Package 'trena'

October 6, 2019

Type Package

Title Fit transcriptional regulatory networks using gene expression, priors, machine learning

Version 1.6.1

Date 2019-06-05

Author Seth Ament <seth.ament@systemsbiology.org>, Paul Shannon <pshannon@systemsbiologyg.org>, Matthew Richards <mrichard@systemsbiology.org>

Maintainer Paul Shannon <paul.thurmond.shannon@gmail.com>

Imports RSQLite, RMySQL, lassopv, randomForest, flare, vbsr, BiocParallel, RPostgreSQL, methods, DBI, BSgenome, BSgenome.Hsapiens.UCSC.hg38, BSgenome.Hsapiens.UCSC.hg19, BSgenome.Mmusculus.UCSC.mm10, SNPlocs.Hsapiens.dbSNP150.GRCh38, org.Hs.eg.db, Biostrings, GenomicRanges, biomaRt, AnnotationDbi

Depends R (>= 3.4.0), utils, glmnet (>= 2.0.3), MotifDb (>= 1.19.17)

Suggests RUnit, plyr, knitr, BiocGenerics, rmarkdown

VignetteBuilder knitr

Description Methods for reconstructing transcriptional regulatory networks, especially in species for which genome-wide TF binding site information is available.

License GPL-3

biocViews Transcription, GeneRegulation, NetworkInference, FeatureExtraction, Regression, SystemsBiology, GeneExpression

Collate 'Solver.R' 'BayesSpikeSolver.R' 'CandidateFilter.R' 'EnsembleSolver.R' 'FootprintFinder.R' 'FootprintFilter.R' 'GeneOntologyFilter.R' 'HumanDHSFilter.R' 'LassoPVSolver.R' 'LassoSolver.R' 'MotifMatcher.R' 'PCAMax.R' 'PearsonSolver.R' 'RandomForestSolver.R' 'RidgeSolver.R' 'SpearmanSolver.R' 'SqrtLassoSolver.R' 'Trena.R' 'VarianceFilter.R' 'help.R' 'sharedFunctions.R' 'utils.R'

Encoding UTF-8

RoxygenNote 6.1.1

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

git_branch RELEASE_3_9

git_last_commit dd13773

git_last_commit_date 2019-06-05

Date/Publication 2019-10-05

R topics documented:

`trena-package` *Inferring Transcriptional Regulation with TReNA*

Description

'trena' provides a framework for using gene expression data to infer relationships between a target gene and a set of transcription factors. It does so using a several classes and their associated methods, briefly documented below

Details

#' Solver Class Objects

The Solver class is a base class used within 'trena'. A particular Solver object also contains the name of the selected solver and dispatches the correct feature selection method when run is called on the object. It is inherited by all the following subclasses, representing the different feature selection methods: BayesSpikeSolver, EnsembleSolver, LassoPVSolver, LassoSolver, PearsonSolver, RandomForestSolver, RidgeSolver, SpearmanSolver, SqrtLassoSolver.

CandidateFilter Class Objects

The CandidateFilter class is a base class that is generally used to filter the transcription factors in the expression matrix to obtain a set of candidate regulators. Filtering method depends on the filter type chosen; there are currently the following subclasses: FootprintFilter, HumanDHSFilter, GeneOntologyFilter, and VarianceFilter. The filters are applied using the getCandidates method on a given CandidateFilter object.

FootprintFinder Class Objects

The FootprintFinder class is designed to allow extraction of gene footprinting information from existing PostgreSQL or SQLite databases. In standard use of the 'trena' package, it is used solely by the getCandidates method for a FootprintFilter object. However, a FootprintFinder object has many more available methods that allow it to extract information more flexibly.

---

addStats,PCAMax-method

add PCA-based summary stats on all TFs in the model

Description

add PCA-based summary stats on all TFs in the model

Usage

```
## S4 method for signature 'PCAMax'
addStats(obj, varianceToInclude = 0.75,
  scalePCA = FALSE)
```

Arguments

obj	An object of the class PCAMax
varianceToInclude	numeric variance to include in the PCA
scalePCA	logical

Value

the original model with extra columns: pcaMax, cov, PC1, PC2,

addStatsSimple,PCAMax-method

add PCA-based summary stats on all TFs in the model

Description

add PCA-based summary stats on all TFs in the model

Usage

```
## S4 method for signature 'PCAMax'
addStatsSimple(obj, varianceToInclude = 0.75,
  scalePCA = FALSE, excludeLasso = TRUE, quiet = TRUE)
```

Arguments

obj	An object of the class PCAMax
varianceToInclude	numeric variance to include in the PCA
scalePCA	logical
excludeLasso	logical excluding lasso avoids dropping TFs due to shrinkage
quiet	logical

Value

the original model with extra columns: pcaMax, cov, PC1, PC2,

assessSnp,Trena-method

Assess the effect of a SNP using a Trena object

Description

Assess the effect of a SNP using a Trena object

Usage

```
## S4 method for signature 'Trena'
assessSnp(obj, pfms, variant, shoulder,
  pwmMatchMinimumAsPercentage, relaxedMatchDelta = 25)
```

Arguments

obj	An object of class Trena
pfms	A set of motif matrices, generally retrieved using MotifDb
variant	A variant of interest
shoulder	A distance from the TSS to use as a window
pwmMatchMinimumAsPercentage	A minimum match percentage for the motifs
relaxedMatchDelta	A numeric indicating the degree of the match (default = 25)

Value

A data frame containing the gene model

Examples

```
## Not run:
# Create a Trena object for human, assign a variant, then assess the effects of the variant
trena <- Trena("hg38")

library(MotifDb)
jaspar.human.pfms <- as.list(query(query(MotifDb, "jaspar2016"), "sapiens"))[21:25]

variant <- "rs3875089" # chr18:26865469  T->C

tbl <- assessSnp(trena, jaspar.human.pfms, variant, shoulder = 3,
pwmMatchMinimumAsPercentage = 65)

## End(Not run)
```

