Package 'xcore'

July 19, 2025

Title xcore expression regulators inference

Version 1.13.0

Description xcore is an R package for transcription factor activity modeling based on known molecular signatures and user's gene expression data. Accompanying xcoredata package provides a collection of molecular signatures, constructed from publicly available ChiP-seq experiments. xcore use ridge regression to model changes in expression as a linear combination of molecular signatures and find their unknown activities. Obtained, estimates can be further tested for significance to select molecular signatures with the highest predicted effect on the observed expression changes.

License GPL-2

Encoding UTF-8

LazyData false

Depends R (>= 4.2)

Imports DelayedArray (>= 0.18.0), edgeR (>= 3.34.1), foreach (>= 1.5.1), GenomicRanges (>= 1.44.0), glmnet (>= 4.1.2), IRanges (>= 2.26.0), iterators (>= 1.0.13), magrittr (>= 2.0.1), Matrix (>= 1.3.4), methods (>= 4.1.1), MultiAssayExperiment (>= 1.18.0), stats, S4Vectors (>= 0.30.0), utils

Suggests AnnotationHub (>= 3.0.2), BiocGenerics (>= 0.38.0), BiocParallel (>= 1.28), BiocStyle (>= 2.20.2), data.table (>= 1.14.0), devtools (>= 2.4.2), doParallel (>= 1.0.16), ExperimentHub (>= 2.2.0), knitr (>= 1.37), pheatmap (>= 1.0.12), proxy (>= 0.4.26), ridge (>= 3.0), rmarkdown (>= 2.11), rtracklayer (>= 1.52.0), testthat (>= 3.0.0), usethis (>= 2.0.1), xcoredata

VignetteBuilder knitr

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.2

Config/testthat/edition 3

biocViews GeneExpression, GeneRegulation, Epigenetics, Regression, Sequencing

git_url https://git.bioconductor.org/packages/xcore

git_branch devel

git_last_commit a16d749

git_last_commit_date 2025-04-15

Repository Bioconductor 3.22

Date/Publication 2025-07-18

Author Maciej Migdał [aut, cre] (ORCID:
<https://orcid.org/0000-0002-8021-7263>),
Bogumił Kaczkowski [aut] (ORCID:
<https://orcid.org/0000-0001-6554-5608>)

Maintainer Maciej Migdał <mcjmigdal@gmail.com>

Contents

addSignatures *Add molecular signatures to MultiAssayExperiment*

Description

addSignatures extends mae by adding to it new experiments. Rows consistency is ensured by taking an intersection of rows after new experiments are added.

Usage

```
addSignatures(mae, ..., intersect_rows = TRUE)
```

Arguments

mae	MultiAssayExperiment object.
...	named experiments to be added to mae.
intersect_rows	logical flag indicating if only common rows across experiments should be included. Only set to FALSE if you know what you are doing.

Value

MultiAssayExperiment object with new experiments added.

Examples

```
data("rinderpest_mini", "remap_mini")
base_lvl <- "00hr"
design <- matrix(
  data = c(1, 0, 0,
           1, 0, 0,
           1, 0, 0,
           0, 1, 0,
           0, 1, 0,
           0, 1, 0,
           0, 0, 1,
           0, 0, 1,
           0, 0, 1),
  ncol = 3,
  nrow = 9,
  byrow = TRUE,
  dimnames = list(colnames(rinderpest_mini), c("00hr", "12hr", "24hr")))
mae <- prepareCountsForRegression(
  counts = rinderpest_mini,
  design = design,
  base_lvl = base_lvl)
mae <- addSignatures(mae, remap = remap_mini)
```

applyOverColumnGroups *Apply function over groups of columns*

Description

Returns a array obtained by applying a function to rows of submatrices of the input matrix, where the submatrices are divided into specified groups of columns.

Usage

```
applyOverColumnGroups(mat, groups, f, ...)
```

Arguments

mat	a matrix.
groups	a vector giving columns grouping.
f	function to be applied.
...	optional arguments to f.

