Package 'simulatorZ'

September 23, 2020

Type Package

Title Simulator for Collections of Independent Genomic Data Sets

Version 1.22.0

Date 2014-08-03

Author Yuqing Zhang, Christoph Bernau, Levi Waldron

Maintainer Yuqing Zhang <zhangyuqing.pkusms@gmail.com>

Description simulatorZ is a package intended primarily to simulate collections of independent genomic data sets, as well as performing training and validation with predicting algorithms. It supports ExpressionSet and RangedSummarizedExperiment objects.

License Artistic-2.0

Encoding UTF-8

Depends R (>= 3.5), Biobase, SummarizedExperiment, survival, CoxBoost, BiocGenerics

Imports graphics, stats, gbm, Hmisc, GenomicRanges, methods

Suggests RUnit, BiocStyle, curatedOvarianData, parathyroidSE, superpc

URL https://github.com/zhangyuqing/simulatorZ

BugReports https://github.com/zhangyuqing/simulatorZ

biocViews Survival

NeedsCompilation yes

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

git_branch RELEASE_3_11

git_last_commit feb6805

git_last_commit_date 2020-04-27

Date/Publication 2020-09-23

R topics documented:

`cvSubsets` *cvSubsets*

Description

To generate a list of subsets(indices of observations) from one set

Usage

```
cvSubsets(obj, fold)
```

Arguments

`obj`	a ExpressionSet, matrix or RangedSummarizedExperiment object. If it is a matrix,columns represent samples
`fold`	the number of folds in cross validation. Number of observations in the set does not need to be a multiple of fold

Value

returns the list of indices of subsets

Author(s)

Yuqing Zhang, Christoph Bernau, Levi Waldron

Examples

```
library(curatedOvarianData)

data(E.MTAB.386_eset)



id <- cvSubsets(E.MTAB.386_eset, 3)

subsets <- lapply(1:3, function(i){E.MTAB.386_eset[1:10, id[[i]]]})
```

```
sapply(subsets, dim)

rm(subsets)



## Number of observations in the set does not need to be a multiple of

## the fold parameter

id2 <- cvSubsets(E.MTAB.386_eset, 5)

subsets <- lapply(1:5, function(j){E.MTAB.386_eset[1:10, id2[[j]]]})

sapply(subsets, dim)

rm(subsets)
```

---

funCV *funCV*

Description

Cross validation function

Usage

```
funCV(obj, fold, y.var, trainFun = masomenos, funCvSubset = cvSubsets,

    covar = NULL)
```

Arguments

obj	a ExpressionSet, matrix or RangedSummarizedExperiment object. If it is a matrix, columns represent samples
fold	the number of folds in cross validation
y.var	response variable, matrix, data.frame(with 2 columns) or Surv object
trainFun	training function, which takes gene expression matrix X and response variable y as input, the coefficients as output
funCvSubset	function to divide one Expression Set into subsets for cross validation
covar	other covariates to be added in as predictors

Value

returns the c statistics of cross validation(CV)

Author(s)

Yuqing Zhang, Christoph Bernau, Levi Waldron

Examples

```
library(curatedOvarianData)

library(GenomicRanges)

set.seed(8)

data( E.MTAB.386_eset )

eset <- E.MTAB.386_eset[1:100, 1:30]

rm(E.MTAB.386_eset)



time <- eset$days_to_death

cens.chr <- eset$vital_status

cens <- rep(0, length(cens.chr))

cens[cens.chr=="living"] <- 1

y <- Surv(time, cens)

y1 <- cbind(time, cens)



nrows <- 200; ncols <- 6
```

```
counts <- matrix(runif(nrows * ncols, 1, 1e4), nrows)

rowRanges <- GRanges(rep(c("chr1", "chr2"), c(50, 150)),

                     IRanges(floor(runif(200, 1e5, 1e6)), width=100),

                     strand=sample(c("+", "-"), 200, TRUE))

colData <- DataFrame(Treatment=rep(c("ChIP", "Input"), 3),

                     row.names=LETTERS[1:6])

sset <- SummarizedExperiment(assays=SimpleList(counts=counts),

                             rowRanges=rowRanges, colData=colData)

time <- c(1588,1929,1813,1542,1830,1775)

cens <- c(1,0,1,1,1,1)

y.vars <- Surv(time, cens)



funCV(eset, 3, y)

funCV(exprs(eset), 3, y1)

funCV(sset, 3, y.vars)

## any training function will do as long as it takes the gene expression matrix X

## and response variable y(matrix, data.frame or Surv object) as parameters, and

## return the coefficients as its value
```