BayesSpikeSolver *Create a Solver class object using the Bayes Spike Solver*

Description

Create a Solver class object using the Bayes Spike Solver

Usage

```
BayesSpikeSolver(mtx.assay = matrix(), targetGene, candidateRegulators,
  nOrderings = 10, quiet = TRUE)
```

Arguments

mtx.assay	An assay matrix of gene expression data
targetGene	A designated target gene that should be part of the mtx.assay data
candidateRegulators	The designated set of transcription factors that could be associated with the target gene
nOrderings	An integer denoting the number of random starts to use for the Bayes Spike method (default = 10)
quiet	A logical denoting whether or not the solver should print output

Value

A Solver class object with Bayes Spike as the solver

See Also

solve.BayesSpike, getAssayData

Other Solver class objects: EnsembleSolver, HumanDHSFilter-class, LassoPVSolver, LassoSolver, PearsonSolver, RandomForestSolver, RidgeSolver, Solver-class, SpearmanSolver, SqrtLassoSolver

Examples

```
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
target.gene <- "MEF2C"
tfs <- setdiff(rownames(mtx.sub), target.gene)
bayes.solver <- BayesSpikeSolver(mtx.sub, target.gene, tfs)
```

BayesSpikeSolver-class

An S4 class to represent a Bayes Spike solver

Description

An S4 class to represent a Bayes Spike solver

CandidateFilter-class *CandidateFilter*

Description

A CandidateFilter is an S4 class to represent a gene candidate filter. These filters can employ a variety of methods to reduce the number of transcription factors used as predictors for solving a Solver object.

Usage

```
CandidateFilter(quiet = TRUE)
```

Arguments

quiet	A logical denoting whether or not the CandidateFilter object should print output

Value

An object of the Candidate filter class

Slots

quiet A logical denoting whether or not the CandidateFilter object should print output

See Also

getCandidates

Examples

```
# Create an empty candidate filter
candidate.filter <- CandidateFilter(quiet=TRUE)
```

`closeDatabaseConnections,FootprintFinder-method`
Close a Footprint Database Connection

Description

This method takes a FootprintFinder object and closes connections to the footprint databases if they are currently open.

Usage

```
## S4 method for signature 'FootprintFinder'
closeDatabaseConnections(obj)
```

Arguments

`obj`	An object of class FootprintFinder

Value

Closes the specified database connection

See Also

Other FootprintFinder methods: `FootprintFinder-class`, `getChromLoc,FootprintFinder-method`, `getFootprintsForGene,FootprintFinder-method`, `getFootprintsInRegion,FootprintFinder-method`, `getGenePromoterRegion,FootprintFinder-method`, `getGtfGeneBioTypes,FootprintFinder-method`, `getGtfMoleculeTypes,FootprintFinder-method`, `getPromoterRegionsAllGenes,FootprintFinder-method`

`createGeneModelFromRegulatoryRegions,Trena-method`
Create a model for a target gene using a Trena object

Description

Create a model for a target gene using a Trena object

Usage

```
## S4 method for signature 'Trena'
createGeneModelFromRegulatoryRegions(obj, targetGene,
  solverNames, tbl.regulatoryRegions, mtx)
```

Arguments

`obj`	An object of class Trena
`targetGene`	The name of a target gene to use for building a model
`solverNames`	A character vector containing the solver names to be used for building the model
`tbl.regulatoryRegions`	A data frame of regulatory regions, typically generated by using a filter
`mtx`	An assay matrix of expression data

Value

A data frame containing the gene model

Examples

```
if(interactive()){  # takes too long for the bioconductor build
  # Create a Trena object for human and make a gene model for "MEF2C" using a footprint filter
   trena <- Trena("hg38")
   chromosome <- "chr5"
   mef2c.tss <- 88904257
   loc.start <- mef2c.tss - 1000
   loc.end   <- mef2c.tss + 1000

  database.filename <- system.file(package="trena", "extdata", "mef2c.neigborhood.hg38.footprints.db")
   database.uri <- sprintf("sqlite://%s", database.filename)
   sources <- c(database.uri)
  load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))

  motifs.list <- getRegulatoryChromosomalRegions(trena, chromosome, mef2c.tss-1000, mef2c.tss+1000,
   sources, "MEF2C", mef2c.tss)

   library(MotifDb)
  tbl.motifs.tfs <- associateTranscriptionFactors(MotifDb, motifs.list[[1]], source="MotifDb", expand.rows=T
  model.mef2c <- createGeneModelFromRegulatoryRegions(trena, "MEF2C", c("lasso","ridge","randomforest"),
                                                      tbl.motifs.tfs, mtx.sub)
   } # if interactive
```

---

createGeneModelFromTfList,Trena-method

Create a model for a target gene using a Trena object

Description

Create a model for a target gene using a Trena object

Usage

```
## S4 method for signature 'Trena'
createGeneModelFromTfList(obj, targetGene, solverNames,
  tfList, mtx)
```

Arguments

obj	An object of class Trena
targetGene	The name of a target gene to use for building a model
solverNames	A character vector containing the solver names to be used for building the model
tfList	A character list, often the gene symbols for known transcription factors
mtx	An assay matrix of expression data

Value

A data frame containing the gene model

Examples

```
if(interactive()){  # takes too long for the bioconductor build
  # Create a Trena object for human and make a gene model for "MEF2C" using a footprint filter
   trena <- Trena("hg38")
   chromosome <- "chr5"
   mef2c.tss <- 88904257
   loc.start <- mef2c.tss - 1000
   loc.end   <- mef2c.tss + 1000

  database.filename <- system.file(package="trena", "extdata", "mef2c.neigborhood.hg38.footprints.db")
   database.uri <- sprintf("sqlite://%s", database.filename)
   sources <- c(database.uri)
  load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))

  model.mef2c <- createGeneModelFromTfList(trena, "MEF2C", c("lasso","ridge","randomforest"),
                                           tfList=c())
   } # if interactive
```

`elasticNetSolver` *Run the Elastic Net Solvers*

Description

Given a TReNA object with either LASSO or Ridge Regression as the solver, use the glmnet function to estimate coefficients for each transcription factor as a predictor of the target gene's expression level.