Value

a matrix of dimensions nrow(mat) x nlevels(groups).

applyOverDFList *Apply function over selected column in list of data frames*

Description

applyOverDFList operates on a list of data frames where all data frames has the same size and columns. Column of interest is extracted from each data frame and column binded in groups, next fun is applied over rows. Final result is a matrix with result for each group on a separate column. Function is parallelized over groups.

Usage

```
applyOverDFList(list_of_df, col_name, fun, groups)
```

Arguments

list_of_df	list of data.frames.
col_name	string specifying column in data.frames to apply fun on.
fun	function to apply, should take a single vector as a argument.
groups	factor defining how elements of list_of_df should be grouped.

Value

matrix with nrow(list_of_df[[1]]) rows and nlevels(groups) columns.

`design2factor` *Transform design matrix to factor*

Description

Transform design matrix to factor

Usage

```
design2factor(design)
```

Arguments

`design`	design matrix

Value

factor

Examples

```
## Not run:
design <- matrix(data = c(1, 1, 0, 0, 0, 0, 1, 1),
                 nrow = 4,
                 ncol = 2,
                 dimnames = list(c(paste("sample", 1:4)), c("gr1", "gr2")))
design2factor(design)

## End(Not run)
```

`estimateStat` *Estimate linear models goodness of fit statistic*

Description

Estimate goodness of fit statistic of penalized linear regression models. Works with different goodness of fit statistic functions.

Usage

```
estimateStat(x, y, u, s, method = "cv", nfold = 10, statistic = rsq, alpha = 0)
```

Arguments

x	input matrix, of dimension nobs x nvars; each row is an observation vector. Can be in sparse matrix format (inherit from class "sparseMatrix" as in package Matrix)
y	response variable. Quantitative for family="gaussian", or family="poisson" (non-negative counts). For family="binomial" should be either a factor with two levels, or a two-column matrix of counts or proportions (the second column is treated as the target class; for a factor, the last level in alphabetical order is the target class). For family="multinomial", can be a nc>=2 level factor, or a matrix with nc columns of counts or proportions. For either "binomial" or "multinomial", if y is presented as a vector, it will be coerced into a factor. For family="cox", preferably a Surv object from the survival package: see Details section for more information. For family="mgaussian", y is a matrix of quantitative responses.
u	offset vector as in glmnet. "U" experiment in mae.
s	user supplied lambda.
method	currently only cross-validation is implemented.
nfold	number of fold to use in cross-validation.
statistic	function computing goodness of fit statistic. Should accept y, x, offset arguments and return a numeric vector of the same length. See rsq, mse for examples.
alpha	The elasticnet mixing parameter, with $0 \leq \alpha \leq 1$. The penalty is defined as $$(1-\alpha)/2||\beta||_2^2 + \alpha||\beta||_1.$$ alpha=1 is the lasso penalty, and alpha=0 the ridge penalty.

Value

numeric vector of statistic estimates.

---

filterSignatures *Filter signatures by coverage*

Description

Filter signatures overlapping low or high number of promoters. Useful to get rid of signatures that have very low variance.

Usage

```
filterSignatures(
  mae,
  min = 0.05,
  max = 0.95,
  ref_experiment = "Y",
  omit_experiments = c("Y", "U")
)
```

Arguments

mae	MultiAssayExperiment object.
min	length one numeric between 0 and 1 defining minimum promoter coverage for the signature to pass filtering.
max	length one numeric between 0 and 1 defining maximum promoter coverage for the signature to pass filtering.
ref_experiment	string giving name of experiment to use for inferring total number of promoters.
omit_experiments	character giving names of experiments to exclude from filtering.

Value

MultiAssayExperiment object with selected experiments filtered.

Examples

```
data("rinderpest_mini", "remap_mini")
base_lvl <- "00hr"
design <- matrix(
  data = c(1, 0, 0,
           1, 0, 0,
           1, 0, 0,
           0, 1, 0,
           0, 1, 0,
           0, 1, 0,
           0, 0, 1,
           0, 0, 1,
           0, 0, 1),
  ncol = 3,
  nrow = 9,
  byrow = TRUE,
  dimnames = list(colnames(rinderpest_mini), c("00hr", "12hr", "24hr")))
mae <- prepareCountsForRegression(
  counts = rinderpest_mini,
  design = design,
  base_lvl = base_lvl)
mae <- addSignatures(mae, remap = remap_mini)
mae <- filterSignatures(mae)
```

---

fisherMethod *Combine p-values using Fisher method*

Description

Fisher's method is a meta-analysis technique used to combine the results from independent statistical tests with the same hypothesis (Wikipedia article).

