Package ‘WGCNA’
January 14, 2016

Version 1.49
Date 2015-12-27
Title Weighted Correlation Network Analysis
Author Peter Langfelder <Peter.Langfelder@gmail.com> and Steve Horvath <SHorvath@mednet.ucla.edu> with contributions by Chaochao Cai, Jun Dong, Jeremy Miller, Lin Song, Andy Yip, and Bin Zhang
Maintainer Peter Langfelder <Peter.Langfelder@gmail.com>
Depends R (>= 3.0), dynamicTreeCut (>= 1.62), fastcluster
Imports stats, grDevices, utils, matrixStats (>= 0.8.1), Hmisc, impute, splines, foreach, doParallel, preprocessCore, survival, parallel, GO.db, AnnotationDbi
Suggests org.Hs.eg.db, org.Mm.eg.db, infotheo, entropy, minet
ZipData no
License GPL (>= 2)
Description Functions necessary to perform Weighted Correlation Network Analysis on high-dimensional data. Includes functions for rudimentary data cleaning, construction of correlation networks, module identification, summarization, and relating of variables and modules to sample traits. Also includes a number of utility functions for data manipulation and visualization.
URL http://www.genetics.ucla.edu/labs/horvath/CoexpressionNetwork/Rpackages/WGCNA/

R topics documented:

WGCNA-package .............................................. 5
accuracyMeasures ........................................... 11
addErrorBars ............................................... 13
addGrid ...................................................... 13
addGuideLines .............................................. 14
addTraitToMEs .............................................. 15
adjacency .................................................... 16
adjacency.polyReg ........................................ 17
adjacency.splineReg ....................................... 19
AFcorMI ..................................................... 21
alignExpr ................................................... 22
allocateJobs ............................................... 22
allowWGCNAThreads ....................................... 23
### R topics documented:

- `automaticNetworkScreening` .................................................. 24
- `automaticNetworkScreeningGS` ................................................. 26
- `bicor` ..................................................................................... 27
- `bicorAndPvalue` ................................................................. 29
- `bicovWeights` ............................................................... 31
- `blockSize` ............................................................................ 32
- `blockwiseConsensusModules` .................................................. 33
- `blockwiseIndividualTOMs` .................................................... 43
- `blockwiseModules` .......................................................... 47
- `BloodLists` ........................................................................... 54
- `blueWhiteRed` ................................................................. 55
- `BrainLists` ............................................................................ 56
- `BrainRegionMarkers` ........................................................... 56
- `branchEigengeneDissim` ....................................................... 57
- `branchSplit` ........................................................................... 58
- `branchSplit.dissim` ............................................................ 59
- `branchSplitFromStabilityLabels` ............................................. 59
- `checkAdjMat` ....................................................................... 61
- `checkSets` ............................................................................ 61
- `chooseOneHubInEachModule` .................................................. 62
- `chooseTopHubInEachModule` .................................................. 64
- `clusterCoef` .......................................................................... 65
- `coClustering` ......................................................................... 65
- `coClustering.permutationTest` ............................................... 66
- `collapseRows` ....................................................................... 68
- `collapseRowsUsingKME` ......................................................... 73
- `collectGarbage` ................................................................... 74
- `colQuantileC` ....................................................................... 74
- `conformityBasedNetworkConcepts` ........................................ 75
- `conformityDecomposition` ...................................................... 77
- `consensusDissTOMandTree` ..................................................... 79
- `consensusKME` ..................................................................... 81
- `consensusMEDissimilarity` ...................................................... 85
- `consensusOrderMEs` ............................................................ 86
- `consensusProjectiveKMeans` .................................................. 87
- `consensusTOM` ..................................................................... 89
- `cor` ...................................................................................... 96
- `corAndPvalue` ....................................................................... 99
- `corPredictionSuccess` .......................................................... 100
- `corPvalueFisher` .............................................................. 101
- `corPvalueStudent` ............................................................ 102
- `correlationPreservation` ....................................................... 102
- `coxRegressionResiduals` ....................................................... 103
- `cutreeStatic` ........................................................................ 105
- `cutreeStaticColor` ............................................................. 106
- `displayColors` ..................................................................... 106
- `dynamicMergeCut` .............................................................. 107
- `empiricalBayesLM` ............................................................ 108
- `exportNetworkToCytoscape` ................................................ 111
- `exportNetworkToVisANT` ..................................................... 112
- `fixDataStructure` .............................................................. 113
- `formatLabels` ...................................................................... 114
fundamentalNetworkConcepts ........................................... 115
GOenrichmentAnalysis ................................................. 117
goodGenes ................................................................. 120
goodGenesMS ............................................................... 122
goodSamples ............................................................... 123
goodSamplesGenes ....................................................... 124
goodSamplesGenesMS .................................................... 126
goodSamplesMS ............................................................ 127
greenBlackRed ............................................................. 128
greenWhiteRed ............................................................. 129
GTOMdist ................................................................. 130
hubGeneSignificance ..................................................... 131
ImmunePathwayLists ..................................................... 131
Inline display of progress .............................................. 132
intramodularConnectivity .............................................. 133
isMultiData ............................................................... 135
keepCommonProbes ....................................................... 136
kMEcomparisonScatterplot ............................................. 137
labeledBarplot .............................................................. 138
labeledHeatmap ........................................................... 140
labeledHeatmap.multiPage .............................................. 144
labelPoints ................................................................. 147
labels2colors .............................................................. 148
list2multiData .............................................................. 150
lowerTri2matrix ............................................................ 150
matchLabels ............................................................... 151
matrixToNetwork ......................................................... 153
mergeCloseModules ....................................................... 154
metaAnalysis .............................................................. 157
metaZfunction ............................................................ 162
moduleColor.getMEprefix .............................................. 163
moduleEigengenes ........................................................ 163
moduleMergeUsingKME .................................................. 167
moduleNumber ............................................................ 169
modulePreservation ...................................................... 170
mtd.apply ................................................................. 174
mtd.mapply ............................................................... 176
mtd.rbindSelf ............................................................. 178
mtd.setAttr ............................................................... 179
mtd.setColnames ........................................................ 180
mtd.simplify ............................................................. 181
mtd.subset ............................................................... 182
multiData ................................................................. 183
multiData.eigengeneSignificance ..................................... 184
multiSetMEs ............................................................... 185
multiUnion ............................................................... 189
mutualInfoAdjacency .................................................... 189
nearestCentroidPredictor .............................................. 192
nearestNeighborConnectivity ......................................... 196
nearestNeighborConnectivityMS ..................................... 197
networkConcepts .......................................................... 198
networkScreening ......................................................... 201
R topics documented:

- networkScreeningGS ........................................ 203
- normalizeLabels ........................................... 204
- nPresent .................................................... 204
- nSets ....................................................... 205
- numbers2colors ............................................ 205
- orderBranchesUsingHubGenes ............................... 206
- orderMEs ................................................... 209
- overlapTable ............................................... 210
- overlapTableUsingKME ...................................... 211
- pickHardThreshold ......................................... 213
- pickSoftThreshold ......................................... 215
- plotClusterTreeSamples .................................... 217
- plotColorUnderTree ........................................ 219
- plotCor ..................................................... 221
- plotDendroAndColors ....................................... 221
- plotEigengeneNetworks ..................................... 224
- plotMat ...................................................... 226
- plotMEpairs ................................................ 227
- plotModuleSignificance ..................................... 228
- plotNetworkHeatmap ........................................ 229
- populationMeansInAdmixture .................. 230
- pquantile .................................................. 232
- prepComma ................................................. 234
- prependZeros .............................................. 234
- preservationNetworkConnectivity ..................... 235
- projectiveKMeans ......................................... 237
- proportionsInAdmixture .................................. 239
- propVarExplained ......................................... 241
- PWLists .................................................... 242
- qvalue ...................................................... 242
- qvalue.restricted ......................................... 244
- randIndex .................................................. 244
- rankPvalue ................................................ 245
- recutBlockwiseTrees ....................................... 247
- recutConsensusTrees ...................................... 251
- redWhiteGreen ............................................ 255
- relativeCorPredictionSuccess ......................... 256
- removeGreyME .............................................. 257
- removePrincipalComponents ............................... 257
- returnGeneSetsAsList ...................................... 258
- rgcolors.func ............................................. 260
- scaleFreeFitIndex ......................................... 260
- scaleFreePlot ............................................... 261
- SCsLists ................................................... 262
- setCorrelationPreservation ............................... 263
- shortenStrings ............................................. 264
- sigmoidAdjacencyFunction ................................ 265
- signedKME ................................................ 266
- signumAdjacencyFunction ................................ 267
- simulateDatExpr ........................................... 267
- simulateDatExpr5Modules ................................. 270
- simulateEigengeneNetwork ............................... 272
Description

Functions necessary to perform Weighted Correlation Network Analysis. WGCNA is also known as weighted gene co-expression network analysis when dealing with gene expression data. Many functions of WGCNA can also be used for general association networks specified by a symmetric adjacency matrix.

Details

Package: WGCNA
Version: 1.49
Date: 2015-12-27
Depends: R (>= 3.0), dynamicTreeCut (>= 1.62), fastcluster, Hmisc
Imports: stats, impute, grDevices, utils, splines, reshape, foreach, doParallel, matrixStats (>= 0.8.1), GO.db, AnnotationDbi
Suggests: org.Hs.eg.db, org.Mm.eg.db, infotheo, entropy, minet, survival
ZipData: no
License: GPL (>= 2)
URL: http://www.genetics.ucla.edu/labs/horvath/CoexpressionNetwork/Rpackages/WGCNA/
Index:

GTOMdist Generalized Topological Overlap Measure
TOMdist Topological overlap matrix dissimilarity
TOMplot Graphical representation of the Topological Overlap Matrix
TOMsimilarity Topological overlap matrix similarity
TOMsimilarityFromExpr Topological overlap matrix similarity
WGCNA-package Weighted Gene Co-Expression Network Analysis
accuracyMeasures Accuracy measures for a 2x2 confusion matrix
addErrorBars Add error bars to a barplot.
addGrid Add grid lines to an existing plot.
addGuideLines Add vertical "guide lines" to a dendrogram plot
addTraitToMEs Add trait information to multi-set module
eigengene structure
adjacency Calculate network adjacency
adjacency.fromSimilarity Calculate network adjacency from a similarity matrix
adjacency.polyReg Adjacency based on polynomial regression
adjacency.splineReg Adjacency based on natural cubic spline regression
alignExpr Align expression data with given vector
automaticNetworkScreening One-step automatic network gene screening
automaticNetworkScreeningGS One-step automatic network gene screening with external gene significance
AFcorMI Prediction of weighted mutual information adjacency matrix by correlation
bicor Biweight Midcorrelation
bicorAndPvalue Biweight Midcorrelation and the associated p-value
blockwiseConsensusModules Find consensus modules across several datasets.
blockwiseIndividualTOMs Calculate individual topological overlaps across multi-set data
blockwiseModules Automatic network construction and module detection
BloodLists (data) Gene sets for user enrichment analysis
blueWhiteRed Blue-white-red color sequence
BrainLists (data) Gene sets for user enrichment analysis
BrainRegionMarkers (data) Gene Markers for Regions of the Human Brain
checkAdjMat Check adjacency matrix
checkSets Check structure and retrieve sizes of a group of datasets
checkSimilarity Check a similarity matrix
chooseOneHubInEachModule Choose a hub gene in each module
chooseTopHubInEachModule Choose the top hub gene in each module
clusterCoef Clustering coefficient calculation
coClustering Cluster preservation based on co-clustering
coClustering.permutationTest Permutation test for co-clustering
collapseRows Collapse Rows in Numeric Matrix
collapseRowsUsingKME Selects one representative row per group based on KM
collectGarbage Iterative garbage collection
colQuantileC Fast column-wise quantile of a matrix
conformityBasedNetworkConcepts Calculation of conformity-based network concepts
conformityDecomposition Conformity vector(s) and factorizability measure(s) of a network
consensusDissTOMandTree Consensus TOM-based dissimilarity and clustering tree
consensusKME Consensus eigengene-based connectivity
consensusMEDissimilarity Consensus dissimilarity of module eigengenes
consensusOrderMEs Put close eigenvectors next to each other in several sets.
consensusProjectiveKMeans Consensus projective K-means (pre-)clustering of expression data
cor Faster calculation of Pearson correlations
corAndPvalue Correlation and the associated p-value
cor1 Faster calculation of column correlations of a matrix
corfast Faster calculation of Pearson correlations
corPredictionSuccess \(^{-}\)function to do ... ~
corPvalueFisher Fisher's asymptotic p-value for correlation
corPvalueStudent Student asymptotic p-value for correlation
correlationPreservation Preservation of eigengene correlations
coxRegressionResiduals Deviance- and martingale residuals from a Cox regression model
cutreeStatic Constant height tree cut
cutreeStaticColor Constant height tree cut using color labels
displayColors Show colors used to label modules
dynamicMergeCut Threshold for module merging
exportNetworkToVisANT Export network data in format readable by VisANT
exportNetworkToCytoscape Export network data in format readable by Cytoscape
fixDataStructure Put single-set data into a form useful for multiset calculations
fundamentalNetworkConcepts Calculation of fundamental network concepts
GOenrichmentAnalysis Calculate enrichment p-values of clusters in GO terms
goodGenes Filter genes with too many missing entries
goodGenesMS Filter genes with too many missing entries across multiple sets
goodSamples Filter samples with too many missing entries
goodSamplesGenes Iterative filtering of samples and genes with too many missing entries
goodSamplesGenesMS Iterative filtering of samples and genes with too many missing entries across multiple data sets
goodSamplesMS Filter samples with too many missing entries across multiple data sets
greenBlackRed Green-black-red color sequence
greenWhiteRed Green-white-red color sequence
hubGeneSignificance Hubgene significance
ImmunePathwayLists (data) Immune Pathways with Corresponding Gene Markers
initProgInd
  Inline display of progress
intramodularConnectivity
intramodularConnectivity$.fromExpr
  Calculation of intramodular connectivity
keepCommonProbes
  Keep probes that are shared among given data sets
kMEcomparisonScatterplot
  Scatterplots of eigengene-based connectivity
labeledBarplot
  Barplot with text or color labels
labeledHeatmap
  Produce a labeled heatmap plot
labelPoints
  Attempt to intelligently label points in a scatterplot
labels2colors
  Convert numerical labels to colors
lowerTri2matrix
  Reconstruct a symmetric matrix from a distance
  (lower-triangular) representation
matchLabels
  Relabel modules to best approximate a reference labeling
mergeCloseModules
  Merge close modules in gene expression data
metaAnalysis
  Meta-analysis significance statistics
metazFunction
  Meta-analysis Z statistic
moduleColor.getMEprefix
  Get the prefix used to label module eigengenes
moduleEigengenes
  Calculate module eigengenes
moduleMergeUsingKME
  Merge modules and reassign genes using KME
moduleNumber
  Fixed-height cut of a dendrogram
modulePreservation
  Calculation of module preservation statistics
multiSetMEs
  Calculate module eigengenes
multiData.eigengeneSignificance
  Calculate eigengene significance for multiple data sets
mutualInfoAdjacency
  Calculate weighted adjacency matrices based on mutual information
nPresents
  Number of present data entries
nearestNeighborConnectivity
  Connectivity to a constant number of nearest neighbors
nearestNeighborConnectivityMS
  Connectivity to a constant number of nearest neighbors across multiple data sets
nearestCentroidPredictor
  Nearest centroid predictor for two-class problems
networkConcepts
  Calculations of network concepts
networkScreening
  Network screening
networkScreeningGS
  Network screening with external gene significance
normalizeLabels
  Transform numerical labels into normal order
numbers2colors
  Color representation for a numeric variable
orderBranchesUsingHubGenes
  Optimize dendrogram using branch swaps and reflections
orderMEs
  Put close eigenvectors next to each other
overlapTable
  Overlap counts and Fisher exact tests for two sets of module labels
overlapTableUsingKME
  Overlap counts and Fisher exact tests for two sets of module labels based on KME
pickHardThreshold
  Determines significant overlap between modules in two networks based on kME
pickHardThreshold$.fromSimilarity
  Analysis of scale free topology for hard-thresholding
pickSoftThreshold
  Analysis of scale free topology for soft-thresholding
pickSoftThreshold$.fromSimilarity
  Analysis of scale free topology for soft-thresholding
plotClusterTreeSamples
Annotated clustering dendrogram of microarray samples
plotColorUnderTree
plotDendroAndColors
plotEigengeneNetworks
plotMEpairs
plotModuleSignificance
plotNetworkHeatmap
pmean
pmedian
populationMeansInAdmixture
Estimation of population-specific mean values in an admixed population
pquantile
preservationNetworkConnectivity
Network preservation calculations
projectiveKMeans
Projective K-means (pre-)clustering of expression data
propVarExplained
Proportion of variance explained by eigengenes
proportionsInAdmixture
Estimation of proportion of pure populations in an admixture
qvalue
q-value calculation from package qvalue
randIndex
Calculation of (adjusted) Rand index
randomGLMPredictor
Ensemble predictor based on bagging of generalized linear models
rankPvalue
Estimate the p-value for ranking consistently high (or low) on multiple lists
recutBlockwiseTrees
Repeat blockwise module detection from pre-calculated data
recutConsensusTrees
Repeat blockwise consensus module detection from pre-calculated data
redWhiteGreen
Red-white-green color sequence
reflectTwoBranches
Reflect branches in a dendrogram
relativeCorPredictionSuccess
Compare prediction success
removeGreyME
Removes the grey eigengene from a given collection of eigengenes.
removePrincipalComponents
Remove leading principal components from data
returnGeneSetsAsLists
Return pre-defined gene lists in several biomedical categories.
scaleFreeFitIndex
Calculation of fitting statistics for evaluating scale free topology fit.
scaleFreePlot
Visual check of scale-free topology
SCsLists
(data) Stem Cell-Related Genes with Corresponding Gene Markers
selectBranch
Find a branch in a dendrogram
setCorrelationPreservation
Summary correlation preservation measure
sigmoidAdjacencyFunction
Sigmoid-type adjacency function
signedKME
Signed eigengene-based connectivity
signumAdjacencyFunction
Hard-thresholding adjacency function
simulateDatExpr
Simulation of expression data
simulateDatExpr5Modules
simulateEigengeneNetwork
Simulate eigengene network from a causal model

**simulateModule**
Simulate a gene co-expression module

**simulateMultiExpr**
Simulate multi-set expression data

**simulateSmallLayer**
Simulate small modules

**sizeGrWindow**
Open a graphics window of given width and height

**softConnectivity**
Calculation of soft (weighted) connectivity

**softConnectivityFromSimilarity**
Calculation of soft (weighted) connectivity

**spaste**
Space-less paste

**standardColors**
Colors this library uses for labeling modules

**standardScreeningBinaryTrait**
Standard screening for a binary trait

**standardScreeningCensoredTime**
Standard screening with regard to a Censored Time Variable

**stderr**
Standard error

**stratifiedBarplot**
Bar plots of data across two splitting parameters

**swapTwoBranches**
Swap branches in a dendrogram

**TrueTrait**
Estimate the true trait underlying a list of surrogate markers

**transposeBigData**
Block-by-block transpose of large matrices

**unsignedAdjacency**
Calculation of unsigned adjacency

**userListEnrichment**
Measure enrichment between inputted and user-defined lists

**vectorTOM**
Topological overlap for a subset of the whole set of genes

**vectorizeMatrix**
Turn a matrix into a vector of non-redundant components

**verboseBarplot**
Barplot with error bars, annotated by Kruskal-Wallis p-value

**verboseBoxplot**
Boxplot annotated by a Kruskal-Wallis p-value

**verboseScatterplot**
Scatterplot annotated by regression line and p-value

**verboseIplot**
Scatterplot annotated by regression line, p-value, and color for density

**votingLinearPredictor**
Voting linear predictor

---

**Author(s)**

Peter Langfelder <Peter.Langfelder@gmail.com> and Steve Horvath <SHorvath@mednet.ucla.edu>, with contributions by Jun Dong, Jeremy Miller, Lin Song, Andy Yip, and Bin Zhang

Maintainer: Peter Langfelder <Peter.Langfelder@gmail.com>

---

**References**


accuracyMeasures


accuracyMeasures

Accuracy measures for a 2x2 confusion matrix or for vectors of predicted and observed values.

Description

The function calculates various prediction accuracy statistics for predictions of binary or quantitative (continuous) responses. For binary classification, the function calculates the error rate, accuracy, sensitivity, specificity, positive predictive value, and other accuracy measures. For quantitative prediction, the function calculates correlation, R-squared, error measures, and the C-index.

Usage

accuracyMeasures(
predicted,
observed = NULL,
type = c("auto", "binary", "quantitative"),
levels = if (isTRUE(all.equal(dim(predicted), c(2,2)))) colnames(predicted)
else if (is.factor(predicted))
  sort(unique(c(as.character(predicted), as.character(observed))))
else sort(unique(c(observed, predicted))),
negativeLevel = levels[2],
positiveLevel = levels[1])

Arguments

predicted either a a 2x2 confusion matrix (table) whose entries contain non-negative integers, or a vector of predicted values. Predicted values can be binary or quantitative (see type below). If a 2x2 matrix is given, it must have valid column and row names that specify the levels of the predicted and observed variables whose counts the matrix is giving (e.g., the function table sets the names appropriately.) If it is a 2x2 table and the table contains non-negative real (non-integer) numbers the function outputs a warning.

observed if predicted is a vector of predicted values, this (observed) must be a vector of the same length giving the "gold standard" (or observed) values. Ignored if predicted is a 2x2 table.

type character string specifying the type of the prediction problem (i.e., values in the predicted and observed vectors). The default "auto" decides type automatically: if predicted is a 2x2 table or if the number of unique values in the concatenation of predicted and observed is 2, the prediction problem (type) is assumed to be binary, otherwise it is assumed to be quantitative. Inconsistent specification (for example, when predicted is a 2x2 matrix and type is "quantitative") trigger errors.
**levels**

A 2-element vector specifying the two levels of binary variables. Only used if type is "binary" (or "auto" that results in the binary type). Defaults to either the column names of the confusion matrix (if the matrix is specified) or to the sorted unique values of observed and opredicted.

**negativeLevel**

The binary value (level) that corresponds to the negative outcome. Note that the default is the second of the sorted levels (for example, if levels are 1,2, the default negative level is 2). Only used if type is "binary" (or "auto" that results in the binary type).

**positiveLevel**

The binary value (level) that corresponds to the positive outcome. Note that the default is the second of the sorted levels (for example, if levels are 1,2, the default negative level is 2). Only used if type is "binary" (or "auto" that results in the binary type).

**Details**

The rows of the 2x2 table tab must correspond to a test (or predicted) outcome and the columns to a true outcome ("gold standard"). A table that relates a predicted outcome to a true test outcome is also known as confusion matrix. Warning: To correctly calculate sensitivity and specificity, the positive and negative outcome must be properly specified so they can be matched to the appropriate rows and columns in the confusion table.

Interchanging the negative and positive levels swaps the estimates of the sensitivity and specificity but has no effect on the error rate or accuracy. Specifically, denote by pos the index of the positive level in the confusion table, and by neg the index of the negative level in the confusion table. The function then defines number of true positives=TP=tab[pos, pos], no.false positives =FP=tab[pos, neg], no.false negatives=FN=tab[neg, pos], no.true negatives=TN=tab[neg, neg]. Then Specificity=TN/(FP+TN) Sensitivity=TP/(TP+FN) NegativePredictiveValue=TN/(FN + TN) PositivePredictiveValue=TP/(TP + FP) FalsePositiveRate = 1-Specificity FalseNegativeRate = 1-Sensitivity Power = Sensitivity LikelihoodRatioPositive = Sensitivity / (1-Specificity) LikelihoodRatioNegative = (1-Sensitivity)/Specificity. The naive error rate is the error rate of a constant (naive) predictor that assigns the same outcome to all samples. The prediction of the naive predictor equals the most frequently observed outcome. Example: Assume you want to predict disease status and 70 percent of the observed samples have the disease. Then the naive predictor has an error rate of 30 percent (since it only misclassifies 30 percent of the healthy individuals).

**Value**

Data frame with two columns:

**Measure**

This column contains character strings that specify name of the accuracy measure.

**Value**

This column contains the numeric estimates of the corresponding accuracy measures.

**Author(s)**

Steve Horvath and Peter Langfelder

**References**

**addErrorBars**

Examples

```r
m=100
trueOutcome=sample( c(1,2), m, replace=TRUE)
predictedOutcome=trueOutcome
# now we noise half of the entries of the predicted outcome
predictedOutcome[ 1:(m/2) ] = sample( predictedOutcome[ 1:(m/2) ] )
tab=table( predictedOutcome, trueOutcome )
accuracyMeasures( tab )
# Same result:
accuracyMeasures( predictedOutcome, trueOutcome )
```

---

**Description**

This function adds error bars to an existing barplot.

**Usage**

```r
addErrorBars( means, errors, two.side = FALSE )
```

**Arguments**

- **means**: vector of means plotted in the barplot
- **errors**: vector of standard errors (single positive values) to be plotted.
- **two.side**: should the error bars be two-sided?

**Value**

None.

**Author(s)**

Steve Horvath and Peter Langfelder

---

**addGrid**

Add grid lines to an existing plot.

**Description**

This function adds horizontal and/or vertical grid lines to an existing plot. The grid lines are aligned with tick marks.

**Usage**

```r
addGrid( linesPerTick = NULL, horiz = TRUE, vert = FALSE, col = "grey30", lty = 3 )
```
**Arguments**

- `linesPerTick`: Number of lines between successive tick marks (including the line on the tick-marks themselves)
- `horiz`: Draw horizontal grid lines?
- `vert`: Draw vertical tick lines?
- `col`: Specifies color of the grid lines
- `lty`: Specifies line type of grid lines. See `par`.

**Details**

If `linesPerTick` is not specified, it is set to 5 if number of ticks is 5 or less, and it is set to 2 if number of ticks is greater than 5.

**Note**

The function does not work whenever logarithmic scales are in use.

**Author(s)**

Peter Langfelder

**Examples**

```r
plot(c(1:10), c(1:10))
addGrid();
```

---

**Description**

Adds vertical “guide lines” to a dendrogram plot.

**Usage**

```r
addGuideLines(dendro,
    all = FALSE,
    count = 50,
    positions = NULL,
    col = "grey30",
    lty = 3,
    hang = 0)
```
**Description**

Adds trait information to multi-set module eigengene structure.

**Usage**

```r
addTraitToMEs(multiME, multiTraits)
```

**Arguments**

- `multiME` Module eigengenes in multi-set format. A vector of lists, one list per set. Each list must contain an element named `data` that is a data frame with module eigengenes.
- `multiTraits` Microarray sample trait(s) in multi-set format. A vector of lists, one list per set. Each list must contain an element named `data` that is a data frame in which each column corresponds to a trait, and each row to an individual sample.

**Details**

The function simply `cbind`s the module eigengenes and traits for each set. The number of sets and numbers of samples in each set must be consistent between `multiMEs` and `multiTraits`.

**Value**

A multi-set structure analogous to the input: a vector of lists, one list per set. Each list will contain a component `data` with the merged eigengenes and traits for the corresponding set.

**Author(s)**

Peter Langfelder

**See Also**

`checkSets, moduleEigengenes`
adjacency

Calculate network adjacency

Description

Calculates (correlation or distance) network adjacency from given expression data or from a similarity.

Usage

adjacency(datExpr,
   selectCols = NULL,
   type = "unsigned",
   power = if (type=="distance") 1 else 6,
   corFnc = "cor", corOptions = "use = 'p'",
   distFnc = "dist", distOptions = "method = 'euclidean'"
)

adjacency.fromSimilarity(similarity,
   type = "unsigned",
   power = if (type=="distance") 1 else 6)

Arguments

datExpr  data frame containing expression data. Columns correspond to genes and rows to samples.
similarity a (signed) similarity matrix: square, symmetric matrix with entries between -1 and 1.
selectCols for correlation networks only (see below); can be used to select genes whose adjacencies will be calculated. Should be either a numeric vector giving the indices of the genes to be used, or a boolean vector indicating which genes are to be used.
type network type. Allowed values are (unique abbreviations of) "unsigned", "signed", "signed hybrid", "distance".
power soft thresholding power.
corFnc character string specifying the function to be used to calculate co-expression similarity for correlation networks. Defaults to Pearson correlation. Any function returning values between -1 and 1 can be used.
corOptions character string specifying additional arguments to be passed to the function given by corFnc. Use "use = 'p'", method = 'spearman'" to obtain Spearman correlation.
distFnc character string specifying the function to be used to calculate co-expression similarity for distance networks. Defaults to the function dist. Any function returning non-negative values can be used.
distOptions character string specifying additional arguments to be passed to the function given by distFnc. For example, when the function dist is used, the argument method can be used to specify various ways of computing the distance.
Details

The argument type determines whether a correlation (type one of "unsigned", "signed", "signed hybrid"), or a distance network (type equal "distance") will be calculated. In correlation networks the adjacency is constructed from correlations (values between -1 and 1, with high numbers meaning high similarity). In distance networks, the adjacency is constructed from distances (non-negative values, high values mean low similarity).

The function calculates the similarity of columns (genes) in datExpr by calling the function given in corFnc (for correlation networks) or distFnc (for distance networks), transforms the similarity according to type and raises it to power, resulting in a weighted network adjacency matrix. If selectCols is given, the corFnc function will be given arguments (datExpr, datExpr[selectCols], ...); hence the returned adjacency will have rows corresponding to all genes and columns corresponding to genes selected by selectCols.

Correlation and distance are transformed as follows: for type = "unsigned", adjacency = |cor|^power; for type = "signed", adjacency = (0.5 * (1+cor))^power; for type = "signed hybrid", adjacency = cor^power if cor>0 and 0 otherwise; and for type = "distance", adjacency = (1-(dist/max(dist))^2)^power.

The function adjacency.fromSimilarity inputs a similarity matrix, that is it skips the correlation calculation step but is otherwise identical.

Value

Adjacency matrix of dimensions ncol(datExpr) times ncol(datExpr) (or the same dimensions as similarity). If selectCols was given, the number of columns will be the length (if numeric) or sum (if boolean) of selectCols.

Note

When calculated from the datExpr, the network is always calculated among the columns of datExpr irrespective of whether a correlation or a distance network is requested.

Author(s)

Peter Langfelder and Steve Horvath

References


adjacency.polyReg Adjacency matrix based on polynomial regression
Description

adjacency.polyReg calculates a network adjacency matrix by fitting polynomial regression models to pairs of variables (i.e., pairs of columns from datExpr). Each polynomial fit results in a model fitting index R.squared. Thus, the n columns of datExpr result in an n x n dimensional matrix whose entries contain R.squared measures. This matrix is typically non-symmetric. To arrive at a (symmetric) adjacency matrix, one can specify different symmetrization methods with symmetrizationMethod.

Usage

```
adjacency.polyReg(datExpr, degree=3, symmetrizationMethod = "mean")
```

Arguments

- **datExpr** data frame containing numeric variables. Example: Columns may correspond to genes and rows to observations (samples).
- **degree** the degree of the polynomial. Must be less than the number of unique points.
- **symmetrizationMethod** character string (eg "none", "min","max","mean") that specifies the method used to symmetrize the pairwise model fitting index matrix (see details).

Details

A network adjacency matrix is a symmetric matrix whose entries lie between 0 and 1. It is a special case of a similarity matrix. Each variable (column of datExpr) is regressed on every other variable, with each model fitting index recorded in a square matrix. Note that the model fitting index of regressing variable x and variable y is usually different from that of regressing y on x. From the polynomial regression model glm(y ~ poly(x,degree)) one can calculate the model fitting index R.squared(y,x). R.squared(y,x) is a number between 0 and 1. The closer it is to 1, the better the polynomial describes the relationship between x and y and the more significant is the pairwise relationship between the 2 variables. One can also reverse the roles of x and y to arrive at a model fitting index R.squared(x,y). If degree>1 then R.squared(x,y) is typically different from R.squared(y,x). Assume a set of n variables x1,...,xn (corresponding to the columns of datExpr) then one can define R.squared(xi,xj). The model fitting indices for the elements of an n x n dimensional matrix (R.squared(ij)). symmetrizationMethod implements the following symmetrization methods:

- A.min(ij)=min(R.squared(ij),R.squared(ji)), A.ave(ij)=(R.squared(ij)+R.squared(ji))/2, A.max(ij)=max(R.squared(ij),R.squared(ji)).

Value

An adjacency matrix of dimensions ncol(datExpr) times ncol(datExpr).

Author(s)

Lin Song, Steve Horvath

References


See Also
For more information about polynomial regression, please refer to functions `poly` and `glm`

Examples

```r
# Simulate a data frame datE which contains 5 columns and 50 observations
m=50
x1=rnorm(m)
r=.5; x2=r*x1+sqrt(1-r^2)*rnorm(m)
r=.3; x3=r*(x1-.5)^2+sqrt(1-r^2)*rnorm(m)
x4=rnorm(m)
r=.3; x5=r*x4+sqrt(1-r^2)*rnorm(m)
datE=data.frame(x1,x2,x3,x4,x5)
# calculate adjacency by symmetrizing using max
A.max=adjacency.polyReg(datE, symmetrizationMethod="max")
A.max
# calculate adjacency by symmetrizing using max
A.mean=adjacency.polyReg(datE, symmetrizationMethod="mean")
A.mean
# output the unsymmetrized pairwise model fitting indices R.squared
R.squared=adjacency.polyReg(datE, symmetrizationMethod="none")
R.squared
```

adjacency.splineReg  Calculate network adjacency based on natural cubic spline regression

Description
adjacency.splineReg calculates a network adjacency matrix by fitting spline regression models to pairs of variables (i.e. pairs of columns from `datExpr`). Each spline regression model results in a fitting index $R^2$. Thus, the $n$ columns of `datExpr` result in an $n \times n$ dimensional matrix whose entries contain $R^2$ measures. This matrix is typically non-symmetric. To arrive at a (symmetric) adjacency matrix, one can specify different symmetrization methods with `symmetrizationMethod`.

Usage

```r
adjacency.splineReg(
  datExpr, 
  df = 6-(nrow(datExpr)<100)-(nrow(datExpr)<30),
  symmetrizationMethod = "mean",
  ...
)
```

Arguments

data.frame containing numeric variables. Example: Columns may correspond to genes and rows to observations (samples).

`df`
degrees of freedom in generating natural cubic spline. The default is as follows: if `nrow(datExpr)>100` use 6, if `nrow(datExpr)>30` use 4, otherwise use 5.

`symmetrizationMethod`
character string (eg "none", "min","max","mean") that specifies the method used to symmetrize the pairwise model fitting index matrix (see details).

... other arguments from function `ns`
Details

A network adjacency matrix is a symmetric matrix whose entries lie between 0 and 1. It is a special case of a similarity matrix. Each variable (column of datExpr) is regressed on every other variable, with each model fitting index recorded in a square matrix. Note that the model fitting index of regressing variable x and variable y is usually different from that of regressing y on x. From the spline regression model glm(y ~ ns(x, df)) one can calculate the model fitting index R.squared(y,x). R.squared(y,x) is a number between 0 and 1. The closer it is to 1, the better the spline regression model describes the relationship between x and y and the more significant is the pairwise relationship between the 2 variables. One can also reverse the roles of x and y to arrive at a model fitting index R.squared(x,y). R.squared(x,y) is typically different from R.squared(y,x).

Assume a set of n variables x1, ..., xn (corresponding to the columns of datExpr) then one can define R.squared(xi,xj). The model fitting indices for the elements of an n x n dimensional matrix (R.squared(ij)). symmetrizationMethod implements the following symmetrization methods: A.min(ij)=min(R.squared(ij),R.squared(ji)), A.ave(ij)=(R.squared(ij)+R.squared(ji))/2, A.max(ij)=max(R.squared(ij),R.squared(ji)).

For more information about natural cubic spline regression, please refer to functions "ns" and "glm".

Value

An adjacency matrix of dimensions ncol(datExpr) times ncol(datExpr).

Author(s)

Lin Song, Steve Horvath

References


See Also

ns, glm

Examples

#Simulate a data frame datE which contains 5 columns and 50 observations
m=50
x1=rnorm(m)
rd=.5; x2=r*r*x1+sqrt(1-r^2)*rnorm(m)
rd=.3; x3=r*(x1-.5)^2+sqrt(1-r^2)*rnorm(m)
x4=rnorm(m)
rd=.3; x5=r*x4+sqrt(1-r^2)*rnorm(m)
datE=data.frame(x1,x2,x3,x4,x5)
#calculate adjacency by symmetrizing using max
A.max=adjacency.splineReg(datE, symmetrizationMethod="max")
A.max
#calculate adjacency by symmetrizing using max
A.ave=adjacency.splineReg(datE, symmetrizationMethod="mean")
A.ave
# output the unsymmetrized pairwise model fitting indices R.squared
R.squared=adjacency.splineReg(datE, symmetrizationMethod="none")
R.squared
AFcorMI

Prediction of Weighted Mutual Information Adjacency Matrix by Correlation

Description
AFcorMI computes a predicted weighted mutual information adjacency matrix from a given correlation matrix.

Usage
AFcorMI(r, m)

Arguments
r  a symmetric correlation matrix with values from -1 to 1.
m  number of observations from which the correlation was calculated.

Details
This function is a one-to-one prediction when we consider correlation as unsigned. The prediction corresponds to the AdjacencyUniversalVersion2 discussed in the help file for the function mutualInfoAdjacency. For more information about the generation and features of the predicted mutual information adjacency, please refer to the function mutualInfoAdjacency.

Value
A matrix with the same size as the input correlation matrix, containing the predicted mutual information of type AdjacencyUniversalVersion2.

Author(s)
Steve Horvath, Lin Song, Peter Langfelder

See Also
mutualInfoAdjacency

Examples
#Simulate a data frame datE which contains 5 columns and 50 observations
m=50
x1=rnorm(m)
r=.5; x2=r*x1+sqrt(1-r^2)*rnorm(m)
rc=r^2*(x1-.5)^2+sqrt(1-r^2)*rnorm(m)
x4=rnorm(m)
r=.3; x5=r*x4+sqrt(1-r^2)*rnorm(m)
datE=data.frame(x1,x2,x3,x4,x5)
#calculate predicted AUV2
cor.data=cor(datE, use="p")
AUV2=AFcorMI(r=cor.data, m=mrow(datE))
alignExpr  
*Align expression data with given vector*

**Description**

Multiplies genes (columns) in given expression data such that their correlation with given reference vector is non-negative.

**Usage**

```
alignExpr(datExpr, y = NULL)
```

**Arguments**

- `datExpr` expression data to be aligned. A data frame with columns corresponding to genes and rows to samples.
- `y` reference vector of length equal the number of samples (rows) in `datExpr`.

**Details**

The function basically multiplies each column in `datExpr` by the sign of its correlation with `y`. If `y` is not given, the first column in `datExpr` will be used as the reference vector.

**Value**

A data frame containing the aligned expression data, of the same dimensions as the input data frame.

**Author(s)**

Steve Horvath and Peter Langfelder

allocateJobs  
*Divide tasks among workers*

**Description**

This function calculates an even splitting of a given number of tasks among a given number of workers (threads).

**Usage**

```
allocateJobs(nTasks, nWorkers)
```

**Arguments**

- `nTasks` number of tasks to be divided
- `nWorkers` number of workers
allowWGCNAThreads

Details

Tasks are labeled consecutively 1, 2, ..., nTasks. The tasks are split in contiguous blocks as evenly as possible.

Value

A list with one component per worker giving the task indices to be worked on by each worker. If there are more workers than tasks, the tasks for the extra workers are 0-length numeric vectors.

Author(s)

Peter Langfelder

Examples

```r
allocateJobs(10, 3);
allocateJobs(2, 4);
```

allowWGCNAThreads | Allow and disable multi-threading for certain WGCNA calculations

Description

These functions allow and disable multi-threading for WGCNA calculations that can optionally be multi-threaded, which includes all functions using `cor` or `bicor` functions.

Usage

```r
allowWGCNAThreads(nThreads = NULL)
enableWGCNAThreads(nThreads = NULL)
disableWGCNAThreads()
WGCNAThreads()
```

Arguments

- **nThreads**: Number of threads to allow. If not given, the number of processors online (as reported by system configuration) will be used. There appear to be some cases where the automatically-determined number is wrong; please check the output to see that the number of threads makes sense. Except for testing and/or torturing your system, the number of threads should be no more than the number of actual processors/cores.
Details

allowWGCAThreads enables parallel calculation within the compiled code in WGCNA, principally for calculation of correlations in the presence of missing data. This function is now deprecated; use enableWGCAThreads instead.

enableWGCAThreads enables parallel calculations within user-level R functions as well as within the compiled code, and registers an appropriate parallel calculation back-end for the operating system/platform.

disableWGCAThreads disables parallel processing.

WGCAThreads returns the number of threads (parallel processes) that WGCNA is currently configured to run with.

Value

allowWGCAThreads, enableWGCAThreads, and disableWGCAThreads return the maximum number of threads WGCNA calculations will be allowed to use.

Note

Multi-threading within compiled code is not available on Windows; R code parallelization works on all platforms.

Author(s)

Peter Langfelder

Description

This function performs gene screening based on a given trait and gene network properties

Usage

automaticNetworkScreening(
  datExpr,
  y,
  power = 6,
  networkType = "unsigned",
  detectCutHeight = 0.995,
  minModuleSize = min(20, ncol(as.matrix(datExpr))/2),
  datME = NULL,
  getQValues = TRUE,
  ...)

automaticNetworkScreening

One-step automatic network gene screening
automaticNetworkScreening

Arguments

- datExpr: data frame containing the expression data, columns corresponding to genes and rows to samples.
- y: vector containing trait values for all samples in datExpr.
- power: soft thresholding power used in network construction.
- networkType: character string specifying network type. Allowed values are (unique abbreviations of) "unsigned", "signed", "hybrid".
- detectCutHeight: cut height of the gene hierarchical clustering dendrogram. See cutreeDynamic for details.
- minModuleSize: minimum module size to be used in module detection procedure.
- datME: optional specification of module eigengenes. A data frame whose columns are the module eigengenes. If given, module analysis will not be performed.
- getQValues: logical: should q-values (local FDR) be calculated?
- ...: other arguments to the module identification function blockwiseModules.

Details

Network screening is a method for identifying genes that have a high gene significance and are members of important modules at the same time. If datME is given, the function calls networkScreening with the default parameters. If datME is not given, module eigengenes are first calculated using network analysis based on supplied parameters.

Value

A list with the following components:

- networkScreening: a data frame containing results of the network screening procedure. See networkScreening for more details.
- datME: calculated module eigengenes (or a copy of the input datME, if given).
- hubGeneSignificance: hub gene significance for all calculated modules. See hubGeneSignificance.

Author(s)

Steve Horvath

See Also

networkScreening, hubGeneSignificance, networkScreening, cutreeDynamic
automaticNetworkScreeningGS

*One-step automatic network gene screening with external gene significance*

**Description**

This function performs gene screening based on external gene significance and their network properties.

**Usage**

```r
automaticNetworkScreeningGS(
  datExpr, GS,
  power = 6, networkType = "unsigned",
  detectCutHeight = 0.995, minModuleSize = min(20, ncol(as.matrix(datExpr))/2),
  datME = NULL)
```

**Arguments**

- `datExpr` : data frame containing the expression data, columns corresponding to genes and rows to samples
- `GS` : vector containing gene significance for all genes given in `datExpr`
- `power` : soft thresholding power used in network construction
- `networkType` : character string specifying network type. Allowed values are (unique abbreviations of) "unsigned", "signed", "hybrid".
- `detectCutHeight` : cut height of the gene hierarchical clustering dendrogram. See `cutreeDynamic` for details.
- `minModuleSize` : minimum module size to be used in module detection procedure.
- `datME` : optional specification of module eigengenes. A data frame whose columns are the module eigengenes. If given, module analysis will not be performed.

**Details**

Network screening is a method for identifying genes that have a high gene significance and are members of important modules at the same time. If `datME` is given, the function calls `networkScreeningGS` with the default parameters. If `datME` is not given, module eigengenes are first calculated using network analysis based on supplied parameters.

**Value**

A list with the following components:

- `networkScreening` : a data frame containing results of the network screening procedure. See `networkScreeningGS` for more details.
- `datME` : calculated module eigengenes (or a copy of the input `datME`, if given).
- `hubGeneSignificance` : hub gene significance for all calculated modules. See `hubGeneSignificance`.
**bicor**

**Author(s)**
Steve Horvath

**See Also**

networkScreening, hubGeneSignificance, networkScreening, cutreeDynamic

---

**Description**

Calculate biweight midcorrelation efficiently for matrices.

**Usage**

bicor(x, y = NULL, robustx = TRUE, robusty = TRUE, use = "all.obs", maxPOutliers = 1, quick = 0, pearsonFallback = "individual", cosine = FALSE, cosineX = cosine, cosineY = cosine, nThreads = 0, verbose = 0, indent = 0)

**Arguments**

- **x**: a vector or matrix-like numeric object
- **y**: a vector or matrix-like numeric object
- **robustX**: use robust calculation for \(x\)?
- **robustY**: use robust calculation for \(y\)?
- **use**: specifies handling of NAs. One of (unique abbreviations of) "all.obs", "pairwise.complete.obs".
- **maxPOutliers**: specifies the maximum percentile of data that can be considered outliers on either side of the median separately. For each side of the median, if higher percentile than \(\text{maxPOutliers}\) is considered an outlier by the weight function based on \(9 \times \text{mad}(x)\), the width of the weight function is increased such that the percentile of outliers on that side of the median equals \(\text{maxPOutliers}\). Using \(\text{maxPOutliers}=1\) will effectively disable all weight function broadening; using \(\text{maxPOutliers}=0\) will give results that are quite similar (but not equal to) Pearson correlation.
- **quick**: real number between 0 and 1 that controls the handling of missing data in the calculation of correlations. See details.
pearsonFallback

Specifies whether the bicor calculation should revert to Pearson when median absolute deviation (mad) is zero. Recognized values are (abbreviations of) "none", "individual", "all". If set to "none", zero mad will result in NA for the corresponding correlation. If set to "individual", Pearson calculation will be used only for columns that have zero mad. If set to "all", the presence of a single zero mad will cause the whole variable to be treated in Pearson correlation manner (as if the corresponding robust option was set to FALSE).

cosine

logical: calculate cosine biweight midcorrelation? Cosine bicorrelation is similar to standard bicorrelation but the median subtraction is not performed.

cosineX

logical: use the cosine calculation for x? This setting does not affect y and can be used to give a hybrid cosine-standard bicorrelation.

cosineY

logical: use the cosine calculation for y? This setting does not affect x and can be used to give a hybrid cosine-standard bicorrelation.

nThreads

non-negative integer specifying the number of parallel threads to be used by certain parts of correlation calculations. This option only has an effect on systems on which a POSIX thread library is available (which currently includes Linux and Mac OSX, but excludes Windows). If zero, the number of online processors will be used if it can be determined dynamically, otherwise correlation calculations will use 2 threads.

verbose

if non-zero, the underlying C function will print some diagnostics.

indent

indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

This function implements biweight midcorrelation calculation (see references). If y is not supplied, midcorrelation of columns of x will be calculated; otherwise, the midcorrelation between columns of x and y will be calculated. Thus, bicor(x) is equivalent to bicor(x, x) but is more efficient.

The options robustX, robustY allow the user to revert the calculation to standard correlation calculation. This is important, for example, if any of the variables is binary (or, more generally, discrete) as in such cases the robust methods produce meaningless results. If both robustX, robustY are set to FALSE, the function calculates the standard Pearson correlation (but is slower than the function cor).

The argument quick specifies the precision of handling of missing data in the correlation calculations. Value quick = 0 will cause all calculations to be executed accurately, which may be significantly slower than calculations without missing data. Progressively higher values will speed up the calculations but introduce progressively larger errors. Without missing data, all column medians and median absolute deviations (MADs) can be pre-calculated before the covariances are calculated. When missing data are present, exact calculations require the column medians and MADs to be calculated for each covariance. The approximate calculation uses the pre-calculated median and MAD and simply ignores missing data in the covariance calculation. If the number of missing data is high, the pre-calculated medians and MADs may be very different from the actual ones, thus potentially introducing large errors. The quick value times the number of rows specifies the maximum difference in the number of missing entries for median and MAD calculations on the one hand and covariance on the other hand that will be tolerated before a recalculation is triggered. The hope is that if only a few missing data are treated approximately, the error introduced will be small but the potential speedup can be significant.

The choice "all" for pearsonFallback is not fully implemented in the sense that there are rare but possible cases in which the calculation is equivalent to "individual". This may happen if the
use option is set to "pairwise.complete.obs" and the missing data are arranged such that each individual mad is non-zero, but when two columns are analyzed together, the missing data from both columns may make a mad zero. In such a case, the calculation is treated as Pearson, but other columns will be treated as bicor.

Value

A matrix of biweight midcorrelations. Dimnames on the result are set appropriately.

Author(s)

Peter Langfelder

References


bicorAndPvalue Calculation of biweight midcorrelations and associated p-values

Description

A faster, one-step calculation of Student correlation p-values for multiple biweight midcorrelations, properly taking into account the actual number of observations.

Usage

bicorAndPvalue(x, y = NULL, use = "pairwise.complete.obs", alternative = c("two.sided", "less", "greater"), ...)

Arguments

x a vector or a matrix
y a vector or a matrix. If NULL, the correlation of columns of x will be calculated.
use determines handling of missing data. See bicor for details.
alternative specifies the alternative hypothesis and must be (a unique abbreviation of) one of "two.sided", "greater" or "less": the initial letter. "greater" corresponds to positive association, "less" to negative association.
... other arguments to the function bicor.
bicorAndPvalue

Details
The function calculates the biweight midcorrelations of a matrix or of two matrices and the corresponding Student p-values. The output is not as full-featured as cor.test, but can work with matrices as input.

Value
A list with the following components, each a marix:

- **bicor**: the calculated correlations
- **p**: the Student p-values corresponding to the calculated correlations
- **Z**: Fisher transform of the calculated correlations
- **t**: Student t statistics of the calculated correlations
- **n0bs**: Numbers of observations for the correlation, p-values etc.

Author(s)
Peter Langfelder and Steve Horvath

References

See Also
- **bicor** for calculation of correlations only;
- **cor.test** for another function for significance test of correlations

Examples
```r
# generate random data with non-zero correlation
set.seed(1);
a = rnorm(100);
b = rnorm(100) + a;
x = cbind(a, b);
# Call the function and display all results
bicorAndPvalue(x)
# Set some components to NA
x[c(1:4), 1] = NA
corAndPvalue(x)
# Note that changed number of observations.
```
bicovWeights

Weights used in biweight midcovariance

Description

The function calculates weights used in the calculation of biweight midcovariance and midcorrelation. The weights are designed such that outliers get smaller weights; the weights become zero for data points more than 9 median absolute deviations from the median.

Usage

```
bicovWeights(x, pearsonFallback = TRUE, maxPOutliers = 1, 
   outlierReferenceWeight = 0.5625, 
   defaultWeight = 0)
```

Arguments

- **x**: A vector or a two-dimensional array (matrix or data frame). If two-dimensional, the weights will be calculated separately on each column.
- **pearsonFallback**: Logical: if the median absolute deviation is zero, should standard deviation be substituted?
- **maxPOutliers**: Optional specification of the maximum proportion of outliers, i.e., data with weights equal to `outlierReferenceWeight` below.
- **outlierReferenceWeight**: A number between 0 and 1 specifying what is to be considered an outlier when calculating the proportion of outliers.
- **defaultWeight**: Value used for weights that correspond to a finite `x` but the weights themselves would not be finite, for example, when a column in `x` is constant.

Value

A vector or matrix of the same dimensions as the input `x` giving the weights.

Author(s)

Peter Langfelder

References

This function is based on Equation (3) in

That article also describes the Pearson fallback and maximum proportion of outliers in detail. For a full discussion of the biweight midcovariance and midcorrelation, see

bicovWeights
blockSize

Description

The function uses a rather primitive way to estimate available memory and use it to suggest a block size appropriate for the many block-by-block calculations in this package.

Usage

```r
blockSize(
  matrixSize,
  rectangularBlocks = TRUE,
  maxMemoryAllocation = NULL,
  overheadFactor = 3);
```

Arguments

- `matrixSize`: the relevant dimension (usually the number of columns) of the matrix that is to be operated on block-by-block.
- `rectangularBlocks`: logical indicating whether the blocks of data are rectangular (of size `blockSize` times `matrixSize`) or square (of size `blockSize` times `blockSize`).
- `maxMemoryAllocation`: maximum desired memory allocation, in bytes. Should not exceed 2GB or total installed RAM (whichever is greater) on 32-bit systems, while on 64-bit systems it should not exceed the total installed RAM. If not supplied, the available memory will be estimated internally.
- `overheadFactor`: overhead factor for the memory use by R. Recommended values are between 2 (for simple calculations) and 4 or more for complicated calculations where intermediate results (for which R must also allocate memory) take up a lot of space.

Details

Multiple functions within the WGCNA package use a divide-and-conquer (also known as block-by-block, or block-wise) approach to handling large data sets. This function is meant to assist in choosing a suitable block size, given the size of the data and the available memory.

If the entire expected result fits into the allowed memory (after taking into account the expected overhead), the returned block size will equal the input `matrixSize`.

See Also

bicor

Examples

```r
x = rnorm(100);
x[1] = 10;
plot(x, bicovWeights(x));
```
The internal estimation of available memory works by returning the size of largest successfully allocated block of memory. It is hoped that this will lead to reasonable results but some operating systems may actually allocate more than is available. It is therefore preferable that the user specifies the available memory by hand.

**Value**

A single integer giving the suggested block size, or `matrixSize` if the entire calculation is expected to fit into memory in one piece.

**Author(s)**

Peter Langfelder

**Examples**

```r
# Suitable blocks for handling 30,000 genes within 2GB (=2^31 bytes) of memory
tomSize(30000, rectangularBlocks = TRUE, maxMemoryAllocation = 2^31)
```

**Description**

Perform network construction and consensus module detection across several datasets.

**Usage**

```r
blockwiseConsensusModules(
  multiExpr,
  # Data checking options
  checkMissingData = TRUE,
  # Blocking options
  blocks = NULL,
  maxBlockSize = 5000,
  blockSizePenaltyPower = 5,
  randomSeed = 12345,
  # TOM precalculation arguments, if available
  individualTOMInfo = NULL,
  useIndivTOMSubset = NULL,
  # Network construction arguments: correlation options
  corType = "pearson",
  maxPOutliers = 1,
)```
quickCor = 0,
pearsonFallback = "individual",
cosineCorrelation = FALSE,

# Adjacency function options

power = 6,
networkType = "unsigned",
checkPower = TRUE,

# Topological overlap options

TOMType = "unsigned",
TOMDenom = "min",

# Save individual TOMs?

saveIndividualTOMs = TRUE,
individualTOMFileNames = "individualTOM-Set%sb.Block%b.RData",

# Consensus calculation options: network calibration

networkCalibration = c("single quantile", "full quantile", "none"),

# Simple quantile calibration options

calibrationQuantile = 0.95,
sampleForCalibration = TRUE, sampleForCalibrationFactor = 1000,
getNetworkCalibrationSamples = FALSE,

# Consensus definition

consensusQuantile = 0,
useMean = FALSE,
setWeights = NULL,

# Saving the consensus TOM

saveConsensusTOMs = FALSE,
consensusTOMFileNames = "consensusTOM-block.%b.RData",

# Internal handling of TOMs

useDiskCache = TRUE, chunkSize = NULL,
cacheBase = ".blockConsModsCache",
cacheDir = ".",

# Alternative consensus TOM input from a previous calculation

consensusTOMInfo = NULL,

# Basic tree cut options
# Basic tree cut options
depthSplit = 2,
detectCutHeight = 0.995, minModuleSize = 20,
checkMinModuleSize = TRUE,

# Advanced tree cut options
maxCoreScatter = NULL, minGap = NULL,
maxAbsCoreScatter = NULL, minAbsGap = NULL,
minSplitHeight = NULL, minAbsSplitHeight = NULL,
useBranchEigennodeDissim = FALSE,
minBranchEigennodeDissim = mergeCutHeight,
stabilityLabels = NULL,
minStabilityDissim = NULL,
pamStage = TRUE, pamRespectsDendro = TRUE,

# Gene reassignment and trimming from a module, and module "significance" criteria
reassignThresholdPS = 1e-4,
trimmingConsensusQuantile = consensusQuantile,
minCoreKME = 0.5, minCoreKMESize = minModuleSize/3,
minKMEtoStay = 0.2,

# Module eigengene calculation options
impute = TRUE,
trapErrors = FALSE,

# Module merging options
equalizeQuantilesForModuleMerging = FALSE,
quantileSummaryForModuleMerging = "mean",
mergeCutHeight = 0.15,
mergeConsensusQuantile = consensusQuantile,

# Output options
numericLabels = FALSE,

# General options
nThreads = 0,
verbose = 2, indent = 0, ...

Arguments

multiExpr expression data in the multi-set format (see checkSets). A vector of lists, one per set. Each set must contain a component data that contains the expression data, with rows corresponding to samples and columns to genes or probes.
checkMissingData
logical: should data be checked for excessive numbers of missing entries in genes and samples, and for genes with zero variance? See details.

blocks
optional specification of blocks in which hierarchical clustering and module detection should be performed. If given, must be a numeric vector with one entry per gene of multiExpr giving the number of the block to which the corresponding gene belongs.

maxBlockSize
integer giving maximum block size for module detection. Ignored if blocks above is non-NULL. Otherwise, if the number of genes in datExpr exceeds maxBlockSize, genes will be pre-clustered into blocks whose size should not exceed maxBlockSize.

blockSizePenaltyPower
number specifying how strongly blocks should be penalized for exceeding the maximum size. Set to a large number or Inf if not exceeding maximum block size is very important.

randomSeed
integer to be used as seed for the random number generator before the function starts. If a current seed exists, it is saved and restored upon exit. If NULL is given, the function will not save and restore the seed.

individualTOMInfo
Optional data for TOM matrices in individual data sets. This object is returned by the function blockwiseIndividualTOMs. If not given, appropriate topological overlaps will be calculated using the network construction options below.

useIndivTOMSubset
If individualTOMInfo is given, this argument allows to only select a subset of the individual set networks contained in individualTOMInfo. It should be a numeric vector giving the indices of the individual sets to be used. Note that this argument is NOT applied to multiExpr.

corType
character string specifying the correlation to be used. Allowed values are (unique abbreviations of) "pearson" and "bicor", corresponding to Pearson and bid-weight midcorrelation, respectively. Missing values are handled using the pairwise.complete.obs option.

maxPOutliers
only used for corType=="bicor". Specifies the maximum percentile of data that can be considered outliers on either side of the median separately. For each side of the median, if higher percentile than maxPOutliers is considered an outlier by the weight function based on 9*mad(x), the width of the weight function is increased such that the percentile of outliers on that side of the median equals maxPOutliers. Using maxPOutliers=1 will effectively disable all weight function broadening; using maxPOutliers=0 will give results that are quite similar (but not equal to) Pearson correlation.

quickCor
real number between 0 and 1 that controls the handling of missing data in the calculation of correlations. See details.

pearsonFallback
Specifies whether the bicor calculation, if used, should revert to Pearson when median absolute deviation (mad) is zero. Recognized values are (abbreviations of) "none", "individual", "all". If set to "none", zero mad will result in NA for the corresponding correlation. If set to "individual", Pearson calculation will be used only for columns that have zero mad. If set to "all", the presence of a single zero mad will cause the whole variable to be treated in Pearson correlation manner (as if the corresponding robust option was set to FALSE). Has no effect for Pearson correlation. See bicor.
blockwiseConsensusModules

cosineCorrelation
logical: should the cosine version of the correlation calculation be used? The cosine calculation differs from the standard one in that it does not subtract the mean.

power
soft-thresholding power for network construction.

networkType
network type. Allowed values are (unique abbreviations of) "unsigned", "signed", "signed hybrid". See adjacency.

checkPower
logical: should basic sanity check be performed on the supplied power? If you would like to experiment with unusual powers, set the argument to FALSE and proceed with caution.

TOMType
one of "none", "unsigned", "signed". If "none", adjacency will be used for clustering. If "unsigned", the standard TOM will be used (more generally, TOM function will receive the adjacency as input). If "signed", TOM will keep track of the sign of correlations between neighbors.

TOMDenom
a character string specifying the TOM variant to be used. Recognized values are "min" giving the standard TOM described in Zhang and Horvath (2005), and "mean" in which the min function in the denominator is replaced by mean. The "mean" may produce better results but at this time should be considered experimental.

saveIndividualTOMs
logical: should individual TOMs be saved to disk for later use?

individualTOMfilenames
character string giving the file names to save individual TOMs into. The following tags should be used to make the file names unique for each set and block: %s will be replaced by the set number; %N will be replaced by the set name (taken from names(multiExpr)) if it exists, otherwise by set number; %b will be replaced by the block number. If the file names turn out to be non-unique, an error will be generated.

networkCalibration
network calibration method. One of "single quantile", "full quantile", "none" (or a unique abbreviation of one of them).

calibrationQuantile
if networkCalibration is "single quantile", topological overlaps (or adjacencies if TOMs are not computed) will be scaled such that their calibrationQuantile quantiles will agree.

sampleForCalibration
if TRUE, calibration quantiles will be determined from a sample of network similarities. Note that using all data can double the memory footprint of the function and the function may fail.

sampleForCalibrationFactor
determines the number of samples for calibration: the number is 1/calibrationQuantile * sampleForCalibration. Should be set well above 1 to ensure accuracy of the sampled quantile.

getNetworkCalibrationSamples
logical: should samples used for TOM calibration be saved for future analysis? This option is only available when sampleForCalibration is TRUE.

consensusQuantile
quantile at which consensus is to be defined. See details.

useMean
logical: should the consensus be determined from a (possibly weighted) mean across the data sets rather than a quantile?
Optional vector (one component per input set) of weights to be used for weighted mean consensus. Only used when useMean above is TRUE.

Should the consensus topological overlap matrices for each block be saved and returned?

character string containing the file names containing the consensus topological overlaps. The tag %b will be replaced by the block number. If the resulting file names are non-unique (for example, because the user gives a file name without a %b tag), an error will be generated. These files are standard R data files and can be loaded using the load function.

should calculated network similarities in individual sets be temporarily saved to disk? Saving to disk is somewhat slower than keeping all data in memory, but for large blocks and/or many sets the memory footprint may be too big.

network similarities are saved in smaller chunks of size chunkSize.

character string containing the desired name for the cache files. The actual file names will consists of cacheBase and a suffix to make the file names unique.

character string containing the desired path for the cache files.

optional list summarizing consensus TOM, output of consensusTOM. It contains information about pre-calculated consensus TOM. Supplying this argument replaces TOM calculation, so none of the individual or consensus TOM calculation arguments are taken into account.

integer value between 0 and 4. Provides a simplified control over how sensitive module detection should be to module splitting, with 0 least and 4 most sensitive. See cutreeDynamic for more details.

Dendrogram cut height for module detection. See cutreeDynamic for more details.

minimum module size for module detection. See cutreeDynamic for more details.

logical: should sanity checks be performed on minModuleSize?

maximum scatter of the core for a branch to be a cluster, given as the fraction of cutHeight relative to the 5th percentile of joining heights. See cutreeDynamic for more details.

minimum cluster gap given as the fraction of the difference between cutHeight and the 5th percentile of joining heights. See cutreeDynamic for more details.

maximum scatter of the core for a branch to be a cluster given as absolute heights. If given, overrides maxCoreScatter. See cutreeDynamic for more details.

minimum cluster gap given as absolute height difference. If given, overrides minGap. See cutreeDynamic for more details.

Minimum split height given as the fraction of the difference between cutHeight and the 5th percentile of joining heights. Branches merging below this height will automatically be merged. Defaults to zero but is used only if minAbsSplitHeight below is NULL.
**blockwiseConsensusModules**

**minAbsSplitHeight**
Minimum split height given as an absolute height. Branches merging below this height will automatically be merged. If not given (default), will be determined from \texttt{minSplitHeight} above.

**useBranchEigennodeDissim**
Logical: should branch eigennode (eigengene) dissimilarity be considered when merging branches in Dynamic Tree Cut?

**minBranchEigennodeDissim**
Minimum consensus branch eigennode (eigengene) dissimilarity for branches to be considered separate. The branch eigennode dissimilarity in individual sets is simply 1-correlation of the eigennodes; the consensus is defined as quantile with probability \texttt{consensusQuantile}.

**stabilityLabels**
Optional matrix of cluster labels that are to be used for calculating branch dissimilarity based on split stability. The number of rows must equal the number of genes in \texttt{multiExpr}; the number of columns (clusterings) is arbitrary. See \texttt{branchSplitFromStabilityLabels} for details.

**minStabilityDissim**
Minimum stability dissimilarity criterion for two branches to be considered separate. Should be a number between 0 (essentially no dissimilarity required) and 1 (perfect dissimilarity or distinguishability based on \texttt{stabilityLabels}). See \texttt{branchSplitFromStabilityLabels} for details.

**pamStage**
logical. If \texttt{TRUE}, the second (PAM-like) stage of module detection will be performed. See \texttt{cutreeDynamic} for more details.

**pamRespectsDendro**
Logical, only used when \texttt{pamStage} is \texttt{TRUE}. If \texttt{TRUE}, the PAM stage will respect the dendrogram in the sense an object can be PAM-assigned only to clusters that lie below it on the branch that the object is merged into. See \texttt{cutreeDynamic} for more details.

**reassignThresholdPS**
per-set p-value ratio threshold for reassigning genes between modules. See Details.

**trimmingConsensusQuantile**
a number between 0 and 1 specifying the consensus quantile used for kME calculation that determines module trimming according to the arguments below.

**minCoreKME**
a number between 0 and 1. If a detected module does not have at least \texttt{minModuleKMESize} genes with eigengene connectivity at least \texttt{minCoreKME}, the module is disband (its genes are unlabeled and returned to the pool of genes waiting for module detection).

**minCoreKMESize**
see \texttt{minCoreKME} above.

**minKMEToStay**
genes whose eigengene connectivity to their module eigengene is lower than \texttt{minKMEToStay} are removed from the module.

**impute**
logical: should imputation be used for module eigengene calculation? See \texttt{moduleEigengenes} for more details.

**trapErrors**
logical: should errors in calculations be trapped?

**equalizeQuantilesForModuleMerging**
Logical: equalize quantiles of the module eigengene networks before module merging? If \texttt{TRUE}, the quantiles of the eigengene correlation matrices (interpreted as a single vectors of non-redundant components) will be equalized across the input data sets. Note that although this seems like a reasonable option, it should be considered experimental and not necessarily recommended.
quantileSummaryForModuleMerging
One of “mean” or “median”. If quantile equalization of the module eigengene networks is performed, the resulting “normal” quantiles will be given by this function of the corresponding quantiles across the input data sets.

mergeCutHeight  dendrogram cut height for module merging.
mergeConsensusQuantile  consensus quantile for module merging. See mergeCloseModules for details.
numericLabels  logical: should the returned modules be labeled by colors (FALSE), or by numbers (TRUE)?
nThreads  non-negative integer specifying the number of parallel threads to be used by certain parts of correlation calculations. This option only has an effect on systems on which a POSIX thread library is available (which currently includes Linux and Mac OSX, but excludes Windows). If zero, the number of online processors will be used if it can be determined dynamically, otherwise correlation calculations will use 2 threads.
verbose  integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
indent  indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.
...  Other arguments. At present these can include reproduceBranchEigennodeQuantileError that instructs the function to reproduce a bug in branch eigennode dissimilarity calculations for purposes if reproducing old results.

Details
The function starts by optionally filtering out samples that have too many missing entries and genes that have either too many missing entries or zero variance in at least one set. Genes that are filtered out are left unassigned by the module detection. Returned eigengenes will contain NA in entries corresponding to filtered-out samples.

If blocks is not given and the number of genes exceeds maxBlockSize, genes are pre-clustered into blocks using the function consensusProjectiveKMeans; otherwise all genes are treated in a single block.

For each block of genes, the network is constructed and (if requested) topological overlap is calculated in each set. To minimize memory usage, calculated topological overlaps are optionally saved to disk in chunks until they are needed again for the calculation of the consensus network topological overlap.

Before calculation of the consensus Topological Overlap, individual TOMs are optionally calibrated. Calibration methods include single quantile scaling and full quantile normalization.

Single quantile scaling raises individual TOM in sets 2, 3,... to a power such that the quantiles given by calibrationQuantile agree with the quantile in set 1. Since the high TOMs are usually the most important for module identification, the value of calibrationQuantile is close to (but not equal) 1. To speed up quantile calculation, the quantiles can be determined on a randomly-chosen component subset of the TOM matrices.

Full quantile normalization, implemented in normalize.quantiles, adjusts the TOM matrices such that all quantiles equal each other (and equal to the quantiles of the component-wise average of the individual TOM matrices).

Note that network calibration is performed separately in each block, i.e., the normalizing transformation may differ between blocks. This is necessary to avoid manipulating a full TOM in memory.
The consensus TOM is calculated as the component-wise consensus quantile quantile of the individual (set) TOMs; that is, for each gene pair (TOM entry), the consensus quantile across all input sets. Alternatively, one can also use (weighted) component-wise mean across all input data sets. If requested, the consensus topological overlaps are saved to disk for later use.

Genes are then clustered using average linkage hierarchical clustering and modules are identified in the resulting dendrogram by the Dynamic Hybrid tree cut. Found modules are trimmed of genes whose consensus module membership kME (that is, correlation with module eigengene) is less than \( \text{minKMEtoStay} \). Modules in which fewer than \( \text{minCoreKME} \) genes have consensus KME higher than \( \text{minCoreKME} \) are disbanded, i.e., their constituent genes are pronounced unassigned.

After all blocks have been processed, the function checks whether there are genes whose KME in the module they assigned is lower than KME to another module. If p-values of the higher correlations are smaller than those of the native module by the factor \( \text{reassignThresholdPS} \) (in every set), the gene is re-assigned to the closer module.

In the last step, modules whose eigengenes are highly correlated are merged. This is achieved by clustering module eigengenes using the dissimilarity given by one minus their correlation, cutting the dendrogram at the height \( \text{mergeCutHeight} \) and merging all modules on each branch. The process is iterated until no modules are merged. See \( \text{mergeCloseModules} \) for more details on module merging.

The argument quick specifies the precision of handling of missing data in the correlation calculations. Zero will cause all calculations to be executed precisely, which may be significantly slower than calculations without missing data. Progressively higher values will speed up the calculations but introduce progressively larger errors. Without missing data, all column means and variances can be pre-calculated before the covariances are calculated. When missing data are present, exact calculations require the column means and variances to be calculated for each covariance. The approximate calculation uses the pre-calculated mean and variance and simply ignores missing data in the covariance calculation. If the number of missing data is high, the pre-calculated means and variances may be very different from the actual ones, thus potentially introducing large errors. The quick value times the number of rows specifies the maximum difference in the number of missing entries for mean and variance calculations on the one hand and covariance on the other hand that will be tolerated before a recalculation is triggered. The hope is that if only a few missing data are treated approximately, the error introduced will be small but the potential speedup can be significant.

Value

A list with the following components:

- **colors** module assignment of all input genes. A vector containing either character strings with module colors (if input numericLabels was unset) or numeric module labels (if numericLabels was set to TRUE). The color "grey" and the numeric label 0 are reserved for unassigned genes.

- **unmergedColors** module colors or numeric labels before the module merging step.

- **multiMEs** module eigengenes corresponding to the modules returned in **colors**, in multi-set format. A vector of lists, one per set, containing eigengenes, proportion of variance explained and other information. See **multiSetMEs** for a detailed description.

- **goodSamples** a list, with one component per input set. Each component is a logical vector with one entry per sample from the corresponding set. The entry indicates whether the sample in the set passed basic quality control criteria.
goodGenes a logical vector with one entry per input gene indicating whether the gene passed basic quality control criteria in all sets.

dendrograms a list with one component for each block of genes. Each component is the hierarchical clustering dendrogram obtained by clustering the consensus gene dissimilarity in the corresponding block.

TOMFiles if saveConsensusTOMs==TRUE, a vector of character strings, one string per block, giving the file names of files (relative to current directory) in which blockwise topological overlaps were saved.

blockGenes a list with one component for each block of genes. Each component is a vector giving the indices (relative to the input multiExpr) of genes in the corresponding block.

blocks if input blocks was given, its copy; otherwise a vector of length equal number of genes giving the block label for each gene. Note that block labels are not necessarily sorted in the order in which the blocks were processed (since we do not require this for the input blocks). See blockOrder below.

blockOrder a vector giving the order in which blocks were processed and in which blockGenes above is returned. For example, blockOrder[1] contains the label of the first-processed block.

originCount if the input consensusQuantile==0, this vector will contain counts of how many times each set contributed the consensus gene similarity value. If the counts are highly unbalanced, the consensus may be biased.

networkCalibrationSamples if the input getNetworkCalibrationSamples is TRUE, this component is a list with one component per block. Each component is again a list with two components: sampleIndex contains indices of the distance structure in which TOM is stored that were sampled, and TOMSamples is a matrix whose rows correspond to TOM samples and columns to individual set. Hence, networkCalibrationSamples[[blockNo]]$TOM contains the TOM entry that corresponds to element networkCalibrationSamples[[blockNo]]$sampleIndex of the TOM distance structure in block blockNo and set setNo. (For details on the distance structure, see dist.)

Note

If the input datasets have large numbers of genes, consider carefully the maxBlockSize as it significantly affects the memory footprint (and whether the function will fail with a memory allocation error). From a theoretical point of view it is advantageous to use blocks as large as possible; on the other hand, using smaller blocks is substantially faster and often the only way to work with large numbers of genes. As a rough guide, it is unlikely a standard desktop computer with 4GB memory or less will be able to work with blocks larger than 7000 genes.

Author(s)

Peter Langfelder

References

See Also
goodSamplesGenesMS for basic quality control and filtering;
adjacency, TOMsimilarity for network construction;
hclust for hierarchical clustering;
cutreeDynamic for adaptive branch cutting in hierarchical clustering dendrograms;
mergeCloseModules for merging of close modules.

---

blockwiseIndividualTOMs

*Calculation of block-wise topological overlaps*

**Description**

Calculates topological overlaps in the given (expression) data. If the number of variables (columns) in the input data is too large, the data is first split using pre-clustering, then topological overlaps are calculated in each block.

**Usage**

```r
blockwiseIndividualTOMs(
  multiExpr,
  # Data checking options
  checkMissingData = TRUE,
  # Blocking options
  blocks = NULL,
  maxBlockSize = 5000,
  blockSizePenaltyPower = 5,
  randomSeed = 12345,
  # Network construction arguments: correlation options
  corType = "pearson",
  maxPOutliers = 1,
  quickCor = 0,
  pearsonFallback = "individual",
  cosineCorrelation = FALSE,
  # Adjacency function options
  power = 6,
  networkType = "unsigned",
  checkPower = TRUE,
  # Topological overlap options
)```

TOMType = "unsigned",
TOMDenom = "min",

# Save individual TOMs? If not, they will be returned in the session.
saveTOMs = TRUE,
individualTOMFileNames = "individualTOM-Set$s-Block$b.RData",

# General options
nThreads = Ø,
verbose = 2, indent = Ø)

Arguments

multiExpr expression data in the multi-set format (see checkSets). A vector of lists, one per set. Each set must contain a component data that contains the expression data, with rows corresponding to samples and columns to genes or probes.

checkMissingData logical: should data be checked for excessive numbers of missing entries in genes and samples, and for genes with zero variance? See details.

blocks optional specification of blocks in which hierarchical clustering and module detection should be performed. If given, must be a numeric vector with one entry per gene of multiExpr giving the number of the block to which the corresponding gene belongs.

maxBlockSize integer giving maximum block size for module detection. Ignored if blocks above is non-NULL. Otherwise, if the number of genes in datExpr exceeds maxBlockSize, genes will be pre-clustered into blocks whose size should not exceed maxBlockSize.

blockSizePenaltyPower number specifying how strongly blocks should be penalized for exceeding the maximum size. Set to a large number or Inf if not exceeding maximum block size is very important.

randomSeed integer to be used as seed for the random number generator before the function starts. If a current seed exists, it is saved and restored upon exit. If NULL is given, the function will not save and restore the seed.

corType character string specifying the correlation to be used. Allowed values are (unique abbreviations of) "pearson" and "bicor", corresponding to Pearson and bidirectional midcorrelation, respectively. Missing values are handled using the parwise.complete.obs option.

maxPOutliers only used for corType=="bicor". Specifies the maximum percentile of data that can be considered outliers on either side of the median separately. For each side of the median, if higher percentile than maxPOutliers is considered an outlier by the weight function based on 9*mad(x), the width of the weight function is increased such that the percentile of outliers on that side of the median equals maxPOutliers. Using maxPOutliers=1 will effectively disable all weight function broadening; using maxPOutliers=Ø will give results that are quite similar (but not equal to) Pearson correlation.

quickCor real number between 0 and 1 that controls the handling of missing data in the calculation of correlations. See details.
blockwiseIndividualTOMs

**pearsonFallback**
Specifies whether the bicor calculation, if used, should revert to Pearson when median absolute deviation (mad) is zero. Recognized values are (abbreviations of) "none", "individual", "all". If set to "none", zero mad will result in NA for the corresponding correlation. If set to "individual", Pearson calculation will be used only for columns that have zero mad. If set to "all", the presence of a single zero mad will cause the whole variable to be treated in Pearson correlation manner (as if the corresponding robust option was set to FALSE). Has no effect for Pearson correlation. See `bicor`.