Usage

```
elasticNetSolver(obj, target.gene, tfs, tf.weights, alpha, lambda,
  keep.metrics)
```

Arguments

`obj`	An object of class Solver
`target.gene`	A designated target gene that should be part of the mtx.assay data
`tfs`	The designated set of transcription factors that could be associated with the target gene.
`tf.weights`	A set of weights on the transcription factors (default = rep(1, length(tfs)))
`alpha`	The LASSO/Ridge tuning parameter
`lambda`	The penalty tuning parameter for elastic net
`keep.metrics`	A binary variable indicating whether or not to keep metrics

Value

A data frame containing the coefficients relating the target gene to each transcription factor, plus other fit parameters

See Also

glmnet

EnsembleSolver *Create a Solver class object using an ensemble of solvers*

Description

Create a Solver class object using an ensemble of solvers

Usage

```
EnsembleSolver(mtx.assay = matrix(), targetGene, candidateRegulators,
  solverNames = c("lasso", "lassopv", "pearson", "randomForest", "ridge",
  "spearman"), geneCutoff = 0.1, alpha.lasso = 0.9, alpha.ridge = 0,
  lambda.lasso = numeric(0), lambda.ridge = numeric(0),
  lambda.sqrt = numeric(0), nCores.sqrt = 4, nOrderings.bayes = 10,
  quiet = TRUE)
```

Arguments

mtx.assay	An assay matrix of gene expression data
targetGene	A designated target gene that should be part of the mtx.assay data
candidateRegulators	The designated set of transcription factors that could be associated with the target gene
solverNames	A character vector of strings denoting
geneCutoff	A fraction (0-1) of the supplied candidate regulators to be included in the fetaures output by the solver (default = 0.1)
alpha.lasso	A fraction (0-1) denoting the LASSO-Ridge balance of the 'glmnet' solver used by the LASSO method (default = 0.9)
alpha.ridge	A fraction (0-1) denoting the LASSO-Ridge balance of the 'glmnet' solver used by the Ridge method (default = 0)
lambda.lasso	The penalty parameter for LASSO, used to determine how strictly to penalize the regression coefficients. If none is supplied, this will be determined via permutation testing (default = NULL).
lambda.ridge	The penalty parameter for Ridge, used to determine how strictly to penalize the regression coefficients. If none is supplied, this will be determined via permutation testing (default = NULL).
lambda.sqrt	The penalty parameter for square root LASSO, used to determine how strictly to penalize the regression coefficients. If none is supplied, this will be determined via permutation testing (default = NULL).
nCores.sqrt	An integer denoting the number of computational cores to devote to the square root LASSO solver, which is the slowest of the solvers (default = 4)
nOrderings.bayes	An integer denoting the number of random starts to use for the Bayes Spike method (default = 10)
quiet	A logical denoting whether or not the solver should print output

Value

A Solver class object with Ensemble as the solver

See Also

solve.Ensemble, getAssayData

Other Solver class objects: BayesSpikeSolver, HumanDHSFilter-class, LassoPVSolver, LassoSolver, PearsonSolver, RandomForestSolver, RidgeSolver, Solver-class, SpearmanSolver, SqrtLassoSolver

Examples

```
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
target.gene <- "MEF2C"
tfs <- setdiff(rownames(mtx.sub), target.gene)
ensemble.solver <- EnsembleSolver(mtx.sub, target.gene, tfs)
```

EnsembleSolver-class *Class EnsembleSolver*

Description

Class EnsembleSolver

---

findMatchesByChromosomalRegion,MotifMatcher-method *Find Motif Matches by Chromosomal Region*

Description

Given a MotifMatcher object, a table of chromosomal regions, and a minimum match percentage, pull out a list containing a data frame of the motifs in those regions and a character vector of their associated transcription factors.

Usage

```
## S4 method for signature 'MotifMatcher'
findMatchesByChromosomalRegion(obj, tbl.regions,
  pwmMatchMinimumAsPercentage, variants = NA_character_)
```

Arguments

obj	An object of class MotifMatcher
tbl.regions	A data frame where each row contains a chromosomal region with the fields "chrom", "start", and "end".
pwmMatchMinimumAsPercentage	A percentage (0-100) used as a cutoff for what constitutes a motif match
variants	A character containing variants to use for the matching (default = NA_character_). The variants should either have the same number of entries as rows in the tbl.regions, or they should not be supplied.

Value

A list containing a data frame of the motifs in the given regions and a character vector of their associated transcription factors

Examples

```
## Not run:
# Perform a simple match in the rs13384219 neighborhood
library(MotifDb)
motifMatcher <- MotifMatcher(genomeName="hg38",
pfms = as.list(query(query(MotifDb, "sapiens"),"jaspar2016")), quiet=TRUE)
tbl.regions <- data.frame(chrom="chr2", start=57907313, end=57907333, stringsAsFactors=FALSE)
x <- findMatchesByChromosomalRegion(motifMatcher, tbl.regions, pwmMatchMinimumAsPercentage=92)

# Perform the same match, but now include a variant
x.mut <- findMatchesByChromosomalRegion(motifMatcher, tbl.regions,
pwmMatchMinimumAsPercentage=92, variants = "rs13384219")

## End(Not run)
```

FootprintFilter-class *Create a FootprintFilter object*

Description

A FootprintFilter object allows for filtering based on gene footprinting databases. Using its associated `getCandidates` method and URIs for both a genome database and project database, a FootprintFilter object can be used to filter a list of possible transcription factors to those that match footprint motifs for a supplied target gene.