Usage

```
fisherMethod(p.value, lower.tail = FALSE, log.p = TRUE)
```

Arguments

p.value	a numeric vector of p-values to combine.
lower.tail	logical; if TRUE (default), probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.
log.p	logical; if TRUE, probabilities p are given as log(p).

Value

a number giving combined p-value.

---

getCoverage *Calculate regions coverage*

Description

getCoverage calculates coverage of regions (rows in interaction matrix) by features (columns). It is possible to specify features grouping variable gr then coverage tells how many distinct groups the region overlap with.

Usage

```
getCoverage(mat, gr)
```

Arguments

mat	dgCMatrix interaction matrix such as produced by getInteractionMatrix.
gr	factor specifying features groups. Must have length equal to number of columns in mat.

Value

Numeric vector.

Examples

```
data("remap_mini")
y <- colnames(remap_mini)

# simple coverage
gr <- seq_along(y) %>% as.factor()
getCoverage(remap_mini, gr)

# per cell type coverage
gr <- sub(".*\\.", "", y) %>% as.factor()
getCoverage(remap_mini, gr)
```

`getInteractionMatrix` *Compute interaction matrix*

Description

`getInteractionMatrix` construct interaction matrix between two Granges objects. Names of object a became row names and names of b column names.

Usage

```
getInteractionMatrix(a, b, ext = 500, count = FALSE)
```

Arguments

a	GRanges object.
b	GRanges object.
ext	Integer specifying number of base pairs the a coordinates should be extended in upstream and downstream directions.
count	Logical indicating if matrix should hold number of overlaps between a and b or if FALSE presence / absence indicators.

Value

Sparse matrix of class dgCMatrix, with rows corresponding to a and columns to b. Each cell holds a number indicating how many times a and b overlapped.

Examples

```
a <- GenomicRanges::GRanges(
  seqnames = c("chr20", "chr4"),
  ranges = IRanges::IRanges(
    start = c(62475984L, 173530220L),
    end = c(62476001L, 173530236L)),
  strand = c("-", "-"),
  name = c("hg19::chr20:61051039..61051057,-;hg_188273.1",
           "hg19::chr4:174451370..174451387,-;hg_54881.1"))
b <- GenomicRanges::GRanges(
  seqnames = c("chr4", "chr20"),
  ranges = IRanges::IRanges(
    start = c(173530229L, 63864270L),
    end = c(173530236L, 63864273L)),
  strand = c("-", "-"),
  name = c("HAND2", "GATA5"))
getInteractionMatrix(a, b)
```

getVarianceWeightedAvgCoeff *Calculate variance weighted average coefficients matrix*

Description

Calculate variance weighted average coefficients matrix

Usage

```
getVarianceWeightedAvgCoeff(pvalues, groups)
```

Arguments

pvalues	list of data.frames outputs from ridgePvals.
groups	factor giving the grouping.

Value

variance weighted average coefficients matrix

isTRUEorFALSE *Check if argument is a binary flag*

Description

Check if argument is a binary flag

Usage

```
isTRUEorFALSE(x)
```

Arguments

x	object to test

Value

binary flag

mae *Calculate Mean Absolute Error*

Description

Calculate Mean Absolute Error

Usage

```
mae(y, yhat, ...)
```

Arguments

y	numeric vector of observed expression values.
yhat	numeric vector of predicted expression values.
...	not used.

Value

numeric vector

maeSummary *Helper summarizing MAE object*

Description

Helper summarizing MAE object

Usage

```
maeSummary(mae)
```

Arguments

mae	MultiAssayExperiment object.

Value

named list giving number of rows and columns, overall mean and standard deviation in mae's experiments.

