Package ‘mlDNA’

February 20, 2015

Type Package
Title Machine Learning-based Differential Network Analysis of Transcriptome Data
Version 1.1
Date 2013-11-18
Author Chuang Ma and Xiangfeng Wang
Maintainer Chuang Ma <chuangma2006@gmail.com>
Depends R (>= 2.15.1), snowfall (>= 1.84-4), igraph (>= 0.6.6), rsgcc (>= 1.0.6), e1071 (>= 1.6-1), randomForest(>= 4.6-7)
Imports pROC (>= 1.5.4), ROCR(>= 1.0-5)
Suggests bigmemory, GeneSelector, gplots, nnet, cairoDevice, fBasics
Description Functions necessary to perform the machine learning-based differential network analysis of transcriptome data.
URL http://www.cmbb.arizona.edu/mlDNA
LazyLoad yes
License GPL (>= 2)
NeedsCompilation no
Repository CRAN
Date/Publication 2013-11-19 07:37:01

R topics documented:

mlDNA-package ....................................................... 2
AverageRankScore .................................................. 3
classifier ............................................................. 4
ConditionSpecificGenes ............................................. 6
ConvergenceDegree .................................................. 8
cross_validation .................................................... 9
data ................................................................. 11
exp2net ............................................................... 12
Description

This package provides functions for performing the machine learning (ML)-based differential network analysis of transcriptome data. It can be used to:

1) perform machine learning-based gene filtering with positive sample-only learning algorithm for identifying a set of candidate genes with four different classes of expression characteristics, including the absolute expression values, the within-condition expression variations, the between-condition expression changes, and the coefficient of variation;

2) construct gene co-expression networks from gene expression data with seven optional methods, including five correlation and two non-correlation measures;

3) perform a comprehensive network comparison with more than thirty network-based characteristics including degree, closeness, eccentricity, eigenvector, and PageRank;

4) identify biologically important genes with different ML algorithms by combining network-based characteristics generated from differential network analysis;

5) estimate the convergence degree between different experimental conditions;

6) quantify the activity of pathways;

7) detect condition specifically expressed genes.

The tutorial of the mlDNA package can be found at: http://www.cmbb.arizona.edu/mlDNA.

Details

Package: mlDNA
Type: Package
Version: 1.1
Date: 2013-11-18
License: GPL(>=2)
AverageRankScore

Author(s)

Chuang Ma, Xiangfeng Wang
Maintainer: Chuang Ma <chuangma2006@gmail.com>

References


| AverageRankScore | Average-based rank score |

Description

This function calculates the activities of pathways in the whole genome with the average-based rank scoring algorithm (Yang, et al., 2011; Ma and Wang, 2013). This rank-based statistics is robust for directly comparing the activities of pathways with different gene numbers under different experimental conditions, since it produces a normalized value with the consideration of gene number in the analyzed pathways and whole genomes.

Usage

```
AverageRankScore( featureMat, selGenes )
```

Arguments

- **featureMat** a numeric matrix recording the expression levels or changes of all genes in the genome at given conditions.
- **selGenes** a character vector recording a set of genes in the analyzed pathway.

Value

- **value** a numeric vector recording the activities of interested genes (selGenes) at different conditions.

Author(s)

Chuang Ma, Xiangfeng Wang

References


Examples

```r
## Not run:

## generate expression feature matrix
sampleVec1 <- c(1, 2, 3, 4, 5, 6)
sampleVec2 <- c(1, 2, 3, 4, 5, 6)
featureMat <- expFeatureMatrix(expMat1 = ControlExpMat, sampleVec1 = sampleVec1,
                                expMat2 = SaltExpMat, sampleVec2 = sampleVec2,
                                logTransformed = TRUE, base = 2,
                                features = "foldchange")

## for an interested set of genes, the average-based rank score can be calculated:
genes <- rownames(featureMat)[1:100]
res <- AverageRankScore( featureMat = featureMat, selGenes = genes )
```

## End(Not run)

---

**classifier**

*Building machine learning-based classification models*

**Description**

This function builds classification models with different machine learning algorithms including random forest (randomForest), support vector machine (svm), and neural network (nnet).

**Usage**

```r
classifier( method = c("randomForest", "svm", "nnet"),
            featureMat, positivesamples, negativesamples,
            tunecontrol = tune.control(sampling = "cross", cross = 5), ...)
```

**Arguments**

- `method`: a character string specifying machine learning method used to construct prediction models. Currently the user-optional values are "randomForest", "svm" or "nnet".
- `featureMat`: a numeric matrix recording feature values for each sample.
- `positivesamples`: a character vector recording positive samples used to train classification model.
- `negativesamples`: a character vector recording negative samples used to train classification model.
- `tunecontrol`: control parameters for the tune function. The value is assigned by the function "tune.control" in e1071 package.
- `...`: Further parameters passed to train classification model.
classifier

Details

For the random forest algorithm, the important parameters are mtry (number of features randomly selected for building the decision tree. Default: \( \text{sqrt}(\text{ncol(featureMat)}) \)) and ntree (number of trees to be built. Default: 500). More information about the parameters related to random forest can be found in R package RandomForest.