Usage

```
FootprintFilter(genomeDB, footprintDB, regions = data.frame(),
  quiet = TRUE)
```

Arguments

`genomeDB`	A connection to a database that contains genome information
`footprintDB`	A connection to a database that contains footprint information
`regions`	A data frame that specifies the regions of interest (default = data.frame())
`quiet`	A logical denoting whether or not the filter should print output

Value

An object of the FootprintFilter class

See Also

`getCandidates-FootprintFilter`

Other Filtering Objects: `VarianceFilter-class`

Examples

```
## Not run:
  load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
   db.address <- system.file(package="trena", "extdata")
  genome.db.uri <- paste("sqlite:/",db.address,"mef2c.neighborhood.hg38.gtfAnnotation.db", sep = "/")
  project.db.uri <- paste("sqlite:/",db.address,"mef2c.neigborhood.hg38.footprints.db", sep = "/")
   target.gene <- "MEF2C"
   size.upstream <- 1000
   size.downstream <- 1000

      # Construct a Trena object and use it to retrive the regions
   trena <- Trena("hg38")
   regions <- getProximalPromoter(trena,target.gene, size.upstream, size.downstream)

  footprint.filter <- FootprintFilter(genomeDB = genome.db.uri, footprintDB = project.db.uri,
                                      regions = regions)

## End(Not run)
```

`FootprintFinder-class` *Class FootprintFinder*

Description

The FootprintFinder class is designed to query 2 supplied footprint databases (a genome database and a project database) for supplied genes or regions. Within the TReNA package, the FootprintFinder class is mainly used by the FootprintFilter class, but the FootprintFinder class offers more flexibility in constructing queries.

Usage

```
FootprintFinder(genome.database.uri, project.database.uri, quiet = TRUE)
```

Arguments

`genome.database.uri`	The address of a genome database for use in filtering. This database must contain the tables "gtf" and "motifsgenes" at a minimum. The URI format is as follows: "dbtype://host/database" (e.g. "postgres://localhost/genomedb")
`project.database.uri`	The address of a project database for use in filtering. This database must contain the tables "regions" and "hits" at a minimum. The URI format is as follows: "dbtype://host/database" (e.g. "postgres://localhost/projectdb")
`quiet`	A logical denoting whether or not the FootprintFinder object should print output

Value

An object of the FootprintFinder class

Slots

genome.db The address of a genome database for use in filtering

project.db The address of a project database for use in filtering

quiet A logical argument denoting whether the FootprintFinder object should behave quietly

See Also

FootprintFilter

Other FootprintFinder methods: closeDatabaseConnections,FootprintFinder-method, getChromLoc,FootprintF
getFootprintsForGene,FootprintFinder-method, getFootprintsInRegion,FootprintFinder-method,
getGenePromoterRegion,FootprintFinder-method, getGtfGeneBioTypes,FootprintFinder-method,
getGtfMoleculeTypes,FootprintFinder-method, getPromoterRegionsAllGenes,FootprintFinder-method

---

GeneOntologyFilter-class

Create a GeneOntologyFilter object

Description

A GeneOntologyFilter object allows for filtering based on gene ontology (GO) terms. Its associated getCandidates method uses an organism database and a GO term (e.g. GO:#######) to filter a list of possible regulators factors to those that match the GO term.

Usage

```
GeneOntologyFilter(organismDatabase = org.Hs.eg.db::org.Hs.eg.db,
  GOTerm = "GO:0006351", quiet = TRUE)
```

Arguments

organismDatabase	An organism-specific database of type 'OrgDb'
GOTerm	A character matching an accepted gene ontology term. The default term corresponds to "transcription, DNA-templated". (default="GO:0006351")
quiet	A logical denoting whether or not the filter should print output

Value

A GeneOntologyFilter object

See Also

CandidateFilter

Examples

```
# Make a filter for "transcription, DNA-templated"
library(org.Hs.eg.db)
goFilter <- GeneOntologyFilter(org.Hs.eg.db, GOTerm="GO:0006351")
```

getAssayData,Solver-method

Retrieve the assay matrix of gene expression data from a Solver object

Description

Retrieve the assay matrix of gene expression data from a Solver object

Usage

```
## S4 method for signature 'Solver'
getAssayData(obj)
```

Arguments

obj	An object of class Solver

Value

The assay matrix of gene expression data associated with a Solver object

Examples

```
# Create a Solver object using the included Alzheimer's data and retrieve the matrix
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
targetGene <- "MEF2C"
candidateRegulators <- setdiff(rownames(mtx.sub), targetGene)
solver <- Solver(mtx.sub, targetGene, candidateRegulators)
mtx <- getAssayData(solver)
```

getAvailableSolvers *Get the available solvers for use in trena*

Description

Retrieve the vector of different methods that can be used as solvers in trena. Solver names in the returned vector correspond to the exact capitalization used in their Solver subclasses (i.e. a Solver object using LASSO is LassoSolver)

Usage

```
getAvailableSolvers()
```

Value

A character vector of all solvers currently available in trena.