**cosineCorrelation**
logical: should the cosine version of the correlation calculation be used? The cosine calculation differs from the standard one in that it does not subtract the mean.

**power**
soft-thresholding power for network construction.

**networkType**
network type. Allowed values are (unique abbreviations of) "unsigned", "signed", "signed hybrid". See `adjacency`.

**checkPower**
logical: should basic sanity check be performed on the supplied power? If you would like to experiment with unusual powers, set the argument to FALSE and proceed with caution.

**TOMType**
one of "none", "unsigned", "signed". If "none", adjacency will be used for clustering. If "unsigned", the standard TOM will be used (more generally, TOM function will receive the adjacency as input). If "signed", TOM will keep track of the sign of correlations between neighbors. Note that the "unsigned" vs. "signed" distinction is only relevant when `networkType` is "unsigned". When `networkType` is "signed" or "signed hybrid", there is no difference between TOMType="signed" and TOMType="unsigned".

**TOMDenom**
a character string specifying the TOM variant to be used. Recognized values are "min" giving the standard TOM described in Zhang and Horvath (2005), and "mean" in which the min function in the denominator is replaced by mean. The "mean" may produce better results in certain special situations but at this time should be considered experimental.

**saveTOMs**
logical: should calculated TOMs be saved to disk (TRUE) or returned in the return value (FALSE)? Returning calculated TOMs via the return value ay be more convenient but not always feasible if the matrices are too big to fit all in memory at the same time.

**individualTOMFileNames**
character string giving the file names to save individual TOMs into. The following tags should be used to make the file names unique for each set and block: %s will be replaced by the set number; %N will be replaced by the set name (taken from names(multiExpr)) if it exists, otherwise by set number; %b will be replaced by the block number. If the file names turn out to be non-unique, an error will be generated.

**nThreads**
non-negative integer specifying the number of parallel threads to be used by certain parts of correlation calculations. This option only has an effect on systems on which a POSIX thread library is available (which currently includes Linux and Mac OSX, but excludes Windows). If zero, the number of online processors will be used if it can be determined dynamically, otherwise correlation calculations will use 2 threads.

**verbose**
integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

The function starts by optionally filtering out samples that have too many missing entries and genes that have either too many missing entries or zero variance in at least one set. Genes that are filtered out are excluded from the TOM calculations.

If `blocks` is not given and the number of genes exceeds `maxBlockSize`, genes are pre-clustered into blocks using the function `consensusProjectiveKMeans`; otherwise all genes are treated in a single block.

For each block of genes, the network is constructed and (if requested) topological overlap is calculated in each set. The topological overlaps can be saved to disk as RData files, or returned directly within the return value (see below). Note that the matrices can be big and returning them within the return value can quickly exhaust the system’s memory. In particular, if the block-wise calculation is necessary, it is nearly certain that returning all matrices via the return value will be impossible.

Value

A list with the following components:

- `actualTOMFileNames` Only returned if input `saveTOMs` is TRUE. A matrix of character strings giving the file names in which each block TOM is saved. Rows correspond to data sets and columns to blocks.

- `TOMSimilarities` Only returned if input `saveTOMs` is FALSE. A list in which each component corresponds to one block. Each component is a matrix of dimensions (N times (number of sets)), where N is the length of a distance structure corresponding to the block. That is, if the block contains n genes, N=n*(n-1)/2. Each column of the matrix contains the topological overlap of variables in the corresponding set (and the corresponding block), arranged as a distance structure. Do note however that the topological overlap is a similarity (not a distance).

- `blocks` if input `blocks` was given, its copy; otherwise a vector of length equal number of genes giving the block label for each gene. Note that block labels are not necessarily sorted in the order in which the blocks were processed (since we do not require this for the input `blocks`). See `blockOrder` below.

- `blockGenes` a list with one component for each block of genes. Each component is a vector giving the indices (relative to the input `multiExpr`) of genes in the corresponding block.

- `goodSamplesAndGenes` if input `checkMissingData` is TRUE, the output of the function `goodSamplesGenesMS`. A list with components `goodGenes` (logical vector indicating which genes passed the missing data filters), `goodSamples` (a list of logical vectors indicating which samples passed the missing data filters in each set), and `allOK` (a logical indicating whether all genes and all samples passed the filters). See `goodSamplesGenesMS` for more details. If `checkMissingData` is FALSE, `goodSamplesAndGenes` contains a list of the same type but indicating that all genes and all samples passed the missing data filters.

The following components are present mostly to streamline the interaction of this function with `blockwiseConsensusModules`.
blockwiseModules

nGGenes: Number of genes that passed missing data filters (if input checkMissingData is TRUE), or the number of all genes (if checkMissingData is FALSE).
gBlocks: the vector blocks (above), restricted to good genes only.
nThreads: number of threads used to calculate correlation and TOM matrices.
saveTOMs: logical: were calculated matrices saved in files (TRUE) or returned in the return value (FALSE)?

intNetworkType, intCorType: integer codes for network and correlation type.
nSets: number of sets in input data.
setNames: the names attribute of input multiExpr.

Author(s)

Peter Langfelder

References

For a general discussion of the weighted network formalism, see

The blockwise approach is briefly described in the article describing this package,

See Also

blockwiseConsensusModules

Description

This function performs automatic network construction and module detection on large expression datasets in a block-wise manner.

Usage

blockwiseModules(
  datExpr,  
  checkMissingData = TRUE,
  # Options for splitting data into blocks
)

Automatic network construction and module detection

This function performs automatic network construction and module detection on large expression datasets in a block-wise manner.
blocks = NULL,
maxBlockSize = 5000,
blockSizePenaltyPower = 5,
randomSeed = 12345,

# load TOM from previously saved file?
loadTOM = FALSE,

# Network construction arguments: correlation options
corType = "pearson",
maxPOutliers = 1,
quickCor = 0,
pearsonFallback = "individual",
cosineCorrelation = FALSE,

# Adjacency function options
power = 6,
networkType = "unsigned",

# Topological overlap options
TOMType = "signed",
TOMDenom = "min",

# Saving or returning TOM
getTOMs = NULL,
saveTOMs = FALSE,
saveTOMFileBase = "blockwiseTOM",

# Basic tree cut options
depthSplit = 2,
detectCutHeight = 0.995,
minModuleSize = min(20, ncol(datExpr)/2 ),

# Advanced tree cut options
maxCoreScatter = NULL, minGap = NULL,
maxAbsCoreScatter = NULL, minAbsGap = NULL,
minSplitHeight = NULL, minAbsSplitHeight = NULL,

useBranchEigenNodeDissim = FALSE,
minBranchEigenNodeDissim = mergeCutHeight,

stabilityLabels = NULL,
minStabilityDissim = NULL,
pamStage = TRUE, pamRespectsDendro = TRUE,

# Gene reassignment, module trimming, and module "significance" criteria
reassignThreshold = 1e-6,
minCoreKME = 0.5,
minCoreKMESize = minModuleSize/3,
minKMEtoStay = 0.3,

# Module merging options
mergeCutHeight = 0.15,
impute = TRUE,
trapErrors = FALSE,

# Output options
numericLabels = FALSE,

# Options controlling behaviour
nThreads = 0,
verbose = 0, indent = 0,
...

Arguments

datExpr expression data. A data frame in which columns are genes and rows are samples. NAs are allowed, but not too many.

checkMissingData logical: should data be checked for excessive numbers of missing entries in genes and samples, and for genes with zero variance? See details.

blocks optional specification of blocks in which hierarchical clustering and module detection should be performed. If given, must be a numeric vector with one entry per column (gene) of exprData giving the number of the block to which the corresponding gene belongs.

maxBlockSize integer giving maximum block size for module detection. Ignored if blocks above is non-NULL. Otherwise, if the number of genes in datExpr exceeds maxBlockSize, genes will be pre-clustered into blocks whose size should not exceed maxBlockSize.

BlockSizePenaltyPower number specifying how strongly blocks should be penalized for exceeding the maximum size. Set to a large number or Inf if not exceeding maximum block size is very important.

randomSeed integer to be used as seed for the random number generator before the function starts. If a current seed exists, it is saved and restored upon exit. If NULL is given, the function will not save and restore the seed.

loadTOM logical: should Topological Overlap Matrices be loaded from previously saved files (TRUE) or calculated (FALSE)? It may be useful to load previously saved TOM matrices if these have been calculated previously, since TOM calculation
is often the most computationally expensive part of network construction and module identification. See \texttt{saveTOMs} and \texttt{saveTOMFileBase} below for when and how TOM files are saved, and what the file names are. If \texttt{loadTOM} is TRUE but the files cannot be found, or do not contain the correct TOM data, TOM will be recalculated.

\textbf{corType} \hspace{1cm} character string specifying the correlation to be used. Allowed values are (unique abbreviations of) "pearson" and "bicor", corresponding to Pearson and bid-weight midcorrelation, respectively. Missing values are handled using the \texttt{pairwise.complete.obs} option.

\textbf{maxPOutliers} \hspace{1cm} only used for \texttt{corType=="bicor"}. Specifies the maximum percentile of data that can be considered outliers on either side of the median separately. For each side of the median, if higher percentile than \texttt{maxPOutliers} is considered an outlier by the weight function based on $9*\text{mad}(x)$, the width of the weight function is increased such that the percentile of outliers on that side of the median equals \texttt{maxPOutliers}. Using \texttt{maxPOutliers=1} will effectively disable all weight function broadening; using \texttt{maxPOutliers=0} will give results that are quite similar (but not equal to) Pearson correlation.

\textbf{quickCor} \hspace{1cm} real number between 0 and 1 that controls the handling of missing data in the calculation of correlations. See details.

\textbf{pearsonFallback} \hspace{1cm} Specifies whether the bicor calculation, if used, should revert to Pearson when median absolute deviation (mad) is zero. Recognized values are (abbreviations of) "none", "individual", "all". If set to "none", zero mad will result in NA for the corresponding correlation. If set to "individual", Pearson calculation will be used only for columns that have zero mad. If set to "all", the presence of a single zero mad will cause the whole variable to be treated in Pearson correlation manner (as if the corresponding robust option was set to \texttt{FALSE}). Has no effect for Pearson correlation. See \texttt{bicor}.

\textbf{cosineCorrelation} \hspace{1cm} logical: should the cosine version of the correlation calculation be used? The cosine calculation differs from the standard one in that it does not subtract the mean.

\textbf{power} \hspace{1cm} soft-thresholding power for network construction.

\textbf{networkType} \hspace{1cm} network type. Allowed values are (unique abbreviations of) "unsigned", "signed", "signed hybrid". See \texttt{adjacency}.

\textbf{TOMType} \hspace{1cm} one of "none", "unsigned", "signed". If "none", adjacency will be used for clustering. If "unsigned", the standard TOM will be used (more generally, TOM function will receive the adjacency as input). If "signed", TOM will keep track of the sign of correlations between neighbors.

\textbf{TOMDenom} \hspace{1cm} a character string specifying the TOM variant to be used. Recognized values are "min" giving the standard TOM described in Zhang and Horvath (2005), and "mean" in which the \texttt{min} function in the denominator is replaced by \texttt{mean}. The "mean" may produce better results but at this time should be considered experimental.

\textbf{getTOMs} \hspace{1cm} deprecated, please use \texttt{saveTOMs} below.

\textbf{saveTOMs} \hspace{1cm} logical: should the consensus topological overlap matrices for each block be saved and returned?

\textbf{saveTOMFileBase} \hspace{1cm} character string containing the file name base for files containing the consensus topological overlaps. The full file names have "block.1.RData", "block.2.RData"
blockwiseModules

etc. appended. These files are standard R data files and can be loaded using the load function.

deepSplit integer value between 0 and 4. Provides a simplified control over how sensitive module detection should be to module splitting, with 0 least and 4 most sensitive. See cuttreeDynamic for more details.

detectCutHeight dendrogram cut height for module detection. See cuttreeDynamic for more details.

minModuleSize minimum module size for module detection. See cuttreeDynamic for more details.

maxCoreScatter maximum scatter of the core for a branch to be a cluster, given as the fraction of cutHeight relative to the 5th percentile of joining heights. See cuttreeDynamic for more details.

minGap minimum cluster gap given as the fraction of the difference between cutHeight and the 5th percentile of joining heights. See cuttreeDynamic for more details.

maxAbsCoreScatter maximum scatter of the core for a branch to be a cluster given as absolute heights. If given, overrides maxCoreScatter. See cuttreeDynamic for more details.

minAbsGap minimum cluster gap given as absolute height difference. If given, overrides minGap. See cuttreeDynamic for more details.

minSplitHeight Minimum split height given as the fraction of the difference between cutHeight and the 5th percentile of joining heights. Branches merging below this height will automatically be merged. Defaults to zero but is used only if minAbsSplitHeight below is NULL.

minAbsSplitHeight Minimum split height given as an absolute height. Branches merging below this height will automatically be merged. If not given (default), will be determined from minSplitHeight above.

useBranchEigenNodeDissim Logical: should branch eigennode (eigengene) dissimilarity be considered when merging branches in Dynamic Tree Cut?

minBranchEigenNodeDissim Minimum consensus branch eigennode (eigengene) dissimilarity for branches to be considered separate. The branch eigennode dissimilarity in individual sets is simply 1-correlation of the eigennodes; the consensus is defined as quantile with probability consensusQuantile.

stabilityLabels Optional matrix of cluster labels that are to be used for calculating branch dissimilarity based on split stability. The number of rows must equal the number of genes in multiExpr; the number of columns (clusterings) is arbitrary. See branchSplitFromStabilityLabels for details.

minStabilityDissim Minimum stability dissimilarity criterion for two branches to be considered separate. Should be a number between 0 (essentially no dissimilarity required) and 1 (perfect dissimilarity or distinguishability based on stabilityLabels). See branchSplitFromStabilityLabels for details.

pamStage logical. If TRUE, the second (PAM-like) stage of module detection will be performed. See cuttreeDynamic for more details.
Logical, only used when pamStage is TRUE. If TRUE, the PAM stage will respect the dendrogram in the sense an object can be PAM-assigned only to clusters that lie below it on the branch that the object is merged into. See cutreeDynamic for more details.

minCoreKME a number between 0 and 1. If a detected module does not have at least minModuleKMEsize genes with eigengene connectivity at least minCoreKME, the module is disbanded (its genes are unlabeled and returned to the pool of genes waiting for module detection).

minCoreKMEsize see minCoreKME above.

minKMEToStay genes whose eigengene connectivity to their module eigengene is lower than minKMEToStay are removed from the module.

reassignThreshold p-value ratio threshold for reassigning genes between modules. See Details.

mergeCutHeight dendrogram cut height for module merging.

impute logical: should imputation be used for module eigengene calculation? See moduleEigengenes for more details.

trapErrors logical: should errors in calculations be trapped?

numericLabels logical: should the returned modules be labeled by colors (FALSE), or by numbers (TRUE)?

nThreads non-negative integer specifying the number of parallel threads to be used by certain parts of correlation calculations. This option only has an effect on systems on which a POSIX thread library is available (which currently includes Linux and Mac OSX, but excludes Windows). If zero, the number of online processors will be used if it can be determined dynamically, otherwise correlation calculations will use 2 threads.

verbose integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.

indent indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

... Other arguments.

Details

Before module detection starts, genes and samples are optionally checked for the presence of NAs. Genes and/or samples that have too many NAs are flagged as bad and removed from the analysis; bad genes will be automatically labeled as unassigned, while the returned eigengenes will have NA entries for all bad samples.

If blocks is not given and the number of genes exceeds maxBlockSize, genes are pre-clustered into blocks using the function projectiveKMeans; otherwise all genes are treated in a single block.

For each block of genes, the network is constructed and (if requested) topological overlap is calculated. If requested, the topological overlaps are returned as part of the return value list. Genes are then clustered using average linkage hierarchical clustering and modules are identified in the resulting dendrogram by the Dynamic Hybrid tree cut. Found modules are trimmed of genes whose correlation with module eigengene (KME) is less than minKMEToStay. Modules in which fewer than minCoreKMEsize genes have KME higher than minCoreKME are disbanded, i.e., their constituent genes are pronounced unassigned.

After all blocks have been processed, the function checks whether there are genes whose KME in the module they assigned is lower than KME to another module. If p-values of the higher correlations
are smaller than those of the native module by the factor `reassignThresholdPS`, the gene is re-assigned to the closer module.

In the last step, modules whose eigengenes are highly correlated are merged. This is achieved by clustering module eigengenes using the dissimilarity given by one minus their correlation, cutting the dendrogram at the height `mergeCutHeight` and merging all modules on each branch. The process is iterated until no modules are merged. See `mergeCloseModules` for more details on module merging.

The argument `quick` specifies the precision of handling of missing data in the correlation calculations. Zero will cause all calculations to be executed precisely, which may be significantly slower than calculations without missing data. Progressively higher values will speed up the calculations but introduce progressively larger errors. Without missing data, all column means and variances can be pre-calculated before the covariances are calculated. When missing data are present, exact calculations require the column means and variances to be calculated for each covariance. The approximate calculation uses the pre-calculated mean and variance and simply ignores missing data in the covariance calculation. If the number of missing data is high, the pre-calculated means and variances may be very different from the actual ones, thus potentially introducing large errors. The `quick` value times the number of rows specifies the maximum difference in the number of missing entries for mean and variance calculations on the one hand and covariance on the other hand that will be tolerated before a recalculation is triggered. The hope is that if only a few missing data are treated approximately, the error introduced will be small but the potential speedup can be significant.

### Value

A list with the following components:

- **colors**: a vector of color or numeric module labels for all genes.
- **unmergedColors**: a vector of color or numeric module labels for all genes before module merging.
- **MEs**: a data frame containing module eigengenes of the found modules (given by `colors`).
- **goodSamples**: numeric vector giving indices of good samples, that is samples that do not have too many missing entries.
- **goodGenes**: numeric vector giving indices of good genes, that is genes that do not have too many missing entries.
- **dendrograms**: a list whose components contain hierarchical clustering dendrograms of genes in each block.
- **TOMFiles**: if `saveTOMs==TRUE`, a vector of character strings, one string per block, giving the file names of files (relative to current directory) in which blockwise topological overlaps were saved.
- **blockGenes**: a list whose components give the indices of genes in each block.
- **blocks**: if input `blocks` was given, its copy; otherwise a vector of length equal number of genes giving the block label for each gene. Note that block labels are not necessarily sorted in the order in which the blocks were processed (since we do not require this for the input `blocks`). See `blockOrder` below.
- **blockOrder**: a vector giving the order in which blocks were processed and in which `blockGenes` above is returned. For example, `blockOrder[1]` contains the label of the first-processed block.
- **MEsOK**: logical indicating whether the module eigengenes were calculated without errors.
**BloodLists**

**Note**

If the input dataset has a large number of genes, consider carefully the `maxBlockSize` as it significantly affects the memory footprint (and whether the function will fail with a memory allocation error). From a theoretical point of view it is advantageous to use blocks as large as possible; on the other hand, using smaller blocks is substantially faster and often the only way to work with large numbers of genes. As a rough guide, it is unlikely a standard desktop computer with 4GB memory or less will be able to work with blocks larger than 8000 genes.

**Author(s)**

Peter Langfelder

**References**


**See Also**

`goodSamplesGenes` for basic quality control and filtering;
`adjacency`, `TOMsimilarity` for network construction;
`hclust` for hierarchical clustering;
`cutreeDynamic` for adaptive branch cutting in hierarchical clustering dendrograms;
`mergeCloseModules` for merging of close modules.

---

**BloodLists**

| BloodCellTypes with Corresponding Gene Markers |

**Description**

This matrix gives a predefined set of marker genes for many blood cell types, as reported in several previously-published studies. It is used with `userListEnrichment` to search user-defined gene lists for enrichment.

**Usage**

`data(BloodLists)`

**Format**

A 2048 x 2 matrix of characters containing Gene / Category pairs. The first column (Gene) lists genes corresponding to a given category (second column). Each Category entry is of the form `<Blood cell type>__<reference>`, where the references can be found at `userListEnrichment`. Note that the matrix is sorted first by Category and then by Gene, such that all genes related to the same category are listed sequentially.

**Source**

For references used in this variable, please see `userListEnrichment`
**blueWhiteRed**

**Examples**

```r
data(BloodLists)
head(BloodLists)
```

---

**blueWhiteRed**  
*Blue-white-red color sequence*

**Description**

Generate a blue-white-red color sequence of a given length.

**Usage**

```r
blueWhiteRed(n, gamma = 1, endSaturation = 1)
```

**Arguments**

- `n` : number of colors to be returned.
- `gamma` : color change power.
- `endSaturation` : a number between 0 and 1 giving the saturation of the colors that will represent the ends of the scale. Lower numbers mean less saturation (lighter colors).

**Details**

The function returns a color vector that starts with blue, gradually turns into white and then to red. The power `gamma` can be used to control the behaviour of the quarter- and three quarter-values (between blue and white, and white and red, respectively). Higher powers will make the mid-colors more white, while lower powers will make the colors more saturated, respectively.

**Value**

A vector of colors of length `n`.

**Author(s)**

Peter Langfelder

**See Also**

`numbers2colors` for a function that produces a color representation for continuous numbers.

**Examples**

```r
par(mfrow = c(3, 1))
displayColors(blueWhiteRed(50)); title("gamma = 1")
displayColors(blueWhiteRed(50, 3)); title("gamma = 3")
displayColors(blueWhiteRed(50, 0.5)); title("gamma = 0.5")
```
BrainRegionMarkers

BrainLists

**Brain-Related Categories with Corresponding Gene Markers**

**Description**

This matrix gives a predefined set of marker genes for many brain-related categories (i.e., cell type, organelle, changes with disease, etc.), as reported in several previously-published studies. It is used with userListEnrichment to search user-defined gene lists for enrichment.

**Usage**

```r
data(BrainLists)
```

**Format**

A 48319 x 2 matrix of characters containing Gene / Category pairs. The first column (Gene) lists genes corresponding to a given category (second column). Each Category entry is of the form `<Brain descriptor>__<reference>`, where the references can be found at userListEnrichment. Note that the matrix is sorted first by Category and then by Gene, such that all genes related to the same category are listed sequentially.

**Source**

For references used in this variable, please see userListEnrichment

**Examples**

```r
data(BrainLists)
head(BrainLists)
```

BrainRegionMarkers

**Gene Markers for Regions of the Human Brain**

**Description**


**Usage**

```r
data(BrainRegionMarkers)
```

**Format**

A 28477 x 2 matrix of characters containing Gene / Category pairs. The first column (Gene) lists genes corresponding to a given category (second column). Each Category entry is of the form `<Brain Region>_<Marker Type>__HBA`. Note that the matrix is sorted first by Category and then by Gene, such that all genes related to the same category are listed sequentially.
**branchEigengeneDissim**

**Source**

For references used in this variable, or other information, please see `userListEnrichment`.

**Examples**

```r
data(BrainRegionMarkers)
head(BrainRegionMarkers)
```

**branchEigengeneDissim**  
*Branch dissimilarity based on eigennodes (eigengenes).*

**Description**

Calculation of branch dissimilarity based on eigennodes (eigengenes) in single set and multi-data situations. This function is used as a plugin for the `dynamicTreeCut` package and the user should not call this function directly. This function is experimental and subject to change.

**Usage**

```r
branchEigengeneDissim(
  expr,
  branch1, branch2,
  corFnc = cor, corOptions = list(use = "p"),
  signed = TRUE, ...)
```

```r
mtd.branchEigengeneDissim(
  multiExpr,
  branch1, branch2,
  corFnc = cor, corOptions = list(use = 'p'),
  consensusQuantile = 0,
  signed = TRUE, reproduceQuantileError = FALSE, ...)
```

**Arguments**

- `expr`  
  Expression data.

- `multiExpr`  
  Expression data in multi-set format.

- `branch1`  
  Branch 1.

- `branch2`  
  Branch 2.

- `corFnc`  
  Correlation function.

- `corOptions`  
  Other arguments to the correlation function.

- `consensusQuantile`  
  Consensus quantile.

- `signed`  
  Should the network be considered signed?

- `reproduceQuantileError`  
  Logical: should an error in the calculation from previous versions, which caused the true consensus quantile to be `1-consensusQuantile` rather than `consensusQuantile`, be reproduced? Use this only to reproduce old calculations.

- `...`  
  Other arguments for compatibility; currently unused.
Value

A single number or a list containing details of the calculation.

Author(s)

Peter Langfelder

---

branchSplit  

*Branch split.*

Description

Calculation of branch split based on expression data. This function is used as a plugin for the dynamicTreeCut package and the user should not call this function directly.

Usage

```r
branchsplitH(exprL, branchQL, branchRL, discardprop \[ PNPUL \], mincentralprop \[ PNWUL \], nconsideredpcs \[ SL \], signed = falseL, getdetails = trueL, NNN)
```

Arguments

- `expr`: Expression data.
- `branchQ`: Branch 1.
- `branchR`: Branch 2.
- `discardprop`: Proportion of data to be discarded as outliers.
- `mincentralprop`: Minimum central proportion
- `nconsideredpcs`: Number of principal components to consider.
- `signed`: Should the network be considered signed?
- `getdetails`: Should details of the calculation be returned?
- `...`: Other arguments. Present for compatibility; currently unused.

Value

A single number or a list containing details of the calculation.

Author(s)

Peter Langfelder
branchSplit.dissim

Branch split based on dissimilarity.

Description
Calculation of branch split based on a dissimilarity matrix. This function is used as a plugin for the dynamicTreeCut package and the user should not call this function directly. This function is experimental and subject to change.

Usage
branchSplit.dissim(
  dissimMat,
  branch1, branch2,
  upperP,
  minNumberInSplit = 5,
  getDetails = FALSE,...)

Arguments
dissimMat  Dissimilarity matrix.
branch1    Branch 1.
branch2    Branch 2.
upperP     Percentile of (closest) objects to be considered.
minNumberInSplit Minimum number of objects to be considered.
getDetails Should details of the calculation be returned?
...        Other arguments for compatibility; currently unused.

Value
A single number or a list containing details of the calculation.

Author(s)
Peter Langfelder

branchSplitFromStabilityLabels

Branch split (dissimilarity) statistic derived from labels determined from a stability study

Description
This function evaluates how different two branches are based on a series of cluster labels that are usually obtained in a stability study but can in principle be arbitrary. The idea is to quantify how well membership on the two tested branches can be predicted from clusters in the given stability labels.
Usage

```
branchSplitFromStabilityLabels(
  branch1, branch2,
  stabilityLabels,
  ignoreLabels = 0,
  ...)
```

Arguments

- `branch1`: A vector of indices giving members of branch 1.
- `branch2`: A vector of indices giving members of branch 1.
- `stabilityLabels`: A matrix of cluster labels. Each column corresponds to one clustering and each row to one object (whose indices `branch1` and `branch2` refer to).
- `ignoreLabels`: Label or labels that do not constitute proper clusters in `stabilityLabels`, for example because they label unassigned objects.
- `...`: Ignored.

Details

The idea is to measure how well clusters in `stabilityLabels` can distinguish the two given branches. For example, if a cluster C intersects with branch 1 but not branch 2, it can distinguish branches 1 and 2 perfectly. On the other hand, if there is a cluster C that contains both branch 1 and branch 2, the two branches are indistinguishable (based on the test clustering).

Formally, for each cluster C in each clustering in `stabilityLabels`, its contribution to the branch similarity is $\min(r_1, r_2)$, where $r_1 = \frac{|\text{intersect}(C, \text{branch1})|}{|\text{branch1}|}$ and $r_2 = \frac{|\text{intersect}(C, \text{branch2})|}{|\text{branch2}|}$. The statistics for clusters in each clustering are added; the sums are then averaged across the clusterings. Since the result is a similarity statistic, the final dissimilarity is defined as $1$-similarity. The dissimilarity ranges between 0 (branch1 and branch2 are indistinguishable) and 1 (branch1 and branch2 are perfectly distinguishable).

This is a very simple statistic that does not attempt to correct for the similarity that would be expected by chance.

Value

Branch dissimilarity (a single number between 0 and 1).

Author(s)

Peter Langfelder

See Also

This function is utilized in `blockwiseModules` and `blockwiseConsensusModules`.
**checkAdjMat**

*Check adjacency matrix*

**Description**

Checks a given matrix for properties that an adjacency matrix must satisfy.

**Usage**

```r
checkAdjMat(adjMat, min = 0, max = 1)
checkSimilarity(similarity, min = -1, max = 1)
```

**Arguments**

- `adjMat`: matrix to be checked
- `similarity`: matrix to be checked
- `min`: minimum allowed value for entries of the input
- `max`: maximum allowed value for entries of the input

**Details**

The function checks whether the given matrix really is a 2-dimensional numeric matrix, whether it is square, symmetric, and all finite entries are between `min` and `max`. If any of the conditions is not met, the function issues an error.

**Value**

None. The function returns normally if all conditions are met.

**Author(s)**

Peter Langfelder

**See Also**

- `adjacency`

---

**checkSets**

*Check structure and retrieve sizes of a group of datasets.*

**Description**

Checks whether given sets have the correct format and retrieves dimensions.

**Usage**

```r
checkSets(data, checkStructure = FALSE, useSets = NULL)
```
chooseOneHubInEachModule

**Arguments**

- `data`: A vector of lists; in each list there must be a component named `data` whose content is a matrix or dataframe or array of dimension 2.
- `checkStructure`: If `FALSE`, incorrect structure of `data` will trigger an error. If `TRUE`, an appropriate flag (see output) will be set to indicate whether `data` has correct structure.
- `useSets`: Optional specification of entries of the vector `data` that are to be checked. Defaults to all components. This may be useful when `data` only contains information for some of the sets.

**Details**

For multiset calculations, many quantities (such as expression data, traits, module eigengenes etc) are presented by a common structure, a vector of lists (one list for each set) where each list has a component `data` that contains the actual (expression, trait, eigengene) data for the corresponding set in the form of a dataframe. This function checks whether `data` conforms to this convention and retrieves some basic dimension information (see output).

**Value**

A list with components

- `nSets`: Number of sets (length of the vector `data`).
- `nGenes`: Number of columns in the `data` components in the lists. This number must be the same for all sets.
- `nSamples`: A vector of length `nSets` giving the number of rows in the `data` components.
- `structureOK`: Only set if the argument `checkStructure` equals `TRUE`. The value is `TRUE` if the parameter `data` passes a few tests of its structure, and `FALSE` otherwise. The tests are not exhaustive and are meant to catch obvious user errors rather than be bulletproof.

**Author(s)**

Peter Langfelder, <Peter.Langfelder@gmail.com>

---

**Description**

`chooseOneHubInEachModule` returns one gene in each module with high connectivity, given a number of randomly selected genes to test.
Usage

chooseOneHubInEachModule(
  datExpr,
  colorh,
  numGenes = 100,
  omitColors = "grey",
  power = 2,
  type = "signed",
  ...
)

Arguments

datExpr  Gene expression data with rows as samples and columns as genes.
colorh   The module assignments (color vectors) corresponding to the rows in datExpr.
numGenes Th number of random genes to select per module. Higher number of genes increases the accuracy of hub selection but slows down the function.
omitColors All colors in this character vector (default is "grey") are ignored by this function.
power    Power to use for the adjacency network (default = 2).
type     What type of network is being entered. Common choices are "signed" (default) and "unsigned". With "signed" negative correlations count against, whereas with "unsigned" negative correlations are treated identically as positive correlations.
...      Any other parameters accepted by the *adjacency* function

Value

Both functions output a character vector of genes, where the genes are the hub gene picked for each module, and the names correspond to the module in which each gene is a hub.

Author(s)

Jeremy Miller

Examples

```r
## Example: first simulate some data.

MEturquoise = sample(1:100,50)
MEblue = sample(1:100,50)
MEbrown = sample(1:100,50)
MEyellow = sample(1:100,50)
MEgreen = c(MEyellow[1:30], sample(1:100,20))
MERed = c(MEbrown[1:20], sample(1:100,30))
MEblack = c(MEblue [1:25], sample(1:100,25))
ME = data.frame(MEturquoise, MEblue, MEbrown, MEyellow, MEgreen, MERed, MEblack)
dat1 = simulateDatExpr(ME,300,c(0.2,0.1,0.08,0.051,0.05,0.042,0.041,0.3),
  signed=TRUE)
TOM1 = TOMsimilarityFromExpr(dat1$datExpr, networkType="signed")
colnames(TOM1) <- rownames(TOM1) <- colnames(dat1$datExpr)
treel <- tree2 <- fastcluster:::hclust(as.dist(1-TOM1), method="average")
colorh = labels2colors(dat1$allLabels)
hubs = chooseOneHubInEachModule(dat1$datExpr, colorh)
hubs
```
chooseTopHubInEachModule

*Chooses the top hub gene in each module*

**Description**

chooseTopHubInEachModule returns the gene in each module with the highest connectivity, looking at all genes in the expression file.

**Usage**

```r
chooseTopHubInEachModule(
  datExpr,  
  colorh,  
  omitColors = "grey",  
  power = 2,  
  type = "signed",  
  ...
)
```

**Arguments**

- `datExpr`: Gene expression data with rows as samples and columns as genes.
- `colorh`: The module assignments (color vectors) corresponding to the rows in `datExpr`.
- `omitColors`: All colors in this character vector (default is "grey") are ignored by this function.
- `power`: Power to use for the adjacency network (default = 2).
- `type`: What type of network is being entered. Common choices are "signed" (default) and "unsigned". With "signed" negative correlations count against, whereas with "unsigned" negative correlations are treated identically as positive correlations.
- `...`: Any other parameters accepted by the *adjacency* function

**Value**

Both functions output a character vector of genes, where the genes are the hub gene picked for each module, and the names correspond to the module in which each gene is a hub.

**Author(s)**

Jeremy Miller

**Examples**

```r
## Example: first simulate some data.

MEturquoise = sample(1:100,50)
Mblue = sample(1:100,50)
MBrown = sample(1:100,50)
MEyellow = sample(1:100,50)
MEgreen = c(MEyellow[1:30], sample(1:100,20))
MERed = c(MEbrowan[1:20], sample(1:100,30))
MEblack = c(MEblue[1:25], sample(1:100,25))
ME = data.frame(MEturquoise, Mblue, MEbrown, MEyellow, MEgreen, MERed, MEblack)
```
clusterCoef

Clustering coefficient calculation

Description
This function calculates the clustering coefficients for all nodes in the network given by the input adjacency matrix.

Usage
clusterCoef(adjMat)

Arguments
adjMat adjacency matrix

Value
A vector of clustering coefficients for each node.

Author(s)
Steve Horvath
coclustering

Co-clustering measure of cluster preservation between two clusterings

Description
The function calculates the co-clustering statistics for each module in the reference clustering.

Usage
coclustering(clusters.ref, clusters.test, tupleSize = 2, unassignedLabel = 0)

Arguments
clusters.ref Reference input clustering. A vector in which each element gives the cluster label of an object.
clusters.test Test input clustering. Must be a vector of the same size as clusters.ref.
tupleSize Co-clustering tuplet size.
unassignedLabel Optional specification of a clustering label that denotes unassigned objects. Objects with this label are excluded from the calculation.
Details

Co-clustering of cluster q in the reference clustering and cluster q’ in the test clustering measures
the overlap of clusters q and q’ by the number of tuplets that can be chosen from the overlap of
clusters q and q’ relative to the number of tuplets in cluster q. To arrive at a co-clustering measure
for cluster q, we sum the co-clustering of q and q’ over all clusters q’ in the test clustering. A value
close to 1 indicates high preservation of the reference cluster in the test clustering, while a value
close to zero indicates a low preservation.

Value

A vector in which each component corresponds to a cluster in the reference clustering. Entries give
the co-clustering measure of cluster preservation.

Author(s)

Peter Langfelder

References

For example, see Langfelder P, Luo R, Oldham MC, Horvath S (2011) Is My Network Module
Methods Supplement (Supplementary text 1) of that article.

See Also

modulePreservation for a large suite of module preservation statistics coClustering.permutationTest
for a permutation test for co-clustering significance

Examples

# An example with random (unrelated) clusters:

set.seed(1);  
nModules = 10;  
nGenes = 1000;  
c11 = sample(1:nModules, nGenes, replace = TRUE);  
c12 = sample(1:nModules, nGenes, replace = TRUE);  
coClustering(c11, c12)  

# For the same reference and test clustering:

coClustering(c11, c11)
Description

This function calculates permutation Z statistics that measure how different the co-clustering of modules in a reference and test clusterings is from random.

Usage

```r
coClustering.permutationTest(
    clusters.ref, clusters.test,
    tupletSize = 2,
    nPermutations = 100,
    unassignedLabel = 0,
    randomSeed = 12345, verbose = 0, indent = 0)
```

Arguments

- `clusters.ref`: Reference input clustering. A vector in which each element gives the cluster label of an object.
- `clusters.test`: Test input clustering. Must be a vector of the same size as `clusters.ref`.
- `tupletSize`: Co-clustering tuplet size.
- `nPermutations`: Number of permutations to execute. Since the function calculates parametric p-values, a relatively small number of permutations (at least 50) should be sufficient.
- `unassignedLabel`: Optional specification of a clustering label that denotes unassigned objects. Objects with this label are excluded from the calculation.
- `randomSeed`: Random seed for initializing the random number generator. If `NULL`, the generator is not initialized (useful for calling the function sequentially). The default assures reproducibility.
- `verbose`: If non-zero, function will print out progress messages.
- `indent`: Indentation for progress messages. Each unit adds two spaces.

Details

This function performs a permutation test to determine whether observed co-clustering statistics are significantly different from those expected by chance. It returns the observed co-clustering as well as the permutation Z statistic, calculated as \( \frac{\text{observed} - \text{mean}}{\text{sd}} \), where mean and sd are the mean and standard deviation of the co-clustering when the test clustering is repeatedly randomly permuted.

Value

- `observed`: the observed co-clustering measures for clusters in `clusters.ref`
- `Z`: permutation Z statistics
- `permuted.mean`: means of the co-clustering measures when the test clustering is permuted
- `permuted.sd`: standard deviations of the co-clustering measures when the test clustering is permuted
- `permuted.cc`: values of the co-clustering measure for each permutation of the test clustering. A matrix of dimensions (number of permutations) x (number of clusters in reference clustering).
Author(s)

Peter Langfelder

References


See Also
coclustering for calculation of the "observed" co-clustering measure modulePreservation for a large suite of module preservation statistics

Examples

set.seed(1);

nmodules = 5;
ngenomes = 100;

c1 = sample(c(1:nmodules), nGenes, replace = TRUE);
c2 = sample(c(1:nmodules), nGenes, replace = TRUE);

cc = coclustering(c1, c2)

# Choose a low number of permutations to make the example fast
ccPerm = coClustering.permutationTest(c1, c2, nPermutations = 20, verbose = 1);

ccPerm$observed

ccPerm$Z

# Combine c1 and c2 to obtain clustering that is somewhat similar to c1:

c3 = c2;
from1 = sample(c(TRUE, FALSE), nGenes, replace = TRUE);
c3[from1] = c1[from1];

ccPerm = coClustering.permutationTest(c1, c3, nPermutations = 20, verbose = 1);

# observed co-clustering is higher than before:
ccPerm$observed

# Note the high preservation Z statistics:
ccPerm$Z

Description

Abstractly speaking, the function allows one to collapse the rows of a numeric matrix, e.g. by forming an average or selecting one representative row for each group of rows specified by a grouping variable (referred to as rowGroup). The word "collapse" reflects the fact that the method yields a
new matrix whose rows correspond to other rows of the original input data. The function implements several network-based and biostatistical methods for finding a representative row for each group specified in rowGroup. Optionally, the function identifies the representative row according to the least number of missing data, the highest sample mean, the highest sample variance, the highest connectivity. One of the advantages of this function is that it implements default settings which have worked well in numerous applications. Below, we describe these default settings in more detail.

Usage

collapseRows(datET, rowGroup, rowID,  
method="MaxMean", connectivityBasedCollapsing=FALSE,  
methodFunction=NULL, connectivityPower=1,  
selectFewestMissing=TRUE, thresholdCombine=NA)

Arguments

datET matrix or data frame containing numeric values where rows correspond to variables (e.g. microarray probes) and columns correspond to observations (e.g. microarrays). Each row of datET must have a unique row identifier (specified in the vector rowID). The group label of each row is encoded in the vector rowGroup. While rowID should have non-missing, unique values (identifiers), the values of the vector rowGroup will typically not be unique since the function aims to pick a representative row for each group.

rowGroup character vector whose components contain the group label (e.g. a character string) for each row of datET. This vector needs to have the same length as the vector rowID. In gene expression applications, this vector could contain the gene symbol (or a co-expression module label).

rowID character vector of row identifiers. This should include all the rows from rownames(datET), but can include other rows. Its entries should be unique (no duplicates) and no missing values are permitted. If the row identifier is missing for a given row, we suggest you remove this row from datET before applying the function.

method character string for determining which method is used to choose a probe among exactly 2 corresponding rows or when connectivityBasedCollapsing=FALSE. These are the options: "MaxMean" (default) or "MinMean" = choose the row with the highest or lowest mean value, respectively. "maxRowVariance" = choose the row with the highest variance (across the columns of datET). "absMaxMean" or "absMinMean" = choose the row with the highest or lowest mean absolute value. "ME" = choose the eigenrow (first principal component of the rows in each group). Note that with this method option, connectivityBasedCollapsing is automatically set to FALSE. "Average" = for each column, take the average value of the rows in each group "function" = use this method for a user-input function (see the description of the argument methodFunction"). Note: if method="ME", "Average" or "function", the output parameters "group2row" and "selectedRow" are not informative.

connectivityBasedCollapsing logical value. If TRUE, groups with 3 or more corresponding rows will be represented by the row with the highest connectivity according to a signed weighted correlation network adjacency matrix among the corresponding rows. Recall that the connectivity is defined as the rows sum of the adjacency matrix. The signed weighted adjacency matrix is defined as A=(0.5+0.5*COR)^power where power is determined by the argument connectivityPower and COR denotes
collapseRows

The matrix of pairwise Pearson correlation coefficients among the corresponding rows.

methodFunction character string. It only needs to be specified if method="function" otherwise its input is ignored. Must be a function that takes a Nr x Nc matrix of numbers as input and outputs a vector with the length Nc (e.g., colMeans). This will then be the method used for collapsing values for multiple rows into a single value for the row.

connectivityPower Positive number (typically integer) for specifying the threshold (power) used to construct the signed weighted adjacency matrix, see the description of connectivityBasedCollapsing. This option is only used if connectivityBasedCollapsing=TRUE.

selectFewestMissing logical values. If TRUE (default), the input expression matrix is trimmed such that for each group only the rows with the fewest number of missing values are retained. In situations where an equal number of values are missing (or where there is no missing data), all rows for a given group are retained. Whether this value is set to TRUE or FALSE, all rows with >90% missing data are omitted from the analysis.

thresholdCombine Number between -1 and 1, or NA. If NA (default), this input is ignored. If a number between -1 and 1 is input, this value is taken as a threshold value, and collapseRows proceeds following the "maxMean" method, but ONLY for ids with correlations of R>thresholdCombine. Specifically: ...1) If there is one id/group, keep the id ...2) If there are 2 ids/group, take the maximum mean expression if their correlation is > thresholdCombine ...3) If there are 3+ ids/group, iteratively repeat (2) for the 2 ids with the highest correlation until all ids remaining have correlation < thresholdCombine for each group Note that this option usually results in more than one id per group; therefore, one must use care when implementing this option for use in comparisons between multiple matrices / data frames.

Details

The function is robust to missing data. Also, if rowIDs are missing, they are inferred according to the rownames of datET when possible. When a group corresponds to only 1 row then it is represented by this row since there is no other choice. Having said this, the row may be removed if it contains an excessive amount of missing data (90 percent or more missing values), see the description of the argument selectFewestMissing for more details.

A group is represented by a corresponding row with the fewest number of missing data if selectFewestMissing has been set to TRUE. Often several rows have the same minimum number of missing values (or no missing values) and a representative must be chosen among those rows. In this case we distinguish 2 situations: (1) If a group corresponds to exactly 2 rows then the corresponding row with the highest average is selected if method="maxMean". Alternative methods can be chosen as described in method. (2) If a group corresponds to more than 2 rows, then the function calculates a signed weighted correlation network (with power specified in connectivityPower) among the corresponding rows if connectivityBasedCollapsing=TRUE. Next the function calculates the network connectivity of each row (closely related to the sum or correlations with the other matching rows). Next it chooses the most highly connected row as representative. If connectivityBasedCollapsing=FALSE, then method is used. For both situations, if more than one row has the same value, the first such row is chosen.

Setting thresholdCombine is a special case of this function, as not all ids for a single group are necessarily collapsed–only those with similar expression patterns are collapsed. We suggest using
this option when the goal is to decrease the number of ids for computational reasons, but when ALL
ids for a single group should not be combined (for example, if two probes could represent different
splice variants for the same gene for many genes on a microarray).

Example application: when dealing with microarray gene expression data then the rows of datET
may correspond to unique probe identifiers and rowGroup may contain corresponding gene sym-
boles. Recall that multiple probes (specified using rowID=ProbeID) may correspond to the same
gene symbol (specified using rowGroup=GeneSymbol). In this case, datET contains the input ex-
pression data with rows as rowIDs and output expression data with rows as gene symbols, collapsing
all probes for a given gene symbol into one representative.

Value

The output is a list with the following components.

datETcollapsed is a numeric matrix with the same columns as the input matrix datET, but with
rows corresponding to the different row groups rather than individual row identi-
fiers. (If thresholdCombine is set, then rows still correspond to individual row
identifiers.)

group2row is a matrix whose rows correspond to the unique group labels and whose 2
columns report which group label (first column called group) is represented
by what row label (second column called selectedRowID). Set to NULL if
method="ME" or "function".

selectedRow is a logical vector whose components are TRUE for probes selected as represen-
tatives and FALSE otherwise. It has the same length as the vector probeID. Set
to NULL if method="ME" or "function".

Author(s)

Jeremy A. Miller, Steve Horvath, Peter Langfelder, Chaochao Cai

References


Examples

```
# EXAMPLE 1:
# The code simulates a data frame (called dat1) of correlated rows.
# You can skip this part and start at the line called Typical Input Data
# The first column of the data frame will contain row identifiers
# number of columns (e.g. observations or microarrays)
m=60
# number of rows (e.g. variables or probes on a microarray)
m=500
# seed module eigenvector for the simulateModule function
#true=rnorm(m)
# numeric data frame of n rows and m columns
datNumeric=data.frame(t(simulateModule(Mtrue,n)))
RowIdentifier=paste("Probe", 1:n, sep="")
ColumnName=paste("Sample",1:m, sep="")
```
We use collapseRows to calculate the module eigengene. First we create some sample data as in example 1 (or use your own!)

```r
m = 60
n = 500
Mtrue = rnorm(m)
numeric = data.frame(t(simulateModule(Mtrue, n)))
```

In this example, rows are genes, and groups are modules. We simulate a vector with n/100 modules, i.e. each row group corresponds to 100 rows.
### Description

This function selects only the most informative probe for each gene in a kME table, only keeping the probe which has the highest kME with respect to any module in the module membership matrix. This function is a special case of the function `collapseRows`.

### Usage

```r
collapseRowsUsingKME(MM, Gin, Pin = NULL, kMEcols = 1:dim(MM)[2])
```

### Arguments

- **MM**: A module membership (kME) table with at least a subset of the columns corresponding to kME values.
- **Gin**: Genes labels in a 1 to 1 correspondence with the rows of MM.
- **Pin**: If NULL (default), rownames of MM are assumed to be probe IDs. If entered, Pin must be the same length as Gin and correspond to probe IDs for MM.
- **kMEcols**: A numeric vector showing which columns in MM correspond to kME values. The default is all of them.

### Value

- **datETcollapsed**: A numeric matrix with the same columns as the input matrix MM, but with rows corresponding to the genes rather than the probes.
- **group2row**: A matrix whose rows correspond to the unique gene labels and whose 2 columns report which gene label (first column called group) is represented by what probe (second column called selectedRowID).
- **selectedRow**: A logical vector whose components are TRUE for probes selected as representatives and FALSE otherwise. It has the same length as the vector Pin.

### Author(s)

Jeremy Miller

### See Also

`collapseRows`
Examples

```r
# Example: first simulate some data
set.seed(100)
ME.A = sample(1:100,50); ME.B = sample(1:100,50)
ME.C = sample(1:100,50); ME.D = sample(1:100,50)
ME1 = data.frame(ME.A, ME.B, ME.C, ME.D)
simDataA = simulateDatExpr(ME1, 1000, c(0.2, 0.1, 0.08, 0.05, 0.3), signed=TRUE)
simDataB = simulateDatExpr(ME1, 1000, c(0.2, 0.1, 0.08, 0.05, 0.3), signed=TRUE)
Gin = c(colnames(simDataA$datExpr), colnames(simDataB$datExpr))
Pin = paste("Probe", 1:length(Gin), sep=".")
datExpr = cbind(simDataA$datExpr, simDataB$datExpr)
MM = corAndPvalue(datExpr, ME1)$cor

# Now run the function and see some example output
results = collapseRowsUsingKME(MM, Gin, Pin)
head(results$MMcollapsed)
head(results$group2Row)
head(results$selectedRow)
```

**collectGarbage**

*Iterative garbage collection.*

**Description**

Performs garbage collection until free memory indicators show no change.

**Usage**

```r
collectGarbage()
```

**Value**

None.

**Author(s)**

Steve Horvath

**colQuantileC**

*Fast column- and row-wise quantile of a matrix.*

**Description**

Fast calculation of column- and row-wise quantiles of a matrix at a single probability. Implemented via compiled code, it is much faster than the equivalent `apply(data, 2, quantile, prob = p)`.

**Usage**

```r
colQuantileC(data, p)
rowQuantileC(data, p)
```
Arguments

- **data**: a numerical matrix column-wise quantiles are desired. Missing values are removed.
- **p**: a single probability at which the quantile is to be calculated.

Details

At present, only one quantile type is implemented, namely the default type 7 used by R.

Value

A vector of length equal the number of columns (for `colQuantileC`) or rows (for `rowQuantileC`) in `data` containing the column- or row-wise quantiles.

Author(s)

Peter Langfelder

See Also

`quantile`

---

**conformityBasedNetworkConcepts**

*Calculation of conformity-based network concepts.*

Description

This function computes 3 types of network concepts (also known as network indices or statistics) based on an adjacency matrix and optionally a node significance measure.

Usage

```r
conformityBasedNetworkConcepts(adj, GS = NULL)
```

Arguments

- **adj**: adjacency matrix. A symmetric matrix with components between 0 and 1.
- **GS**: optional node significance measure. A vector with length equal the dimension of `adj`.

Details

This function computes 3 types of network concepts (also known as network indices or statistics) based on an adjacency matrix and optionally a node significance measure. Specifically, it computes I) fundamental network concepts, II) conformity based network concepts, and III) approximate conformity based network concepts. These network concepts are defined for any symmetric adjacency matrix (weighted and unweighted). The network concepts are described in Dong and Horvath (2007) and Horvath and Dong (2008). In the following, we use the term gene and node interchangeably since these methods were originally developed for gene networks. In the following, we briefly describe the 3 types of network concepts:
Type I: fundamental network concepts are defined as a function of the off-diagonal elements of an adjacency matrix $A$ and/or a node significance measure $GS$. Type II: conformity-based network concepts are functions of the off-diagonal elements of the conformity based adjacency matrix $A.CF = CF \cdot t(CF)$ and/or the node significance measure. These network concepts are defined for any network for which a conformity vector can be defined. Details: For any adjacency matrix $A$, the conformity vector $CF$ is calculated by requiring that $A[i,j]$ is approximately equal to $CF[i] \cdot CF[j]$. Using the conformity one can define the matrix $A.CF = CF \cdot t(CF)$ which is the outer product of the conformity vector with itself. In general, $A.CF$ is not an adjacency matrix since its diagonal elements are different from 1. If the off-diagonal elements of $A.CF$ are similar to those of $A$ according to the Frobenius matrix norm, then $A$ is approximately factorizable. To measure the factorizability of a network, one can calculate the Factorizability, which is a number between 0 and 1 (Dong and Horvath 2007). The conformity is defined using a monotonic, iterative algorithm that maximizes the factorizability measure. Type III: approximate conformity based network concepts are functions of all elements of the conformity based adjacency matrix $A.CF$ (including the diagonal) and/or the node significance measure $GS$. These network concepts are very useful for deriving relationships between network concepts in networks that are approximately factorizable.

**Value**

A list with the following components:

- **Factorizability**
  - A number between 0 and 1 giving the factorizability of the matrix. The closer to 1 the higher the evidence of factorizability, that is, $A-I$ is close to outer($CF,CF$)-diag($CF^2$).

- **fundamentalNCs**
  - fundamental network concepts, that is network concepts calculated directly from the given adjacency matrix $adj$. A list with components ScaledConnectivity (giving the scaled connectivity of each node), Connectivity (connectivity of each node), ClusterCoef (the clustering coefficient of each node), MAR (maximum adjacency ratio of each node), Density (the mean density of the network), Centralization (the centralization of the network), Heterogeneity (the heterogeneity of the network). If the input node significance $gs$ is specified, the following additional components are included: NetworkSignificance (network significance, the mean node significance), and HubNodeSignificance (hub node significance given by the linear regression of node significance on connectivity).

- **conformityBasedNCs**
  - network concepts based on an approximate adjacency matrix given by the outer product of the conformity vector but with unit diagonal. A list with components Conformity (the conformity vector) and Connectivity.CF, ClusterCoef.CF, MAR.CF, Density.CF giving the conformity-based analogs of the above network concepts.

- **approximateConformityBasedNCs**
  - network concepts based on an approximate adjacency matrix given by the outer product of the conformity vector. A list with components Conformity (the conformity vector) and Connectivity.CF.App, ClusterCoef.CF.App, MAR.CF.App, Density.CF.App giving the conformity-based analogs of the above network concepts.

**Author(s)**

Steve Horvath
References


See Also

networkConcepts for calculation of eigennode based network concepts for a correlation network;

fundamentalNetworkConcepts for calculation of fundamental network concepts only.

correlationDecomposition

Conformity and module based decomposition of a network adjacency matrix.

Description

The function calculates the conformity based approximation $A_{CF}$ of an adjacency matrix and a factorizability measure codeFactorizability. If a module assignment $Cl$ is provided, it also estimates a corresponding intermodular adjacency matrix. In this case, function automatically carries out the module- and conformity based decomposition of the adjacency matrix described in chapter 2 of (Horvath 2011).

Usage

conformityDecomposition(adj, Cl = NULL)

Arguments

adj a symmetric numeric matrix (or data frame) whose entries lie between 0 and 1.
Cl a vector (or factor variable) of length equal to the number of rows of adj. The variable assigns each network node (row of adj) to a module. The entries of Cl could be integers or character strings.

Details

We distinguish two situation depending on whether or not Cl equals NULL. 1) Let us start out assuming that Cl = NULL. In this case, the function calculates the conformity vector for a general, possibly non-factorizable network adj by minimizing a quadratic (sums of squares) loss function. The conformity and factorizability for an adjacency matrix is defined in (Dong and Horvath 2007, Horvath and Dong 2008) but we briefly describe it in the following. A network is called exactly factorizable if the pairwise connection strength (adjacency) between 2 network nodes can be factored into node specific contributions, named node 'conformity', i.e. if $adj[i,j]=Conformity[i]*Conformity[j]$. The conformity turns out to be highly related to the network connectivity (aka degree). If adj is not exactly factorizable, then the function conformityDecomposition calculates a conformity vector of the exactly factorizable network that best approximates adj. The factorizability measure Factorizability is a number between 0 and 1. The higher Factorizability, the more factorizable is adj. Warning: the algorithm may only converge to a local optimum and it may not converge at all. Also see the notes below.
2) Let us now assume that $C_1$ is not NULL, i.e. it specifies the module assignment of each node. Then the function calculates a module- and CF-based approximation of $\text{adj}$ (explained in chapter 2 in Horvath 2011). In this case, the function calculates a conformity vector $\text{Conformity}$ and a matrix $\text{IntermodularAdjacency}$ such that $\text{adj}[i,j]$ is approximately equal to $\text{Conformity}[i] \times \text{Conformity}[j] \times \text{IntermodularAdjacency}$, where $\text{module.index}[i]$ is the row of the matrix $\text{IntermodularAdjacency}$ that corresponds to the module assigned to node $i$. To estimate $\text{Conformity}$ and a matrix $\text{IntermodularAdjacency}$, the function attempts to minimize a quadratic loss function (sums of squares). Currently, the function only implements a heuristic algorithm for optimizing the objective function (chapter 2 of Horvath 2011). Another, more accurate Majorization Minorization (MM) algorithm for the decomposition is implemented in the function $\text{propensityDecomposition}$ by Ranola et al (2011).

**Value**

- **A.CF**
  
  A symmetric matrix that approximates the input matrix $\text{adj}$. Roughly speaking, the $ij$-the element of the matrix equals $\text{Conformity}[i] \times \text{Conformity}[j] \times \text{IntermodularAdjacency}$, where $\text{module.index}[i]$ is the row of the matrix $\text{IntermodularAdjacency}$ that corresponds to the module assigned to node $i$.

- **Conformity**
  
  A numeric vector whose entries correspond to the rows of $\text{codeadj}$. If $C_1$=NULL then $\text{Conformity}[i]$ is the conformity. If $C_1$ is not NULL then $\text{Conformity}[i]$ is the intramodular conformity with respect to the module that node $i$ belongs to.

- **IntermodularAdjacency**
  
  A symmetric matrix (data frame) whose rows and columns correspond to the number of modules specified in $C_1$. Interpretation: it measures the similarity (adjacency) between the modules. In this case, the rows (and columns) of $\text{IntermodularAdjacency}$ correspond to the entries of $C_1\.level$.

- **Factorizability**
  
  A number between 0 and 1. If $C_1$=NULL then it equals 1, if (and only if) $\text{adj}$ is exactly factorizable. If $C_1$ is a vector, then it measures how well the module- and CF based decomposition approximates $\text{adj}$.

- **$C_1\.level$**
  
  A vector of character strings which correspond to the factor levels of the module assignment $C_1$. Incidentally, the function automatically turns $C_1$ into a factor variable. The components of $\text{Conformity}$ and $\text{IntramodularFactorizability}$ correspond to the entries of $C_1\.level$.

- **IntramodularFactorizability**
  
  A numeric vector of length equal to the number of modules specified by $C_1$. Its entries report the factorizability measure for each module. The components correspond to the entries of $C_1\.level$.

**listConformity**

**Note**

Regarding the situation when $C_1$=NULL. One can easily show that the conformity vector is not unique if $\text{adj}$ contains only 2 nodes. However, for more than 2 nodes the conformity is uniquely defined when dealing with an exactly factorizable weighted network whose entries $\text{adj}[i,j]$ are larger than 0. In this case, one can get explicit formulas for the conformity (Dong and Horvath 2007).

**Author(s)**

Steve Horvath
References


See Also

conformityBasedNetworkConcepts

Examples

# assume the number of nodes can be divided by 2 and by 3
n=6
# here is a perfectly factorizable matrix
A=matrix(1,nrow=n,ncol=n)
# this provides the conformity vector and factorizability measure
conformityDecomposition(adj=A)
# now assume we have a class assignment
Cl=rep(c(1,2),c(n/2,n/2))
conformityDecomposition(adj=A,Cl=Cl)
# here is a block diagonal matrix
blockdiag.A=A
blockdiag.A[1:(n/3),(n/3+1):n]=0
blockdiag.A[(n/3+1):n , 1:(n/3)]=0
block.Cl=rep(c(1,2),c(n/3,2*n/3))
conformityDecomposition(adj= blockdiag.A,Cl=block.Cl)

# another block diagonal matrix
blockdiag.A=A
blockdiag.A[1:(n/3),(n/3+1):n]=0.3
blockdiag.A[(n/3+1):n , 1:(n/3)]=0.3
block.Cl=rep(c(1,2),c(n/3,2*n/3))
conformityDecomposition(adj= blockdiag.A,Cl=block.Cl)
Arguments

multiExpr  Expression data in the multi-set format (see checkSets). A vector of lists, one per set. Each set must contain a component data that contains the expression data. Rows correspond to samples and columns to genes or probes. Two or more sets of data must be included and adjacencies cannot be used.

softPower  Soft thresholding power used to make each of the networks in multiExpr.

TOM  A LIST of matrices holding the topological overlap corresponding to the sets in multiExpr, if they have already been calculated. Otherwise, keep TOM set as NULL (default), and TOM similarities will be calculated using the WGCNA defaults. If inputted, this variable must be a list with each entree a TOM corresponding to the same entries in multiExpr.

Value

consensusTOM  The TOM difference matrix (1-TOM similarity) corresponding to the consensus network.

consTree  Returned value is the same as that of hclust: An object of class hclust which describes the tree produced by the clustering process. This tree corresponds to the dissimilarity matrix consensusTOM.

