Package 'ADaM2'

November 27, 2019

Type Package

Title Adaptive Daisy Model

Version 0.1.0

Author Clare Pacini, Emre Karakoc and Francesco Iorio

Maintainer Francesco Iorio <francesco.iorio@sanger.ac.uk>

Description The ADaM package implements a semi-supervised algorithm for computing a fuzzy-intersection of non-fuzzy sets by adaptively determining the minimal number of sets to which an element should belong in order to be a member of the fuzzy-intersection (the membership threshold). This threshold maximises the deviance from expectation of the cardinality of the resulting fuzzy-intersection, as well as the coverage of predefined elements. This method can be used to identify the minimal number of cell lines from a given tissue in which the inactivation of a gene (for example via CRISPR-Cas9 targeting) should exert a reduction of viabilty (or fitness effect) in order for that gene to be considered a core-fitness essential gene for the tissue under consideration. This method is used to discriminate between core-fitness and context-specific essential genes in a study describing a large scale genome-wide CRISPR-Cas9 pooled drop-out screening (Behan FM & Iorio F & Picco G et al., Prioritisation of cancer therapeutic targets using CRISPR-Cas9 screens. Nature, In press).

License GPL-2

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

Imports mixdist

NeedsCompilation no

R topics documented:

`ADAM2.CalculateBayesianfactor`

Calculate the Log transformed Bayesian Factor given a bimodal distribution of gene ranks or slopes of the linear fit of ranks.

Description

This function calculates the log transformed bayesian factor between essential and non-essential genes, by fitting two normal distributins to the bimodal distribution of the ranks of genes in their *N*-th percentile least dependent cell lines ordered wrt gene effect score/estimated slopes from linear fitting of the ordered ranks of genes in cell lines. The function fits a two normal distributions on the bimodal distribution and estimates the mean the standard deviation of these distributions. Based on the estimations the bayesian factor is calculated for each gene using the probabilities of genes coming from essential or non-essential distributions.

Usage

```
ADAM2.CalculateBayesianfactor(RankDistribution,
                display=TRUE,
                prefix=BayesianFactor)
```

Arguments

`RankDistribution`
: Quantative knockout screen dependency matrix where rows are genes and columns are samples. A real number in position *[i,j]* represents the strength of dependency which indicates the amaount of loss of fitness in the *j*-th sample in case of the inactivation of the *i*-th gene. Higher strength of dependency indicates higher probability of beign a core fitness gene. These values are used for ranking the genes in terms of their dependecy strength.

`display`
: Boolean, default is TRUE. Should plots of the bimodal normal distribution fitting

`prefix`
: if the display is false the plots are generated in the working directory using the prefix.

Details

This function is using the mixdist R library to fit the given data into a distribution that can be represented as a mixture of two normal distributions. The mean and the standard deviations for each normal distribution is estimated Each normal distribution corresponds to the essential and non-essential genes. For each gene a bayesian factor is calculated which can be defined as the Prob(gene|essential)/Prob(gene|non-essential). The log transformed bayesian factors are reported by this function.

Value

A data frame with the following columns:

`Gene`	Gene name
`logBF`	Log of the Bayesian Factor

Author(s)

C. Pacini, E. Karakoc & F. Iorio

Examples

```
data(exampleSBFData)
results <- ADAM2.SlopeCF(depMat=exampleSBFData,display=TRUE)
bfresults <- ADAM2.CalculateBayesianfactor(RankDistribution=results$LeastDependent)
```

`ADAM2.coreFitnessGenes`

Calculate the Core Fitness genes given the binary dependency matrix and the minimal number of cell line threshold.

Description

This function identifies the Core Fitness genes using the Adaptive Daisy Model (implemented ADAM) starting from a binary dependency matrix.