Examples

```
all.solvers <- getAvailableSolvers()
```

getCandidates *Get candidate genes using a CandidateFilter object*

Description

Get candidate genes using a CandidateFilter object

Usage

```
getCandidates(obj)
```

Arguments

obj	An object of a CandidateFilter class

Value

A vector containing genes from the assay matrix that are selected by the filter

See Also

Other getCandidate Methods: getCandidates,FootprintFilter-method, getCandidates,GeneOntologyFilter-me
getCandidates,HumanDHSFilter-method, getCandidates,VarianceFilter-method

getCandidates,FootprintFilter-method *Get candidate genes using the footprint filter*

Description

Get candidate genes using the footprint filter

Usage

```
## S4 method for signature 'FootprintFilter'
getCandidates(obj)
```

Arguments

obj	An object of class FootprintFilter

Value

A list, where one element is the transcription factors found in the footprints and the other is a data frame containing all the meta data for the footprints

See Also

FootprintFilter

Other getCandidate Methods: getCandidates,GeneOntologyFilter-method, getCandidates,HumanDHSFilter-met
getCandidates,VarianceFilter-method, getCandidates

Examples

```
# Use footprint filter with the included SQLite database for MEF2C to filter candidates
# in the included Alzheimer's dataset, using the Trena object to get regions
## Not run:
   target.gene <- "MEF2C"
   db.address <- system.file(package="trena", "extdata")
  genome.db.uri <- paste("sqlite:/",db.address,"mef2c.neighborhood.hg38.gtfAnnotation.db", sep = "/")
  project.db.uri <- paste("sqlite:/",db.address,"mef2c.neigborhood.hg38.footprints.db", sep = "/")
   size.upstream <- 1000
   size.downstream <- 1000
      # Construct a Trena object and use it to retrive the regions
   trena <- Trena("hg38")
   regions <- getProximalPromoter(trena,target.gene, size.upstream, size.downstream)
  footprint.filter <- FootprintFilter(genomeDB = genome.db.uri, footprintDB = project.db.uri,
                                      regions = regions)
   footprints <- getCandidates(footprint.filter)

## End(Not run)
```

---

getCandidates,GeneOntologyFilter-method

Get candidate genes using a gene ontology filter

Description

Get candidate genes using a gene ontology filter

Usage

```
## S4 method for signature 'GeneOntologyFilter'
getCandidates(obj)
```

Arguments

obj	An object of class GeneOntologyFilter

Value

A list, where one element a character vector of transcription factors that match the GO term and the other is an empty data frame.

See Also

GeneOntologyFilter

Other getCandidate Methods: getCandidates,FootprintFilter-method, getCandidates,HumanDHSFilter-method
getCandidates,VarianceFilter-method, getCandidates

Examples

```
# Make a filter for "transcription, DNA-templated" and use it to filter candidates
library(org.Hs.eg.db)
goFilter <- GeneOntologyFilter(org.Hs.eg.db, GOTerm="GO:0006351")
candidates <- getCandidates(goFilter)
```

`getCandidates,HumanDHSFilter-method`

Get candidate genes using a human DHS filter

Description

Get candidate genes using a human DHS filter

Usage

```
## S4 method for signature 'HumanDHSFilter'
getCandidates(obj)
```

Arguments

`obj`	An object of class FootprintFilter

Value

A list, where one element a character vector of transcription factors that match the GO term and the other is an empty data frame.

See Also

`FootprintFilter`

Other getCandidate Methods: `getCandidates,FootprintFilter-method`, `getCandidates,GeneOntologyFilter-me`
`getCandidates,VarianceFilter-method`, `getCandidates`

Examples

```
# Make a filter for "transcription, DNA-templated" and use it to filter candidates
## Not run:
#' load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
targetGene <- "VRK2"
promoter.length <- 1000
genomeName <- "hg38"
db.address <- system.file(package="trena", "extdata")
genome.db.uri   <- paste("sqlite:/", db.address, "vrk2.neighborhood.hg38.gtfAnnotation.db",  sep = "/")
jaspar.human <- as.list(query(query(MotifDb, "sapiens"),"jaspar2016"))

# Grab regions for VRK2 using shoulder size of 1000
trena <- Trena(genomeName)
tbl.regions <- getProximalPromoter(trena, "VRK2", 1000, 1000)

hd.filter <- HumanDHSFilter(genomeName, pwmMatchPercentageThreshold = 85,
```

```
geneInfoDatabase.uri = genome.db.uri, regions = tbl.regions, pfms = jaspar.human)

getCandidates(hd.filter)

## End(Not run)
```

---

getCandidates,VarianceFilter-method
Get candidate genes using the variance filter

Description

Get candidate genes using the variance filter

Usage

```
## S4 method for signature 'VarianceFilter'
getCandidates(obj)
```

Arguments

obj	An object of class VarianceFilter

Value

A vector containing all genes with variances less than the target gene

See Also

VarianceFilter

Other getCandidate Methods: getCandidates,FootprintFilter-method, getCandidates,GeneOntologyFilter-me
getCandidates,HumanDHSFilter-method, getCandidates

Examples

```
# Using the included Alzheimer's dataset, filter out only those transcription factors with variance
# within 50% of the variance of MEF2C
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
variance.filter <- VarianceFilter(mtx.assay = mtx.sub, targetGene = "MEF2C")
tfs <- getCandidates(variance.filter)
```

getChromLoc,FootprintFinder-method
Get Chromosome Location

Description

Using the gtf table in the genome database contained in a FootprintFinder object, get the locations of chromosomes with the specified gene name, biological unit type, and molecule type

Usage

```
## S4 method for signature 'FootprintFinder'
getChromLoc(obj, name,
  biotype = "protein_coding", moleculetype = "gene")
```

Arguments

obj	An object of class FootprintFinder
name	A gene name or ID
biotype	A type of biological unit (default="protein_coding")
moleculetype	A type of molecule (default="gene")

Value

A dataframe containing the results of a database query pertaining to the specified name, biotype, and molecule type. This dataframe contains the following columns: gene_id, gene_name, chr, start, endpos, strand