For the svm algorithm, the default kernel function is the "radial" (radial basis function; RBF). Other optional kernels are the "linear", the "polynomial", and the "sigmoid" functions. The range of degree (for the kernel type of polynomial), gamma (for radial), coef0 (for polynomial and sigmoid) and cost (the cost parameters for all kernels) should be designated. Please refer the description of "svm" function in R package e1071 for more information about the parameters of svm.

For the neural network, the important parameters are size (number of units in the hidden layer), decay (parameter for weight decay. Default 0), trace (switch for tracing optimization. Default TRUE.). More information about the parameters of neural network is described in nnet and e1071 packages.

Value

An object of class tune (see tune function in e1071 package) is returned, including the components:

- bestNparameters
- bestNperformance
- bestNmodel

Author(s)

Chuang Ma, Xiangfeng Wang

Examples

```r
## Not run:

## Generate expression feature matrix
sampleVec1 <- c(1, 2, 3, 4, 5, 6)
sampleVec2 <- c(1, 2, 3, 4, 5, 6)
featureMat <- expFeatureMatrix( expMat1 = ControlExpMat, sampleVec1 = sampleVec1,
                              expMat2 = SaltExpMat, sampleVec2 = sampleVec2,
                              logTransformed = TRUE, base = 2,
                              features = c("zscore", "foldchange", "cv", "expression"))

## Positive samples
positiveSamples <- as.character(sampleData$KnownSaltGenes)
## Unlabeled samples
unlabelSamples <- setdiff( rownames(featureMat), positiveSamples )
idx <- sample(length(unlabelSamples))
## Randomly selecting 1000 unlabeled samples as negative samples
```
negativeSamples <- unlabelSamples[idx[1:1000]]

### for random forest, and using five-fold cross validation for obtaining optimal parameters
clf <- classifier( method = "randomForest",
                  featureMat = featureMat,
                  positiveSamples = positiveSamples,
                  negativeSamples = negativeSamples,
                  tunecontrol = tune.control(sampling = "cross", cross = 5),
                  ntree = 100 ) #build 100 trees for the forest

### svm and using five-fold cross validation for obtaining optimal parameters
clf <- classifier( method = "svm",
                  featureMat = featureMat,
                  positiveSamples = positiveSamples,
                  negativeSamples = negativeSamples,
                  tunecontrol = tune.control(sampling = "cross", cross = 5),
                  kernel = "radial",
                  probability = TRUE,
                  ranges = list(gamma = 2^{(-2:2)},
                                cost = 2^{(-4:4)}) ) #for radial kernel and the parameter space.

### neural network, using one split for training/validation set
clf <- classifier( method = "nnet",
                  featureMat = featureMat,
                  positiveSamples = positiveSamples,
                  negativeSamples = negativeSamples,
                  tunecontrol = tune.control(sampling = "fix"),
                  trace = TRUE, size = 10 ) #for nnet parameters.

### End(Not run)

---

**ConditionSpecificGenes**

*Condition specifically expressed genes*

**Description**

This function detects condition specifically expressed genes which are highly expressed in only one condition (e.g., stress), but are not or lowly expressed in the other conditions.

**Usage**

ConditionSpecificGenes( expmat, logtransformed = TRUE, base = 2, threshold = 0.75 )

**Arguments**

expmat: a numeric matrix recording the expression level of genes at different conditions. The column labels are samples names. For two samples from the same condition C, their names should be assigned as C.1 and C.2, respectively.
ConditionSpecificGenes

logtransformed logical indicating whether the gene expression values in expmat have been log-transformed.
base a numeric value indicating the base of logarithm.
threshold a numeric value giving the threshold of condition specificity score. The condition specificity score is 1, if the gene is only expressed at one condition. Otherwise, the condition specificity score will be smaller than 1.

Value

A list with following components:

- **CSGenes** a character vector containing the condition specially expressed genes.
- **CSScoreMat** a data matrix recording the condition-specificity scores for all conditions, the maximal condition-specificity score and its corresponding condition.
- **uniqueCS** a data matrix containing the condition information obtained from the colnames(expmat).

Author(s)

Chuang Ma, Xiangfeng Wang

References


Examples

```r
## Not run:

##show colnames of SaltExpMat
colnames(SaltExpMat)

##as the dot is used to indicate the replications in one condition, we
##have to change the colname of genes at the 0.5 time point.
colnames(SaltExpMat)[1] <- "Salt_0.5h"

##get condition specifically expressed genes
res <- ConditionSpecificGenes( expmat = SaltExpMat, logtransformed = TRUE,
                               base = 2, threshold = 0.75 )

##number of condition specifically expressed genes
length(res$CSGenes)

## End(Not run)
```
ConvergenceDegree

Calculating coverage degree

Description

This function calculates the convergence degree between two sets of elements, and has been used
to estimate the degree of convergence between genes responsive to different stresses in Arabidopsis
(Ma and Wang, 2013)

Usage

ConvergenceDegree( vec1, vec2 )

Arguments

vec1 a vector recording a set of elements for one condition.
vec2 a vector recording a set of elements for the other condition.