Usage

```
ADAM2.coreFitnessGenes(depMat,
                               crossoverpoint)
```

Arguments

`depMat`	Binary dependency matrix, rows are genes and columns are samples. 1 in position *[i,j]* indicates that inactivation of the *i*-th gene exerts a significant loss of fitness in the *j*-th sample, 0 otherwise.
`crossoverpoint`	minimum number of cell lines in which a gene needs to be fitness in order to be called core-fitness

Details

This function calculates the Core Fitness essential genes based on the calculated minimum number of cell lines that optimizes the True positive rates with log10 odds ratios. log10 odd ratios are calculated of observed vs. expected profiles of cumulative number of fitness genes in fixed number of cell lines. Expected values are the mean of those observed across randomised version of the observed binary matrix.

Value

A vector that containing the Core Fitness Genes:

Author(s)

C. Pacini, E. Karakoc & F. Iorio

Examples

```
data(exampleDepMat)
cfgenes <- ADAM2.coreFitnessGenes(depMat=exampleDepMat,crossoverpoint=3800)
```

ADAM2.coreFitnessGenesWrapper

Calculate the Core Fitness genes, starting from the binary dependency matrix.

Description

This function identifies the Core Fitness genes using the Adaptive Daisy Model (implemented ADAM) starting from a binary dependency matrix.

Usage

```
ADAM2.coreFitnessGenesWrapper(depMat,
                      display=TRUE,
                      main_suffix=fitness genes in at least 1 cell line,
                      xlab=n. dependent cell lines,
                      ntrials=1000)
```

Arguments

depMat	Binary dependency matrix, rows are genes and columns are samples. 1 in position *[i,j]* indicates that inactivation of the *i*-th gene exerts a significant loss of fitness in the *j*-th sample, 0 otherwise.
display	Boolean, default is TRUE. Should bar plots of the dependency profiles be plotted
main_suffix	If display=TRUE, title suffix to give to plot of number of genes depleted in a give number of cell lines, default is 'genes depleted in at least 1 cell line'
xlab	label to give to x-axis of the plots, default is 'n. cell lines'
ntrials	Integer, default =1000. How many times to randomly perturb dependency matrix to generate the null distributions.

Details

This function calculates the Core Fitness essential genes based on the calculated minimum number of cell lines that optimizes the True positive rates with log10 odds ratios. log10 odd ratios are calculated of observed vs. expected profiles of cumulative number of fitness genes in fixed number of cell lines. Expected values are the mean of those observed across randomised version of the observed binary matrix.

Value

A vector that containing the Core Fitness Genes:

Author(s)

C. Pacini, E. Karakoc & F. Iorio

Examples

```
data(exampleDepMat)
cfgenes <- ADAM2.coreFitnessGenesWrapper(depMat=exampleDepMat,ntrials=1000)
```

ADAM2.empiricalOdds *Empirical odds of number of fitness genes per number of cell lines*

Description

This function calculates log10 odd ratios of observed vs. expected profiles of cumulative number of fitness genes in fixed number of cell lines. Expected values are the mean of those observed across randomised version of the observed binary matrix.

Usage

```
ADAM2.empiricalOdds(observedCumSum,
                    simulatedCumSum)
```

Arguments

observedCumSum	Observed profile of cumulative sum of numbers of fitness genes in fixed number of cell lines. This is generated by the ADAM.panessprofile function.
simulatedCumSum	Random profiles of cumulative sum of fitness genes in fixed number of cell lines. This is generated by the function ADAM.generateNullModel.

Value

A named vector:

odds	log base 10 odd ratios of observed versus expected cumulative sums of number of fitness genes across fixed numbers of cell lines.

Author(s)

C. Pacini, E. Karakoc & F. Iorio

See Also

ADAM2.panessprofile, ADAM2.generateNullModel

Examples

```
data(exampleDepMat)
observed<-ADAM2.panessprofile(depMat=exampleDepMat)
null_m<-ADAM2.generateNullModel(depMat=exampleDepMat)
logOdds <- ADAM2.empiricalOdds(observedCumSum=observed$CUMsums,simulatedCumSum=null_m$nullCumSUM)
logOdds
```

`ADAM2.generateNullModel` *Generate null profile of number of fitness genes across fixed numbers of cell lines and cumulative sums.*

Description

This function randomly perturbs the binary dependency matrix to generate a null distribution of profiles of fitness genes across fixed number of cell lines, and corresponding null distribution of cumulative sums.

