anRichment tutorial

Peter Langfelder

August 10, 2024

Abstract

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

Contents

1 Introduction

What is anRichment?

`anRichment` is an add-on package for the R statististical language and environment (www.r-project.org) that aims to:

- 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 NCBI (Entrez) identifiers between organisms, creating of functional annotation labels for user gene classes (modules) and other tasks commonly encountered when performing functional enrichment analysis.

History

Jeremy Miller collected multiple brain- and blood-related gene sets from the literature and wrote the function `userListEnrichment` [2] 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!

Package `anRichment` should still be considered experimental and everything, including the package name, may change in the future.

2 Installation of anRichment

To run an analysis, one needs to install R, and, within R, install package `anRichment`and its dependencies. As `anRichment`is now hosted on GitHub, the simplest installation method is using function `github_install` from package `devtools` or alternatively function `githubinstall` from the eponymous package `githubinstall`.

```
library("devtools");
install_github("anRichment", username = "plangfelder");
```

The alternative, using `githubinstall`:

```
library("githubinstall");
githubinstall("anRichment");
```

These commands should automatically install all dependecies as well as the `anRichment` package.
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

```
install.packages("BiocManager");
BiocManager::install(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)

```
BiocManager::install(c("org.Rn.eg.db"));
```

Finally, download the `anRichment` package from GitHub[1], save it on your disk (noting the folder where they were saved), then type in R

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

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

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. [3] 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 loading the R package `anRichment` as well as other necesary packages and changing the working directory to the folder with the example data. Change the path in the `setwd` call below to the folder created for this analysis.

[1] https://github.com/plangfelder/anRichment/archive/refs/heads/release.zip
[2] http://labs.genetics.ucla.edu/horvath/htdocs/CoexpressionNetwork/GeneAnnotation/

```
setwd("path/to/analysis");
```

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

We next load the module assignments generated by Oldham et al. and convert gene symbols to NCBI (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
##            188           1037            868             28
##          green    greenyellow           grey       honeydew
##            502             14            158             60
##   midnightblue         orange      palegreen           pink
##             25            129             16             98
##     powderblue         purple            red         salmon
##             24             30            316             30
##            tan         tomato      turquoise         yellow
##             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.na(entrez))

##
## FALSE  TRUE
## 17674   957

# Build a Entrez ID-symbol translation table
entrez2symbol = data.frame(Entrez = entrez, Symbol = symbol);
```

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 functions `enrichmentAnalysis.Entrez` (recommended for enrichment based on Entrez or NCBI identifiers), `enrichmentAnalysis.general` (recommended for enrichment analysis based on general identifiers, not necessarily on genes) or `enrichmentAnalysis` (the original function retained for back-compatibility). These functions 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 either enrichment analysis function. Several reference collections are available directly in `anRichment`. Here we use the GO collection, accessible by calling the function `buildGOcollection`. This function in turns uses annotation packages from Bioconductor to access GO data.

```
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.Entrez(
     classLabels = moduleColor, identifiers = entrez,
     refCollection = GOcollection,
     useBackground = "given",
     threshold = 1e-4,
     thresholdType = "Bonferroni",
     getOverlapEntrez = TRUE,
     getOverlapSymbols = TRUE,
     ID2symbol = entrez2symbol,
     ignoreLabels = "grey");

##  enrichmentAnalysis: preparing data..
##   ..calculating overlaps..

gc();

##            used  (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells 12024739 642.2   33540765 1791.3  41925956 2239.1
## Vcells 51735839 394.8  169670187 1294.5 169670187 1294.5
```

The returned object is a list with several components:

```
names(GOenrichment)

##  [1] "enrichmentIsValid"       "enrichmentTable"
##  [3] "pValues"                 "Bonferroni"
##  [5] "FDR"                     "countsInDataSet"
##  [7] "enrichmentRatio"         "effectiveBackgroundSize"
##  [9] "effectiveDataSetSizes"   "effectiveClassSizes"
## [11] "effectiveClass2bg"       "dataSetDetails"
```