Author(s)

Peter Langfelder, Steve Horvath, Jeremy Miller

References


See Also

blockwiseConsensusModules

Examples

# Example consensus network using two simulated data sets

set.seed = 100
MEturquoise = sample(1:100,50)
MEblue = sample(1:100,50)
MEbrown = sample(1:100,50)
MEyellow = sample(1:100,50)
MEgreen = sample(1:100,50)

ME = data.frame(MEturquoise, MEblue, MEbrown, MEyellow, MEgreen)
dat1 = simulateDataExpr(ME,300,c(0.2, 0.10, 0.10, 0.10, 0.2), signed=TRUE)
dat2 = simulateDataExpr(ME,300,c(0.18, 0.11, 0.11, 0.09, 0.11, 0.23), signed=TRUE)
multiExpr = list(S1=list(data=dat1$datExpr),S2=list(data=dat2$datExpr))
softPower=8

consensusNetwork = consensusDissTOMandTree(multiExpr, softPower)
plotDendroAndColors(consensusNetwork$consTree, cbind(labels2colors(dat1$allLabels), labels2colors(dat2$allLabels)), c("S1","S2"), dendroLabels=FALSE)
Calculate consensus kME (eigengene-based connectivities) across multiple data sets.

Description

Calculate consensus kME (eigengene-based connectivities) across multiple data sets, typically following a consensus module analysis.

Usage

consensusKME(
  multiExpr,
  moduleLabels,
  multiEigengenes = NULL,
  consensusQuantile = 0,
  signed = TRUE,
  useModules = NULL,
  metaAnalysisWeights = NULL,
  corAndPvalueFnc = corAndPvalue,
  corOptions = list(),
  corComponent = "cor",
  getQvalues = FALSE,
  useRankPvalue = TRUE,
  rankPvalueOptions = list(calculateQvalue = getQvalues,
                           pValueMethod = "scale"),
  setNames = NULL,
  excludeGrey = TRUE,
  greyLabel = if (is.numeric(moduleLabels)) 0 else "grey")

Arguments

multiExpr  Expression (or other numeric) data in a multi-set format. A vector of lists; in each list there must be a component named 'data' whose content is a matrix or dataframe or array of dimension 2.

moduleLabels  Module labels: one label for each gene in multiExpr.

multiEigengenes  Optional eigengenes of modules specified in moduleLabels. If not given, will be calculated from multiExpr.

signed  logical: should the network be considered signed? In signed networks (TRUE), negative kME values are not considered significant and the corresponding p-values will be one-sided. In unsigned networks (FALSE), negative kME values are considered significant and the corresponding p-values will be two-sided.

useModules  Optional specification of module labels to which the analysis should be restricted. This could be useful if there are many modules, most of which are not interesting. Note that the "grey" module cannot be used with useModules.

consensusQuantile  Quantile for the consensus calculation. Should be a number between 0 (minimum) and 1.

metaAnalysisWeights  Optional specification of meta-analysis weights for each input set. If given, must be a numeric vector of length equal the number of input data sets (i.e.,
consensusKME

These weights will be used in addition to constant weights and weights proportional to number of samples (observations) in each set.

corAndPvalueFnc
Function that calculates associations between expression profiles and eigengenes. See details.

corOptions
List giving additional arguments to function corAndPvalueFnc. See details.

corComponent
Name of the component of output of corAndPvalueFnc that contains the actual correlation.

getQvalues
logical: should q-values (estimates of FDR) be calculated?

useRankPvalue
Logical: should the rankPvalue function be used to obtain alternative meta-analysis statistics?

rankPvalueOptions
Additional options for function rankPvalue. These include na.last (default "keep"), ties.method (default "average"), calculateQvalue (default copied from input getQvalues), and pValueMethod (default "scale"). See the help file for rankPvalue for full details.

setNames
names for the input sets. If not given, will be taken from names(multiExpr). If those are NULL as well, the names will be "Set_1", "Set_2", ....

excludeGrey
logical: should the grey module be excluded from the kME tables? Since the grey module is typically not a real module, it makes little sense to report kME values for it.

greyLabel
label that labels the grey module.

Details

The function corAndPvalueFnc is currently expected to accept arguments x (gene expression profiles), y (eigengene expression profiles), and alternative with possibilities at least "greater", "two.sided". Any additional arguments can be passed via corOptions.

The function corAndPvalueFnc should return a list which at the least contains (1) a matrix of associations of genes and eigengenes (this component should have the name given by corComponent), and (2) a matrix of the corresponding p-values, named "p" or "p.value". Other components are optional but for full functionality should include (3) nObs giving the number of observations for each association (which is the number of samples less number of missing data - this can in principle vary from association to association), and (4) Z giving a Z statistic for each observation. If these are missing, nObs is calculated in the main function, and calculations using the Z statistic are skipped.

Value

Data frame with the following components (for easier readability the order here is not the same as in the actual output):

ID
Gene ID, taken from the column names of the first input data set

consensus.kME.1, consensus.kME.2, ...
Consensus KME (that is, the requested quantile of the kMEs in the individual data sets) in each module for each gene across the input data sets. The module labels (here 1, 2, etc.) correspond to those in moduleLabels.

weightedAverage.equalWeights.kME1, weightedAverage.equalWeights.kME2, ...
Average kME in each module for each gene across the input data sets.
Weighted average kME in each module for each gene across the input data sets. The weight of each data set is proportional to the square root of the number of samples in the set.

Weighted average kME in each module for each gene across the input data sets. The weight of each data set is proportional to the number of samples in the set.

(Only present if input metaAnalysisWeights is non-NULL.) Weighted average kME in each module for each gene across the input data sets. The weight of each data set is given in metaAnalysisWeights.

Meta-analysis Z statistic for kME in each module, obtained by weighing the Z scores in each set equally. Only returned if the function corAndPvalueFnc returns the Z statistics corresponding to the correlations.

Meta-analysis Z statistic for kME in each module, obtained by weighing the Z scores in each set by the square root of the number of samples. Only returned if the function corAndPvalueFnc returns the Z statistics corresponding to the correlations.

Meta-analysis Z statistic for kME in each module, obtained by weighing the Z scores in each set by the number of samples. Only returned if the function corAndPvalueFnc returns the Z statistics corresponding to the correlations.

Meta-analysis Z statistic for kME in each module, obtained by weighing the Z scores in each set by metaAnalysisWeights. Only returned if metaAnalysisWeights is non-NULL and the function corAndPvalueFnc returns the Z statistics corresponding to the correlations.

p-values obtained from the equal-weight meta-analysis Z statistics. Only returned if the function corAndPvalueFnc returns the Z statistics corresponding to the correlations.

p-values obtained from the meta-analysis Z statistics with weights proportional to the square root of the number of samples. Only returned if the function corAndPvalueFnc returns the Z statistics corresponding to the correlations.

p-values obtained from the degree-of-freedom weight meta-analysis Z statistics. Only returned if the function corAndPvalueFnc returns the Z statistics corresponding to the correlations.

p-values obtained from the user-supplied weight meta-analysis Z statistics. Only returned if metaAnalysisWeights is non-NULL and the function corAndPvalueFnc returns the Z statistics corresponding to the kME values.

q-values obtained from the equal-weight meta-analysis p-values. Only present if getQvalues is TRUE and the function corAndPvalueFnc returns the Z statistics corresponding to the kME values.

q-values obtained from the meta-analysis p-values with weights proportional to the square root of the number of samples. Only present if getQvalues is TRUE
and the function \texttt{corAndPvalueFnc} returns the Z statistics corresponding to the kME values.

\texttt{meta.q.DoFWeights.kME1, meta.q.DoFWeights.kME2, ...}

q-values obtained from the degree-of-freedom weight meta-analysis p-values. Only present if \texttt{getQvalues} is \texttt{TRUE} and the function \texttt{corAndPvalueFnc} returns the Z statistics corresponding to the kME values.

\texttt{meta.q.userWeights.kME1, meta.q.userWeights.kME2, ...}

q-values obtained from the user-specified weight meta-analysis p-values. Only present if \texttt{metaAnalysisWeights} is non-\texttt{NULL}, \texttt{getQvalues} is \texttt{TRUE} and the function \texttt{corAndPvalueFnc} returns the Z statistics corresponding to the kME values.

The next set of columns contain the results of function \texttt{rankPvalue} and are only present if input \texttt{useRankPvalue} is \texttt{TRUE}. Some columns may be missing depending on the options specified in \texttt{rankPvalueOptions}. We explicitly list columns that are based on weighing each set equally; names of these columns carry the suffix `.equalWeights`

\begin{verbatim}
pValueExtremeRank.ME1.equalWeights, pValueExtremeRank.ME2.equalWeights, ...

This is the minimum between \texttt{pValueLowRank} and \texttt{pValueHighRank}, i.e. \texttt{min(pValueLow, pValueHigh)}

dpValueLowRank.ME1.equalWeights, dpValueLowRank.ME2.equalWeights, ...

Asymptotic p-value for observing a consistently low value across the columns of \texttt{datS} based on the rank method.

dpValueHighRank.ME1.equalWeights, dpValueHighRank.ME2.equalWeights, ...

Asymptotic p-value for observing a consistently low value across the columns of \texttt{datS} based on the rank method.

dpValueExtremeScale.ME1.equalWeights, dpValueExtremeScale.ME2.equalWeights, ...

This is the minimum between \texttt{pValueLowScale} and \texttt{pValueHighScale}, i.e. \texttt{min(pValueLow, pValueHigh)}

dpValueLowScale.ME1.equalWeights, dpValueLowScale.ME2.equalWeights, ...

Asymptotic p-value for observing a consistently low value across the columns of \texttt{datS} based on the Scale method.

dpValueHighScale.ME1.equalWeights, dpValueHighScale.ME2.equalWeights, ...

Asymptotic p-value for observing a consistently low value across the columns of \texttt{datS} based on the Scale method.

qValueExtremeRank.ME1.equalWeights, qValueExtremeRank.ME2.equalWeights, ...

local false discovery rate (q-value) corresponding to the p-value \texttt{pValueExtremeRank}

qValueLowRank.ME1.equalWeights, qValueLowRank.ME2.equalWeights, ...

local false discovery rate (q-value) corresponding to the p-value \texttt{pValueLowRank}

qValueHighRank.ME1.equalWeights, qValueHighRank.ME2.equalWeights, ...

local false discovery rate (q-value) corresponding to the p-value \texttt{pValueHighRank}

qValueExtremeScale.ME1.equalWeights, qValueExtremeScale.ME2.equalWeights, ...

local false discovery rate (q-value) corresponding to the p-value \texttt{pValueExtremeScale}

qValueLowScale.ME1.equalWeights, qValueLowScale.ME2.equalWeights, ...

local false discovery rate (q-value) corresponding to the p-value \texttt{pValueLowScale}

qValueHighScale.ME1.equalWeights, qValueHighScale.ME2.equalWeights, ...

local false discovery rate (q-value) corresponding to the p-value \texttt{pValueHighScale}
\end{verbatim}
Analogous columns corresponding to weighing individual sets by the square root of the number of samples, by number of samples, and by user weights (if given). The corresponding column name suffixes are .RootDoFWeights, .DoFWeights, and .userWeights.

The following set of columns summarize kME in individual input data sets.

- \( \text{kME1.Set}_1, \text{kME1.Set}_2, \ldots, \text{kME2.Set}_1, \text{kME2.Set}_2, \ldots \) kME values for each gene in each module in each given data set.
- \( \text{p.kME1.Set}_1, \text{p.kME1.Set}_2, \ldots, \text{p.kME2.Set}_1, \text{p.kME2.Set}_2, \ldots \) p-values corresponding to kME values for each gene in each module in each given data set.
- \( \text{q.kME1.Set}_1, \text{q.kME1.Set}_2, \ldots, \text{q.kME2.Set}_1, \text{q.kME2.Set}_2, \ldots \) q-values corresponding to kME values for each gene in each module in each given data set. Only returned if getQvalues is TRUE.
- \( \text{Z.kME1.Set}_1, \text{Z.kME1.Set}_2, \ldots, \text{Z.kME2.Set}_1, \text{Z.kME2.Set}_2, \ldots \) Z statistics corresponding to kME values for each gene in each module in each given data set. Only present if the function corAndPvalueFnc returns the Z statistics corresponding to the kME values.

Author(s)

Peter Langfelder

References


See Also

- signedKME for eigengene based connectivity in a single data set. corAndPvalue, bicorAndPvalue for two alternatives for calculating correlations and the corresponding p-values and Z scores. Both can be used with this function.

---

**consensusMEDissimilarity**

*Consensus dissimilarity of module eigengenes.*

**Description**

Calculates consensus dissimilarity (1-corr) of given module eigengenes relaized in several sets.

**Usage**

```r
consensusMEDissimilarity(MEs, useAbs = FALSE, useSets = NULL, method = "consensus")
```
**consensusOrderMEs**

### Arguments

- **MEs**: Module eigengenes of the same modules in several sets.
- **useAbs**: Controls whether absolute value of correlation should be used instead of correlation in the calculation of dissimilarity.
- **useSets**: If the consensus is to include only a selection of the given sets, this vector (or scalar in the case of a single set) can be used to specify the selection. If NULL, all sets will be used.
- **method**: A character string giving the method to use. Allowed values are (abbreviations of) "consensus" and "majority". The consensus dissimilarity is calculated as the minimum of given set dissimilarities for "consensus" and as the average for "majority".

### Details

This function calculates the individual set dissimilarities of the given eigengenes in each set, then takes the (parallel) maximum or average over all sets. For details on the structure of input data, see `checkSets`.

### Value

A dataframe containing the matrix of dissimilarities, with names and rownames set appropriately.

### Author(s)

Peter Langfelder, <Peter.Langfelder@gmail.com>

### See Also

- `checkSets`
Arguments

**MEs**
Module eigengenes of several sets in a multi-set format (see `checkSets`). A vector of lists, with each list corresponding to one dataset and the module eigengenes in the component data, that is `MEs[[set]]$data[[sample, module]]` is the expression of the eigengene of module `module` in sample `sample` in dataset `set`. The number of samples can be different between the sets, but the modules must be the same.

**useAbs**
Controls whether vector similarity should be given by absolute value of correlation or plain correlation.

**useSets**
Allows the user to specify for which sets the eigengene ordering is to be performed.

**greyLast**
Normally the color grey is reserved for unassigned genes; hence the grey module is not a proper module and it is conventional to put it last. If this is not desired, set the parameter to `FALSE`.

**greyName**
Name of the grey module eigengene.

**method**
A character string giving the method to be used calculating the consensus dissimilarity. Allowed values are (abbreviations of) "consensus" and "majority". The consensus dissimilarity is calculated as the maximum of given set dissimilarities for "consensus" and as the average for "majority".

Details

Ordering module eigengenes is useful for plotting purposes. This function calculates the consensus or majority dissimilarity of given eigengenes over the sets specified by `useSets` (defaults to all sets). A hierarchical dendrogram is calculated using the dissimilarity and the order given by the dendrogram is used for the eigengenes in all other sets.

Value

A vector of lists of the same type as `MEs` containing the re-ordered eigengenes.

Author(s)

Peter Langfelder, `<Peter.Langfelder@gmail.com>`

See Also

`moduleEigengenes`, `multiSetMEs`, `orderMEs`

---

**consensusProjectiveKMeans**

Consensus projective K-means (pre-)clustering of expression data

Description

Implementation of a consensus variant of K-means clustering for expression data across multiple data sets.
Usage

```r
consensusProjectiveKMeans(
  multiExpr,
  preferredSize = 5000,
  nCenters = NULL,
  sizePenaltyPower = 4,
  networkType = "unsigned",
  randomSeed = 54321,
  checkData = TRUE,
  imputeMissing = TRUE,
  useMean = (length(multiExpr) > 3),
  maxIterations = 1000,
  verbose = 0, indent = 0)
```

Arguments

- **multiExpr**: expression data in the multi-set format (see checkSets). A vector of lists, one per set. Each set must contain a component data that contains the expression data, with rows corresponding to samples and columns to genes or probes.
- **preferredSize**: preferred maximum size of clusters.
- **nCenters**: number of initial clusters. Empirical evidence suggests that more centers will give a better preclustering; the default is \( \text{as.integer}(\min(n\text{Genes}/20, \text{preferredSize}/2/n\text{Genes})) \) and is an attempt to arrive at a reasonable number given the resources available.
- **sizePenaltyPower**: parameter specifying how severe is the penalty for clusters that exceed `preferredSize`.
- **networkType**: network type. Allowed values are (unique abbreviations of) "unsigned", "signed", "signed hybrid". See adjacency.
- **randomSeed**: integer to be used as seed for the random number generator before the function starts. If a current seed exists, it is saved and restored upon exit.
- **checkData**: logical: should data be checked for genes with zero variance and genes and samples with excessive numbers of missing samples? Bad samples are ignored; returned cluster assignment for bad genes will be NA.
- **imputeMissing**: logical: should missing values in `datExpr` be imputed before the calculations start? If the missing data are not imputed, they will be replaced by 0 which can be problematic.
- **useMean**: logical: should mean distance across sets be used instead of maximum? See details.
- **maxIterations**: maximum iterations to be attempted.
- **verbose**: integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
- **indent**: indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

The principal aim of this function within WGCNA is to pre-cluster a large number of genes into smaller blocks that can be handled using standard WGCNA techniques.

This function implements a variant of K-means clustering that is suitable for co-expression analysis. Cluster centers are defined by the first principal component, and distances by correlation. Consensus
distance across several sets is defined as the maximum of the corresponding distances in individual sets; however, if useMean is set, the mean distance will be used instead of the maximum. The distance between a gene and a center of a cluster is multiplied by a factor of $\max(clusterSize/preferredSize, 1)^{sizePenaltyPower}$, thus penalizing clusters whose size exceeds preferredSize. The function starts with randomly generated cluster assignment (hence the need to set the random seed for repeatability) and executes iterations of calculating new centers and reassigning genes to nearest (in the consensus sense) center until the clustering becomes stable. Before returning, nearby clusters are iteratively combined if their combined size is below preferredSize.

Consensus distance defined as maximum of distances in all sets is consistent with the approach taken in blockwiseConsensusModules, but the procedure may not converge. Hence it is advisable to use the mean as consensus in cases where there are multiple data sets (4 or more, say) and/or if the input data sets are very different.

The standard principal component calculation via the function svd fails from time to time (likely a convergence problem of the underlying lapack functions). Such errors are trapped and the principal component is approximated by a weighted average of expression profiles in the cluster. If verbose is set above 2, an informational message is printed whenever this approximation is used.

Value

A list with the following components:

clusters a numerical vector with one component per input gene, giving the cluster number in which the gene is assigned.

centers a vector of lists, one list per set. Each list contains a component data that contains a matrix whose columns are the cluster centers in the corresponding set.

unmergedClusters a numerical vector with one component per input gene, giving the cluster number in which the gene was assigned before the final merging step.

unmergedCenters a vector of lists, one list per set. Each list contains a component data that contains a matrix whose columns are the cluster centers before merging in the corresponding set.

Author(s)

Peter Langfelder

See Also

projectiveKMeans

---

Consensus network (topological overlap).

Description

Calculation of a consensus network (topological overlap).
Usage

```r
consensusTOM(
  # Supply either ...
  # ... information needed to calculate individual TOMs
  multiExpr,
  # Data checking options
  checkMissingData = TRUE,
  # Blocking options
  blocks = NULL,
  maxBlockSize = 5000,
  randomSeed = 12345,
  # Network construction arguments: correlation options
  corType = \"pearson\",
  maxPOutliers = 1,
  quickCor = \0,
  pearsonFallback = \"individual\",
  cosineCorrelation = FALSE,
  # Adjacency function options
  power = 6,
  networkType = \"unsigned\",
  checkPower = TRUE,
  # Topological overlap options
  TOMType = \"unsigned\",
  TOMDenom = \"min\",
  # Save individual TOMs?
  saveIndividualTOMs = TRUE,
  individualTOMFileNames = \"individualTOM-Set%s-Block%b.RData\",
  # ... or individual TOM information
  individualTOMInfo = NULL,
  useIndivTOMSubset = NULL,
  # Consensus calculation options
  useBlocks = NULL,
  networkCalibration = c(\"single quantile\", \"full quantile\", \"none\"),
  # Save calibrated TOMs?
  saveCalibratedIndividualTOMs = FALSE,
  calibratedIndividualTOMFilePattern = \"calibratedIndividualTOM-Set%s-Block%b.RData\",
)```
# Simple quantile calibration options
real calibrationQuantile = 0.95,
logical sampleForCalibration = TRUE, sampleForCalibrationFactor = 10000,
real getNetworkCalibrationSamples = FALSE,

# Consensus definition
real consensusQuantile = 0,
logical useMean = FALSE,
character string setWeights = "null",

# Return options
logical saveConsensusTOMs = TRUE,
character string consensusTOMfilenames = "consensusTOM-Block\%b.RData",
logical returnTOMs = FALSE,

# Internal handling of TOMs
logical useDiskCache = TRUE, real chunkSize = "null",
character string cacheDir = ".",
character string cacheBase = ".blockConsModsCache",
real nThreads = 1,

# Diagnostic messages
logical verbose = 1,
real indent = 0)

Arguments

multiExpr
expression data in the multi-set format (see checkSets). A vector of lists, one per set. Each set must contain a component data that contains the expression data, with rows corresponding to samples and columns to genes or probes.

checkMissingData
logical: should data be checked for excessive numbers of missing entries in genes and samples, and for genes with zero variance? See details.

blocks
optional specification of blocks in which hierarchical clustering and module detection should be performed. If given, must be a numeric vector with one entry per gene of multiExpr giving the number of the block to which the corresponding gene belongs.

maxBlockSize
integer giving maximum block size for module detection. Ignored if blocks above is non-NULL. Otherwise, if the number of genes in datExpr exceeds maxBlockSize, genes will be pre-clustered into blocks whose size should not exceed maxBlockSize.

randomSeed
integer to be used as seed for the random number generator before the function starts. If a current seed exists, it is saved and restored upon exit. If null is given, the function will not save and restore the seed.

corType
character string specifying the correlation to be used. Allowed values are (unique abbreviations of) "pearson" and "bicor", corresponding to Pearson and bid-weight midcorrelation, respectively. Missing values are handled using the pairwise.complete.obs option.

maxPOutliers
only used for corType=="bicor". Specifies the maximum percentile of data that can be considered outliers on either side of the median separately. For each
side of the median, if higher percentile than maxPOutliers is considered an outlier by the weight function based on 9*mad(x), the width of the weight function is increased such that the percentile of outliers on that side of the median equals maxPOutliers. Using maxPOutliers=1 will effectively disable all weight function broadening; using maxPOutliers=0 will give results that are quite similar (but not equal to) Pearson correlation.

quickCor real number between 0 and 1 that controls the handling of missing data in the calculation of correlations. See details.

pearsonFallback Specifies whether the bicor calculation, if used, should revert to Pearson when median absolute deviation (mad) is zero. Recognized values are (abbreviations of) "none", "individual", "all". If set to "none", zero mad will result in NA for the corresponding correlation. If set to "individual", Pearson calculation will be used only for columns that have zero mad. If set to "all", the presence of a single zero mad will cause the whole variable to be treated in Pearson correlation manner (as if the corresponding robust option was set to FALSE). Has no effect for Pearson correlation. See bicor.

cosineCorrelation logical: should the cosine version of the correlation calculation be used? The cosine calculation differs from the standard one in that it does not subtract the mean.

power soft-thresholding power for network construction.

networkType network type. Allowed values are (unique abbreviations of) "unsigned", "signed", "signed hybrid". See adjacency.

checkPower logical: should basic sanity check be performed on the supplied power? If you would like to experiment with unusual powers, set the argument to FALSE and proceed with caution.

TOMType one of "none", "unsigned", "signed". If "none", adjacency will be used for clustering. If "unsigned", the standard TOM will be used (more generally, TOM function will receive the adjacency as input). If "signed", TOM will keep track of the sign of correlations between neighbors.

TOMDenom a character string specifying the TOM variant to be used. Recognized values are "min" giving the standard TOM described in Zhang and Horvath (2005), and "mean" in which the min function in the denominator is replaced by mean. The "mean" may produce better results but at this time should be considered experimental.

saveIndividualTOMs logical: should individual TOMs be saved to disk for later use?

individualTOMFileNames character string giving the file names to save individual TOMs into. The following tags should be used to make the file names unique for each set and block: %s will be replaced by the set number; %N will be replaced by the set name (taken from names(multiExpr)) if it exists, otherwise by set number; %b will be replaced by the block number. If the file names turn out to be non-unique, an error will be generated.

individualTOMInfo Optional data for TOM matrices in individual data sets. This object is returned by the function blockwiseIndividualTOMs. If not given, appropriate topological overlaps will be calculated using the network construction options below.
**consensusTOM**

useIndivTOMSubset

If `individualTOMInfo` is given, this argument allows to only select a subset of the individual set networks contained in `individualTOMInfo`. It should be a numeric vector giving the indices of the individual sets to be used. Note that this argument is NOT applied to `multiExpr`.

useBlocks

optional specification of blocks that should be used for the calculations. The default is to use all blocks.

networkCalibration

network calibration method. One of "single quantile", "full quantile", "none" (or a unique abbreviation of one of them).

saveCalibratedIndividualTOMs

logical: should the calibrated individual TOMs be saved?

calibratedIndividualTOMsFilePattern

pattern of file names for saving calibrated individual TOMs.

calibrationQuantile

if `networkCalibration` is "single quantile", topological overlaps (or adjacencies if TOMs are not computed) will be scaled such that their calibrationQuantile quantiles will agree.

sampleForCalibration

if TRUE, calibration quantiles will be determined from a sample of network similarities. Note that using all data can double the memory footprint of the function and the function may fail.

sampleForCalibrationFactor

determines the number of samples for calibration: the number is `1/calibrationQuantile * sampleForCalibrationFactor`. Should be set well above 1 to ensure accuracy of the sampled quantile.

getNetworkCalibrationSamples

logical: should the sampled values used for network calibration be returned?

consensusQuantile

quantile at which consensus is to be defined. See details.

useMean

logical: should the consensus be determined from a (possibly weighted) mean across the data sets rather than a quantile?

setWeights

Optional vector (one component per input set) of weights to be used for weighted mean consensus. Only used when `useMean` above is TRUE.

saveConsensusTOMs

logical: should the consensus topological overlap matrices for each block be saved and returned?

consensusTOMsFileNames

character string containing the file names of containing the consensus topological overlaps. The tag `%b` will be replaced by the block number. If the resulting file names are non-unique (for example, because the user gives a file name without a `%b` tag), an error will be generated. These files are standard R data files and can be loaded using the `load` function.

returnTOMs

logical: should calculated consensus TOM(s) be returned?

useDiskCache

should calculated network similarities in individual sets be temporarily saved to disk? Saving to disk is somewhat slower than keeping all data in memory, but for large blocks and/or many sets the memory footprint may be too big. See chunkSize below for additional information.

chunkSize

network similarities are saved in smaller chunks of size chunkSize. If NULL, an appropriate chunk size will be determined from an estimate of available memory. Note that if the chunk size is greater than the memory required for storing intermediate results, disk cache use will automatically be disabled.
consensusTOM

cacheDir character string containing the directory into which cache files should be written. The user should make sure that the filesystem has enough free space to hold the cache files which can get quite large.

cacheBase character string containing the desired name for the cache files. The actual file names will consist of cacheBase and a suffix to make the file names unique.

nThreads non-negative integer specifying the number of parallel threads to be used by certain parts of correlation calculations. This option only has an effect on systems on which a POSIX thread library is available (which currently includes Linux and Mac OSX, but excludes Windows). If zero, the number of online processors will be used if it can be determined dynamically, otherwise correlation calculations will use 2 threads.

verbose integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.

indent indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

The function starts by optionally filtering out samples that have too many missing entries and genes that have either too many missing entries or zero variance in at least one set. Genes that are filtered out are left unassigned by the module detection. Returned eigengenes will contain NA in entries corresponding to filtered-out samples.

If `blocks` is not given and the number of genes exceeds `maxBlockSize`, genes are pre-clustered into blocks using the function `consensusProjectiveKMeans`; otherwise all genes are treated in a single block.

For each block of genes, the network is constructed and (if requested) topological overlap is calculated in each set. To minimize memory usage, calculated topological overlaps are optionally saved to disk in chunks until they are needed again for the calculation of the consensus network topological overlap.

Before calculation of the consensus Topological Overlap, individual TOMs are optionally calibrated. Calibration methods include single quantile scaling and full quantile normalization.

Single quantile scaling raises individual TOM in sets 2,3,... to a power such that the quantiles given by `calibrationQuantile` agree with the quantile in set 1. Since the high TOMs are usually the most important for module identification, the value of `calibrationQuantile` is close to (but not equal) 1. To speed up quantile calculation, the quantiles can be determined on a randomly-chosen component subset of the TOM matrices.

Full quantile normalization, implemented in `normalizeQuantiles`, adjusts the TOM matrices such that all quantiles equal each other (and equal to the quantiles of the component-wise average of the individual TOM matrices).

Note that network calibration is performed separately in each block, i.e., the normalizing transformation may differ between blocks. This is necessary to avoid manipulating a full TOM in memory.

The consensus TOM is calculated as the component-wise `consensusQuantile` quantile of the individual (set) TOMs; that is, for each gene pair (TOM entry), the `consensusQuantile` quantile across all input sets. Alternatively, one can also use (weighted) component-wise mean across all input data sets. If requested, the consensus topological overlaps are saved to disk for later use.

Value

List with the following components:
consensusTOM only present if input returnTOMs is TRUE. A list containing consensus TOM for each block, stored as a distance structure.

TOMFiles only present if input saveConsensusTOMs is TRUE. A vector of file names, one for each block, in which the TOM for the corresponding block is stored. TOM is saved as a distance structure to save space.

saveConsensusTOMs a copy of the input saveConsensusTOMs.

individualTOMInfo information about individual set TOMs. A copy of the input individualTOMInfo if given; otherwise the result of calling blockwiseIndividualTOMs. See blockwiseIndividualTOMs for details.

Further components are retained for debugging and/or convenience.

useIndivTOMSubset a copy of the input useIndivTOMSubset.

goodSamplesAndGenes a list containing information about which samples and genes are "good" in the sense that they do not contain more than a certain fraction of missing data and (for genes) have non-zero variance. See goodSamplesGenesMS for details.

nGGenes number of "good" genes in goodSamplesGenes above.

nSets number of input sets.

saveCalibratedIndividualTOMs a copy of the input saveCalibratedIndividualTOMs.

calibratedIndividualTOMFileNames if input saveCalibratedIndividualTOMs is TRUE, this component will contain the file names of calibrated individual networks. The file names are arranged in a character matrix with each row corresponding to one input set and each column to one block.

networkCalibrationSamples if input getNetworkCalibrationSamples is TRUE, a list with one component per block. Each component is in turn a list with two components: sampleIndex is a vector contain the indices of the TOM samples (the indices refer to a flattened distance structure), and TOMSamples is a matrix of TOM samples with each row corresponding to a sample in sampleIndex, and each column to one input set.

consensusQuantile a copy of the input consensusQuantile.

originCount a vector with one component per input set. When consensusQuantile equals zero, originCount contains the number of entries in the consensus TOM that come from each set (i.e., the number of times the TOM in the set was the minimum). When consensusQuantile is not zero or the "mean" consensus is used, this vector contains zeroes.

Author(s)

Peter Langfelder
References

WGCNA methodology has been described in

The original reference for the WGCNA package is

For consensus modules, see

This function uses quantile normalization described, for example, in

See Also

blockwiseIndividualTOMs for calculation of topological overlaps across multiple sets.

cor

Fast calculations of Pearson correlation.

Description

These functions implements a faster calculation of Pearson correlation.

The speedup against the R’s standard cor function will be substantial particularly if the input matrix only contains a small number of missing data. If there are no missing data, or the missing data are numerous, the speedup will be smaller but still present.

Usage

cor(x, y = NULL, use = "all.obs", method = c("pearson", "kendall", "spearman"), quick = 0, cosine = FALSE, cosineX = cosine, cosineY = cosine, drop = FALSE, nThreads = 0, verbose = 0, indent = 0)
corFast(x, y = NULL, use = "all.obs", quick = 0, nThreads = 0, verbose = 0, indent = 0)
cor1(x, use = "all.obs", verbose = 0, indent = 0)
Arguments

x a numeric vector or a matrix. If y is null, x must be a matrix.
y a numeric vector or a matrix. If not given, correlations of columns of x will be calculated.
use a character string specifying the handling of missing data. The fast calculations currently support "all.obs" and "pairwise.complete.obs"; for other options, see R's standard correlation function cor. Abbreviations are allowed.
method a character string specifying the method to be used. Fast calculations are currently available only for "pearson".
quick real number between 0 and 1 that controls the precision of handling of missing data in the calculation of correlations. See details.
cosine logical: calculate cosine correlation? Only valid for method="pearson". Cosine correlation is similar to Pearson correlation but the mean subtraction is not performed. The result is the cosine of the angle(s) between (the columns of) x and y.
cosineX logical: use the cosine calculation for x? This setting does not affect y and can be used to give a hybrid cosine-standard correlation.
cosineY logical: use the cosine calculation for y? This setting does not affect x and can be used to give a hybrid cosine-standard correlation.
drop logical: should the result be turned into a vector if it is effectively one-dimensional?
nThreads non-negative integer specifying the number of parallel threads to be used by certain parts of correlation calculations. This option only has an effect on systems on which a POSIX thread library is available (which currently includes Linux and Mac OS X, but excludes Windows). If zero, the number of online processors will be used if it can be determined dynamically, otherwise correlation calculations will use 2 threads.
verbose Controls the level of verbosity. Values above zero will cause a small amount of diagnostic messages to be printed.
indent Indentation of printed diagnostic messages. Each unit above zero adds two spaces.

Details

The fast calculations are currently implemented only for method="pearson" and use either "all.obs" or "pairwise.complete.obs". The corFast function is a wrapper that calls the function cor. If the combination of method and use is implemented by the fast calculations, the fast code is executed; otherwise, R's own correlation cor is executed.

The argument quick specifies the precision of handling of missing data. Zero will cause all calculations to be executed precisely, which may be significantly slower than calculations without missing data. Progressively higher values will speed up the calculations but introduce progressively larger errors. Without missing data, all column means and variances can be pre-calculated before the covariances are calculated. When missing data are present, exact calculations require the column means and variances to be calculated for each covariance. The approximate calculation uses the pre-calculated mean and variance and simply ignores missing data in the covariance calculation. If the number of missing data is high, the pre-calculated means and variances may be very different from the actual ones, thus potentially introducing large errors. The quick value times the number of rows specifies the maximum difference in the number of missing entries for mean and variance calculations on the one hand and covariance on the other hand that will be tolerated before a recalculation is triggered. The hope is that if only a few missing data are treated approximately, the error introduced will be small but the potential speedup can be significant.
Value

The matrix of the Pearson correlations of the columns of \( x \) with columns of \( y \) if \( y \) is given, and the correlations of the columns of \( x \) if \( y \) is not given.

Note

The implementation uses the BLAS library matrix multiplication function for the most expensive step of the calculation. Using a tuned, architecture-specific BLAS may significantly improve the performance of this function.

The values returned by the corFast function may differ from the values returned by R’s function `cor` by rounding errors on the order of 1e-15.

Author(s)

Peter Langfelder

References


See Also

R’s standard Pearson correlation function `cor`.

Examples

```r
## Test the speedup compared to standard function cor

# Generate a random matrix with 200 rows and 1000 columns

set.seed(10)
nrow = 100;
ncol = 500;
data = matrix(rnorm(nrow*ncol), nrow, ncol);

## First test: no missing data

system.time( {corStd = stats::cor(data)} );

system.time( {corFast = cor(data)} );

all.equal(corStd, corFast)

## Here R's standard correlation performs very well.

## We now add a few missing entries.
data[sample(nrow, 10), 1] = NA;

## And test the correlations again...

system.time( {corStd = stats::cor(data, use ='p')} );
```
corAndPvalue

system.time( (corFast = cor(data, use = 'p') ) );

equal(corStd, corFast)

# Here the R's standard correlation slows down considerably
# while corFast still retains its speed. Choosing
# higher ncol above will make the difference more pronounced.

---

**corAndPvalue**

*Calculation of correlations and associated p-values*

**Description**

A faster, one-step calculation of Student correlation p-values for multiple correlations, properly taking into account the actual number of observations.

**Usage**

```r
corAndPvalue(x, y = NULL, use = "pairwise.complete.obs", alternative = c("two.sided", "less", "greater"), ...)
```

**Arguments**

- `x`: a vector or a matrix
- `y`: a vector or a matrix. If `NULL`, the correlation of columns of `x` will be calculated.
- `use`: determines handling of missing data. See `cor` for details.
- `alternative`: specifies the alternative hypothesis and must be (a unique abbreviation of) one of "two.sided", "greater" or "less". The initial letter. "greater" corresponds to positive association, "less" to negative association.
- `...`: other arguments to the function `cor`.

**Details**

The function calculates correlations of a matrix or of two matrices and the corresponding Student p-values. The output is not as full-featured as `cor.test`, but can work with matrices as input.

**Value**

A list with the following components, each a matrix:

- `cor`: the calculated correlations
- `p`: the Student p-values corresponding to the calculated correlations
- `Z`: Fisher transforms of the calculated correlations
- `t`: Student t statistics of the calculated correlations
- `nObs`: Numbers of observations for the correlation, p-values etc.
Author(s)
Peter Langfelder and Steve Horvath

References

See Also
cor for calculation of correlations only;
cor.test for another function for significance test of correlations

Examples
# generate random data with non-zero correlation
set.seed(1);
a = rnorm(100);
b = rnorm(100) + a;
x = cbind(a, b);
# Call the function and display all results
corAndPvalue(x)
# Set some components to NA
x[c(1:4), 1] = NA
corAndPvalue(x)
# Note that changed number of observations.

corPredictionSuccess Qunatification of success of gene screening

Description
This function calculates the success of gene screening.

Usage
corPredictionSuccess(corPrediction, corTestSet, topNumber = 100)

Arguments
corPrediction a vector or a matrix of prediction statistics
corTestSet correlation or other statistics on test set
topNumber a vector of the number of top genes to consider

Details
For each column in corPrediction, the function evaluates the mean corTestSet for the number of top genes (ranked by the column in corPrediction) given in topNumber. The higher the mean corTestSet (for positive corPrediction) or negative (for negative corPrediction), the more successful the prediction.
Value

meancorTestSetOverall
  difference of meancorTestSetPositive and meancorTestSetNegative below
meancorTestSetPositive
  mean corTestSet on top genes with positive corPrediction
meancorTestSetNegative
  mean corTestSet on top genes with negative corPrediction

Author(s)
Steve Horvath

See Also

relativeCorPredictionSuccess

---

corPvalueFisher  

Fisher’s asymptotic p-value for correlation

Description

Calculates Fisher’s asymptotic p-value for given correlations.

Usage

corPvalueFisher(cor, nSamples, twoSided = TRUE)

Arguments

cor  
  A vector of correlation values whose corresponding p-values are to be calculated
nSamples  
  Number of samples from which the correlations were calculated
twoSided  
  logical: should the calculated p-values be two sided?

Value

A vector of p-values of the same length as the input correlations.

Author(s)

Steve Horvath and Peter Langfelder
corPvalueStudent  \hspace{1cm} \textit{Student asymptotic p-value for correlation}

\textbf{Description}
Calculates Student asymptotic p-value for given correlations.

\textbf{Usage}
corPvalueStudent(cor, nSamples)

\textbf{Arguments}
- \textit{cor}: A vector of correlation values whose corresponding p-values are to be calculated
- \textit{nSamples}: Number of samples from which the correlations were calculated

\textbf{Value}
A vector of p-values of the same length as the input correlations.

\textbf{Author(s)}
Steve Horvath and Peter Langfelder

\textbf{correlationPreservation}
\textit{Preservation of eigengene correlations}

\textbf{Description}
Calculates a summary measure of preservation of eigengene correlations across data sets.

\textbf{Usage}
correlationPreservation(multiME, setLabels, excludeGrey = TRUE, greyLabel = "grey")

\textbf{Arguments}
- \textit{multiME}: consensus module eigengenes in a multi-set format. A vector of lists with one list corresponding to each set. Each list must contain a component \textit{data} that is a data frame whose columns are consensus module eigengenes.
- \textit{setLabels}: names to be used for the sets represented in multiME.
- \textit{excludeGrey}: logical: exclude the ‘grey’ eigengene from preservation measure?
- \textit{greyLabel}: module label corresponding to the ‘grey’ module. Usually this will be the character string "grey" if the labels are colors, and the number 0 if the labels are numeric.
Details
The function calculates the preservation of correlation of each eigengene with all other eigengenes (optionally except the 'grey' eigengene) in all pairs of sets.

Value
A data frame whose rows correspond to consensus module eigengenes given in the input multiME, and columns correspond to all possible set comparisons. The two sets compared in each column are indicated in the column name.

Author(s)
Peter Langfelder

References

See Also
multiSetMEs and modulecheckSets in package moduleColor for more on eigengenes and the multi-set format

Description
The function inputs a censored time variable which is specified by two input variables time and event. It outputs i) the martingale residual and ii) deviance residual corresponding to a Cox regression model. By default, the Cox regression model is an intercept only Cox regression model. But optionally, the user can input covariates using the argument datCovariates. The function makes use of the coxph function in the survival library. See help(residuals.coxph) to learn more.

Usage
coxRegressionResiduals(time, event, datCovariates = NULL)

Arguments

time is a numeric variable that contains follow up time or time to event.

event is a binary variable that takes on values 1 and 0. 1 means that the event took place (e.g. person died, or tumor recurrent). 0 means censored, i.e. event has not yet been observed or loss to follow up.

datCovariates a data frame whose columns correspond to covariates that should be used in the Cox regression model. By default, the only covariate the intercept term 1.
Residuals are often used to investigate the lack of fit of a model. For Cox regression, there is no easy analog to the usual "observed minus predicted" residual of linear regression. Instead, several specialized residuals have been proposed for Cox regression analysis. The function calculates residuals that are well defined for an intercept only Cox regression model: the martingale and deviance residuals (Therneau et al 1990). The martingale residual of a subject (person) specifies excess failures beyond the expected baseline hazard. For example, a subject who was censored at 3 years, and whose predicted cumulative hazard at 3 years was 30. Another subject who had an event at 10 years, and whose predicted cumulative hazard at 10 years was 60. Since martingale residuals are not symmetrically distributed, even when the fitted model is correct, it is often advantageous to transform them into more symmetrically distributed residuals: deviance residuals. Thus, deviance residuals are defined as transformations of the martingale residual and the event variable. Deviance residuals are often symmetrically distributed around zero. Deviance Residuals are similar to residuals from ordinary linear regression in that they are symmetrically distributed around 0 and have standard deviation of 1.0. A subject with a large deviance residual is poorly predicted by the model, i.e. is different from the baseline cumulative hazard. A negative value indicates a longer than expected survival time. When covariates are specified in datCovariates, then one can plot deviance (or martingale) residuals against the covariates. Unusual patterns may indicate poor fit of the Cox model. Cryptic comments: Deviance (or martingale) residuals can sometimes be used as (uncensored) quantitative variables instead of the original time censored variable. For example, they could be used as outcome in a regression tree or regression forest predictor.

Value
It outputs a data frame with 2 columns. The first and second column correspond to martingale and deviance residuals respectively.

Note
This function can be considered a wrapper of the coxph function.

Author(s)
Steve Horvath

References

Examples
library(survival)
# simulate time and event data
time1=sample(1:100)
event1=sample(c(1,0), 100, replace=TRUE)

event1[1:5]=NA
time1[1:5]=NA
# no covariates
datResiduals= coxRegressionResiduals(time=time1, event=event1)

# now we simulate a covariate
z= rnorm(100)
cutreeStatic

Module detection in hierarchical dendrograms using a constant-height tree cut. Only branches whose size is at least minSize are retained.

Usage

cutreeStatic(dendro, cutHeight = 0.9, minSize = 50)

Arguments

dendro  a hierarchical clustering dendrogram such as returned by hclust.
cutHeight height at which branches are to be cut.
minSize minimum number of object on a branch to be considered a cluster.

Details

This function performs a straightforward constant-height cut as implemented by cutree, then calculates the number of objects on each branch and only keeps branches that have at least minSize objects on them.

Value

A numeric vector giving labels of objects, with 0 meaning unassigned. The largest cluster is conventionally labeled 1, the next largest 2, etc.

Author(s)

Peter Langfelder

See Also

hclust for hierarchical clustering, cutree and cutreeStatic for other constant-height branch cuts, standardColors to convert the retuned numerical lables into colors for easier visualization.
cutreeStaticColor  

*Constant height tree cut using color labels*

**Description**
Cluster detection by a constant height cut of a hierarchical clustering dendrogram.

**Usage**
```r
cutreeStaticColor(dendro, cutHeight = 0.9, minSize = 50)
```

**Arguments**
- `dendro`: a hierarchical clustering dendrogram such as returned by `hclust`.
- `cutHeight`: height at which branches are to be cut.
- `minSize`: minimum number of objects on a branch to be considered a cluster.

**Details**
This function performs a straightforward constant-height cut as implemented by `cutree`, then calculates the number of objects on each branch and only keeps branches that have at least `minSize` objects on them.

**Value**
A character vector giving color labels of objects, with "grey" meaning unassigned. The largest cluster is conventionally labeled "turquoise", next "blue" etc. Run `standardColors()` to see the sequence of standard color labels.

**Author(s)**
Peter Langfelder

**See Also**
- `hclust` for hierarchical clustering. `cutree` and `cutreeStatic` for other constant-height branch cuts, `standardColors` to see the sequence of color labels that can be assigned.

**Description**
The function plots a barplot using colors that label modules.

**Usage**
```r
displayColors(colors = NULL)
```
**Arguments**

`colors`  colors to be displayed. Defaults to all colors available for module labeling.

**Details**

To see the first \( n \) colors, use argument `colors = standardColors(n)`.

**Value**

None.

**Author(s)**

Peter Langfelder

**See Also**

`standardColors`

**Examples**

```r
displayColors(standardColors(10))
```

---

**Description**

Calculate a suitable threshold for module merging based on the number of samples and a desired Z quantile.

**Usage**

```r
dynamicMergeCut(n, mergeCor = 0.9, Zquantile = 2.35)
```

**Arguments**

- `n`  number of samples
- `mergeCor`  theoretical correlation threshold for module merging
- `Zquantile`  Z quantile for module merging

**Details**

This function calculates the threshold for module merging. The threshold is calculated as the lower boundary of the interval around the theoretical correlation `mergeCor` whose width is given by the Z value `Zquantile`.

**Value**

The correlation threshold for module merging; a single number.
empiricalBayesLM

Author(s)
Steve Horvath

See Also
moduleEigengenes, mergeCloseModules

Examples

dynamicMergeCut(20)
dynamicMergeCut(50)
dynamicMergeCut(100)

empiricalBayesLM  Empirical Bayes-moderated adjustment for unwanted covariates

Description
This function removes variation in high-dimensional data due to unwanted covariates while preserv- ing variation due to retained covariates. To prevent numerical instability, it uses Empirical bayes-moderated linear regression, optionally in a robust (outlier-resistant) form.

Usage
empiricalBayesLM(
  data,
  removedCovariates,
  retainedCovariates = NULL,
  weights = NULL,
  weightType = c("apriori", "empirical"),
  stopOnSmallWeights = TRUE,
  tol = 1e-4, maxIterations = 1000,
  scaleMeanToSamples = NULL,
  robustPriors = FALSE,
  automaticWeights = c("none", "bicov"),
  aw.maxPOutliers = 0.1)

Arguments
data  A 2-dimensional matrix or data frame of numeric data to be adjusted. Variables (for example, genes or methylation profiles) should be in columns and observations (samples) should be in rows.
retainedCovariates  A vector or two-dimensional object (matrix or data frame) giving the covariates whose effect on the data is to be retained. May be NULL if there are no such "retained" covariates.
removedCovariates  A vector or two-dimensional object (matrix or data frame) giving the covariates whose effect on the data is to be removed. At least one such covariate must be given.
weights Optional 2-dimensional matrix or data frame of the same dimensions as data giving weights for each entry in data.

weightType One of (unique abbreviations of) "apriori" or "empirical". Determines whether a standard ("apriori") or a modified ("empirical") weighted regression is used. The "apriori" choice is suitable for weights that have been determined without knowledge of the actual data, while "empirical" is appropriate for situations where one wants to down-weight certain entries of data because they may be outliers. In either case, the weights should be determined in a way that is independent of the covariates (both retained and removed).

stopOnSmallWeights Logical: should presence of small "apriori" weights trigger an error? Because standard weighted regression assumes that all weights are non-zero (otherwise estimates of standard errors will be biased), this function will by default complain about the presence of too small "apriori" weights.

tol Convergence criterion used in the numerical equation solver. When the relative change in coefficients falls below this threshold, the system will be considered to have converged.

maxIterations Maximum number of iterations to use.

scaleMeanToSamples Optional specification of samples (given as a vector of indices) to whose means the resulting adjusted data should be scaled (more precisely, shifted). If not given, the mean of all samples will be used.

robustPriors Logical: should robust priors be used? This essentially means replacing mean by median and covariance by biweight mid-covariance.

automaticWeights One of (unique abbreviations of) "none" or "bicov", instructing the function to calculate weights from the given data. Value "none" will result in trivial weights; value "bicov" will result in biweight midcovariance weights being used.

aw.maxPOutliers If automaticWeights above is "bicov", this argument gets passed to the function bicovWeights and determines the maximum proportion of outliers in calculating the weights. See bicovWeights for more details.

Details

This function uses Empirical Bayes-moderated (EB) linear regression to remove variation in data due to the variables in removedCovariates while retaining variation due to variables in retainedCovariates, if any are given. The EB step uses simple normal priors on the regression coefficients and inverse gamma priors on the variances. The procedure starts with multivariate ordinary linear regression of individual columns in data on retainedCovariates and removedCovariates. To make the coefficients comparable, columns of data are scaled to (weighted if weights are given) mean 0 and variance 1. The resulting regression coefficients are used to determine the parameters of the normal prior (mean, covariance, and inverse gamma or median and biweight mid-covariance if robust priors are used), and the variances are used to determine the parameters of the inverse gamma prior. The EB step then essentially shrinks the coefficients toward their means, with the amount of shrinkage determined by the prior covariance.

Using appropriate weights can make the data adjustment robust to outliers. This can be achieved automatically by using the argument automaticWeights = "bicov". When bicov weights are used, we also recommend setting the argument maxPOutliers to a maximum proportion of samples that could be outliers. This is especially important if some of the design variables are binary and
can be expected to have a strong effect on some of the columns in data, since standard biweight midcorrelation (and its weights) do not work well on bimodal data.

The automatic bicov weights are determined from data only. It is implicitly assumed that there are no outliers in the retained and removed covariates. Outliers in the covariates are more difficult to work with since, even if the regression is made robust to them, they can influence the adjusted values for the sample in which they appear. Unless the covariate outliers can be attributed to a relevant variation in experimental conditions, samples with covariate outliers are best removed entirely before calling this function.

**Value**

A list with the following components:

- `adjustedData` A matrix of the same dimensions as the input data, giving the adjusted data. If input data has non-NULL dimnames, these are copied.
- `residuals` A matrix of the same dimensions as the input data, giving the residuals, that is, adjusted data with zero means.
- `coefficients` A matrix of regression coefficients. Rows correspond to the design matrix variables (mean, retained and removed covariates) and columns correspond to the variables (columns) in data.
- `coefficients.scaled` A matrix of regression coefficients corresponding to columns in data scaled to mean 0 and variance 1.
- `sigmaSq` Estimated error variances (one for each column of input data).
- `sigmaSq.scaled` Estimated error variances corresponding to columns in data scaled to mean 0 and variance 1.
- `fittedValues` Fitted values calculated from the means and coefficients corresponding to the removed covariates, i.e., roughly the values that are subtracted out of the data.
- `adjustedData.OLS` A matrix of the same dimensions as the input data, giving the data adjusted by ordinary least squares. This component should only be used for diagnostic purposes, not as input for further downstream analyses, as the OLS adjustment is inferior to EB adjustment.
- `residuals.OLS` A matrix of the same dimensions as the input data, giving the residuals obtained from ordinary least squares regression, that is, OLS-adjusted data with zero means.
- `coefficients.OLS` A matrix of ordinary least squares regression coefficients. Rows correspond to the design matrix variables (mean, retained and removed covariates) and columns correspond to the variables (columns) in data.
- `coefficients.OLS.scaled` A matrix of ordinary least squares regression coefficients corresponding to columns in data scaled to mean 0 and variance 1. These coefficients are used to calculate priors for the EB step.
- `sigmaSq.OLS` Estimated OLS error variances (one for each column of input data).
- `sigmaSq.OLS.scaled` Estimated OLS error variances corresponding to columns in data scaled to mean 0 and variance 1. These are used to calculate variance priors for the EB step.
exportNetworkToCytoscape

fittedValues.OLS
OLS fitted values calculated from the means and coefficients corresponding to
the removed covariates.

weights
A matrix of weights used in the regression models. The matrix has the same
dimension as the input data.

dataColumnValid
Logical vector with one element per column of input data, indicating whether
the column was adjusted. Columns with zero variance or too many missing data
cannot be adjusted.

dataColumnWithZeroVariance
Logical vector with one element per column of input data, indicating whether
the column had zero variance.

coefficientValid
Logical matrix of the dimension (number of covariates +1) times (number of
variables in data), indicating whether the corresponding regression coefficient
is valid. Invalid regression coefficients may be returned as missing values or as
zeroes.

Author(s)
Peter Langfelder

See Also
bicovWeights for suitable weights that make the adjustment robust to outliers.

exportNetworkToCytoscape

Export network to Cytoscape

Description
This function exports a network in edge and node list files in a format suitable for importing to
Cytoscape.

Usage

eexportNetworkToCytoscape(
  adjMat,
  edgeFile = NULL,
  nodeFile = NULL,
  weighted = TRUE,
  threshold = 0.5,
  nodeNames = NULL,
  altNodeNames = NULL,
  nodeAttr = NULL,
  includeColNames = TRUE)
Arguments

adjMat   adjacency matrix giving connection strengths among the nodes in the network.
edgeFile file name of the file to contain the edge information.
nodeFile file name of the file to contain the node information.
weighted logical: should the exported network be weighted?
threshold adjacency threshold for including edges in the output.
nodeNames names of the nodes. If not given, dimnames of adjMat will be used.
altNodeNames optional alternate names for the nodes, for example gene names if nodes are labeled by probe IDs.
nodeAttr optional node attribute, for example module color. Can be a vector or a data frame.
includeColNames logical: should column names be included in the output files? Note that Cytoscape can read files both with and without column names.

Details

If the corresponding file names are supplied, the edge and node data is written to the appropriate files. The edge and node data is also returned as return value (see below).

Value

A list with the following components:

egdeData a data frame containing the edge data, with one row per edge
nodeData a data frame containing the node data, with one row per node

Author(s)

Peter Langfelder

See Also

exportNetworkToVisANT

Description

Exports network data in a format readable and displayable by the VisANT software.

Usage

exportNetworkToVisANT(adjMat, file = NULL, weighted = TRUE, threshold = 0.5, maxNConnections = NULL, probeToGene = NULL)
fixDataStructure

Arguments

adjMat: adjacency matrix of the network to be exported.
file: character string specifying the file name of the file in which the data should be written. If not given, no file will be created. The file is in a plain text format.
weighted: logical: should the exported network by weighted?
threshold: adjacency threshold for including edges in the output.
maxNConnections: maximum number of exported adjacency edges. This can be used as another filter on the exported edges.
probeToGene: optional specification of a conversion between probe names (that label columns and rows of adjacency) and gene names (that should label nodes in the output).

Details

The adjacency matrix is checked for validity. The entries can be negative, however. The adjacency matrix is expected to also have valid names or dimnames[2] that represent the probe names of the corresponding edges.

Whether the output is a weighted network or not, only edges whose (absolute value of) adjacency are above threshold will be included in the output. If maxNConnections is given, at most maxNConnections will be included in the output.

If probeToGene is given, it is expected to have two columns, the first one corresponding to the probe names, the second to their corresponding gene names that will be used in the output.

Value

A data frame containing the network information suitable as input to VisANT. The same data frame is also written into a file specified by file, if given.

Author(s)

Peter Langfelder

References

VisANT software is available from http://visant.bu.edu/.

---

fixDataStructure: Put single-set data into a form useful for multiset calculations.

Description

Encapsulates single-set data in a wrapper that makes the data suitable for functions working on multiset data collections.

Usage

fixDataStructure(data, verbose = 0, indent = 0)
Arguments

data A dataframe, matrix or array with two dimensions to be encapsulated.
verbose Controls verbosity. 0 is silent.
indent Controls indentation of printed progress messages. 0 means no indentation, every unit adds two spaces.

Details

For multiset calculations, many quantities (such as expression data, traits, module eigengenes etc) are presented by a common structure, a vector of lists (one list for each set) where each list has a component data that contains the actual (expression, trait, eigengene) data for the corresponding set in the form of a dataframe. This function creates a vector of lists of length 1 and fills the component data with the content of parameter data.

Value

As described above, input data in a format suitable for functions operating on multiset data collections.

Author(s)

Peter Langfelder, <Peter.Langfelder@gmail.com>

See Also

checkSets

Examples

```r
singleSetData = matrix(rnorm(100), 10, 10);
encapsData = fixDataStructure(singleSetData);
length(encapsData)
names(encapsData[[1]])
dim(encapsData[[1]]$data)
all.equal(encapsData[[1]]$data, singleSetData);
```

---

formatLabels **Break long character strings into multiple lines**

Description

This function attempts to break long character strings into multiple lines by replacing a given pattern by a newline character.

Usage

```r
formatLabels(labels,
maxCharPerLine = 14,
split = " ", fixed = TRUE,
newsplit = split, keepSplitAtEOL = TRUE)
```
**Arguments**

- **labels**: Character strings to be formatted.
- **maxCharPerLine**: Integer giving the maximum number of characters per line.
- **split**: Pattern to be replaced by newline (\n) characters.
- **fixed**: Logical: Should the pattern be interpreted literally (TRUE) or as a regular expression (FALSE)? See `strsplit` and its argument `fixed`.
- **newsplit**: Character string to replace the occurrences of `split` above with.
- **keepSplitAtEOL**: When replacing an occurrence of `split` with a newline character, should the `newsplit` be added before the newline as well?

**Details**

Each given element of `labels` is processed independently. The character string is split using `strsplit`, with `split` as the splitting pattern. The resulting shorter character strings are then concatenated together with `newsplit` as the separator. Whenever the length of the combined result from the start or the previous newline character exceeds `maxCharPerLine`, a newline character is inserted (at the previous split).

Note that individual segments (i.e., sections of the input between occurrences of `split`) whose number of characters exceeds `maxCharPerLine` will not be split.

**Value**

A character vector of the same length as input `labels`.

**Author(s)**

Peter Langfelder

**Examples**

```r
s = "A quick hare jumps over the brown fox";
formatLabels(s);
```

**Usage**

```r
fundamentalNetworkConcepts(adj, GS = NULL)
```
fundamentalNetworkConcepts

Arguments

adj an adjacency matrix, that is a square, symmetric matrix with entries between 0 and 1
GS a node significance measure: a vector of the same length as the number of rows (and columns) of the adjacency matrix.

Value

A list with the following components:

Connectivity a numerical vector that reports the connectivity (also known as degree) of each node. This fundamental network concept is also known as whole network connectivity. One can also define the scaled connectivity $k = \frac{\text{Connectivity}}{\max(\text{Connectivity})}$ which is used for computing the hub gene significance.

ScaledConnectivity the Connectivity vector scaled by the highest connectivity in the network, i.e., $\frac{\text{Connectivity}}{\max(\text{Connectivity})}$.

ClusterCoef a numerical vector that reports the cluster coefficient for each node. This fundamental network concept measures the cliquishness of each node.

MAR a numerical vector that reports the maximum adjacency ratio for each node. $\text{MAR}[i]$ equals 1 if all non-zero adjacencies between node $i$ and the remaining network nodes equal 1. This fundamental network concept is always 1 for nodes of an unweighted network. This is a useful measure for weighted networks since it allows one to determine whether a node has high connectivity because of many weak connections (small MAR) or because of strong (but few) connections (high MAR), see Horvath and Dong 2008.

Density the density of the network.

Centralization the centralization of the network.

Heterogeneity the heterogeneity of the network.

Author(s)

Steve Horvath

References


See Also

conformityBasedNetworkConcepts for calculation of conformity based network concepts for a network adjacency matrix;

networkConcepts, for calculation of conformity based and eigennode based network concepts for a correlation network.
GOenrichmentAnalysis

Calculation of GO enrichment (experimental)

Description

WARNING: This function should be considered experimental. The arguments and resulting values (in particular, the enrichment p-values) are not yet finalized and may change in the future. The function should only be used to get a quick and rough overview of GO enrichment in the modules in a data set; for a publication-quality analysis, please use an established tool.

Using Bioconductor’s annotation packages, this function calculates enrichments and returns terms with best enrichment values.

Usage

GOenrichmentAnalysis(labels, entrezCodes, yeastORFs = NULL, organism = "human", ontologies = c("BP", "CC", "MF"), evidence = "all", includeOffspring = TRUE, backgroundType = "givenInGO", removeDuplicates = TRUE, leaveOutLabel = NULL, nBestP = 10, pCut = NULL, nBiggest = 0, getTermDetails = TRUE, verbose = 2, indent = 0)

Arguments

labels cluster (module, group) labels of genes to be analyzed. Either a single vector, or a matrix. In the matrix case, each column will be analyzed separately; analyzing a collection of module assignments in one function call will be faster than calling the function several times. For each row, the labels in all columns must correspond to the same gene specified in entrezCodes.

entrezCodes Entrez (a.k.a. LocusLink) codes of the genes whose labels are given in labels. A single vector; the i-th entry corresponds to row i of the matrix labels (or to the i-the entry if labels is a vector).

yeastORFs if organism="yeast" (below), this argument can be used to input yeast open reading frame (ORF) identifiers instead of Entrez codes. Since the GO mappings for yeast are provided in terms of ORF identifiers, this may lead to a more accurate GO enrichment analysis. If given, the argument entrezCodes is ignored.

organism character string specifying the organism for which to perform the analysis. Recognized values are (unique abbreviations of) "human", "mouse", "rat", "malaria", "yeast", "

ontologies vector of character strings specifying GO ontologies to be included in the analysis. Can be any subset of "BP", "CC", "MF". The result will contain the terms with highest enrichment in each specified category, plus a separate list of terms with best enrichment in all ontologies combined.
**evidence**

vector of character strings specifying admissible evidence for each gene in its specific term, or "all" for all evidence codes. See Details or http://www.geneontology.org/GO.evidence.shtml for available evidence codes and their meaning.

**includeOffspring**

logical: should genes belonging to the offspring of each term be included in the term? As a default, only genes belonging directly to each term are associated with the term. Note that the calculation of enrichments with offspring included can be quite slow for large data sets.

**backgroundType**

specification of the background to use. Recognized values are (unique abbreviations of) "allGiven", "allInGO", "givenInGO", meaning that the functions will take all genes given in labels as background ("allGiven"), all genes present in any of the GO categories ("allInGO"), or the intersection of given genes and genes present in GO ("givenInGO"). The default is recommended for genome-wide enrichment studies.

**removeDuplicates**

logical: should duplicate entries in entrezCodes be removed? If TRUE, only the first occurrence of each unique Entrez code will be kept. The cluster labels labels will be adjusted accordingly.

**leaveOutLabel**

optional specifications of module labels for which enrichment calculation is not desired. Can be a single label or a vector of labels to be ignored. However, if in any of the sets no labels are left to calculate enrichment of, the function will stop with an error.

**nBestP**

specifies the number of terms with highest enrichment whose detailed information will be returned.

**pCut**

alternative specification of terms to be returned: all terms whose enrichment p-value is more significant than pCut will be returned. If pCut is given, nBestP is ignored.

**nBiggest**

in addition to returning terms with highest enrichment, terms that contain most of the genes in each cluster can be returned by specifying the number of biggest terms per cluster to be returned. This may be useful for development and testing purposes.

**getTermDetails**

logical indicating whether detailed information on the most enriched terms should be returned.

**verbose**

integer specifying the verbosity of the function. Zero means silent, positive values will cause the function to print progress reports.

**indent**

integer specifying indentation of the diagnostic messages. Zero means no indentation, each unit adds two spaces.

**Details**

This function is basically a wrapper for the annotation packages available from Bioconductor. It requires the packages GO.db, AnnotationDbi, and org.xx.eg.db, where xx is the code corresponding to the organism that the user wishes to analyze (e.g., Hs for human Homo Sapiens, Mm for mouse Mus Musculus etc). For each cluster specified in the input, the function calculates all enrichments in the specified ontologies, and collects information about the terms with highest enrichment. The enrichment p-value is calculated using Fisher exact test. As background we use all of the supplied genes that are present in at least one term in GO (in any of the ontologies).

For best results, the newest annotation libraries should be used. Because of the way Bioconductor is set up, to get the newest annotation libraries you may have to use the current version of R.
According to http://www.geneontology.org/GO.evidence.shtml, the following codes are used by GO:

**Experimental Evidence Codes**
- EXP: Inferred from Experiment
- IDA: Inferred from Direct Assay
- IPI: Inferred from Physical Interaction
- IMP: Inferred from Mutant Phenotype
- IGI: Inferred from Genetic Interaction
- IEP: Inferred from Expression Pattern

**Computational Analysis Evidence Codes**
- ISS: Inferred from Sequence or Structural Similarity
- ISO: Inferred from Sequence Orthology
- ISA: Inferred from Sequence Alignment
- ISM: Inferred from Sequence Model
- IGC: Inferred from Genomic Context
- IBA: Inferred from Biological aspect of Ancestor
- IBD: Inferred from Biological aspect of Descendant
- IKR: Inferred from Key Residues
- IDR: Inferred from Rapid Divergence
- RCA: inferred from Reviewed Computational Analysis

**Author Statement Evidence Codes**
- TAS: Traceable Author Statement
- NAS: Non-traceable Author Statement

**Curator Statement Evidence Codes**
- IC: Inferred by Curator
- ND: No biological Data available

**Automatically-assigned Evidence Codes**
- IEA: Inferred from Electronic Annotation

**Obsolete Evidence Codes**
- NR: Not Recorded

**Value**

A list with the following components:

- **keptForAnalysis**
  logical vector with one entry per given gene. TRUE if the entry was used for enrichment analysis. Depending on the setting of removeDuplicates above, only a single entry per gene may be used.

- **inGO**
  logical vector with one entry per given gene. TRUE if the gene belongs to any GO term, FALSE otherwise. Also FALSE for genes not used for the analysis because of duplication.

If input labels contained only one vector of labels, the following components:

- **countsInTerms**
  a matrix whose rows correspond to given cluster, and whose columns correspond to GO terms, containing number of genes in the intersection of the corresponding module and GO term. Row and column names are set appropriately.
enrichmentP  a matrix whose rows correspond to given cluster, and whose columns correspond to GO terms, containing enrichment p-values of each term in each cluster. Row and column names are set appropriately.

bestPTerms  a list of lists with each inner list corresponding to an ontology given in ontologies in input, plus one component corresponding to all given ontologies combined. The name of each component is set appropriately. Each inner list contains two components: enrichment is a data frame containing the highest enriched terms for each module; and forModule is a list of lists with one inner list per module, appropriately named. Each inner list contains one component per term. If input getTermDetails is TRUE, this component is yet another list and contains components termName (term name), enrichmentP (enrichment P value), termDefinition (GO term definition), termOntology (GO term ontology), geneCodes (Entrez codes of module genes in this term), genePositions (indices of the genes listed in geneCodes within the given labels). Thus, to obtain information on say the second term of the 5th module in ontology BP, one can look at the appropriate row of bestPTerms\$BP\$enrichment, or one can reference bestPTerms\$BP\$forModule[[5]][[2]]. The author of the function apologizes for any confusion this structure of the output may cause.

biggestTerms  a list of the same format as bestPTerms, containing information about the terms with most genes in the module for each supplied ontology.

If input labels contained more than one vector, instead of the above components the return value contains a list named setResults that has one component per given set; each component is a list containing the above components for the corresponding set.

Author(s)
Peter Langfelder

See Also
Bioconductor’s annotation packages such as GO.db and organism-specific annotation packages such as org.Hs.eg.db.

goodGenes  Filter genes with too many missing entries

Description
This function checks data for missing entries and returns a list of genes that have non-zero variance and pass two criteria on maximum number of missing values: the fraction of missing values must be below a given threshold and the total number of missing samples must be below a given threshold.

Usage
goodGenes(datExpr, 
  useSamples = NULL, 
  useGenes = NULL, 
  minFraction = 1/2, 
  minNSamples = .minNSamples, 
  minNGenes = .minNGenes, 
  tol = NULL, 
  verbose = 1, indent = 0)
goodGenes

Arguments

datExpr  expression data. A data frame in which columns are genes and rows are samples.
useSamples  optional specifications of which samples to use for the check. Should be a logical vector; samples whose entries are FALSE will be ignored for the missing value counts. Defaults to using all samples.
useGenes  optional specifications of genes for which to perform the check. Should be a logical vector; genes whose entries are FALSE will be ignored. Defaults to using all genes.
minFraction  minimum fraction of non-missing samples for a gene to be considered good.
minNSamples  minimum number of non-missing samples for a gene to be considered good.
minNGenes  minimum number of good genes for the data set to be considered fit for analysis. If the actual number of good genes falls below this threshold, an error will be issued.
tol  an optional 'small' number to compare the variance against. Defaults to the square of 1e-10 * max(abs(datExpr), na.rm = TRUE). The reason of comparing the variance to this number, rather than zero, is that the fast way of computing variance used by this function sometimes causes small numerical overflow errors which make variance of constant vectors slightly non-zero; comparing the variance to tol rather than zero prevents the retaining of such genes as 'good genes'.
verbose  integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
indent  indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

The constants .minNSamples and .minNGenes are both set to the value 4. For most data sets, the fraction of missing samples criterion will be much more stringent than the absolute number of missing samples criterion.

Value

A logical vector with one entry per gene that is TRUE if the gene is considered good and FALSE otherwise. Note that all genes excluded by useGenes are automatically assigned FALSE.

Author(s)

Peter Langfelder and Steve Horvath

See Also

goodSamples, goodSamplesGenes
goodGenesMS

Filter genes with too many missing entries across multiple sets

Description
This function checks data for missing entries and returns a list of genes that have non-zero variance in all sets and pass two criteria on maximum number of missing values in each given set: the fraction of missing values must be below a given threshold and the total number of missing samples must be below a given threshold.

Usage

\[
goodGenesMS(multiExpr, 
  useSamples = \text{NULL}, 
  useGenes = \text{NULL}, 
  minFraction = 1/2, 
  minNSamples = \ldots \text{minNSamples}, 
  minNGenes = \ldots \text{minNGenes}, 
  tol = \text{NULL}, 
  verbose = 1, indent = 0)
\]

Arguments

- **multiExpr**: expression data in the multi-set format (see checkSets). A vector of lists, one per set. Each set must contain a component `data` that contains the expression data, with rows corresponding to samples and columns to genes or probes.

- **useSamples**: optional specifications of which samples to use for the check. Should be a logical vector; samples whose entries are `FALSE` will be ignored for the missing value counts. Defaults to using all samples.

- **useGenes**: optional specifications of genes for which to perform the check. Should be a logical vector; genes whose entries are `FALSE` will be ignored. Defaults to using all genes.

- **minFraction**: minimum fraction of non-missing samples for a gene to be considered good.

- **minNSamples**: minimum number of non-missing samples for a gene to be considered good.

- **minNGenes**: minimum number of good genes for the data set to be considered fit for analysis. If the actual number of good genes falls below this threshold, an error will be issued.

- **tol**: an optional 'small' number to compare the variance against. For each set in `multiExpr`, the default value is `1e-10 \times \max(abs(multiExpr[[set]]$data), na.rm = TRUE)`. The reason of comparing the variance to this number, rather than zero, is that the fast way of computing variance used by this function sometimes causes small numerical overflow errors which make variance of constant vectors slightly non-zero; comparing the variance to `tol` rather than zero prevents the retaining of such genes as 'good genes'.

- **verbose**: integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.

- **indent**: indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.
**goodSamples**

**Details**

The constants `.minNSamples` and `.minNGenes` are both set to the value 4. For most data sets, the fraction of missing samples criterion will be much more stringent than the absolute number of missing samples criterion.

**Value**

A logical vector with one entry per gene that is `true` if the gene is considered good and `false` otherwise. Note that all genes excluded by `useGenes` are automatically assigned `false`.

**Author(s)**

Peter Langfelder

**See Also**

goodGenes, goodSamples, goodSamplesGenes for cleaning individual sets separately;
goodSamplesMS, goodSamplesGenesMS for additional cleaning of multiple data sets together.

---

**Description**

This function checks data for missing entries and returns a list of samples that pass two criteria on maximum number of missing values: the fraction of missing values must be below a given threshold and the total number of missing genes must be below a given threshold.

**Usage**