geneFilter *geneFilter*

Description

the function to filter genes by Intergrative Correlation

Usage

```
geneFilter(obj, cor.cutoff = 0.5)
```

Arguments

obj	a list of ExpressionSet, matrix or RangedSummarizedExperiment objects. If its elements are matrices, columns represent samples, rows represent genes
cor.cutoff	the cutoff threshold for filtering genes. Only when the integrative correlation between every pair of sets is larger than the cutoff value, will the gene be selected.

Value

returns a list of ExpressionSets matrix or RangedSummarizedExperiment objects with genes filtered

Author(s)

Yuqing Zhang, Christoph Bernau, Levi Waldron

References

Garrett-Mayer, E., Parmigiani, G., Zhong, X., Cope, L., Gabrielson, E., Cross-study validation and combined analysis of gene expression microarray data. Biostatistics. 2008 Apr;9(2):333-354.

Examples

```
set.seed(8)

library(curatedOvarianData)

library(GenomicRanges)

data(GSE17260_eset)

data(E.MTAB.386_eset)
```

```
data(GSE14764_eset)

## to save time, we take a small subset from each dataset

esets.list <- list(GSE17260=GSE17260_eset[1:50, 1:10],

                   E.MTAB.386=E.MTAB.386_eset[1:50, 1:10],

                   GSE14764=GSE14764_eset[1:50, 1:10])

rm(E.MTAB.386_eset, GSE14764_eset, GSE17260_eset)



result.set <- geneFilter(esets.list, 0.1)

dim(result.set[[1]])



## as we cannot calculate correlation with one set, this function just

## delivers the same set if esets has length 1

result.oneset <- geneFilter(esets.list[1])

dim(result.oneset[[1]])





## Support matrices

X.list <- lapply(esets.list, function(eset){

  return(exprs(eset)) ## Columns represent samples!
```

```
})
result.set <- geneFilter(X.list, 0.1)
dim(result.set[[1]])



## Support RangedSummarizedExperiment
nrows <- 200; ncols <- 6
counts <- matrix(runif(nrows * ncols, 1, 1e4), nrows)
rowRanges <- GRanges(rep(c("chr1", "chr2"), c(50, 150)),
                     IRanges(floor(runif(200, 1e5, 1e6)), width=100),
                     strand=sample(c("+", "-"), 200, TRUE))
colData <- DataFrame(Treatment=rep(c("ChIP", "Input"), 3),
                     row.names=LETTERS[1:6])
sset <- SummarizedExperiment(assays=SimpleList(counts=counts),
                             rowRanges=rowRanges, colData=colData)
s.list <- list(sset, sset)
result.set <- geneFilter(s.list, 0.9)
## the same set should resemble each other, no genes filtered
dim(assay(result.set[[1]]))
```

getTrueModel *getTrueModel*

Description

The parametric bootstrap simulation depends on the true model of original sets.
This function is to generate useful values from the true models for further analysis.
We fit CoxBoost to the original sets and use the coefficients to simulate
the survival and censoring time. grid, survH, censH, which are useful for this purpose.
grid=grid corresponding to hazard estimations censH and survH
survH=cumulative hazard for survival times distribution
censH=cumulative hazard for censoring times distribution

Usage

```
getTrueModel(esets, y.vars, parstep, balance.variables = NULL)
```

Arguments

esets	a list of ExpressionSets, matrix or SummarizedExperiment
y.vars	a list of response variables
parstep	CoxBoost parameter
balance.variables	variable names to be balanced.