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

```
names(GOenrichment$enrichmentTable);

##  [1] "class"
##  [2] "rank"
##  [3] "dataSetID"
##  [4] "dataSetName"
##  [5] "inGroups"
##  [6] "pValue"
##  [7] "Bonferroni"
##  [8] "FDR"
```

```
##  [9] "nCommonGenes"
## [10] "fracOfEffectiveClassSize"
## [11] "expectedFracOfEffectiveClassSize"
## [12] "enrichmentRatio"
## [13] "classSize.asGiven"
## [14] "validClassSize"
## [15] "effectiveClassSize"
## [16] "fracOfEffectiveSetSize"
## [17] "effectiveSetSize"
## [18] "shortDataSetName"
## [19] "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
## 1 black    1 GO:0003777      microtubule motor activity GO|GO.MF|GO
## 2 black    2 GO:0048813          dendrite morphogenesis GO|GO.BP|GO
## 3 black    3 GO:0005634                         nucleus GO|GO.CC|GO
## 4 black    4 GO:0097159 organic cyclic compound binding GO|GO.MF|GO
## 5 black    5 GO:0072178      nephric duct morphogenesis GO|GO.BP|GO
## 6 black    6 GO:0031981                   nuclear lumen GO|GO.CC|GO
##         pValue Bonferroni       FDR nCommonGenes
## 1 0.0002510361          1 0.1384928            4
## 2 0.0002981050          1 0.1573712            8
## 3 0.0007081918          1 0.2818689           67
## 4 0.0007314294          1 0.2876457           56
## 5 0.0007923001          1 0.2997587            2
## 6 0.0018704409          1 0.5555638           48
##   fracOfEffectiveClassSize expectedFracOfEffectiveClassSize
## 1               0.03809524                      0.003231888
## 2               0.07619048                      0.016698088
## 3               0.63809524                      0.480474010
## 4               0.53333333                      0.377861567
## 5               0.01904762                      0.000538648
## 6               0.45714286                      0.319148936
##   enrichmentRatio classSize.asGiven validClassSize effectiveClassSize
## 1       11.787302               188            105                105
## 2        4.562826               188            105                105
## 3        1.328054               188            105                105
## 4        1.411452               188            105                105
## 5       35.361905               188            105                105
## 6        1.432381               188            105                105
##   fracOfEffectiveSetSize effectiveSetSize
## 1             0.33333333               12
## 2             0.12903226               62
## 3             0.03755605             1784
## 4             0.03991447             1403
## 5             1.00000000                2
```

```
## 6             0.04050633            1185
##                 shortDataSetName
## 1      microtubule motor activity
## 2          dendrite morphogenesis
## 3                         nucleus
## 4 organic cyclic compound binding
## 5      nephric duct morphogenesis
## 6                  nuclear lumen
##                                                          overlapGenes
## 1               547 (KIF1A)|1780 (DYNC1I1)|3800 (KIF5C)|10513 (APPBP2)
## 2 547 (KIF1A)|2043 (EPHA4)|2932 (GSK3B)|5911 (RAP2A)|9693 (RAPGEF2)...
## 3                                    (More than 50 overlapping genes)
## 4                                    (More than 50 overlapping genes)
## 5                                           1948 (EFNB2)|2043 (EPHA4)
## 6       687 (KLF9)|894 (CCND2)|904 (CCNT1)|1108 (CHD4)|2932 (GSK3B)...
```

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`.
For each class, the data frame contains information on the top gene sets. The individual columns are:

- `class` lists the input class;
- `rank` the rank of the gene set within each class
- `dataSetID` and `dataSetName` report the gene set ID and name
- `inGroups` lists the groups, separated by commas, the gene set belongs to (including groups implied by parent relationships);
- columns `pValue`, `Bonferroni` and `FDR` give the nominal enrichment p-value, Bonferroni correction, and FDR estimate;
- `nCommonGenes` gives the number of genes common between the class and the gene set;
- `fracOfEffectiveModuleSize` reports the number of common genes as a fraction of the effective class size (i.e., number of genes in class and in the background);
- `classSize.asGiven` lists the class sizes as they were input to the function;
- `validClassSize` gives the class size after removal of missing identifiers and potentially identifiers that map to more than one class;
- `effectiveClassSize` lists the effective class size;
- `fracOfEffectiveSetSize` gives the number of common genes as a fraction of the effective gene set size;
- `expectedFracOfEffectiveClassSize` gives the expected fraction of the effective set size assuming the null hypothesis of independence holds;
- `enrichmentRatio` gives the ration of observed and expected fractions,
- `effectiveSetSize` reports the effective gene set size (number of genes in the gene set that are also present in the background). The effective class and set sizes are the sizes that enter the enrichment calculation and are often smaller than the given sizes because of restriction to a common background;
- `overlapIDs` gives the overlap IDs or symbols or both for each comparison as a character string separated by the separator `geneSeparator` `getOverlapSymbols` is `TRUE`). At most `maxReportedOverlapGenes` is reported here; if the overlap is larger, the field will contain the character string `"(More than xxx overlapping genes)"`.