`modelGeneExpression` *Gene expression modeling pipeline*

Description

`modelGeneExpression` uses parallelization if parallel backend is registered. For that reason we advise against passing `parallel` argument to internally called `cv.glmnet` routine.

Usage

```
modelGeneExpression(
  mae,
  yname = "Y",
  uname = "U",
  xnames,
  design = NULL,
  standardize = TRUE,
  parallel = FALSE,
  pvalues = TRUE,
  precalcmodels = NULL,
  ...
)
```

Arguments

`mae`	MultiAssayExperiment object such as produced by `prepareCountsForRegression`.
`yname`	string indicating experiment in `mae` to use as the expression input.
`uname`	string indicating experiment in `mae` to use as the basal expression level.
`xnames`	character indicating experiments in `mae` to use as molecular signatures.
`design`	matrix giving the design matrix for the samples. Default (`NULL`) is to use design found in `mae` metadata. Columns corresponds to samples groups and rows to samples names. Only samples included in the design will be processed.
`standardize`	logical flag indicating if the molecular signatures should be scaled. Advised to be set to `TRUE`.
`parallel`	parallel argument to internally used `cv.glmnet` function. Advised to be set to `FALSE` as it might interfere with parallelization used in `modelGeneExpression`.
`pvalues`	logical flag indicating if significance testing for the estimated molecular signatures activities should be performed.
`precalcmodels`	optional list of precomputed `'cv.glmnet'` objects for each molecular signature and sample. The elements of this list should be matching the `xnames` vector. Each of those elements should be a named list holding `'cv.glmnet'` objects for each sample. If provided those models will be used instead of running regression from scratch.
`...`	arguments passed to glmnet::cv.glmnet.

Details

For speeding up the calculations consider lowering number of folds used in internally run `cv.glmnet` by specifying `nfolds` argument. By default 10 fold cross validation is used.

The relationship between the expression (Y) and molecular signatures (X) is described using linear model formulation. The pipeline attempts to model the change in expression between basal expression level (u) and each sample, with the goal of finding the unknown molecular signatures activities. Linear models are fit using popular ridge regression implementation glmnet (Friedman, Hastie, and Tibshirani 2010).

If `pvalues` is set to `TRUE` the significance of the estimated molecular signatures activities is tested using methodology introduced by (Cule, Vineis, and De Iorio 2011) which original implementation can be found in ridge-package.

If replicates are available the signatures activities estimates and their standard error estimates can be combined. This is done by averaging signatures activities estimates and pooling their significance estimates using Stouffer's method for the Z-scores and Fisher's method for the p-values.

For detailed pipeline description we refer interested user to paper accompanying this package.

Value

Nested list with following elements

regression_models Named list with elements corresponding to signatures specified in `xnames`. Each of these is a list holding `'cv.glmnet'` objects corresponding to each sample.

pvalues Named list with elements corresponding to signatures specified in `xnames`. Each of these is a list holding `data.frame` of signature's p-values and test statistics estimated for each sample.

zscore_avg Named list with elements corresponding to signatures specified in `xnames`. Each of these is a `matrix` holding replicate average Z-scores with columns corresponding to groups in the design.

coef_avg Named list with elements corresponding to signatures specified in `xnames`. Each of these is a `matrix` holding replicate averaged signatures activities with columns corresponding to groups in the design.

results Named list of a `data.frames` holding replicate average molecular signatures, overall molecular signatures Z-score and p-values calculated over groups using Stouffer's and Fisher's methods.

Examples

```
data("rinderpest_mini", "remap_mini")
base_lvl <- "00hr"
design <- matrix(
  data = c(1, 0, 0,
           1, 0, 0,
           1, 0, 0,
           0, 1, 0,
           0, 1, 0,
           0, 1, 0,
           0, 0, 1,
           0, 0, 1,
           0, 0, 1),
  ncol = 3,
  nrow = 9,
  byrow = TRUE,
  dimnames = list(colnames(rinderpest_mini), c("00hr", "12hr", "24hr")))
```

```
mae <- prepareCountsForRegression(
  counts = rinderpest_mini,
  design = design,
  base_lvl = base_lvl)
mae <- addSignatures(mae, remap = remap_mini)
mae <- filterSignatures(mae)
res <- modelGeneExpression(
  mae = mae,
  xnames = "remap",
  nfolds = 5)
```

`modelGeneExpression_ridge_regression_wraper` *Ridge regression wrapper for modelGeneExpression*

Description

Internal function used in `modelGeneExpression`. It runs ridge regression parallelly across signatures and samples as specified by experiment design.