See Also

Other FootprintFinder methods: FootprintFinder-class, closeDatabaseConnections,FootprintFinder-method, getFootprintsForGene,FootprintFinder-method, getFootprintsInRegion,FootprintFinder-method, getGenePromoterRegion,FootprintFinder-method, getGtfGeneBioTypes,FootprintFinder-method, getGtfMoleculeTypes,FootprintFinder-method, getPromoterRegionsAllGenes,FootprintFinder-method

Examples

```
db.address <- system.file(package="trena", "extdata")
genome.db.uri <- paste("sqlite:/",db.address,"mef2c.neighborhood.hg38.gtfAnnotation.db", sep = "/")
project.db.uri <- paste("sqlite:/",db.address,"mef2c.neigborhood.hg38.footprints.db", sep = "/")
fp <- FootprintFinder(genome.db.uri, project.db.uri)

chrom.locs <- getChromLoc(fp, name = "MEF2C")
```

`getCoverage,PCAMax-method`
what percentage of the variance is captured in the first two principal components?

Description

what percentage of the variance is captured in the first two principal components?

Usage

```
## S4 method for signature 'PCAMax'
getCoverage(obj)
```

Arguments

`obj`	An object of the class PCAMax

Value

a number between zero and one

`getEncodeRegulatoryTableNames,HumanDHSFilter-method`
Get Encode regulatory tables using a human DHS filter

Description

Get Encode regulatory tables using a human DHS filter

Usage

```
## S4 method for signature 'HumanDHSFilter'
getEncodeRegulatoryTableNames(obj)
```

Arguments

`obj`	An object of class HumanDHSFilter

Value

A character vector containing the names of the Encode regulatory tables for the regions contained in the HumanDHSFilter object

See Also

`HumanDHSFilter`

Examples

```
## Not run:
  load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
   targetGene <- "VRK2"
   promoter.length <- 1000
   genomeName <- "hg38"
   db.address <- system.file(package="trena", "extdata")
  genome.db.uri   <- paste("sqlite:/", db.address, "vrk2.neighborhood.hg38.gtfAnnotation.db",  sep = "/")
   jaspar.human <- as.list(query(query(MotifDb, "sapiens"),"jaspar2016"))
      # Grab regions for VRK2 using shoulder size of 1000
   trena <- Trena(genomeName)
   tbl.regions <- getProximalPromoter(trena, "VRK2", 1000, 1000)
   hd.filter <- HumanDHSFilter(genomeName, pwmMatchPercentageThreshold = 85,
   geneInfoDatabase.uri = genome.db.uri, regions = tbl.regions, pfms = jaspar.human)
   getEncodeRegulatoryTableNames(hd.filter)

## End(Not run)
```

---

getFootprintsForGene,FootprintFinder-method

Get Footprints for Gene

Description

Using the getGenePromoterRegion and getFootprintsInRegion functions in conjunction with the gtf table inside the genome database specified by the FootprintFinder object, retrieve a dataframe containing the footprints for a specified gene

Usage

```
## S4 method for signature 'FootprintFinder'
getFootprintsForGene(obj, gene,
  size.upstream = 1000, size.downstream = 0,
  biotype = "protein_coding", moleculetype = "gene")
```

Arguments

obj	An object of class FootprintFinder
gene	A gene name of ID
size.upstream	An integer denoting the distance upstream of the target gene to look for footprints (default = 1000)
size.downstream	An integer denoting the distance downstream of the target gene to look for footprints (default = 0)
biotype	A type of biological unit (default="protein_coding")
moleculetype	A type of molecule (default="gene")

Value

A dataframe containing all footprints for the specified gene and accompanying parameters

See Also

Other FootprintFinder methods: FootprintFinder-class, closeDatabaseConnections,FootprintFinder-method, getChromLoc,FootprintFinder-method, getFootprintsInRegion,FootprintFinder-method, getGenePromoterRegion,FootprintFinder-method, getGtfGeneBioTypes,FootprintFinder-method, getGtfMoleculeTypes,FootprintFinder-method, getPromoterRegionsAllGenes,FootprintFinder-method

Examples

```
db.address <- system.file(package="trena", "extdata")
genome.db.uri <- paste("sqlite:/",db.address,"mef2c.neighborhood.hg38.gtfAnnotation.db", sep = "/")
project.db.uri <- paste("sqlite:/",db.address,"mef2c.neigborhood.hg38.footprints.db", sep = "/")
fp <- FootprintFinder(genome.db.uri, project.db.uri)

footprints <- getFootprintsForGene(fp, gene = "MEF2C")
```

---

getFootprintsInRegion,FootprintFinder-method

Get Footprints in a Region

Description

Using the regions and hits tables inside the project database specified by the FootprintFinder object, return the location, chromosome, starting position, and ending positions of all footprints for the specified region.

Usage

```
## S4 method for signature 'FootprintFinder'
getFootprintsInRegion(obj, chromosome, start,
  end)
```

Arguments

obj	An object of class FootprintFinder
chromosome	The name of the chromosome of interest
start	An integer denoting the start of the desired region
end	An integer denoting the end of the desired region

Value

A dataframe containing all footprints for the specified region

See Also

Other FootprintFinder methods: FootprintFinder-class, closeDatabaseConnections,FootprintFinder-method, getChromLoc,FootprintFinder-method, getFootprintsForGene,FootprintFinder-method, getGenePromoterRe
getGtfGeneBioTypes,FootprintFinder-method, getGtfMoleculeTypes,FootprintFinder-method, getPromoterRegionsAllGenes,FootprintFinder-method

Examples

```
db.address <- system.file(package="trena", "extdata")
genome.db.uri <- paste("sqlite:/",db.address,"mef2c.neighborhood.hg38.gtfAnnotation.db", sep = "/")
project.db.uri <- paste("sqlite:/",db.address,"mef2c.neigborhood.hg38.footprints.db", sep = "/")
fp <- FootprintFinder(genome.db.uri, project.db.uri)

footprints <- getFootprintsInRegion(fp, chromosome = "chr5",
start = 88903305, end = 88903319 )
```