Usage

```
ADAM2.generateNullModel(depMat,
                        ntrials=1000,
                        display=TRUE)
```

Arguments

`depMat`	Binary dependency matrix, rows are genes and columns are samples. 1 in position *[i,j]* indicates that inactivation of the *i*-th gene exerts a significant loss of fitness in the *j*-th sample, 0 otherwise.
`ntrials`	Integer, default =100. How many times to randomly perturb dependency matrix to generate the null distributions.
`display`	Boolean, default is TRUE. Should bar plots of the dependency profiles be plotted

Details

For a number of trials specified in (`ntrials`) the inputted binary dependency matrix is randomised, keeping its column marginal sums. The profiles of fitness genes across fixed number of cell lines, and corresponding cumulative sums, are returned for each random perturbation.

Value

A list with the following two named vectors:

`nullProf`	Matrix of number of fitness genes for fixed number of cell lines from. Each rows of matrix corresponds to a random trial.
`nullCumSum`	Matrix of profile of cumulative number of fitness genes in fixed number of cell lines. Each row of matrix is one random trial.

Author(s)

C. Pacini, E. Karakoc & F. Iorio

Examples

```
data(exampleDepMat)
pprofile <- ADAM2.generateNullModel(depMat = exampleDepMat,ntrials=1000)
```

`ADAM2.panessprofile` *Calculate profile of number of fitness genes across fixed numbers of cell lines and cumulative sums.*

Description

This function calculates the numbers (and cumulative numbers) of genes whose inactivation exerts a fitness effect in *n* cell lines, varying *n* from 1 to the number of cell lines in the dataset in input.

Usage

```
ADAM2.panessprofile(depMat,
                    display=TRUE,
                    main_suffix=fitness genes in at least 1 cell line,
                    xlab=n. dependent cell lines)
```

Arguments

`depMat`	Binary dependency matrix, rows are genes and columns are samples. 1 in position *[i,j]* indicates that inactivation of the *i*-th gene exerts a significant loss of fitness in the *j*-th sample, 0 otherwise.
`display`	Boolean, default is TRUE. Should bar plots of the dependency profiles be plotted
`main_suffix`	If display=TRUE, title suffix to give to plot of number of genes depleted in a give number of cell lines, default is 'genes depleted in at least 1 cell line'
`xlab`	If display=TRUE, label to give to x-axis of the plots, default is 'n. cell lines'

Value

A list with the following two named vectors:

`panessprof`	Number of genes that are depleted for a number of cell lines
`CUMsums`	Cumulative number of genes depleted in at least x cell lines

Author(s)

C. Pacini, E. Karakoc & F. Iorio

Examples

```
data(exampleDepMat)
pprofile <- ADAM2.panessprofile(depMat = exampleDepMat)
```

ADAM2.PercentileAverageCF

Calculate the Core Fitness genes using the Average 90th-percentile least dependent cell line from Quantative knockout screen dependency matrix.

Description

This function identifies the Core Fitness genes from a given Quantative knockout screen dependency matrix where each row is gene and each column the cell line. The function uses all the cell lines and identifies the genes that are essential in majority of the cell lines.

Usage

```
ADAM2.PercentileAverageCF(depMat,
                    display=TRUE,
                    percentile=0.9,
                    prefix=PercentileAverageMethod)
```

Arguments

depMat	Quantative knockout screen dependency matrix where rows are genes and columns are samples. A real number in position *[i,j]* represents the strength of dependency which indicates the amaount of loss of fitness in the *j*-th sample in case of the inactivation of the *i*-th gene. Higher strength of dependency indicates higher probability of beign a core fitness gene. These values are used for ranking the genes in terms of their dependecy strength.
display	Boolean, default is TRUE. Should bar plots of the dependency profiles be plotted
percentile	percentage of the cell lines where the given gene should show depletion. The default value is 0.9 indicating least dependent 90th percentile cell line.
prefix	if the display is false the plots are generated in the working directory using the prefix.