Value

value an numeric value

Author(s)

Chuang Ma, Xiangfeng Wang

References

stress-responsive transcriptomes in Arabidopsis thaliana. 2013(Submitted).

Examples

vec1 <- c(1, 2, 4, 10)
vec2 <- c(1, 10, 15:20)
ConvergenceDegree( vec1, vec2 )
Cross validation

**Description**

The ML-based classification model is trained and tested with N-fold cross validation method.

**Usage**

```r
cross_validation(seed = 1, method = c("randomForest", "svm", "nnet"),
  featureMat, positives, negatives, cross = 5, cpus = 1, ...)
```

**Arguments**

- `seed`: an integer number specifying a random seed for randomly partitioning dataset.
- `method`: a character string specifying machine learning method. Possible values are "randomForest", "nnet" or "svm"
- `featureMat`: a numeric feature matrix.
- `positives`: a character vector recording positive samples
- `negatives`: a character vector recording negative samples.
- `cross`: number of fold for cross validation.
- `cpus`: an integer number specifying the number of cpus to be used for parallel computing.
- `...`: Further parameters used to cross validation. Same with the parameters used in the classifier function.

**Details**

In machine learning, the cross validation method has been widely used to evaluate the performance of ML-based classification models (classifiers).

For N-fold cross validation, positive and negative samples are randomly partitioned into N groups with approximately equal amount of samples, and each group is successively used for testing the performance of the ML-based classifier trained with the other N-1 groups of positive and negative samples.

For each round of cross validation, the prediction accuracy of the ML-based classifier was assessed using the receiver operating characteristic (ROC) curve analysis. The ROC curve is a two-dimensional plot of the false positive rate (FPR, x-axis) against the true positive rate (TPR, y-axis) at all possible thresholds. The value of area under the ROC curve (AUC) was used to quantitatively score the prediction accuracy of the ML-based classifier. The AUC value is ranged from 0 to 1.0, with higher AUC value indicates a better prediction accuracy of the ML-based classifier.

After N groups have been successively used as the testing set, the N sets of (FPR, TPR) pairs were imported into R package ROCR to visualize the ROC curves. The mean value of N AUCs was then computed as the overall performance of the ML-based classification model.
**Value**

A list recording results from each fold cross validation including the components:

- `positives.train` - positive samples used to train prediction model.
- `negatives.train` - negative samples used to train prediction model.
- `positives.test` - positive samples used to test prediction model.
- `negatives.test` - negative samples used to test prediction model.
- `ml` - machine learning method.
- `classifier` - prediction model constructed with the best parameters obtained from training dataset.
- `positives.train.score` - scores of positive samples in training dataset predicted by classifier.
- `positives.test.score` - scores of positive samples in testing dataset predicted by classifier.
- `negatives.test.score` - scores of negative samples in testing dataset predicted by classifier.
- `train.AUC` - AUC value of the ML-based classifier on training dataset.
- `test.AUC` - AUC value of the ML-based classifier on testing dataset.

**Author(s)**

Chuang Ma, Xiangfeng Wang

**Examples**

```r
## Not run:

### generate expression feature matrix
sampleVec1 <- c(1, 2, 3, 4, 5, 6)
sampleVec2 <- c(1, 2, 3, 4, 5, 6)
featureMat <- expFeatureMatrix( expMat1 = ControlExpMat, sampleVec1 = sampleVec1,
                              expMat2 = SaltExpMat, sampleVec2 = sampleVec2,
                              logTransformed = TRUE, base = 2,
                              features = c("zscore", "foldchange", "cv", "expression"))

### positive samples
positiveSamples <- as.character(sampleData$KnownSaltGenes)
### unlabeled samples
unlabeledSamples <- setdiff( rownames(featureMat), positiveSamples )
idx <- sample(length(unlabeledSamples))
### randomly selecting a set of unlabeled samples as negative samples
```
```
negativeSamples <- unlabelSamples[idx[1:length(positiveSamples)]]

## five-fold cross validation
seed <- randomSeed() # generate a random seed
cvRes <- cross_validation(seed = seed, method = "randomForest",
                          featureMat = featureMat,
                          positives = positiveSamples,
                          negatives = negativeSamples,
                          cross = 5, cpus = 1,
                          ntree = 100 ) ##parameters for random forest algorithm

## get AUC values for five rounds of cross validation
aucVec <- rep(0, 5)
for( i in 1:5 )
  aucVec[i] = cvRes[[i]]$test.AUC

## average AUC values as the final performance of the ML-based classifier
mean(aucVec)
```

## End(Not run)

---

### data  

**Example data for mIDNA**

**Description**

salt stress expression data and 50 known salt-related genes

**Usage**

data(mIDNA)