Usage

```
modelGeneExpression_ridge_regression_wraper(
  mae,
  yname,
  uname,
  xnames,
  groups,
  standardize,
  parallel,
  precalcmodels,
  ...
)
```

Arguments

`mae`	MultiAssayExperiment object such as produced by `prepareCountsForRegression`.
`yname`	string indicating experiment in `mae` to use as the expression input.
`uname`	string indicating experiment in `mae` to use as the basal expression level.
`xnames`	character indicating experiments in `mae` to use as molecular signatures.
`groups`	factor representation of design matrix.
`standardize`	logical flag indicating if the molecular signatures should be scaled. Advised to be set to `TRUE`.
`parallel`	parallel argument to internally used `cv.glmnet` function. Advised to be set to `FALSE` as it might interfere with parallelization used in `modelGeneExpression`.
`precalcmodels`	optional list of precomputed `'cv.glmnet'` objects for each molecular signature and sample. The elements of this list should be matching the `xnames` vector. Each of those elements should be a named list holding `'cv.glmnet'` objects for each sample. If provided those models will be used instead of running regression from scratch.
`...`	arguments passed to glmnet::cv.glmnet.

Value

Named list with elements corresponding to signatures specified in `xnames`. Each of these is a list holding `'cv.glmnet'` objects corresponding to each sample.

`modelGeneExpression_significance_testing_wraper`
Statistical testing of ridge regression estimates wrapper for modelGeneExpression

---

Description

Internal function used in `modelGeneExpression`. It runs `ridgePvals` parallelly across signatures and samples as specified by experiment design.

Usage

```
modelGeneExpression_significance_testing_wraper(
  mae,
  yname,
  uname,
  xnames,
  groups,
  standardize,
  regression_models
)
```

Arguments

`mae`	MultiAssayExperiment object such as produced by `prepareCountsForRegression`.
`yname`	string indicating experiment in `mae` to use as the expression input.
`uname`	string indicating experiment in `mae` to use as the basal expression level.
`xnames`	character indicating experiments in `mae` to use as molecular signatures.
`groups`	factor representation of design matrix.
`standardize`	logical flag indicating if the molecular signatures should be scaled. Advised to be set to `TRUE`.
`regression_models`	Named list with elements corresponding to signatures specified in `xnames`. Each of these is a list holding `'cv.glmnet'` objects corresponding to each sample. Usually returned by `modelGeneExpression_ridge_regression_wraper`.

Value

Named list with elements corresponding to signatures specified in `xnames`. Each of these is a list holding `data.frame` of signature's p-values and test statistics estimated for each sample.

mse *Calculate Mean Squared Error*

Description

Calculate Mean Squared Error

Usage

```
mse(y, yhat, ...)
```

Arguments

y	numeric vector of observed expression values.
yhat	numeric vector of predicted expression values.
...	not used.

Value

numeric vector

prepareCountsForRegression *Process count matrix for expression modeling*

Description

Expression counts are processed using edgeR following User's Guide. Shortly, counts for each sample are filtered for lowly expressed promoters, normalized for the library size and transformed into counts per million (CPM). Optionally, CPM are log2 transformed with addition of pseudo count. Basal level expression is calculated by averaging `base_lvl` samples expression values.

Usage

```
prepareCountsForRegression(
  counts,
  design,
  base_lvl,
  log2 = TRUE,
  pseudo_count = 1L,
  drop_base_lvl = TRUE
)
```

Arguments

counts	matrix of read counts.
design	matrix giving the design matrix for the samples. Columns corresponds to samples groups and rows to samples names.
base_lvl	string indicating group in design corresponding to basal expression level. The reference samples to which expression change will be compared.
log2	logical flag indicating if counts should be log2(counts per million) should be returned.
pseudo_count	integer count to be added before taking log2.
drop_base_lvl	logical flag indicating if base_lvl samples should be dropped from resulting MultiAssayExperiment object.