Value

returns a list of values:
beta: True coefficients obtained by fitting CoxBoost to the original ExpressionSets
grid: timeline grid corresponding to hazard estimations censH and survH
survH: cumulative hazard for survival times distribution
censH: cumulative hazard for censoring times distribution
lp: true linear predictors

Author(s)

Yuqing Zhang, Christoph Bernau, Levi Waldron

Examples

```
library(curatedOvarianData)

data(GSE14764_eset)
```

```
data(E.MTAB.386_eset)

esets.list <- list(GSE14764=GSE14764_eset[1:500, 1:20],

                   E.MTAB.386=E.MTAB.386_eset[1:500, 1:20])

rm(E.MTAB.386_eset, GSE14764_eset)



## simulate on multiple ExpressionSets

set.seed(8)



y.list <- lapply(esets.list, function(eset){

  time <- eset$days_to_death

  cens.chr <- eset$vital_status

  cens <- rep(0, length(cens.chr))

  cens[cens.chr=="living"] <- 1

  return(Surv(time, cens))

})



res1 <- getTrueModel(esets.list, y.list, 100)

## Get true model from one set

res2 <- getTrueModel(esets.list[1], y.list[1], 100)
```

```
names(res2)

res2$lp

## note that y.list[1] cannot be replaced by y.list[[1]]
```

---

masomenos *masomenos*

Description

function for Mas-o-menos algorithm

Usage

```
masomenos(X, y, option = "fast", ...)
```

Arguments

X	matrix with rows corresponding to subjects and columns to features resp
y	response variable, a data.frame, matrix, or Surv object: c(time, event)
option	whether to use C or R code to fit the marginal Cox models
...	

Value

return the coefficients

Author(s)

Yuqing Zhang, Christoph Bernau, Levi Waldron

References

Zhao, S., Huttenhower, G. P. C., and Waldron, L. (2013). Mas-o-menos:
a simple sign averaging method for discrimination in genomic data analysis.
http://biostats.bepress.com/harvardbiostat/paper158/. Accessed: 2013-10-24.

Examples

```
set.seed(8)

library(curatedOvarianData)
```

```
data( E.MTAB.386_eset )

eset <- E.MTAB.386_eset[1:100, 1:30]

rm(E.MTAB.386_eset)



X <- t(exprs(eset))



time <- eset$days_to_death

cens <- sample(0:1, 30, replace=TRUE)

y <- Surv(time, cens)



beta <- masomenos(X=X, y=y)

beta
```

---

plusMinus *plusMinus*

Description

function for plusMinus algorithm

Usage

```
plusMinus(X, y, lambda = NULL, tuningpar = "nfeatures", standardize = FALSE,

   directionality = "posneg", ties.method = "average", votingthresholdquantile = 0.5,

    modeltype = "plusminus")
```

Arguments

X	gene expression matrix
y	response variables
lambda	lambda
tuningpar	tuning parameter
standardize	standardize or not
directionality	directionality
ties.method	ties.method
votingthresholdquantile	votingthresholdquantile
modeltype	modeltype

Value

returns regression coefficients

Author(s)

Yuqing Zhang, Christoph Bernau, Levi Waldron

Examples

```
set.seed(8)

library(curatedOvarianData)

data( E.MTAB.386_eset )

eset <- E.MTAB.386_eset[1:100, 1:30]

rm(E.MTAB.386_eset)



X <- t(exprs(eset))



time <- eset$days_to_death

cens <- sample(0:1, 30, replace=TRUE)
```

```
y <- Surv(time, cens)



beta <- plusMinus(X, y)

beta
```

`rowCoxTests` *rowCoxTests*

Description

method for performing Cox regression

Usage

```
rowCoxTests(X, y, option = c("fast", "slow"), ...)
```

Arguments

`X`	Gene expression data. The following formats are available: matrix Rows correspond to observations, columns to variables. data.frame Rows correspond to observations, columns to variables. ExpressionSet rowCoxTests will extract the expressions using exprs().
`y`	Survival Response, an object of class: Surv if X is of type data.frame or matrix character if X is of type ExpressionSet. In this case y is the name of the survival response in the phenoData of X. If survival time and indicator are stored separately in the phenoData one can specify a two-element character vector the first element representing the survival time variable.
`option`	"fast" loops over rows in C, "slow" calls coxph directly in R. The latter method may be used if something goes wrong with the "fast" method.
`...`	currently unused

Value

dataframe with two columns: coef = Cox regression

coefficients, p.value =

Wald Test p-values. Rows correspond to the rows of X.