**Examples**

```r
## load saltData dataset
data(mIDNA)

## known salt stress-related genes as positive samples
positiveSamples <- as.character(sampleData$KnownSaltGenes)

## gene expression data under control condition
ControlExpMat <- as.matrix(sampleData$ControlExpMat)

## gene expression data under salt stress condition
SaltExpMat <- as.matrix(sampleData$StressExpMat)
```
Inferring transcriptional networks from gene expression data

Description

This function infers transcriptional networks from gene expression data with different statistical methods, including five correlation measures (i.e., the Gini correlation coefficient [GCC], the Pearson’s product moment correlation coefficient [PCC], the Kendall tau rank correlation coefficient [KCC], the Spearman’s rank correlation coefficient [SCC] and the Tukey’s biweight correlation coefficient [BiWt]) and two non-correlation measures (mutual information [MI] and the maximal information-based nonparametric exploration [MINE]).

Usage

exp2net(expmat, method = c("GCC", "PCC", "SCC", "KCC", "BiWt", "MI", "MINE"),
        pvalue = 0.01, cpus = 1, expDescribe = "Control",
        connListFlag = TRUE, distmatFlag = TRUE, saveType = "bigmatrix",
        netResFileDic, ... )

Arguments

expmat a numeric matrix recording gene expression data.
method a character string specifying the statistical method will be used to calculating the associations between any pairs of genes.
pvalue a numeric value denoting the significance level of the association will be used to filter unsignficant interactions (i.e., edge) in the network.
cpus an integer specifying the number of cpus will be used for parallel computing.
expDescribe an character string describing the expmat.
connListFlag a logical value indicating whether the connected genes for each gene will be recorded.
distmatFlag a logical value indicating whether the distance matrix will be calculated.
saveType an character string indicating the format ("matrix", "bigmatrix") of matrix.
netResFileDic a character string specifying the file directory will be used to store network-related results.
... Future parameters for calculating distances between two gene sets. For instance, v = c(g1, g2, ..., gn), to = c(g1, g3, ..., gm).

Value

A list with 12 components:

expmat the input gene expression data.
method the method used to calculate the association between two genes.
pvalue the significance level used to detect edges in the network.
exp2net

expdescribe the characterized string for gene expression data.

netResFileDic the file directory for storing network-related result.

adjmat adjacency matrix recording the association between any pairs of genes in the big.matrix format.

adjmat_backingfile the root name for the file for the cache of adjmat. Default: expDescribe_method_adjmat_bfile

adjmatDescriptorfile the file to be used for the description of the adjmat. Default: expDescribe_method_adjmat_dfile

threshold the correlation score at the significance level of pvalue.

graph an igraph object for the constructed network in the edgelist format. This object is save in the file: expDescribe_graph.

connectivityList a list; For each component, it is a list with three component: "pos" (connected genes with positive correlations), "neg" (connected genes with negative correlations), "all" (all connected genes)

distmatrix a numeric matrix; the shorest-path distance between any pair of genes in the network.

Note

[1] The GCC, PCC, SCC and KCC calculate the adjacency matrix more quickly than BiWt, MI and MINE.

[2] The threshold is determined with the permutation method by generating the background distribution of correlations by permuting the expression levels of nrow(expmat) genes from the original expression dataset(expmat) (Carter et al., 2004).

[3] The adjacency and distance matrix can be stored in big.matrix format which can be used to greatly save the memory space. However, this big.matrix optional can only be used on Linux. More information about the big.matrix can be found in the R package bigmemory.

[4] The functions for graph analysis (i.e., getting information of nodes and edges) can be found in the R package igraph.


Author(s)

Chuang Ma, Xiangfeng Wang.

References

Examples

## Not run:

```r
#suppose the network-related results are stored at:
netResFileDic = "/home/wanglab/mlDNA/network/

#build transcriptional network from the first 1000 genes,
#here a higher number of cpus is suggested.
res <- exp2net(
  expmat = ControlExpMat[1:1000],
  method = "GCC",
  pvalue = 0.01, cpus = 2,
  expDescribe = "Control", connListFlag = TRUE,
  distMatFlag = TRUE,
  saveType = "bigmatrix", netResFileDic = netResFileDic,
  v = rownames(ControlExpMat)[1:10],
  to = rownames(ControlExpMat)[100:120]
)
```

## End(Not run)

---

**expFeatureMatrix**  
*Expression-based feature matrix*

**Description**

This function generates expression-based features for each gene with the consideration of z-scores, fold changes and actual expression values.

**Usage**

```r
expFeatureMatrix(expMat1, sampleVec1, expMat2, sampleVec2, 
    logTransformed = TRUE, base = 2,
    features = c("zscore", "foldchange", "cv", "expression"))
```

**Arguments**

- **expMat1**: a numeric matrix recording gene expression data from condition 1. Each row represents the expression values of one gene, each column represents the expression values of all genes from one biological experiment.
- **sampleVec1**: a numeric vector representing biological replication (or different time points) and technical replication for condition 1. For instance, `c(1,1,2,2,3,3)` denotes three biological replications, and two technical replications for each biological sample.
- **expMat2**: a numeric matrix recording gene expression data from condition 2.
- **sampleVec2**: a numeric vector representing biological experiments and technical replications for condition 2. Definition is similar as `sampleVec1`. 
geneRanker

logTransformed

logic value, TRUE indicates that the expression data in expMat1 and expMat2 have been log-transformed.

base

base of log transformation.

features

A character factor specifying different features will be used to generate feature matrix.