Value

MultiAssayExperiment object with two experiments:

U matrix giving expression values averaged over basal level samples

Y matrix of expression values

design with base_lvl dropped is stored in metadata and directly available for modelGeneExpression.

Examples

```
data("rinderpest_mini")
base_lvl <- "00hr"
design <- matrix(
  data = c(1, 0, 0,
           1, 0, 0,
           1, 0, 0,
           0, 1, 0,
           0, 1, 0,
           0, 1, 0,
           0, 0, 1,
           0, 0, 1,
           0, 0, 1),
  ncol = 3,
  nrow = 9,
  byrow = TRUE,
  dimnames = list(colnames(rinderpest_mini), c("00hr", "12hr", "24hr")))
mae <- prepareCountsForRegression(
  counts = rinderpest_mini,
  design = design,
  base_lvl = base_lvl)
```

`regressionData` *Create MultiAssayExperiment object for expression modeling*

Description

`regressionData` orgnize expression data and experiment design into MultiAssayExperiment object that can be further used in `xcore` framework. Additionally, function calculate basal expression level, for latter use in expression modeling, by averaging `base_lvl` samples expression values.

Usage

```
regressionData(expr_mat, design, base_lvl, drop_base_lvl = TRUE)
```

Arguments

`expr_mat`	matrix of expression values.
`design`	matrix giving the design matrix for the samples. Columns corresponds to samples groups and rows to samples names.
`base_lvl`	string indicating group in `design` corresponding to basal expression level. The reference samples to which expression change will be compared.
`drop_base_lvl`	logical flag indicating if `base_lvl` samples should be dropped from resulting MultiAssayExperiment object.

Details

Note that `regressionData` does not apply any normalization or transformation to the input data! Use `prepareCountsForRegression` if you want to start with raw expression counts.

Value

MultiAssayExperiment object with two experiments:

U matrix giving expression values averaged over basal level samples

Y matrix of expression values

design with `base_lvl` dropped is stored in metadata and directly available for `modelGeneExpression`.

Examples

```
data("rinderpest_mini")
base_lvl <- "00hr"
design <- matrix(
  data = c(1, 0, 0,
           1, 0, 0,
           1, 0, 0,
           0, 1, 0,
           0, 1, 0,
           0, 1, 0,
           0, 0, 1,
           0, 0, 1,
           0, 0, 1),
  ncol = 3,
```

```
  nrow = 9,
  byrow = TRUE,
  dimnames = list(colnames(rinderpest_mini), c("00hr", "12hr", "24hr")))
mae <- regressionData(
  expr_mat = rinderpest_mini,
  design = design,
  base_lvl = base_lvl)
```

`remap_mini` *xcore example molecular signatures*

Description

Molecular signatures data intended for use in xcore vignette and examples. It is build ReMap2020 molecular signatures constructed against FANTOM5 annotation, which can be found in `xcoredata` package. Here the data is only a subset limited to core promoters (`promoters_f5_core`) and randomly selected 600 signatures.

Usage

```
data(remap_mini)
```

Format

A `dgCMatrix` with 14191 rows and 600 columns holding interaction matrix for subset of ReMap2020 molecular signatures against FANTOM5 annotation. Rows corresponds to FANTOM5 promoters and columns to signatures.

`repVarianceWeightedAvgZscore`
Calculate replicate variance weighted averaged Z-scores

Description

Replicate averaged Z-scores is calculated by dividing replicate average coefficient by replicate pooled standard error.

Usage

```
repVarianceWeightedAvgZscore(pvalues, groups)
```

Arguments

`pvalues`	Data frame with `'se'` (standard error) and `'coef'` (coefficient) columns. Such as in `pvalues` output of `modelGeneExpression` .
`groups`	Factor giving group membership for samples in `pvalues`.