Author(s)

Yuqing Zhang, Christoph Bernau, Levi Waldron

Examples

```
#test

##regressor-matrix (gene expressions)

X<-matrix(rnorm(1e6),nrow=10000)

#seed

set.seed(123)

#times

time<-rnorm(n=ncol(X),mean=100)

#censoring(1->death)

status<-rbinom(n=ncol(X),size=1, prob=0.8)



##survival object

y<-Surv(time,status)



## Do 10,000 Cox regressions:

system.time(output <- rowCoxTests(X=X,y=y, option="fast"))
```

simBootstrap *simBootstrap*

Description

the driver function to perform three-step bootstrap resampling
to get independent genomic data sets

Usage

```
simBootstrap(obj, y.vars, n.samples, parstep, type = "two-steps",

    balance.variables = NULL, funSimData = simData, funTrueModel = getTrueModel,

    funSurvTime = simTime)
```

Arguments

obj	a list of ExpressionSet, matrix or RangedSummarizedExperiment
y.vars	a list of reponse variables, elements can be class Surv, matrix or data.frame
n.samples	number of samples to resample in each set
parstep	step number to fit CoxBoost
type	whether to include resampling set labels
balance.variables	covariate names to balance in the simulated sets
funSimData	function to perform non-parametric bootstrap
funTrueModel	function to construct true models in original sets
funSurvTime	function to perform parametric bootstrap

Value

a list of values including:
obj.list = a list of simulated objects the same type as input
indices.list = a list of indices indicating which sample the simulated sample is in the
original set
setsID = a vector to indicate the original ID of simulated sets, if
type=="original", setsID should be 1,2,3,...
lp.list = a list of true linear predictor of each original data sets
beta.list = a list of true coefficients used for simulating observations
survH.list = list of cumulative survival hazard
censH.list = list of cumulative censoring hazard
grid.list = list of timeline grid corresponding to survH and censH respectivley

Author(s)

Yuqing Zhang, Christoph Bernau, Levi Waldron

Examples

```
library(curatedOvarianData)

library(GenomicRanges)

data(E.MTAB.386_eset)

data(GSE14764_eset)

esets.list <- list(E.MTAB.386=E.MTAB.386_eset[1:200, 1:20], GSE14764=GSE14764_eset[1:200, 1:20])

rm(E.MTAB.386_eset, GSE14764_eset)



## simulate on multiple ExpressionSets

set.seed(8)



y.list <- lapply(esets.list, function(eset){

  time <- eset$days_to_death

  cens.chr <- eset$vital_status

  cens <- rep(0, length(cens.chr))

  cens[cens.chr=="living"] <- 1

  return(Surv(time, cens))

})
```

```
simmodels <- simBootstrap(obj=esets.list, y.vars=y.list, 10, 100)

simmodels$obj.list[[1]]



# balance covariates

simmodels <- simBootstrap(obj=esets.list, y.vars=y.list, 10, 100,

                          balance.variables="tumorstage")

rm(esets.list, simmodels)



## Support RangedSummarizedExperiment

nrows <- 200; ncols <- 10

counts <- matrix(runif(nrows * ncols, 1, 1e4), nrows)

rowRanges <- GRanges(rep(c("chr1", "chr2"), c(50, 150)),

                     IRanges(floor(runif(200, 1e5, 1e6)), width=100),

                     strand=sample(c("+", "-"), 200, TRUE))

colData <- DataFrame(Treatment=rep(c("ChIP", "Input"), 5),

                     row.names=LETTERS[1:10])

sset <- SummarizedExperiment(assays=SimpleList(counts=counts),

                             rowRanges=rowRanges, colData=colData)
```

```
s.list <- list(sset[,1:5], sset[,6:10])

time <- c(540, 527, 668, 587, 620, 540, 527, 668, 587, 620)

cens <- c(1, 0, 0, 1, 0, 1, 0, 0, 1, 0)

y.vars <- Surv(time, cens)

y.vars <- list(y.vars[1:5,],y.vars[1:5,])

simmodels <- simBootstrap(obj=s.list, y.vars=y.vars, 20, 100)
```

---

simData *simData*

Description

simData is a function to perform non-parametric bootstrap resampling
on a list of (original) data sets, both on set level and patient level,
in order to simulate independent genomic sets.