---

getGeneModelTableColumnNames,Trena-method

Retrieve the column names in the gene model table for a Trena object

---

Description

Retrieve the column names in the gene model table for a Trena object

Usage

```
## S4 method for signature 'Trena'
getGeneModelTableColumnNames(obj)
```

Arguments

obj	An object of class Trena

Value

A character vector listing the column names in the Trena object gene model table

Examples

```
# Create a Trena object and retrieve the column names of the gene model table
trena <- Trena("mm10")
tbl.cols <- getRegulatoryTableColumnNames(trena)
```

---

getGenePromoterRegion,FootprintFinder-method

Get Gene Promoter Region

Description

Using the getChromLoc function in conjunction with the gtf table inside the genome database specified by the FootprintFinder object, get the chromosome, starting location, and ending location for gene promoter region.

Usage

```
## S4 method for signature 'FootprintFinder'
getGenePromoterRegion(obj, gene,
  size.upstream = 1000, size.downstream = 0,
  biotype = "protein_coding", moleculetype = "gene")
```

Arguments

obj	An object of class FootprintFinder
gene	A gene name of ID
size.upstream	An integer denoting the distance upstream of the target gene to look for footprints (default = 1000)
size.downstream	An integer denoting the distance downstream of the target gene to look for footprints (default = 0)
biotype	A type of biological unit (default="protein_coding")
moleculetype	A type of molecule (default="gene")

Value

A list containing 3 elements: 1) chr : The name of the chromosome containing the promoter region for the specified gene 2) start : The starting location of the promoter region for the specified gene 3) end : The ending location of the promoter region for the specified gene

See Also

Other FootprintFinder methods: FootprintFinder-class, closeDatabaseConnections,FootprintFinder-method, getChromLoc,FootprintFinder-method, getFootprintsForGene,FootprintFinder-method, getFootprintsInRe getGtfGeneBioTypes,FootprintFinder-method, getGtfMoleculeTypes,FootprintFinder-method, getPromoterRegionsAllGenes,FootprintFinder-method

Examples

```
db.address <- system.file(package="trena", "extdata")
genome.db.uri <- paste("sqlite:/",db.address,"mef2c.neighborhood.hg38.gtfAnnotation.db", sep = "/")
project.db.uri <- paste("sqlite:/",db.address,"mef2c.neigborhood.hg38.footprints.db", sep = "/")
fp <- FootprintFinder(genome.db.uri, project.db.uri)

prom.region <- getGenePromoterRegion(fp, gene = "MEF2C")
```

---

getGtfGeneBioTypes,FootprintFinder-method
Get the List of Biotypes

Description

Using the gtf table in the genome database contained in a FootprintFinder object, get the list of different types of biological units (biotypes) contained in the table.

Usage

```
## S4 method for signature 'FootprintFinder'
getGtfGeneBioTypes(obj)
```

Arguments

obj	An object of class FootprintFinder

Value

A sorted list of the types of biological units contained in the gtf table of the genome database.

See Also

Other FootprintFinder methods: FootprintFinder-class, closeDatabaseConnections,FootprintFinder-method, getChromLoc,FootprintFinder-method, getFootprintsForGene,FootprintFinder-method, getFootprintsInRe getGenePromoterRegion,FootprintFinder-method, getGtfMoleculeTypes,FootprintFinder-method, getPromoterRegionsAllGenes,FootprintFinder-method

Examples

```
db.address <- system.file(package="trena", "extdata")
genome.db.uri <- paste("sqlite:/",db.address,"mef2c.neighborhood.hg38.gtfAnnotation.db", sep = "/")
project.db.uri <- paste("sqlite:/",db.address,"mef2c.neigborhood.hg38.footprints.db", sep = "/")
fp <- FootprintFinder(genome.db.uri, project.db.uri)

biotypes <- getGtfGeneBioTypes(fp)
```

---

getGtfMoleculeTypes,FootprintFinder-method

Get the List of Molecule Types

Description

Using the gtf table in the genome database contained in a FootprintFinder object, get the list of different types of molecules contained in the table.

Usage

```
## S4 method for signature 'FootprintFinder'
getGtfMoleculeTypes(obj)
```

Arguments

obj	An object of class FootprintFinder

Value

A sorted list of the types of molecules contained in the gtf table of the genome database.

See Also

Other FootprintFinder methods: FootprintFinder-class, closeDatabaseConnections,FootprintFinder-method, getChromLoc,FootprintFinder-method, getFootprintsForGene,FootprintFinder-method, getFootprintsInRe getGenePromoterRegion,FootprintFinder-method, getGtfGeneBioTypes,FootprintFinder-method, getPromoterRegionsAllGenes,FootprintFinder-method

Examples

```
db.address <- system.file(package="trena", "extdata")
genome.db.uri <- paste("sqlite:/",db.address,"mef2c.neighborhood.hg38.gtfAnnotation.db", sep = "/")
project.db.uri <- paste("sqlite:/",db.address,"mef2c.neigborhood.hg38.footprints.db", sep = "/")
fp <- FootprintFinder(genome.db.uri, project.db.uri)

mol.types <- getGtfMoleculeTypes(fp)
```

---

getPfms,MotifMatcher-method

Retrieve the motifs from the pfms slot

Description

Given a MotifMatcher object, return the motifs, which are stored in the pfms slot.

Usage

```
## S4 method for signature 'MotifMatcher'
getPfms(obj)
```

Arguments

obj	An object of class MotifMatcher

Value

The list of motif matrices stored in the pfms slot.