```r
goodSamples(datExpr, 
    useSamples = NULL, 
    useGenes = NULL, 
    minFraction = 1/2, 
    minNSamples = .minNSamples, 
    minNGenes = .minNGenes, 
    verbose = 1, indent = 0)
```

**Arguments**

- `datExpr` expression data. A data frame in which columns are genes and rows ar samples.
- `useSamples` optional specifications of which samples to use for the check. Should be a logical vector; samples whose entries are `FALSE` will be ignored for the missing value counts. Defaults to using all samples.
- `useGenes` optional specifications of genes for which to perform the check. Should be a logical vector; genes whose entries are `FALSE` will be ignored. Defaults to using all genes.
- `minFraction` minimum fraction of non-missing samples for a gene to be considered good.
minNNSamples  minimum number of good samples for the data set to be considered fit for analysis. If the actual number of good samples falls below this threshold, an error will be issued.

minNNGenes  minimum number of non-missing samples for a sample to be considered good.

verbose  integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.

indent  indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

The constants \texttt{minNNSamples} and \texttt{minNNGenes} are both set to the value 4. For most data sets, the fraction of missing samples criterion will be much more stringent than the absolute number of missing samples criterion.

Value

A logical vector with one entry per sample that is \texttt{TRUE} if the sample is considered good and \texttt{FALSE} otherwise. Note that all samples excluded by \texttt{useSamples} are automatically assigned \texttt{FALSE}.

Author(s)

Peter Langfelder and Steve Horvath

See Also

\texttt{goodsamples}, \texttt{goodsamplesgenes}

---

\texttt{goodsamplesgenes}  \textit{Iterative filtering of samples and genes with too many missing entries}

Description

This function checks data for missing entries and zero-variance genes, and returns a list of samples and genes that pass criteria maximum number of missing values. If necessary, the filtering is iterated.

Usage

\begin{verbatim}
  goodsamplesgenes(
    datExpr, 
    minFraction = 1/2, 
    minNNSamples = \texttt{.minNNSamples}, 
    minNNGenes = \texttt{.minNNGenes}, 
    tol = NULL, 
    verbose = 1, indent = 0)
\end{verbatim}
### Arguments

- **datExpr**: expression data. A data frame in which columns are genes and rows are samples.
- **minFraction**: minimum fraction of non-missing samples for a gene to be considered good.
- **minNSamples**: minimum number of non-missing samples for a gene to be considered good.
- **minNGenes**: minimum number of good genes for the data set to be considered fit for analysis. If the actual number of good genes falls below this threshold, an error will be issued.
- **tol**: an optional 'small' number to compare the variance against. Defaults to the square of \(1e-10 \times \text{max(abs(datExpr), na.rm = TRUE)}\). The reason of comparing the variance to this number, rather than zero, is that the fast way of computing variance used by this function sometimes causes small numerical overflow errors which make variance of constant vectors slightly non-zero; comparing the variance to tol rather than zero prevents the retaining of such genes as 'good genes'.
- **verbose**: integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
- **indent**: indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

### Details

This function iteratively identifies samples and genes with too many missing entries and genes with zero variance. Iterations may be required since excluding samples effectively changes criteria on genes and vice versa. The process is repeated until the lists of good samples and genes are stable. The constants .minNSamples and .minNGenes are both set to the value 4.

### Value

A list with the following components:

- **goodSamples**: A logical vector with one entry per sample that is TRUE if the sample is considered good and FALSE otherwise.
- **goodGenes**: A logical vector with one entry per gene that is TRUE if the gene is considered good and FALSE otherwise.

### Author(s)

Peter Langfelder

### See Also

- goodSamples
- goodGenes
goodSamplesGenesMS

Iterative filtering of samples and genes with too many missing entries across multiple data sets

Description

This function checks data for missing entries and zero variance across multiple data sets and returns a list of samples and genes that pass criteria: maximum number of missing values. If necessary, the filtering is iterated.

Usage

goodSamplesGenesMS(
  multiExpr,
  minFraction = 1/2,
  minNSamples = minNSamples,
  minNGenes = minNGenes,
  tol = NULL,
  verbose = 2, indent = 0)

Arguments

- **multiExpr**: expression data in the multi-set format (see checkSets). A vector of lists, one per set. Each set must contain a component data that contains the expression data, with rows corresponding to samples and columns to genes or probes.
- **minFraction**: minimum fraction of non-missing samples for a gene to be considered good.
- **minNSamples**: minimum number of non-missing samples for a gene to be considered good.
- **minNGenes**: minimum number of good genes for the data set to be considered fit for analysis. If the actual number of good genes falls below this threshold, an error will be issued.
- **tol**: an optional 'small' number to compare the variance against. For each set in multiExpr, the default value is \(1e-10 \times \max(abs(multiExpr[[\text{set}]] \cdot \text{data}), \text{na.rm} = \text{TRUE})\). The reason of comparing the variance to this number, rather than zero, is that the fast way of computing variance used by this function sometimes causes small numerical overflow errors which make variance of constant vectors slightly non-zero; comparing the variance to tol rather than zero prevents the retaining of such genes as 'good genes'.
- **verbose**: integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
- **indent**: indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

This function iteratively identifies samples and genes with too many missing entries, and genes with zero variance. Iterations may be required since excluding samples effectively changes criteria on genes and vice versa. The process is repeated until the lists of good samples and genes are stable. The constants minNSamples and minNGenes are both set to the value 4.
Value

A list with the following components:

`goodSamples`  A list with one component per given set. Each component is a logical vector with one entry per sample in the corresponding set that is `TRUE` if the sample is considered good and `FALSE` otherwise.

`goodGenes`  A logical vector with one entry per gene that is `TRUE` if the gene is considered good and `FALSE` otherwise.

Author(s)

Peter Langfelder

See Also

`goodGenes`, `goodSamples`, `goodSamplesGenes` for cleaning individual sets separately;
`goodSamplesMS`, `goodGenesMS` for additional cleaning of multiple data sets together.

Description

This function checks data for missing entries and returns a list of samples that pass two criteria on maximum number of missing values: the fraction of missing values must be below a given threshold and the total number of missing genes must be below a given threshold.

Usage

```r
goodSamplesMS(multiExpr, 
useSamples = NULL, 
useGenes = NULL, 
minFraction = 1/2, 
minNSamples = ..minNSamples, 
minNGenes = ..minNGenes, 
verbose = 1, indent = 0)
```

Arguments

- `multiExpr`  expression data in the multi-set format (see `checkSets`). A vector of lists, one per set. Each set must contain a component `data` that contains the expression data, with rows corresponding to samples and columns to genes or probes.
- `useSamples`  optional specifications of which samples to use for the check. Should be a logical vector; samples whose entries are `FALSE` will be ignored for the missing value counts. Defaults to using all samples.
- `useGenes`  optional specifications of genes for which to perform the check. Should be a logical vector; genes whose entries are `FALSE` will be ignored. Defaults to using all genes.
- `minFraction`  minimum fraction of non-missing samples for a gene to be considered good.
minNSamples  minimum number of good samples for the data set to be considered fit for analysis. If the actual number of good samples falls below this threshold, an error will be issued.

minNGenes  minimum number of non-missing samples for a sample to be considered good.

verbose  integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.

indent  indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

The constants `minNSamples` and `minNGenes` are both set to the value 4. For most data sets, the fraction of missing samples criterion will be much more stringent than the absolute number of missing samples criterion.

Value

A list with one component per input set. Each component is a logical vector with one entry per sample in the corresponding set, indicating whether the sample passed the missing value criteria.

Author(s)

Peter Langfelder and Steve Horvath

See Also

goodGenes, goodSamples, goodSamplesGenes for cleaning individual sets separately;
goodGenesMS, goodSamplesGenesMS for additional cleaning of multiple data sets together.

greenBlackRed  Green-black-red color sequence

Description

Generate a green-black-red color sequence of a given length.

Usage

greenBlackRed(n, gamma = 1)

Arguments

n  number of colors to be returned

gamma  color correction power

Details

The function returns a color vector that starts with pure green, gradually turns into black and then to red. The power `gamma` can be used to control the behaviour of the quarter- and three quarter-values (between green and black, and black and red, respectively). Higher powers will make the mid-colors more green and red, respectively.
**greenWhiteRed**

Value

A vector of colors of length \( n \).

Author(s)

Peter Langfelder

Examples

```r
par(mfrow = c(3, 1))
displayColors(greenBlackRed(50));
displayColors(greenBlackRed(50, 2));
displayColors(greenBlackRed(50, 0.5));
```

<table>
<thead>
<tr>
<th>greenWhiteRed</th>
<th>Green-white-red color sequence</th>
</tr>
</thead>
</table>

Description

Generate a green-white-red color sequence of a given length.

Usage

`greenWhiteRed(n, gamma = 1, warn = TRUE)`

Arguments

- \( n \): number of colors to be returned
- \( gamma \): color change power
- \( warn \): logical: should the user be warned that this function produces a palette unsuitable for people with most common color blindness?

Details

The function returns a color vector that starts with green, gradually turns into white and then to red. The power \( gamma \) can be used to control the behaviour of the quarter- and three quarter-values (between green and white, and white and red, respectively). Higher powers will make the mid-colors more white, while lower powers will make the colors more saturated, respectively.

Typical use of this function is to produce (via function `numbers2colors`) a color representation of numbers within a symmetric interval around 0, for example, the interval \([-1, 1]\). Note though that since green and red are not distinguishable by people with the most common type of color blindness, we recommend using the analogous palette returned by the function `blueWhiteRed`.

Value

A vector of colors of length \( n \).

Author(s)

Peter Langfelder
See Also

blueWhiteRed for a color sequence more friendly to people with the most common type of color blindness;
numbers2colors for a function that produces a color representation for continuous numbers.

Examples

par(mfrow = c(3, 1))
displayColors(greenWhiteRed(50));
title("gamma = 1")
displayColors(greenWhiteRed(50, 3));
title("gamma = 3")
displayColors(greenWhiteRed(50, 0.5));
title("gamma = 0.5")

---

GTOMdist  Generalized Topological Overlap Measure

Description

Generalized Topological Overlap Measure, taking into account interactions of higher degree.

Usage

GTOMdist(adjMat, degree = 1)

Arguments

adjMat  adjacency matrix. See details below.
degree  integer specifying the maximum degree to be calculated.

Value

Matrix of the same dimension as the input adjMat.

Author(s)

Steve Horvath and Andy Yip

References

**hubGeneSignificance**  

**Hubgene significance**

**Description**

Calculate approximate hub gene significance for all modules in network.

**Usage**

```r
hubGeneSignificance(datKME, GS)
```

**Arguments**

- `datKME`: a data frame (or a matrix-like object) containing eigengene-based connectivities of all genes in the network.
- `GS`: a vector with one entry for every gene containing its gene significance.

**Details**

In `datKME` rows correspond to genes and columns to modules.

**Value**

A vector whose entries are the hub gene significances for each module.

**Author(s)**

Steve Horvath

**References**


---

**ImmunePathwayLists**  

**Immune Pathways with Corresponding Gene Markers**

**Description**

This matrix gives a predefined set of marker genes for many immune response pathways, as assembled by Brian Modena (a member of Daniel R Salomon’s lab at Scripps Research Institute), and colleagues. It is used with userListEnrichment to search user-defined gene lists for enrichment.

**Usage**

```r
data(ImmunePathwayLists)
```
Format

A 3597 x 2 matrix of characters containing Gene / Category pairs. The first column (Gene) lists genes corresponding to a given category (second column). Each Category entry is of the form `<Immune Pathway>__ImmunePathway`. Note that the matrix is sorted first by Category and then by Gene, such that all genes related to the same category are listed sequentially.

Source

For more information about this list, please see `userListEnrichment`.

Examples

data(ImmunePathwayLists)
head(ImmunePathwayLists)

Description

These functions provide an inline display of progress.

Usage

`initProgInd(leadStr = "..", trailStr = "", quiet = !interactive())`
`updateProgInd(newFrac, progInd, quiet = !interactive())`

Arguments

- `leadStr` character string that will be printed before the actual progress number.
- `trailStr` character string that will be printed after the actual progress number.
- `quiet` can be used to silence the indicator for non-interactive sessions whose output is typically redirected to a file.
- `newFrac` new fraction of progress to be displayed.
- `progInd` an object of class `progressIndicator` that encodes previously printed message.

Details

A progress indicator is a simple inline display of progress intended to satisfy impatient users during lengthy operations. The function `initProgInd` initializes a progress indicator (at zero); `updateProgInd` updates it to a specified fraction.

Note that excessive use of `updateProgInd` may lead to a performance penalty (see examples).

Value

Both functions return an object of class `progressIndicator` that holds information on the last printed value and should be used for subsequent updates of the indicator.
intramodularConnectivity

Author(s)

Peter Langfelder

Examples

```r
max = 10;
prog = initProgInd("Counting: ", "done");
for (c in 1:max)
{
    Sys.sleep(0.10);
    prog = updateProgInd(c/max, prog);
}
printFlush("");

printFlush("Example 2:");
prog = initProgInd();
for (c in 1:max)
{
    Sys.sleep(0.10);
    prog = updateProgInd(c/max, prog);
}

## Example of a significant slowdown:
## Without progress indicator:

system.time( { a = 0; for (i in 1:10000) a = a+i; } )

## With progress indicator, some 50 times slower:

system.time(
{
    prog = initProgInd("Counting: ", "done");
    a = 0;
    for (i in 1:10000)
    {
        a = a+i;
        prog = updateProgInd(i/10000, prog);
    }
})
```

Description

Calculates intramodular connectivity, i.e., connectivity of nodes to other nodes within the same module.
intramodularConnectivity(adjMat, colors, scaleByMax = FALSE)

intramodularConnectivity_fromExpr(datExpr, colors, 
corfnc = "cor", corOptions = "use = 'p'", 
distFnc = "dist", distOptions = "method = 'euclidean'", 
networkType = "unsigned", power = if (networkType=="distance") 1 else 6, 
scaleByMax = FALSE, 
ignoreColors = if (is.numeric(colors)) 0 else "grey", 
getWholeNetworkConnectivity = TRUE)

Arguments

adjMat adjacency matrix, a square, symmetric matrix with entries between 0 and 1.
colors module labels. A vector of length ncol(adjMat) giving a module label for each gene (node) of the network.
scaleByMax logical: should intramodular connectivities be scaled by the maximum IM connectivity in each module?
datExpr data frame containing expression data. Columns correspond to genes and rows to samples.
corfnc character string specifying the function to be used to calculate co-expression similarity for correlation networks. Defaults to Pearson correlation. Any function returning values between -1 and 1 can be used.
corOptions character string specifying additional arguments to be passed to the function given by corFnc. Use "use = 'p', method = 'spearman'" to obtain Spearman correlation.
distFnc character string specifying the function to be used to calculate co-expression similarity for distance networks. Defaults to the function dist. Any function returning non-negative values can be used.
distOptions character string specifying additional arguments to be passed to the function given by distFnc. For example, when the function dist is used, the argument method can be used to specify various ways of computing the distance.
networkType network type. Allowed values are (unique abbreviations of) "unsigned", "signed", "signed hybrid", "distance".
power soft thresholding power.
ignoreColors level(s) of colors that identifies unassigned genes. The intramodular connectivity in this "module" will not be calculated.
getWholeNetworkConnectivity logical: should whole-network connectivity be computed as well? For large networks, this can be quite time-consuming.

Details

The module labels can be numeric or character. For each node (gene), the function sums adjacency entries (excluding the diagonal) to other nodes within the same module. Optionally, the connectivities can be scaled by the maximum connectivity in each module.
isMultiData

Value

If input getWholeNetworkConnectivity is TRUE, a data frame with 4 columns giving the total connectivity, intramodular connectivity, extra-modular connectivity, and the difference of the intra- and extra-modular connectivities for all genes; otherwise a vector of intramodular connectivities.

Author(s)

Steve Horvath and Peter Langfelder

References


See Also

adjacency

isMultiData

Determine whether the supplied object is a valid multiData structure

Description

Attempts to determine whether the supplied object is a valid multiData structure (see Details).

Usage

isMultiData(x, strict = TRUE)

Arguments

x
An object.

strict
Logical: should the structure of multiData be checked for "strict" compliance?

Details

A multiData structure is intended to store (the same type of) data for multiple, possibly independent, realizations (for example, expression data for several independent experiments). It is a list where each component corresponds to an (independent) data set. Each component is in turn a list that can hold various types of information but must have a data component. In a "strict" multiData structure, the data components are required to each be a matrix or a data frame and have the same number of columns. In a "loose" multiData structure, the data components can be anything (but for most purposes should be of comparable type and content).

This function checks whether the supplied x is a multiData structure in the "strict" (when strict = TRUE or "loose" strict = FALSE) sense.

Value

Logical: TRUE if the input x is a multiData structure, FALSE otherwise.
keepCommonProbes

Author(s)

Peter Langfelder

See Also

Other multiData handling functions whose names start with mtd.

---

keepCommonProbes  Keep probes that are shared among given data sets

Description

This function strips out probes that are not shared by all given data sets, and orders the remaining common probes using the same order in all sets.

Usage

keepCommonProbes(multiExpr, orderBy = 1)

Arguments

multiExpr  expression data in the multi-set format (see checkSets). A vector of lists, one per set. Each set must contain a component data that contains the expression data, with rows corresponding to samples and columns to genes or probes.

orderBy  index of the set by which probes are to be ordered.

Value

Expression data in the same format as the input data, containing only common probes.

Author(s)

Peter Langfelder

See Also

checkSets
**kMEcomparissonScatterplot**

*Function to plot kME values between two comparable data sets.*

**Description**

Plots the kME values of genes in two groups of expression data for each module in an inputted color vector.

**Usage**

```r
kMEcomparissonScatterplot(
    datExpr1, datExpr2, colorh,
    inA = NULL, inB = NULL, MEsA = NULL, MEsB = NULL,
    nameA = "A", nameB = "B",
    plotAll = FALSE, noGrey = TRUE, maxPlot = 1000, pch = 19,
    fileName = if (plotAll) paste("kME_correlations_between_",nameA,"_and_",
    nameB,"_.all.pdf",sep="")) else
    paste("kME_correlations_between_",nameA,"_and_",
    nameB,"_.inMod.pdf",sep=""), ...)
```

**Arguments**

- **datExpr1**
  The first expression matrix (samples=rows, genes=columns). This can either include only the data for group A (in which case datExpr2 must be entered), or can contain all of the data for groups A and B (in which case inA and inB must be entered).

- **datExpr2**
  The second expression matrix, or set to NULL if all data is from same expression matrix. If entered, datExpr2 must contain the same genes as datExpr1 in the same order.

- **colorh**
  The common color vector (module labels) corresponding to both sets of expression data.

- **inA, inB**
  Vectors of TRUE/FALSE indicating whether a sample is in group A/B, or a vector of numeric indices indicating which samples are in group A/B. If datExpr2 is entered, these inputs are ignored (thus default = NULL). For these and all other A/B inputs, "A" corresponds to datExpr1 and "B" corresponds to datExpr2 if datExpr2 is entered; otherwise "A" corresponds to datExpr1[inA,] while "B" corresponds to datExpr1[inB,].

- **MEsA, MEsB**
  Either the module eigengenes or NULL (default) in which case the module eigengenes will be calculated. In inputted, MEs MUST be calculated using "moduleEigengenes(<parameters>)$eigengenes" for function to work properly.

- **nameA, nameB**
  The names of these groups (defaults = "A" and "B"). The resulting file name (see below) and x and y axis labels for each scatter plot depend on these names.

- **plotAll**
  If TRUE, plot gene-ME correlations for all genes. If FALSE, plot correlations for only genes in the plotted module (default). Note that the output file name will be different depending on this parameter, so both can be run without overwriting results.

- **noGrey**
  If TRUE (default), the grey module genes are ignored. This parameter is only used if MEsA and MEsB are calculated.
labeledBarplot

maxPlot The maximum number of random genes to include (default=1000). Smaller values lead to smaller and less cluttered plots, usually without significantly affecting the resulting correlations. This parameter is only used if plotAll=TRUE.

pch See help file for "points". Setting pch=19 (default) produces solid circles.

fileName Name of the file to hold the plots. Since the output format is pdf, the extension should be .pdf.

Other plotting parameters that are allowable inputs to verboseScatterplot.

Value

The default output is a file called "kME_correlations_between_[nameA]_and_[nameB]_[all/inMod].pdf", where [nameA] and [nameB] correspond to the nameA and nameB input parameters, and [all/inMod] depends on whether plotAll=TRUE or FALSE. This output file contains all of the plots as separate pdf images, and will be located in the current working directory.

Note

The function "pdf", which can be found in the grDevices library, is required to run this function.

Author(s)

Jeremy Miller

Examples

# Example output file ("kME_correlations_between_A_and_B_inMod.pdf") using simulated data.

set.seed = 100
ME=matrix(0,50,5)
for (i in 1:5) ME[,i]=sample(1:100,50)
simData1 = simulateDataExpr5Modules(MEturquoise=ME[,1],MEblue=ME[,2],
MEbrown=ME[,3],MEyellow=ME[,4], MEgreen=ME[,5])
simData2 = simulateDataExpr5Modules(MEturquoise=ME[,1],MEblue=ME[,2],
MEbrown=ME[,3],MEyellow=ME[,4], MEgreen=ME[,5])
kMecomparisonScatterplot(simData1$datExpr,simData2$datExpr,simData1$trueModule)

labeledBarplot

Barplot with text or color labels.

Description

Produce a barplot with extra annotation.

Usage

labeledBarplot(
Matrix, labels,
colorLabels = FALSE,
colored = TRUE,
setStdMargins = TRUE,
stdErrors = NULL,
Arguments

Matrix
vector or a matrix to be plotted.

labels
labels to annotate the bars underneath the barplot.

colorLabels
logical: should the labels be interpreted as colors? If TRUE, the bars will be labeled by colored squares instead of text. See details.

colored
logical: should the bars be divided into segments and colored? If TRUE, assumes the labels can be interpreted as colors, and the input Matrix is square and the rows have the same labels as the columns. See details.

setStdMargins
if TRUE, the function will set margins c(3, 3, 2, 2)+0.2.

stdErrors
if given, error bars corresponding to 1.96*stdErrors will be plotted on top of the bars.

cex.lab
character expansion factor for axis labels, including the text labels underneath the barplot.

xLabelsAngle
angle at which text labels under the barplot will be printed.

... other parameters for the function barplot.

Details

Individual bars in the barplot can be identified either by printing the text of the corresponding entry in labels underneath the bar at the angle specified by xLabelsAngle, or by interpreting the labels entry as a color (see below) and drawing a correspondingly colored square underneath the bar.

For reasons of compatibility with other functions, labels are interpreted as colors after stripping the first two characters from each label. For example, the label "BmeturquoiseB" is interpreted as the color turquoise.

If colored is set, the code assumes that labels can be interpreted as colors, and the input Matrix is square and the rows have the same labels as the columns. Each bar in the barplot is then sectioned into contributions from each row entry in Matrix and is colored by the color given by the entry in labels that corresponds to the row.

Value
None.

Author(s)
Peter Langfelder
labeledHeatmap

 Produce a labeled heatmap plot

Description

Plots a heatmap plot with color legend, row and column annotation, and optional text within the heatmap.

Usage

labeledHeatmap(
  Matrix,
  xLabels, yLabels = NULL,
  xSymbols = NULL, ySymbols = NULL,
  colorLabels = NULL,
  xColorLabels = FALSE, yColorLabels = FALSE,
  checkColorsValid = TRUE,
  invertColors = FALSE,
  setStdMargins = TRUE,
  xLabelsPosition = "bottom",
  xLabelsAngle = 45,
  xLabelsAdj = 1,
  xColorWidth = 0.05,
  yColorWidth = 0.05,
  xColorOffset = par("cxy")[1]/3,
  yColorOffset = par("cxy")[2]/3,
  colors = NULL,
  naColor = "grey",
  textMatrix = NULL,
  cex.text = NULL,
  textAdj = c(0.5, 0.5),
  cex.lab = NULL,
  cex.lab.x = cex.lab,
  cex.lab.y = cex.lab,
  colors.lab.x = 1,
  colors.lab.y = 1,
  bg.lab.x = NULL,
  bg.lab.y = NULL,
  plotLegend = TRUE,
  keepLegendSpace = plotLegend,

  # Separator line specification
  verticalSeparator.x = NULL,
  verticalSeparator.col = 1,
  verticalSeparator.lty = 1,
  verticalSeparator.lwd = 1,
  verticalSeparator.ext = 0,

  horizontalSeparator.y = NULL,
  horizontalSeparator.col = 1,
)
labeledHeatmap

horizontalSeparator.lty = 1,
horizontalSeparator.lwd = 1,
horizontalSeparator.ext = 0,
...)

Arguments

Matrix    numerical matrix to be plotted in the heatmap.
xLabels    labels for the columns. See Details.
yLabels    labels for the rows. See Details.
xSymbols    additional labels used when xLabels are interpreted as colors. See Details.
ySymbols    additional labels used when yLabels are interpreted as colors. See Details.
colorLabels    logical: should xLabels and yLabels be interpreted as colors? If given, overrides xColorLabels and yColorLabels below.
xColorLabels    logical: should xLabels be interpreted as colors?
yColorLabels    logical: should yLabels be interpreted as colors?
checkColorsValid    logical: should given colors be checked for validity against the output of colors()? If this argument is FALSE, invalid color specification will trigger an error.
invertColors    logical: should the color order be inverted?
setStdMargins    logical: should standard margins be set before calling the plot function? Standard margins depend on colorLabels: they are wider for text labels and narrower for color labels. The defaults are static, that is the function does not attempt to guess the optimal margins.
xLabelsPosition    a character string specifying the position of labels for the columns. Recognized values are (unique abbreviations of) “top”, “bottom”.
xLabelsAngle    angle by which the column labels should be rotated.
xLabelsAdj    justification parameter for column labels. See par and the description of parameter “adj”.
xColorWidth    width of the color labels for the x axis expressed as a fraction of the smaller of the range of the x and y axis.
yColorWidth    width of the color labels for the y axis expressed as a fraction of the smaller of the range of the x and y axis.
xColorOffset    gap between the y axis and color labels as a fraction of the range of x axis.
yColorOffset    gap between the x axis and color labels as a fraction of the range of y axis.
colors    color palette to be used in the heatmap. Defaults to heat.colors.
nacolor    color to be used for encoding missing data.
textMatrix    optional text entries for each cell. Either a matrix of the same dimensions as Matrix or a vector of the same length as the number of entries in Matrix.
cex.text    character expansion factor for textMatrix.
textAdj    Adjustment for the entries in the text matrix. See the adj argument to text.
cex.lab    character expansion factor for text labels labeling the axes.
cex.lab.x    character expansion factor for text labels labeling the x axis. Overrides cex.lab above.
The function basically plots a standard heatmap plot of the given Matrix and embellishes it with row and column labels and/or with text within the heatmap entries. Row and column labels can be either character strings or color squares, or both.

To get simple text labels, use colorLabels=FALSE and pass the desired row and column labels in yLabels and xLabels, respectively.

To label rows and columns by color squares, use colorLabels=TRUE; yLabels and xLabels are then expected to represent valid colors. For reasons of compatibility with other functions, each entry in yLabels and xLabels is expected to consist of a color designation preceded by 2 characters: an
example would be `#turquoise`. The first two characters can be arbitrary, they are stripped. Any labels that do not represent valid colors will be considered text labels and printed in full, allowing the user to mix text and color labels.

It is also possible to label rows and columns by both color squares and additional text annotation. To achieve this, use the above technique to get color labels and, additionally, pass the desired text annotation in the `xSymbols` and `ySymbols` arguments.

**Value**

None.

**Author(s)**

Peter Langfelder

**See Also**

`heatmap`, `colors`

**Examples**

```r
# This example illustrates 4 main ways of annotating columns and rows of a heatmap.
# Copy and paste the whole example into an R session with an interactive plot window;
# alternatively, you may replace the command sizeGrWindow below by opening
# another graphical device such as pdf.

# Generate a matrix to be plotted
nCol = 8; nRow = 7;
mat = matrix(rnorm(nCol*nRow, min = -1, max = 1), nRow, nCol);
rowColors = standardColors(nRow);
colColors = standardColors(nRow + nCol)[(nRow+1):(nRow + nCol)];

rowColors;
colColors;

sizeGrWindow(9,7)
par(mfrow = c(2,2))
par(mar = c(4, 5, 4, 6));

# Label rows and columns by text:
labeledHeatmap(mat, xLabels = colColors, yLabels = rowColors,
               colors = greenWhiteRed(50),
               setStdMargins = FALSE,
               textMatrix = signif(mat, 2),
               main = "Text-labeled heatmap");

# Label rows and columns by colors:
rowLabels = paste("ME", rowColors, sep="");
collLabels = paste("ME", colColors, sep="");
labeledHeatmap(mat, xLabels = collLabels, yLabels = rowLabels,
               colors = greenWhiteRed(50),
               setStdMargins = FALSE,
               textMatrix = signif(mat, 2),
               main = "Text-labeled heatmap");
```
```r
colorLabels = TRUE,
colors = greenWhiteRed(50),
setStdMargins = FALSE,
textMatrix = signif(mat, 2),
main = "Color-labeled heatmap";

# Mix text and color labels:
rowLabels[3] = "Row 3";
collLabels[1] = "Column 1";
labeledHeatmap(mat, xLabels = collLabels, yLabels = rowLabels,
colorLabels = TRUE,
colors = greenWhiteRed(50),
setStdMargins = FALSE,
textMatrix = signif(mat, 2),
main = "Mix-labeled heatmap");

# Color labels and additional text labels
rowLabels = paste("ME", rowColors, sep="");
collLabels = paste("ME", colColors, sep="");
extraRowLabels = paste("Row", c(1:nRow));
extraColLabels = paste("Column", c(1:nCol));

# Extend margins to fit all labels
par(mar = c(6, 6, 4, 6));
labeledHeatmap(mat, xLabels = collLabels, yLabels = rowLabels,
xSymbols = extraColLabels,
ySymbols = extraRowLabels,
colorLabels = TRUE,
colors = greenWhiteRed(50),
setStdMargins = FALSE,
textMatrix = signif(mat, 2),
main = "Text- + color-labeled heatmap");
```

---

**labeledHeatmap.multiPage**

*Labeled heatmap divided into several separate plots.*

**Description**

This function produces labeled heatmaps divided into several plots. This is useful for large heatmaps where labels on individual columns and rows may become unreadably small (or overlap).

**Usage**

```r
labeledHeatmap.multiPage(
    # Input data and ornaments
    Matrix,
    xLabels, yLabels = NULL,
    xSymbols = NULL, ySymbols = NULL,
```

textMatrix = NULL,

# Paging options
rowsPerPage = NULL, maxRowsPerPage = 20,
colsPerPage = NULL, maxColsPerPage = 10,
addPageNumberToMain = TRUE,

# Further arguments to labeledHeatmap
zlim = NULL,
signed = TRUE,
main = "",

# Separator line specification
verticalSeparator.x = NULL,
verticalSeparator.col = 1,
verticalSeparator.lty = 1,
verticalSeparator.lwd = 1,
verticalSeparator.ext = 0,

horizontalSeparator.y = NULL,
horizontalSeparator.col = 1,
horizontalSeparator.lty = 1,
horizontalSeparator.lwd = 1,
horizontalSeparator.ext = 0,

...

Arguments
Matrix numerical matrix to be plotted in the heatmap.
xLabels labels for the columns. See Details.
yLabels labels for the rows. See Details.
xSymbols additional labels used when xLabels are interpreted as colors. See Details.
ySymbols additional labels used when yLabels are interpreted as colors. See Details.
textMatrix optional text entries for each cell. Either a matrix of the same dimensions as Matrix or a vector of the same length as the number of entries in Matrix.
rowsPerPage optional list in which each component is a vector specifying which rows should appear together in each plot. If not given, will be generated automatically based on maxRowsPerPage below and the number of rows in Matrix.
maxRowsPerPage integer giving maximum number of rows appearing on each plot (page).
colsPerPage optional list in which each component is a vector specifying which columns should appear together in each plot. If not given, will be generated automatically based on maxColsPerPage below and the number of rows in Matrix.
maxColsPerPage integer giving maximum number of columns appearing on each plot (page).
addPageNumberToMain logical: should plot/page number be added to the main title of each plot?
zlim Optional specification of the extreme values for the color scale. If not given, will be determined from the input Matrix.
main Main title for each plot/page, optionally with the plot/page number added.
signed logical: should the input matrix be converted to colors using a scale centered at zero?
verticalSeparator.x indices of columns after which separator lines (vertical lines between columns) should be drawn. NULL means no lines will be drawn.
verticalSeparator.col color(s) of the vertical separator lines. Recycled if need be.
verticalSeparator.lty line type of the vertical separator lines. Recycled if need be.
verticalSeparator.lwd line width of the vertical separator lines. Recycled if need be.
verticalSeparator.ext number giving the extension of the separator line into the margin as a fraction of the margin width. 0 means no extension, 1 means extend all the way through the margin.
horizontalSeparator.y indices of columns after which separator lines (horizontal lines between columns) should be drawn. NULL means no lines will be drawn.
horizontalSeparator.col color(s) of the horizontal separator lines. Recycled if need be.
horizontalSeparator.lty line type of the horizontal separator lines. Recycled if need be.
horizontalSeparator.lwd line width of the horizontal separator lines. Recycled if need be.
horizontalSeparator.ext number giving the extension of the separator line into the margin as a fraction of the margin width. 0 means no extension, 1 means extend all the way through the margin.
... other arguments to function labeledHeatmap.

Details

The function labeledHeatmap is used to produce each plot/page; most arguments are described in more detail in the help file for that function.

In each plot/page labeledHeatmap plots a standard heatmap plot of an appropriate sub-rectangle of matrix and embellishes it with row and column labels and/or with text within the heatmap entries. Row and column labels can be either character strings or color squares, or both.

To get simple text labels, use colorLabels=FALSE and pass the desired row and column labels in yLabels and xLabels, respectively.

To label rows and columns by color squares, use colorLabels=TRUE; yLabels and xLabels are then expected to represent valid colors. For reasons of compatibility with other functions, each entry in yLabels and xLabels is expected to consist of a color designation preceded by 2 characters: an example would be "#eturquoise". The first two characters can be arbitrary, they are stripped. Any labels that do not represent valid colors will be considered text labels and printed in full, allowing the user to mix text and color labels.

It is also possible to label rows and columns by both color squares and additional text annotation. To achieve this, use the above technique to get color labels and, additionally, pass the desired text annotation in the xSymbols and ySymbols arguments.

If rowsPerPage (colsPerPage) is not given, rows (columns) are allocated automatically as uniformly as possible, in contiguous blocks of size at most maxRowsPerPage (maxColsPerPage). The allocation is performed by the function allocateJobs.
Value

None.

Author(s)

Peter Langfelder

See Also

The workhorse function `labeledHeatmap` for the actual heatmap plot;
function `allocateJobs` for the allocation of rows/columns to each plot.

labelPoints

Label scatterplot points

Description

Given scatterplot point coordinates, the function tries to place labels near the points such that the
labels overlap as little as possible. User beware: the algorithm implemented here is quite primitive
and while it will help in many cases, it is by no means perfect. Consider this function experimental.
We hope to improve the algorithm in the future to make it useful in a broader range of situations.

Usage

labelPoints(
  x, y, labels,
  cex = 0.7, offs = 0.01, xpd = TRUE,
  jiggle = 0, protectEdges = TRUE,
  doPlot = TRUE, ...)

Arguments

  x        a vector of x coordinates of the points
  y        a vector of y coordinates of the points
  labels   labels to be placed next to the points
  cex      character expansion factor for the labels
  offs     offset of the labels from the plotted coordinates in inches
  xpd      logical: controls truncating labels to fit within the plotting region. See `par`.
  jiggle   amount of random noise to be added to the coordinates. This may be useful if
            the scatterplot is too regular (such as all points on one straight line).
  protectEdges logical: should labels be shifted inside the (actual or virtual) frame of the plot?
  doPlot   logical: should the labels be actually added to the plot? Value FALSE may be use-
            ful if the user would like to simply compute the best label positions the function
            can come up with.
  ...      other arguments to function `text`.
Details

The algorithm basically works by finding the direction of most surrounding points, and attempting to place the label in the opposite direction. There are (not uncommon) situations in which this placement is suboptimal; the author promises to further develop the function sometime in the future.

Note that this function does not plot the actual scatterplot; only the labels are plotted. Plotting the scatterplot is the responsibility of the user.

The argument `offs` needs to be carefully tuned to the size of the plotted symbols. Sorry, no automation here yet.

The argument `protectEdges` can be used to shift labels that would otherwise extend beyond the plot to within the plot. Sometimes this may cause some overlapping with other points or labels; use with care.

Value

Invisibly, a data frame with 3 columns, giving the x and y positions of the labels, and the labels themselves.

Author(s)

Peter Langfelder

See Also

`plot.default.text`

Examples

```r
# generate some random points
set.seed(11);
  n = 20;
  x = runif(n);
  y = runif(n);

# Create a basic scatterplot
col = standardColors(n);
  plot(x,y, pch = 21, col =1, bg = col, cex = 2.6,
     xlim = c(-0.1, 1.1), ylim = c(-0.1, 1.0));
  labelPoints(x, y, paste("Pt", c(1:n), sep=""), offs = 0.10, cex = 1);

# label points using longer text labels. Note the positioning is not perfect, but close enough.
plot(x,y, pch = 21, col =1, bg = col, cex = 2.6,
     xlim = c(-0.1, 1.1), ylim = c(-0.1, 1.0));
  labelPoints(x, y, col, offs = 0.10, cex = 0.8);
```

Description

Converts a vector or array of numerical labels into a corresponding vector or array of colors corresponding to the labels.
Usage

labels2colors(labels, zeroIsGrey = TRUE, colorSeq = NULL, naColor = "grey", 
commonColorCode = TRUE)

Arguments

labels Vector or matrix of non-negative integer or other (such as character) labels. See 
details.
zeroIsGrey If TRUE, labels 0 will be assigned color grey. Otherwise, labels below 1 will 
trigger an error.
colorSeq Color sequence corresponding to labels. If not given, a standard sequence will 
be used.
naColor Color that will encode missing values.
commonColorCode logical: if labels is a matrix, should each column have its own colors?

Details

If labels is numeric, it is used directly as index to the standard color sequence. If 0 is present 
among the labels and zeroIsGrey=TRUE, labels 0 are given grey color.

If labels is not numeric, its columns are turned into factors and the numeric representation of each 
factor is used to assign the corresponding colors. In this case commonColorCode governs whether 
each column gets its own color code, or whether the color code will be universal.

The standard sequence start with well-distinguishable colors, and after about 40 turns into a quasi-
random sampling of all colors available in R with the exception of all shades of grey (and gray).

If the input labels have a dimension attribute, it is copied into the output, meaning the dimensions 
of the returned value are the same as those of the input labels.

Value

A vector or array of character strings of the same length or dimensions as labels.

Author(s)

Peter Langfelder, <Peter.Langfelder@gmail.com>

Examples

labels = c(0:20);
labels2colors(labels);
labels = matrix(letters[1:9], 3,3);
labels2colors(labels)
# Note the difference when commonColorCode = FALSE
labels2colors(labels, commonColorCode = FALSE)
**list2multiData** Convert a list to a multiData structure and vice-versa.

**Description**

`list2multiData` converts a list to a multiData structure; `multiData2list` does the inverse.

**Usage**

```r
list2multiData(data)
multiData2list(multiData)
```

**Arguments**

- `data` A list to be converted to a multiData structure.
- `multiData` A multiData structure to be converted to a list.

**Details**

A multiData structure is a vector of lists (one list for each set) where each list has a component `data` containing some useful information.

**Value**

For `list2multiData`, a multiData structure; for `multiData2list`, the corresponding list.

**Author(s)**

Peter Langfelder

---

**lowerTri2matrix** Reconstruct a symmetric matrix from a distance (lower-triangular) representation

**Description**

Assuming the input vector contains a vectorized form of the distance representation of a symmetric matrix, this function creates the corresponding matrix. This is useful when re-forming symmetric matrices that have been vectorized to save storage space.

**Usage**

```r
lowerTri2matrix(x, diag = 1)
```

**Arguments**

- `x` a numeric vector
- `diag` value to be put on the diagonal. Recycled if necessary.
Details

The function assumes that x contains the vectorized form of the distance representation of a symmetric matrix. In particular, x must have a length that can be expressed as n^2(n-1)/2, with n an integer. The result of the function is then an n times n matrix.

Value

A symmetric matrix whose lower triangle is given by x.

Author(s)

Peter Langfelder

Examples

```r
# Create a symmetric matrix
m = matrix(c(1:16), 4,4)
mat = (m + t(m));
diag(mat) = 0;

# Print the matrix
mat

# Take the lower triangle and vectorize it (in two ways)
x1 = mat[lower.tri(mat)]
x2 = as.vector(as.dist(mat))

all.equal(x1, x2) # The vectors are equal

# Turn the vectors back into matrices
new.mat = lowerTri2matrix(x1, diag = 0);

# Did we get back the same matrix?
all.equal(mat, new.mat)
```

matchLabels

Relabel module labels to best match the given reference labels

Description

Given a source and reference vectors of module labels, the function produces a module labeling that is equivalent to source, but individual modules are re-labeled so that modules with significant overlap in source and reference have the same labels.

Usage

```r
matchLabels(source, reference, pThreshold = 5e-2, na.rm = TRUE, ignoreLabels = if (is.numeric(reference)) 0 else "grey")
```
matchLabels

extraLabels = if (is.numeric(reference)) c(1:1000) else standardColors()

Arguments

source a vector or a matrix of reference labels. The labels may be numeric or character.
reference a vector of reference labels.
pThreshold threshold of Fisher’s exact test for considering modules to have a significant overlap.
na.rm logical: should missing values in either source or reference be removed? If not, missing values may be treated as a standard label or the function may throw an error (exact behaviour depends on whether the input labels are numeric or not).
ignoreLabels labels in source and reference to be considered unmatchable. These labels are excluded from the re-labeling procedure.
extraLabels a vector of labels for modules in source that cannot be matched to any modules in reference. The user should ensure that this vector contains enough labels since the function automatically removes a values that occur in either source, reference or ignoreLabels, to avoid possible confusion.

Details

Each column of source is treated separately. Unlike in previous version of this function, source and reference labels can be any labels, not necessarily of the same type.

The function calculates the overlap of the source and reference modules using Fisher’s exact test. It then attempts to relabel source modules such that each source module gets the label of the reference module that it overlaps most with, subject to not renaming two source modules to the same reference module. (If two source modules point to the same reference module, the one with the more significant overlap is chosen.)

Those source modules that cannot be matched to a reference module are labeled using those labels from extraLabels that do not occur in either of source, reference or ignoreLabels.

Value

A vector (if the input source labels are a vector) or a matrix (if the input source labels are a matrix) of the new labels.

Author(s)

Peter Langfelder

See Also

overlapTable for calculation of overlap counts and p-values;
standardColors for standard non-numeric WGCNA labels.
Construct a network from a matrix

Description

Constructs a network

Usage

```r
matrixToNetwork(
  mat,
  symmetrizeMethod = c("average", "min", "max"),
  signed = TRUE,
  min = NULL, max = NULL,
  power = 12,
  diagEntry = 1)
```

Arguments

- `mat` - matrix to be turned into a network. Must be square.
- `symmetrizeMethod` - method for symmetrizing the matrix. The method will be applied to each component of `mat` and its transpose.
- `signed` - logical: should the resulting network be signed? Unsigned networks are constructed from `abs(mat)`.
- `min` - minimum allowed value for `mat`. If `NULL`, the actual attained minimum of `mat` will be used. Missing data are ignored. Values below `min` are truncated to `min`.
- `max` - maximum allowed value for `mat`. If `NULL`, the actual attained maximum of `mat` will be used. Missing data are ignored. Values below `max` are truncated to `max`.
- `power` - the soft-thresholding power.
- `diagEntry` - the value of the entries on the diagonal in the result. This is usually 1 but some applications may require a zero (or even NA) diagonal.

Details

If `signed` is `FALSE`, the matrix `mat` is first converted to its absolute value.

This function then symmetrizes the matrix using the `symmetrizeMethod` component-wise on `mat` and `t(mat)` (i.e., the transpose of `mat`).

In the next step, the symmetrized matrix is linearly scaled to the interval [0,1] using either `min` and `max` (each either supplied or determined from the matrix). Values outside of the `[min, max]` range are truncated to `min` or `max`.

Lastly, the adjacency is calculated by raising the matrix to `power`. The diagonal of the result is set to `diagEntry`. Note that most WGCNA functions expect the diagonal of an adjacency matrix to be 1.

Value

The adjacency matrix that encodes the network.
mergeCloseModules

Description

Merges modules in gene expression networks that are too close as measured by the correlation of their eigengenes.

Usage

```r
mergeCloseModules(
  # input data
eexprData, colors,

  # Optional starting eigengenes
  MEs = NULL,

  # Optional restriction to a subset of all sets
  useSets = NULL,

  # If missing data are present, impute them?
  impute = TRUE,

  # Input handling options
  checkDataFormat = TRUE,
  unassdColor = if (is.numeric(colors)) 0 else "grey",

  # Options for eigengene network construction
  corFnc = cor, corOptions = list(use = "p"),
  useAbs = FALSE,

  # Options for constructing the consensus
  equalizeQuantiles = FALSE,
  quantileSummary = "mean",
  consensusQuantile = 0,

  # Merging options
  cutHeight = 0.2,
  iterate = TRUE,
)
```
mergeCloseModules

# Output options
relabel = FALSE,
colorSeq = NULL,
getNewMEs = TRUE,
getNewUnassdME = TRUE,

# Options controlling behaviour of the function
trapErrors = FALSE,
verbose = 1, indent = 0)

Arguments

exprData  Expression data, either a single data frame with rows corresponding to samples and columns to genes, or in a multi-set format (see checkSets). See checkDataStructure below.
colors    A vector (numeric, character or a factor) giving module colors for genes. The method only makes sense when genes have the same color label in all sets, hence a single vector.
MEs       If module eigengenes have been calculated before, the user can save some computational time by inputting them. MEs should have the same format as exprData. If they are not given, they will be calculated.
useSets   A vector of scalar allowing the user to specify which sets will be used to calculate the consensus dissimilarity of module eigengenes. Defaults to all given sets.
impute    Should missing values be imputed in eigengene calculation? If imputation is disabled, the presence of NA entries will cause the eigengene calculation to fail and eigengenes will be replaced by their hubgene approximation. See moduleEigengenes for more details.
checkDataFormat
If TRUE, the function will check exprData and MEs for correct multi-set structure. If single set data is given, it will be converted into a format usable for the function. If FALSE, incorrect structure of input data will trigger an error.
unassdColor Specifies the string that labels unassigned genes. Module of this color will not enter the module eigengene clustering and will not be merged with other modules.
corFnc    Correlation function to be used to calculate correlation of module eigengenes.
corOptions Can be used to specify options to the correlation function, in addition to argument x which is used to pass the actual data to calculate the correlation of.
useAbs    Specifies whether absolute value of correlation or plain correlation (of module eigengenes) should be used in calculating module dissimilarity.
equalizeQuantiles Logical: should quantiles of the eigengene dissimilarity matrix be equalized ("quantile normalized")? The default is FALSE for reproducibility of old code, but better results will probably be achieved if quantile equalization is used.
quantileSummary One of "mean" or "median". Controls how a reference dissimilarity is computed from the input ones (using mean or median, respectively).
mergeCloseModules

consensusQuantile A number giving the desired quantile to use in the consensus similarity calculation (see details).

cutHeight Maximum dissimilarity (i.e., 1-correlation) that qualifies modules for merging.

iterate Controls whether the merging procedure should be repeated until there is no change. If FALSE, only one iteration will be executed.

relabel Controls whether, after merging, color labels should be ordered by module size.

colorSeq Color labels to be used for relabeling. Defaults to the standard color order used in this package if colors are not numeric, and to integers starting from 1 if colors is numeric.

getNewMEs Controls whether module eigengenes of merged modules should be calculated and returned.

getNewUnassdME When doing module eigengene manipulations, the function does not normally calculate the eigengene of the ‘module’ of unassigned (‘grey’) genes. Setting this option to TRUE will force the calculation of the unassigned eigengene in the returned newMEs, but not in the returned oldMEs.

trapErrors Controls whether computational errors in calculating module eigengenes, their dissimilarity, and merging trees should be trapped. If TRUE, errors will be trapped and the function will return the input colors. If FALSE, errors will cause the function to stop.

verbose Controls verbosity of printed progress messages. 0 means silent, up to (about) 5 the verbosity gradually increases.

indent A single non-negative integer controlling indentation of printed messages. 0 means no indentation, each unit above that adds two spaces.

Details

This function returns the color labels for modules that are obtained from the input modules by merging ones that are closely related. The relationships are quantified by correlations of module eigengenes; a “consensus” measure is defined as the “consensus quantile” over the corresponding relationship in each set. Once the (dis-)similarity is calculated, average linkage hierarchical clustering of the module eigengenes is performed, the dendrogram is cut at the height cutHeight and modules on each branch are merged. The process is (optionally) repeated until no more modules are merged.

If, for a particular module, the module eigengene calculation fails, a hubgene approximation will be used.

The user should be aware that if a computational error occurs and trapErrors==TRUE, the returned list (see below) will not contain all of the components returned upon normal execution.

Value

If no errors occurred, a list with components

colors Color labels for the genes corresponding to merged modules. The function attempts to mimic the mode of the input colors: if the input colors is numeric, character and factor, respectively, so is the output. Note, however, that if the function performs relabeling, a standard sequence of labels will be used: integers starting at 1 if the input colors is numeric, and a sequence of color labels otherwise (see colorSeq above).
Hierarchical clustering dendrogram (average linkage) of the eigengenes of the most recently computed tree. If \texttt{iterate} was set \texttt{TRUE}, this will be the dendrogram of the merged modules, otherwise it will be the dendrogram of the original modules.

Hierarchical clustering dendrogram (average linkage) of the eigengenes of the original modules.

The input cutHeight.

Module eigengenes of the original modules in the sets given by \texttt{useSets}.

Module eigengenes of the merged modules in the sets given by \texttt{useSets}.

A boolean set to \texttt{TRUE}.

If an error occurred and \texttt{trapErrors==TRUE}, the list only contains these components:

A copy of the input colors.

A boolean set to \texttt{FALSE}.

Author(s)

Peter Langfelder, <Peter.Langfelder@gmail.com>

Description

This is a meta-analysis complement to functions \texttt{standardScreeningBinaryTrait} and \texttt{standardScreeningNumericTrait}. Given expression (or other) data from multiple independent data sets, and the corresponding clinical traits or outcomes, the function calculates multiple screening statistics in each data set, then calculates meta-analysis Z scores, p-values, and optionally q-values (False Discovery Rates). Three different ways of calculating the meta-analysis Z scores are provided: the Stouffer method, weighted Stouffer method, and using user-specified weights.

Usage

\begin{verbatim}
metaAnalysis(multiExpr, multiTrait,
  binary = NULL,
  metaAnalysisWeights = NULL,
  corFnc = cor, corOptions = list(use = "p"),
  getQvalues = FALSE,
  getAreaUnderROC = FALSE,
  useRankPvalue = TRUE,
  rankPvalueOptions = list(),
  setNames = NULL,
  kruskalTest = FALSE, var.equal = FALSE,
  metaKruskal = kruskalTest, na.action = "na.exclude")
\end{verbatim}
Arguments

**multiExpr**
Expression data (or other data) in multi-set format (see checkSets). A vector of lists; in each list there must be a component named data whose content is a matrix or dataframe or array of dimension 2.

**multiTrait**
Trait or outcome data in multi-set format. Only one trait is allowed; consequently, the data component of each component list can be either a vector or a data frame (matrix, array of dimension 2).

**binary**
Logical: is the trait binary (TRUE) or continuous (FALSE)? If not given, the decision will be made based on the content of multiTrait.

**metaAnalysisWeights**
Optional specification of set weights for meta-analysis. If given, must be a vector of non-negative weights, one entry for each set contained in multiExpr.

**corFnc**
Correlation function to be used for screening. Should be either the default cor or its robust alternative, bicor.

**corOptions**
A named list giving extra arguments to be passed to the correlation function.

**getQvalues**
Logical: should q-values (FDRs) be calculated?

**getAreaUnderROC**
Logical: should area under the ROC be calculated? Caution, enabling the calculation will slow the function down considerably for large data sets.

**useRankPvalue**
Logical: should the rankPvalue function be used to obtain alternative meta-analysis statistics?

**rankPvalueOptions**
Additional options for function rankPvalue. These include na.last (default "keep"), ties.method (default "average"), calculateQvalue (default copied from input getQvalues), and pValueMethod (default "all"). See the help file for rankPvalue for full details.

**setNames**
Optional specification of set names (labels). These are used to label the corresponding components of the output. If not given, will be taken from the names attribute of multiExpr. If names(multiExpr) is NULL, generic names of the form Set_1, Set_2, ... will be used.

**kruskalTest**
Logical: should the Kruskal test be performed in addition to t-test? Only applies to binary traits.

**var.equal**
Logical: should the t-test assume equal variance in both groups? If TRUE, the function will warn the user that the returned test statistics will be different from the results of the standard t.test function.

**metaKruskal**
Logical: should the meta-analysis be based on the results of Krusal test (TRUE) or Student t-test (FALSE)?

**na.action**
Specification of what should happen to missing values in t.test.

Details

The Stouffer method of combines Z statistics by simply taking a mean of input Z statistics and multiplying it by sqrt(n), where n is the number of input data sets. We refer to this method as Stouffer.equalWeights. In general, a better (i.e., more powerful) method of combining Z statistics is to weigh them by the number of degrees of freedom (which approximately equals n). We refer to this method as weightedStouffer. Finally, the user can also specify custom weights, for example if a data set needs to be downweighted due to technical concerns; however, specifying own weights by hand should be done carefully to avoid possible selection biases.

**metaAnalysis**
metaAnalysis

Data frame with the following components:

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Identifier of the input genes (or other variables)</td>
</tr>
<tr>
<td>z.equalWeights</td>
<td>Meta-analysis Z statistics obtained using Stouffer’s method with equal weights</td>
</tr>
<tr>
<td>p.equalWeights</td>
<td>p-values corresponding to z.equalWeights</td>
</tr>
<tr>
<td>q.equalWeights</td>
<td>q-values corresponding to p.equalWeights, only present if getQvalues is TRUE</td>
</tr>
<tr>
<td>z.RootDoFWeights</td>
<td>Meta-analysis Z statistics obtained using Stouffer’s method with weights given by the square root of the number of (non-missing) samples in each data set</td>
</tr>
<tr>
<td>p.RootDoFWeights</td>
<td>p-values corresponding to z.RootDoFWeights</td>
</tr>
<tr>
<td>q.RootDoFWeights</td>
<td>q-values corresponding to p.RootDoFWeights, only present if getQvalues is TRUE</td>
</tr>
<tr>
<td>z.DoFWeights</td>
<td>Meta-analysis Z statistics obtained using Stouffer’s method with weights given by the number of (non-missing) samples in each data set</td>
</tr>
<tr>
<td>p.DoFWeights</td>
<td>p-values corresponding to z.DoFWeights</td>
</tr>
<tr>
<td>q.DoFWeights</td>
<td>q-values corresponding to p.DoFWeights, only present if getQvalues is TRUE.</td>
</tr>
<tr>
<td>z.userWeights</td>
<td>Meta-analysis Z statistics obtained using Stouffer’s method with user-defined weights. Only present if input metaAnalysisWeights are present.</td>
</tr>
<tr>
<td>p.userWeights</td>
<td>p-values corresponding to z.userWeights</td>
</tr>
<tr>
<td>q.userWeights</td>
<td>q-values corresponding to p.userWeights, only present if getQvalues is TRUE.</td>
</tr>
</tbody>
</table>

The next set of columns is present only if input userankpvalue is TRUE and contain the output of the function rankPvalue with the same column weights as the above meta-analysis. Depending on the input options calculateQvalue and pValueMethod in rankPvalueOptions, some columns may be missing. The following columns are calculated using equal weights for each data set.

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pValueExtremeRank.equalWeights</td>
<td>This is the minimum between pValueLowRank and pValueHighRank, i.e. ( \min(p\text{ValueLow}, p\text{ValueHigh}) )</td>
</tr>
<tr>
<td>pValueLowRank.equalWeights</td>
<td>Asymptotic p-value for observing a consistently low value across the columns of dataSet based on the rank method.</td>
</tr>
<tr>
<td>pValueHighRank.equalWeights</td>
<td>Asymptotic p-value for observing a consistently low value across the columns of dataSet based on the rank method.</td>
</tr>
<tr>
<td>pValueExtremeScale.equalWeights</td>
<td>This is the minimum between pValueLowScale and pValueHighScale, i.e. ( \min(p\text{ValueLow}, p\text{ValueHigh}) )</td>
</tr>
<tr>
<td>pValueLowScale.equalWeights</td>
<td>Asymptotic p-value for observing a consistently low value across the columns of dataSet based on the Scale method.</td>
</tr>
<tr>
<td>pValueHighScale.equalWeights</td>
<td>Asymptotic p-value for observing a consistently low value across the columns of dataSet based on the Scale method.</td>
</tr>
<tr>
<td>qValueExtremeRank.equalWeights</td>
<td>local false discovery rate (q-value) corresponding to the p-value pValueExtremeRank</td>
</tr>
</tbody>
</table>


qValueLowRank.equalWeights
  local false discovery rate (q-value) corresponding to the p-value pValueLowRank
qValueHighRank.equalWeights
  local false discovery rate (q-value) corresponding to the p-value pValueHighRank
qValueExtremeScale.equalWeights
  local false discovery rate (q-value) corresponding to the p-value pValueExtremeScale
qValueLowScale.equalWeights
  local false discovery rate (q-value) corresponding to the p-value pValueLowScale
qValueHighScale.equalWeights
  local false discovery rate (q-value) corresponding to the p-value pValueHighScale

...  Analogous columns calculated by weighting each input set using the square root of the number of samples, number of samples, and user weights (if given). The corresponding column names carry the suffixes RootDofWeights, DofWeights, UserWeights.

The following columns contain results returned by `standardscreeningBinaryTrait` or `standardscreeningNumericTrait` (depending on whether the input trait is binary or continuous).
For binary traits, the following information is returned for each set:

corPearson.Set_1, corPearson.Set_2, ...
  Pearson correlation with a binary numeric version of the input variable. The numeric variable equals 1 for level 1 and 2 for level 2. The levels are given by levels(factor(y)).
t.Student.Set_1, t.Student.Set_2, ...
  Student t-test statistic
pvalueStudent.Set_1, pvalueStudent.Set_2, ...
  two-sided Student t-test p-value.
qvalueStudent.Set_1, qvalueStudent.Set_2, ...
  (if input qvalues==TRUE) q-value (local false discovery rate) based on the Student T-test p-value (Storey et al 2004).
foldChange.Set_1, foldChange.Set_2, ...
  a (signed) ratio of mean values. If the mean in the first group (corresponding to level 1) is larger than that of the second group, it equals meanFirstGroup/meanSecondGroup. But if the mean of the second group is larger than that of the first group it equals -meanSecondGroup/meanFirstGroup (notice the minus sign).
meanFirstGroup.Set_1, meanSecondGroup.Set_2, ...
  means of columns in input datExpr across samples in the second group.
SE.FirstGroup.Set_1, SE.FirstGroup.Set_2, ...
  standard errors of columns in input datExpr across samples in the first group. Recall that SE(x)=sqrt(var(x)/n) where n is the number of non-missing values of x.
SE.SecondGroup.Set_1, SE.SecondGroup.Set_2, ...
  standard errors of columns in input datExpr across samples in the second group.
areaUnderROC.Set_1, areaUnderROC.Set_2, ...
  the area under the ROC, also known as the concordance index or C.index. This is a measure of discriminatory power. The measure lies between 0 and 1 where 0.5 indicates no discriminatory power. 0 indicates that the "opposite" predictor
has perfect discriminatory power. To compute it we use the function `rcorr.cens`
with `outx=TRUE` (from Frank Harrel’s package Hmisc).

```r
nPresentSamples.Set_1, nPresentSamples.Set_2, ...
```
number of samples with finite measurements for each gene.

If input `kruskalTest` is TRUE, the following columns further summarize results of Kruskal-Wallis test:

```r
stat.Kruskal.Set_1, stat.Kruskal.Set_2, ...
```
Kruskal-Wallis test statistic.

```r
stat.Kruskal.signed.Set_1, stat.Kruskal.signed.Set_2,...
```
(Warning: experimental) Kruskal-Wallis test statistic including a sign that indicates whether the average rank is higher in second group (positive) or first group (negative).

```r
pvaluekruskal.Set_1, pvaluekruskal.Set_2, ...
```
Kruskal-Wallis test p-value.

```r
qkruskal.Set_1, qkruskal.Set_2, ...
```
q-values corresponding to the Kruskal-Wallis test p-value (if input `qvalues`==TRUE).