Usage

```
simData(obj, n.samples, y.vars = list(), type = "two-steps",

    balance.variables = NULL)
```

Arguments

obj	a list of ExpressionSets, matrices or RangedSummarizedExperiments. If elements are matrices, columns represent samples
n.samples	an integer indicating how many samples should be resampled from each set
y.vars	a list of response variables, can be Surv object, or matrix or data.frame with two columns
type	string "one-step" or "two-steps". If type="one-step", the function will skip resampling the datasets, and directly resample from the original list of obj

`balance.variables`
: balance.variables will be a vector of covariate names that should be balanced in the simulation. After balancing, the prevalence of covariate in each result set should be the same as the overall distribution across all original data sets. Default is set as NULL, when it will not balance over any covariate. if isn't NULL, esets parameter should only be of class ExpressionSet

Value

returns a list of simulated ExpressionSets, with names indicating its original set, and indices of the original patients.

prob.desired and prob.real are only useful when balance.varaibles is set.

prob.desired shows overall distrubition of the specified covariate. prob.list

shows the sampling probability in each set after balancing

Author(s)

Yuqing Zhang, Christoph Bernau, Levi Waldron

Examples

```
library(curatedOvarianData)

library(GenomicRanges)



data(E.MTAB.386_eset)

data(GSE14764_eset)

esets.list <- list(E.MTAB.386=E.MTAB.386_eset[1:100, 1:10], GSE14764=GSE14764_eset[1:100, 1:10])

rm(E.MTAB.386_eset, GSE14764_eset)



## simulate on multiple ExpressionSets

set.seed(8)
```

```
# one-step bootstrap: skip resampling set labels

simmodels <- simData(esets.list, 20, type="one-step")

# two-step-non-parametric bootstrap

simmodels <- simData(esets.list, 10, type="two-steps")



## simulate one set

simmodels <- simData(list(esets.list[[1]]), 10, type="two-steps")



## balancing covariates

# single covariate

simmodels <- simData(list(esets.list[[1]]), 5, balance.variables="tumorstage")



# multiple covariates

simmodels <- simData(list(esets.list[[1]]), 5,

               balance.variables=c("tumorstage", "age_at_initial_pathologic_diagnosis"))



## Support matrices

X.list <- lapply(esets.list, function(eset){

  return(exprs(eset))
```

```
})

simmodels <- simData(X.list, 20, type="two-steps")



## Support RangedSummarizedExperiment

nrows <- 200; ncols <- 6

counts <- matrix(runif(nrows * ncols, 1, 1e4), nrows)

rowRanges <- GRanges(rep(c("chr1", "chr2"), c(50, 150)),

                     IRanges(floor(runif(200, 1e5, 1e6)), width=100),

                     strand=sample(c("+", "-"), 200, TRUE))

colData <- DataFrame(Treatment=rep(c("ChIP", "Input"), 3),

                     row.names=LETTERS[1:6])

sset <- SummarizedExperiment(assays=SimpleList(counts=counts),

                             rowRanges=rowRanges, colData=colData)



s.list <- list(sset[,1:3], sset[,4:6])

simmodels <- simData(s.list, 20, type="two-steps")
```

---

simTime *simTime*

Description

simTime is a function to perform the parametric-bootstrap step, where we use the true coefficients and cumulative hazard to simulate survival and censoring.

Usage

```
simTime(simmodels, original.yvars, result)
```

Arguments

simmodels	a list in the form of the return value of simData() which consists of three lists: obj: a list of ExpressionSets, matrices or RangedSummarizedExperiments setsID: a list of set labels indicating which original set the simulated one is from indices: a list of patient labels to tell which patient in the original set is drawn
original.yvars	response variable in the order of original sets(without sampling)
result	a list in the form of return of getTrueModel() which consists of five lists: Beta: a list of coefficients obtained by grid: timeline grid corresponding to hazard estimations censH and survH survH: cumulative hazard for survival times distribution censH: cumulative hazard for censoring times distribution lp: true linear predictors

Value

survival time is saved in phenodata, here the function still returns the ExpressionSets

Author(s)