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"
##  [4] "darkolivegreen" "green"          "greenyellow"
##  [7] "honeydew"       "midnightblue"   "orange"
## [10] "palegreen"      "pink"           "powderblue"
## [13] "purple"         "red"            "salmon"
## [16] "tan"            "tomato"         "turquoise"
## [19] "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"        "commonGeneIDs"
## [7] "commonGeneSymbols"
```

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]]$commonGeneIDs
```

```
##  [1] "327"    "687"    "894"    "904"    "1108"   "1780"   "2348"
##  [8] "2932"   "3204"   "3842"   "4082"   "4750"   "4820"   "4926"
## [15] "5087"   "5295"   "5775"   "6738"   "6925"   "7013"   "7267"
## [22] "7270"   "7415"   "7528"   "7536"   "7707"   "7750"   "8123"
## [29] "8202"   "8204"   "9140"   "9194"   "9360"   "9529"   "9686"
## [36] "9910"   "9975"   "10098"  "10240"  "10513"  "10608"  "10659"
## [43] "10725"  "11137"  "23099"  "23131"  "23301"  "23389"  "23469"
## [50] "23524"  "26043"  "29123"  "29855"  "55149"  "57062"  "58517"
## [57] "58528"  "65258"  "79982"  "80034"  "80155"  "146057" "192669"
## [64] "204851" "221692" "253461" "389677"
```

or

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

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.Entrez`.

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"
```

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

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

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

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

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

```
## character(0)
```

One can also use these two functions to create a mapping between data set IDs and names.

```
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 [1] (https://www.ncbi.nlm.nih.gov/Structure/biosystems/docs/biosystems_about.html). Unfortunately, the BioSystems database has been retired and its content subsumed into the Pubchem database, so this collection is not updated anymore. Conversely, it contains what is likely the last or one of the last public snapshots of the KEGG pathway database and we hope it will remain useful into the future. 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`.

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 [4]. 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 variuous 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 XML 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`.

4.9 Phenopedia

Phenopedia [5] provides a disease-centered view of genetic association studies summarized in the online Human Genome Epidemiology (HuGE) encyclopedia. The collection can be retrieved using the function `PhenopediaCollection`:

```
phenopediaColl = PhenopediaCollection(organism = "human")
```

4.10 The `HuntingtonsDiseaseWGCNACollection` 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

```
WGCNA.HD.coll = HuntingtonsDiseaseWGCNACollection("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"
```

4.11 Single cell RNA-seq cell type collection and brain disease collection

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

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

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] "High-throughput association analysis"
##  [8] "Differential mRNA expression induced by Htt CAG length expansion"
##  [9] "KEGG"
## [10] "BIOCYC"
## [11] "Striatum"
## [12] "JAM"
## [13] "Brain region marker enriched gene sets"
## [14] "JA Miller"
## [15] "Pathway Interaction Database"
## [16] "Brain region markers"
## [17] "BrainRegionMarkers.HBA"
## [18] "mRNA WGCNA"
## [19] "Cortex"
## [20] "Cerebellum"
## [21] "Brain"
## [22] "BrainRegionMarkers"
## [23] "BrainRegionMarkers.HBA.localMarker(top200)"
## [24] "Differential protein expression induced by Htt CAG length expansion"
## [25] "Cell type markers"
## [26] "Hippocampus"
## [27] "BrainRegionMarkers.HBA.localMarker(FC>2)"
```