```r
Z.Set1, Z.Set2, ...
```
Z statistics obtained from `pvalueStudent.Set1`, `pvalueStudent.Set2`, ...
or from `pvaluekruskal.Set1`, `pvaluekruskal.Set2`, .... depending on input `metaKruskal`.

For numeric traits, the following columns are returned:

```r
cor.Set_1, cor.Set_2, ...
correlations of all genes with the trait
```

```r
Z.Set1, Z.Set2, ...
```
Fisher Z statistics corresponding to the correlations

```r
pvalueStudent.Set_1, pvalueStudent.Set_2, ...
```
Student p-values of the correlations

```r
qvalueStudent.Set_1, qvalueStudent.Set_1, ...
```
(if input `qvalues`==TRUE) q-values of the correlations calculated from the p-values

```r
AreaUnderROC.Set_1, AreaUnderROC.Set_2, ...
```
area under the ROC

```r
nPresentSamples.Set_1, nPresentSamples.Set_2, ...
```
number of samples present for the calculation of each association.

**Author(s)**

Peter Langfelder

**References**

For Stouffer’s method, see


A discussion of weighted Stouffer’s method can be found in

Whitlock, M. C., Combining probability from independent tests: the weighted Z-method is superior to Fisher’s approach, Journal of Evolutionary Biology 18:5 1368 (2005)
**metaZfunction**

**Description**

The function calculates a meta analysis Z statistic based on an input data frame of Z statistics.

**Usage**

`metaZfunction(datZ, columnweights = NULL)`

**Arguments**

- `datZ`: Matrix or data frame of Z statistics (assuming standard normal distribution under the null hypothesis). Rows correspond to genes, columns to independent data sets.
- `columnweights`: Optional vector of non-negative numbers for weighing the columns of `datZ`.

**Details**

For example, if `datZ` has 3 columns whose columns are labelled Z1, Z2, Z3 then ZMeta = (Z1 + Z2 + Z3) / sqrt(3).

Under the null hypothesis (where all Z statistics follow a standard normal distribution and the Z statistics are independent), ZMeta also follows a standard normal distribution. To calculate a 2 sided p-value, one can use the following code: `pvalue = 2 * pnorm(-abs(ZMeta))`

**Value**

Vector of meta analysis Z statistic. Under the null hypothesis this should follow a standard normal distribution.

**Author(s)**

Steve Horvath
moduleColor.getMEprefix

Get the prefix used to label module eigengenes.

Description

Returns the currently used prefix used to label module eigengenes. When returning module eigengenes in a dataframe, names of the corresponding columns will start with the given prefix.

Usage

moduleColor.getMEprefix()

Details

Returns the prefix used to label module eigengenes. When returning module eigengenes in a dataframe, names of the corresponding columns will consist of the corresponding color label preceded by the given prefix. For example, if the prefix is "PC" and the module is turquoise, the corresponding module eigengene will be labeled "PCturquoise". Most of old code assumes "PC", but "ME" is more instructive and used in some newer analyses.

Value

A character string.

Note

Currently the standard prefix is "ME" and there is no way to change it.

Author(s)

Peter Langfelder, <Peter.Langfelder@gmail.com>

See Also

moduleEigengenes

moduleEigengenes

Calculate module eigengenes.

Description

Calculates module eigengenes (1st principal component) of modules in a given single dataset.
Usage

```r
moduleEigengenes(expr,
  colors,
  impute = TRUE,
  nPC = 1,
  align = "along average",
  excludeGrey = FALSE,
  grey = if (is.numeric(colors)) 0 else "grey",
  subHubs = TRUE,
  trapErrors = FALSE,
  returnValidOnly = trapErrors,
  softPower = 6,
  scale = TRUE,
  verbose = 0, indent = 0)
```

Arguments

- **expr**: Expression data for a single set in the form of a data frame where rows are samples and columns are genes (probes).
- **colors**: A vector of the same length as the number of probes in `expr`, giving module color for all probes (genes). Color "grey" is reserved for unassigned genes.
- **impute**: If `true`, expression data will be checked for the presence of NA entries and if the latter are present, numerical data will be imputed, using function `impute.knn` and probes from the same module as the missing datum. The function `impute.knn` uses a fixed random seed giving repeatable results.
- **nPC**: Number of principal components and variance explained entries to be calculated. Note that only the first principal component is returned; the rest are used only for the calculation of proportion of variance explained. The number of returned variance explained entries is currently `min(nPC, 10)`. If given `nPC` is greater than 10, a warning is issued.
- **align**: Controls whether eigengenes, whose orientation is undetermined, should be aligned with average expression (`align = "along average"`, the default) or left as they are (`align = ""`). Any other value will trigger an error.
- **excludeGrey**: Should the improper module consisting of ‘grey’ genes be excluded from the eigengenes?
- **grey**: Value of `colors` designating the improper module. Note that if `colors` is a factor of numbers, the default value will be incorrect.
- **subHubs**: Controls whether hub genes should be substituted for missing eigengenes. If `TRUE`, each missing eigengene (i.e., eigengene whose calculation failed and the error was trapped) will be replaced by a weighted average of the most connected hub genes in the corresponding module. If this calculation fails, or if `subHubs==FALSE`, the value of `trapErrors` will determine whether the offending module will be removed or whether the function will issue an error and stop.
- **trapErrors**: Controls handling of errors from that may arise when there are too many NA entries in expression data. If `TRUE`, errors from calling these functions will be trapped without abnormal exit. If `FALSE`, errors will cause the function to stop. Note, however, that `subHubs` takes precedence in the sense that if `subHubs==TRUE` and `trapErrors==FALSE`, an error will be issued only if both the principal component and the hubgene calculations have failed.
returnValidOnly

logical; controls whether the returned data frame of module eigengenes contains columns corresponding only to modules whose eigengenes or hub genes could be calculated correctly (TRUE), or whether the data frame should have columns for each of the input color labels (FALSE).

softPower

The power used in soft-thresholding the adjacency matrix. Only used when the hubgene approximation is necessary because the principal component calculation failed. It must be non-negative. The default value should only be changed if there is a clear indication that it leads to incorrect results.

scale

logical; can be used to turn off scaling of the expression data before calculating the singular value decomposition. The scaling should only be turned off if the data has been scaled previously, in which case the function can run a bit faster. Note however that the function first imputes, then scales the expression data in each module. If the expression contain missing data, scaling outside of the function and letting the function impute missing data may lead to slightly different results than if the data is scaled within the function.

verbose

Controls verbosity of printed progress messages. 0 means silent, up to (about) 5 the verbosity gradually increases.

indent

A single non-negative integer controlling indentation of printed messages. 0 means no indentation, each unit above that adds two spaces.

Details

Module eigengene is defined as the first principal component of the expression matrix of the corresponding module. The calculation may fail if the expression data has too many missing entries. Handling of such errors is controlled by the arguments subHubs and trapErrors. If subHubs==TRUE, errors in principal component calculation will be trapped and a substitute calculation of hubgenes will be attempted. If this fails as well, behaviour depends on trapErrors: if TRUE, the offending module will be ignored and the return value will allow the user to remove the module from further analysis; if FALSE, the function will stop.

From the user’s point of view, setting trapErrors=FALSE ensures that if the function returns normally, there will be a valid eigengene (principal component or hubgene) for each of the input colors. If the user sets trapErrors=TRUE, all calculational (but not input) errors will be trapped, but the user should check the output (see below) to make sure all modules have a valid returned eigengene.

While the principal component calculation can fail even on relatively sound data (it does not take all that many “well-placed” NA to torpedo the calculation), it takes many more irregularities in the data for the hubgene calculation to fail. In fact such a failure signals there likely is something seriously wrong with the data.

Value

A list with the following components:

eigengenes Module eigengenes in a dataframe, with each column corresponding to one eigengene. The columns are named by the corresponding color with an “+ME” prepended, e.g., +Meturquoise etc. If returnValidOnly==FALSE, module eigengenes whose calculation failed have all components set to NA.

averageExpr If align == “along average”, a dataframe containing average normalized expression in each module. The columns are named by the corresponding color with an “+AE” prepended, e.g., +Aturquoise etc.
varExplained A dataframe in which each column corresponds to a module, with the component `varExplained[PC, module]` giving the variance of module explained by the principal component no. PC. The calculation is exact irrespective of the number of computed principal components. At most 10 variance explained values are recorded in this dataframe.

nPC A copy of the input nPC.

validMEs A boolean vector. Each component (corresponding to the columns in data) is TRUE if the corresponding eigengene is valid, and FALSE if it is invalid. Valid eigengenes include both principal components and their hubgene approximations. When `returnValidOnly==FALSE`, by definition all returned eigengenes are valid and the entries of `validMEs` are all TRUE.

validColors A copy of the input colors with entries corresponding to invalid modules set to grey if given, otherwise 0 if colors is numeric and "grey" otherwise.

allOK Boolean flag signalling whether all eigengenes have been calculated correctly, either as principal components or as the hubgene average approximation.

allPC Boolean flag signalling whether all returned eigengenes are principal components.

isPC Boolean vector. Each component (corresponding to the columns in `eigengenes`) is TRUE if the corresponding eigengene is the first principal component and FALSE if it is the hubgene approximation or is invalid.

isHub Boolean vector. Each component (corresponding to the columns in `eigengenes`) is TRUE if the corresponding eigengene is the hubgene approximation and FALSE if it is the first principal component or is invalid.

validAEs Boolean vector. Each component (corresponding to the columns in `eigengenes`) is TRUE if the corresponding module average expression is valid.

allAEOK Boolean flag signalling whether all returned module average expressions contain valid data. Note that `returnValidOnly==TRUE` does not imply `allAEOK==TRUE`: some invalid average expressions may be returned if their corresponding eigengenes have been calculated correctly.

Author(s)

Steve Horvath <SHorvath@mednet.ucla.edu>, Peter Langfelder <Peter.Langfelder@gmail.com>

References


See Also

`svd`, `impute.knn`
**moduleMergeUsingKME**

*Merge modules and reassign genes using kME.*

**Description**

This function takes an expression data matrix (and other user-defined parameters), calculates the module membership (kME) values, and adjusts the module assignments, merging modules that are not sufficiently distinct and reassigning modules that were originally assigned suboptimally.

**Usage**

```r
moduleMergeUsingKME(
  datExpr, colorh, ME = NULL,
  threshPercent = 50, mergePercent = 25,
  reassignModules = TRUE,
  convertGrey = TRUE,
  omitColors = "grey",
  reassignScale = 1,
  threshNumber = NULL
)
```

**Arguments**

- **datExpr**: An expression data matrix, with samples as rows, genes (or probes) as column.
- **colorh**: The color vector (module assignments) corresponding to the columns of datExpr.
- **ME**: Either NULL (default), at which point the module eigengenes will be calculated, or pre-calculated module eigengenes for each of the modules, with samples as rows (corresponding to datExpr), and modules corresponding to columns (column names MUST be module colors or module colors prefixed by "ME" or "PC").
- **threshPercent**: Threshold percent of the number of genes in the module that should be included for the various analyses. For example, in a module with 200 genes, if threshPercent=50 (default), then 50 genes will be checked for reassignment and used to test whether two modules should be merged. See also threshNumber.
- **mergePercent**: If greater than this percent of the assigned genes are above the threshold are in a module other than the assigned module, then these two modules will be merged. For example, if mergePercent=25 (default), and the 70 out of 200 genes in the blue module were more highly correlated with the black module eigengene, then all genes in the blue module would be reassigned to the black module.
- **reassignModules**: If TRUE (default), genes are reassigned to the module with which they have the highest module membership (kME), but only if their kME is above the threshPercent (or threshNumber) threshold of that module.
- **convertGrey**: If TRUE (default), unassigned (grey) genes are assigned as in "reassignModules".
- **omitColors**: These are all of the module assignments which indicate genes that are not assigned to modules (default="grey"). These genes will all be assigned as "grey" by this function.
reassignScale  
A value between 0 and 1 (default) which determines how the threshPercent gets scaled for reassigning genes. Smaller values reassign more genes, but does not affect the merging process.

threshNumber  
Either NULL (default) or, if entered, every module is counted as having exactly threshNumber genes, and threshPercent if ignored. This parameter should have the effect of

Value

moduleColors  
The NEW color vector (module assignments) corresponding to the columns of datExpr, after module merging and reassignments.

mergeLog  
A log of the order in which modules were merged, for reference.

Note

Note that this function should be considered "experimental" as it has only been beta tested. Please e-mail jeremyinla@gmail.com if you have any issues with the function.

Author(s)

Jeremy Miller

Examples

```r
## First simulate some data and the resulting network dendrogram
set.seed(100)
MEturquoise = sample(1:100,50)
MEblue = sample(1:100,50)
MEbrown = sample(1:100,50)
MEyellow = sample(1:100,50)
MEgreen = c(MEyellow[1:30], sample(1:100,20))
MEred = c(MEbrown[1:20], sample(1:100,30))
#MEblack = c(MEblue [1:25], sample(1:100,25))
ME = data.frame(MEturquoise, MEblue, MEbrown, MEyellow, MEgreen, MEred, MEblack)
dat1 = simulateDatExpr(ME, 400, c(0.1,0.13,0.12,0.10,0.09,0.09,0.1), signed=TRUE)
TOM1 = TOMsimilarityFromExpr(dat1$datExpr, networkType="signed")
tree1 = fastcluster::hclust(as.dist(1-TOM1),method="average")

## Here is an example using different mergePercentages, setting an inclusive threshPercent (91)
colorh1 <- colorPlot <- labels2colors(dat1$allLabels)
merges = c(65,40,20,5)
for (m in merges)
  colorPlot = cbind(colorPlot,
    moduleMergeUsingKME(dat1$datExpr, colorh1, threshPercent=91, mergePercent=m)$moduleColors)
plotDendroAndColors(tree1, colorPlot, c("ORIG",merges), dendroLabels=FALSE)

## Here is an example using a lower reassignScale (so that more genes get reassigned)
colorh1 <- colorPlot <- labels2colors(dat1$allLabels)
merges = c(65,40,20,5)
for (m in merges)
  colorPlot = cbind(colorPlot,
    moduleMergeUsingKME(dat1$datExpr, colorh1, threshPercent=91,
    reassignScale=0.7, mergePercent=m)$moduleColors)
```
moduleNumber

Fixed-height cut of a dendrogram.

Description

Detects branches of on the input dendrogram by performing a fixed-height cut.

Usage

moduleNumber(dendro, cutHeight = 0.9, minSize = 50)

Arguments

dendro a hierarchical clustering dendrogram such as one returned by hclust.
cutHeight Maximum joining heights that will be considered.
minSize Minimum cluster size.

Details

All contiguous branches below the height cutHeight that contain at least minSize objects are assigned unique positive numerical labels; all unassigned objects are assigned label 0.

Value

A vector of numerical labels giving the assignment of each object.

Note

The numerical labels may not be sequential. See normalizeLabels for a way to put the labels into a standard order.

Author(s)

Peter Langfelder, <Peter.Langfelder@gmail.com>

See Also

hclust, cutree, normalizeLabels
Description

Calculations of module preservation statistics between independent data sets.

Usage

modulePreservation(
    multiData, multiColor, dataIsExpr = TRUE,
    networkType = "unsigned", corFnc = "cor",
    corOptions = "use = 'p'",
    referenceNetworks = 1, testNetworks = NULL,
    nPermutations = 100, includekMEallInSummary = FALSE,
    restrictSummaryForGeneralNetworks = TRUE,
    calculateQvalue = FALSE, randomSeed = 12345,
    maxGoldModuleSize = 1000, maxModuleSize = 1000,
    quickCor = 1, ccTupletSize = 2,
    calculateCor.kIMall = FALSE, calculateClusterCoeff = FALSE,
    useInterpolation = FALSE, checkData = TRUE,
    greyName = NULL, savePermutedStatistics = TRUE,
    loadPermutedStatistics = FALSE,
    permutedStatisticsFile = if (useInterpolation) "permutedStats-intrModules.RData"
        else "permutedStats-actualModules.RData",
    plotInterpolation = TRUE,
    interpolationPlotFile = "modulePreservationInterpolationPlots.pdf",
    discardInvalidOutput = TRUE,
    parallelCalculation = FALSE,
    verbose = TRUE, indent = 0)

Arguments

multiData expression data or adjacency data in the multi-set format (see checkSets). A vector of lists, one per set. Each set must contain a component data that contains the expression or adjacency data. If expression data are used, rows correspond to samples and columns to genes or probes. In case of adjacencies, each data matrix should be a symmetric matrix ith entries between 0 and 1 and unit diagonal. Each component of the outermost list should be named.
modulePreservation

**multiColor**
a list in which every component is a vector giving the module labels of genes in multiExpr. The components must be named using the same names that are used in multiExpr; these names are used to match labels to expression data sets. See details.

**dataIsExpr**
logical; if TRUE, multiData will be interpreted as expression data; if FALSE, multiData will be interpreted as adjacencies.

**networkType**
network type. Allowed values are (unique abbreviations of) "unsigned", "signed", "signed hybrid". See adjacency.

**corFnc**
character string specifying the function to be used to calculate co-expression similarity. Defaults to Pearson correlation. Another useful choice is bicor. More generally, any function returning values between -1 and 1 can be used.

**corOptions**
character string specifying additional arguments to be passed to the function given by corFnc. Use "use = 'p', method = 'spearman'" to obtain Spearman correlation.

**referenceNetworks**
a vector giving the indices of expression data to be used as reference networks. Reference networks must have their module labels given in multiColor.

**testNetworks**
a list with one component per each entry in referenceNetworks above, giving the test networks in which to evaluate module preservation for the corresponding reference network. If not given, preservation will be evaluated in all networks (except each reference network). If referenceNetworks is of length 1, testNetworks can also be a vector (instead of a list containing the single vector).

**nPermutations**
specifies the number of permutations that will be calculated in the permutation test.

**includeKMEAllInSummary**
logical: should cor.KMEall be included in the calculated summary statistics? Because KMEall takes into account all genes in the network, this statistic measures preservation of the full network with respect to the eigengene of the module. This may be undesirable, hence the default is FALSE.

**restrictSummaryForGeneralNetworks**
logical: should the summary statistics for general (not correlation) networks be restricted (density to meanAdj, connectivity to cor.KIM and cor.Adj)? The default TRUE corresponds to published work.

**calculateQvalue**
logical: should q-values (local FDR estimates) be calculated? Package qvalue must be installed for this calculation. Note that q-values may not be meaningful when the number of modules is small and/or most modules are preserved.

**randomSeed**
seed for the random number generator. If NULL, the seed will not be set. If non-NULL and the random generator has been initialized prior to the function call, the latter's state is saved and restored upon exit

**maxGoldModuleSize**
m maximum size of the "gold" module, i.e., the random sample of all network genes.

**maxModuleSize**
m maximum module size used for calculations. Modules larger than maxModuleSize will be reduced by randomly sampling maxModuleSize genes.

**quickCor**
n number between 0 and 1 specifying the handling of missing data in calculation of correlation. Zero means exact but potentially slower calculations; one means potentially faster calculations, but with potentially inaccurate results if the proportion of missing data is large. See cor for more details.
modulePreservation

ccTupletSize  tuplet size for co-clustering calculations.

calculateCor.KMall
logical: should cor.KMall be calculated? This option is only valid for adjacency input. If FALSE, cor.KMall will not be calculated, potentially saving significant amount of time if the input adjacencies are large and contain many modules.

calculateClusterCoeff
logical: should statistics based on the clustering coefficient be calculated? While these statistics may be interesting, the calculations are also computationally expensive.

checkData
logical: should data be checked for excessive number of missing entries? See goodSamplesGenesMS for details.

greyName
label used for unassigned genes. Traditionally such genes are labeled by grey color or numeric label 0. These values are the default when multiColor contains character or numeric vectors, respectively.

savePermutedStatistics
logical: should calculated permutation statistics be saved? Saved statistics may be re-used if the calculation needs to be repeated.

permutedStatisticsFile
file name to save the permutation statistics into.

loadPermutedStatistics
logical: should permutation statistics be loaded? If a previously executed calculation needs to be repeated, loading permutation study results can cut the calculation time many-fold.

useInterpolation
logical: should permutation statistics be calculated by interpolating an artificial set of evenly spaced modules? This option may potentially speed up the calculations, but it restricts calculations to density measures.

plotInterpolation
logical: should interpolation plots be saved? If interpolation is used (see useInterpolation above), the function can optionally generate diagnostic plots that can be used to assess whether the interpolation makes sense.

interpolationPlotFile
file name to save the interpolation plots into.

discardInvalidOutput
logical: should output columns containing no valid data be discarded? This option may be useful when input dataIsExpr is FALSE and some of the output statistics cannot be calculated. This option causes such statistics to be dropped from output.

parallelCalculation
logical: should calculations be done in parallel? Note that parallel calculations are turned off by default and will lead to somewhat DIFFERENT results than serial calculations because the random seed is set differently. For the calculation to actually run in parallel mode, a call to enableWGCNAThreads must be made before this function is called.

verbose
integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.

indent
indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.
Details

This function calculates module preservation statistics pair-wise between given reference sets and all other sets in `multiExpr`. Reference sets must have their corresponding module assignment specified in `multiColor`; module assignment is optional for test sets. Individual expression sets and their module labels are matched using names of the corresponding components in `multiExpr` and `multiColor`.

For each reference-test pair, the function calculates module preservation statistics that measure how well the modules of the reference set are preserved in the test set. If the `multiColor` also contains module assignment for the test set, the calculated statistics also include cross-tabulation statistics that make use of the test module assignment.

For each reference-test pair, the function only uses genes (columns of the data component of each component of `multiExpr`) that are in common between the reference and test set. Columns are matched by column names, so column names must be valid.

In addition to preservation statistics, the function also calculates several statistics of module quality, that is measures of how well-defined modules are in the reference set. The quality statistics are calculated with respect to genes in common with with a test set; thus the function calculates a set of quality statistics for each reference-test pair. This may be somewhat counter-intuitive, but it allows a direct comparison of corresponding quality and preservation statistics.

The calculated p-values are determined from the Z scores of individual measures under assumption of normality. No p-value is calculated for the Zsummary measures. Bonferoni correction to the number of tested modules. Because the p-values for strongly preserved modules are often extremely low, the function reports natural logarithms (base e) of the p-values. However, q-values are reported untransformed since they are calculated that way in package qvalue.

Missing data are removed (but see `quickCor` above).

Value

The function returns a nested list of preservation statistics. At the top level, the list components are:

- **quality**: observed values, Z scores, log p-values, Bonferroni-corrected log p-values, and (optionally) q-values of quality statistics. All logarithms are in base 10.
- **preservation**: observed values, Z scores, log p-values, Bonferroni-corrected log p-values, and (optionally) q-values of density and connectivity preservation statistics. All logarithms are in base 10.
- **accuracy**: observed values, Z scores, log p-values, Bonferroni-corrected log p-values, and (optionally) q-values of cross-tabulation statistics. All logarithms are in base 10.
- **referenceSeparability**: observed values, Z scores, log p-values, Bonferroni-corrected log p-values, and (optionally) q-values of module separability in the reference network. All logarithms are in base 10.
- **testSeparability**: observed values, Z scores, p-values, Bonferroni-corrected p-values, and (optionally) q-values of module separability in the test network. All logarithms are in base 10.
- **permutationDetails**: results of individual permutations, useful for diagnostics

All of the above are lists. The lists `quality`, `preservation`, `referenceSeparability`, and `testSeparability` each contain 4 or 5 components: observed contains observed values, Z contains the corresponding Z scores, `log.p` contains base 10 logarithms of the p-values, `log.pBonf`
contains base 10 logarithms of the Bonferoni corrected p-values, and optionally q contains the associated q-values. The list accuracy contains observed, Z, log.p, log.pBonf, optionally q, and additional components observedOverlapCounts and observedFisherPvalues that contain the observed matrices of overlap counts and Fisher test p-values.

Each of the lists observed, Z, log.p, log.pBonf, optionally q, observedOverlapCounts and observedFisherPvalues is structured as a 2-level list where the outer components correspond to reference sets and the inner components to tests sets. As an example, preservation$observed[[1]][[2]] contains the density and connectivity preservation statistics for the preservation of set 1 modules in set 2, that is set 1 is the reference set and set 2 is the test set. preservation$observed[[1]][[2]] is a data frame in which each row corresponds to a module in the reference network plus one row for the unassigned objects, and one row for a “module” that contains randomly sampled objects and that represents a whole-network average. Each column corresponds to a statistic as indicated by the column name.

**Note**

For large data sets, the permutation study may take a while (typically on the order of several hours). Use `verbose = 3` to get detailed progress report as the calculations advance.

**Author(s)**

Rui Luo and Peter Langfelder

**References**

Peter Langfelder, Rui Luo, Michael C. Oldham, and Steve Horvath, to appear

**See Also**

Network construction and module detection functions in the WGCNA package such as `adjacency`, `blockwiseModules`; rudimentary cleaning in `goodSamplesGenesMS`; the WGCNA implementation of correlation in `cor`.

---

### mtd.apply

**Apply a function to each set in a multiData structure.**

#### Description

Inspired by `lapply`, these functions apply a given function to each data component in the input `multiData` structure, and optionally simplify the result to an array if possible.

#### Usage

```r
mtd.apply(  
  # What to do
  multiData, FUN, ...,  

  # Pre-existing results and update options
  mdaExistingResults = NULL, mdaUpdateIndex = NULL,  
  mdaCopyNonData = FALSE,
)```
# Output formatting options
mdaSimplify = FALSE,
returnList = FALSE,

# Internal behaviour options
mdaVerbose = 0, mdaIndent = 0)

mtd.applyToSubset(
  # What to do
  multiData, FUN, ...

  # Which rows and cols to keep
  mdaRowIndex = NULL, mdaColIndex = NULL,

  # Pre-existing results and update options
  mdaExistingResults = NULL, mdaUpdateIndex = NULL,
  mdaCopyNonData = FALSE,

  # Output formatting options
  mdaSimplify = FALSE,
  returnList = FALSE,

  # Internal behaviour options
  mdaVerbose = 0, mdaIndent = 0)

Arguments

multiData A multiData structure to apply the function over
FUN Function to be applied.
... Other arguments to the function FUN.
mdaRowIndex If given, must be a list of the same length as multiData. Each element must be
a logical or numeric vector that specifies rows in each data component to select
before applying the function.
mdaColIndex A logical or numeric vector that specifies columns in each data component to
select before applying the function.
mdaExistingResults Optional list that contains previously calculated results. This can be useful if
only a few sets in multiData have changed and recalculating the unchanged
ones is computationally expensive. If not given, all calculations will be per-
formed. If given, components of this list are copied into the output. See mdaUpdateIndex
for which components are re-calculated by default.
mdaUpdateIndex Optional specification of which sets in multiData the calculation should actu-
ally be carried out. This argument has an effect only if mdaExistingResults
is non-NULL. If the length of mdaExistingResults (call the length ‘k’) is less
than the number of sets in multiData, the function assumes that the existing
results correspond to the first ‘k’ sets in multiData and the rest of the sets are
automatically calculated, irrespective of the setting of mdaUpdateIndex. The
argument mdaUpdateIndex can be used to specify re-calculation of some (or
all) of the results that already exist in mdaExistingResults.
mdaCopyNonData Logical: should non-data components of multiData be copied into the output? Note that the copying is incompatible with simplification; enabling both will trigger an error.

mdaSimplify Logical: should the result be simplified to an array, if possible? Note that this may lead to errors; if so, disable simplification.

returnList Logical: should the result be turned into a list (rather than a multiData structure)? Note that this is incompatible with simplification: if mdaSimplify is TRUE, this argument is ignored.

mdaVerbose Integer specifying whether progress diagnostics should be printed out. Zero means silent, increasing values will lead to more diagnostic messages.

mdaIndent Integer specifying the indentation of the printed progress messages. Each unit equals two spaces.

Details

A multiData structure is intended to store (the same type of) data for multiple, possibly independent, realizations (for example, expression data for several independent experiments). It is a list where each component corresponds to an (independent) data set. Each component is in turn a list that can hold various types of information but must have a data component. In a "strict" multiData structure, the data components are required to each be a matrix or a data frame and have the same number of columns. In a "loose" multiData structure, the data components can be anything (but for most purposes should be of comparable type and content).

mtd.apply works on any "loose" multiData structure; mtd.applyToSubset assumes (and checks for) a "strict" multiData structure.

Value

A multiData structure containing the results of the supplied function on each data component in the input multiData structure. Other components are simply copied.

Author(s)

Peter Langfelder

See Also

multiData to create a multiData structure; mtd.applyToSubset for applying a function to a subset of a multiData structure; mtd.mapply for vectorizing over several arguments.

mtd.mapply Apply a function to elements of given multiData structures.

Description

Inspired by mapply, this function applies a given function to each data component in the input multiData arguments, and optionally simplify the result to an array if possible.
Usage

mtd.mapply(
    # What to do
    FUN, ..., MoreArgs = NULL,

    # How to interpret the input
    mdma.argIsMultiData = NULL,

    # Copy previously known results?
    mdmaExistingResults = NULL, mdmaUpdateIndex = NULL,

    # How to format output
    mdmaSimplify = FALSE,
    returnList = FALSE,

    # Options controlling internal behaviour
    mdma.doCollectGarbage = FALSE,
    mdmaVerbose = 0, mdmaIndent = 0)

Arguments

FUN Function to be applied.

... Arguments to be vectorized over. These can be multiData structures or simple vectors (e.g., lists).

MoreArgs A named list that specifies the scalar arguments (if any) to FUN.

mdma.argIsMultiData Optional specification whether arguments are multiData structures. A logical vector where each component corresponds to one entry of ... If not given, multiData status will be determined using isMultiData with argument strict=FALSE.

mdmaExistingResults Optional list that contains previously calculated results. This can be useful if only a few sets in multiData have changed and recalculating the unchanged ones is computationally expensive. If not given, all calculations will be performed. If given, components of this list are copied into the output. See mdmUpdateIndex for which components are re-calculated by default.

mdmaUpdateIndex Optional specification of which sets in multiData the calculation should actually be carried out. This argument has an effect only if mdmaExistingResults is non-NULL. If the length of mdmaExistingResults (call the length 'k') is less than the number of sets in multiData, the function assumes that the existing results correspond to the first 'k' sets in multiData and the rest of the sets are automatically calculated, irrespective of the setting of mdmaUpdateIndex. The argument mdmaUpdateIndex can be used to specify re-calculation of some (or all) of the results that already exist in mdmaExistingResults.

mdmaSimplify Logical: should simplification of the result to an array be attempted? The simplification is fragile and can produce unexpected errors; use the default FALSE if that happens.
**mtd.rbindSelf**

**Details**

A multiData structure is intended to store (the same type of) data for multiple, possibly independent, realizations (for example, expression data for several independent experiments). It is a list where each component corresponds to an (independent) data set. Each component is in turn a list that can hold various types of information but must have a data component. In a "strict" multiData structure, the data components are required to each be a matrix or a data frame and have the same number of columns. In a "loose" multiData structure, the data components can be anything (but for most purposes should be of comparable type and content).

This function applies the function FUN to each data component of those arguments in ... that are multiData structures in the "loose" sense, and to each component of those arguments in ... that are not multiData structures.

**Value**

A multiData structure containing (as the data components) the results of FUN. If simplification is successful, an array instead.

**Author(s)**

Peter Langfelder

**See Also**

`multidata` to create a multiData structure;
`multidataApply` for application of a function to a single multiData structure.

---

**mtd.rbindSelf**: Turn a multiData structure into a single matrix or data frame.

**Description**

This function "rbinds" the data components of all sets in the input into a single matrix or data frame.

**Usage**

mtd.rbindSelf(multiData)

**Arguments**

- **returnList**: Logical: should the result be turned into a list (rather than a multiData structure)? Note that this is incompatible with simplification: if mdasimplify is TRUE, this argument is ignored.
- **mdma.doCollectGarbage**: Should garbage collection be forced after each application of FUN?
- **mdmaVerbose**: Integer specifying whether progress diagnostics should be printed out. Zero means silent, increasing values will lead to more diagnostic messages.
- **mdmaIndent**: Integer specifying the indentation of the printed progress messages. Each unit equals two spaces.
Arguments

multiData A multiData structure.

Details

A multiData structure is intended to store (the same type of) data for multiple, possibly independent, realizations (for example, expression data for several independent experiments). It is a list where each component corresponds to an (independent) data set. Each component is in turn a list that can hold various types of information but must have a data component. In a "strict" multiData structure, the data components are required to each be a matrix or a data frame and have the same number of columns. In a "loose" multiData structure, the data components can be anything (but for most purposes should be of comparable type and content).

This function requires a "strict" multiData structure.

Value

A single matrix or data frame containing the "rbinded" result.

Author(s)

Peter Langfelder

See Also

multiData to create a multiData structure;
rbind for various subtleties of the row binding operation.

mtd.setAttr

Set attributes on each component of a multiData structure

Description

Set attributes on each data component of a multiData structure

Usage

mtd.setAttr(multiData, attribute, valueList)

Arguments

multiData A multiData structure.
attribute Name for the attribute to be set
valueList List that gives the attribute value for each set in the multiData structure.

Value

The input multiData with the attribute set on each data component.

Author(s)

Peter Langfelder
See Also

- `multidata` to create a multiData structure;
- `isMultiData` for a description of the multiData structure.

---

### mtd.setColnames

**Get and set column names in a multiData structure.**

**Description**

Get and set column names on each data component in a multiData structure.

**Usage**

```r
mtd.colnames(multiData)
mtd.setColnames(multiData, colnames)
```

**Arguments**

- `multiData` A multiData structure
- `colnames` A vector (coercible to character) of column names.

**Details**

A multiData structure is intended to store (the same type of) data for multiple, possibly independent, realizations (for example, expression data for several independent experiments). It is a list where each component corresponds to an (independent) data set. Each component is in turn a list that can hold various types of information but must have a data component. In a "strict" multiData structure, the data components are required to each be a matrix or a data frame and have the same number of columns. In a "loose" multiData structure, the data components can be anything (but for most purposes should be of comparable type and content).

The `mtd.colnames` and `mtd.setColnames` assume (and checks for) a "strict" multiData structure.

**Value**

- `mtd.colnames` returns the vector of column names of the data component. The function assumes the column names in all sets are the same.
- `mtd.setColnames` returns the multiData structure with the column names set in all data components.

**Author(s)**

Peter Langfelder

**See Also**

- `multidata` to create a multiData structure.
mtd.simplify

If possible, simplify a multiData structure to a 3-dimensional array.

Description
This function attempts to put all data components into a 3-dimensional array, with the last dimension corresponding to the sets. This is only possible if all data components are matrices or data frames with the same dimension.

Usage
mtd.simplify(multiData)

Arguments
multiData A multiData structure in the "strict" sense (see below).

Details
A multiData structure is intended to store (the same type of) data for multiple, possibly independent, realizations (for example, expression data for several independent experiments). It is a list where each component corresponds to an (independent) data set. Each component is in turn a list that can hold various types of information but must have a data component. In a "strict" multiData structure, the data components are required to each be a matrix or a data frame and have the same number of columns. In a "loose" multiData structure, the data components can be anything (but for most purposes should be of comparable type and content).

This function assumes a "strict" multiData structure.

Value
A 3-dimensional array collecting all data components.

Note
The function is relatively fragile and may fail. Use at your own risk.

Author(s)
Peter Langfelder

See Also
multiData to create a multiData structure;
multiData2list for converting multiData structures to plain lists.
mtd.subset  Subset rows and columns in a multiData structure

Description

The function restricts each data component to the given columns and rows.

Usage

mtd.subset(
  # Input
  multiData,

  # Rows and columns to keep
  rowIndex = NULL, colIndex = NULL,

  # Strict or permissive checking of structure?
  permissive = FALSE,

  # Output formatting options
  drop = FALSE)

Arguments

multiData  A multiData structure.
rowIndex   A list in which each component corresponds to a set and is a vector giving the rows to be retained in that set. All indexing methods recognized by R can be used (numeric, logical, negative indexing, etc). If NULL, all columns will be retained in each set. Note that setting individual elements of rowIndex to NULL will lead to errors.
colIndex   A vector giving the columns to be retained. All indexing methods recognized by R can be used (numeric, logical, negative indexing, etc). In addition, column names of the retained columns may be given; if a given name cannot be matched to a column, an error will be thrown. If NULL, all columns will be retained.
permissive logical: should the function tolerate "loose" multiData input? Note that the subsetting may lead to cryptic errors if the input multiData does not follow the "strict" format.
drop      logical: should dimensions with extent 1 be dropped?

Details

A multiData structure is intended to store (the same type of) data for multiple, possibly independent, realizations (for example, expression data for several independent experiments). It is a list where each component corresponds to an (independent) data set. Each component is in turn a list that can hold various types of information but must have a data component. In a "strict" multiData structure, the data components are required to each be a matrix or a data frame and have the same number of columns. In a "loose" multiData structure, the data components can be anything (but for most purposes should be of comparable type and content).

This function assumes a "strict" multiData structure unless permissive is TRUE.
multiData

Value
A multiData structure containing the selected rows and columns. Note that result always retains its dimension and other attributes.

Author(s)
Peter Langfelder

See Also
multiData to create a multiData structure.

---

multiData
Create a multiData structure.

Description
This function creates a multiData structure by storing its input arguments as the 'data' components.

Usage
multiData(...)

Arguments
... Arguments to be stored in the multiData structure.

Details
A multiData structure is intended to store (the same type of) data for multiple, possibly independent, realizations (for example, expression data for several independent experiments). It is a list where each component corresponds to an (independent) data set. Each component is in turn a list that can hold various types of information but must have a data component. In a "strict" multiData structure, the data components are required to each be a matrix or a data frame and have the same number of columns. In a "loose" multiData structure, the data components can be anything (but for most purposes should be of comparable type and content).

Value
The resulting multiData structure.

Author(s)
Peter Langfelder

See Also
multiData2list for converting a multiData structure to a list; list2multiData for an alternative way of creating a multiData structure; mtd.apply, mtd.applyToSubset, mtd.mapply for ways of applying a function to each component of a multiData structure.
**Examples**

data1 = matrix(rnorm(100), 20, 5);
data2 = matrix(rnorm(50), 10, 5);

md = multiData(set1 = data1, set2 = data2);

checkSets(md)

---

**Description**

This function calculates eigengene significance and the associated significance statistics (p-values, q-values etc) across several data sets.

**Usage**

```r
multiData.eigengeneSignificance(
  multiData, multiTrait,
  moduleLabels, multiEigengenes = NULL,
  useModules = NULL,
  corAndPvalueFnc = corAndPvalue, corOptions = list(),
  corComponent = "cor",
  getQvalues = FALSE, setNames = NULL,
  excludeGrey = TRUE, greyLabel = ifelse(is.numeric(moduleLabels), 0, "grey")
)
```

**Arguments**

- `multiData` Expression data (or other data) in multi-set format (see `checkSets`). A vector of lists; in each list there must be a component named `data` whose content is a matrix or dataframe or array of dimension 2.
- `multiTrait` Trait or outcome data in multi-set format. Only one trait is allowed; consequently, the `data` component of each component list can be either a vector or a dataframe (matrix, array of dimension 2).
- `moduleLabels` Module labels: one label for each gene in `multiExpr`.
- `multiEigengenes` Optional eigengenes of modules specified in `moduleLabels`. If not given, will be calculated from `multiExpr`.
- `useModules` Optional specification of module labels to which the analysis should be restricted. This could be useful if there are many modules, most of which are not interesting. Note that the "grey" module cannot be used with `useModules`.
- `corAndPvalueFnc` Function that calculates associations between expression profiles and eigengenes. See details.
- `corOptions` List giving additional arguments to function `corAndPvalueFnc`. See details.
- `corComponent` Name of the component of output of `corAndPvalueFnc` that contains the actual correlation.
multiSetMEs

getqvalues logical: should q-values (estimates of FDR) be calculated?

setNames names for the input sets. If not given, will be taken from names(multiExpr). If those are NULL as well, the names will be "Set_1", "Set_2", ....

excludeGrey logical: should the grey module be excluded from the kME tables? Since the grey module is typically not a real module, it makes little sense to report kME values for it.

greyLabel label that labels the grey module.

Details

This is a convenience function that calculates module eigengene significances (i.e., correlations of module eigengenes with a given trait) across all sets in a multi-set analysis. Also returned are p-values, Z scores, numbers of present (i.e., non-missing) observations for each significance, and optionally the q-values (false discovery rates) corresponding to the p-values.

The function corAndPvalueFnc is currently expected to accept arguments x (gene expression profiles) and y (eigengene expression profiles). Any additional arguments can be passed via corOptions.

The function corAndPvalueFnc should return a list which at the least contains (1) a matrix of associations of genes and eigengenes (this component should have the name given by corComponent), and (2) a matrix of the corresponding p-values, named "p" or "p.value". Other components are optional but for full functionality should include (3) nObs giving the number of observations for each association (which is the number of samples less number of missing data - this can in principle vary from association to association), and (4) Z giving a Z static for each observation. If these are missing, nObs is calculated in the main function, and calculations using the Z statistic are skipped.

Value

A list containing the following components. Each component is a matrix in which the rows correspond to module eigengenes and columns to data sets. Row and column names are set appropriately.

eigengeneSignificance Module eigengene significance.

p.value p-values (returned by corAndPvalueFnc).

q.value q-values corresponding to the p-values above. Only returned in input getWvalues is TRUE.

Z Z statistics (if returned by corAndPvalueFnc).

nObservations Number of non-missing observations in each correlation/p-value.

Author(s)

Peter Langfelder

multiSetMEs Calculate module eigengenes.

Description

Calculates module eigengenes for several sets.
Usage

```r
multiSetMEs(exprData,
    colors,
    universalColors = NULL,
    useSets = NULL,
    useGenes = NULL,
    impute = TRUE,
    npc = 1,
    align = "along average",
    excludeGrey = FALSE,
    grey = if (is.null(universalColors)) {
        if (is.numeric(colors)) 0 else "grey"
    } else
        if (is.numeric(universalColors)) 0 else "grey",
    subHubs = TRUE,
    trapErrors = FALSE,
    returnValidOnly = trapErrors,
    softPower = 6,
    verbose = 1, indent = 0)
```

Arguments

- `exprData` Expression data in a multi-set format (see `checkSets`). A vector of lists, with each list corresponding to one microarray dataset and expression data in the component data, that is `expr[[set]][sample, probe]` is the expression of probe `probe` in sample `sample` in dataset `set`. The number of samples can be different between the sets, but the probes must be the same.

- `colors` A matrix of dimensions (number of probes, number of sets) giving the module assignment of each gene in each set. The color "grey" is interpreted as unassigned.

- `universalColors` Alternative specification of module assignment. A single vector of length (number of probes) giving the module assignment of each gene in all sets (that is the modules are common to all sets). If given, takes precedence over color.

- `useSets` If calculations are requested in (a) selected set(s) only, the set(s) can be specified here. Defaults to all sets.

- `useGenes` Can be used to restrict calculation to a subset of genes (the same subset in all sets). If given, validColors in the returned list will only contain colors for the genes specified in `useGenes`.

- `impute` Logical. If TRUE, expression data will be checked for the presence of NA entries and if the latter are present, numerical data will be imputed, using function `impute.knn` and probes from the same module as the missing datum. The function `impute.knn` uses a fixed random seed giving repeatable results.

- `npc` Number of principal components to be calculated. If only eigengenes are needed, it is best to set it to 1 (default). If variance explained is needed as well, use value `NULL`. This will cause all principal components to be computed, which is slower.

- `align` Controls whether eigengenes, whose orientation is undetermined, should be aligned with average expression (`align = "along average"`, the default) or left as they are (`align = ""`). Any other value will trigger an error.
multiSetMEs

excludeGrey Should the improper module consisting of 'grey' genes be excluded from the eigengenes?

grey Value of colors or universalColors (whichever applies) designating the improper module. Note that if the appropriate colors argument is a factor of numbers, the default value will be incorrect.

subHubs Controls whether hub genes should be substituted for missing eigengenes. If TRUE, each missing eigengene (i.e., eigengene whose calculation failed and the error was trapped) will be replaced by a weighted average of the most connected hub genes in the corresponding module. If this calculation fails, or if subHubs==FALSE, the value of trapErrors will determine whether the offending module will be removed or whether the function will issue an error and stop.

trapErrors Controls handling of errors from that may arise when there are too many NA entries in expression data. If TRUE, errors from calling these functions will be trapped without abnormal exit. If FALSE, errors will cause the function to stop. Note, however, that subHubs takes precedence in the sense that if subHubs==TRUE and trapErrors==FALSE, an error will be issued only if both the principal component and the hubgene calculations have failed.

returnValidOnly Boolean. Controls whether the returned data frames of module eigengenes contain columns corresponding only to modules whose eigengenes or hub genes could be calculated correctly in every set (TRUE), or whether the data frame should have columns for each of the input color labels (FALSE).

softPower The power used in soft-thresholding the adjacency matrix. Only used when the hubgene approximation is necessary because the principal component calculation failed. It must be non-negative. The default value should only be changed if there is a clear indication that it leads to incorrect results.

verbose Controls verbosity of printed progress messages. 0 means silent, up to (about) 5 the verbosity gradually increases.

indent A single non-negative integer controlling indentation of printed messages. 0 means no indentation, each unit above that adds two spaces.

Details

This function calls moduleEigengenes for each set in exprData.

Module eigengene is defined as the first principal component of the expression matrix of the corresponding module. The calculation may fail if the expression data has too many missing entries. Handling of such errors is controlled by the arguments subHubs and trapErrors. If subHubs==TRUE, errors in principal component calculation will be trapped and a substitute calculation of hubgenes will be attempted. If this fails as well, behaviour depends on trapErrors: if TRUE, the offending module will be ignored and the return value will allow the user to remove the module from further analysis; if FALSE, the function will stop. If universalColors is given, any offending module will be removed from all sets (see validMEs in return value below).

From the user’s point of view, setting trapErrors=FALSE ensures that if the function returns normally, there will be a valid eigengene (principal component or hubgene) for each of the input colors. If the user sets trapErrors=TRUE, all calculational (but not input) errors will be trapped, but the user should check the output (see below) to make sure all modules have a valid returned eigengene.

While the principal component calculation can fail even on relatively sound data (it does not take all that many "well-placed" NA to torpedo the calculation), it takes many more irregularities in the data for the hubgene calculation to fail. In fact such a failure signals there likely is something seriously wrong with the data.
Value

A vector of lists similar in spirit to the input exprData. For each set there is a list with the following components:

data

Module eigengenes in a data frame, with each column corresponding to one eigengene. The columns are named by the corresponding color with an "ME" prepended, e.g., MEturquoise etc. Note that, when trapErrors == TRUE and returnValidOnly==FALSE, this data frame also contains entries corresponding to removed modules, if any. (validMEs below indicates which eigengenes are valid and allOK whether all module eigengens were successfully calculated.)

averageExpr

If align == "along average", a dataframe containing average normalized expression in each module. The columns are named by the corresponding color with an "AE" prepended, e.g., AEturquoise etc.

varExplained

A dataframe in which each column corresponds to a module, with the component varExplained[PC, module] giving the variance of module explained by the principal component no. PC. This is only accurate if all principal components have been computed (inputnpc = NULL). At most 5 principal components are recorded in this dataframe.

npc

A copy of the input npc.

validMEs

A boolean vector. Each component (corresponding to the columns in data) is TRUE if the corresponding eigengene is valid, and FALSE if it is invalid. Valid eigengenes include both principal components and their hubgene approximations. When returnValidOnly==FALSE, by definition all returned eigengenes are valid and the entries of validMEs are all TRUE.

validColors

A copy of the input colors (universalColors if set, otherwise colors[set]) with entries corresponding to invalid modules set to grey if given, otherwise 0 if the appropriate input colors are numeric and "grey" otherwise.

allOK

Boolean flag signalling whether all eigengenes have been calculated correctly, either as principal components or as the hubgene approximation. If universalColors is set, this flag signals whether all eigengenes are valid in all sets.

allPC

Boolean flag signalling whether all returned eigengenes are principal components. This flag (as well as the subsequent ones) is set independently for each set.

isPC

Boolean vector. Each component (corresponding to the columns in eigengenes) is TRUE if the corresponding eigengene is the first principal component and FALSE if it is the hubgene approximation or is invalid.

isHub

Boolean vector. Each component (corresponding to the columns in eigengenes) is TRUE if the corresponding eigengene is the hubgene approximation and FALSE if it is the first principal component or is invalid.

validAEs

Boolean vector. Each component (corresponding to the columns in eigengenes) is TRUE if the corresponding module average expression is valid.

allAEOK

Boolean flag signalling whether all returned module average expressions contain valid data. Note that returnValidOnly==TRUE does not imply allAEOK==TRUE: some invalid average expressions may be returned if their corresponding eigengenes have been calculated correctly.

Author(s)

Peter Langfelder, <Peter.Langfelder@gmail.com>
multiUnion

See Also

moduleEigengenes

multiUnion  Union and intersection of multiple sets

Description

Union and intersection of multiple sets. These function generalize the standard functions \texttt{union} and \texttt{intersect}.

Usage

\begin{verbatim}
multiUnion(setList)
multiIntersect(setList)
\end{verbatim}

Arguments

- setList: A list containing the sets to be performed upon.

Value

The union or intersection of the given sets.

Author(s)

Peter Langfelder

See Also

The "standard" functions \texttt{union} and \texttt{intersect}.

mutualInfoAdjacency  Calculate weighted adjacency matrices based on mutual information

Description

The function calculates different types of weighted adjacency matrices based on the mutual information between vectors (corresponding to the columns of the input data frame datE). The mutual information between pairs of vectors is divided by an upper bound so that the resulting normalized measure lies between 0 and 1.

Usage

\begin{verbatim}
mutualInfoAdjacency(
  datE,
  discretizeColumns = TRUE,
  entropyEstimationMethod = "MM",
  numberBins = NULL)
\end{verbatim}
mutualInfoAdjacency

Arguments

date

date is a data frame or matrix whose columns correspond to variables and whose rows correspond to measurements. For example, the columns may correspond to genes while the rows correspond to microarrays. The number of nodes in the mutual information network equals the number of columns of date.

discretizeColumns

is a logical variable. If it is set to TRUE then the columns of date will be discretized into a user-defined number of bins (see numberBins).

entropyEstimationMethod

takes a text string for specifying the entropy and mutual information estimation method. If entropyEstimationMethod="MM" then the Miller-Madow asymptotic bias corrected empirical estimator is used. If entropyEstimationMethod="ML" the maximum likelihood estimator (also known as plug-in or empirical estimator) is used. If entropyEstimationMethod="shrink", the shrinkage estimator of a Dirichlet probability distribution is used. If entropyEstimationMethod="SG", the Schurmann-Grassberger estimator of the entropy of a Dirichlet probability distribution is used.

numberBins

is an integer larger than 0 which specifies how many bins are used for the discretization step. This argument is only relevant if discretizeColumns has been set to TRUE. By default numberBins is set to sqrt(m) where m is the number of samples, i.e. the number of rows of date. Thus the default is numberBins=sqrt(nrow(date)).

Details

The function inputs a data frame date and outputs a list whose components correspond to different weighted network adjacency measures defined between the columns of date. Make sure to install the following R packages entropy, minet, infotheo since the function mutualInfoAdjacency makes use of the entropy function from the R package entropy (Hausser and Strimmer 2008) and functions from the minet and infotheo package (Meyer et al 2008). A weighted network adjacency matrix is a symmetric matrix whose entries take on values between 0 and 1. Each weighted adjacency matrix contains scaled versions of the mutual information between the columns of the input data frame date. We assume that date contains numeric values which will be discretized unless the user chooses the option discretizeColumns=FALSE. The raw (unscaled) mutual information and entropy measures have units "nat", i.e. natural logarithms are used in their definition (base e=2.71...).

Several mutual information estimation methods have been proposed in the literature (reviewed in Hausser and Strimmer 2008, Meyer et al 2008). While mutual information networks allows one to detect non-linear relationships between the columns of date, they may overfit the data if relatively few observations are available. Thus, if the number of rows of date is smaller than say 200, it may be better to fit a correlation using the function adjacency.

Value

The function outputs a list with the following components:

Entropy

is a vector whose components report entropy estimates of each column of date. The natural logarithm (base e) is used in the definition. Using the notation from the Wikipedia entry (http://en.wikipedia.org/wiki/Mutual_information), this vector contains the values Hx where x corresponds to a column in date.

MutualInformation

is a symmetric matrix whose entries contain the pairwise mutual information measures between the columns of date. The diagonal of the matrix MutualInformation equals Entropy. In general, the entries of this matrix can be larger than 1, i.e.
this is not an adjacency matrix. Using the notation from the Wikipedia entry, this matrix contains the mutual information estimates $I(X;Y)$

AdjacencySymmetricUncertainty

is a weighted adjacency matrix whose entries are based on the mutual information. Using the notation from the Wikipedia entry, this matrix contains the mutual information estimates $\text{AdjacencySymmetricUncertainty} = 2*I(X;Y)/(H(X)+H(Y))$. Since $I(X;X)=H(X)$, the diagonal elements of AdjacencySymmetricUncertainty equal 1. In general the entries of this symmetric matrix AdjacencySymmetricUncertainty lie between 0 and 1.

AdjacencyUniversalVersion1

is a weighted adjacency matrix that is a simple function of the AdjacencySymmetricUncertainty. Specifically, $\text{AdjacencyUniversalVersion1} = \text{AdjacencySymmetricUncertainty}/(2-\text{AdjacencySymmetricUncertainty})$. Note that $f(x)=x/(2-x)$ is a monotonically increasing function on the unit interval $[0,1]$ whose values lie between 0 and 1. The reason why we call it the universal adjacency is that $\text{dissUA}=1-\text{AdjacencyUniversalVersion1}$ turns out to be a universal distance function, i.e. it satisfies the properties of a distance (including the triangle inequality) and it takes on a small value if any other distance measure takes on a small value (Kraskov et al 2003).

AdjacencyUniversalVersion2

is a weighted adjacency matrix for which $\text{dissUVersion2}=1-\text{AdjacencyUniversalVersion2}$ is also a universal distance measure. Using the notation from Wikipedia, the entries of the symmetric matrix AdjacencyUniversalVersion2 are defined as follows $\text{AdjacencyUniversalVersion2} = I(X;Y)/\max(H(X),H(Y))$.

Author(s)

Steve Horvath, Lin Song, Peter Langfelder

References


See Also

adjacency

Examples

# Load requisite packages. These packages are considered "optional", # so WGCNA does not load them automatically.
if (require(infotheo, quietly = TRUE) && 
   require(minet, quietly = TRUE) && 
   require(entropy, quietly = TRUE))
{
  # Example can be executed.
#Simulate a data frame datE which contains 5 columns and 50 observations
m=50
x1=rnorm(m)
r=.5; x2=r*x1+sqrt(1-r*r)*rnorm(m)
r=.3; x3=r*(x1-.5)*2+sqrt(1-r*r)*rnorm(m)
x4=rnorm(m)
r=.3; x5=r*x4+sqrt(1-r*r)*rnorm(m)
datE=data.frame(x1,x2,x3,x4,x5)

#calculate entropy, mutual information matrix and weighted adjacency
# matrices based on mutual information.
MIadj=mutualInfoAdjacency(datE=datE)
}
else
printFlush(paste("Please install packages infotheo, minet and entropy",
"before running this example."));

nearestCentroidPredictor

Nearest centroid predictor

Description

Nearest centroid predictor for binary (i.e., two-outcome) data. Implements a whole host of options and improvements such as accounting for within-class heterogeneity using sample networks, various ways of feature selection and weighing etc.

Usage

nearestCentroidPredictor(

# Input training and test data
x, y,
xtest = NULL,

# Feature weights and selection criteria
featureSignificance = NULL,
assocFnc = "cor", assocOptions = "use = 'p'",
assocCut.hi = NULL, assocCut.lo = NULL,
nFeatures.hi = 10, nFeatures.lo = 10,
weighFeaturesByAssociation = 0,
scaleFeatureMean = TRUE, scaleFeatureVar = TRUE,

# Predictor options
centroidMethod = c("mean", "eigensample"),
simFnc = "cor", simOptions = "use = 'p'",
useQuantile = NULL,
sampleWeights = NULL,
weighSimByPrediction = 0,

# What should be returned
CVfold = 0, returnFactor = FALSE,
Arguments

x  Training features (predictive variables). Each column corresponds to a feature and each row to an observation.

y  The response variable. Can be a single vector or a matrix with arbitrary many columns. Number of rows (observations) must equal to the number of rows (observations) in x.

xtest Optional test set data. A matrix of the same number of columns (i.e., features) as x. If test set data are not given, only the prediction on training data will be returned.

featuresignificance Optional vector of feature significance for the response variable. If given, it is used for feature selection (see details). Should preferably be signed, that is features can have high negative significance.

assocfnc Character string specifying the association function. The association function should behave roughly as \texttt{link{cor}} in that it takes two arguments (a matrix and a vector) plus options and returns the vector of associations between the columns of the matrix and the vector. The associations may be signed (i.e., negative or positive).

assocoptions Character string specifying options to the association function.

assoccutNhi Association (or featureSignificance) threshold for including features in the predictor. Features with associaton higher than assoccutNhi will be included. If not given, the threshold method will not be used; instead, a fixed number of features will be included as specified by nFeatures.hi and nFeatures.lo.

assoccutNlo Association (or featureSignificance) threshold for including features in the predictor. Features with associaton lower than assoccutNlo will be included. If not given, defaults to -assoccut.hi. If assoccut.hi is NULL, the threshold method will not be used; instead, a fixed number of features will be included as specified by nFeatures.hi and nFeatures.lo.

nFeatures.hi Number of highest-associated features (or features with highest featureSignificance) to include in the predictor. Only used if assoccut.hi is NULL.

nFeatures.lo Number of lowest-associated features (or features with highest featureSignificance) to include in the predictor. Only used if assoccut.hi is NULL.

weighFeaturesByAssociation (Optional) power to downweigh features that are less associated with the response. See details.

scaleFeatureMean Logical: should the training features be scaled to mean zero? Unless there are good reasons not to scale, the features should be scaled.

scaleFeatureVar Logical: should the training features be scaled to unit variance? Again, unless there are good reasons not to scale, the features should be scaled.

centroidMethod One of “mean” and “eigensample”. specifies how the centroid should be calculated. “mean” takes the mean across all samples (or all samples within a sample module, if sample networks are used), whereas “eigensample” calculates the first principal component of the feature matrix and uses that as the centroid.
nearestCentroidPredictor

simFnc  
Character string giving the similarity function for measuring the similarity between test samples and centroids. This function should behave roughly like the function `cor` in that it takes two arguments (x, y) and calculates the pair-wise similarities between columns of x and y. For convenience, the value "dist" is treated specially: the Euclidean distance between the columns of x and y is calculated and its negative is returned (so that smallest distance corresponds to highest similarity). Since values of this function are only used for ranking centroids, its values are not restricted to be positive or within certain bounds.

simOptions  
Character string specifying the options to the similarity function.

useQuantile  
If non-NULL, the "nearest quantiloid" will be used instead of the nearest centroid. See details.

sampleWeights  
Optional specification of sample weights. Useful for example if one wants to explore boosting.

weighSimByPrediction  
(Optional) power to downweight features that are not well predicted between training and test sets. See details.

CVfold  
Non-negative integer specifying cross-validation. Zero means no cross-validation will be performed. values above zero specify the number of samples to be considered test data for each step of cross-validation.

returnFactor  
Logical: should a factor be returned?

randomSeed  
Integer specifying the seed for the random number generator. If NULL, the seed will not be set. See `set.seed`.

verbose  
Integer controlling how verbose the diagnostic messages should be. Zero means silent.

indent  
Indentation for the diagnostic messages. Zero means no indentation, each unit adds two spaces.

**Details**

Nearest centroid predictor works by forming a representative profile (centroid) across features for each class from the training data, then assigning each test sample to the class of the nearest representative profile. The representative profile can be formed either as mean or as the first principal component ("eigensample"; this choice is governed by the option `centroidMethod`).

When the number of features is large and only a small fraction is likely to be associated with the outcome, feature selection can be used to restrict the features that actually enter the centroid. Feature selection can be based either on their association with the outcome calculated from the training data using `assocFnc`, or on user-supplied feature significance (e.g., derived from literature, argument `featureSignificance`). In either case, features can be selected by high and low association thresholds or by taking a fixed number of highest- and lowest-associated features.

As an alternative to centroids, the predictor can also assign test samples based on a given quantile of the distances from the training samples in each class (argument `useQuantile`). This may be advantageous if the samples in each class form irregular clusters. Note that setting `useQuantile=0` (i.e., using minimum distance in each class) essentially gives a nearest neighbor predictor: each test sample will be assigned to the class of its nearest training neighbor.

If features exhibit non-trivial correlations among themselves (such as, for example, in gene expression data), one can attempt to down-weight features that do not exhibit the same correlation in the test set. This is done by using essentially the same predictor to predict `features` from all other features in the test data (using the training data to train the feature predictor). Because test features are known, the prediction accuracy can be evaluated. If a feature is predicted badly (meaning the
error in the test set is much larger than the error in the cross-validation prediction in training data),
it may mean that its quality in the training or test data is low (for example, due to excessive noise or
outliers). Such features can be downweighed using the argument weighByPrediction. The extra
factor is \( \min(1, (\text{root mean square prediction error in test set})/(\text{root mean square cross-validation}
\text{prediction error in the training data})^{\text{weighByPrediction}}) \), that is it is never bigger than 1.

Unless the features’ mean and variance can be ascribed clear meaning, the (training) features should
be scaled to mean 0 and variance 1 before the centroids are formed.

The function implements a basic option for removal of spurious effects in the training and test data,
by removing a fixed number of leading principal components from the features. This sometimes
leads to better prediction accuracy but should be used with caution.

If samples within each class are heterogeneous, a single centroid may not represent each class well.
This function can deal with within-class heterogeneity by clustering samples (separately in each
class), then using a one representative (mean, eigensample) or quantile for each cluster in each
class to assign test samples. Various similarity measures, specified by adjFnc, can be used to
construct the sample network adjacency. Similarly, the user can specify a clustering function using
clusteringFnc. The requirements on the clustering function are described in a separate section
below.

Value

A list with the following components:

- **predicted** The back-substitution prediction in the training set.
- **predictedTest** Prediction in the test set.
- **featureSignificance** A vector of feature significance calculated by assocFnc or a copy of the input
featureSignificance if the latter is non-NULL.
- **selectedFeatures** A vector giving the indices of the features that were selected for the predictor.
- **centroidProfile** The representative profiles of each class (or cluster). Only returned in useQuntile
is NULL.
- **testSample2centroidSimilarities** A matrix of calculated similarities between the test samples and class/cluster
centroids.
- **featureValidationWeights** A vector of validation weights (see Details) for the selected features. If weighFeaturesByValidation
is 0, a unit vector is used and returned.
- **CVpredicted** Cross-validation prediction on the training data. Present only if CVfold is non-zero.
- **sampleClusterLabels** A list with two components (one per class). Each component is a vector of
sample cluster labels for samples in the class.

Author(s)

Peter Langfelder

See Also

votingLinearPredictor
nearestNeighborConnectivity

*Connectivity to a constant number of nearest neighbors*

**Description**

Given expression data and basic network parameters, the function calculates connectivity of each gene to a given number of nearest neighbors.

**Usage**

    nearestNeighborConnectivity(datExpr, 
                               nNeighbors = 50, power = 6, type = "unsigned", 
                               corFnc = "cor", corOptions = "use = 'p'", 
                               blockSize = 1000, 
                               sampleLinks = NULL, nLinks = 5000, setSeed = 38457, 
                               verbose = 1, indent = 0)

**Arguments**

- `datExpr`: a data frame containing expression data, with rows corresponding to samples and columns to genes. Missing values are allowed and will be ignored.
- `nNeighbors`: number of nearest neighbors to use.
- `power`: soft thresholding power for network construction. Should be a number greater than 1.
- `type`: a character string encoding network type. Recognized values are (unique abbreviations of) "unsigned", "signed", and "signed hybrid".
- `corFnc`: character string containing the name of the function to calculate correlation. Suggested functions include "cor" and "bicor".
- `corOptions`: further argument to the correlation function.
- `blockSize`: correlation calculations will be split into square blocks of this size, to prevent running out of memory for large gene sets.
- `sampleLinks`: logical: should network connections be sampled (TRUE) or should all connections be used systematically (FALSE)?
- `nLinks`: number of links to be sampled. Should be set such that `nLinks * nNeighbors` be several times larger than the number of genes.
- `setSeed`: seed to be used for sampling, for repeatability. If a seed already exists, it is saved before the sampling starts and restored upon exit.
- `verbose`: integer controlling the level of verbosity. 0 means silent.
- `indent`: integer controlling indentation of output. Each unit above 0 adds two spaces.

**Details**

Connectivity of gene $i$ is the sum of adjacency strengths between gene $i$ and other genes; in this case we take the $nNeighbors$ nodes with the highest connection strength to gene $i$. The adjacency strengths are calculated by correlating the given expression data using the function supplied in `corFnc` and transforming them into adjacency according to the given network `type` and `power`. 
Value

A vector with one component for each gene containing the nearest neighbor connectivity.

Author(s)

Peter Langfelder

See Also

adjacency, softConnectivity

Description

Given expression data from several sets and basic network parameters, the function calculates connectivity of each gene to a given number of nearest neighbors in each set.

Usage

nearestNeighborConnectivityMS(multiExpr, nNeighbors = 50, power = 6, type = "unsigned", corFnc = "cor", corOptions = "use = 'p'", blockSize = 1000, sampleLinks = NULL, nLinks = 5000, setSeed = 36492, verbose = 1, indent = 0)

Arguments

multiExpr  
expression data in multi-set format. A vector of lists, one list per set. In each list there must be a component named data whose content is a matrix or dataframe or array of dimension 2 containing the expression data. Rows correspond to samples and columns to genes (probes).

nNeighbors  
number of nearest neighbors to use.

power  
soft thresholding power for network construction. Should be a number greater than 1.

type  
a character string encoding network type. Recognized values are (unique abbreviations of) "unsigned", "signed", and "signed hybrid".

corFnc  
character string containing the name of the function to calculate correlation. Suggested functions include "cor" and "bicor".

corOptions  
further argument to the correlation function.

blockSize  
correlation calculations will be split into square blocks of this size, to prevent running out of memory for large gene sets.

sampleLinks  
logical: should network connections be sampled (TRUE) or should all connections be used systematically (FALSE)?

nLinks  
number of links to be sampled. Should be set such that nLinks * nNeighbors be several times larger than the number of genes.
setSeed

seed to be used for sampling, for repeatability. If a seed already exists, it is saved before the sampling starts and restored after.

verbose

integer controlling the level of verbosity. 0 means silent.

indent

integer controlling indentation of output. Each unit above 0 adds two spaces.

Details

Connectivity of gene \( i \) is the sum of adjacency strengths between gene \( i \) and other genes; in this case we take the \( n \) Neighbors nodes with the highest connection strength to gene \( i \). The adjacency strengths are calculated by correlating the given expression data using the function supplied in \texttt{corFNC} and transforming them into adjacency according to the given network type and power.

Value

A matrix in which columns correspond to sets and rows to genes; each entry contains the nearest neighbor connectivity of the corresponding gene.

Author(s)

Peter Langfelder

See Also

\texttt{adjacency, softConnectivity, nearestNeighborConnectivity}

Description

This function calculates various network concepts (topological properties, network indices) of a network calculated from expression data. See details for a detailed description.

Usage

\begin{verbatim}
networkConcepts(datExpr, power = 1, trait = NULL, networkType = "unsigned")
\end{verbatim}

Arguments

\begin{itemize}
  \item \texttt{datExpr} a data frame containing the expression data, with rows corresponding to samples and columns to genes (nodes).
  \item \texttt{power} soft thresholding power.
  \item \texttt{trait} optional specification of a sample trait. A vector of length equal the number of samples in \texttt{datExpr}.
  \item \texttt{networkType} network type. Recognized values are (unique abbreviations of) "unsigned", "signed", and "signed hybrid".
\end{itemize}
Details

This function computes various network concepts (also known as network statistics, topological properties, or network indices) for a weighted correlation network. The nodes of the weighted correlation network will be constructed between the columns (interpreted as nodes) of the input `datExpr`. If the option `networkType="unsigned"` then the adjacency between nodes i and j is defined as \[ A[i,j] = \text{abs}(\text{cor}(\text{datExpr}[i,:], \text{datExpr}[j,:]))^{\text{power}} \]. In the following, we use the term gene and node interchangeably since these methods were originally developed for gene networks. The function computes the following 4 types of network concepts (introduced in Horvath and Dong 2008):

Type I: fundamental network concepts are defined as a function of the off-diagonal elements of an adjacency matrix \( A \) and/or a node significance measure GS. These network concepts can be defined for any network (not just correlation networks). The adjacency matrix of an unsigned weighted correlation network is given by \( A = \text{abs}(\text{cor}(\text{datExpr}, \text{use}="p"))^{\text{power}} \) and the trait based gene significance measure is given by \( G = \text{abs}(\text{cor}(\text{datExpr}, \text{trait}, \text{use}="p"))^{\text{power}} \) where `datExpr`, `trait`, `power` are input parameters.

Type II: conformity-based network concepts are functions of the off-diagonal elements of the conformity based adjacency matrix \( A \cdot CF = CF \cdot t(CF) \) and/or the node significance measure. These network concepts are defined for any network for which a conformity vector can be defined. Details: For any adjacency matrix \( A \), the conformity vector \( CF \) is calculated by requiring that \( A[i,j] \) is approximately equal to \( CF[i] \cdot CF[j] \). Using the conformity one can define the matrix \( A \cdot CF = CF \cdot t(CF) \) which is the outer product of the conformity vector with itself. In general, \( A \cdot CF \) is not an adjacency matrix since its diagonal elements are different from 1. If the off-diagonal elements of \( A \cdot CF \) are similar to those of \( A \) according to the Frobenius matrix norm, then \( A \) is approximately factorizable. To measure the factorizability of a network, one can calculate the Factorizability, which is a number between 0 and 1 (Dong and Horvath 2007). The conformity is defined using a monotonic, iterative algorithm that maximizes the factorizability measure.

Type III: approximate conformity based network concepts are functions of all elements of the conformity based adjacency matrix \( A \cdot CF \) (including the diagonal) and/or the node significance measure GS. These network concepts are very useful for deriving relationships between network concepts in networks that are approximately factorizable.

Type IV: eigengene-based (also known as eigennode-based) network concepts are functions of the eigengene-based adjacency matrix \( A \cdot E = \text{ConformityE} \cdot t(\text{ConformityE}) \) (diagonal included) and/or the corresponding eigengene-based gene significance measure GSE. These network concepts can only be defined for correlation networks. Details: The columns (nodes) of `datExpr` can be summarized with the first principal component, which is referred to as Eigengene in coexpression network analysis. In general correlation networks, it is called eigengene. The eigengene-based conformity \( \text{ConformityE}[i] \) is defined as \( \text{abs}(\text{cor}(\text{datE}[i,:], \text{Eigengene}))^{\text{power}} \) where the power corresponds to the power used for defining the weighted adjacency matrix \( A \cdot CF \). The eigengene-based conformity can also be used to define an eigengene-based adjacency matrix \( A \cdot E = \text{ConformityE} \cdot t(\text{ConformityE}) \). The eigengene based factorizability \( EF(\text{datE}) \) is a number between 0 and 1 that measures how well \( A \cdot E \) approximates \( A \) when the power parameter equals 1. \( EF(\text{datE}) \) is defined with respect to the singular values of `datExpr`. For a trait based node significance measure \( GSE[i] = \text{abs}(\text{cor}(\text{Eigengene}, \text{trait}))^{\text{power}} \), one can also define an eigengene-based node significance measure \( GSE[i] = \text{abs}(\text{cor}(\text{EigengeneStart}, \text{trait}))^{\text{power}} \) where the eigengene significance \( \text{abs}(\text{cor}(\text{Eigengene}, \text{trait}))^{\text{power}} \) is defined as power of the absolute value of the correlation between eigengene and trait. Eigengene-based network concepts are very useful for providing a geometric interpretation of network concepts and for deriving relationships between network concepts. For example, the hub gene significance measure and its eigengene-based analog have been used to characterize networks where highly connected hub genes are important with regard to a trait based gene significance measure (Horvath and Dong 2008).
Value

A list with the following components:

**Summary**
- A data frame whose rows report network concepts that only depend on the adjacency matrix. Density (mean adjacency), Centralization, Heterogeneity (coefficient of variation of the connectivity), Mean ClusterCoef, Mean Connectivity. The columns of the data frame report the 4 types of network concepts mentioned in the description: Fundamental concepts, eigengene-based concepts, conformity-based concepts, and approximate conformity-based concepts.

**Size**
- Reports the network size, i.e., the number of nodes, which equals the number of columns of the input data frame `datExpr`.

**Factorizability**
- A number between 0 and 1. The closer it is to 1, the better the off-diagonal elements of the conformity based network $A_{CF}$ approximate those of $A$ (according to the Frobenius norm).

**Eigengene**
- The first principal component of the standardized columns of `datExpr`. The number of components of this vector equals the number of rows of `datExpr`.

**VarExplained**
- The proportion of variance explained by the first principal component (the Eigengene). It is numerically different from the eigengene based factorizability. While VarExplained is based on the squares of the singular values of `datExpr`, the eigengene-based factorizability is based on fourth powers of the singular values.

**Conformity**
- A numerical vector giving the conformity. The number of components of the conformity vector equals the number of columns in `datExpr`. The conformity is often highly correlated with the vector of node connectivities. The conformity is computed using an iterative algorithm for maximizing the factorizability measure. The algorithm and related network concepts are described in Dong and Horvath 2007.