Details

This function implements the idea that if a gene is essential then it should fall in the top Z most depleted genes in at least 90 For a given gene, we can rank its gene effect score in each cell line, then arrange cell lines in order of increasing gene effect score for that gene. The average ranks of the genes in the 90th percentile of least depleted genes are calculated. Z is choosen as the minimum density between the two gaussian distributions that are estimated from these average rankings. All genes with average rank less than this threshold in their 90th percentile least depleted cell lines are reported.

Value

A list of the following vectors:

cfgenes	Vector of number of genes that are core fitness genes
LeastDependent	A dataframe where each row corresponds to a gene.There are two columns: *Value* stores the average rank of the gene at the *N*-th percentile least dependent cell lines and the *Gene* stores the gene name
threshold	The rank threshold for core fitness genes

Author(s)

C. Pacini, E. Karakoc & F. Iorio

Examples

```
data(exampleSBFData)
results <- ADAM2.PercentileAverageCF(depMat=exampleSBFData)
cfgenes <- results$cfgenes
```

`ADAM2.PercentileCF` *Calculate the Core Fitness genes using the 90th-percentile least dependent cell line from Quantative knockout screen dependency matrix.*

Description

This function identifies the Core Fitness genes from a given Quantative knockout screen dependency matrix where each row is gene and each column the cell line. The function uses all the cell lines and identifies the genes that are essential in majority of the cell lines.

Usage

```
ADAM2.PercentileCF(depMat,
             display=TRUE,
             percentile=0.9,
             prefix=PercentileMethod)
```

Arguments

`depMat`	Quantative knockout screen dependency matrix where rows are genes and columns are samples. A real number in position *[i,j]* represents the strength of dependency which indicates the amaount of loss of fitness in the *j*-th sample in case of the inactivation of the *i*-th gene. Higher strength of dependency indicates higher probability of beign a core fitness gene. These values are used for ranking the genes in terms of their dependecy strength.
`display`	Boolean, default is TRUE. Should bar plots of the dependency profiles be plotted
`percentile`	percentage of the cell lines where the given gene should show depletion. The default value is 0.9 indicating 90-th percentile least dependent cell line.
`prefix`	if the display is false the plots are generated in the working directory using the prefix.

Details

This function implements the idea that if a gene is essential then it should fall in the top Z most depleted genes in at least 90 For a given gene, we can rank its gene effect score in each cell line, then arrange cell lines in order of increasing gene effect score for that gene. This creates a bimodal distribution of gene ranks in their 90th-percentile least depleted lines. Z is choosen as the minimum density between the two normal distributions that are estimated from these ranks. All genes with rank less than this threshold in their 90th percentile cell lines are reported

Value

A list of the following vectors:

`cfgenes`	Vector of number of genes that are core fitness genes
`LeastDependent`	A dataframe where each row corresponds to a gene.There are two columns: *Value* stores the rank of the gene at the *N*-th percentile least dependent cell line and the *Gene* stores the gene name
`threshold`	The rank threshold for core fitness genes

Author(s)

C. Pacini, E. Karakoc & F. Iorio

Examples

```
data(exampleSBFData)
results <- ADAM2.PercentileCF(depMat=exampleSBFData,display=TRUE)
cfgenes <- results$cfgenes
```

`ADAM2.randomisedepMat` *Binary matrix randomisation preserving column totals*

Description

This function takes in input a matrix and shuffles its entries column wisely. If the matrix is binary then then matrix resulting from this shuffling will have the same column marginal totals of the inpputted one.

Usage

```
ADAM2.randomisedepMat(depMat)
```

Arguments

`depMat`	A numerical matrix.

Value

The matrix given in input with entries shuffled column wisely.

Author(s)

C. Pacini, E. Karakoc & F. Iorio

Examples

```
data(exampleDepMat)
rnd_exampleDepMat<-ADAM2.randomisedepMat(exampleDepMat)
```

`ADAM2.setEssentialGenes` *Set reference set of predefined essential genes*

Description

This function takes in input a filename that contains the predefined essential sets that are used for calculating true positive rates

Usage

```
ADAM2.setEssentialGenes(reffile)
```

Arguments

`reffile`	A text file that contains a gene name per line. These genes are predefined essential genes.