```
## [28] "Cell type marker enriched gene sets"
## [29] "Liver"
## [30] "Neuron"
## [31] "ImmunePathways"
## [32] "BrainLists"
## [33] "DNA methylation profiling"
## [34] "Heart"
## [35] "BrainRegionMarkers.HBA.globalMarker(top200)"
## [36] "BrainLists.CTX"
## [37] "Protein WGCNA"
## [38] "BrainLists.MO"
## [39] "Adipose"
## [40] "BrainLists.HumanMeta"
## [41] "BrainLists.Sugino/Winden"
## [42] "Microglia"
## [43] "Thalamus"
## [44] "BloodAtlases"
## [45] "BrainLists.Blalock_AD"
## [46] "BrainLists.CA1vsCA3"
## [47] "BrainLists.MO.Foster"
## [48] "BrainLists.MouseMeta"
## [49] "Cerebrum"
## [50] "Retina"
## [51] "Microglia markers"
## [52] "BrainLists.DiseaseGenes"
## [53] "BrainLists.HumanChimp"
## [54] "StemCellLists"
## [55] "Astrocyte"
## [56] "BrainLists.ADvsCT_inCA1"
## [57] "BrainLists.Cahoy"
## [58] "Oligodendrocyte"
## [59] "Protein-protein interactions"
## [60] "StemCellLists.Lee"
## [61] "BloodAtlases.Blood(composite)"
## [62] "BloodAtlases.Whitney"
## [63] "BrainLists.ABA"
## [64] "BrainLists.EarlyAD"
## [65] "BrainLists.JAXdiseaseGene"
## [66] "BrainLists.Voineagu"
## [67] "Medium spiny neuron"
## [68] "Palazzolo-Wang"
## [69] "BrainLists.Cahoy.Definite"
## [70] "BrainLists.Cahoy.Probable"
## [71] "BrainLists.Lu_Aging"
## [72] "BrainLists.MicroglialMarkers.GSE1910"
## [73] "Brainstem"
## [74] "Skin"
## [75] "BrainLists.MicroglialMarkers.3treatments_Thomas"
## [76] "BrainLists.MicroglialMarkers.AitGhezala"
## [77] "BrainLists.MitochondrialType"
## [78] "BrainLists.MO.2+_26Mar08"
## [79] "BrainLists.MO.Sugino"
## [80] "Corpus callosum"
```

```
## [81] "PWLists.PMID_17500595_Kaltenbach_2007"
## [82] "StemCellLists.Cui"
## [83] "BloodAtlases.Gnatenko"
## [84] "BloodAtlases.Gnatenko2"
## [85] "BloodAtlases.Kabanova"
## [86] "BrainLists.Bayes"
## [87] "BrainLists.MicroglialMarkers.GSE772"
## [88] "BrainLists.MO.Bachoo"
## [89] "BrainLists.MO.Morciano"
## [90] "Cerebrospinal fluid"
## [91] "PWLists.PMID_22556411_Culver_2012"
## [92] "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.7177
## 3 black    3 HDSigDb.human.2923
## 4 black    4 HDSigDb.human.4308
## 5 black    5 HDSigDb.human.3103
## 6 black    6 HDSigDb.human.8921
##                                                                                 dataSetName
## 1                    Coexpressed gene module M11A from cerebral cortex (Oldham via Miller/HDSigDB)
## 2     Down-regulated, HD grade 3 markers in human frontal lobe BA4 (HG-U133A) (Hodges via HDSigDB)
## 3                          Coexpressed gene module M7 from human brain (Miller via Miller/HDSigDB)
## 4                      HD grade 3 markers in human frontal lobe BA4 (HG-U133A) (Hodges via HDSigDB)
## 5 Alzheimer disease up-regulated genes in CA1 area of hippocampus Liang (Liang via Miller/HDSigDB)
## 6             Up-regulated genes in cerebellum of 6 mon HD Q140 mice vs Q20 (Aaronson via HDSigDB)
##
## 1
## 2                                        HDSigDB|High-throughput association analysis|Differential mRNA
## 3
## 4                                                                    HDSigDB|Differential mRNA
## 5
## 6 HDSigDB|High-throughput association analysis|Differential mRNA expression induced by Htt CAG length ex
##         pValue   Bonferroni          FDR nCommonGenes
## 1 1.468821e-81 8.015358e-76 8.015358e-77           38
## 2 1.353922e-19 7.388352e-14 9.595263e-16           17
## 3 6.092734e-18 3.324805e-12 3.675748e-14           10
## 4 2.117159e-12 1.155334e-06 7.967818e-09           17
```