Value

value

a numeric matrix with express-based features for each gene.

Note

The expression-based features including:

1. expression variance across all samples calculated with z-scores of genes under two conditions;
2. fold change of gene expression under two conditions;
3. expression values of genes in two conditions used for retaining the information from actual expression values.

Author(s)

Chuang Ma, Xiangfeng Wang

Examples

```R
## Not run:

## generated expression features
sampleVec1 <- c(1, 2, 3, 4, 5, 6)
sampleVec2 <- c(1, 2, 3, 4, 5, 6)
featureMat <- expFeatureMatrix(expMat1 = ControlExpMat, sampleVec1 = sampleVec1, expMat2 = SaltExpMat, sampleVec2 = sampleVec2, logTransformed = TRUE, base = 2, features = c("zscore", "foldchange", "cv", "expression"))

## End(Not run)
```
geneRanker

Usage

geneRanker(expmat1, expmat2, genes,
    rankers = c("ttest", "SAM", "Limma"), verbose = FALSE)

Arguments

expmat1 a numeric matrix, gene expression matrix under condition 1.
expmat2 a numeric matrix, gene expression matrix under condition 2.
genres a character vector, genes to be analyzed.
rankers a character vector, differential expression methods
verbose logical. TRUE: intermediate results will be printed to the screen.

Details

Different methods can be selected. More information can be referred in GeneSelector package (Boulesteix and Slawski, 2009). To run this function, please first install geneSelector package with the commands: source("http://bioconductor.org/biocLite.R"); biocLite("GeneSelector")

Value

A list containing different components:
ranker a numeric matrix containing gene rank, statistic and p-value from the ranker.
dot a numeric matrix containing gene rank, statistic and p-value from the rankers.
pvalMat a numeric matrix containing p-values from different rankers.

Author(s)

Chuang Ma, Xiangfeng Wang.

References


Examples

## Not run:

```r
# differential expression analysis
res <- geneRanker(expmat1 = ControlExpMat, expmat2 = SaltExpMat,
    genes = rownames(ControlExpMat)[1:100],
    rankers = c("ttest", "SAM", "Limma"),
    verbose = FALSE )

# the p-value for differential method
res$pvalMat[1:10,]
```
interactionRemoval

## Description

The interaction-removal method is a gene selection approach based on the removal of intersected interactions in two networks, and has been applied to analyze the rewiring of genetic interaction maps for identifying yeast genes responsive to DNA damage (Bandyopadhyay, et al., 2010; Ideker and Krogan, 2012).

## Usage

interactionRemoval( adjmat1, adjmat2, threshold1, threshold2 )

## Arguments

- **adjmat1**: adjacency matrix for condition1. The adjacency matrix can be generated using adjacencymatrix function in rsgcc package.
- **adjmat2**: adjacency matrix for condition2.
- **threshold1**: threshold used to mask low correlations in adjmat1.
- **threshold2**: threshold used to mask low correlations in adjmat2.

## Value

- **value**: a matrix recording the specific degree of genes in the network constructed for condition 1.

## Author(s)

Chuang Ma, Xiangfeng Wang

## References

netFeatureMatrix  Generating network feature matrix

Description

This function generates a feature matrix containing 33 network characteristics by differential network analysis.

Usage

```r
netFeatureMatrix( net1, net2, nodes = NULL, knodes = NULL,
                 cpus = 1, verbose = TRUE, netResFileDic,
                 features = c( "expDistance", "ASC", "corDistance",
                               "AllConnectivity", "PosConnectivity",
                               "NegConnectivity", "closeness",
                               "eccentricity", "eigenvector",
                               "page.rank", "dis2knodes",
                               "closeness2knodes", "eccentricity2knodes") )
```

Arguments

- **net1**: exp2net output for condition 1.
- **net2**: exp2net output for condition 2.
- **nodes**: a character vector, a set of genes will be analyzed.
- **knodes**: a character vector, a set of interested genes (e.g., known stress-related genes).
- **cpus**: an integrator value, the number of cpus will be used for parallel computing.
- **verbose**: logical value indicate whether the progress information will be output.
- **netResFileDic**: file directory where the network-related results will be output.
- **features**: a character vector specifying the network characteristics will be analyzed.

Value

A numeric matrix, feature matrix generated from network comparison analysis.

Note

1) More information about these network properties has been described in (Ma and Wang, 2013).
2) To run `netFeatureMatrix`, the parameters "v" and "to" in "exp2net" function should be the whole genes in expmat.

