user to group together data sets (gene sets or gene properties) that share a common theme. Each data set can belong to multiple groups or no group at all. Groups are simple lists with 5 scalar components, as the illustrated by the group `PLgroup` created above:

```
names(PLgroup)

## [1] "name"           "description"    "source"         "alternateNames"
## [5] "parents"
```

Additionally, the `class` vector contains `"PL-group"`. Membership of data sets in groups is encoded in data sets: Each data set contains the component `groups` that is a vector containing the names of groups the data set belongs to.

9.3 Collection

A collection is a list containing data sets, groups, additional information for gene properties (a vector of identifiers a data frame of weights), and a data frame of available evidence codes. The components are shown in this example:

```
names(internalColl)

## [1] "dataSets"      "groups"        "identifiers"   "weights"
## [5] "evidenceCodes"
```

The components `dataSets` and `groups` are simple lists (of data sets and groups, respectively).

References

[1] Jeremy Miller, Chaochao Cai, Peter Langfelder, Daniel Geschwind, Sunil Kurian, Daniel Salomon, and Steve Horvath. Strategies for aggregating gene expression data: The collapserows r function. *BMC Bioinformatics*, 12(1):322, 2011.

[2] Michael C. Oldham, Genevieve Konopka, Kazuya Iwamoto, Peter Langfelder, Tadafumi Kato, Steve Horvath, and Daniel H. Geschwind. Functional organization of the transcriptome in human brain. *Nature Neuroscience*, 11(11):1271–1282, October 2008.

[3] James K. T. Wang, Peter Langfelder, Steve Horvath, and Michael J. Palazzolo. Exosomes and homeostatic synaptic plasticity are linked to each other and to huntington's, parkinson's, and other neurodegenerative diseases by database-enabled analyses of comprehensively curated datasets. *Frontiers in Neuroscience*, 11:149, 2017.