**ClusterCoef**
- A numerical vector that reports the cluster coefficient for each node. This fundamental network concept measures the cliquishness of each node.

**Connectivity**
- A numerical vector that reports the connectivity (also known as degree) of each node. This fundamental network concept is also known as whole network connectivity. One can also define the scaled connectivity $k=\text{Connectivity}/\max(\text{Connectivity})$ which is used for computing the hub gene significance.

**MAR**
- A numerical vector that reports the maximum adjacency ratio for each node. $\text{MAR}[i]$ equals 1 if all non-zero adjacencies between node $i$ and the remaining network nodes equal 1. This fundamental network concept is always 1 for nodes of an unweighted network. This is a useful measure for weighted networks since it allows one to determine whether a node has high connectivity because of many weak connections (small MAR) or because of strong (but few) connections (high MAR), see Horvath and Dong 2008.

**ConformityE**
- A numerical vector that reports the eigengene based (aka eigennode based) conformity for the correlation network. The number of components equals the number of columns of `datExpr`.

**GS**
- A numerical vector that encodes the node (gene) significance. The i-th component equals the node significance of the i-th column of `datExpr` if a sample trait was supplied to the function (input trait). $\text{GS}[i]=\abs(\text{cor}(`datE[i], trait, use="p"))*\text{power}$

**GSE**
- A numerical vector that reports the eigengene based gene significance measure. Its i-th component is given by $\text{GSE}[i]=\text{ConformityE}[i]*\text{EigengeneSignificance}$
where the eigengene significance \( \text{abs}(\text{cor}(\text{Eigengene}, \text{trait}))^{\text{power}} \) is defined as power of the absolute value of the correlation between eigengene and trait.

**Significance**  
A data frame whose rows report network concepts that also depend on the trait based node significance measure. The rows correspond to network concepts and the columns correspond to the type of network concept (fundamental versus eigengene based). The first row of the data frame reports the network significance. The fundamental version of this network concept is the average gene significance = mean(GS). The eigengene based analog of this concept is defined as mean(GSE). The second row reports the hub gene significance which is defined as slope of the intercept only regression model that regresses the gene significance on the scaled network connectivity \( K \). The third row reports the eigengene significance \( \text{abs}(\text{cor}(\text{Eigengene}, \text{trait}))^{\text{power}} \). More details can be found in Horvath and Dong (2008).

**Author(s)**  
Jun Dong, Steve Horvath, Peter Langfelder

**References**  


**See Also**  
`conformityBasedNetworkConcepts` for approximate conformity-based network concepts  
`fundamentalNetworkConcepts` for calculation of fundamental network concepts only.

---

**networkScreening**  
*Identification of genes related to a trait*

**Description**  
This function blends standard and network approaches to selecting genes (or variables in general) highly related to a given trait.

**Usage**  
```r
networkScreening(y, datME, datExpr,  
corfnc = "cor", corOptions = "use = 'p'",  
oddPower = 3,  
blockSize = 1000,  
minimumSampleSize = ..minNSamples,  
addMEy = TRUE, removeDiag = FALSE,  
weightESy = 0.5, getQValues = TRUE)
```
Arguments

y clinical trait given as a numeric vector (one value per sample)
datME data frame of module eigengenes
datExpr data frame of expression data
corfnc character string specifying the function to be used to calculate co-expression similarity. Defaults to Pearson correlation. Any function returning values between -1 and 1 can be used.
corOptions character string specifying additional arguments to be passed to the function given by corFnc. Use “use = 'p', method = 'spearman'” to obtain Spearman correlation.
oddPower odd integer used as a power to raise module memberships and significances
blockSize block size to use for calculations with large data sets
minimumSampleSize minimum acceptable number of samples. Defaults to the default minimum number of samples used throughout the WGCNA package, currently 4.
addMEy logical: should the trait be used as an additional "module eigengene"?
removeDiag logical: remove the diagonal?
weightESy weight to use for the trait as an additional eigengene; should be between 0 and 1
getQValues logical: should q-values be calculated?

Details

This function should be considered experimental. It takes into account both the "standard" and the network measures of gene importance for the trait.

Value

datout = data.frame(p.Weighted, q.Weighted, Cor.Weighted, Z.Weighted, p.Standard, q.Standard, Cor.Standard, Z.Standard) Data frame reporting the following quantities for each given gene:

p.Weighted weighted p-value of association with the trait
q.Weighted q-value (local FDR) calculated from p.Weighted
cor.Weighted correlation of trait with gene expression weighted by a network term
Z.Weighted Fisher Z score of the weighted correlation
p.Standard standard Student p-value of association of the gene with the trait
q.Standard q-value (local FDR) calculated from p.Standard
cor.Standard correlation of gene with the trait
Z.Standard Fisher Z score of the standard correlation

Author(s)

Steve Horvath
Network gene screening with an external gene significance measure

Description

This function blends standard and network approaches to selecting genes (or variables in general) with high gene significance.

Usage

networkscreeningGS(
  datExpr,
  datME,
  GS,
  oddPower = 3,
  blockSize = 1000,
  minimumSampleSize = .minNsamples,
  addGS = TRUE)

Arguments

  datExpr     data frame of expression data
  datME       data frame of module eigengenes
  GS          numeric vector of gene significances
  oddPower    odd integer used as a power to raise module memberships and significances
  blockSize   block size to use for calculations with large data sets
  minimumSampleSize minimum acceptable number of samples. Defaults to the default minimum number of samples used throughout the WGCNA package, currently 4.
  addGS       logical: should gene significances be added to the screening statistics?

Details

This function should be considered experimental. It takes into account both the "standard" and the network measures of gene importance for the trait.

Value

  GS.Weighted weighted gene significance
  GS            copy of the input gene significances (only if addGS=TRUE)

Author(s)

  Steve Horvath

See Also

  networkScreening, automaticNetworkScreeningGS
**normalizeLabels**  
*Transform numerical labels into normal order.*

**Description**

Transforms numerical labels into normal order, that is the largest group will be labeled 1, next largest 2 etc. Label 0 is optionally preserved.

**Usage**

`normalizeLabels(labels, keepZero = TRUE)`

**Arguments**

- `labels`  
  Numerical labels.
- `keepZero`  
  If TRUE (the default), labels 0 are preserved.

**Value**

A vector of the same length as input, containing the normalized labels.

**Author(s)**

Peter Langfelder, <Peter.Langfelder@gmail.com>

---

**nPresent**  
*Number of present data entries.*

**Description**

A simple sum of present entries in the argument.

**Usage**

`nPresent(x)`

**Arguments**

- `x`  
  data in which to count number of present entries.

**Value**

A single number giving the number of present entries in `x`.

**Author(s)**

Steve Horvath
nSets

Description
A convenience function that returns the number of sets in a multi-set variable.

Usage
nSets(multiData, ...)

Arguments

multiData vector of lists; in each list there must be a component named data whose content is a matrix or dataframe or array of dimension 2.

... Other arguments to function checkSets.

Value
A single integer that equals the number of sets given in the input multiData.

Author(s)
Peter Langfelder

See Also
checkSets

numbers2colors

Description
The function creates a color representation for the given numeric input.

Usage
numbers2colors(
x,
signed = NULL,
centered = signed,
lim = NULL,
commonLim = FALSE,
colors = if (signed) blueWhiteRed(100) else blueWhiteRed(100)[51:100],
naColor = "grey")
orderBranchesUsingHubGenes

Arguments

- **x**: a vector or matrix of numbers. Missing values are allowed and will be assigned the color given in `naColor`. If a matrix, each column of the matrix is processed separately and the return value will be a matrix of colors.
- **signed**: logical: should `x` be considered signed? If TRUE, the default setting is to use a palette that starts with green for the most negative values, continues with white for values around zero and turns red for positive values. If FALSE, the default palette ranges from white for minimum values to red for maximum values. If not given, the behaviour is controlled by values in `x`: if there are both positive and negative values, `signed` will be considered TRUE, otherwise FALSE.
- **centered**: logical. If TRUE and `signed`==TRUE, numeric value zero will correspond to the middle of the color palette. If FALSE or `signed`==FALSE, the middle of the color palette will correspond to the average of the minimum and maximum value. If neither `signed` nor `centered` are given, `centered` will follow `signed` (see above).
- **lim**: optional specification of limits, that is numeric values that should correspond to the first and last entry of `colors`.
- **commonLim**: logical: should limits be calculated separately for each column of `x`, or should the limits be the same for all columns? Only applies if `lim` is NULL.
- **colors**: color palette to represent the given numbers.
- **naColor**: color to represent missing values in `x`.

Details

Each column of `x` is processed individually, meaning that the color palette is adjusted individually for each column of `x`.

Value

A vector or matrix (of the same dimensions as `x`) of colors.

Author(s)

Peter Langfelder

See Also

- `labels2colors` for color coding of ordinal labels.

---

**orderBranchesUsingHubGenes**

*Optimize dendrogram using branch swaps and reflections.*

Description

This function takes as input the hierarchical clustering tree as well as a subset of genes in the network (generally corresponding to branches in the tree), then returns a semi-optimally ordered tree. The idea is to maximize the correlations between adjacent branches in the dendrogram, in as much as that is possible by adjusting the arbitrary positionings of the branches by swapping and reflecting branches.
orderBranchesUsingHubGenes

Usage

orderBranchesUsingHubGenes(
    hierTOM,
    datExpr = NULL, colorh = NULL,
    type = "signed", adj = NULL, iter = NULL,
    useReflections = FALSE, allowNonoptimalSwaps = FALSE)

Arguments

hierTOM A hierarchical clustering object (or gene tree) that is used to plot the dendrogram. For example, the output object from the function hclust or fastcluster::hclust. Note that elements of hierTOM$order MUST be named (for example, with the corresponding gene name).

datExpr Gene expression data with rows as samples and columns as genes, or NULL if a pre-made adjacency is entered. Column names of datExpr must be a subset of gene names of hierTOM$order.

colorh The module assignments (color vectors) corresponding to the rows in datExpr, or NULL if a pre-made adjacency is entered.

type What type of network is being entered. Common choices are "signed" (default) and "unsigned". With "signed" negative correlations count against, whereas with "unsigned" negative correlations are treated identically as positive correlations.

adj Either NULL (default) or an adjacency (or any other square) matrix with rows and columns corresponding to a subset of the genes in hierTOM$order. If entered, datExpr, colorh, and type are all ignored. Typically, this would be left blank but could include correlations between module eigengenes, with rows and columns renamed as genes in the corresponding modules, for example.

iter The number of iterations to run the function in search of optimal branch ordering. The default is the square of the number of modules (or the square of the number of genes in the adjacency matrix).

useReflections If TRUE, both reflections and branch swapping will be used to optimize dendrogram. If FALSE (default) only branch swapping will be used.

allowNonoptimalSwaps If TRUE, there is chance (that decreases with each iteration) of swapping / reflecting branches whether or not the new correlation between expression of genes in adjacent branches is better or worse. The idea (which has not been sufficiently tested), is that this would prevent the function from getting stuck at a local maxima of correlation. If FALSE (default), the swapping / reflection of branches only occurs if it results in a higher correlation between adjacent branches.

Value

hierTOM A hierarchical clustering object with the hierTOM$order variable properly adjusted, but all other variables identical as the heirTOM input.

changeLog A log of all of the changes that were made to the dendrogram, including what change was made, on what iteration, and the Old and New scores based on correlation. These scores have arbitrary units, but higher is better.
Note

This function is very slow and is still in an "experimental" function. We have not had problems with ~10 modules across ~5000 genes, although theoretically it should work for many more genes and modules, depending upon the speed of the computer running R. Please address any problems or suggestions to jeremyinla@gmail.com.

Author(s)

Jeremy Miller

Examples

```r
## Example: first simulate some data.

MEturquoise = sample(1:100,50)
MEblue = c(MEturquoise[1:25], sample(1:100,25))
MEbrown = sample(1:100,50)
MEyellow = sample(1:100,50)
MEgreen = c(MEyellow[1:30], sample(1:100,20))
MERed = c(MEbrowm[1:20], sample(1:100,30))
ME = data.frame(MEturquoise, MEblue, MEbrown, MEyellow, MEgreen, MERed)
dat1 = simulateDataExpr(ME, 400, c(0.16,0.12,0.11,0.10,0.10,0.10,0.10,0.1), signed=TRUE)
TOM1 = TOMsimilarityFromExpr(dat1$datExpr, networkType="signed")
colnames(TOM1) <- rownames(TOM1) <- colnames(dat1$datExpr)
tree1 = fastcluster::hclust(as.dist(1-TOM1),method="average")
colorh = labels2colors(dat1$allLabels)
plotDendroAndColors(tree1,colorh,dendroLabels=FALSE)

## Reassign modules using the selectBranch and chooseOneHubInEachModule functions

datExpr = dat1$datExpr
hubs = chooseOneHubInEachModule(datExpr, colorh)
colorh2 = rep("grey", length(colorh))
colorh2[selectBranch(tree1,hubs["blue"],hubs["turquoise")]] = "blue"
colorh2[selectBranch(tree1,hubs["turquoise"],hubs["blue")]] = "turquoise"
colorh2[selectBranch(tree1,hubs["green"],hubs["yellow")]] = "green"
colorh2[selectBranch(tree1,hubs["yellow"],hubs["green")]] = "yellow"
colorh2[selectBranch(tree1,hubs["red"],hubs["brown")]] = "red"
colorh2[selectBranch(tree1,hubs["brown"],hubs["red")]] = "brown"
plotDendroAndColors(tree1,cbind(colorh,colorh2),c("Old","New"),dendroLabels=FALSE)

## Now swap and reflect some branches, then optimize the order of the branches
# and output pdf with resulting images

## Not run:

pdf("DENDROGRAM_PLOTS.pdf",width=10,height=5)
plotDendroAndColors(tree1,colorh2,dendroLabels=FALSE,main="Starting Dendrogram")
tree1 = swapTwoBranches(tree1,hubs["red"],hubs["turquoise")])
plotDendroAndColors(tree1,colorh2,dendroLabels=FALSE,main="Swap blue/turquoise and red/brown")
tree1 = reflectBranch(tree1,hubs["blue"],hubs["green")]
plotDendroAndColors(tree1,colorh2,dendroLabels=FALSE,main="Reflect turquoise/blue")

# (This function will take a few minutes)
orderMEs

Put close eigenvectors next to each other

Description

Reorder given (eigen-)vectors such that similar ones (as measured by correlation) are next to each other.

Usage

orderMEs(MEs, greyLast = TRUE,
  greyName = paste(moduleColor.getMEprefix(), "grey", sep=""),
  orderBy = 1, order = NULL,
  useSets = NULL, verbose = 0, indent = 0)

Arguments

MEs Module eigengenes in a multi-set format (see checkSets). A vector of lists, with each list corresponding to one dataset and the module eigengenes in the component data, that is MEs[[set]]$data[sample, module] is the expression of the eigengene of module module in sample sample in dataset set. The number of samples can be different between the sets, but the modules must be the same.

greyLast Normally the color grey is reserved for unassigned genes; hence the grey module is not a proper module and it is conventional to put it last. If this is not desired, set the parameter to FALSE.

greyName Name of the grey module eigengene.

orderBy Specifies the set by which the eigengenes are to be ordered (in all other sets as well). Defaults to the first set in useSets (or the first set, if useSets is not given).

order Allows the user to specify a custom ordering.

useSets Allows the user to specify for which sets the eigengene ordering is to be performed.

verbose Controls verbosity of printed progress messages. 0 means silent, nonzero verbose.

indent A single non-negative integer controlling indentation of printed messages. 0 means no indentation, each unit above zero adds two spaces.
Details

Ordering module eigengenes is useful for plotting purposes. For this function the order can be specified explicitly, or a set can be given in which the correlations of the eigengenes will determine the order. For the latter, a hierarchical dendrogram is calculated and the order given by the dendrogram is used for the eigengenes in all other sets.

Value

A vector of lists of the same type as MEs containing the re-ordered eigengenes.

Author(s)

Peter Langfelder, <Peter.Langfelder@gmail.com>

See Also

moduleEigengenes, multisetMEs, consensusOrderMEs

---

overlapTable

Calculate overlap of modules

Description

The function calculates overlap counts and Fisher exact test p-values for the given two sets of module assignments.

Usage

    overlapTable(
        labelsQ, labelsR, 
        na.rm = TRUE, ignore = NULL,
        levelsQ = NULL, levelsR = NULL)

Arguments

- **labelsQ**: a vector containing module labels.
- **labelsR**: a vector containing module labels to be compared to labelsQ.
- **na.rm**: logical: should entries missing in either labels1 or labels2 be removed?
- **ignore**: an optional vector giving label levels that are to be ignored.
- **levelsQ**: optional vector giving levels for labelsQ. Defaults to sorted unique non-missing values in labelsQ that are not present in ignore.
- **levelsR**: optional vector giving levels for labelsR. Defaults to sorted unique non-missing values in labelsR that are not present in ignore.
Value

A list with the following components:

- **countTable**: a matrix whose rows correspond to modules (unique labels) in `labels1` and whose columns correspond to modules (unique labels) in `labels2`, giving the number of objects in the intersection of the two respective modules.

- **pTable**: a matrix whose rows correspond to modules (unique labels) in `labels1` and whose columns correspond to modules (unique labels) in `labels2`, giving Fisher’s exact test significance p-values for the overlap of the two respective modules.

Author(s)

Peter Langfelder

See Also

- `fisher.test`
- `matchLabels`

Description

Takes two sets of expression data (or kME tables) as input and returns a table listing the significant overlap between each module in each data set, as well as the actual genes in common for every module pair. Modules can be defined in several ways (generally involving kME) based on user input.

Usage

```r
overlapTableUsingKME(
  dat1, dat2, 
  colorh1, colorh2, 
  MES1 = NULL, MES2 = NULL, 
  name1 = "MM1", name2 = "MM2", 
  cutoffMethod = "assigned", cutoff = 0.5, 
  omitGrey = TRUE, datIsExpression = TRUE)
```

Arguments

- **dat1, dat2**: Either expression data sets (with samples as rows and genes as columns) or module membership (kME) tables (with genes as rows and modules as columns). Function reads these inputs based on whether `datIsExpression=TRUE` or `FALSE`. ***Be sure that these inputs include relevant row and column names, or else the function will not work properly.***

- **colorh1, colorh2**: Color vector (module assignments) corresponding to the genes from dat1/2. This vector must be the same length as the Gene dimension from dat1/2.
If entered (default=NULL), these are the module eigengenes that will be used to form the kME tables. Rows are samples and columns are module assignments. Note that if datIsExpression=FALSE, these inputs are ignored.

The names of the two data sets being compared. These names affect the output parameters.

This variable is used to determine how modules are defined in each data set. Must be one of four options: (1) "assigned" -> use the module assignments in colorh (default); (2) "kME" -> any gene with kME > cutoff is in the module; (3) "numGenes" -> the top cutoff number of genes based on kME is in the module; and (4) "pvalue" -> any gene with correlation pvalue < cutoff is in the module (this includes both positively and negatively-correlated genes).

For all cutoffMethods other than "assigned", this parameter is used as the described cutoff value.

If TRUE the grey modules (non-module genes) for both networks are not returned.

If TRUE (default), dat1/2 is assumed to be expression data. If FALSE, dat1/2 is assumed to be a table of kME values.

A table of p-values showing significance of module overlap based on the hypergeometric test. Note that these p-values are not corrected for multiple comparisons.

A character vector of all genes in common between the two data sets.

A list of character vectors of all genes in each module in both data sets. All genes in the MOD module in data set MM1 could be found using "<outputVariableName>$GenesMM1$MM1_MOD"

A list of character vectors of all genes for each between-set comparison from PvaluesHypergeo. All genes in MOD.A from MM1 that are also in MOD.B from MM2 could be found using "<outputVariableName>$OverlappingGenes$MM1_MOD.A_MM2_MOD.B"

Jeremy Miller

See Also

overlapTable

# Example: first generate simulated data.

set.seed(100)
ME.A = sample(1:100,50); ME.B = sample(1:100,50)
ME.C = sample(1:100,50); ME.D = sample(1:100,50)
ME.E = sample(1:100,50); ME.F = sample(1:100,50)
ME.G = sample(1:100,50); ME.H = sample(1:100,50)
ME1 = data.frame(ME.A, ME.B, ME.C, ME.D, ME.E)
pickHardThreshold

Analysis of scale free topology for hard-thresholding.

Description

Analysis of scale free topology for multiple hard thresholds. The aim is to help the user pick an appropriate threshold for network construction.

Usage

pickHardThreshold(
data,
dataisExpr,
RsquaredCut = 0.85,
cutVector = seq(0.1, 0.9, by = 0.05),
moreNetworkConcepts = FALSE,
removeFirst = FALSE, nBreaks = 10,
corFnc = "cor", corOptions = "use = 'p'"
)

pickHardThreshold.fromSimilarity(
similarity,
RsquaredCut = 0.85,
cutVector = seq(0.1, 0.9, by = 0.05),
moreNetworkConcepts = FALSE,
removeFirst = FALSE, nBreaks = 10)

Arguments

data expression data in a matrix or data frame. Rows correspond to samples and columns to genes.
pickHardThreshold

- `dataIsExpr` logical: should the data be interpreted as expression (or other numeric) data, or as a similarity matrix of network nodes?
- `similarity` similarity matrix: a symmetric matrix with entries between -1 and 1 and unit diagonal.
- `R_squaredCut` desired minimum scale free topology fitting index $R^2$.
- `cutVector` a vector of hard threshold cuts for which the scale free topology fit indices are to be calculated.
- `moreNetworkConcepts` logical: should additional network concepts be calculated? If `TRUE`, the function will calculate how the network density, the network heterogeneity, and the network centralization depend on the power. For the definition of these additional network concepts, see Horvath and Dong (2008). PLoS Comp Biol.
- `removeFirst` should the first bin be removed from the connectivity histogram?
- `nBreaks` number of bins in connectivity histograms
- `corFnc` a character string giving the correlation function to be used in adjacency calculation.
- `corOptions` further options to the correlation function specified in `corFnc`.

**Details**

The function calculates unsigned networks by thresholding the correlation matrix using thresholds given in `cutVector`. For each power the scale free topology fit index is calculated and returned along with other information on connectivity.

**Value**

A list with the following components:

- `cutEstimate` estimate of an appropriate hard-thresholding cut: the lowest cut for which the scale free topology fit $R^2$ exceeds `R_squaredCut`. If $R^2$ is below `R_squaredCut` for all cuts, `NA` is returned.
- `fitIndices` a data frame containing the fit indices for scale free topology. The columns contain the hard threshold, Student p-value for the correlation threshold, adjusted $R^2$ for the linear fit, the linear coefficient, adjusted $R^2$ for a more complicated fit models, mean connectivity, median connectivity and maximum connectivity. If input `moreNetworkConcepts` is `TRUE`, 3 additional columns containing network density, centralization, and heterogeneity.

**Author(s)**

Steve Horvath

**References**


**pickSoftThreshold**

**Description**

Analysis of scale free topology for multiple soft thresholding powers. The aim is to help the user pick an appropriate soft-thresholding power for network construction.

**Usage**

```r
pickSoftThreshold(
  data, 
dataIsExpr = TRUE,
  RsquaredCut = 0.85,
  powerVector = c(seq(1, 10, by = 1), seq(12, 20, by = 2)),
  removeFirst = FALSE, nBreaks = 10, blockSize = NULL,
  corFnc = cor, corOptions = list(use = 'p'),
  networkType = "unsigned",
  moreNetworkConcepts = FALSE,
  verbose = 0, indent = 0)
```

```r
corrSoftThresholdFromSimilarity(
  similarity, 
  RsquaredCut = 0.85,
  powerVector = c(seq(1, 10, by = 1), seq(12, 20, by = 2)),
  removeFirst = FALSE, nBreaks = 10, blockSize = 1000,
  networkType = "unsigned",
  moreNetworkConcepts = FALSE,
  verbose = 0, indent = 0)
```

**Arguments**

- `data`: expression data in a matrix or data frame. Rows correspond to samples and columns to genes.
- `dataIsExpr`: logical: should the data be interpreted as expression (or other numeric) data, or as a similarity matrix of network nodes?
- `similarity`: similarity matrix: a symmetric matrix with entries between -1 and 1 and unit diagonal.
- `RsquaredCut`: desired minimum scale free topology fitting index $R^2$.
- `powerVector`: a vector of soft thresholding powers for which the scale free topology fit indices are to be calculated.
- `removeFirst`: should the first bin be removed from the connectivity histogram?
- `nBreaks`: number of bins in connectivity histograms.
blockSize: block size into which the calculation of connectivity should be broken up. If not given, a suitable value will be calculated using function blockSize and printed if verbose>0. If R runs into memory problems, decrease this value.

corFnc: the correlation function to be used in adjacency calculation.

corOptions: a list giving further options to the correlation function specified in corFnc.

networkType: network type. Allowed values are (unique abbreviations of) "unsigned", "signed", "signed hybrid". See adjacency.

moreNetworkConcepts: logical: should additional network concepts be calculated? If TRUE, the function will calculate how the network density, the network heterogeneity, and the network centralization depend on the power. For the definition of these additional network concepts, see Horvath and Dong (2008). PloS Comp Biol.

verbose: integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.

indent: indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

The function calculates weighted networks either by interpreting data directly as similarity, or first transforming it to similarity of the type specified by networkType. The weighted networks are obtained by raising the similarity to the powers given in powerVector. For each power the scale free topology fit index is calculated and returned along with other information on connectivity.

On systems with multiple cores or processors, the function pickSoftThreshold takes advantage of parallel processing if the function enableWGCNAThreads has been called to allow parallel processing and set up the parallel calculation back-end.

Value

A list with the following components:

powerEstimate: estimate of an appropriate soft-thresholding power: the lowest power for which the scale free topology fit $R^2$ exceeds RsquaredCut. If $R^2$ is below RsquaredCut for all powers, NA is returned.

fitIndices: a data frame containing the fit indices for scale free topology. The columns contain the soft-thresholding power, adjusted $R^2$ for the linear fit, the linear coefficient, adjusted $R^2$ for a more complicated fit models, mean connectivity, median connectivity and maximum connectivity. If input moreNetworkConcepts is TRUE, 3 additional columns containing network density, centralization, and heterogeneity.

Author(s)

Steve Horvath and Peter Langfelder

References


plotClusterTreeSamples

Annotated clustering dendrogram of microarray samples

Description
This function plots an annotated clustering dendrogram of microarray samples.

Usage
plotClusterTreeSamples(
  datExpr,
  y = NULL,
  traitLabels = NULL,
  yLabels = NULL,
  main = if (is.null(y)) "Sample dendrogram" else
    "Sample dendrogram and trait indicator",
  setLayout = TRUE, autocolorHeight = TRUE, colorHeight = 0.3,
  dendroLabels = NULL,
  addGuide = FALSE, guideAll = TRUE,
  guideCount = NULL, guideHang = 0.2,
  cex traitLabels = 0.8,
  cex.dendroLabels = 0.9,
  marAll = c(1, 5, 3, 1),
  saveMar = TRUE,
  abHeight = NULL, abCol = "red",
  ...
)

Arguments

datExpr a data frame containing expression data, with rows corresponding to samples and columns to genes. Missing values are allowed and will be ignored.

y microarray sample trait. Either a vector with one entry per sample, or a matrix in which each column corresponds to a (different) trait and each row to a sample.

traitLabels labels to be printed next to the color rows depicting sample traits. Defaults to column names of y.

yLabels Optional labels to identify colors in the row identifying the sample classes. If given, must be of the same dimensions as y. Each label that occurs will be displayed once.

main title for the plot.

setLayout logical: should the plotting device be partitioned into a standard layout? If FALSE, the user is responsible for partitioning. The function expects two regions of the same width, the first one immediately above the second one.

autocolorHeight logical: should the height of the color area below the dendrogram be automatically adjusted for the number of traits? Only effective if setLayout is TRUE.
colorHeight
Specifies the height of the color area under dendrogram as a fraction of the height of the dendrogram area. Only effective when autoColorHeight above is FALSE.

dendroLabels
dendrogram labels. Set to FALSE to disable dendrogram labels altogether; set to NULL to use row labels of datExpr.

addGuide
logical: should vertical "guide lines" be added to the dendrogram plot? The lines make it easier to identify color codes with individual samples.

guideAll
logical: add a guide line for every sample? Only effective for addGuide set TRUE.

guideCount
number of guide lines to be plotted. Only effective when addGuide is TRUE and guideAll is FALSE.

guideHang
fraction of the dendrogram height to leave between the top end of the guide line and the dendrogram merge height. If the guide lines overlap with dendrogram labels, increase guideHang to leave more space for the labels.

cex.traitLabels
character expansion factor for trait labels.

cex.dendroLabels
character expansion factor for dendrogram (sample) labels.

marAll
a 4-element vector giving the bottom, left, top and right margins around the combined plot. Note that this is not the same as setting the margins via a call to par, because the bottom margin of the dendrogram and the top margin of the color underneath are always zero.

saveMar
logical: save margins setting before starting the plot and restore on exit?

abHeight
optional specification of the height for a horizontal line in the dendrogram, see abline.

abCol
color for plotting the horizontal line.

...other graphical parameters to plot.hclust.

Details
The function generates an average linkage hierarchical clustering dendrogram (see hclust) of samples from the given expression data, using Euclidean distance of samples. The dendrogram is plotted together with color annotation for the samples.

The trait y must be numeric. If y is integer, the colors will correspond to values. If y is continuous, it will be dichotomized to two classes, below and above median.

Value
None.

Author(s)
Steve Horvath and Peter Langfelder

See Also
dist, hclust, plotDendroAndColors
plotColorUnderTree  

Plot color rows in a given order, for example under a dendrogram

Description
Plot color rows encoding information about objects in a given order, for example the order of a clustering dendrogram, usually below the dendrogram or a barplot.

Usage

plotOrderedColors(
    order,
    colors,
    rowLabels = NULL,
    rowWidths = NULL,
    rowText = NULL,
    rowTextAlignment = c("left", "center", "right"),
    rowTextIgnore = NULL,
    textPositions = NULL,
    addTextGuide = TRUE,
    cex.rowLabels = 1,
    cex.rowText = 0.8,
    startAt = 0,
    ...
)

plotColorUnderTree(
    dendro,
    colors,
    rowLabels = NULL,
    rowWidths = NULL,
    rowText = NULL,
    rowTextAlignment = c("left", "center", "right"),
    rowTextIgnore = NULL,
    textPositions = NULL,
    addTextGuide = TRUE,
    cex.rowLabels = 1,
    cex.rowText = 0.8,
    ...
)

Arguments

order  
A vector giving the order of the objects. Must have the same length as colors if colors is a vector, or as the number of rows if colors is a matrix or data frame.
dendro  
A hierarchical clustering dendrogram such one returned by hclust.
colors  
Coloring of objects on the dendrogram. Either a vector (one color per object) or a matrix (can also be an array or a data frame) with each column giving one color per object. Each column will be plotted as a horizontal row of colors under the dendrogram.
rowLabels  Labels for the colorings given in colors. The labels will be printed to the left of the color rows in the plot. If the argument is given, it must be a vector of length equal to the number of columns in colors. If not given, names(colors) will be used if available. If not, sequential numbers starting from 1 will be used.

rowWidths  Optional specification of relative row widths for the color and text (if given) rows. Need not sum to 1.

rowText  Optional labels to identify colors in the color rows. If given, must be of the same dimensions as colors. Each label that occurs will be displayed once.

rowTextAlignment  Character string specifying whether the labels should be left-justified to the start of the largest block of each label, centered in the middle, or right-justified to the end of the largest block.

rowTextIgnore  Optional specifications of labels that should be ignored when displaying them using rowText above.

textPositions  optional numeric vector of the same length as the number of columns in rowText giving the color rows under which the text rows should appear.

addTextGuide  logical: should guide lines be added for the text rows (if given)?

cex.rowLabels  Font size scale factor for the row labels. See par.

cex.rowText  character expansion factor for text rows (if given).

startAt  A numeric value indicating where in relationship to the left edge of the plot the center of the first rectangle should be. Useful values are 0 if plotting color under a dendrogram, and 0.5 if plotting colors under a barplot.

...  Other parameters to be passed on to the plotting method (such as main for the main title etc).

Details

It is often useful to plot dendrograms or other plots (e.g., barplots) of objects together with additional information about the objects, for example module assignment (by color) that was obtained by cutting a hierarchical dendrogram or external color-coded measures such as gene significance. This function provides a way to do so. The calling code should section the screen into two (or more) parts, plot the dendrogram (via plot(hclust)) or other information in the upper section and use this function to plot color annotation in the order corresponding to the dendrogram in the lower section.

Value

None.

Note

This function replaces plotHclustColors in package moduleColor.

Author(s)

Steve Horvath <SHorvath@mednet.ucla.edu> and Peter Langfelder <Peter.Langfelder@gmail.com>

See Also

cutreeDynamic for module detection in a dendrogram;
plotDendroAndColors for automated plotting of dendrograms and colors in one step.
**plotCor**

**Red and Green Color Image of Correlation Matrix**

**Description**

This function produces a red and green color image of a correlation matrix using an RGB color specification. Increasingly positive correlations are represented with reds of increasing intensity, and increasingly negative correlations are represented with greens of increasing intensity.

**Usage**

```r
plotCor(x, new=FALSE, nrgcols=50, labels=FALSE, labcols=1, title="", ...)```

**Arguments**

- `x` — a matrix of numerical values.
- `new` — If `new=FALSE`, `x` must already be a correlation matrix. If `new=TRUE`, the correlation matrix for the columns of `x` is computed and displayed in the image.
- `nrgcols` — the number of colors (>= 1) to be used in the red and green palette.
- `labels` — vector of character strings to be placed at the tickpoints, labels for the columns of `x`.
- `labcols` — colors to be used for the labels of the columns of `x`. `labcols` can have either length 1, in which case all the labels are displayed using the same color, or the same length as `labels`, in which case a color is specified for the label of each column of `x`.
- `title` — character string, overall title for the plot.
- `...` — graphical parameters may also be supplied as arguments to the function (see `par`). For comparison purposes, it is good to set `zlim=c(-1,1)`.

**Author(s)**

Sandrine Dudoit, <sandrine@stat.berkeley.edu>

**See Also**

- `plotMat`, `rgcolors`, `funcCor`, `image`, `rgb`.

**plotDendroAndColors**

**Dendrogram plot with color annotation of objects**

**Description**

This function plots a hierarchical clustering dendrogram and color annotation(s) of objects in the dendrogram underneath.
Usage

plotDendroAndColors(
  dendro,
  colors,
  groupLabels = NULL,
  rowText = NULL,
  rowTextAlignment = c("left", "center", "right"),
  rowTextIgnore = NULL,
  textPositions = NULL,
  setLayout = TRUE,
  autoColorHeight = TRUE,
  colorHeight = 0.2,
  rowWidths = NULL,
  dendroLabels = NULL,
  addGuide = FALSE, guideAll = FALSE,
  guideCount = 50, guideHang = 0.2,
  addTextGuide = FALSE,
  cex.colorLabels = 0.8, cex.dendroLabels = 0.9,
  cex.rowText = 0.8,
  marAll = c(1, 5, 3, 1), saveMar = TRUE,
  abHeight = NULL, abCol = "red", ...)  
)

Arguments

dendro
   a hierarchical clustering dendrogram such as one produced by hclust.

colors
   Coloring of objects on the dendrogram. Either a vector (one color per object)
   or a matrix (can also be an array or a data frame) with each column giving one
   color per object. Each column will be plotted as a horizontal row of colors under
   the dendrogram.

groupLabels
   Labels for the colorings given in colors. The labels will be printed to the left of
   the color rows in the plot. If the argument is given, it must be a vector of length
   equal to the number of columns in colors. If not given, names(colors) will
   be used if available. If not, sequential numbers starting from 1 will be used.

rowText
   Optional labels to identify colors in the color rows. If given, must be either
   the same dimensions as colors or must have the same number of rows and
   textPositions must be used to specify which columns of colors each column
   of rowText corresponds to. Each label that occurs will be displayed once, under
   the largest continuous block of the corresponding colors.

rowTextAlignment
   Character string specifying whether the labels should be left-justified to the start
   of the largest block of each label, centered in the middle, or right-justified to the
   end of the largest block.

rowTextIgnore
   Optional specifications of labels that should be ignored when displaying them
   using rowText above.

textPositions
   optional numeric vector of the same length as the number of columns in rowText
   giving the color rows under which the text rows should appear.

setLayout
   logical: should the plotting device be partitioned into a standard layout? If
   FALSE, the user is responsible for partitioning. The function expects two regions
   of the same width, the first one immediately above the second one.
autoColorHeight

logical: should the height of the color area below the dendrogram be automatically adjusted for the number of traits? Only effective if setLayout is TRUE.

colorHeight

specifies the height of the color area under dendrogram as a fraction of the height of the dendrogram area. Only effective when autoColorHeight above is FALSE.

rowWidths

optional specification of relative row widths for the color and text (if given) rows. Need not sum to 1.

dendroLabels

dendrogram labels. Set to FALSE to disable dendrogram labels altogether; set to NULL to use row labels of datExpr.

addGuide

logical: should vertical "guide lines" be added to the dendrogram plot? The lines make it easier to identify color codes with individual samples.

guideAll

logical: add a guide line for every sample? Only effective for addGuide set TRUE.

guideCount

number of guide lines to be plotted. Only effective when addGuide is TRUE and guideAll is FALSE.

guideHang

fraction of the dendrogram height to leave between the top end of the guide line and the dendrogram merge height. If the guide lines overlap with dendrogram labels, increase guideHang to leave more space for the labels.

addTextGuide

logical: should guide lines be added for the text rows (if given)?

cex.colorLabels

character expansion factor for trait labels.

cex.dendroLabels

character expansion factor for dendrogram (sample) labels.

cex.rowText

character expansion factor for text rows (if given).

marAll

a vector of length 4 giving the bottom, left, top and right margins of the combined plot. There is no margin between the dendrogram and the color plot underneath.

saveMar

logical: save margins setting before starting the plot and restore on exit?

abHeight

optional specification of the height for a horizontal line in the dendrogram, see abline.

abCol

color for plotting the horizontal line.

... other graphical parameters to plot.hclust.

Details

The function splits the plotting device into two regions, plots the given dendrogram in the upper region, then plots color rows in the region below the dendrogram.

Value

None.

Author(s)

Peter Langfelder

See Also

plotColorUnderTree
plotEigeneNetworks

Eigene network plot

Description

This function plots dendrogram and eigene gene representations of (consensus) eigene genes networks. In the case of consensus eigene gene networks the function also plots pairwise preservation measures between consensus networks in different sets.

Usage

plotEigeneNetworks(multiME, setLabels, letterSubPlots = FALSE, Letters = NULL,excludeGrey = TRUE, greyLabel = "grey", plotDendrograms = TRUE, plotHeatmaps = TRUE, setMargins = TRUE, marDendo = NULL, marHeatmap = NULL, colorLabels = TRUE, signed = TRUE, heatmapColors = NULL, plotAdjacency = TRUE, printAdjacency = FALSE, cex.adjacency = 0.9, coloredBarplot = TRUE, barplotMeans = TRUE, barplotErrors = FALSE, plotPreservation = "standard", zlimPreservation = c(0, 1), printPreservation = FALSE, cex.preservation = 0.9, ...

Arguments

multiME either a single data frame containing the module eigene genes, or module eigene genes in the multi-set format (see checkSets). The multi-set format is a vector of lists, one per set. Each set must contain a component data whose rows correspond to samples and columns to eigene genes.

setLabels A vector of character strings that label sets in multiME.

letterSubPlots logical: should subplots be lettered?

Letters optional specification of a sequence of letters for lettering. Defaults to "ABCD"...

excludeGrey logical: should the grey module eigene gene be excluded from the plots?

greyLabel label for the grey module. Usually either "grey" or the number 0.

plotDendrograms logical: should eigene gene dendrograms be plotted?

plotHeatmaps logical: should eigene gene network heatmaps be plotted?

setMargins logical: should margins be set? See par.

marDendo a vector of length 4 giving the margin setting for dendrogram plots. See par. If setMargins is TRUE and marDendo is not given, the function will provide reasonable default values.

marHeatmap a vector of length 4 giving the margin setting for heatmap plots. See par. If setMargins is TRUE and marDendo is not given, the function will provide reasonable default values.
Consensus eigengene networks consist of a fixed set of eigengenes "expressed" in several different sets. Network connection strengths are given by eigengene correlations. This function aims to visualize the networks as well as their similarities and differences across sets. The function partitions the screen appropriately and plots eigengene dendrograms in the top row, then a square matrix of plots: heatmap plots of eigengene networks in each set on the diagonal, heatmap plots of pairwise preservation networks below the diagonal, and barplots of aggregate network preservation of individual eigengenes above the diagonal. A preservation plot or barplot in the row i and column j of the square matrix represents the preservation between sets i and j.

Individual eigengenes are labeled by their name in the dendrograms; in the heatmaps and barplots they can optionally be labeled by color squares. For compatibility with other functions, the color labels are encoded in the eigengene names by prefixing the color with two letters, such as "MeTurquoise".

Two types of network preservation can be plotted: the "standard" is simply the difference between adjacencies in the two compared sets. The "hyperbolic" difference de-emphasizes the preservation of low adjacencies. When "both" is specified, standard preservation is plotted in the lower triangle and hyperbolic in the upper triangle of each preservation heatmap.

If the eigengenes are labeled by color, the bars in the barplot can be split into segments representing the contribution of each eigengene and labeled by the contribution. For example, a yellow segment...
in a bar labeled by a turquoise square represents the preservation of the adjacency between the yellow and turquoise eigengenes in the two networks compared by the barplot. For large numbers of eigengenes and/or sets, it may be difficult to get a meaningful plot fit a standard computer screen. In such cases we recommend using a device such as postscript or pdf where the user can specify large dimensions; such plots can be conveniently viewed in standard pdf or postscript viewers.

Value

None.

Author(s)

Peter Langfelder

References

For theory and applications of consensus eigengene networks, see

See Also

labeledHeatmap, labeledBarplot for annotated heatmaps and barplots;
hclust for hierarchical clustering and dendrogram plots

plotMat

Red and Green Color Image of Data Matrix

Description

This function produces a red and green color image of a data matrix using an RGB color specification. Larger entries are represented with reds of increasing intensity, and smaller entries are represented with greens of increasing intensity.

Usage

plotMat(x, nrgcols=50, rlabels=FALSE, clabels=FALSE, rcols=1, ccols=1, title="",...)

Arguments

x a matrix of numbers.
nrgcols the number of colors (>= 1) to be used in the red and green palette.
rlabels vector of character strings to be placed at the row tickpoints, labels for the rows of x.
clabels vector of character strings to be placed at the column tickpoints, labels for the columns of x.
rcols colors to be used for the labels of the rows of x. rcols can have either length 1, in which case all the labels are displayed using the same color, or the same length as rlabels, in which case a color is specified for the label of each row of x.
The function produces a matrix of plots containing pairwise scatterplots of given eigengenes, the distribution of their values and their pairwise correlations.

Usage

`plotMEpairs(  datME,  y = NULL,  main = "Relationship between module eigengenes",  clusterMEs = TRUE,  ...)`

Arguments

- `datME` a data frame containing expression data, with rows corresponding to samples and columns to genes. Missing values are allowed and will be ignored.
- `y` optional microarray sample trait vector. Will be treated as an additional eigengene.
- `main` main title for the plot.
- `clusterMEs` logical: should the module eigengenes be ordered by their dendrogram?
- `...` additional graphical parameters to the function `pairs`

Details

The function produces an N\times N matrix of plots, where N is the number of eigengenes. In the upper triangle it plots pairwise scatterplots of module eigengenes (plus the trait y, if given). On the diagonal it plots histograms of sample values for each eigengene. Below the diagonal, it displays the pairwise correlations of the eigengenes.
Value

None.

Author(s)

Steve Horvath

See Also

pairs

Description

Plot a barplot of gene significance.

Usage

plotmodulesignificance(
  geneSignificance, 
  colors, 
  boxplot = FALSE, 
  main = "Gene significance across modules," , 
  ylab = "Gene Significance", ...) 

Arguments

geneSignificance  
a numeric vector giving gene significances.

colors  
a character vector specifying module assignment for the genes whose significance is given in geneSignificance. The modules should be labeled by colors.

boxplot  
logical: should a boxplot be produced instead of a barplot?

main  
main title for the plot.

ylab  
y axis label for the plot.

...  
other graphical parameters to plot.

Details

Given individual gene significances and their module assignment, the function calculates the module significance for each module as the average gene significance of the genes within the module. The result is plotted in a barplot or boxplot form. Each bar or box is labeled by the corresponding module color.

Value

None.
Author(s)

Steve Horvath

References


See Also

barplot, boxplot

plotNetworkHeatmap  Network heatmap plot

Description

Network heatmap plot.

Usage

plotNetworkHeatmap(
  datExpr, plotGenes, useTOM = TRUE, power = 6, networkType = "unsigned", main = "Heatmap of the network")

Arguments

datExpr  a data frame containing expression data, with rows corresponding to samples and columns to genes. Missing values are allowed and will be ignored.

plotGenes a character vector giving the names of genes to be included in the plot. The names will be matched against names(datExpr).

useTOM  logical: should TOM be plotted (TRUE), or correlation-based adjacency (FALSE)?

power  soft-thresholding power for network construction.

networkType  a character string giving the network type. Recognized values are (unique abbreviations of) "unsigned", "signed", and "signed hybrid".

main  main title for the plot.

Details

The function constructs a network from the given expression data (selected by plotGenes) using the soft-thresholding procedure, optionally calculates Topological Overlap (TOM) and plots a heatmap of the network.

Note that all network calculations are done in one block and may fail due to memory allocation issues for large numbers of genes.
Value

None.

Author(s)

Steve Horvath

References


See Also

adjacency, TOMsimilarity

populationMeansInAdmixture

Estimate the population-specific mean values in an admixed population.

Description

Uses the expression values from an admixed population and estimates of the proportions of sub-populations to estimate the population specific mean values. For example, this function can be used to estimate the cell type specific mean gene expression values based on expression values from a mixture of cells. The method is described in Shen-Orr et al (2010) where it was used to estimate cell type specific gene expression levels based on a mixture sample.

Usage

populationMeansInAdmixture(
  datProportions, datE.Admixture,
  scaleProportionsTo1 = TRUE,
  scaleProportionsInCelltype = TRUE,
  setMissingProportionsToZero = FALSE)

Arguments

datProportions a matrix of non-negative numbers (ideally proportions) where the rows correspond to the samples (rows of datE.Admixture) and the columns correspond to the sub-populations of the mixture. The function calculates a mean expression value for each column of datProportions. Negative entries in datProportions lead to an error message. But the rows of datProportions do not have to sum to 1, see the argument scaleProportionsTo1.

datE.Admixture a matrix of numbers. The rows correspond to samples (mixtures of populations). The columns contain the variables (e.g. genes) for which the means should be estimated.
populationMeansInAdmixture

scaleProportionsTo1

logical. If set to TRUE (default) then the proportions in each row of datProportions are scaled so that they sum to 1, i.e. datProportions[i,] = datProportions[i,]/max(datProportions[i,]). In general, we recommend to set it to TRUE.

scaleProportionsInCelltype

logical. If set to TRUE (default) then the proportions in each cell types are rescaled and make the mean to 0.

setMissingProportionsToZero

logical. Default is FALSE. If set to TRUE then it sets missing values in datProportions to zero.

Details

The function outputs a matrix of coefficients resulting from fitting a regression model. If the proportions sum to 1, then i-th row of the output matrix reports the coefficients of the following model \( \text{lm}(\text{dat} \cdot \text{Admixture}[,i] - 1, \text{data} = \text{datProportions}) \). Aside, the minus 1 in the formula indicates that no intercept term will be fit. Under certain assumptions, the coefficients can be interpreted as the mean expression values in the sub-populations (Shen-Orr 2010).

Value

a numeric matrix whose rows correspond to the columns of datE.Admixture (e.g. to genes) and whose columns correspond to the columns of datProportions (e.g. sub populations or cell types).

Note

This can be considered a wrapper of the \texttt{lm} function.

Author(s)

Steve Horvath, Chaochao Cai

References


Examples

set.seed(1)
# this is the number of complex (mixed) tissue samples, e.g. arrays
m=10
# true count data (e.g. pure cells in the mixed sample)
datTrueCounts=as.matrix(data.frame(TrueCount1=rpois(m,lambda=16),
    TrueCount2=rpois(m,lambda=8),TrueCount3=rpois(m,lambda=4),
    TrueCount4=rpois(m,lambda=2)))
no.pure=dim(datTrueCounts)[[2]]

# now we transform the counts into proportions
divideBySum=Function(x) t(x)/sum(x)
datProportions= t(apply(datTrueCounts,1,divideBySum))
dimnames(datProportions)[[2]]=paste("TrueProp",1:dim(datTrueCounts)[[2]],sep=".")

# number of genes that are highly expressed in each pure population
no.genesPerPure=rep(5, no.pure)
no.genes= sum(no.genesPerPure)
GeneIndicator=rep(1:no.pure, no.genesPerPure)
# true mean values of the genes in the pure populations
# in the end we hope to estimate them from the mixed samples
datTrueMeans0=matrix( rnorm(no.genes*no.pure, sd=.3), nrow= no.genes, ncol=no.pure)
for (i in 1:no.pure ){
datTrueMeans0[GeneIndicator==i,i]= datTrueMeans0[GeneIndicator==i,i]+1
}
dimnames(datTrueMeans0)[[1]]=paste("Gene",1:dim(datTrueMeans0)[1],sep=". ")
dimnames(datTrueMeans0)[[2]]=paste("MeanPureCellType",1:dim(datTrueMeans0)[2],
   sep=". ")
# plot.mat(datTrueMeans0)
# simulate the (expression) values of the admixed population samples
noise=matrix(rnorm(m*no.genes, sd=.1),nrow=m,ncol= no.genes)
datE.Admixture = as.matrix(datProportions) %*% t(datTrueMeans0) + noise
dimnames(datE.Admixture)[[1]]=paste("MixedTissue",1:m,sep=". ")
datPredictedMeans = populationMeansInAdmixture(datProportions, datE.Admixture)
par(mfrow=c(2,2))
for (i in 1:4 ){
 verboseScatterplot(datPredictedMeans[,i],datTrueMeans0[,i],
 xlab="predicted mean",ylab="true mean",main="all populations")
 abline(0,1)
 }

# assume we only study 2 populations (ie we ignore the others)
selectPopulations=c(1,2)
datPredictedMeansTooFew = populationMeansInAdmixture(datProportions[,selectPopulations],
 datE.Admixture)
par(mfrow=c(2,2))
for (i in 1:length(selectPopulations) ){
 verboseScatterplot(datPredictedMeansTooFew[,i],datTrueMeans0[,i],
 xlab="predicted mean",ylab="true mean",main="too few populations")
 abline(0,1)
 }

# assume we erroneously add a population
datProportionsTooMany = data.frame(datProportions,WrongProp=sample(datProportions[,1]))
datPredictedMeansTooMany = populationMeansInAdmixture(datProportionsTooMany,
   datE.Admixture)
par(mfrow=c(2,2))
for (i in 1:4 ){
 verboseScatterplot(datPredictedMeansTooMany[,i],datTrueMeans0[,i],
 xlab="predicted mean",ylab="true mean",main="too many populations")
 abline(0,1)
 }

pquantile

Parallel quantile, median, mean
**pquantile**

**Description**
Calculation of “parallel” quantiles, medians, and means, across given arguments.

**Usage**

```r
pquantile(prob, ...)
pmedian(...)
pmean(...)
```

**Arguments**

- `prob` A number or vector of probabilities at which to calculate the quantile. See `quantile`.
- `...` Numeric arguments. All arguments must have the same dimensions. See details.

**Details**

Given the arguments, say x,y,z, of equal dimensions, the `pquantile` calculates and returns the quantile of the first components of x,y,z, then the second components, etc. Similarly, `pmedian` and `pmean` calculate the median and mean, respectively.

**Value**

A vector or array containing the quantiles, medians, or means. The dimensions are determined by the dimensions of the input arguments and whether the `prob` input is scalar or a vector. If any of the input variables have `dimnames`, the first non-NULL `dimnames` are copied into the output.

**Author(s)**

Peter Langfelder and Steve Horvath

**See Also**

`pmin` and `pmax` for analogous functions for minimum and maximum, `quantile`, `median`, `mean` for the underlying statistics.

**Examples**

```r
# Generate 2 simple matrices
a = matrix(c(1:12), 3, 4);
b = a + 1;
c = a + 2;

# Set the colnames on matrix a

colnames(a) = spaste("col", c(1:4));

# Example use

pquantile(prob = 0.5, a, b, c)
pquantile(prob = c(0, 0.5, 1), a, b, c);
pmean(a, b, c)
```
prependZeros

Pad numbers with leading zeros to specified total width

Description

This function pads the specified numbers with zeros to a specified total width.

Usage

prependZeros(x, width = max(nchar(x)))

Arguments

x Vector of numbers to be padded.
width Width to pad the numbers to.

Value

Character vector with the 0-padded numbers.

prepComma

Prepend a comma to a non-empty string

Description

Utility function that prepends a comma before the input string if the string is non-empty.

Usage

prepComma(s)

Arguments

s Character string.

Value

If s is non-empty, returns paste(",", s), otherwise returns s.

Author(s)

Peter Langfelder

Examples

prepComma("abc");
prepComma("");
Author(s)
Peter Langfelder

Examples
prependZeros(1:10)
prependZeros(1:10, 4)

Description
This function calculates several measures of gene network preservation. Given gene expression data in several individual data sets, it calculates the individual adjacency matrices, forms the preservation network and finally forms several summary measures of adjacency preservation for each node (gene) in the network.

Usage
preservationNetworkConnectivity(
  multiExpr,
  useSets = NULL, useGenes = NULL,
  corFnc = "cor", corOptions = "use='p'",
  networkType = "unsigned",
  power = 6,
  sampleLinks = NULL, nLinks = 5000,
  blockSize = 1000,
  setSeed = 12345,
  weightPower = 2,
  verbose = 2, indent = 0)

Arguments
multiExpr expression data in the multi-set format (see checkSets). A vector of lists, one per set. Each set must contain a component data that contains the expression data, with rows corresponding to samples and columns to genes or probes.

useSets optional specification of sets to be used for the preservation calculation. Defaults to using all sets.

useGenes optional specification of genes to be used for the preservation calculation. Defaults to all genes.

corFnc character string containing the name of the function to calculate correlation. Suggested functions include "cor" and "bicor".

corOptions further argument to the correlation function.

networkType a character string encoding network type. Recognized values are (unique abbreviations of) "unsigned", "signed", and "signed hybrid".
preservationNetworkConnectivity

power soft thresholding power for network construction. Should be a number greater than 1.
sampleLinks logical: should network connections be sampled (TRUE) or should all connections be used systematically (FALSE)?
nLinks number of links to be sampled. Should be set such that nLinks * nNeighbors be several times larger than the number of genes.
blockSize correlation calculations will be split into square blocks of this size, to prevent running out of memory for large gene sets.
setSeed seed to be used for sampling, for repeatability. If a seed already exists, it is saved before the sampling starts and restored upon exit.
weightPower power with which higher adjacencies will be weighted in weighted means
verbose integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
indent indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

The preservation network is formed from adjacencies of compared sets. For ‘complete’ preservations, all given sets are compared at once; for ‘pairwise’ preservations, the sets are compared in pairs. Unweighted preservations are simple mean preservations for each node; their weighted counterparts are weighted averages in which a preservation of adjacencies \( A_{ij}^{(1)} \) and \( A_{ij}^{(2)} \) of nodes \( i,j \) between sets 1 and 2 is weighted by \( \frac{(A_{ij}^{(1)} + A_{ij}^{(2)})}{2}^{\text{weightPower}} \). The hyperbolic preservation is based on \( \text{tanh}\left(\frac{\text{max} - \text{min}}{(\text{max} + \text{min})^2}\right) \), where \( \text{max} \) and \( \text{min} \) are the componentwise maximum and minimum of the compared adjacencies, respectively.

Value

A list with the following components:

pairwise a matrix with rows corresponding to genes and columns to unique pairs of given sets, giving the pairwise preservation of the adjacencies connecting the gene to all other genes.
complete a vector with one entry for each input gene containing the complete mean preservation of the adjacencies connecting the gene to all other genes.
pairwiseWeighted a matrix with rows corresponding to genes and columns to unique pairs of given sets, giving the pairwise weighted preservation of the adjacencies connecting the gene to all other genes.
completeWeighted a vector with one entry for each input gene containing the complete weighted mean preservation of the adjacencies connecting the gene to all other genes.
pairwiseHyperbolic a matrix with rows corresponding to genes and columns to unique pairs of given sets, giving the pairwise hyperbolic preservation of the adjacencies connecting the gene to all other genes.
completeHyperbolic a vector with one entry for each input gene containing the complete mean hyperbolic preservation of the adjacencies connecting the gene to all other genes.
pairwiseWeightedHyperbolic  
A matrix with rows corresponding to genes and columns to unique pairs of given  
sets, giving the pairwise weighted hyperbolic preservation of the adjacencies  
connecting the gene to all other genes.

completeWeightedHyperbolic  
A vector with one entry for each input gene containing the complete weighted  
hyperbolic mean preservation of the adjacencies connecting the gene to all other  
genes.

Author(s)

Peter Langfelder

References

Langfelder P, Horvath S (2007) Eigengene networks for studying the relationships between co-
expression modules. BMC Systems Biology 2007, 1:54

See Also

adjacency for calculation of adjacency;

Description

Implementation of a variant of K-means clustering for expression data.

Usage

projectiveKMeans(  
datExpr,  
preferredSize = 5000,  
nCenters = as.integer(min(ncol(datExpr)/20, preferredSize^2/ncol(datExpr))),  
sizePenaltyPower = 4,  
networkType = "unsigned",  
randomSeed = 54321,  
checkData = TRUE,  
imputeMissing = TRUE,  
maxIterations = 1000,  
verbose = 0, indent = 0)

Arguments

datExpr  
Expression data. A data frame in which columns are genes and rows are samples.  
NAs are allowed, but not too many.

preferredSize  
Preferred maximum size of clusters.

nCenters  
Number of initial clusters. Empirical evidence suggests that more centers will give  
a better preclustering; the default is an attempt to arrive at a reasonable number.
sizePenaltyPower  
parameter specifying how severe is the penalty for clusters that exceed preferredSize.

networkType  
network type. Allowed values are (unique abbreviations of) "unsigned", "signed", "signed hybrid". See adjacency.

randomSeed  
integer to be used as seed for the random number generator before the function starts. If a current seed exists, it is saved and restored upon exit.

checkData  
logical: should data be checked for genes with zero variance and genes and samples with excessive numbers of missing samples? Bad samples are ignored; returned cluster assignment for bad genes will be NA.

imputeMissing  
logical: should missing values in datExpr be imputed before the calculations start? The early imputation makes the code run faster but may produce slightly different results if re-running older calculations.

maxIterations  
maximum iterations to be attempted.

verbose  
integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.

indent  
indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

The principal aim of this function within WGCNA is to pre-cluster a large number of genes into smaller blocks that can be handled using standard WGCNA techniques.

This function implements a variant of K-means clustering that is suitable for co-expression analysis. Cluster centers are defined by the first principal component, and distances by correlation (more precisely, 1-correlation). The distance between a gene and a cluster is multiplied by a factor of $\max\{\frac{\text{clusterSize}}{\text{preferredSize}}, 1\}^{\text{sizePenaltyPower}}$, thus penalizing clusters whose size exceeds preferredSize. The function starts with randomly generated cluster assignment (hence the need to set the random seed for repeatability) and executes iterations of calculating new centers and reassigning genes to nearest center until the clustering becomes stable. Before returning, nearby clusters are iteratively combined if their combined size is below preferredSize.

The standard principal component calculation via the function svd fails from time to time (likely a convergence problem of the underlying lapack functions). Such errors are trapped and the principal component is approximated by a weighted average of expression profiles in the cluster. If verbose is set above 2, an informational message is printed whenever this approximation is used.

Value

A list with the following components:

clusters  
a numerical vector with one component per input gene, giving the cluster number in which the gene is assigned.

centers  
cluster centers, that is their first principal components.

Author(s)

Peter Langfelder
proportionsInAdmixture

Estimate the proportion of pure populations in an admixed population based on marker expression values.

Description

Assume that datE.Admixture provides the expression values from a mixture of cell types (admixed population) and you want to estimate the proportion of each pure cell type in the mixed samples (rows of datE.Admixture). The function allows you to do this as long as you provide a data frame MarkerMeansPure that reports the mean expression values of markers in each of the pure cell types.

Usage

proportionsInAdmixture(
  MarkerMeansPure,
  datE.Admixture,
  calculateConditionNumber = FALSE,
  coefToProportion = TRUE)

Arguments

MarkerMeansPure

is a data frame whose first column reports the name of the marker and the remaining columns report the mean values of the markers in each of the pure populations. The function will estimate the proportion of pure cells which correspond to columns 2 through of dim(MarkerMeansPure)[2] of MarkerMeansPure. Rows that contain missing values (NA) will be removed.

datE.Admixture

is a data frame of expression data, e.g. the columns of datE.Admixture could correspond to thousands of genes. The rows of datE.Admixture correspond to the admixed samples for which the function estimates the proportions of pure populations. Some of the markers specified in the first column of MarkerMeansPure should correspond to column names of datE.Admixture.

calculateConditionNumber

logical. Default is FALSE. If set to TRUE then it uses the kappa function to calculates the condition number of the matrix MarkerMeansPure[,,-1]. This allows one to determine whether the linear model for estimating the proportions is well specified. Type help(kappa) to learn more. kappa() computes by default (an estimate of) the 2-norm condition number of a matrix or of the R matrix of a QR decomposition, perhaps of a linear fit.

coeffToProportion

logical. By default, it is set to TRUE. When estimating the proportions the function fits a multivariate linear model. Ideally, the coefficients of the linear model correspond to the proportions in the admixed samples. But sometimes the coefficients take on negative values or do not sum to 1. If coefToProportion=TRUE then negative coefficients will be set to 0 and the remaining coefficients will be scaled so that they sum to 1.
Details

The methods implemented in this function were motivated by the gene expression deconvolution approach described by Abbas et al (2009), Lu et al (2003), Wang et al (2006). This approach can be used to predict the proportions of (pure) cells in a complex tissue, e.g. the proportion of blood cell types in whole blood. To define the markers, you may need to have expression data from pure populations. Then you can define markers based on a significant t-test or ANOVA across the pure populations. Next use the pure population data to estimate corresponding mean expression values. Hopefully, the array platforms and normalization methods for datE.MarkerMeansAdmixtureTranspose and MarkerMeansPure are comparable. When dealing with Affymetrix data: we have successfully used it on untransformed MAS5 data. For statisticians: To estimate the proportions, we use the coefficients of a linear model. Specifically: datCoef= t(1m(datE.MarkerMeansAdmixtureTranspose ~MarkerMeansPure[,,-1]), where datCoef is a matrix whose rows correspond to the mixed samples (rows of datE.Ad mixture) and the columns correspond to pure populations (e.g. cell types), i.e. the columns of MarkerMeansPure[,,-1]. More details can be found in Abbas et al (2009).

Value

A list with the following components

**PredictedProportions**

data frame that contains the predicted proportions. The rows of PredictedProportions correspond to the admixed samples, i.e. the rows of datE.Ad mixture. The columns of PredictedProportions correspond to the pure populations, i.e. the columns of MarkerMeansPure[,,-1].

datCoef=datCoef
data frame of numbers that is analogous to PredictedProportions. In general, datCoef will only be different from PredictedProportions if coefToProportion=TRUE. See the description of coefToProportion

**conditionNumber**

This is the condition number resulting from the kappa function. See the description of calculateConditionNumber.

**markersUsed**

vector of character strings that contains the subset of marker names (specified in the first column of MarkerMeansPure) that match column names of datE.Ad mixture and that contain non-missing pure mean values.

Note

This function can be considered a wrapper of the lm function.

Author(s)

Steve Horvath, Chaochao Cai

References


See Also

lm, kappa

propVarExplained

Proportion of variance explained by eigengenes.

Description

This function calculates the proportion of variance of genes in each module explained by the respective module eigengene.

Usage

propVarExplained(datExpr, colors, MES, corFnc = "cor", corOptions = "use = 'p'")

Arguments

datExpr expression data. A data frame in which columns are genes and rows are samples. NAs are allowed and will be ignored.
colors a vector giving module assignment for genes given in datExpr. Unique values should correspond to the names of the eigengenes in MES.
MES a data frame of module eigengenes in which each column is an eigengene and each row corresponds to a sample.
corFnc character string containing the name of the function to calculate correlation. Suggested functions include "cor" and "bicor".
corOptions further argument to the correlation function.

Details

For compatibility with other functions, entries in color are matched to a substring of names(MES) starting at position 3. For example, the entry "turquoise" in colors will be matched to the eigengene named "ME_turquoise". The first two characters of the eigengene name are ignored and can be arbitrary.

Value

A vector with one entry per eigengene containing the proportion of variance of the module explained by the eigengene.

Author(s)

Peter Langfelder

See Also

moduleEigengenes
Pathways with Corresponding Gene Markers - Compiled by Mike Palazzolo and Jim Wang from CHDI

Description

This matrix gives a predefined set of marker genes for many immune response pathways, as assembled by Mike Palazzolo and Jim Wang from CHDI, and colleagues. It is used with userListEnrichment to search user-defined gene lists for enrichment.

Usage

data(PWLists)

Format

A 124350 x 2 matrix of characters containing 2724 Gene / Category pairs. The first column (Gene) lists genes corresponding to a given category (second column). Each Category entry is of the form <gene set>__<reference>.

Source

For more information about this list, please see userListEnrichment.

Examples

data(PWLists)
head(PWLists)

qvalue

Estimate the q-values for a given set of p-values

Description

Estimate the q-values for a given set of p-values. The q-value of a test measures the proportion of false positives incurred (called the false discovery rate) when that particular test is called significant.

Usage

call.qvalue(p, lambda=seq(0,0.90,0.05), pi0.method="smoother", fdr.level=NULL, robust=FALSE, smooth.df=3, smooth.log.pi0=FALSE)

Arguments

p
A vector of p-values (only necessary input)

lambda
The value of the tuning parameter to estimate \( \pi_0 \). Must be in \([0,1)\). Optional, see Storey (2002).

pi0.method
Either "smoother" or "bootstrap"; the method for automatically choosing tuning parameter in the estimation of \( \pi_0 \), the proportion of true null hypotheses
qvalue

fdr.level A level at which to control the FDR. Must be in (0,1]. Optional; if this is selected, a vector of TRUE and FALSE is returned that specifies whether each q-value is less than fdr.level or not.

robust An indicator of whether it is desired to make the estimate more robust for small p-values and a direct finite sample estimate of pFDR. Optional.

smooth.df Number of degrees-of-freedom to use when estimating π₀ with a smoother. Optional.

smooth.log.pi0 If TRUE and pi0.method = "smoother", π₀ will be estimated by applying a smoother to a scatterplot of log π₀ estimates against the tuning parameter λ. Optional.

Details

If no options are selected, then the method used to estimate π₀ is the smoother method described in Storey and Tibshirani (2003). The bootstrap method is described in Storey, Taylor & Siegmund (2004).

Value

A list containing:

call function call

pi0 an estimate of the proportion of null p-values

qvalues a vector of the estimated q-values (the main quantity of interest)

pvalues a vector of the original p-values

significant if fdr.level is specified, and indicator of whether the q-value fell below fdr.level (taking all such q-values to be significant controls FDR at level fdr.level)

Note

This function is adapted from package qvalue. The reason we provide our own copy is that package qvalue contains additional functionality that relies on Tcl/Tk which has led to multiple problems. Our copy does not require Tcl/Tk.

Author(s)

John D. Storey <jstorey@u.washington.edu>, adapted for WGCNA by Peter Langfelder

References


**qvalue.restricted**

*qvalue convenience wrapper*

**Description**

This function calls *qvalue* on finite input p-values, optionally traps errors from the q-value calculation, and returns just the q values.

**Usage**

\[ qvalue.restricted(p, trapErrors = TRUE, \ldots) \]

**Arguments**

- **p**
  a vector of p-values. Missing data are allowed and will be removed.
- **trapErrors**
  logical: should errors generated by function *qvalue* trapped? If TRUE, the errors will be silently ignored and the returned q-values will all be NA.
- ...
  other arguments to function *qvalue*.

**Value**

A vector of q-values. Entries whose corresponding p-values were not finite will be NA.

**Author(s)**

Peter Langfelder

**See Also**

qvalue

**randIndex**

*Rand index of two partitions*

**Description**

Computes the Rand index, a measure of the similarity between two clusterings.

**Usage**

\[ randIndex(tab, adjust = TRUE) \]

**Arguments**

- **tab**
  a matrix giving the cross-tabulation table of two clusterings.
- **adjust**
  logical: should the "adjusted" version be computed?

**Value**

the Rand index of the input table.
Author(s)

Steve Horvath

References


Description

The function `rankPvalue` calculates the p-value for observing that an object (corresponding to a row of the input data frame `dats`) has a consistently high ranking (or low ranking) according to multiple ordinal scores (corresponding to the columns of the input data frame `dats`).