Author(s)

Chuang Ma, Xiangfeng Wang.
optimalScore

References


Examples

```r
## Not run:

##suppose the network-related results are stored at:
netResFileDic = "/home/wanglab/mlDNA/network/

##only consider a subset of genes here
genes <- unique( c( rownames(ControlExpMat)[1:1000], positiveSamples[1:100] )
res_ControlSub <- exp2net( expmat = ControlExpMat[genes,,] , method = "GCC", pvalue = 0.01, cpus = 6, expDescribe = "Control_Sub", connListFlag = TRUE, distMatFlag = TRUE, saveType = "bigmatrix", netResFileDic = netResFileDic, v = genes, to = genes )

res_StressSub <- exp2net( expmat = SaltExpMat[genes,,] , method = "GCC", pvalue = 0.01, cpus = 6, expDescribe = "Stress_Sub", connListFlag = TRUE, distMatFlag = TRUE, saveType = "bigmatrix", netResFileDic = netResFileDic, v = genes, to = genes )

##generate network feature matrix
nodes <- genes
knodes <- intersect( genes, positiveSamples )

## End(Not run)
```
The optimal prediction score is detected with the F-score measure for the machine learning-based classification model which can balance true positives (TPs), false positives (FPs), true negatives (TNs) and false negatives (FNs).

**Usage**

```r
optimalScore( positiveSampleScores, negativeSampleScores, beta = 2, plot = TRUE )
```

**Arguments**

- `positiveSampleScores`: a numeric vector, the prediction scores of positive samples.
- `negativeSampleScores`: a numeric vector, the prediction scores of negative samples.
- `beta`: a positive numeric value, $\beta > 1$ indicating that a higher preference is given to recall than precision; $\beta = 1$ denoting the recall and precision are weighted equally. $\beta < 1$ representing that a higher preference is given to precision than recall.
- `plot`: logical, TRUE indicates the distribution of F-score at different threshold of prediction score is plotted. Otherwise not plotted.

**Value**

A list containing two components:

- `statMat`: a numeric matrix recording the statistic results (i.e., prediction score, TP, FP, TN, FN, Recall, TNR[TN/(TN+FP)], Precision, F-score) at the threshold of all possible prediction scores.
- `optimalScore`: a numeric value, the identified optimal prediction score.

**Author(s)**

Chuang Ma, Xiangfeng Wang

**Examples**

```r
## Not run:

##generate expression feature matrix
sampleVec1 <- c(1, 2, 3, 4, 5, 6)
sampleVec2 <- c(1, 2, 3, 4, 5, 6)
featureMat <- expFeatureMatrix( expMat1 = ControlExpMat, sampleVec1 = sampleVec1,
                               expMat2 = SaltExpMat, sampleVec2 = sampleVec2,
                               logTransformed = TRUE, base = 2,
                               features = c("zscore", "foldchange", "cv", "expression"))

##positive samples
positiveSamples <- as.character(sampleData$KnownSaltGenes)
```
## Description

This function plots ROC curves for estimating the performance of machine learning-based classification model in cross validation experiments.

## Usage

```r
plotROC(cvRes)
```

## Arguments

- **cvRes**: results from the "cross_validation" function.

## Value

- A ROC plot
Author(s)
Chuang Ma, Xiangfeng Wang

Examples

```r
## Not run:

##generate expression feature matrix
sampleVec1 <- c(1, 2, 3, 4, 5, 6)
sampleVec2 <- c(1, 2, 3, 4, 5, 6)
featureMat <- expFeatureMatrix(
  expMat1 = ControlExpMat, sampleVec1 = sampleVec1,
  expMat2 = SaltExpMat, sampleVec2 = sampleVec2,
  logTransformed = TRUE, base = 2,
  features = c("zscore", "foldchange", "cv", "expression"))

##positive samples
positiveSamples <- as.character(sampleData$KnownSaltGenes)
##unlabeled samples
unlabelSamples <- setdiff(rownames(featureMat), positiveSamples)
idx <- sample(length(unlabelSamples))
##randomly selecting a set of unlabeled samples as negative samples
negativeSamples <- unlabelSamples[idx[1: length(positiveSamples)]]

##five-fold cross validation
seed <- randomSeed() #generate a random seed
cvRes <- cross_validation(seed = seed, method = "randomForest",
  featureMat = featureMat,
  positives = positiveSamples, negatives = negativeSamples,
  cross = 5, cpus = 1,
  ntree = 100 ) #parameters for random forest algorithm

#plot ROC curve
plotROC(cvRes)

## End(Not run)
```

### predictor

<table>
<thead>
<tr>
<th>Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score each sample with the machine learning-based classification prediction model already trained with training dataset.</td>
</tr>
</tbody>
</table>

#### Usage

```
predictor(method = c("randomForest", "svm", "nnet"), classifier, featureMat)
```
predictor

Arguments

- **method**: character string specifying the machine learning algorithm used to build classification model.
- **classifier**: trained prediction model obtained from the classifier function.
- **featureMat**: a numeric matrix; feature matrix containing samples to be scored and their feature values.