```
## 5 5.255820e-10 2.868101e-04 1.509527e-06           16
## 6 5.096467e-08 2.781142e-02 1.057469e-04           16
##   fracOfEffectiveClassSize expectedFracOfEffectiveClassSize
## 1                1.0000000                       0.017441860
## 2                0.4473684                       0.022808587
## 3                0.2631579                       0.004919499
## 4                0.4473684                       0.056350626
## 5                0.4210526                       0.067531306
## 6                0.4210526                       0.091681574
##   enrichmentRatio classSize effectiveClassSize fracOfEffectiveSetSize
## 1       57.333333        39                 38             0.97435897
## 2       19.614035        39                 38             0.33333333
## 3       53.492823        39                 38             0.90909091
## 4        7.939014        39                 38             0.13492063
## 5        6.234925        39                 38             0.10596026
## 6        4.592555        39                 38             0.07804878
##   effectiveSetSize
## 1               39
## 2               51
## 3               11
## 4              126
## 5              151
## 6              205
##                                                    shortDataSetName
## 1                       Coexpressed gene module M11A from cerebral cortex
## 2 Down-regulated, HD grade 3 markers in human frontal lobe BA4 (HG-U133A)
## 3                            Coexpressed gene module M7 from human brain
## 4                 HD grade 3 markers in human frontal lobe BA4 (HG-U133A)
## 5   Alzheimer disease up-regulated genes in CA1 area of hippocampus Liang
## 6           Up-regulated genes in cerebellum of 6 mon HD Q140 mice vs Q20
```

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. `anRichment` 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
## 1 black    1 JAM:003034
## 2 black    2 JAM:002893
## 3 black    3 JAM:002948
## 4  blue    1 JAM:002932
## 5  blue    2 JAM:003077
## 6  blue    3 JAM:002895
##                                           dataSetName
## 1 Reticulocytes_genesCorrelatedAcrossIndividuals__Whitney
## 2                                 IL-17 __ImmunePathway
## 3                                   MTor__ImmunePathway
## 4                                   MAPK__ImmunePathway
## 5                           TCR signaling__ImmunePathway
## 6                                 IL-1R1__ImmunePathway
##                               inGroups      pValue Bonferroni
## 1 JAM|BloodAtlases|BloodAtlases.Whitney 0.014046118          1
## 2                   JAM|ImmunePathways 0.050144405          1
## 3                   JAM|ImmunePathways 0.113210447          1
## 4                   JAM|ImmunePathways 0.007451827          1
## 5                   JAM|ImmunePathways 0.024419811          1
## 6                   JAM|ImmunePathways 0.031969320          1
##         FDR nCommonGenes fracOfEffectiveClassSize
## 1 0.5013747            2               0.05263158
## 2 0.9381856            1               0.02631579
## 3 1.0000000            1               0.02631579
## 4 0.3201526            9               0.03409091
## 5 0.7024222            5               0.01893939
## 6 0.7682327            4               0.01515152
##   expectedFracOfEffectiveClassSize enrichmentRatio classSize
## 1                      0.004919499       10.698565        39
## 2                      0.001341682       19.614035        39
## 3                      0.003130590        8.406015        39
## 4                      0.013864043        2.458944       295
```

```
## 5                      0.006708408        2.823232       295
## 6                      0.004919499        3.079890       295
##   effectiveClassSize fracOfEffectiveSetSize
## 1                 38              0.1818182
## 2                 38              0.3333333
## 3                 38              0.1428571
## 4                264              0.2903226
## 5                264              0.3333333
## 6                264              0.3636364
##                                 shortDataSetName
## 1 Reticulocytes_genesCorrelatedAcrossIndividuals
## 2                                          IL-17
## 3                                           MTor
## 4                                           MAPK
## 5                                    TCR signaling
## 6                                          IL-1R1
##                                    overlapGenes
## 1                                     150|10098
## 2                                          2932
## 3                                          2932
## 4 5604|5594|5601|8550|51701|5058|5894|6197|7043
## 5                       5604|5594|5601|5058|5894
## 6                            5604|5594|5601|7334
```