Usage

```r
rankPvalue(dats, columnweights = NULL, 
            na.last = "keep", ties.method = "average", 
            calculateQvalue = TRUE, pValueMethod = "all")
```

Arguments

- `dats` a data frame whose rows represent objects that will be ranked. Each column of `dats` represents an ordinal variable (which can take on negative values). The columns correspond to (possibly signed) object significance measures, e.g., statistics (such as Z statistics), ranks, or correlations.
- `columnweights` allows the user to input a vector of non-negative numbers reflecting weights for the different columns of `dats`. If it is set to `NULL` then all weights are equal.
- `na.last` controls the treatment of missing values (NAs) in the rank function. If `TRUE`, missing values in the data are put last (i.e. they get the highest rank values). If `FALSE`, they are put first; if `NA`, they are removed; if "keep" they are kept with rank NA. See `rank` for more details.
- `ties.method` represents the ties method used in the rank function for the percentile rank method. See `rank` for more details.
- `calculateQvalue` logical: should q-values be calculated? If set to `TRUE` then the function calculates corresponding q-values (local false discovery rates) using the qvalue package, see Storey JD and Tibshirani R. (2003). This option assumes that qvalue package has been installed.
- `pValueMethod` determines which method is used for calculating p-values. By default it is set to "all", i.e. both methods are used. If it is set to "rank" then only the percentile rank method is used. If it set to "scale" then only the scale method will be used.
**Details**

The function calculates asymptotic p-values (and optionally q-values) for testing the null hypothesis that the values in the columns of datS are independent. This allows us to find objects (rows) with consistently high (or low) values across the columns.

Example: Imagine you have 5 vectors of Z statistics corresponding to the columns of datS. Further assume that a gene has ranks 1,1,1,1,20 in the 5 lists. It seems very significant that the gene ranks number 1 in 4 out of the 5 lists. The function rankPvalue can be used to calculate a p-value for this occurrence.

The function uses the central limit theorem to calculate asymptotic p-values for two types of test statistics that measure consistently high or low ordinal values. The first method (referred to as percentile rank method) leads to accurate estimates of p-values if datS has at least 4 columns but it can be overly conservative. The percentile rank method replaces each column datS by the ranked version rank(datS[,i]) (referred to as low ranking) and by rank(-datS[,i]) (referred to as high ranking). Low ranking and high ranking allow one to find consistently small values or consistently large values of datS, respectively. All ranks are divided by the maximum rank so that the result lies in the unit interval [0,1]. In the following, we refer to rank/max(rank) as percentile rank. For a given object (corresponding to a row of datS) the observed percentile rank follows approximately a uniform distribution under the null hypothesis. The test statistic is defined as the sum of the percentile ranks (across the columns of datS). Under the null hypothesis that there is no relationship between the rankings of the columns of datS, this (row sum) test statistic follows a distribution that is given by the convolution of random uniform distributions. Under the null hypothesis, the individual percentile ranks are independent and one can invoke the central limit theorem to argue that the row sum test statistic follows asymptotically a normal distribution. It is well-known that the speed of convergence to the normal distribution is extremely fast in case of identically distributed uniform distributions. Even when datS has only 4 columns, the difference between the normal approximation and the exact distribution is negligible in practice (Killmann et al 2001). In summary, we use the central limit theorem to argue that the sum of the percentile ranks follows a normal distribution whose mean and variance can be calculated using the fact that the mean value of a uniform random variable (on the unit interval) equals 0.5 and its variance equals 1/12.

The second method for calculating p-values is referred to as scale method. It is often more powerful but its asymptotic p-value can only be trusted if either datS has a lot of columns or if the ordinal scores (columns of datS) follow an approximate normal distribution. The scale method scales (or standardizes) each ordinal variable (column of datS) so that it has mean 0 and variance 1. Under the null hypothesis of independence, the row sum follows approximately a normal distribution if the assumptions of the central limit theorem are met. In practice, we find that the second approach is often more powerful but it makes more distributional assumptions (if datS has few columns).

**Value**

A list whose actual content depends on which p-value methods is selected, and whether q-values are calculated. The following inner components are calculated, organized in outer components datoutrank and datoutscale:

- **pValueExtremeRank**
  This is the minimum between pValueLowRank and pValueHighRank, i.e. \( \min(pValueLow, pValueHigh) \)

- **pValueLowRank**
  Asymptotic p-value for observing a consistently low value across the columns of datS based on the rank method.

- **pValueHighRank**
  Asymptotic p-value for observing a consistently low value across the columns of datS based on the rank method.
RecutBlockwiseTrees

pValueExtremeScale
This is the minimum between pValueLowScale and pValueHighScale, i.e. min(pValueLow, pValueHigh)

pValueLowScale
Asymptotic p-value for observing a consistently low value across the columns of datS based on the Scale method.

pValueHighScale
Asymptotic p-value for observing a consistently low value across the columns of datS based on the Scale method.

qValueExtremeRank
local false discovery rate (q-value) corresponding to the p-value pValueExtremeRank

qValueLowRank
local false discovery rate (q-value) corresponding to the p-value pValueLowRank

qValueHighRank
local false discovery rate (q-value) corresponding to the p-value pValueHighRank

qValueExtremeScale
local false discovery rate (q-value) corresponding to the p-value pValueExtremeScale

qValueLowScale
local false discovery rate (q-value) corresponding to the p-value pValueLowScale

qValueHighScale
local false discovery rate (q-value) corresponding to the p-value pValueHighScale

Author(s)
Steve Horvath

References

See Also
rank, qvalue

RecutBlockwiseTrees Repeat blockwise module detection from pre-calculated data

Description
Given consensus networks constructed for example using blockwiseModules, this function (re-)detects modules in them by branch cutting of the corresponding dendrograms. If repeated branch cuts of the same gene network dendrograms are desired, this function can save substantial time by re-using already calculated networks and dendrograms.
Usage

recutBlockwiseTrees(
  datExpr,
  goodSamples, goodGenes,
  blocks,
  TOMFiles,
  dendrograms,
  corType = "pearson",
  networkType = "unsigned",
  deepSplit = 2,
  detectCutHeight = 0.995, minModuleSize = min(20, ncol(datExpr)/2 ),
  maxCoreScatter = NULL, minGap = NULL,
  maxAbsCoreScatter = NULL, minAbsGap = NULL,
  minSplitHeight = NULL, minAbsSplitHeight = NULL,
  useBranchEigennodeDissim = FALSE,
  minBranchEigennodeDissim = mergeCutHeight,
  pamStage = TRUE, pamRespectsDendro = TRUE,
  minCoreKME = 0.5, minCoreKMESize = minModuleSize/3,
  minKMEToStay = 0.3,
  reassignThreshold = 1e-6,
  mergeCutHeight = 0.15, impute = TRUE,
  trapErrors = FALSE, numericLabels = FALSE,
  verbose = 0, indent = 0,
  ...
)

Arguments

datExpr expression data. A data frame in which columns are genes and rows are samples. NAs are allowed, but not too many.
goodSamples a logical vector specifying which samples are considered "good" for the analysis. See goodSamplesGenes.
goodGenes a logical vector with length equal number of genes in multiExpr that specifies which genes are considered "good" for the analysis. See goodSamplesGenes.
blocks specification of blocks in which hierarchical clustering and module detection should be performed. A numeric vector with one entry per gene of multiExpr giving the number of the block to which the corresponding gene belongs.
TOMFiles a vector of character strings specifying file names in which the block-wise topological overlaps are saved.
dendrograms a list of length equal the number of blocks, in which each component is a hierarchical clustering dendrograms of the genes that belong to the block.
corType character string specifying the correlation to be used. Allowed values are (unique abbreviations of) "pearson" and "bicor", corresponding to Pearson and bid-weight midcorrelation, respectively. Missing values are handled using the pairwise.complete.obs option.
networkType network type. Allowed values are (unique abbreviations of) "unsigned", "signed", "signed hybrid". See adjacency.
**deepSplit**
integer value between 0 and 4. Provides a simplified control over how sensitive module detection should be to module splitting, with 0 least and 4 most sensitive. See `cutreeDynamic` for more details.

**detectCutHeight**
dendrogram cut height for module detection. See `cutreeDynamic` for more details.

**minModuleSize**
minimum module size for module detection. See `cutreeDynamic` for more details.

**maxCoreScatter**
maximum scatter of the core for a branch to be a cluster, given as the fraction of cutHeight relative to the 5th percentile of joining heights. See `cutreeDynamic` for more details.

**minGap**
minimum cluster gap given as the fraction of the difference between cutHeight and the 5th percentile of joining heights. See `cutreeDynamic` for more details.

**maxAbsCoreScatter**
maximum scatter of the core for a branch to be a cluster given as absolute heights. If given, overrides maxCoreScatter. See `cutreeDynamic` for more details.

**minAbsGap**
minimum cluster gap given as absolute height difference. If given, overrides minGap. See `cutreeDynamic` for more details.

**minSplitHeight**
Minimum split height given as the fraction of the difference between cutHeight and the 5th percentile of joining heights. Branches merging below this height will automatically be merged. Defaults to zero but is used only if minAbsSplitHeight below is NULL.

**minAbsSplitHeight**
Minimum split height given as an absolute height. Branches merging below this height will automatically be merged. If not given (default), will be determined from minSplitHeight above.

**useBranchEigennodeDissim**
Logical: should branch eigennode (eigengene) dissimilarity be considered when merging branches in Dynamic Tree Cut?

**minBranchEigennodeDissim**
Minimum consensus branch eigengene (eigengene) dissimilarity for branches to be considered separate. The branch eigengene dissimilarity in individual sets is simply 1-correlation of the eigennodes; the consensus is defined as quantile with probability consensusQuantile.

**pamStage**
logical. If TRUE, the second (PAM-like) stage of module detection will be performed. See `cutreeDynamic` for more details.

**pamRespectsDendro**
Logical, only used when pamStage is TRUE. If TRUE, the PAM stage will respect the dendrogram in the sense an object can be PAM-assigned only to clusters that lie below it on the branch that the object is merged into. See `cutreeDynamic` for more details.

**minCoreKME**
a number between 0 and 1. If a detected module does not have at least minModuleKMESize genes with eigengene connectivity at least minCoreKME, the module is disbanded (its genes are unlabeled and returned to the pool of genes waiting for module detection).

**minCoreKMESize**
see minCoreKME above.

**minKMEtoStay**
genes whose eigengene connectivity to their module eigengene is lower than minKMEtoStay are removed from the module.
reassignThreshold
p-value ratio threshold for reassigning genes between modules. See Details.
mergeCutHeight
dendrogram cut height for module merging.
impute
logical: should imputation be used for module eigengene calculation? See moduleEigengenes for more details.
trapErrors
logical: should errors in calculations be trapped?
numericLabels
logical: should the returned modules be labeled by colors (FALSE), or by numbers (TRUE)?
verbose
integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
indent
indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

... Other arguments.

Details
For details on blockwise module detection, see blockwiseModules. This function implements the module detection subset of the functionality of blockwiseModules; network construction and clustering must be performed in advance. The primary use of this function is to experiment with module detection settings without having to re-execute long network and clustering calculations whose results are not affected by the cutting parameters.

This function takes as input the networks and dendrograms that are produced by blockwiseModules. Working block by block, modules are identified in the dendrogram by the Dynamic Hybrid Tree Cut algorithm. Found modules are trimmed of genes whose correlation with module eigengene (KME) is less than minKMEtoStay. Modules in which fewer than minCoreKMEsize genes have KME higher than minCoreKME are disbanded, i.e., their constituent genes are pronounced unassigned.

After all blocks have been processed, the function checks whether there are genes whose KME in the module they assigned is lower than KME to another module. If p-values of the higher correlations are smaller than those of the native module by the factor reassignThresholdPS, the gene is re-assigned to the closer module.

In the last step, modules whose eigengenes are highly correlated are merged. This is achieved by clustering module eigengenes using the dissimilarity given by one minus their correlation, cutting the dendrogram at the height mergeCutHeight and merging all modules on each branch. The process is iterated until no modules are merged. See mergeCloseModules for more details on module merging.

Value
A list with the following components:

colors
a vector of color or numeric module labels for all genes.
unmergedColors
a vector of color or numeric module labels for all genes before module merging.
MEs
a data frame containing module eigengenes of the found modules (given by colors).
MEsOK
logical indicating whether the module eigengenes were calculated without errors.
**Author(s)**

Peter Langfelder

**References**


**See Also**

- `blockwiseModules` for full module calculation;
- `cutreeDynamic` for adaptive branch cutting in hierarchical clustering dendrograms;
- `mergeCloseModules` for merging of close modules.

**Description**

Given consensus networks constructed for example using `blockwiseConsensusModules`, this function (re-)detects modules in them by branch cutting of the corresponding dendrograms. If repeated branch cuts of the same gene network dendrograms are desired, this function can save substantial time by re-using already calculated networks and dendrograms.

**Usage**

```r
recutConsensusTrees(
  multiExpr,  # multi-expression data
  goodSamples,  # good samples
  goodGenes,  # good genes
  blocks,  # blocks
  TOMFiles,  # TOM files
  dendrograms,  # dendrograms
  corType = "pearson",  # correlation type
  networkType = "unsigned",  # network type
  deepSplit = 2,  # deep split
  detectCutHeight = 0.995,  # detect cut height
  minModuleSize = 20,  # minimum module size
  checkMinModuleSize = TRUE,  # check minimum module size
  maxCoreScatter = NULL,  # max core scatter
  minGap = NULL,  # min gap
  maxAbsCoreScatter = NULL,  # max abs core scatter
  minAbsGap = NULL,  # min abs gap
  minSplitHeight = NULL,  # min split height
  minAbsSplitHeight = NULL,  # min abs split height
  useBranchEigennodeDissim = FALSE,  # use branch eigennode dissim
  minBranchEigennodeDissim = detectCutHeight,  # minimum branch eigennode dissim
  pamStage = TRUE,  # PAM stage
  pamRespectsDendro = TRUE,  # PAM respects dendrogram
  trimmingConsensusQuantile = 0,  # trimming consensus quantile
  minCoreKME = 0.5,  # minimum core KME
  minCoreKMESize = minModuleSize/3,  # minimum core KME size
  minKMEtoStay = 0.2,  # minimum KME to stay
)
```
reassignThresholdPS = 1e-4,
mergeCutHeight = 0.15,
mergeConsensusQuantile = trimmingConsensusQuantile,
impute = TRUE,
trapErrors = FALSE,
numericLabels = FALSE,
verbose = 2, indent = 0)

Arguments

multiExpr expression data in the multi-set format (see checkSets). A vector of lists, one per set. Each set must contain a component data that contains the expression data, with rows corresponding to samples and columns to genes or probes.
goodSamples a list with one component per set. Each component is a logical vector specifying which samples are considered "good" for the analysis. See goodSamplesGenesMS.
goodGenes a logical vector with length equal number of genes in multiExpr that specifies which genes are considered "good" for the analysis. See goodSamplesGenesMS.
blocks specification of blocks in which hierarchical clustering and module detection should be performed. A numeric vector with one entry per gene of multiExpr giving the number of the block to which the corresponding gene belongs.
TOMFiles a vector of character strings specifying file names in which the block-wise topological overlaps are saved.
dendrograms a list of length equal the number of blocks, in which each component is a hierarchical clustering dendrograms of the genes that belong to the block.
corType character string specifying the correlation to be used. Allowed values are (unique abbreviations of) "pearson" and "bicor", corresponding to Pearson and bid-weight midcorrelation, respectively. Missing values are handled using the pairwise.complete.obs option.
networkType network type. Allowed values are (unique abbreviations of) "unsigned", "signed", "signed hybrid". See adjacency. Note that while no new networks are computed in this function, this parameter affects the interpretation of correlations in this function.
deepSplit integer value between 0 and 4. Provides a simplified control over how sensitive module detection should be to module splitting, with 0 least and 4 most sensitive. See cutreeDynamic for more details.
detectCutHeight dendrogram cut height for module detection. See cutreeDynamic for more details.
minModuleSize minimum module size for module detection. See cutreeDynamic for more details.
checkMinModuleSize logical: should sanity checks be performed on minModuleSize?
maxCoreScatter maximum scatter of the core for a branch to be a cluster, given as the fraction of cutHeight relative to the 5th percentile of joining heights. See cutreeDynamic for more details.
minGap minimum cluster gap given as the fraction of the difference between cutHeight and the 5th percentile of joining heights. See cutreeDynamic for more details.
**maxAbsCoreScatter**
maximum scatter of the core for a branch to be a cluster given as absolute heights. If given, overrides maxCoreScatter. See cutreeDynamic for more details.

**minAbsGap**
minimum cluster gap given as absolute height difference. If given, overrides minGap. See cutreeDynamic for more details.

**minSplitHeight**
Minimum split height given as the fraction of the difference between cutHeight and the 5th percentile of joining heights. Branches merging below this height will automatically be merged. Defaults to zero but is used only if minAbsSplitHeight below is NULL.

**minAbsSplitHeight**
Minimum split height given as an absolute height. Branches merging below this height will automatically be merged. If not given (default), will be determined from minSplitHeight above.

**useBranchEigenNodeDissim**
Logical: should branch eigennode (eigengene) dissimilarity be considered when merging branches in Dynamic Tree Cut?

**minBranchEigenNodeDissim**
Minimum consensus branch eigennode (eigengene) dissimilarity for branches to be considered separate. The branch eigennode dissimilarity in individual sets is simply 1-correlation of the eigennodes; the consensus is defined as quantile with probability consensusQuantile.

**pamStage**
logical. If TRUE, the second (PAM-like) stage of module detection will be performed. See cutreeDynamic for more details.

**pamRespectsDendro**
Logical, only used when pamStage is TRUE. If TRUE, the PAM stage will respect the dendrogram in the sense an object can be PAM-assigned only to clusters that lie below it on the branch that the object is merged into. See cutreeDynamic for more details.

**trimmingConsensusQuantile**
a number between 0 and 1 specifying the consensus quantile used for kME calculation that determines module trimming according to the arguments below.

**minCoreKME**
a number between 0 and 1. If a detected module does not have at least minModuleKMESize genes with eigengene connectivity at least minCoreKME, the module is disbanded (its genes are unlabeled and returned to the pool of genes waiting for module detection).

**minCoreKMESize**
see minCoreKME above.

**minKMEtoStay**
genesis whose eigengene connectivity to their module eigengene is lower than minKMEToStay are removed from the module.

**reassignThresholdPS**
per-set p-value ratio threshold for reassigning genes between modules. See Details.

**mergeCutHeight**
dendrogram cut height for module merging.

**mergeConsensusQuantile**
consensus quantile for module merging. See mergeCloseModules for details.

**impute**
logical: should imputation be used for module eigengene calculation? See moduleEigengenes for more details.

**trapErrors**
logical: should errors in calculations be trapped?
numericLabels logical: should the returned modules be labeled by colors (FALSE), or by numbers (TRUE)?
verbose integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
indent indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

For details on blockwise consensus module detection, see `blockwiseConsensusModules`. This function implements the module detection subset of the functionality of `blockwiseConsensusModules`; network construction and clustering must be performed in advance. The primary use of this function is to experiment with module detection settings without having to re-execute long network and clustering calculations whose results are not affected by the cutting parameters.

This function takes as input the networks and dendrograms that are produced by `blockwiseConsensusModules`. Working block by block, modules are identified in the dendrograms by the Dynamic Hybrid tree cut. Found modules are trimmed of genes whose consensus module membership kME (that is, correlation with module eigengene) is less than `minkmetostay`. Modules in which fewer than `minCorekMESize` genes have consensus KME higher than `minCoreKME` are disbanded, i.e., their constituent genes are pronounced unassigned.

After all blocks have been processed, the function checks whether there are genes whose KME in the module they assigned is lower than KME to another module. If p-values of the higher correlations are smaller than those of the native module by the factor `reassignThresholdPS` (in every set), the gene is re-assigned to the closer module.

In the last step, modules whose eigengenes are highly correlated are merged. This is achieved by clustering module eigengenes using the dissimilarity given by one minus their correlation, cutting the dendrogram at the height `mergeCutHeight` and merging all modules on each branch. The process is iterated until no modules are merged. See `mergeCloseModules` for more details on module merging.

Value

A list with the following components:

- `colors` module assignment of all input genes. A vector containing either character strings with module colors (if input numericLabels was unset) or numeric module labels (if numericLabels was set to TRUE). The color "grey" and the numeric label 0 are reserved for unassigned genes.
- `unmergedColors` module colors or numeric labels before the module merging step.
- `multiMEs` module eigengenes corresponding to the modules returned in `colors`, in multi-set format. A vector of lists, one per set, containing eigengenes, proportion of variance explained and other information. See `multiSetMEs` for a detailed description.

Note

Basic sanity checks are performed on given arguments, but it is left to the user’s responsibility to provide valid input.
redWhiteGreen

Author(s)
Peter Langfelder

References

See Also
blockwiseConsensusModules for the full blockwise modules calculation. Parts of its output are natural input for this function.
cutreeDynamic for adaptive branch cutting in hierarchical clustering dendrograms;
mergeCloseModules for merging of close modules.

redWhiteGreen Red-white-green color sequence

Description
Generate a red-white-green color sequence of a given length.

Usage
redWhiteGreen(n, gamma = 1)

Arguments
n number of colors to be returned
gamma color correction power

Details
The function returns a color vector that starts with pure green, gradually turns into white and then to red. The power gamma can be used to control the behaviour of the quarter- and three quarter-values (between red and white, and white and green, respectively). Higher powers will make the mid-colors more white, while lower powers will make the colors more saturated, respectively.

Value
A vector of colors of length n.

Author(s)
Peter Langfelder

Examples
par(mfrow = c(3, 1))
displayColors(redWhiteGreen(50));
displayColors(redWhiteGreen(50, 3));
displayColors(redWhiteGreen(50, 0.5));
relativeCorPredictionSuccess

*Compare prediction success*

**Description**

Compare prediction success of several gene screening methods.

**Usage**

```r
relativeCorPredictionSuccess(
  corPredictionNew,  
  corPredictionStandard,  
  corTestSet,  
  topNumber = 100)
```

**Arguments**

- `corPredictionNew`: Matrix of predictor statistics
- `corPredictionStandard`: Reference predictor statistics
- `corTestSet`: Correlations of predictor variables with trait in test set
- `topNumber`: A vector giving the numbers of top genes to consider

**Value**

Data frame with components

- `topNumber`: copy of the input `topNumber`
- `kruskalp`: Kruskal-Wallis p-values

**Author(s)**

Steve Horvath

**See Also**

- `corPredictionSuccess`
removeGreyME

Removes the grey eigengene from a given collection of eigengenes.

Description

Given module eigengenes either in a single data frame or in a multi-set format, removes the grey eigengenes from each set. If the grey eigengenes are not found, a warning is issued.

Usage

removeGreyME(MEs, greyMEName = paste(moduleColor.getMEprefix(), "grey", sep=""))

Arguments

MEs Module eigengenes, either in a single data frame (typically for a single set), or in a multi-set format. See checkSets for a description of the multi-set format.

greyMEName Name of the module eigengene (in each corresponding data frame) that corresponds to the grey color. This will typically be "PCgrey" or "MEgrey". If the module eigengenes were calculated using standard functions in this library, the default should work.

Value

Module eigengenes in the same format as input (either a single data frame or a vector of lists) with the grey eigengene removed.

Author(s)

Peter Langfelder, <Peter.Langfelder@gmail.com>

removePrincipalComponents

Remove leading principal components from data

Description

This function calculates a fixed number of the first principal components of the given data and returns the residuals of a linear regression of each column on the principal components.

Usage

removePrincipalComponents(x, n)

Arguments

x Input data, a numeric matrix. All entries must be non-missing and finite.

n Number of principal components to remove. This must be smaller than the smaller of the number of rows and columns in x.
Value

A matrix of residuals of the same dimensions as x.

Author(s)

Peter Langfelder

See Also

svd for singular value decomposition, lm for linear regression

returnGeneSetsAsList

Return pre-defined gene lists in several biomedical categories.

Description

This function returns gene sets for use with other R functions. These gene sets can include inputted lists of genes and files containing user-defined lists of genes, as well as a pre-made collection of brain, blood, and other biological lists. The function returns gene lists associated with each category for use with other enrichment strategies (i.e., GSVA).

Usage

returnGeneSetsAsList(
  fnIn = NULL, catNmIn = fnIn,
  useBrainLists = FALSE, useBloodAtlases = FALSE,
  useStemCellLists = FALSE, useBrainRegionMarkers = FALSE,
  useImmunePathwayLists = FALSE, geneSubset=NULL)

Arguments

fnIn A vector of file names containing user-defined lists. These files must be in one of three specific formats (see details section). The default (NULL) may only be used if one of the "use_____" parameters is TRUE.

catNmIn A vector of category names corresponding to each fnIn. This name will be appended to each overlap corresponding to that filename. The default sets the category names as the corresponding file names.

useBrainLists If TRUE, a pre-made set of brain-derived enrichment lists will be added to any user-defined lists for enrichment comparison. The default is FALSE. See references section for related references.

useBloodAtlases If TRUE, a pre-made set of blood-derived enrichment lists will be added to any user-defined lists for enrichment comparison. The default is FALSE. See references section for related references.

useStemCellLists If TRUE, a pre-made set of stem cell (SC)-derived enrichment lists will be added to any user-defined lists for enrichment comparison. The default is FALSE. See references section for related references.
useBrainRegionMarkers

If TRUE, a pre-made set of enrichment lists for human brain regions will be added to any user-defined lists for enrichment comparison. The default is FALSE. These lists are derived from data from the Allen Human Brain Atlas (http://human.brain-map.org/). See references section for more details.

useImmunePathwayLists

If TRUE, a pre-made set of enrichment lists for immune system pathways will be added to any user-defined lists for enrichment comparison. The default is FALSE. These lists are derived from the lab of Daniel R Saloman. See references section for more details.

geneSubset

A vector of gene (or other) identifiers. If entered, only genes in this list will be returned in the output, otherwise all genes in each category will be returned (default, geneSubset=NULL).

Details

User-inputted files for fnIn can be in one of three formats:

1) Text files (must end in ".txt") with one list per file, where the first line is the list descriptor and the remaining lines are gene names corresponding to that list, with one gene per line. For example: Ribosome RPS4 RPS8 ...

2) Gene / category files (must be csv files), where the first line is the column headers corresponding to Genes and Lists, and the remaining lines correspond to the genes in each list, for any number of genes and lists. For example: Gene, Category RPS4, Ribosome RPS8, Ribosome ... NDUF1, Mitochondria NDUF3, Mitochondria ... MAPT, AlzheimersDisease PSEN1, AlzheimersDisease PSEN2, AlzheimersDisease ...

3) Module membership (kME) table in csv format. Currently, the module assignment is the only thing that is used, so as long as the Gene column is 2nd and the Module column is 3rd, it doesn’t matter what is in the other columns. For example, PSID, Gene, Module, <other columns> <psid>, RPS4, blue, <other columns> <psid>, NDUF1, red, <other columns> <psid>, RPS8, blue, <other columns> <psid>, NDUF3, red, <other columns> <psid>, MAPT, green, <other columns> ...

Value

geneSets

A list of categories in alphabetical order, where each component of the list is a character vector of all genes corresponding to the named category. For example:

geneSets = list(category1=c("gene1","gene2"),category2=c("gene3","gene4","gene5"))

Author(s)

Jeremy Miller

References

Please see the help file for userListEnrichment in the WGCNA library for references for the pre-defined lists.

Examples

# Example: Return a list of genes for various immune pathways
genesets = returnGeneSetsAsList(useImmunePathwayLists=TRUE)
genesets[7:8]
rgcolors.func | Red and Green Color Specification

Description
This function creates a vector of n “contiguous” colors, corresponding to n intensities (between 0 and 1) of the red, green and blue primaries, with the blue intensities set to zero. The values returned by rgcolors.func can be used with a col= specification in graphics functions or in par.

Usage
rgcolors.func(n=50)

Arguments

n the number of colors (>= 1) to be used in the red and green palette.

Value
a character vector of color names. Colors are specified directly in terms of their RGB components with a string of the form "#RRGGBB", where each of the pairs RR, GG, BB consist of two hexadecimal digits giving a value in the range 00 to FF.

Author(s)
Sandrine Dudoit, <sandrine@stat.berkeley.edu>
Jane Fridlyand, <janef@stat.berkeley.edu>

See Also
plotcor, plotMat, colors, rgb, image.

Examples

rgcolors.func(n=5)
## The following vector is returned:
## "#00FF00" "#40BF00" "#808000" "#BF4000" "#FF0000"

scaleFreeFitIndex | Calculation of fitting statistics for evaluating scale free topology fit.

Description
The function scaleFreeFitIndex calculates several indices (fitting statistics) for evaluating scale free topology fit. The input is a vector (of connectivities) k. Next k is discretized into nBreaks number of equal-width bins. Let’s denote the resulting vector dk. The relative frequency for each bin is denoted p.dk.

Usage

scaleFreeFitIndex(k, nBreaks = 10, removeFirst = FALSE)
Arguments

- **k**
  numeric vector whose components contain non-negative values

- **nBreaks**
  positive integer. This determines the number of equal width bins.

- **removeFirst**
  logical. If TRUE then the first bin will be removed.

Value

Data frame with columns

- **R.squared.Nsft**
  the model fitting index (R.squared) from the following model: \( \text{lm}(\log.p.dk \sim \log.dk) \)

- **slope.Nsft**
  the slope estimate from model: \( \text{lm}(\log(p(k)) \sim \log(k)) \)

- **truncatedExponentialAdjR.squared**
  the adjusted R.squared measure from the truncated exponential model given by \( \text{lm2} = \text{lm}(\log.p.dk \sim \log.dk + dk) \).

Author(s)

Steve Horvath

---

**scaleFreePlot**  
Visual check of scale-free topology

Description

A simple visual check of scale-free network topology.

Usage

```r
scaleFreePlot(
  connectivity,
  nBreaks = 10,
  truncated = FALSE,
  removeFirst = FALSE,
  main = "", ...
)
```

Arguments

- **connectivity**
  vector containing network connectivities.

- **nBreaks**
  number of breaks in the connectivity dendrogram.

- **truncated**
  logical: should a truncated exponential fit be calculated and plotted in addition to the linear one?

- **removeFirst**
  logical: should the first bin be removed from the fit?

- **main**
  main title for the plot.

- **...**
  other graphical parameter to the plot function.
Details

The function plots a log-log plot of a histogram of the given connectivities, and fits a linear model plus optionally a truncated exponential model. The $R^2$ of the fit can be considered an index of the scale freedom of the network topology.

Value

None.

Author(s)

Steve Horvath

References


See Also

softconnectivity for connectivity calculation in weighted networks.

---

SCsLists

*Stem Cell-Related Genes with Corresponding Gene Markers*

Description

This matrix gives a predefined set of genes related to several stem cell (SC) types, as reported in two previously-published studies. It is used with userListEnrichment to search user-defined gene lists for enrichment.

Usage

data(SCsLists)

Format

A 14003 x 2 matrix of characters containing Gene / Category pairs. The first column (Gene) lists genes corresponding to a given category (second column). Each Category entry is of the form <Stem cell-related category>_<reference>, where the references can be found at userListEnrichment. Note that the matrix is sorted first by Category and then by Gene, such that all genes related to the same category are listed sequentially.

Source

For references used in this variable, please see userListEnrichment

Examples

data(SCsLists)
head(SCsLists)
Description

Given consensus eigengenes, the function calculates the average correlation preservation pair-wise for all pairs of sets.

Usage

```r
setCorrelationPreservation(
  multiME,
  setLabels,
  excludeGrey = TRUE, greyLabel = "grey",
  method = "absolute"
)
```

Arguments

- `multiME`: consensus module eigengenes in a multi-set format. A vector of lists with one list corresponding to each set. Each list must contain a component `data` that is a data frame whose columns are consensus module eigengenes.
- `setLabels`: names to be used for the sets represented in `multiME`.
- `excludeGrey`: logical: exclude the 'grey' eigengene from preservation measure?
- `greyLabel`: module label corresponding to the 'grey' module. Usually this will be the character string "grey" if the labels are colors, and the number 0 if the labels are numeric.
- `method`: character string giving the correlation preservation measure to use. Recognized values are (unique abbreviations of) "absolute", "hyperbolic".

Details

For each pair of sets, the function calculates the average preservation of correlation among the eigengenes. Two preservation measures are available, the absolute preservation (high if the two correlations are similar and low if they are different), and the hyperbolically scaled preservation, which de-emphasizes preservation of low correlation values.

Value

A data frame with each row and column corresponding to a set given in `multiME`, containing the pairwise average correlation preservation values. Names and rownames are set to entries of `setLabels`.

Author(s)

Peter Langfelder

References

shortenStrings

Shorten given character strings by truncating at a suitable separator.

Description

This function shortens given character strings so they are not longer than a given maximum length.

Usage

shortenStrings(strings, maxLength = 25, minLength = 10,
split = " ", fixed = TRUE,
ellipsis = "...", countEllipsisInLength = FALSE)

Arguments

strings Character strings to be shortened.
maxLength Maximum length (number of characters) in the strings to be retained. See details for when the returned strings can exceed this length.
minLength Minimum length of the returned strings. See details.
split Character string giving the split at which the strings can be truncated. This can be a literal string or a regular expression (if the latter, fixed below must be set to FALSE).
fixed Logical: should split be interpreted as a literal specification (TRUE) or as a regular expression (FALSE)?
ellipsis Character string that will be appended to every shorten string, to indicate that the string has been shortened.
countEllipsisInLength Logical: should the length of the ellipsis count toward the minimum and maximum length?

Details

Strings whose length (number of characters) is at most maxLength are returned unchanged. For those that are longer, the function uses gregexpr to search for the occurrences of split in each given character string. If such occurrences are found at positions between minLength and maxLength, the string will be truncated at the last such split; otherwise, the string will be truncated at maxLength. The ellipsis is appended to each truncated string.

Value

A character vector of strings, shortened as necessary. If the input strings had non-NULL dimensions and dimnames, these are copied to the output.

Author(s)

Peter Langfelder
See Also

`gregexpr`, the workhorse pattern matching function `formatLabels` for splitting strings into multiple lines

sigmoidAdjacencyFunction

**Sigmoid-type adjacency function.**

Description

Sigmoid-type function that converts a similarity to a weighted network adjacency.

Usage

```
sigmoidAdjacencyFunction(ss, mu = 0.8, alpha = 20)
```

Arguments

- `ss` similarity, a number between 0 and 1. Can be given as a scalar, vector or a matrix.
- `mu` shift parameter.
- `alpha` slope parameter.

Details

The sigmoid adjacency function is defined as \( \frac{1}{1 + \exp[-\alpha(ss - \mu)]]} \).

Value

Adjacencies returned in the same form as the input `ss`

Author(s)

Steve Horvath

References

signedKME

Signed eigengene-based connectivity

Description

Calculation of (signed) eigengene-based connectivity, also known as module membership.

Usage

```r
signedKME(
  datExpr, datME,
  outputColumnName = "kME",
  corFnc = "cor", corOptions = "use = 'p'")
```

Arguments

- **datExpr**: a data frame containing the gene expression data. Rows correspond to samples and columns to genes. Missing values are allowed and will be ignored.
- **datME**: a data frame containing module eigengenes. Rows correspond to samples and columns to module eigengenes.
- **outputColumnName**: a character string specifying the prefix of column names of the output.
- **corFnc**: character string specifying the function to be used to calculate co-expression similarity. Defaults to Pearson correlation. Any function returning values between -1 and 1 can be used.
- **corOptions**: character string specifying additional arguments to be passed to the function given by corFnc. Use "use = 'p', method = 'spearman'" to obtain Spearman correlation.

Details

Signed eigengene-based connectivity of a gene in a module is defined as the correlation of the gene with the corresponding module eigengene. The samples in datExpr and datME must be the same.

Value

A data frame in which rows correspond to input genes and columns to module eigengenes, giving the signed eigengene-based connectivity of each gene with respect to each eigengene.

Author(s)

Steve Horvath

References


**signumAdjacencyFunction**

*Hard-thresholding adjacency function*

**Description**

This function transforms correlations or other measures of similarity into an unweighted network adjacency.

**Usage**

```
signumAdjacencyFunction(corMat, threshold)
```

**Arguments**

- `corMat`: a matrix of correlations or other measures of similarity.
- `threshold`: threshold for connecting nodes: all nodes whose `corMat` is above the threshold will be connected in the resulting network.

**Value**

An unweighted adjacency matrix of the same dimensions as the input `corMat`.

**Author(s)**

Steve Horvath

**References**


**See Also**

`adjacency` for soft-thresholding and creating weighted networks.

---

**simulateDatExpr**

*Simulation of expression data*

**Description**

Simulation of expression data with a customizable modular structure and several different types of noise.
Usage

simulateDatExpr(
  eigengenes,
  nGenes,
  modProportions,
  minCor = 0.3,
  maxCor = 1,
  corPower = 1,
  signed = FALSE,
  propNegativeCor = 0.3,
  geneMeans = NULL,
  backgroundNoise = 0.1,
  leaveOut = NULL,
  nSubmoduleLayers = 0,
  nScatteredModuleLayers = 0,
  averageNGenesInSubmodule = 10,
  averageExprInSubmodule = 0.2,
  submoduleSpacing = 2,
  verbose = 1, indent = 0)

Arguments

eigengenes a data frame containing the seed eigengenes for the simulated modules. Rows correspond to samples and columns to modules.

nGenes total number of genes to be simulated.

modProportions a numeric vector with length equal the number of eigengenes in eigengenes plus one, containing fractions of the total number of genes to be put into each of the modules and into the "grey module", which means genes not related to any of the modules. See details.

minCor minimum correlation of module genes with the corresponding eigengene. See details.

maxCor maximum correlation of module genes with the corresponding eigengene. See details.

corPower controls the dropoff of gene-eigengene correlation. See details.

signed logical: should the genes be simulated as belonging to a signed network? If TRUE, all genes will be simulated to have positive correlation with the eigengene. If FALSE, a proportion given by propNegativeCor will be simulated with negative correlations of the same absolute values.

propNegativeCor proportion of genes to be simulated with negative gene-eigengene correlations. Only effective if signed is FALSE.

geneMeans optional vector of length nGenes giving desired mean expression for each gene. If not given, the returned expression profiles will have mean zero.

backgroundNoise amount of background noise to be added to the simulated expression data.

leaveOut optional specification of modules that should be left out of the simulation, that is their genes will be simulated as unrelated ("grey"). This can be useful when simulating several sets, in some which a module is present while in others it is absent.
simulateDatExpr

nSubmoduleLayers
number of layers of ordered submodules to be added. See details.
nScatteredModuleLayers
number of layers of scattered submodules to be added. See details.
averageNGenesInSubmodule
average number of genes in a submodule. See details.
averageExprInSubmodule
average strength of submodule expression vectors.
submoduleSpacing
a number giving submodule spacing: this multiple of the submodule size will lie between the submodule and the next one.
verbose
integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
indent
indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

Given eigengenes can be unrelated or they can exhibit non-trivial correlations. Each module is simulated separately from others. The expression profiles are chosen such that their correlations with the eigengene run from just below maxCor to minCor (hence minCor must be between 0 and 1, not including the bounds). The parameter corPower can be chosen to control the behaviour of the simulated correlation with the gene index; values higher than 1 will result in the correlation approaching minCor faster and lower than 1 slower.

Numbers of genes in each module are specified (as fractions of the total number of genes nGenes) by modProportions. The last entry in modProportions corresponds to the genes that will be simulated as unrelated to anything else ("grey" genes). The proportion must add up to 1 or less. If the sum is less than one, the remaining genes will be partitioned into groups and simulated to be "close" to the proper modules, that is with small but non-zero correlations (between minCor and 0) with the module eigengene.

If signed is set FALSE, the correlation for some of the module genes is chosen negative (but the absolute values remain the same as they would be for positively correlated genes). To ensure consistency for simulations of multiple sets, the indices of the negatively correlated genes are fixed and distributed evenly.

In addition to the primary module structure, a secondary structure can be optionally simulated. Modules in the secondary structure have sizes chosen from an exponential distribution with mean equal averageNGenesInSubmodule. Expression vectors simulated in the secondary structure are simulated with expected standard deviation chosen from an exponential distribution with mean equal averageExprInSubmodule; the higher this coefficient, the more pronounced will the submodules be in the main modules. The secondary structure can be simulated in several layers; their number is given by SubmoduleLayers. Genes in these submodules are ordered in the same order as in the main modules.

In addition to the ordered submodule structure, a scattered submodule structure can be simulated as well. This structure can be viewed as noise that tends to correlate random groups of genes. The size and effect parameters are the same as for the ordered submodules, and the number of layers added is controlled by nScatteredModuleLayers.

Value

A list with the following components:
`simulateDatExpr5Modules`  

**datExpr**  
simulated expression data in a data frame whose columns correspond genes and rows to samples.

**setLabels**  
simulated module assignment. Module labels are numeric, starting from 1. Genes simulated to be outside of proper modules have label 0. Modules that are left out (specified in `leaveout`) are indicated as 0 here.

**allLabels**  
simulated module assignment. Genes that belong to leftout modules (specified in `leaveout`) are indicated by their would-be assignment here.

**labelOrder**  
a vector specifying the order in which labels correspond to the given eigen-genes, that is `labelOrder[1]` is the label assigned to module whose seed is `eigengenes[,1]` etc.

**Author(s)**

Peter Langfelder

**References**

A short description of the simulation method can also be found in the Supplementary Material to the article


**See Also**

- `simulateEigengeneNetwork` for a simulation of eigengenes with a given causal structure;
- `simulateModule` for simulations of individual modules;
- `simulateDatExpr5Modules` for a simplified interface to expression simulations;
- `simulateMultiExpr` for a simulation of several related data sets.

---

**simulateDatExpr5Modules**  

*Simplified simulation of expression data*

**Description**

This function provides a simplified interface to the expression data simulation, at the cost of considerably less flexibility.

**Usage**

```r
simulateDatExpr5Modules(
  nGenes = 2000,
  colorLabels = c("turquoise", "blue", "brown", "yellow", "green"),
  simulateProportions = c(0.1, 0.08, 0.06, 0.04, 0.02),
  METurquoise, MEblue, MEbrown, MEyellow, MEgreen,
  SDnoise = 1, backgroundCor = 0.3)
```
**simulateDatExpr5Modules**

**Arguments**

- nGenes: total number of genes to be simulated.
- colorLabels: labels for simulated modules.
- simulateProportions: a vector of length 5 giving proportions of the total number of genes to be placed in each individual module. The entries must be positive and sum to at most 1. If the sum is less than 1, the leftover genes will be simulated outside of modules.
- MEturquoise: seed module eigengene for the first module.
- MEBlue: seed module eigengene for the second module.
- MEbrown: seed module eigengene for the third module.
- MEyellow: seed module eigengene for the fourth module.
- MEGreen: seed module eigengene for the fifth module.
- SDnoise: level of noise to be added to the simulated expressions.
- backgroundCor: background correlation. If non-zero, a component will be added to all genes such that the average correlation of otherwise unrelated genes will be backgroundCor.

**Details**

Roughly one-third of the genes are simulated with a negative correlation to their seed eigengene. See the functions `simulatemodule` and `simulateDatExpr` for more details.

**Value**

A list with the following components:

- datExpr: the simulated expression data in a data frame, with rows corresponding to samples and columns to genes.
- truemodule: a vector with one entry per gene containing the simulated module membership.
- datME: a data frame containing a copy of the input module eigengenes.

**Author(s)**

Steve Horvath and Peter Langfelder

**See Also**

- `simulatemodule` for simulation of individual modules;
- `simulateDatExpr` for a more comprehensive data simulation interface.
simulateEigengeneNetwork

Simulate eigengene network from a causal model

Description

Simulates a set of eigengenes (vectors) from a given set of causal anchors and a causal matrix.

Usage

simulateEigengeneNetwork(
  causeMat, anchorIndex, anchorVectors, noise = 1, verbose = 0, indent = 0)

Arguments

causeMat     causal matrix. The entry \([i,j]\) is the influence (path coefficient) of vector \(j\) on vector \(i\).
anchorIndex  specifies the indices of the anchor vectors.
anchorVectors a matrix giving the actual anchor vectors as columns. Their number must equal the length of anchorIndex.
noise        standard deviation of the noise added to each simulated vector.
verbose      level of verbosity. 0 means silent.
indent       indentation for diagnostic messages. Zero means no indentation; each unit adds two spaces.

Details

The algorithm starts with the anchor vectors and iteratively generates the rest from the path coefficients given in the matrix causeMat.

Value

A list with the following components:

eigengenes generated eigengenes.
causeMat   a copy of the input causal matrix
levels     useful for debugging. A vector with one entry for each eigengene giving the number of generations of parents of the eigengene. Anchors have level 0, their direct causal children have level 1 etc.
anchorIndex a copy of the input anchorIndex.

Author(s)

Peter Langfelder
**simulateModule**  

Simulate a gene co-expression module

**Description**

Simulation of a single gene co-expression module.

**Usage**

```r
simulateModule(
  ME,  
  nGenes, 
  nNearGenes = 0, 
  minCor = 0.3, maxCor = 1, corPower = 1, 
  signed = FALSE, propNegativeCor = 0.3, 
  geneMeans = NULL, 
  verbose = 0, indent = 0)
```

**Arguments**

- `ME`: seed module eigengene.
- `nGenes`: number of genes in the module to be simulated. Must be non-zero.
- `nNearGenes`: number of genes to be simulated with low correlation with the seed eigengene.
- `minCor`: minimum correlation of module genes with the eigengene. See details.
- `maxCor`: maximum correlation of module genes with the eigengene. See details.
- `corPower`: controls the dropoff of gene-eigengene correlation. See details.
- `signed`: logical: should the genes be simulated as belonging to a signed network? If `true`, all genes will be simulated to have positive correlation with the eigengene. If `false`, a proportion given by `propNegativeCor` will be simulated with negative correlations of the same absolute values.
- `propNegativeCor`: proportion of genes to be simulated with negative gene-eigengene correlations. Only effective if `signed` is `false`.
- `geneMeans`: optional vector of length `nGenes` giving desired mean expression for each gene. If not given, the returned expression profiles will have mean zero.
- `verbose`: integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
- `indent`: indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

**Details**

Module genes are simulated around the eigengene by choosing them such that their (expected) correlations with the seed eigengene decrease progressively from (just below) `maxCor` to `minCor`. The genes are otherwise independent from one another. The variable `corPower` determines how fast the correlation drops towards `minCor`. Higher powers lead to a faster drop-off; `corPower` must be above zero but need not be integer.
If `signed` is FALSE, the genes are simulated so as to be part of an unsigned network module, that is some genes will be simulated with a negative correlation with the seed eigengene (but of the same absolute value that a positively correlated gene would be simulated with). The proportion of genes with negative correlation is controlled by `propNegativeCor`.

Optionally, the function can also simulate genes that are "near" the module, meaning they are simulated with a low but non-zero correlation with the seed eigengene. The correlations run between `minCor` and zero.

Value

A matrix containing the expression data with rows corresponding to samples and columns to genes.

Author(s)

Peter Langfelder

References

A short description of the simulation method can also be found in the Supplementary Material to the article


See Also

- `simulateEigengeneNetwork` for a simulation of eigengenes with a given causal structure;
- `simulateDatExpr` for simulations of whole datasets consisting of multiple modules;
- `simulateDatExprSModules` for a simplified interface to expression simulations;
- `simulateMultiExpr` for a simulation of several related data sets.

**simulateMultiExpr**

**Simulate multi-set expression data**

**Description**

Simulation of expression data in several sets with relate module structure.

**Usage**

```r
simulateMultiExpr(eigengenes,
    nGenes,
    modProportions,
    minCor = 0.5, maxCor = 1,
    corPower = 1,
    backgroundNoise = 0.1,
    leaveOut = NULL,
    signed = FALSE,
    propNegativeCor = 0.3,
    geneMeans = NULL,
    )
```
**simulateMultiExpr**

nSubmoduleLayers = 0,
nScatteredModuleLayers = 0,
averageNGenesInSubmodule = 10,
averageExprInSubmodule = 0.2,
submoduleSpacing = 2,
verbose = 1, indent = 0)

**Arguments**

- **eigengenes** the seed eigengenes for the simulated modules in a multi-set format. A list with one component per set. Each component is again a list that must contain a component data. This is a data frame of seed eigengenes for the corresponding data set. Columns correspond to modules, rows to samples. Number of samples in the simulated data is determined from the number of samples of the eigengenes.

- **nGenes** integer specifying the number of simulated genes.

- **modProportions** a numeric vector with length equal the number of eigengenes in eigengenes plus one, containing fractions of the total number of genes to be put into each of the modules and into the "grey module", which means genes not related to any of the modules. See details.

- **minCor** minimum correlation of module genes with the corresponding eigengene. See details.

- **maxCor** maximum correlation of module genes with the corresponding eigengene. See details.

- **corPower** controls the dropoff of gene-eigengene correlation. See details.

- **backgroundNoise** amount of background noise to be added to the simulated expression data.

- **leaveOut** optional specification of modules that should be left out of the simulation, that is their genes will be simulated as unrelated ("grey"). A logical matrix in which columns correspond to sets and rows to modules. Wherever TRUE, the corresponding module in the corresponding data set will not be simulated, that is its genes will be simulated independently of the eigengene.

- **signed** logical: should the genes be simulated as belonging to a signed network? If TRUE, all genes will be simulated to have positive correlation with the eigengene. If FALSE, a proportion given by propNegativeCor will be simulated with negative correlations of the same absolute values.

- **propNegativeCor** proportion of genes to be simulated with negative gene-eigengene correlations. Only effective if signed is FALSE.

- **geneMeans** optional vector of length nGenes giving desired mean expression for each gene. If not given, the returned expression profiles will have mean zero.

- **nSubmoduleLayers** number of layers of ordered submodules to be added. See details.

- **nScatteredModuleLayers** number of layers of scattered submodules to be added. See details.

- **averageNGenesInSubmodule** average number of genes in a submodule. See details.

- **averageExprInSubmodule** average strength of submodule expression vectors.
submoduleSpacing

a number giving submodule spacing: this multiple of the submodule size will
lie between the submodule and the next one.

verbose

integer level of verbosity. Zero means silent, higher values make the output
progressively more and more verbose.

indent

indentation for diagnostic messages. Zero means no indentation, each unit adds
two spaces.

Details

For details of simulation of individual data sets and the meaning of individual set simulation argu-
ments, see simulateDatExpr. This function simulates several data sets at a time and puts the result
in a multi-set format. The number of genes is the same for all data sets. Module memberships are
also the same, but modules can optionally be “dissolved”, that is their genes will be simulated as
unassigned. Such “dissolved”, or left out, modules can be specified in the matrix leaveOut.

Value

A list with the following components:

multiExpr

simulated expression data in multi-set format analogous to that of the input
eigengenes. A list with one component per set. Each component is again a
list that must contains a component data. This is a data frame of expression
data for the corresponding data set. Columns correspond to genes, rows to sam-

ples.

setLabels

a matrix of dimensions (number of genes) times (number of sets) that contains
module labels for each genes in each simulated data set.

allLabels

a matrix of dimensions (number of genes) times (number of sets) that contains
the module labels that would be simulated if no module were left out using
leaveOut. This means that all columns of the matrix are equal; the columns are
repeated for convenience so allLabels has the same dimensions as setLabels.

labelOrder

a matrix of dimensions (number of modules) times (number of sets) that contains
the order in which module labels were assigned to genes in each set. The first
label is assigned to genes 1...(module size of module labeled by first label), the
second label to the following batch of genes etc.

Author(s)

Peter Langfelder

References

A short description of the simulation method can also be found in the Supplementary Material to
the article
Langfelder P, Horvath S (2007) Eigengene networks for studying the relationships between co-

The material is posted at http://www.genetics.ucla.edu/labs/horvath/CoexpressionNetwork/EigengeneNetwork/Suppleme
simulateSmallLayer

See Also

simulateEigengeneNetwork for a simulation of eigengenes with a given causal structure;
simulateDatExpr for simulation of individual data sets;
simulateDatExpr5Modules for a simple simulation of a data set consisting of 5 modules;
simulateModule for simulations of individual modules;

Description

This function simulates a set of small modules. The primary purpose is to add a submodule structure to the main module structure simulated by simulateDatExpr.

Usage

simulateSmallLayer(
  order, nsamples,
  minCor = 0.3, maxCor = 0.5, corPower = 1,
  averageModuleSize, averageExpr,
  moduleSpacing,
  verbose = 4, indent = 0)

Arguments

order a vector giving the simulation order for vectors. See details.
nsamples integer giving the number of samples to be simulated.
minCor a multiple of maxCor (see below) giving the minimum correlation of module genes with the corresponding eigengene. See details.
maxCor maximum correlation of module genes with the corresponding eigengene. See details.
corPower controls the dropoff of gene-eigengene correlation. See details.
averageModuleSize average number of genes in a module. See details.
averageExpr average strength of module expression vectors.
moduleSpacing a number giving module spacing: this multiple of the module size will lie between the module and the next one.
verbose integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
indent indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.
Details

Module eigenvectors are chosen randomly and independently. Module sizes are chosen randomly from an exponential distribution with mean equal to \( \text{averageModulSize} \). Two thirds of genes in each module are simulated as proper module genes and one third as near-module genes (see \texttt{simulateModule} for details). Between each successive pairs of modules a number of genes given by \texttt{moduleSpacing} will be left unsimulated (zero expression). Module expression, that is the expected standard deviation of the module expression vectors, is chosen randomly from an exponential distribution with mean equal to \( \text{averageExpr} \). The expression profiles are chosen such that their correlations with the eigengene run from just below \( \text{maxCor} \) to \( \text{minCor} \) \( \times \) \( \text{maxCor} \) (hence \( \text{minCor} \) must be between 0 and 1, not including the bounds). The parameter \texttt{corPower} can be chosen to control the behaviour of the simulated correlation with the gene index; values higher than 1 will result in the correlation approaching \( \text{minCor} \) \( \times \) \( \text{maxCor} \) faster and lower than 1 slower.

The simulated genes will be returned in the order given in \texttt{order}.

Value

A matrix of simulated gene expressions, with dimension \((\text{nSamples}, \text{length(order)})\).

Author(s)

Peter Langfelder

See Also

\texttt{simulateModule} for simulation of individual modules;
\texttt{simulateDatExpr} for the main gene expression simulation function.

\begin{tabular}{ll}
\textbf{sizeGrWindow} & \textit{Opens a graphics window with specified dimensions} \\
\end{tabular}

Description

If a graphic device window is already open, it is closed and re-opened with specified dimensions (in inches); otherwise a new window is opened.

Usage

\texttt{sizeGrWindow(width, height)}

Arguments

- \texttt{width} \quad \text{desired width of the window, in inches.}
- \texttt{height} \quad \text{desired height of the window, in inches.}

Value

None.

Author(s)

Peter Langfelder
softConnectivity

Calculates connectivity of a weighted network.

Description

Given expression data or a similarity, the function constructs the adjacency matrix and for each node calculates its connectivity, that is the sum of the adjacency to the other nodes.

Usage

softConnectivity(
  datExpr,
  corFnc = "cor", corOptions = "use = 'p'",
  type = "unsigned",
  power = if (type == "signed") 15 else 6,
  blockSize = 1500,
  minNSamples = NULL,
  verbose = 2, indent = 0)

softConnectivity.fromSimilarity(
  similarity,
  type = "unsigned",
  power = if (type == "signed") 15 else 6,
  blockSize = 1500,
  verbose = 2, indent = 0)

Arguments

datExpr a data frame containing the expression data, with rows corresponding to samples and columns to genes.
similarity a similarity matrix: a square symmetric matrix with entries between -1 and 1.
corFnc character string giving the correlation function to be used for the adjacency calculation. Recommended choices are "cor" and "bicor", but other functions can be used as well.
corOptions character string giving further options to be passed to the correlation function.
type network type. Allowed values are (unique abbreviations of) "unsigned", "signed", "signed hybrid".
power soft thresholding power.
blockSize block size in which adjacency is to be calculated. Too low (say below 100) may make the calculation inefficient, while too high may cause R to run out of physical memory and slow down the computer. Should be chosen such that an array of doubles of size (number of genes) * (block size) fits into available physical memory.
minNSamples minimum number of samples available for the calculation of adjacency for the adjacency to be considered valid. If not given, defaults to the greater of .minNSamples (currently 4) and number of samples divided by 3. If the number of samples falls below this threshold, the connectivity of the corresponding gene will be returned as NA.
verbose integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.

indent indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Value
A vector with one entry per gene giving the connectivity of each gene in the weighted network.

Author(s)
Steve Horvath

References

See Also
adjacency

spaste
Space-less paste

Description
A convenient wrapper for the paste function with sep="".

Usage
spaste(...)

Arguments
... standard arguments to function paste except sep.

Value
The result of the corresponding paste.

Note
Do not use the sep argument. Using will lead to an error.

Author(s)
Peter Langfelder

See Also
paste
standardColors

Examples

```r
a = 1;
paste("a=" , a);
spaste("a=" , a);
```

```r
standardColors

Colors this library uses for labeling modules.

Description

Returns the vector of color names in the order they are assigned by other functions in this library.

Usage

standardColors(n = NULL)

Arguments

n
Number of colors requested. If NULL, all (approx. 450) colors will be returned. Any other invalid argument such as less than one or more than maximum (length(standardColors())) will trigger an error.

Value

A vector of character color names of the requested length.

Author(s)

Peter Langfelder, <Peter.Langfelder@gmail.com>

Examples

```r
standardColors(10);
```

standardScreeningBinaryTrait

Standard screening for binatry traits

Description

The function standardScreeningBinaryTrait computes widely used statistics for relating the columns of the input data frame (argument data) to a binary sample trait (argument y). The statistics include Student t-test p-value and the corresponding local false discovery rate (known as q-value, Storey et al 2004), the fold change, the area under the ROC curve (also known as C-index), mean values etc. If the input option KruskalTest is set to TRUE, it also computes the Kruskal Wallist test p-value and corresponding q-value. The Kruskal Wallis test is a non-parametric, rank-based group comparison test.
Usage

```r
standardScreeningBinaryTrait(
  datExpr, y,
  corFnc = cor, corOptions = list(use = 'p'),
  kruskalTest = FALSE, qValues = FALSE,
  var.equal=FALSE, na.action="na.exclude",
  getAreaUnderROC = TRUE)
```

Arguments

- `datExpr`: a data frame or matrix whose columns will be related to the binary trait
- `y`: a binary vector whose length (number of components) equals the number of rows of `datExpr`
- `corFnc`: correlation function. Defaults to Pearson correlation.
- `corOptions`: a list specifying options to `corFnc`. An empty list must be specified as `list()`, (supplying `NULL` instead will trigger an error).
- `kruskalTest`: logical: should the Kruskal test be performed?
- `qValues`: logical: should the q-values be calculated?
- `var.equal`: logical input parameter for the Student t-test. It indicates whether to treat the two variances (corresponding to the binary grouping) as being equal. If `TRUE` then the pooled variance is used to estimate the variance otherwise the Welch (or Satterthwaite) approximation to the degrees of freedom is used. Warning: here the default value is `TRUE` which is different from the default value of `t.test`. Type `help(t.test)` for more details.
- `na.action`: character string for the Student t-test: indicates what should happen when the data contain missing values NAs.
- `getAreaUnderROC`: logical: should area under the ROC curve be calculated? The calculation slows the function down somewhat.

Value

A data frame whose rows correspond to the columns of `datExpr` and whose columns report

- `ID`: column names of the input `datExpr`.
- `corPearson`: pearson correlation with a binary numeric version of the input variable. The numeric variable equals 1 for level 1 and 2 for level 2. The levels are given by `levels(factor(y))`.
- `t.Student`: Student’s t-test statistic
- `pvalueStudent`: two-sided Student t-test p-value.
- `qvalueStudent`: (if input `qValues==TRUE`) q-value (local false discovery rate) based on the Student T-test p-value (Storey et al 2004).
- `foldChange`: a (signed) ratio of mean values. If the mean in the first group (corresponding to level 1) is larger than that of the second group, it equals `meanFirstGroup/meanSecondGroup`. But if the mean of the second group is larger than that of the first group it equals `-meanSecondGroup/meanFirstGroup` (notice the minus sign).
- `meanFirstGroup`: means of columns in input `datExpr` across samples in the first group.
**meanSecondGroup**

Means of columns in input `datExpr` across samples in the second group.

**SE.FirstGroup**

Standard errors of columns in input `datExpr` across samples in the first group. Recall that $SE(x) = \sqrt{var(x)/n}$ where $n$ is the number of non-missing values of $x$.

**SE.SecondGroup**

Standard errors of columns in input `datExpr` across samples in the second group.

**areaUnderROC**

The area under the ROC, also known as the concordance index or C-index. This is a measure of discriminatory power. The measure lies between 0 and 1 where 0.5 indicates no discriminatory power. 0 indicates that the "opposite" predictor has perfect discriminatory power. To compute it we use the function `rcorr.cens` with `outx=TRUE` (from Frank Harrel's package Hmisc). Only present if input `getAreaUnderROC` is `TRUE`.

**nPresentsSamples**

Number of samples with finite measurements for each gene.

If input `kruskalTest` is `TRUE`, the following columns further summarize results of Kruskal-Wallis test:

**stat.Kruskal**

Kruskal-Wallis test statistic.

**stat.Kruskal.signed**

(Warning: experimental) Kruskal-Wallis test statistic including a sign that indicates whether the average rank is higher in second group (positive) or first group (negative).

**pvalue.kruskal**

Kruskal-Wallis test p-values.

**q.kruskal**

q-values corresponding to the Kruskal-Wallis test p-value (if input `qValues` is `TRUE`).

**Author(s)**

Steve Horvath

**References**


**Examples**

```r
require(survival) # For is.Surv in rcorr.cens
m=50
y=sample(c(1,2),m,replace=TRUE)
datExprSignal=simulateModule(scale(y),30)
datExprNoise=simulateModule(rnorm(m),150)
datExpr=data.frame(datExprSignal,datExprNoise)

Result1=standardScreeningBinaryTrait(datExpr)
Result1[1:5,]

# use unequal variances and calculate q-values
Result2=standardScreeningBinaryTrait(datExpr, var.equal=FALSE,qValue=TRUE)
Result2[1:5,]
```
# calculate Kruskal Wallis test and q-values
Result3=standardScreeningBinaryTrait(datExpr,y,kruskalTest=TRUE,qValue=TRUE)
Result3[1:5,]

---

```r
# calculate Kruskal Wallis test and q-values
Result3=standardScreeningBinaryTrait(datExpr,y,kruskalTest=TRUE,qValue=TRUE)
Result3[1:5,]
```

---

## Description

The function `standardScreeningCensoredTime` computes association measures between the columns of the input data `datExpr` and a censored time variable (e.g. survival time). The censored time is specified using two input variables "time" and "event". The event variable is binary where 1 indicates that the event took place (e.g. the person died) and 0 indicates censored (i.e. lost to follow up). The function fits univariate Cox regression models (one for each column of `datExpr`) and outputs a Wald test p-value, a logrank p-value, corresponding local false discovery rates (known as q-values, Storey et al 2004), hazard ratios. Further it reports the concordance index (also known as area under the ROC curve) and optionally results from dichotomizing the columns of `datExpr`.

## Usage

```r
standardScreeningCensoredTime(
  time,
  event,
  datExpr,
  percentiles = seq(from = 0.1, to = 0.9, by = 0.2),
  dichotomizationResults = FALSE,
  qValues = TRUE,
  fastCalculation = TRUE)
```

## Arguments

- **time**: numeric variable showing time to event or time to last follow up.
- **event**: Input variable time specifies the time to event or time to last follow up. Input variable event indicates whether the event happened (=1) or whether there was censoring (=0).
- **datExpr**: a data frame or matrix whose columns will be related to the censored time.
- **percentiles**: numeric vector which is only used when `dichotomizationResults=T`. Each value should lie between 0 and 1. For each value specified in the vector percentiles, a binary vector will be defined by dichotomizing the column value according to the corresponding quantile. Next a corresponding p-value will be calculated.
- **dichotomizationResults**: logical. If this option is set to TRUE then the values of the columns of `datExpr` will be dichotomized and corresponding Cox regression p-values will be calculated.
- **qValues**: logical. If this option is set to TRUE (default) then q-values will be calculated for the Cox regression p-values.
Standard Screening Censored Time

fastCalculation
 logical. If set to TRUE, the function outputs correlation test p-values (and q-values) for correlating the columns of datE with the expected hazard (if no covariate is fit). Specifically, the expected hazard is defined as the deviance residual of an intercept only Cox regression model. The results are very similar to those resulting from a univariate Cox model where the censored time is regressed on the columns of dat. Specifically, this computational speed up is facilitated by the insight that the p-values resulting from a univariate Cox regression coxph(Surv(time,event)~datE[,i]) are very similar to those from corPvalueFisher(cor(devianceResidual,datE[,i]), nSamples).

Details

If input option fastCalculation=TRUE, then the function outputs correlation test p-values (and q-values) for correlating the columns of datE with the expected hazard (if no covariate is fit). Specifically, the expected hazard is defined as the deviance residual of an intercept only Cox regression model. The results are very similar to those resulting from a univariate Cox model where the censored time is regressed on the columns of dat. Specifically, this computational speed up is facilitated by the insight that the p-values resulting from a univariate Cox regression coxph(Surv(time,event)~datE[,i]) are very similar to those from corPvalueFisher(cor(devianceResidual,datE[,i]), nSamples).

Value

If fastCalculation is FALSE, the function outputs a data frame whose rows correspond to the columns of datE and whose columns report

<table>
<thead>
<tr>
<th>ID</th>
<th>column names of the input data datExpr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>pvalueWald</td>
<td>Wald test p-value from fitting a univariate Cox regression model where the censored time is regressed on each column of datExpr.</td>
</tr>
<tr>
<td>qValueWald</td>
<td>local false discovery rate (q-value) corresponding to the Wald test p-value.</td>
</tr>
<tr>
<td>pvalueLogrank</td>
<td>Logrank p-value resulting from the Cox regression model. Also known as score test p-value. For large sample sizes this could be similar to the Wald test p-value.</td>
</tr>
<tr>
<td>qValueLogrank</td>
<td>local false discovery rate (q-value) corresponding to the Logrank test p-value.</td>
</tr>
<tr>
<td>HazardRatio</td>
<td>hazard ratio resulting from the Cox model. If the value is larger than 1, then high values of the column are associated with shorter time, e.g. increased hazard of death. A hazard ratio equal to 1 means no relationship between the column and time. HR&lt;1 means that high values are associated with longer time, i.e. lower hazard.</td>
</tr>
<tr>
<td>CI.LowerLimitHR</td>
<td>Lower bound of the 95 percent confidence interval of the hazard ratio.</td>
</tr>
<tr>
<td>CI.UpperLimitHR</td>
<td>Upper bound of the 95 percent confidence interval of the hazard ratio.</td>
</tr>
<tr>
<td>C.index</td>
<td>concordance index, also known as C-index or area under the ROC curve. Calculated with the rcorr.cens option outx=TRUE (ties are ignored).</td>
</tr>
<tr>
<td>MinimumDichotPvalue</td>
<td>This is the smallest p-value from the dichotomization results. To see which dichotomized variable (and percentile) corresponds to the minimum, study the following columns.</td>
</tr>
</tbody>
</table>
This column report the p-value when the column is dichotomized according to the specified percentile (here 0.1). The percentiles are specified in the input option percentiles.

The p-value resulting from using a correlation test to relate the expected hazard (deviance residual) with each (undichotomized) column of dat. Specifically, the Fisher transformation is used to calculate the p-value for the Pearson correlation. The resulting p-value should be very similar to that of a univariate Cox regression model.

Local false discovery rate (q-value) corresponding to pvalueDeviance.

Pearson correlation between the expected hazard (deviance residual) with each (undichotomized) column of datExpr.

Author(s)

Steve Horvath

standardScreeningNumericTrait

Standard screening for numeric traits

Description

Standard screening for numeric traits based on Pearson correlation.

Usage

standardScreeningNumericTrait(datExpr, yNumeric, corFnc = cor, corOptions = list(use = 'p'), alternative = c("two.sided", "less", "greater"), qValues = TRUE, areaUnderROC = TRUE)

Arguments

datExpr       data frame containing expression data (or more generally variables to be screened), with rows corresponding to samples and columns to genes (variables)
yNumeric      a numeric vector giving the trait measurements for each sample
corFnc        correlation function. Defaults to Pearson correlation but can also be bicor.
corOptions    list specifying additional arguments to be passed to the correlation function given by corFnc.
alternative   alternative hypothesis for the correlation test
qValues       logical: should q-values be calculated?
areaUnderROC  logical: should are under the receiver-operating curve be calculated?

Details

The function calculates the correlations, associated p-values, area under the ROC, and q-values
Value

Data frame with the following components:

- ID: Gene (or variable) identifiers copied from colnames(datExpr)
- cor: correlations of all genes with the trait
- Z: Fisher Z statistics corresponding to the correlations
- pvalueStudent: Student p-values of the correlations
- qvalueStudent: (if input qValues==TRUE) q-values of the correlations calculated from the p-values
- AreaUnderROC: (if input areaUnderROC==TRUE) area under the ROC
- nPresentSamples: number of samples present for the calculation of each association.

Author(s)

Steve Horvath

See Also

standardscreeningbinarytrait, standardscreeningcensoredtime

stdErr

Standard error of the mean of a given vector.

Description

Returns the standard error of the mean of a given vector. Missing values are ignored.

Usage

stdErr(x)

Arguments

- x: a numeric vector

Value

Standard error of the mean of x.

Author(s)

Steve Horvath
**stratifiedBarplot**  
*Bar plots of data across two splitting parameters*

**Description**
This function takes an expression matrix which can be split using two separate splitting parameters (ie, control vs AD with multiple brain regions), and plots the results as a barplot. Group average, standard deviations, and relevant Kruskal-Wallis p-values are returned.

**Usage**

```r
stratifiedBarplot(
  expAll,  
groups, split, subset,
  genes = NA,
  scale = "N", graph = TRUE,
  las1 = 2, cex1 = 1.5, ...)
```

**Arguments**
- **expAll**: An expression matrix, with rows as samples and genes/probes as columns. If genes=NA, then column names must be included.
- **groups**: A character vector corresponding to the samples in expAll, with each element the group name of the relevant sample or NA for samples not in any group. For example: NA, NA, NA, Con, Con, Con, Con, AD, AD, AD, AD, NA, NA. This trait will be plotted as adjacent bars for each split.
- **split**: A character vector corresponding to the samples in expAll, with each element the group splitting name of the relevant sample or NA for samples not in any group. For example: NA, NA, NA, Hip, Hip, EC, EC, Hip, Hip, EC, EC, NA, NA. This trait will be plotted as the same color across each split of the barplot. For the function to work properly, the same split values should be inputted for each group.
- **subset**: A list of one or more genes to compare the expression with. If the list contains more than one gene, the first element contains the group name. For example, Ribosomes, RPL3, RPL4, RPS3.
- **genes**: If entered, this parameter is a list of gene/probe identifiers corresponding to the columns in expAll.
- **scale**: For subsets of genes that include more than one gene, this parameter determines how the genes are combined into a single value. Currently, there are five options: 1) ("N")o scaling (default); 2) first divide each gene by the ("A")verage across samples; 3) first scale genes to ("Z")-score across samples; 4) only take the top ("H")ub gene (ignore all but the highest-connected gene); and 5) take the ("M")odule eigengene. Note that these scaling methods have not been sufficiently tested, and should be considered experimental.
- **graph**: If TRUE (default), bar plot is made. If FALSE, only the results are returned, and no plot is made.
- **cex1**: Sets the graphing parameters of cex.axis and cex.names (default=1.5)
- **las1**: Sets the graphing parameter las (default=2).
Other graphing parameters allowed in the barplot function. Note that the parameters for cex.axis, cex.names, and las are superseded by cex1 and las1 and will therefore be ignored.

**Value**

- **splitGroupMeans**: The group/split averaged expression across each group and split combination. This is the height of the bars in the graph.
- **splitGroupSDs**: The standard deviation of group/split expression across each group and split combination. This is the height of the error bars in the graph.
- **splitPvals**: Kruskal-Wallis p-values for each splitting parameter across groups.
- **grouppvals**: Kruskal-Wallis p-values for each group parameter across splits.

**Author(s)**

Jeremy Miller

**See Also**

`barplot`, `verboseBarplot`

**Examples**

```r
# Example: first simulate some data
set.seed(100)
ME.A = sample(1:100,50); ME.B = sample(1:100,50)
ME.C = sample(1:100,50); ME.D = sample(1:100,50)
ME1 = data.frame(ME.A, ME.B, ME.C, ME.D)
simDatA = simulateDatExpr(ME1,1000,c(0.2,0.1,0.08,0.05,0.3), signed=TRUE)
datExpr = simDatA$datExpr+5
datExpr[1:10,] = datExpr[1:10,]+2
datExpr[41:50,] = datExpr[41:50,]-1

# Now split up the data and plot it!
subset = c("Random Genes", "Gene.1", "Gene.234", "Gene.56", "Gene.789")
groups = rep(c("A","A","A","B","B","B","C","C","C"),5)
split = c(rep("ZZ",10), rep("YY",10), rep("XX",10), rep("Ww",10), rep("VV",10))
par(mfrow = c(1,1))
results = stratifiedBarplot(datExpr, groups, split, subset)
results

# Now plot it the other way
results = stratifiedBarplot(datExpr, split, groups, subset)
```

---

**subsetTOM**  
*Topological overlap for a subset of a whole set of genes*

**Description**

This function calculates topological overlap of a subset of vectors with respect to a whole data set.
Usage

subsetTOM(
  datExpr,
  subset,
  corFnc = "cor", corOptions = "use = 'p'",
  networkType = "unsigned",
  power = 6,
  verbose = 1, indent = 0)

Arguments

datExpr a data frame containing the expression data of the whole set, with rows corresponding to samples and columns to genes.

subset a single logical or numeric vector giving the indices of the nodes for which the TOM is to be calculated.

corFnc character string giving the correlation function to be used for the adjacency calculation. Recommended choices are "cor" and "bicor", but other functions can be used as well.

corOptions character string giving further options to be passed to the correlation function.

networkType character string giving network type. Allowed values are (unique abbreviations of) "unsigned", "signed", "signed hybrid". See adjacency.

power soft-thresholding power for network construction.

verbose integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.

indent indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

This function is designed to calculated topological overlaps of small subsets of large expression data sets, for example in individual modules.

Value

A matrix of dimensions n*n, where n is the number of entries selected by block.

Author(s)

Peter Langfelder

References


See Also

TOMsimilarity for standard calculation of topological overlap.
swapTwoBranches  

Select, swap, or reflect branches in a dendrogram.

Description

swapTwoBranches takes the a gene tree object and two genes as input, and swaps the branches containing these two genes at the nearest branch point of the dendrogram.

reflectBranch takes the a gene tree object and two genes as input, and reflects the branch containing the first gene at the nearest branch point of the dendrogram.

selectBranch takes the a gene tree object and two genes as input, and outputs indices for all genes in the branch containing the first gene, up to the nearest branch point of the dendrogram.

Usage

swapTwoBranches(hierTOM, g1, g2)
reflectBranch(hierTOM, g1, g2, both = FALSE)
selectBranch(hierTOM, g1, g2)

Arguments

hierTOM  
A hierarchical clustering object (or gene tree) that is used to plot the dendrogram. For example, the output object from the function hclust or fastcluster::hclust. Note that elements of hierTOM$order MUST be named (for example, with the corresponding gene name).

$g_1$  
Any gene in the branch of interest.

$g_2$  
Any gene in a branch directly adjacent to the branch of interest.

both  
Logical: should the selection include the branch gene $g_2$?

Value

swapTwoBranches and reflectBranch return a hierarchical clustering object with the hierTOM$order variable properly adjusted, but all other variables identical as the heirTOM input.

selectBranch returns a numeric vector corresponding to all genes in the requested branch.

Author(s)

Jeremy Miller

Examples

```r
## Example: first simulate some data.