Yuqing Zhang, Christoph Bernau, Levi Waldron

Examples

```
library(curatedOvarianData)

data(E.MTAB.386_eset)

data(GSE14764_eset)

esets.list <- list(E.MTAB.386=E.MTAB.386_eset[1:100, 1:20], GSE14764=GSE14764_eset[1:100, 1:20])

rm(E.MTAB.386_eset, GSE14764_eset)



## simulate on multiple ExpressionSets

set.seed(8)
```

```
y.list <- lapply(esets.list, function(eset){
  time <- eset$days_to_death
  cens.chr <- eset$vital_status
  cens <- rep(0, length(cens.chr))
  cens[cens.chr=="living"] <- 1
  return(Surv(time, cens))
})

# To perform both parametric and non-parametric bootstrap, you can call simBootstrap()
# or, you can divide the steps into:
res <- getTrueModel(esets.list, y.list, 100)
simmodels <- simData(obj=esets.list, y.vars=y.list, n.samples=10)

# Then, use this function
simmodels <- simTime(simmodels=simmodels, original.yvars=y.list, result=res)

# it also supports performing only the parametrc bootstrap step on a list of expressionsets
# but you need to construct the parameter by scratch
```

```
res <- getTrueModel(esets.list, y.list, 100)

setsID <- seq_along(esets.list)

indices <- list()

for(i in setsID){

  indices[[i]] <- seq_along(sampleNames(esets.list[[i]]))

}

simmodels <- list(obj=esets.list, y.vars=y.list, indices=indices, setsID=setsID)



new.simmodels <- simTime(simmodels=simmodels, original.yvars=y.list, result=res)
```

---

zmatrix *zmatrix*

Description

generate a matrix of c statistics

Usage

```
zmatrix(obj, y.vars, fold, trainingFun = masomenos, cvFun = funCV,

    cvSubsetFun = cvSubsets, covar = NULL)
```

Arguments

obj	a list of ExpressionSet, matrix or RangedSummarizedExperiment objects. If its elements are matrices, columns represent samples
y.vars	a list of response variables, all the response variables shold be matrix, data.frame(with 2 columns) or Surv object
fold	cvFun parameter, in this case passes to funCV()
trainingFun	training function

cvFun	function to perform cross study within one set
cvSubsetFun	function to divide the expression sets into subsets for cross validation
covar	other covariates to be added as predictors

Value

outputs one matrix of validation statistics

Author(s)

Yuqing Zhang, Christoph Bernau, Levi Waldron

Examples

```
library(curatedOvarianData)

library(GenomicRanges)

data(E.MTAB.386_eset)

data(GSE14764_eset)

esets.list <- list(E.MTAB.386=E.MTAB.386_eset[1:100, 1:30], GSE14764=GSE14764_eset[1:100, 1:30])

rm(E.MTAB.386_eset, GSE14764_eset)



## simulate on multiple ExpressionSets

set.seed(8)



y.list <- lapply(esets.list, function(eset){

  time <- eset$days_to_death

  cens.chr <- eset$vital_status

  cens <- rep(0, length(cens.chr))
```

```
  cens[cens.chr=="living"] <- 1

  return(Surv(time, cens))

})



# generate on original ExpressionSets

z <- zmatrix(esets.list, y.list, 3)



# generate on simulated ExpressionSets

simmodels <- simBootstrap(esets.list, y.list, 100, 100)

z <- zmatrix(simmodels$obj.list, simmodels$y.vars.list, 3)



# support matrix

X.list <- lapply(esets.list, function(eset){

  return(exprs(eset)) ### columns represent samples !!

})

z <- zmatrix(X.list, y.list, 3)



# support RangedSummarizedExperiment

nrows <- 200; ncols <- 6
```

```
counts <- matrix(runif(nrows * ncols, 1, 1e4), nrows)

rowRanges <- GRanges(rep(c("chr1", "chr2"), c(50, 150)),

                     IRanges(floor(runif(200, 1e5, 1e6)), width=100),

                     strand=sample(c("+", "-"), 200, TRUE))

colData <- DataFrame(Treatment=rep(c("ChIP", "Input"), 3),

                     row.names=LETTERS[1:6])

sset <- SummarizedExperiment(assays=SimpleList(counts=counts),

                             rowRanges=rowRanges, colData=colData)



time <- sample(4500:4700, 6, replace=TRUE)

cens <- sample(0:1, 6, replace=TRUE)

y.vars <- Surv(time, cens)



z <- zmatrix(list(sset[,1:3], sset[,4:6]), list(y.vars[1:3,],y.vars[4:6,]), 3)
```

Index