Value

- **value**: A numeric vector containing the prediction score of each sample.

Author(s)

Chuang Ma, Xiangfeng Wang

Examples

```r
## Not run:

## generate expression feature matrix
sampleVec1 <- c(1, 2, 3, 4, 5, 6)
sampleVec2 <- c(1, 2, 3, 4, 5, 6)
featureMat <- expFeatureMatrix(
    expMat1 = ControlExpMat, sampleVec1 = sampleVec1,
    expMat2 = SaltExpMat, sampleVec2 = sampleVec2,
    logTransformed = TRUE, base = 2,
    features = c("zscore", "foldchange", "cv", "expression"))

## positive samples
positiveSamples <- as.character(sampleData$KnownSaltGenes)
## unlabeled samples
unlabeledSamples <- setdiff(rownames(featureMat), positiveSamples)
idx <- sample(length(unlabeledSamples))
## randomly selecting a set of unlabeled samples as negative samples
negativeSamples <- unlabeledSamples[idx[1:length(positiveSamples)]]

## for random forest, and using five-fold cross validation
## for obtaining optimal parameters
c1 <- classifier(method = "randomForest", featureMat = featureMat,
    positiveSamples = positiveSamples, negativeSamples = negativeSamples,
    tunecontrol = tune.control(sampling = "cross", cross = 5),
    ntree = 100 ) # build 100 trees for the forest

## constructed prediction model
predModel <- c1$best.model

## perform prediction
predResult <- predictor(method = "randomForest",
    classifier = predModel,
```
PSOL_InitialNegativeSelection

Initial negative set selection for building machine learning-based classification model

Description

This function selects an initial negative set with the machine learning (ML)-based positive-only sample learning (PSOL) algorithm. The PSOL algorithm has been previously applied to predict genomic loci encoding functional non-coding RNAs (Wang, et al. 2006). We have employed this algorithm to identify stress-related candidate genes in Arabidopsis based on the stress microarray datasets (Ma and Wang, 2013).

Usage

PSOL_InitialNegativeSelection(featureMatrix, positives, unlabels,
    negNum = length(positives), cpus = 1, PSOLResDic )

Arguments

- **featureMatrix**: a numeric matrix recording the features for all sample.
- **positives**: a character vector recording positive samples
- **unlabels**: a character vector recording unlabeled samples.
- **negNum**: an integer number specifying the size of negative samples will be selected.
- **cpus**: an integer number specifying the number of cpus will be used for parallel computing.
- **PSOLResDic**: a character string specifying the file directionry storing PSOL results.

Value

A list containing three components:

- **positives**: a character vector including the input positive samples.
- **negatives**: a character vector recording the selected negative samples.
- **unlabels**: a character vector recording the unlabeled samples.

Author(s)

Chuang Ma and Xiangfeng Wang.
References


Examples

## Not run:

```r
##generate expression feature matrix
sampleVec1 <- c(1, 2, 3, 4, 5, 6)
sampleVec2 <- c(1, 2, 3, 4, 5, 6)
featureMat <- expFeatureMatrix(
  expMat1 = ControlExpMat, sampleVec1 = sampleVec1,
  expMat2 = SaltExpMat, sampleVec2 = sampleVec2,
  logTransformed = TRUE, base = 2,
  features = c("zscore", "foldchange",
             "cv","expression")
)

##positive samples
positiveSamples <- as.character(sampleData$KnownSaltGenes)
##unlabeled samples
unlabelSamples <- setdiff( rownames(featureMat), positiveSamples )

##selecting an intial set of negative samples
##for building ML-based classification model
##suppose the PSOL results will be stored in:
PSOLResDic <- "/home/wanglab/mlDNA/PSOL/
res <- PSOL_InitialNegativeSelection(featureMatrix = featureMat,
  positives = positiveSamples,
  unlabels = unlabelSamples,
  negNum = length(positiveSamples),
  cpus = 6, PSOLResDic = PSOLResDic )

##initial negative samples extracted from unlabelled samples with PSOL algorithm
negatives <- res$negatives

## End(Not run)
```
Description

This function expands the negative sample set using PSOL algorithm.

Usage

PSOL_NegativeExpansion(featureMat, positives, negatives, unlabels, cpus = 1,
                        iterator = 50, cross = 5, TPR = 0.98, method = "randomForest",
                        plot = TRUE, trace = TRUE, PSOLResDic, ...)

Arguments

- **featureMat**: a feature matrix recording the feature values for all samples.
- **positives**: a character string recording the positive samples.
- **negatives**: a character string recording the negative samples.
- **unlabels**: a character string recording the unlabeled samples.
- **cpus**: an integer value, cpu number
- **iterator**: an integer value, iteration times.
- **cross**: an integer value, cross-times cross validation.
- **TPR**: a numeric value ranged from 0 to 1.0, used to select the prediction score cutoff.
- **method**: a character string, machine learning method
- **plot**: a logic value specifying whether the score distribution of positive and unlabeled samples will be plotted.
- **trace**: logic. TRUE: the intermediate results will be saved as "RData" format.
- **PSOLResDic**: a character string, PSOL Result directory
- **...**: Further parameters used in PSOL_ExpandSelection. see the further parameters in function classifier.