Value

vector of predefined reference genes that can be used in ADAM.truePositiveRate function

Author(s)

C. Pacini, E. Karakoc & F. Iorio

`ADAM2.SlopeCF` *Calculate the Core Fitness genes using linear modeling of the ranks of genes in all cell lines ordered wrt to their gene score effects*

Description

This function identifies the Core Fitness genes from a given Quantative knockout screen dependency matrix where each row is gene and each column the cell line. The function uses all the cell lines and identifies the genes that are essential in majority of the cell lines.

Usage

```
ADAM2.SlopeCF(depMat,
              display=TRUE,
              prefix=SlopeMethod)
```

Arguments

depMat	Quantative knockout screen dependency matrix where rows are genes and columns are samples. A real number in position *[i,j]* represents the strength of dependency which indicates the amount of loss of fitness in the *j*-th sample in case of the inactivation of the *i*-th gene. Higher strength of dependency indicates higher probability of beign a core fitness gene. These values are used for ranking the genes wrt to gene effect scores.
display	Boolean, default is TRUE. Should bar plots of the dependency profiles be plotted
prefix	if the display is false the plots are generated in the working directory using the prefix.

Details

This function implements the idea that if a gene is essential then it should have ranked better in all cell lines including the least dependent cell lines. Instead of calculating rank threshold the ranks are modeled as a linear relation. For a given gene, we can rank its gene effect score in each cell line, then arrange cell lines in order of increasing gene effect score for that gene. The ranks of the genes in these cell lines are fitted to a linear model where smaller slope values indicates higher dependency in all the cell lines. A slope threshold is choosen as the minimum density between the two gaussian distributions that are estimated from the distribution of slopes. All genes with a slope less than this threshold is reported. Notice that we do not need to put a constraint such as 90th percentile least depleated cell lines.

Value

A list of the following vectors:

cfgenes	Vector of number of genes that are core fitness genes
LeastDependent	A dataframe where each row corresponds to a gene.There are two columns: *Value* stores the slope of linear model that fits the rank of the gene in allcell lines and the *Gene* stores the gene name
threshold	The slope threshold for core fitness genes

Author(s)

C. Pacini, E. Karakoc & F. Iorio

Examples

```
data(exampleSBFData)
results <- ADAM2.SlopeCF(depMat=exampleSBFData)
cfgenes <- results$cfgenes
```

ADAM2.tradeoffEO_TPR *Calculate ADAM model threshold*

Description

This function finds the minimum number of cell lines in which a gene needs to be fitness in order to be called core-fitness. This is defined as the *n* providing the best trade-off between i) coverage of priori-known essential genes in the resulting set of predicted core-fitness genes, i.e. fitness in at least *n* cell lines, and ii) deviance from expectation of the number of fitness genes in *n* cell lines.

Usage

```
ADAM2.tradeoffEO_TPR(EO,
                     TPR,
                     test_set_name)
```

Arguments

EO	Profile of empirical odds values. Computed with the ADAM2.empiricalOdds function.
TPR	Profile of True positive rates for across number of cell line. Computed with the ADAM2.truePositiveRate function.
test_set_name	Name to give to the analysis, used for plotting titles.

Details

Compare and plot the log10 odds ratios with the true positive rates to find the cross over point where the true positive rate falls below the odds ratio.

Value

ADAM model threshold:

point	Number of cell lines for which a gene needs to be a fitness gene in order to be predicted as core-fitness gene.