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 6.1 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
## 1 active.1    1 GO:0005829                              cytosol
## 2 active.1    2 GO:0051641                 cellular localization
## 3 active.1    3 GO:0005737                              cytoplasm
```

```
## 4 active.1    4 GO:0070727 cellular macromolecule localization
## 5 active.1    5 GO:0008104                protein localization
## 6 active.1    6 GO:0005622  intracellular anatomical structure
##      inGroups       pValue   Bonferroni          FDR nCommonGenes
## 1 GO|GO.CC|GO 2.067764e-19 4.665083e-15 4.665083e-15          430
## 2 GO|GO.BP|GO 5.350136e-17 1.207044e-12 6.035221e-13          292
## 3 GO|GO.CC|GO 2.972165e-16 6.705502e-12 2.235167e-12          718
## 4 GO|GO.BP|GO 6.979968e-16 1.574751e-11 3.771206e-12          231
## 5 GO|GO.BP|GO 8.357799e-16 1.885603e-11 3.771206e-12          230
## 6 GO|GO.CC|GO 2.556110e-15 5.766839e-11 9.611399e-12          808
##   fracOfEffectiveClassSize expectedFracOfEffectiveClassSize
## 1                0.4965358                         0.3536212
## 2                0.3371824                         0.2191085
## 3                0.8290993                         0.7145109
## 4                0.2667436                         0.1635870
## 5                0.2655889                         0.1628864
## 6                0.9330254                         0.8487608
##   enrichmentRatio classSize effectiveClassSize fracOfEffectiveSetSize
## 1        1.404146       871                866             0.10648836
## 2        1.538883       871                866             0.11670663
## 3        1.160373       871                866             0.08800098
## 4        1.630592       871                866             0.12366167
## 5        1.630516       871                866             0.12365591
## 6        1.099280       871                866             0.08336773
##                      shortDataSetName
## 1                             cytosol
## 2               cellular localization
## 3                           cytoplasm
## 4 cellular macromolecule localization
## 5                protein localization
## 6  intracellular anatomical structure
##                        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 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.

6.1 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.

6.2 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.na(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           HTP          Inferred from High Throughput Experiment
## 3           HDA        Inferred from High Throughput Direct Assay
## 4           HMP    Inferred from High Throughput Mutant Phenotype
## 5           HGI Inferred from High Throughput Genetic Interaction
## 6           HEP  Inferred from High Throughput Expression Pattern
## 7           IDA                          Inferred from Direct Assay
## 8           IPI                Inferred from Physical Interaction
## 9           IMP                    Inferred from Mutant Phenotype
## 10          IGI                 Inferred from Genetic Interaction
## 11          IEP                  Inferred from Expression Pattern
## 12          ISS   Inferred from Sequence or Structural Similarity
## 13          ISO                  Inferred from Sequence Orthology
## 14          ISA                  Inferred from Sequence Alignment
## 15          ISM                      Inferred from Sequence Model
## 16          IGC                    Inferred from Genomic Context
## 17          IBA       Inferred from Biological aspect of Ancestor
## 18          IBD     Inferred from Biological aspect of Descendant
## 19          IKR                          Inferred from Key Residues
## 20          IMR                    Inferred from Missing Residues
## 21          IRD                    Inferred from Rapid Divergence
## 22          RCA     Inferred from Reviewed Computational Analysis
## 23          TAS                          Traceable Author Statement
## 24          NAS                    Non-traceable Author Statement
## 25           IC                                Inferred by Curator
## 26           ND                      No biological Data available
## 27          IEA               Inferred from Electronic Annotation
## 28           NR                                      Not Recorded
## 29        other                                             other
##                    evidenceType
## 1                  Experimental
## 2  High throughput experimental
## 3  High throughput experimental
## 4  High throughput experimental
## 5  High throughput experimental
## 6  High throughput experimental
## 7                  Experimental
## 8                  Experimental
## 9                  Experimental
## 10                 Experimental
## 11                 Experimental
## 12                Computational
## 13                Computational
## 14                Computational
## 15                Computational
## 16                Computational
## 17                Computational
```