Value

The PSOL-related results are output in the "resultDic" directory.

Author(s)

Chuang Ma, Xiangfeng Wang.

Examples

```r
## Not run:

## generate expression feature matrix
sampleVec1 <- c(1, 2, 3, 4, 5, 6)
sampleVec2 <- c(1, 2, 3, 4, 5, 6)
featureMat <- expFeatureMatrix(
  expMat1 = ControlExpMat,
  sampleVec1 = sampleVec1,
  expMat2 = SaltExpMat,
)```
sampleVec2 = sampleVec2,
logTransformed = TRUE,
base = 2,
features = c("zscore",
"foldchange", "cv",
"expression"))

###positive samples
positiveSamples <- as.character(sampleData$knownSaltGenes)
###unlabeled samples
unlabelSamples <- setdiff(rownames(featureMat), positiveSamples)

###selecting an initial set of negative samples
###for building ML-based classification model
###suppose the PSOL results will be stored in:
PSOLResDic <- "/home/wanglab/mlDNA/PSOL/
res <- PSOL_InitialNegativeSelection(featureMatrix = featureMat,
            positives = positiveSamples,
            unlabels = unlabelSamples,
            negNum = length(positiveSamples),
            cpus = 6, PSOLResDic = PSOLResDic)

###initial negative samples extracted from unlabelled samples with PSOL algorithm
negatives <- res$negatives

###negative sample expansion
PSOL_NegativeExpansion(featureMat = featureMat, positives = positiveSamples,
            negatives = res$negatives, unlabels = res$unlabels,
            cpus = 2, iterator = 50, cross = 5, TPR = 0.98,
            method = "randomForest", plot = TRUE, trace = TRUE,
            PSOLResDic = PSOLResDic,
            ntrees = 200) # parameters for ML-based classifier

## End(Not run)

---

**PSOL_ResultExtraction  PSOL result extraction**

**Description**

This function extracts the PSOL result.

**Usage**

```
PSOL_ResultExtraction(PSOLResDic, iterations = c(1:4))
```
randomSeed

Arguments

- **PSOLResDic**  
  PSOL result file directory.
- **iterations**  
  a numeric vector specifying the results at given iteration times will be extracted.

Value

A list with length(iterations) components. For each component, there is a list with four components:

- **AUC**  
  The AUC from cross validation experiments testing classifiers with positives and negatives.
- **positives**  
  positive samples
- **negatives**  
  negative samples
- **unlabels**  
  unlabeled samples

Author(s)

Chuang Ma, Xiangfeng Wang

Examples

```r
## Not run:

##after run PSOL_InitialNegativeSelection and PSOL_NegativeExpansion
##extract the PSOL results at specified iteration times with the command:
PSOLResDic <- "/home/wanglab/mlDNA/PSOL/"
PSOLRes <- PSOL_ResultExtraction( PSOLResDic = PSOLResDic, iterations = c(1:4) )

## End(Not run)
```

---

randomSeed  

*Random seed generator*

Description

This function generates a random seed based on the system time.

Usage

randomSeed()

Value

- **value**  
  an integer value
randomSeed

Author(s)

Chuang Ma, Xiangfeng Wang

Examples

randomSeed()
Index

*Topic **PSOL**
  PSOL_InitialNegativeSelection, 24
  PSOL_NegativeExpansion, 25
  PSOL_ResultExtraction, 27
*Topic **datasets**
data, 11
*Topic **differential expression**
  expFeatureMatrix, 14
*Topic **expression**
geneRanker, 15
*Topic **machine learning**
classifier, 4
cross_validation, 9
optimalScore, 19
plotROC, 21
predictor, 22
PSOL_InitialNegativeSelection, 24
PSOL_NegativeExpansion, 25
PSOL_ResultExtraction, 27
*Topic **network**
  exp2net, 12
interactionRemoval, 17
netFeatureMatrix, 18
*Topic **package**
  mLDNA (mLDNA-package), 2
  mLDNA-package, 2
  netFeatureMatrix, 18
  optimalScore, 19
plotROC, 21
predictor, 22
PSOL_InitialNegativeSelection, 24
PSOL_NegativeExpansion, 25
PSOL_ResultExtraction, 27
randomSeed, 28
saltData (data), 11

AverageRankScore, 3

classifier, 4
ConditionSpecificGenes, 6
ConvergenceDegree, 8
cross_validation, 9
data, 11
exp2net, 12
expFeatureMatrix, 14
geneRanker, 15
interactionRemoval, 17
mLDNA (mLDNA-package), 2
mLDNA-package, 2
netFeatureMatrix, 18
optimalScore, 19
plotROC, 21
predictor, 22
PSOL_InitialNegativeSelection, 24
PSOL_NegativeExpansion, 25
PSOL_ResultExtraction, 27
randomSeed, 28
saltData (data), 11