Value

Numeric matrix of averaged Z-scores. Columns correspond to groups and rows to predictors.

ridgePvals *Significance testing in linear ridge regression*

Description

Standard error estimation and significance testing for coefficients estimated in linear ridge regression. `ridgePvals` re-implement original method by (Cule et al. BMC Bioinformatics 2011.) found in ridge-package. This function is intended to use with `cv.glmnet` output.

Usage

```
ridgePvals(x, y, beta, lambda, standardizex = TRUE, svdX = NULL)
```

Arguments

`x`	input matrix, same as used in `cv.glmnet`.
`y`	response variable, same as used in `cv.glmnet`.
`beta`	matrix of coefficients, estimated using `cv.glmnet`.
`lambda`	lambda value for which `beta` was estimated.
`standardizex`	logical flag for x variable standardization, should be set to same value as `standarize` flag in `cv.glmnet`.
`svdX`	optional singular-value decomposition of `x` matrix. One can be obtained using `link[base]{svd}`. Passing this argument omits internal call to `link[base]{svd}`, this is useful when calling `ridgePvals` repeatedly using same `x`.

Value

a data.frame with columns

coef `beta`'s names

se `beta`'s standard errors

tstat `beta`'s test statistic

pval `beta`'s p-values

rinderpest_mini *xcore example expression data*

Description

Expression data intended for use in xcore vignette and examples. It is build from FANTOM5's 293SLAM rinderpest infection time course dataset. Here the data is only a subset limited to core promoters (`promoters_f5_core`).

Usage

```
data(rinderpest_mini)
```

Format

A matrix with 14191 rows and 6 columns holding expression counts from CAGE-seq experiment. Rows corresponds to FANTOM5 promoters and columns to time points at which expression was measured 0 and 24 hours post infection.

---

rsq *Calculate R^2*

Description

Calculate R^2

Usage

```
rsq(y, yhat, offset)
```

Arguments

y	numeric vector of observed expression values.
yhat	numeric vector of predicted expression values.
offset	numeric vector giving basal expression level.

Value

numeric vector

---

simplifyInteractionMatrix *Simplify Interaction Matrix*

Description

Simplify Interaction Matrix

Usage

```
simplifyInteractionMatrix(mat, alpha = 0.5, colname = NA)
```

Arguments

mat	dgCMatrix interaction matrix such as produced by getInteractionMatrix.
alpha	Number between 0 and 1 specifying voting threshold. Eg. for 3 column matrix alpha 0.5 will give voting criteria >= 2.
colname	character giving new column name.

Value

dgCMatrix

stoufferZMethod *Combine Z-scores using Stouffer's method*

Description

Stouffer's Z-score method is a meta-analysis technique used to combine the results from independent statistical tests with the same hypothesis. It is closely related to Fisher's method, but operates on Z-scores instead of p-values (Wikipedia article).

Usage

```
stoufferZMethod(z)
```

Arguments

z	a numeric vector of Z-score to combine.

Value

a number giving combined Z-score.

subsetWithMissing *Subset keeping missing*

Description

Subset matrix keeping unmatched rows as NA.

Usage

```
subsetWithMissing(mat, rows)
```

Arguments

mat	matrix
rows	character

Value

a matrix

translateCounts *Translate counts matrix rownames*

Description

translateCounts renames counts matrix rownames according to supplied dictionary. Function can handle many to one assignments by taking a sum or an average over counts rows. Other types of ambiguous assignments are not supported.

Usage

```
translateCounts(counts, dict)
```

Arguments

counts	matrix of expression values.
dict	named character vector mapping counts rownames to new values. Values of vector should correspond to new desired rownames, and its names to current rownames.

Value

matrix of expression values with new rownames.

Examples

```
counts <- matrix(
data = c(5, 4, 3, 2),
nrow = 2,
   dimnames = list(
     c("ENSG00000130700", "ENSG00000089225"),
     c("treatment", "control")
   )
 )
dict <- c(ENSG00000130700 = "GATA5", ENSG00000089225 = "TBX5")
translateCounts(counts, dict)
```

%>% *re-export magrittr pipe operator*

Description

re-export magrittr pipe operator

Index