```
## 18                Computational
## 19                Computational
## 20                Computational
## 21                Computational
## 22                Computational
## 23             Author statement
## 24             Author statement
## 25            Curator statement
## 26            Curator statement
## 27       Automatically assigned
## 28                    Obsolete
## 29                       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.Entrez(
    classLabels = moduleColor, identifiers = entrez,
    refCollection = PLcollection,
    useBackground = "given",
    threshold = 5e-2,
    ID2symbol = entrez2symbol,
    nBestDataSets = 3,
    thresholdType = "Bonferroni");

##  enrichmentAnalysis: preparing data..
##   ..calculating overlaps..

head(PLenrichment$enrichmentTable[, -16])

##            class rank   dataSetID dataSetName inGroups       pValue
## 1          black    1 dummy000001  bb_M00_CTX       PL 4.476065e-78
## 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 8.952131e-77 4.476065e-77          105               1.00000000
## 2 0.000000e+00 0.000000e+00          621               1.00000000
## 3 1.000000e+00 1.000000e+00            7               0.01140065
## 4 1.000000e+00 1.000000e+00            0               0.00000000
## 5 1.000000e+00 1.000000e+00            5               0.01385042
## 6 1.000000e+00 1.000000e+00            0               0.00000000
##   expectedFracOfEffectiveClassSize enrichmentRatio classSize.asGiven
## 1                        0.1949906      5.12845304               188
## 2                        0.1949906      5.12845304              1037
## 3                        0.1949906      0.05846771               868
## 4                        0.1949906      0.00000000                28
## 5                        0.1949906      0.07103121               502
## 6                        0.1949906      0.00000000                14
##   validClassSize effectiveClassSize effectiveSetSize shortDataSetName
## 1            105               105             724       bb_M00_CTX
## 2            621               621             724       bb_M00_CTX
## 3            614               614             724       bb_M00_CTX
## 4             19                19             724       bb_M00_CTX
## 5            361               361             724       bb_M00_CTX
## 6              8                 8             724       bb_M00_CTX
##                           overlapGenes
## 1        (More than 50 overlapping genes)
## 2        (More than 50 overlapping genes)
## 3 5999|26003|6738|10580|23065|84271|11057
## 4
## 5               9774|10513|687|8899|23469
## 6
```

6.3 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
## 1 JAM:003031 RedBloodCell__Kabanova RedBloodCell
##              description                 source organism
## 1 Red blood cell markers Kabanova S, et al. ...    human
##         internalClassification                       groups
## 1 JAM|BloodAtlases|BloodAtl... JAM|BloodAtlases|BloodAtl...
##   lastModified alternateNames externalDB externalAccession webLink
## 1   2011-04-19
```

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.

6.4 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.

6.5 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 `anRichment` 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.

7 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
```

7.1 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.

8 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.

8.1 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"
##  [3] "name"                  "shortName"
##  [5] "description"           "source"
##  [7] "organism"              "internalClassification"
##  [9] "groups"                "lastModified"
## [11] "alternateNames"        "externalDB"
## [13] "externalAccession"     "webLink"
## [15] "weightIndex"           "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"
```

8.2 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"
## [4] "alternateNames" "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.

8.3 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).