MEturquoise = sample(1:100, 50)
MEblue = c(MEturquoise[1:25], sample(1:100, 25))
MEbrown = sample(1:100, 50)
MEyellow = sample(1:100, 50)
MEgreen = c(MEyellow[1:30], sample(1:100, 20))
MERed = c(MEbrowm[1:20], sample(1:100, 30))
ME = data.frame(MEturquoise, MEblue, MEbrown, MEyellow, MEgreen, MERed)
dat1 = simulateDatExpr(ME, 400, c(0.16, 0.12, 0.11, 0.10, 0.10, 0.09, 0.15),
  signed=TRUE)
```
```r
tOMI = TOMsimilarityFromExpr(dat1$datExpr, networkType="signed")
colnames(TOMI) <- rownames(TOMI) <- colnames(dat1$datExpr)
tree1 = fastcluster::hclust(as.dist(1-TOMI), method="average")
colorh = labels2colors(dat1$allLabels)

plotDendroAndColors(tree1, colorh, dendroLabels=FALSE)

## Reassign modules using the selectBranch and chooseOneHubInEachModule functions

datExpr = dat1$datExpr
hubs = chooseOneHubInEachModule(datExpr, colorh)
colorh2 = rep("grey", length(colorh))
colorh2[selectBranch(tree1, hubs["blue"], hubs["turquoise"])] = "blue"
colorh2[selectBranch(tree1, hubs["turquoise"], hubs["blue"])] = "turquoise"
colorh2[selectBranch(tree1, hubs["green"], hubs["yellow"])] = "green"
colorh2[selectBranch(tree1, hubs["yellow"], hubs["green"])] = "yellow"
colorh2[selectBranch(tree1, hubs["red"], hubs["brown"])] = "red"
colorh2[selectBranch(tree1, hubs["brown"], hubs["red"])] = "brown"
plotDendroAndColors(tree1, cbind(colorh, colorh2), c("Old", "New"), dendroLabels=FALSE)

## Now swap and reflect some branches, then optimize the order of the branches

# Open a suitably sized graphics window

sizeGrWindow(12, 9);

# partition the screen for 3 dendrogram + module color plots

layout(matrix(c(1:6), 6, 1), heights = c(0.8, 0.2, 0.8, 0.2, 0.8, 0.2));

plotDendroAndColors(tree1, colorh2, dendroLabels=FALSE, main="Starting Dendrogram", setLayout = FALSE)
tree1 = swapTwoBranches(tree1, hubs["red"], hubs["turquoise"])
plotDendroAndColors(tree1, colorh2, dendroLabels=FALSE, main="Swap blue/turquoise and red/brown", setLayout = FALSE)
tree1 = reflectBranch(tree1, hubs["blue"], hubs["green"])
plotDendroAndColors(tree1, colorh2, dendroLabels=FALSE, main="Reflect turquoise/blue", setLayout = FALSE)
```

---

**TOMplot**

*Graphical representation of the Topological Overlap Matrix*

**Description**

Graphical representation of the Topological Overlap Matrix using a heatmap plot combined with the corresponding hierarchical clustering dendrogram and module colors.

**Usage**

```
TOMplot(dissim,
```
dendro,  
Colors = NULL,  
ColorsLeft = Colors,  
terrainColors = FALSE,  
setLayout = TRUE,  
...)

Arguments

dissim a matrix containing the topological overlap-based dissimilarity
dendro the corresponding hierarchical clustering dendrogram
Colors optional specification of module colors to be plotted on top
ColorsLeft optional specification of module colors on the left side. If NULL, Colors will be used.
terrainColors logical: should terrain colors be used?
setLayout logical: should layout be set? If TRUE, standard layout for one plot will be used. Note that this precludes multiple plots on one page. If FALSE, the user is responsible for setting the correct layout.
...
other graphical parameters to heatmap.

Details

The standard heatmap function uses the layout function to set the following layout (when Colors is given):

\[
\begin{array}{c}
0 & 0 & 5 \\
0 & 0 & 2 \\
4 & 1 & 3 \\
\end{array}
\]

To get a meaningful heatmap plot, user-set layout must respect this geometry.

Value

None.

Author(s)

Steve Horvath and Peter Langfelder

See Also

heatmap, the workhorse function doing the plotting.
Description

Calculation of the topological overlap matrix from a given adjacency matrix.

Usage

TOMsimilarity(adjMat, TOMType = "unsigned", TOMDenom = "min", verbose = 1, indent = 0)
TOMdist(adjMat, TOMType = "unsigned", TOMDenom = "min", verbose = 1, indent = 0)

Arguments

adjMat adjacency matrix, that is a square, symmetric matrix with entries between 0 and 1 (negative values are allowed if TOMType="signed").
TOMType a character string specifying TOM type to be calculated. One of "unsigned", "signed". If "unsigned", the standard TOM will be used (more generally, TOM function will receive the adjacency as input). If "signed", TOM will keep track of the sign of the adjacency between neighbors.
TOMDenom a character string specifying the TOM variant to be used. Recognized values are "min" giving the standard TOM described in Zhang and Horvath (2005), and "mean" in which the \(\min\) function in the denominator is replaced by \(\text{mean}\). The "mean" may produce better results but at this time should be considered experimental.
verbose integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
indent indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Details

The functions perform basically the same calculations of topological overlap. TOMdist turns the overlap (which is a measure of similarity) into a measure of dissimilarity by subtracting it from 1.

Basic checks on the adjacency matrix are performed and missing entries are replaced by zeros. If TOMType = "unsigned", entries of the adjacency matrix are required to lie between 0 and 1; for TOMType = "signed" they can be between -1 and 1. In both cases the resulting TOM entries, as well as the corresponding dissimilarities, lie between 0 and 1.

The underlying C code assumes that the diagonal of the adjacency matrix equals 1. If this is not the case, the diagonal of the input is set to 1 before the calculation begins.

Value

A matrix holding the topological overlap.

Author(s)

Peter Langfelder
References


See Also

TOMsimilarityFromExpr

TOMsimilarityFromExpr  Topological overlap matrix

Description

Calculation of the topological overlap matrix from given expression data.

Usage

TOMsimilarityFromExpr(datExpr, corType = "pearson", networkType = "unsigned", power = 6, TOMType = "signed", TOMDenom = "min", maxPOutliers = 1, quickCor = 0, pearsonFallback = "individual", cosineCorrelation = FALSE, nThreads = 0, verbose = 1, indent = 0)

Arguments

datExpr  expression data. A data frame in which columns are genes and rows ar samples. NAs are allowed, but not too many.
corType  character string specifying the correlation to be used. Allowed values are (unique abbreviations of) "pearson" and "bicor", corresponding to Pearson and bid-weight midcorrelation, respectively. Missing values are handled using the pairwise.complete.obs option.
networkType  network type. Allowed values are (unique abbreviations of) "unsigned", "signed", "signed hybrid". See adjacency.
power  soft-thresholding power for netwoek construction.
TOMType  one of "none", "unsigned", "signed". If "none", adjacency will be used for clustering. If "unsigned", the standard TOM will be used (more generally, TOM function will receive the adjacency as input). If "signed", TOM will keep track of the sign of correlations between neighbors.
a character string specifying the TOM variant to be used. Recognized values are "min" giving the standard TOM described in Zhang and Horvath (2005), and "mean" in which the min function in the denominator is replaced by mean. The "mean" may produce better results but at this time should be considered experimental.

maxPOutliers only used for corType="bicor". Specifies the maximum percentile of data that can be considered outliers on either side of the median separately. For each side of the median, if higher percentile than maxPOutliers is considered an outlier by the weight function based on 9*mad(x), the width of the weight function is increased such that the percentile of outliers on that side of the median equals maxPOutliers. Using maxPOutliers=1 will effectively disable all weight function broadening; using maxPOutliers=0 will give results that are quite similar (but not equal to) Pearson correlation.

quickCor real number between 0 and 1 that controls the handling of missing data in the calculation of correlations. See details.

pearsonFallback Specifies whether the bicor calculation, if used, should revert to Pearson when median absolute deviation (mad) is zero. Recognized values are (abbreviations of) "none", "individual", "all". If set to "none", zero mad will result in NA for the corresponding correlation. If set to "individual", Pearson calculation will be used only for columns that have zero mad. If set to "all", the presence of a single zero mad will cause the whole variable to be treated in Pearson correlation manner (as if the corresponding robust option was set to FALSE). Has no effect for Pearson correlation. See bicor.

cosineCorrelation logical: should the cosine version of the correlation calculation be used? The cosine calculation differs from the standard one in that it does not subtract the mean.

nThreads non-negative integer specifying the number of parallel threads to be used by certain parts of correlation calculations. This option only has an effect on systems on which a POSIX thread library is available (which currently includes Linux and Mac OSX, but excludes Windows). If zero, the number of online processors will be used if it can be determined dynamically, otherwise correlation calculations will use 2 threads.

verbose integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.

indent indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Value
A matrix holding the topological overlap.

Author(s)
Peter Langfelder

References
See Also

TOMsimilarity

transposeBigData  Transpose a big matrix or data frame

Description

This transpose command partitions a big matrix (or data frame) into blocks and applies the t() function to each block separately.

Usage

transposeBigData(x, blocksize = 20000)

Arguments

  x  a matrix or data frame

  blocksize  a positive integer larger than 1, which determines the block size. Default is 20k.

Details

Assume you have a very large matrix with say 500k columns. In this case, the standard transpose function of R t() can take a long time. Solution: Split the original matrix into sub-matrices by dividing the columns into blocks. Next apply t() to each sub-matrix. The same holds if the large matrix contains a large number of rows. The function transposeBigData automatically checks whether the large matrix contains more rows or more columns. If the number of columns is larger than or equal to the number of rows then the block wise splitting will be applied to columns otherwise to the rows.

Value

A matrix or data frame (depending on the input x) which is the transpose of x.

Note

This function can be considered a wrapper of t().

Author(s)

Steve Horvath, UCLA

References

Any linear algebra book will explain the transpose.

See Also

The standard function t.
**Examples**

```r
x=data.frame(matrix(1:10000,nrow=4,ncol=2500))
dimnames(x)[[2]]=paste("Y",1:2500,sep="")
x=transpose=transposeBigData(x)
x[1:4,1:4]
xTranspose[1:4,1:4]
```

**TrueTrait**  
*Estimate the true trait underlying a list of surrogate markers.*

**Description**

Assume an imprecisely measured trait \(y\) that is related to the true, unobserved trait \(y_{TRUE}\) as follows \(y_{TRUE} = y + \text{noise}\) where noise is assumed to have mean zero and a constant variance. Assume you have 1 or more surrogate markers for \(y_{TRUE}\) corresponding to the columns of \(\text{datx}\). The function implements several approaches for estimating \(y_{TRUE}\) based on the inputs \(y\) and/or \(\text{datx}\).

**Usage**

```r
TrueTrait(datX, y, datXtest=NULL, 
corfnc = "bicor", corOptions = "use = 'pairwise.complete.obs'", 
LeaveOneOut.CV=FALSE, skipMissingVariables=TRUE,
addLinearModel=FALSE)
```

**Arguments**

- **datX**
  - is a vector or data frame whose columns correspond to the surrogate markers (variables) for the true underlying trait. The number of rows of \(\text{datX}\) equals the number of observations, i.e. it should equal the length of \(y\).

- **y**
  - is a numeric vector which specifies the observed trait.

- **datXtest**
  - can be set as a matrix or data frame of a second, independent test data set. Its columns should correspond to those of \(\text{datX}\), i.e. the two data sets should have the same number of columns but the number or rows (test set observations) can be different.

- **corFnc**
  - Character string specifying the correlation function to be used in the calculations. Recommended values are the default Pearson correlation "cor" or biweight mid-correlation "bicor". Additional arguments to the correlation function can be specified using `corOptions`.

- **corOptions**
  - Character string giving additional arguments to the function specified in `corFnc`.

- **LeaveOneOut.CV**
  - logical. If TRUE then leave one out cross validation estimates will be calculated for \(y_{true1}\) and \(y_{true2}\) based on \(\text{datX}\).

- **skipMissingVariables**
  - logical. If TRUE then variables whose values are missing for a given observation will be skipped when estimating the true trait of that particular observation. Thus, the estimate of a particular observation are determined by all the variables whose values are non-missing.

- **addLinearModel**
  - logical. If TRUE then the function also estimates the true trait based on the predictions of the linear model `lm(y~., data=datX)`.
Details

This R function implements formulas described in Klemera and Doubal (2006). The assumptions underlying these formulas are described in Klemera et al. But briefly, the function provides several estimates of the true underlying trait under the following assumptions: 1) There is a true underlying trait that affects $y$ and a list of surrogate markers corresponding to the columns of datX. 2) There is a linear relationship between the true underlying trait and $y$ and the surrogate markers. 3) $y_{\text{TRUE}} = y + \text{Noise}$ where the Noise term has a mean of zero and a fixed variance. 4) Weighted least squares estimation is used to relate the surrogate markers to the underlying trait where the weights are proportional to $1/\text{ssq.j}$ where ssq.j is the noise variance of the j-th marker.

Specifically, output $y_{\text{true1}}$ corresponds to formula 31, $y_{\text{true2}}$ corresponds to formula 25, and $y_{\text{true3}}$ corresponds to formula 34.

Although the true underlying trait $y_{\text{TRUE}}$ is not known, one can estimate the standard deviation between the estimate $y_{\text{true2}}$ and $y_{\text{TRUE}}$ using formula 33. Similarly, one can estimate the SD for the estimate $y_{\text{true3}}$ using formula 42. These estimated SDs correspond to output components 2 and 3, respectively. These SDs are valuable since they provide a sense of how accurate the measure is.

To estimate the correlations between $y$ and the surrogate markers, one can specify different correlation measures. The default method is based on the Person correlation but one can also specify the biweight midcorrelation by choosing "bicor", see help(bicor) to learn more.

When the datX is comprised of observations measured in different strata (e.g. different batches or independent data sets) then one can obtain stratum specific estimates by specifying the strata using the argument Strata. In this case, the estimation focuses on one stratum at a time.

Value

A list with the following components.

- datEstimates is a data frame whose columns corresponds to estimates of the true underlying trait. The number of rows equals the number of observations, i.e. the length of $y$. The first column $y_{\text{true1}}$ is the average value of standardized columns of datX where standardization subtracts out the intercept term and divides by the slope of the linear regression model lm(marker~y). Since this estimate ignores the fact that the surrogate markers have different correlations with $y$, it is typically inferior to $y_{\text{true2}}$. The second column $y_{\text{true2}}$ equals the weighted average value of standardized columns of datX. The standardization is described in section 2.4 of Klemera et al. The weights are proportional to $r^2/(1+r^2)$ where $r$ denotes the correlation between the surrogate marker and $y$. Since this estimate does not include $y$ as additional surrogate marker, it may be slightly inferior to $y_{\text{true3}}$. Having said this, the difference between $y_{\text{true2}}$ and $y_{\text{true3}}$ is often negligible. An additional column called $y_{\text{1m}}$ is added if codeaddLinearModel=TRUE. In this case, $y_{\text{1m}}$ reports the linear model predictions. Finally, the column $y_{\text{true3}}$ is very similar to $y_{\text{true2}}$ but it includes $y$ as additional surrogate marker. It is expected to be the best estimate of the underlying true trait (see Klemera et al 2006).

- datEstimatetest is output only if a test data set has been specified in the argument datXtest. In this case, it contains a data frame with columns $y_{\text{true1}}$ and $y_{\text{true2}}$. The number of rows equals the number of test set observations, i.e the number of rows of datXtest. Since the value of $y$ is not known in case of a test data set, one cannot calculate $y_{\text{true3}}$. An additional column with linear model predictions $y_{\text{1m}}$ is added if codeaddLinearModel=TRUE.
is output only if the argument LeaveOneOut.CV has been set to TRUE. In this case, it contains a data frame with leave-one-out cross validation estimates of $y_{true1}$ and $y_{true2}$. The number of rows equals the length of $y$. Since the value of $y$ is not known in case of a test data set, one cannot calculate $y_{true3}$

SD.$y_{true2}$ is a scalar. This is an estimate of the standard deviation between the estimate $y_{true2}$ and the true (unobserved) $y_{TRUE}$. It corresponds to formula 33.

SD.$y_{true3}$ is a scalar. This is an estimate of the standard deviation between $y_{true3}$ and the true (unobserved) $y_{TRUE}$. It corresponds to formula 42.

datVariableInfo
is a data frame that reports information for each variable (column of datX) when it comes to the definition of $y_{true2}$. The rows correspond to the number of variables. Columns report the variable name, the center (intercept that is subtracted to scale each variable), the scale (i.e. the slope that is used in the denominator), and finally the weights used in the weighted sum of the scaled variables.

datEstimatesByStratum
a data frame that will only be output if Strata is different from NULL. In this case, it is has the same dimensions as datEstimates but the estimates were calculated separately for each level of Strata.

SD.$y_{true2}$ByStratum
a vector of length equal to the different levels of Strata. Each component reports the estimate of SD.$y_{true2}$ for observations in the stratum specified by unique(Strata).

datVariableInfoByStratum
a list whose components are matrices with variable information. Each list component reports the variable information in the stratum specified by unique(Strata).

Author(s)
Steve Horvath

References


Examples

```r
# observed trait
y=rnorm(1000,mean=50,sd=20)
# unobserved, true trait
yTRUE =y + rnorm(100,sd=10)
# now we simulate surrogate markers around the true trait
datX=simulateModule(yTRUE,nGenes=20, minCor=.4,maxCor=.9, geneMeans=rnorm(20,50,30) )
TrueTrait=RealTrait(datX=datX,y=y)
datTrue=RealTrait$datEstimates
par(mfrow=c(2,2))
for (i in 1:dim(datTrue)[2]) {
  meanAbsDev= mean(abs(yTRUE-datTrue[,i]))
  verboseScatterplot(datTrue[,i],yTRUE,xlab=names(datTrue)[i],
```
Calculation of unsigned adjacency

Calculation of the unsigned network adjacency from expression data. The restricted set of parameters for this function should allow a faster and less memory-hungry calculation.

Usage

unsignedAdjacency(
  datExpr,
  datExpr2 = NULL,
  power = 6,
  corFnc = "cor", corOptions = "use = 'p'"
)

Arguments

datExpr expression data. A data frame in which columns are genes and rows are samples. Missing values are ignored.
datExpr2 optional specification of a second set of expression data. See details.
power soft-thresholding power for network construction.
corFnc character string giving the correlation function to be used for the adjacency calculation. Recommended choices are "cor" and "bicor", but other functions can be used as well.
corOptions character string giving further options to be passed to the correlation function

Details

The correlation function will be called with arguments datExpr, datExpr2 plus any extra arguments given in corOptions. If datExpr2 is NULL, the standard correlation functions will calculate the correlation of columns in datExpr.

Value

Adjacency matrix of dimensions n×n, where n is the number of genes in datExpr.

Author(s)

Steve Horvath and Peter Langfelder
References


See Also

adjacency

**userListEnrichment**

Measure enrichment between inputted and user-defined lists

**Description**

This function measures list enrichment between inputted lists of genes and files containing user-defined lists of genes. Significant enrichment is measured using a hypergeometric test. A pre-made collection of brain-related lists can also be loaded. The function writes the significant enrichments to a file, but also returns all overlapping genes across all comparisons.

**Usage**

```r
userListEnrichment(
  geneR, labelR,
  fnIn = NULL, catNmIn = fnIn,
  nameOut = "enrichment.csv",
  useBrainLists = FALSE, useBloodAtlases = FALSE, omitCategories = "grey",
  outputCorrectedPvalues = TRUE, useStemCellLists = FALSE,
  outputGenes = FALSE,
  minGenesInCategory = 1,
  useBrainRegionMarkers = FALSE, useImmunePathwayLists = FALSE,
  usePalazzoloWang = FALSE)
```

**Arguments**

- **geneR**: A vector of gene (or other) identifiers. This vector should include ALL genes in your analysis (i.e., the genes corresponding to your labeled lists AND the remaining background reference genes).
- **labelR**: A vector of labels (for example, module assignments) corresponding to the geneR list. NOTE: For all background reference genes that have no corresponding label, use the label "background" (or any label included in the omitCategories parameter).
- **fnIn**: A vector of file names containing user-defined lists. These files must be in one of three specific formats (see details section). The default (NULL) may only be used if one of the "use____" parameters is TRUE.
- **catNmIn**: A vector of category names corresponding to each fnIn. This name will be appended to each overlap corresponding to that filename. The default sets the category names as the corresponding file names.
- **nameOut**: Name of the file where the output enrichment information will be written. (Note that this file includes only a subset of what is returned by the function.)
userListEnrichment

useBrainLists If TRUE, a pre-made set of brain-derived enrichment lists will be added to any user-defined lists for enrichment comparison. The default is FALSE. See references section for related references.

useBloodAtlases If TRUE, a pre-made set of blood-derived enrichment lists will be added to any user-defined lists for enrichment comparison. The default is FALSE. See references section for related references.

omitCategories Any labelR entries corresponding to these categories will be ignored. The default ("grey") will ignore unassigned genes in a standard WGCNA network.

outputCorrectedPvalues If TRUE (default) only p-values that are significant after correcting for multiple comparisons (using Bonferroni method) will be outputted to nameOut. Otherwise the uncorrected p-values will be outputted to the file. Note that both sets of p-values for all comparisons are reported in the returned "pValues" parameter.

useStemCellLists If TRUE, a pre-made set of stem cell (SC)-derived enrichment lists will be added to any user-defined lists for enrichment comparison. The default is FALSE. See references section for related references.

outputGenes If TRUE, will output a list of all genes in each returned category, as well as a count of the number of genes in each category. The default is FALSE.

minGenesInCategory Will omit all significant categories with fewer than minGenesInCategory genes (default is 1).

useBrainRegionMarkers If TRUE, a pre-made set of enrichment lists for human brain regions will be added to any user-defined lists for enrichment comparison. The default is FALSE. These lists are derived from data from the Allen Human Brain Atlas (http://human.brainmap.org/). See references section for more details.

useImmunePathwayLists If TRUE, a pre-made set of enrichment lists for immune system pathways will be added to any user-defined lists for enrichment comparison. The default is FALSE. These lists are derived from the lab of Daniel R Saloman. See references section for more details.

usePalazzoloWang If TRUE, a pre-made set of enrichment lists compiled by Mike Palazzolo and Jim Wang from CHDI will be added to any user-defined lists for enrichment comparison. The default is FALSE. See references section for more details.

Details

User-inputted files for fnIn can be in one of three formats:

1) Text files (must end in ".txt") with one list per file, where the first line is the list descriptor and the remaining lines are gene names corresponding to that list, with one gene per line. For example: Ribosome RPS4 RPS8 ...

2) Gene / category files (must be csv files), where the first line is the column headers corresponding to Genes and Lists, and the remaining lines correspond to the genes in each list, for any number of genes and lists. For example: Gene, Category RPS4, Ribosome RPS8, Ribosome ... NDUF1, Mitochondria NDUF3, Mitochondria ... MAPT, AlzheimersDisease PSEN1, AlzheimersDisease PSEN2, AlzheimersDisease ...

3) Module membership (kME) table in csv format. Currently, the module assignment is the only thing that is used, so as long as the Gene column is 2nd and the Module column is 3rd, it doesn’t...
matter what is in the other columns. For example, PSID, Gene, Module, <other columns> <psid>, RPS4, blue, <other columns> <psid>, NDUF1, red, <other columns> <psid>, RPS8, blue, <other columns> <psid>, NDUF3, red, <other columns> <psid>, MAPT, green, <other columns> ...

Value

pValues A data frame showing, for each comparison, the input category, user defined category, type, the number of overlapping genes and both the uncorrected and Bonferroni corrected p-values for every pair of list overlaps tested.

ovGenes A list of character vectors corresponding to the overlapping genes for every pair of list overlaps tested. Specific overlaps can be found by typing <variable-Name>$ovGenes$'<labelR> – <comparisonCategory>'. See example below.

sigOverlaps Identical information that is written to nameOut. A data frame ith columns giving the input category, user defined category, type, and P-values (corrected or uncorrected, depending on outputCorrectedPvalues) corresponding to all significant enrichments.

Author(s)

Jeremy Miller

References


If you have any suggestions for lists to add to this function, please e-mail Jeremy Miller at jerryinla@gmail.com

——— References for the pre-defined brain lists (useBrainLists=TRUE, in alphabetical order by category descriptor) are as follows:


CA1vsCA3 ==> Lists of genes enriched in CA1 and CA3 relative to other each and to other areas of the brain, from several studies: 1. Ginsberg => Ginsberg SD, Che S (2005) Expression profile analysis within the human hippocampus: comparison of CA1 and CA3 pyramidal neurons. J Comp Neurol 487:107-118. 2. Lein => Lein E, Zhao X, Gage F (2004) Defining a molecular atlas of


DiseaseGenes ==> Probable (C or better rating as of 16 Mar 2011) and possible (all genes in database as of ~2008) genetics-based disease genes from: http://www.alzforum.org/


JAXdiseaseGene ==> Genes where mutations in mouse and/or human are known to cause any disease. WARNING: this list represents an oversimplification of data! This list was created from the Jackson Laboratory: Bult CJ, Eppig JT, Kadin JA, Richardson JE, Blake JA; Mouse Genome Database Group (2008) The Mouse Genome Database (MGD): Mouse biology and model systems. Nucleic Acids Res 36 (database issue):D724-D728.


MO ==> Markers for many different things provided to my by Mike Oldham. These were originally from several sources: 1. 2+_26Mar08 => Genetics-based disease genes in two or more studies from http://www.alzforum.org/ (compiled by Mike Oldham). 2. Bachoo => Bachoo, R.M. et al. (2004) Molecular diversity of astrocytes with implications for neurological disorders. PNAS 101, 8384-8389. 3. Foster => Foster, LJ, de Hoog, CL, Zhang, Y, Zhang, Y, Xie, X, Mootha, VK,


References for the pre-defined blood atlases (useBloodAtlases=TRUE, in alphabetical order by category descriptor) are as follows:


References for the pre-defined stem cell (SC) lists (useStemCellLists=TRUE, in alphabetical order by category descriptor) are as follows:


References and more information for the pre-defined human brain region lists (useBrainRegionMarkers=TRUE):

HBA ==> Hawrylycz MJ, Lein ES, Guillozet-Bongaarts AL, Shen EH, Ng L, Miller JA, et al. (2012) An Anatomically Comprehensive Atlas of the Adult Human Brain Transcriptome. Nature (in press) Three categories of marker genes are presented: 1. globalMarker(top200) = top 200 global marker genes for 22 large brain structures. Genes are ranked based on fold change enrichment (expression in region vs. expression in rest of brain) and the ranks are averaged between brains 2001 and 2002 (human.brain-map.org). 2. localMarker(top200) = top 200 local marker genes for 90 large brain structures. Same as 1, except fold change is defined as expression in region vs. expression in larger region (format: <region>_IN_<largerRegion>). For example, enrichment in CA1 is relative to other subcompartments of the hippocampus. 3. localMarker(FC>2) = same as #2, but only local marker genes with fold change > 2 in both brains are included. Regions with <10 marker genes are omitted.

More information for the pre-defined immune pathways lists (useImmunPathwayLists=TRUE):

ImmunePathway ==> These lists were created by Brian Modena (a member of Daniel R Salomon’s lab at Scripps Research Institute), with input from Sunil M Kurian and Dr. Salomon, using Ingenuity, WikiPathways and literature search to assemble them. They reflect knowledge-based immune pathways and were in part informed by Dr. Salomon and colleague’s work in expression profiling of biopsies and peripheral blood but not in some highly organized process. These lists are not from any particular publication, but are culled to include only genes of reasonably high confidence.

References for the pre-defined lists from CHDI (usePalazzoloWang=TRUE, in alphabetical order by category descriptor) are as follows:


Kegg NCBI Biosystems ==> Several gene sets from the "Kegg" component of NCBI Biosystems: Geer LY et al 2010 (full citation above).

Palazzolo and Wang ==> These gene sets were compiled from a variety of sources by Mike Palazzolo and Jim Wang at CHDI.

Pathway Interaction Database NCBI Biosystems ==> Several gene sets from the "Pathway Interaction Database" component of NCBI Biosystems: Geer LY et al 2010 (full citation above).


Reactome NCBI Biosystems ==> Several gene sets from the "Reactome" component of NCBI Biosystems: Geer LY et al 2010 (full citation above).
Wiki Pathways NCBI Biosystems ==> Several gene sets from the "Wiki Pathways" component of
NCBI Biosystems: Geer LY et al 2010 (full citation above).
Yang ==> These gene sets were compiled from a variety of sources by Mike Palazzolo and Jim
Wang at CHDI.

Examples

# Example: first, read in some gene names and split them into categories
data(BrainLists);
listGenes = unique(as.character(BrainLists[,1]))
set.seed(100)
geneR = sort(sample(listGenes,2000))
categories = sort(rep(standardColors(10),200))
categories[sample(1:2000,200)] = "grey"
write(c("TESTLIST1",geneR[300:400], sep="\n"),"TESTLIST1.txt")
write(c("TESTLIST2",geneR[800:1000],sep="\n"),"TESTLIST2.txt")

# Now run the function!
testResults = userListEnrichment(geneR, labelR=categories,
   fnIn=c("TESTLIST1.txt","TESTLIST2.txt"),
   catNmIn=c("TEST1","TEST2"),
   nameOut = "testEnrichment.csv",useBrainLists=TRUE, omitCategories ="grey")

# To see a list of all significant enrichments, either open
# the file "testEnrichments.csv" in the current directory, or type:
testResults$sigOverlaps

# To see all of the overlapping genes between two categories
#(whether or not the p-value is significant), type
#testResults$ovGenes$<labelR> -- <comparisonCategory>' . For example:
testResults$ovGenes"black -- TESTLIST1__TEST1"
testResults$ovGenes"red -- salmon_M12_Ribosome_HumanMeta"

# More detailed overlap information is in the pValue output. For example:
head(testResults$pValue)

---

vectorizeMatrix **Turn a matrix into a vector of non-redundant components**

**Description**

A convenient function to turn a matrix into a vector of non-redundant components. If the matrix
is non-symmetric, returns a vector containing all entries of the matrix. If the matrix is symmetric,
only returns the upper triangle and optionally the diagonal.

**Usage**

```r
vectorizeMatrix(M, diag = FALSE)
```

**Arguments**

- **M** the matrix or data frame to be vectorized.
- **diag** logical: should the diagonal be included in the output?
Value

A vector containing the non-redundant entries of the input matrix.

Author(s)

Steve Horvath

---

vectorTOM

Topological overlap for a subset of the whole set of genes

Description

This function calculates topological overlap of a small set of vectors with respect to a whole data set.

Usage

vectorTOM(
  datExpr,  # a data frame containing the expression data of the whole set, with rows corresponding to samples and columns to genes.
  vect,     # a single vector or a matrix-like object containing vectors whose topological overlap is to be calculated.
  subtract1 = FALSE,  # logical: should calculation be corrected for self-correlation? Set this to TRUE if vect contains a subset of datExpr.
  blockSize = 2000,  # maximum block size for correlation calculations. Only important if vect contains a large number of columns.
  corFnc = "cor",  # character string giving the correlation function to be used for the adjacency calculation. Recommended choices are "cor" and "bicor", but other functions can be used as well.
  corOptions = "use = 'p'",  # character string giving further options to be passed to the correlation function.
  networkType = "unsigned",  # character string giving network type. Allowed values are (unique abbreviations of) "unsigned", "signed", "signed hybrid". See adjacency.
  power = 6,  # soft-thresholding power for network construction.
  verbose = 1,  # integer level of verbosity. Zero means silent, higher values make the output progressively more and more verbose.
  indent = 0)  # indentation for diagnostic messages. Zero means no indentation, each unit adds two spaces.

Arguments

datExpr

vect

subtract1

blockSize

corFnc

corOptions

networkType

power

verbose

indent
Details
Topological overlap can be viewed as the normalized count of shared neighbors encoded in an adjacency matrix. In this case, the adjacency matrix is calculated between the columns of `vect` and `datExpr` and the topological overlap of vectors in `vect` measures the number of shared neighbors in `datExpr` that vectors of `vect` share.

Value
A matrix of dimensions $n \times n$, where $n$ is the number of columns in `vect`.

Author(s)
Peter Langfelder

References

See Also
`tomsimilarity` for standard calculation of topological overlap.

Description
Produce a barplot with error bars, annotated by Kruskal-Wallis or ANOVA p-value.

Usage
```r
verbosebarplot(x, g, 
  main = "", xlab = NA, ylab = NA, 
  cex = 1, cex.axis = 1.5, cex.lab = 1.5, cex.main = 1.5, 
  color = "grey", numberStandardErrors = 1, 
  KruskalTest = TRUE, AnovaTest = FALSE, two.sided = TRUE, 
  addCellCounts=FALSE, horiz = FALSE, ...)
```

Arguments
- `x` numerical or binary vector of data whose group means are to be plotted
- `g` a factor or a an object coercible to a factor giving the groups whose means are to be calculated.
- `main` main title for the plot.
- `xlab` label for the x-axis.
- `ylab` label for the y-axis.
- `cex` character expansion factor for plot annotations.
verboseBarplot

cex.axis character expansion factor for axis annotations.
cex.lab character expansion factor for axis labels.
cex.main character expansion factor for the main title.
color a vector giving the colors of the bars in the barplot.
numberStandardErrors size of the error bars in terms of standard errors. See details.
KruskalTest logical: should Kruskal-Wallis test be performed? See details.
AnovaTest logical: should ANOVA be performed? See details.
two.sided logical: should the printed p-value be two-sided? See details.
addCellCounts logical: should counts be printed above each bar?
horiz logical: should the bars be drawn horizontally?
... other parameters to function barplot

Details

This function creates a barplot of a numeric variable (input x) across the levels of a grouping variable (input g). The height of the bars equals the mean value of x across the observations with a given level of g. By default, the barplot also shows plus/minus one standard error. If you want only plus one standard error (not minus) choose two.sided=TRUE. But the number of standard errors can be determined with the input numberStandardErrors. For example, if you want a 95% confidence interval around the mean, choose numberStandardErrors=2. If you don’t want any standard errors set numberStandardErrors=-1. The function also outputs the p-value of a Kruskal Wallis test (Fisher test for binary input data), which is a non-parametric multi group comparison test. Alternatively, one can use Analysis of Variance (Anova) to compute a p-value by setting AnovaTest=TRUE. Anova is a generalization of the Student t-test to multiple groups. In case of two groups, the Anova p-value equals the Student t-test p-value. Anova should only be used if x follows a normal distribution. Anova also assumes homoscedasticity (equal variances). The Kruskal Wallis test is often advantageous since it makes no distributional assumptions. Since the Kruskal Wallis test is based on the ranks of x, it is more robust with regard to outliers. All p-values are two-sided.

Value

None.

Author(s)

Steve Horvath

See Also

barplot

Examples

```r
group<sample(c(1,2),100,replace=TRUE)
height<rnorm(100,mean=group)
par(mfrow=c(2,2))
verboseBarplot(height,group, main="1 SE, Kruskal Test")
```
verboseBoxplot(height, group, numberStandardErrors=2, main="2 SE, Kruskal Test")

verboseBoxplot(height, group, numberStandardErrors=2, AnovaTest=TRUE, main="2 SE, Anova")

verboseBoxplot(height, group, numberStandardErrors=2, AnovaTest=TRUE, main="2 SE, Anova, only plus SE", two.sided=FALSE)

---

**verboseBoxplot**  
*Boxplot annotated by a Kruskal-Wallis p-value*

**Description**
Plot a boxplot annotated by the Kruskal-Wallis p-value. Uses the function `boxplot` for the actual drawing.

**Usage**
```r
verboseBoxplot(x, g, main = "", xlab = NA, ylab = NA,  
cex = 1, cex.axis = 1.5, cex.lab = 1.5, cex.main = 1.5,  
notch = TRUE, varwidth = TRUE, ...)
```

**Arguments**
- **x**: numerical vector of data whose group means are to be plotted
- **g**: a factor or an object coercible to a factor giving the groups that will go into each box.
- **main**: main title for the plot.
- **xlab**: label for the x-axis.
- **ylab**: label for the y-axis.
- **cex**: character expansion factor for plot annotations.
- **cex.axis**: character expansion factor for axis annotations.
- **cex.lab**: character expansion factor for axis labels.
- **cex.main**: character expansion factor for the main title.
- **notch**: logical: should the notches be drawn? See `boxplot` and `boxplot.stats` for details.
- **varwidth**: logical: if TRUE, the boxes are drawn with widths proportional to the square-roots of the number of observations in the groups.
- **...**: other arguments to the function `boxplot`. Of note is the argument `las` that specifies label orientation. Value `las=1` will result in horizontal labels (the default), while `las=2` will result in vertical labels, useful when the labels are long.

**Value**
Returns the value returned by the function `boxplot`. 
**verboseiplot**

**Author(s)**

Steve Horvath

**See Also**

*boxplot*

---

**Description**

Produce a scatterplot that shows density with color and is annotated by the correlation, MSE, and regression line.

**Usage**

```
verboseiplot(
  x, y,
  xlim = NA, ylim = NA,
  nBinsX = 150, nBinsY = 150,
  ztransf = function(x) {x}, gamma = 1,
  sample = NULL, corFnc = "cor", corOptions = "use = 'p'",
  main = "", xlab = NA, ylab = NA, cex = 1,
  cex.axis = 1.5, cex.lab = 1.5, cex.main = 1.5,
  abline = FALSE, abline.color = 1, abline.lty = 1,
  corLabel = corFnc, ...)
```

**Arguments**

- **x**
  numerical vector to be plotted along the x axis.
- **y**
  numerical vector to be plotted along the y axis.
- **xlim**
  define the range in x axis
- **ylim**
  define the range in y axis
- **nBinsX**
  number of bins along the x axis
- **nBinsY**
  number of bins along the y axis
- **ztransf**
  Function to transform the number of counts per pixel, which will be mapped by the function in colramp to well defined colors. The user has to make sure that the transformed density lies in the range \([0, zmax]\), where \(zmax\) is any positive number (\(\geq 2\)).
- **gamma**
  color correction power
- **sample**
  either a number of points to be sampled or a vector of indices input x and y for points to be plotted. Useful when the input vectors are large and plotting all points is not practical.
- **corFnc**
  character string giving the correlation function to annotate the plot.
- **corOptions**
  character string giving further options to the correlation function.
- **main**
  main title for the plot.
verboseScatterplot

xlab label for the x-axis.
ylab label for the y-axis.
cex character expansion factor for plot annotations.
cex.axis character expansion factor for axis annotations.
cex.lab character expansion factor for axis labels.
cex.main character expansion factor for the main title.
abline logical: should the linear regression fit line be plotted?
abline.color color specification for the fit line.
abline.lty line type for the fit line.
corlabel character string to be used as the label for the correlation value printed in the main title.
...
other arguments to the function plot.

Details
Irrespective of the specified correlation function, the MSE is always calculated based on the residuals of a linear model.

Value
If sample above is given, the indices of the plotted points are returned invisibly.

Note
This function is based on verboseScatterplot (Steve Horvath and Peter Langfelder), iplot (Andreas Ruckstuhl, Rene Locher) and greenWhiteRed(Peter Langfelder)

Author(s)
Chaochao Cai, Steve Horvath

See Also
image for more parameters

---

verboseScatterplot Scatterplot annotated by regression line and p-value

Description
Produce a scatterplot annotated by the correlation, p-value, and regression line.
verboseScatterplot

Usage

verboseScatterplot(x, y,
    sample = NULL,
    corFnc = "cor", corOptions = "use = 'p'",
    main = "", xlab = NA, ylab = NA,
    cex = 1, cex.axis = 1.5, cex.lab = 1.5, cex.main = 1.5,
    abline = FALSE, abline.color = 1, abline.lty = 1,
    corLabel = corFnc,
    displayAsZero = 1e-5,
    col = 1, bg = 0,
    lmfnc = lm,
    ...
)

Arguments

x        numerical vector to be plotted along the x axis.
y        numerical vector to be plotted along the y axis.
sample   determines whether x and y should be sampled for plotting, useful to keep the
          plot manageable when x and y are large vectors. The default NULL value implies
          no sampling. A single numeric value will be interpreted as the number of points
          to sample randomly. If a vector is given, it will be interpreted as the indices of
          the entries in x and y that should be plotted. In either case, the correlation and p
          value will be determined from the full vectors x and y.
corFnc   character string giving the correlation function to annotate the plot.
corOptions character string giving further options to the correlation function.
main     main title for the plot.
xlab     label for the x-axis.
ylab     label for the y-axis.
cex      character expansion factor for plot annotations.
cex.axis character expansion factor for axis annotations.
cex.lab  character expansion factor for axis labels.
cex.main character expansion factor for the main title.
abline   logical: should the linear regression fit line be plotted?
abline.color color specification for the fit line.
abline.lty line type for the fit line.
corLabel character string to be used as the label for the correlation value printed in the
          main title.
displayAsZero Correlations whose absolute value is smaller than this number will be displayed
          as zero. This can result in a more intuitive display (for example, cor=0 instead
          of cor=2.6e-17).
col        color of the plotted symbols. Recycled as necessary.
bg         fill color of the plotted symbols (used for certain symbols). Recycled as neces-
          sary.
lmfnc      linear model fit function. Used to calculate the linear model fit line if 'abline'
          is TRUE. For example, robust linear models are implemented in the function rlm.
...       other arguments to the function plot.
Details

Irrespective of the specified correlation function, the p-value is always calculated for pearson correlation.

Value

If sample above is given, the indices of the plotted points are returned invisibly.

Author(s)

Steve Horvath and Peter Langfelder

See Also

plot.default for standard scatterplots

---

votingLinearPredictor  Voting linear predictor

Description

Predictor based on univariate regression on all or selected given features that pools all predictions using weights derived from the univariate linear models.

Usage

votingLinearPredictor(
  x, y, xtest = NULL,
  classify = FALSE,
  CVfold = 0,
  randomSeed = 12345,
  assocFnc = "cor", assocOptions = "use = 'p'",
  featureWeightPowers = NULL, priorWeights = NULL,
  weighByPrediction = 0,
  nFeatures.hi = NULL, nFeatures.lo = NULL,
  dropUnusedDimensions = TRUE,
  verbose = 2, indent = 0)

Arguments

x  Training features (predictive variables). Each column corresponds to a feature and each row to an observation.

y  The response variable. Can be a single vector or a matrix with arbitrary many columns. Number of rows (observations) must equal to the number of rows (observations) in x.

xtest  Optional test set data. A matrix of the same number of columns (i.e., features) as x. If test set data are not given, only the prediction on training data will be returned.

classify  Should the response be treated as a categorical variable? Classification really only works with two classes. (The function will run for multiclass problems as well, but the results will be sub-optimal.)
### votingLinearPredictor

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVfold</td>
<td>Optional specification of cross-validation fold. If 0 (the default), no cross-validation is performed.</td>
</tr>
<tr>
<td>randomSeed</td>
<td>Random seed, used for observation selection for cross-validation. If NULL, the random generator is not reset.</td>
</tr>
<tr>
<td>assocFnc</td>
<td>Function to measure association. Usually a measure of correlation, for example Pearson correlation or \texttt{bicor}.</td>
</tr>
<tr>
<td>assocOptions</td>
<td>Character string specifying the options to be passed to the association function.</td>
</tr>
<tr>
<td>featureWeightPowers</td>
<td>Powers to which to raise the result of \texttt{assocFnc} to obtain weights. Can be a single number or a vector of arbitrary length; the returned value will contain one prediction per power.</td>
</tr>
<tr>
<td>priorWeights</td>
<td>Prior weights for the features. If given, must be either (1) a vector of the same length as the number of features (columns in \texttt{x}); (2) a matrix of dimensions length(featureWeightPowers)x(number of features); or (3) array of dimensions (number of response variables)xlength(featureWeightPowers)x(number of features).</td>
</tr>
<tr>
<td>weighByPrediction</td>
<td>(Optional) power to downweigh features that are not well predicted between training and test sets.</td>
</tr>
<tr>
<td>nFeatures.hi</td>
<td>Optional restriction of the number of features to use. If given, this many features with the highest association and lowest association (if nFeatures.lo is not given) will be used for prediction.</td>
</tr>
<tr>
<td>nFeatures.lo</td>
<td>Optional restriction of the number of lowest (i.e., most negatively) associated features to use. Only used if nFeatures.hi is also non-NULL.</td>
</tr>
<tr>
<td>dropUnusedDimensions</td>
<td>Logical: should unused dimensions be dropped from the result?</td>
</tr>
<tr>
<td>verbose</td>
<td>Integer controlling how verbose the diagnostic messages should be. Zero means silent.</td>
</tr>
<tr>
<td>indent</td>
<td>Indentation for the diagnostic messages. Zero means no indentation, each unit adds two spaces.</td>
</tr>
</tbody>
</table>

### Details

The predictor calculates the association of each (selected) feature with the response and uses the association to calculate the weight of the feature as \( \text{sign(association)} \times (\text{association})^{\text{featureWeightPower}} \). Optionally, this weight is multiplied by \texttt{priorWeights}. Further, a feature prediction weight can be used to downweigh features that are not well predicted by other features (see below).

For classification, the (continuous) result of the above calculation is turned into ordinal values essentially by rounding.

If features exhibit non-trivial correlations among themselves (such as, for example, in gene expression data), one can attempt to down-weigh features that do not exhibit the same correlation in the test set. This is done by using essentially the same predictor to predict \_features\_ from all other features in the test data (using the training data to train the feature predictor). Because test features are known, the prediction accuracy can be evaluated. If a feature is predicted badly (meaning the error in the test set is much larger than the error in the cross-validation prediction in training data), it may mean that its quality in the training or test data is low (for example, due to excessive noise or outliers). Such features can be downweighed using the argument \texttt{weighByPrediction}. The extra factor is \( \min(1, (\text{root mean square prediction error in test set})/(\text{root mean square cross-validation prediction error in the training data})^{\text{weighByPrediction}}) \), that is it is never bigger than 1.
Value

A list with the following components:

- **predicted**: The back-substitution prediction on the training data. Normally an array of dimensions (number of observations) x (number of response variables) x length(featureWeightPowers), but unused are dropped unless dropUnusedDimensions = FALSE.
- **weightBase**: Absolute value of the associations of each feature with each response.
- **variableImportance**: The weight of each feature in the prediction (including the sign).
- **predictedTest**: If input xtest is non-NULL, the predicted test response, in format analogous to predicted above.
- **CVpredicted**: If input CVfold is non-zero, cross-validation prediction on the training data.

Note

It makes little practical sense to supply neither xtest nor CVfold since the prediction accuracy on training data will be highly biased.

Author(s)

Peter Langfelder

See Also

- bicor for robust correlation that can be used as an association measure
Index

*Topic cluster
  consensusProjectiveKMeans, 87
  moduleNumber, 169
  projectiveKMeans, 237

*Topic color
  blueWhiteRed, 55
  greenBlackRed, 128
  greenWhiteRed, 129
  labels2colors, 148
  redWhiteGreen, 255
  rgcolors.func, 260
  standardColors, 281

*Topic datasets
  BloodLists, 54
  BrainLists, 56
  BrainRegionMarkers, 56
  ImmunePathwayLists, 131
  PWLLists, 242
  SCsLists, 262

*Topic graphics
  verboseIplot, 313

*Topic hplot
  addErrorBars, 13
  addGrid, 13
  addGuidelines, 14
  labeledBarplot, 138
  labeledHeatmap, 140
  plotClusterTreeSamples, 217
  plotColorUnderTree, 219
  plotCor, 221
  plotDendroAndColors, 221
  plot EigengeneNetworks, 224
  plotMat, 226
  plotMEpairs, 227
  plotModuleSignificance, 228
  plotNetworkHeatmap, 229
  verboseScatterplot, 314

*Topic misc
  accuracyMeasures, 11
  addTraitToMEs, 15
  adjacency, 16
  adjacency.polyReg, 17
  adjacency.splineReg, 19
  AFcorMI, 21
  alignExpr, 22
  allocateJobs, 22
  allowWGCNAThreads, 23
  automaticNetworkScreening, 24
  automaticNetworkScreeningGS, 26
  bicovWeights, 31
  blockSize, 32
  blockwiseConsensusModules, 33
  blockwiseIndividualTOMs, 43
  blockwiseModules, 47
  branchEigengeneDissim, 57
  branchSplit, 58
  branchSplit.dissim, 59
  branchSplitFromStabilityLabels, 59
  checkAdjMat, 61
  checkSets, 61
  chooseOneHubInEachModule, 62
  chooseTopHubInEachModule, 64
  clusterCoef, 65
  coClustering, 65
  coClustering.permutationTest, 66
  collapseRows, 68
  collapseRowsUsingKME, 73
  colQuantileC, 74
  conformityBasedNetworkConcepts, 75
  conformityDecomposition, 77
  consensusDissTOMAndTree, 79
  consensusKME, 81
  consensusMEDissimilarity, 85
  consensusOrderMEs, 86
  consensusTOM, 89
  cor, 96
  corPredictionSuccess, 100
  corPvalueFisher, 101
  corPvalueStudent, 102
  correlationPreservation, 102
  coxRegressionResiduals, 103
  cuttreeStatic, 105
  cuttreeStaticColor, 106
  displayColors, 106
  dynamicMergeCut, 107
  exportNetworkToCytoscape, 111
exportNetworkToVisANT, 112
fixDataStructure, 113
formatLabels, 114
fundamentalNetworkConcepts, 115
GEnrichmentAnalysis, 117
goodGenes, 120
goodGenesMS, 122
goodSamples, 123
goodSamplesGenes, 124
goodSamplesGenesMS, 126
goodSamplesMS, 127
GTOMdist, 130
hubGeneSignificance, 131
Inline display of progress, 132
intramodularConnectivity, 133
isMultiData, 135
keepCommonProbes, 136
kMEcomparisonScatterplot, 137
labeledHeatmap.multiPage, 144
list2multiData, 150
lowerTri2matrix, 150
matchLabels, 151
matrixToNetwork, 153
mergeCloseModules, 154
metaAnalysis, 157
metaZFunction, 162
moduleColor.getMEprefix, 163
moduleEigengenes, 163
moduleMergeUsingKME, 167
modulePreservation, 170
mtd.apply, 174
mtd.mapply, 176
mtd.rbindingSelf, 178
mtd.setAttr, 179
mtd.setColNames, 180
mtd.simplify, 181
mtd.subset, 182
multiData, 183
multiData.eigengeneSignificance, 184
multiSetMEs, 185
multiUnion, 189
mutualInfoAdjacency, 189
nearestCentroidPredictor, 192
nearestNeighborConnectivity, 196
nearestNeighborConnectivityMS, 197
networkConcepts, 198
networkScreening, 201
networkScreeningGS, 203
normalizeLabels, 204
nPresent, 204
nSets, 205
numbers2colors, 205
orderBranchesUsingHubGenes, 206
orderMEs, 209
overlapTable, 210
overlapTableUsingKME, 211
pickHardThreshold, 213
pickSoftThreshold, 215
plotClusterTreeSamples, 217
plotModuleSignificance, 228
populationMeansInAdmixture, 230
pquantile, 232
prepComma, 234
prependZeros, 234
preservationNetworkConnectivity, 235
proportionsInAdmixture, 239
propVarExplained, 241
qvalue, 242
qvalue.restricted, 244
randIndex, 244
rankPvalue, 245
recutBlockwiseTrees, 247
recutConsensusTrees, 251
relativeCorPredictionSuccess, 256
removeGreyME, 257
removePrincipalComponents, 257
returnGeneSetsAsList, 258
scaleFreeIndex, 260
scaleFreePlot, 261
setCorrelationPreservation, 263
shortenStrings, 264
sigmoidAdjacencyFunction, 265
signedKME, 266
signumAdjacencyFunction, 267
simulateDatExpr, 267
simulateDatExpr5Modules, 270
simulateEigengeneNetwork, 272
simulateModule, 273
simulateMultiExpr, 274
simulateSmallLayer, 277
sizeGrWindow, 278
softConnectivity, 279
spaste, 280
standardColors, 281
standardScreeningBinaryTrait, 281
standardScreeningCensoredTime, 284
standardScreeningNumericTrait, 286
stderr, 287
stratifiedBarplot, 288
subsetTOM, 289
swapTwoBranches, 291
TOMplot, 292
INDEX

TOMsimilarity, 294
TOMsimilarityFromExpr, 295
transposeBigData, 297
TrueTrait, 298
unsignedAdjacency, 301
userListEnrichment, 302
vectorizeMatrix, 308
vectorTOM, 309
verboseBarplot, 310
verboseBoxplot, 312
votingLinearPredictor, 316

*Topic models
  empiricalBayesLM, 108

*Topic package
  WGCNA-package, 5

*Topic plot
  labelPoints, 147

*Topic regression
  empiricalBayesLM, 108

*Topic robust
  bicc, 27

*Topic stats
  bicorAndPvalue, 29
  corAndPvalue, 99

*Topic utilities
  collectGarbage, 74

abline, 218, 223
accuracyMeasures, 11
addErrorBars, 13
addGrid, 13
addGuidelines, 14
addTraitToMets, 15
adjacency.polyReg, 17
adjacency.splineReg, 19
AFcorMI, 21
alignExpr, 22
allocateJobs, 22, 146, 147
allowWGCNAThreads, 23
automaticNetworkScreening, 24
automaticNetworkScreeningGS, 26, 203

barplot, 139, 229, 289, 311
bicor, 23, 27, 29, 30, 32, 36, 45, 50, 92, 158, 171, 286, 296, 317, 318
bicorAndPvalue, 29, 85
bicovWeights, 31, 109, 111
blockSize, 32
blockwiseConsensusModules, 33, 46, 47, 60, 80, 89, 251, 254, 255
blockwiseIndividualTOMs, 36, 43, 92, 96
blockwiseModules, 25, 47, 60, 174, 247, 250, 251
BloodLists, 54
blueWhiteRed, 55, 129, 130
boxplot, 229, 312, 313
boxplot.stats, 312
BrainLists, 56
BrainRegionMarkers, 56
branchEigeneigene dissim, 57
branchSplit, 58
branchSplit.dissim, 59
branchSplitFromStabilityLabels, 39, 51, 59

checkAdjMat, 61
checkSets, 15, 35, 44, 61, 86–88, 91, 103, 114, 122, 126, 127, 136, 155, 158, 170, 184, 186, 205, 209, 224, 235, 252, 257
checkSimilarity (checkAdjMat), 61
chooseOneHubInEachModule, 62
chooseTopHubInEachModule, 64
clusterCoef, 65
coclustering, 65, 68
coclustering.permutationTest, 66, 66
collapseRows, 68, 73
collapseRowsUsingKME, 73
collectGarbage, 74
colors, 143, 260
colQuantileC, 74
conformityBasedNetworkConcepts, 75, 79, 116, 201
conformityDecomposition, 77
consensusDissToMandTree, 79
consensusKME, 81
consensusMEDissimilarity, 85
consensusOrderMets, 86, 210
consensusProjectiveKMeans, 40, 46, 87, 94
consensusTOM, 38, 89
cor, 23, 28, 96, 96, 97–100, 158, 171, 174, 194, 221, 227
corr.test, 30, 99, 100
corr (cor), 96
corrAndPvalue, 85, 99
corFast (cor), 96
corPredictionSuccess, 100, 256
corPvalueFisher, 101
corPvalueStudent, 102
correlationPreservation, 102
coxRegressionResiduals, 103
INDEX

kappa, 241
keepCommonProbes, 136
kMEcomparisonScatterplot, 137

labeledBarplot, 138, 226
labeledHeatmap, 140, 146, 147, 225, 226
labeledHeatmap.multiPage, 144
labelPoints, 147
labels2colors, 148, 206
lapply, 174
layout, 293
list2multiData, 150, 183
lm, 241, 258
load, 38, 51, 93
lowerTri2matrix, 150

mapply, 176
matchLabels, 151, 211
matrixToNetwork, 153
mean, 233
median, 233
metaAnalysis, 157
metaZfunction, 162
moduleColor.getMEprefix, 163
moduleMergeUsingKME, 167
moduleNumber, 169
modulePreservation, 66, 68, 170
mtd. apply, 174, 183
mtd. applyToSubset, 176, 183
mtd. applyToSubset (mtd. apply), 174
mtd. branchEigengeneDissim
   (branchEigengeneDissim), 57
mtd. colnames (mtd. setColnames), 180
mtd. mapply, 176, 176, 183
mtd. rbindSelf, 178
mtd. setAttr, 179
mtd. setColnames, 180
mtd. simplify, 181
mtd. subset, 182
multiData, 176, 178–181, 183, 183
multiData.eigengeneSignificance, 184
multiData2list, 181, 183
multiData2list (list2multiData), 150
multiIntersect (multiUnion), 189
multiSetMEs, 41, 87, 103, 185, 210, 254, 264
multiUnion, 189
mutualInfoAdjacency, 21, 189

cutree, 105, 106, 169
cutreeDynamic, 25, 27, 38, 39, 43, 51, 52, 54, 220, 249, 251–253, 255
cutreeStatic, 105, 105, 106
cutreeStaticColor, 106
disableWGCNAThreads
   (allowWGCNAThreads), 23
displayColors, 106
dist, 16, 42, 134, 218
dynamicMergeCut, 107
empiricalBayesLM, 108
enableWGCNAThreads, 172, 216
enableWGCNAThreads (allowWGCNAThreads), 23
exportNetworkToCytoscape, 111
exportNetworkToVisANT, 112, 112
fisher.test, 211
fixDataStructure, 113
formatLabels, 114, 265
fundamentalNetworkConcepts, 77, 115, 201
glm, 19, 20
GOenrichmentAnalysis, 117
goodGenes, 120, 123, 125, 127, 128
goodGenesMS, 122, 127, 128
goodSamples, 121, 123, 123, 124, 125, 127, 128
goodSamplesGenes, 54, 121, 123, 124, 127, 128, 248
goodSamplesGenesMS, 43, 46, 95, 123, 126, 128, 172, 174, 252
goodSamplesMS, 123, 127, 127
greenBlackRed, 128
greenWhiteRed, 129
gregexpr, 264, 265
GTOMdist, 130
hclust, 15, 43, 54, 105, 106, 169, 218, 219, 222, 226
heat.colors, 141, 225
heatmap, 142, 143, 293
hubGeneSignificance, 25–27, 131
image, 221, 227, 260, 314
immunePathwayLists, 131
initProgInd (Inline display of progress), 132
Inline display of progress, 132
intersect, 189
intraModularConnectivity, 133
isMultiData, 135, 177
nearestCentroidPredictor, 192
nearestNeighborConnectivity, 196, 198
nearestNeighborConnectivityMS, 197
networkConcepts, 77, 116, 198
networkScreening, 25, 27, 201, 203
networkScreeningGS, 26, 203
normalize.quantiles, 40, 94
normalize.labels, 169, 204
nPresent, 204
ns, 19, 20
nSets, 205
numbers2colors, 55, 129, 130, 205
orderBranchesUsingHubGenes, 206
orderMEs, 86, 87, 209
overlapTable, 152, 210, 212
overlapTableUsingKME, 211
pairs, 227, 228
par, 14, 15, 141, 147, 218, 220, 221, 224, 227, 260
paste, 280
pdf, 226
pickHardThreshold, 213
pickSoftThreshold, 215
plot, 228, 315
plot.default, 148, 316
plot.hclust, 218, 223
plotClusterTreeSamples, 217
plotColorUnderTree, 219, 223
plotCor, 221, 227, 260
plotDendroAndColors, 218, 220, 221
plot EigeneNetworks, 224, 264
plotMat, 221, 226, 260
plotMEpairs, 227
plotModuleSignificance, 228
plotNetworkHeatmap, 229
plotOrderedColors (plotColorUnderTree), 219
pmax, 233
pmean (pquantile), 232
pmedian (pquantile), 232
pmin, 233
poly, 19
populationMeansInAdmixture, 230
postscript, 226
pquantile, 232
prepComma, 234
prependZeros, 234
preservationNetworkConnectivity, 235
projectiveKMeans, 52, 89, 237
proportionsInAdmixture, 239
propVarExplained, 241
PWLLists, 242
quantile, 75, 233
qvalue, 242, 244, 247
qvalue.restricted, 244
randIndex, 244
rank, 245, 247
rankPvalue, 82, 84, 158, 159, 245
rbind, 179
rcorr.cens, 161, 283
recutBlockwiseTrees, 247
recutConsensusTrees, 251
redWhiteGreen, 225, 255
reflectBranch (swapTwoBranches), 291
relativeCorPredictionSuccess, 101, 256
removeGreyME, 257
removePrincipalComponents, 257
returnGeneSetsAsList, 258
rgb, 221, 227, 260
rgcolors.func, 221, 227, 260
rlm, 315
rowQuantileC (colQuantileC), 74
scaleFreeFitIndex, 260
scaleFreePlot, 261
SCSLists, 262
selectBranch (swapTwoBranches), 291
set.seed, 194
setCorrelationPreservation, 263
shortenStrings, 264
sigmoidAdjacencyFunction, 265
signedKME, 85, 266
signumAdjacencyFunction, 215, 267
simulateDataExp, 267, 271, 274, 276–278
simulateDataExpr5Modules, 270, 270, 274, 277
simulateEigengeneNetwork, 270, 272, 274, 277
simulateModule, 270, 271, 273, 277, 278
simulateMultiExpr, 270, 274, 274
simulateSmallLayer, 277
sizeGrWindow, 278
softConnectivity, 197, 198, 217, 262, 279
spaste, 280
standardColors, 105–107, 152, 281
standardScreeningBinaryTrait, 157, 160, 162, 281, 287
standardScreeningCensoredTime, 284, 287
standardScreeningNumericTrait, 157, 160, 162, 286
stderr, 287
stratifiedBarplot, 288
strsplit, 115
subsetTOM, 289
svd, 166, 258
swapTwoBranches, 291

t, 297
t.test, 158

table, 11
text, 141, 147, 148
TOMdist (TOMsimilarity), 294
TOMplot, 292
TOMsimilarity, 43, 54, 230, 290, 294, 297, 310
TOMsimilarityFromExpr, 295, 295
transposeBigData, 297
TrueTrait, 298

union, 189
unsignedAdjacency, 301
updateProgInd (Inline display of progress), 132
userListEnrichment, 54, 56, 57, 132, 242, 262, 302

vectorizeMatrix, 308
vectorTOM, 309
verboseBarplot, 289, 310
verboseBoxplot, 312
verboseIplot, 313
verboseScatterplot, 314
votingLinearPredictor, 195, 316

WGCNA (WGCNA-package), 5
WGCNA-package, 5
WGCNAThreads (allowWGCNAThreads), 23