Author(s)

C. Pacini, E. Karakoc & F. Iorio

See Also

ADAM2.empiricalOdds, ADAM2.truePositiveRate

Examples

```
#load in example binary depletion matrix
data(exampleDepMat)

# Generate the profiles of number of fitness genes across number of cell lines from
# observed data and corresponding comulative sums.
pprofile<-ADAM2.panessprofile(depMat=exampleDepMat)
```

```
# Generate a set of random profiles of number of genes depleted for a number of cell lines
# and corresponding cumulative sums by perturbing observed data.
nullmodel<-ADAM2.generateNullModel(depMat=exampleDepMat,ntrials = 1000)

#load a reference set of essential genes
data(curated_BAGEL_essential)

# Calculate log10 odd ratios of observed/expected profiles of cumulative number of fitness
# genes in fixed number of cell lines.
# Observed values are from the ADAM.panessprofile function and expected are the average of
# random set from ADAM2.generateNullModel
EO<-ADAM2.empiricalOdds(observedCumSum = pprofile$CUMsums,simulatedCumSum =nullmodel$nullCumSUM )

# Calculate True positive rates for fitness genes in at least n cell lines in the observed
# dependency matrix, with positive cases from a reference set of essential genes
TPR<-ADAM2.truePositiveRate(exampleDepMat,curated_BAGEL_essential)

# Calculate minimum number of cell lines a gene needs to be a fitness gene in order to
# be considered as a core-fitness gene
crossoverpoint<-ADAM2.tradeoffEO_TPR(EO,TPR$TPR,test_set_name = curated BAGEL essential)
```

`ADAM2.truePositiveRate` *Profile of True Positive Rates*

Description

This function calculates a profile of True Positive Rates for fitness genes in at least n cell lines, with positive cases from a reference set of essential genes.

Usage

```
ADAM2.truePositiveRate(depMat,
                       essentialGeneSet)
```

Arguments

`depMat`	Binary dependency matrix, rows are genes and columns are samples. 1 in position *[i,j]* indicates that inactivation of the *i*-th gene exerts a significant loss of fitness in the *j*-th sample, 0 otherwise.
`essentialGeneSet`	Reference set of predefined essential genes. This is used to define positive cases.

Details

This function calculates true positive rates for fitness genes in at least n cell lines (for each n). First, this function calculates the number of cell lines each gene is a fitness gene. Second, for a given number of cell lines, the set of genes that are fitness genes in at least that number of cell lines is determined. Finally, this set of genes is then compared to the reference set of essential genes to calculate a true positive rate.

Value

A list of the following vectors:

P	Vector of number of genes that are depleted for a number of cell lines.
TP	Vector of number of genes in sets of P are true positives, i.e. in the essentialGeneSet.
TPR	TP divided by number of genes in set essentialGeneSet to give the true positive rate.

Author(s)

C. Pacini, E. Karakoc & F. Iorio

Examples

```
data(exampleDepMat)
pprofile<-ADAM2.panessprofile(depMat=exampleDepMat)
nullmodel<-ADAM2.generateNullModel(depMat=exampleDepMat,ntrials = 1000)
data(curated_BAGEL_essential)
EO<-ADAM2.empiricalOdds(observedCumSum = pprofile$CUMsums,simulatedCumSum =nullmodel$nullCumSUM )
TPR<-ADAM2.truePositiveRate(exampleDepMat,curated_BAGEL_essential)
```

---

curated_BAGEL_essential

Predefined essential core set

Description

A dataset containing predefined essential genes and it is used for calculating the true positive rates in ADAM Model

Usage

```
curated_BAGEL_essential
```

Format

A vector of 326 strings

columns Gene names

exampleDepMat *Binary dependency map of 17,995 genes for 32 cell lines*

Description

A dataset containing the binary dependencies where rows are genes and columns are samples. 1 indicates that inactivation of the corresponding gene exerts a significant loss of fitness in the corresponding cell line, 0 otherwise.

Usage

```
exampleDepMat
```

Format

A matrix with 17995 rows and 32 columns:

rows Genes

columns cell lines, samples

exampleSBFData *Quantitative dependency map of 17,995 genes for 31 cell lines*

Description

A dataset representing the dependencies where rows are genes and columns are samples. Each cell is a double value that indicates dependency score where higher values corresponds to a significant loss of fitness incase of inactivation

Usage

```
exampleSBFData
```

Format

A matrix with 17995 rows and 31 columns:

rows Genes

columns cell lines, samples

Index