9 sessionInfo

Session info and packages loaded by this tutorial:

```
sessionInfo()

## R version 4.4.1 (2024-06-14)
## Platform: x86_64-pc-linux-gnu
## Running under: Fedora Linux 40 (Forty)
##
## Matrix products: default
## BLAS/LAPACK: /usr/lib64/libopenblasp-r0.3.26.so;  LAPACK version 3.12.0
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
```

```
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## time zone: Asia/Taipei
## tzcode source: system (glibc)
##
## attached base packages:
## [1] stats4    stats     graphics  grDevices utils     datasets
## [7] methods   base
##
## other attached packages:
##  [1] org.Hs.eg.db_3.19.1
##  [2] Hmisc_5.1-3
##  [3] anRichment_1.50-1
##  [4] GO.db_3.19.1
##  [5] WGCNA_1.72-5
##  [6] fastcluster_1.2.6
##  [7] dynamicTreeCut_1.63-1
##  [8] TxDb.Mmusculus.UCSC.mm10.knownGene_3.10.0
##  [9] TxDb.Hsapiens.UCSC.hg19.knownGene_3.2.2
## [10] GenomicFeatures_1.56.0
## [11] AnnotationDbi_1.66.0
## [12] Biobase_2.64.0
## [13] GenomicRanges_1.56.1
## [14] GenomeInfoDb_1.40.1
## [15] IRanges_2.38.1
## [16] S4Vectors_0.42.1
## [17] BiocGenerics_0.50.0
## [18] knitr_1.48
##
## loaded via a namespace (and not attached):
##  [1] DBI_1.2.3                   bitops_1.0-7
##  [3] gridExtra_2.3               rlang_1.1.4
##  [5] magrittr_2.0.3              matrixStats_1.3.0
##  [7] compiler_4.4.1              RSQLite_2.3.7
##  [9] png_0.1-8                   vctrs_0.6.5
## [11] stringr_1.5.1               pkgconfig_2.0.3
## [13] crayon_1.5.3                fastmap_1.2.0
## [15] backports_1.5.0             XVector_0.44.0
## [17] utf8_1.2.4                  Rsamtools_2.20.0
## [19] rmarkdown_2.27              preprocessCore_1.44.0
## [21] UCSC.utils_1.0.0            bit_4.0.5
## [23] xfun_0.45                   zlibbioc_1.50.0
## [25] cachem_1.1.0                jsonlite_1.8.8
## [27] blob_1.2.4                  highr_0.11
## [29] DelayedArray_0.30.1         BiocParallel_1.38.0
## [31] parallel_4.4.1              cluster_2.1.6
## [33] R6_2.5.1                    stringi_1.8.4
## [35] rtracklayer_1.64.0          rpart_4.1.23
## [37] Rcpp_1.0.12                 SummarizedExperiment_1.34.0
## [39] iterators_1.0.14            base64enc_0.1-3
## [41] Matrix_1.7-0                splines_4.4.1
```

```
## [43] nnet_7.3-19                tidyselect_1.2.1
## [45] rstudioapi_0.16.0          abind_1.4-5
## [47] yaml_2.3.9                 doParallel_1.0.17
## [49] codetools_0.2-20           curl_5.2.1
## [51] lattice_0.22-6             tibble_3.2.1
## [53] KEGGREST_1.44.1            evaluate_0.24.0
## [55] foreign_0.8-87             survival_3.7-0
## [57] Biostrings_2.72.1          pillar_1.9.0
## [59] MatrixGenerics_1.16.0      checkmate_2.3.1
## [61] foreach_1.5.2              generics_0.1.3
## [63] RCurl_1.98-1.16            ggplot2_3.5.1
## [65] munsell_0.5.1              scales_1.3.0
## [67] glue_1.7.0                 tools_4.4.1
## [69] BiocIO_1.14.0              data.table_1.15.4
## [71] GenomicAlignments_1.40.0   XML_3.99-0.17
## [73] grid_4.4.1                 impute_1.78.0
## [75] colorspace_2.1-0           GenomeInfoDbData_1.2.12
## [77] htmlTable_2.4.2            restfulr_0.0.15
## [79] Formula_1.2-5              cli_3.6.3
## [81] fansi_1.0.6                S4Arrays_1.4.1
## [83] dplyr_1.1.4                gtable_0.3.5
## [85] digest_0.6.36              SparseArray_1.4.8
## [87] rjson_0.2.21               htmlwidgets_1.6.4
## [89] memoise_2.0.1              htmltools_0.5.8.1
## [91] lifecycle_1.0.4            httr_1.4.7
## [93] bit64_4.0.5
```

References

[1] Lewis Y. Geer, Aron Marchler-Bauer, Renata C. Geer, Lianyi Han, Jane He, Siqian He, Chunlei Liu, Wenyao Shi, and Stephen H. Bryant. The NCBI BioSystems database. *Nucleic Acids Research*, 38(suppl_1):D492–D496, 10 2009.

[2] 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.

[3] 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.

[4] 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.

[5] W. Yu, M. Clyne, M. J. Khoury, and M. Gwinn. Phenopedia and Genopedia: disease-centered and gene-centered views of the evolving knowledge of human genetic associations. *Bioinformatics*, 26(1):145–146, 10 2009.