Examples

```
# Return the default matrix of JASPAR motifs
library(MotifDb)
motifMatcher <- MotifMatcher(genomeName="hg38", pfms = as.list(query(MotifDb, "sapiens")))
motifs <- getPfms(motifMatcher)
```

---

getPromoterRegionsAllGenes,FootprintFinder-method

Get Promoter Regions for All Genes

Description

Using the gtf table inside the genome database specified by the FootprintFinder object, return the promoter regions for every protein-coding gene in the database.

Usage

```
## S4 method for signature 'FootprintFinder'
getPromoterRegionsAllGenes(obj,
  size.upstream = 10000, size.downstream = 10000,
  use_gene_ids = TRUE)
```

Arguments

obj	An object of class FootprintFinder
size.upstream	An integer denoting the distance upstream of each gene's transcription start site to include in the promoter region (default = 1000)
size.downstream	An integer denoting the distance downstream of each gene's transcription start site to include in the promoter region (default = 1000)
use_gene_ids	A binary indicating whether to return gene IDs or gene names (default = T)

Value

A GRanges object containing the promoter regions for all genes

See Also

Other FootprintFinder methods: FootprintFinder-class, closeDatabaseConnections,FootprintFinder-method, getChromLoc,FootprintFinder-method, getFootprintsForGene,FootprintFinder-method, getFootprintsInRe getGenePromoterRegion,FootprintFinder-method, getGtfGeneBioTypes,FootprintFinder-method, getGtfMoleculeTypes,FootprintFinder-method

Examples

```
db.address <- system.file(package="trena", "extdata")
genome.db.uri <- paste("sqlite:/",db.address,"mef2c.neighborhood.hg38.gtfAnnotation.db", sep = "/")
project.db.uri <- paste("sqlite:/",db.address,"mef2c.neigborhood.hg38.footprints.db", sep = "/")
fp <- FootprintFinder(genome.db.uri, project.db.uri)

footprints <- getPromoterRegionsAllGenes(fp)
```

---

getProximalPromoter,Trena-method

Grab the region of the proximal promoter for a given gene symbol

---

Description

For the genome of a given Trena object, retrieve a data frame containing the region surrounding a target gene.

Usage

```
## S4 method for signature 'Trena'
getProximalPromoter(obj, geneSymbols,
  tssUpstream = 1000, tssDownstream = 1000)
```

Arguments

obj	An object of class Trena
geneSymbols	A vector containing genes of interest
tssUpstream	A designated distance upstream of the promoter to use as a shoulder (default = 1000)
tssDownstream	A designated distance downstream of the promoter to use as a shoulder (default = 1000)

Value

A dataframe containing the regions surrounding the proximal promoter

Examples

```
if(interactive()) {  # too slow for the bioc windows build
   # Retrieve the proximal promoter for MEF2C using a shoulder size of 2000 on each side
   trena <- Trena("hg38")
   regions <- getProximalPromoter(trena, "MEF2C", 2000, 2000)
   }
```

---

getRegulators,Solver-method

Retrieve the candiate regulators from a Solver object

Description

Retrieve the candiate regulators from a Solver object

Usage

```
## S4 method for signature 'Solver'
getRegulators(obj)
```

Arguments

obj	An object of class Solver

Value

The candidate regulators associated with a Solver object

Examples

```
# Create a Solver object using the included Alzheimer's data and retrieve the regulators
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
targetGene <- "MEF2C"
candidateRegulators <- setdiff(rownames(mtx.sub), targetGene)
solver <- Solver(mtx.sub, targetGene, candidateRegulators)
regs <- getRegulators(solver)
```

`getRegulatoryChromosomalRegions,Trena-method`
Get the regulatory chromosomal regions for a Trena object

Description

Get the regulatory chromosomal regions for a Trena object

Usage

```
## S4 method for signature 'Trena'
getRegulatoryChromosomalRegions(obj, chromosome,
  chromStart, chromEnd, regulatoryRegionSources, targetGene, targetGeneTSS,
  combine = FALSE)
```

Arguments

`obj`	An object of class Trena
`chromosome`	A choromosome of interest
`chromStart`	The beginning of the desired region
`chromEnd`	The end of the desired region
`regulatoryRegionSources`	A vector containing the names of sources for chromosome information. These can be addresses of footprint databases or the names of DHS databases
`targetGene`	A target gene of interest
`targetGeneTSS`	An integer giving the location of the target gene's transcription start site
`combine`	A logical indicating whether or not to combine the output into one data frame (default = FALSE)

Value

A list of regulatory regions for the supplied target gene. If `combine` is set to `TRUE`, the list is converted into a data frame.

Examples

```
# Get regulatory regions for MEF2C from a footprint database
database.filename <- system.file(package="trena", "extdata", "mef2c.neigborhood.hg38.footprints.db")
database.uri <- sprintf("sqlite://%s", database.filename)
sources <- c(database.uri)

trena <- Trena("hg38")
chromosome <- "chr5"
mef2c.tss <- 88904257
loc.start <- mef2c.tss - 1000
loc.end   <- mef2c.tss + 1000

regions <- getRegulatoryChromosomalRegions(trena, chromosome, mef2c.tss-1000, mef2c.tss+1000,
sources, "MEF2C", mef2c.tss)
```

```
# Get regulatory regions for AQP4 from a Human DHS source
trena <- Trena("hg38")
aqp4.tss <- 26865884
chromosome <- "chr18"
sources <- c("encodeHumanDHS")

regions <- getRegulatoryChromosomalRegions(trena, chromosome, aqp4.tss-1, aqp4.tss+3, sources, "AQP4", aqp4.t
```

`getRegulatoryRegions,HumanDHSFilter-method`
Get a tabel of regulatory regions for a Human DHS filter

Description

Get a tabel of regulatory regions for a Human DHS filter

Usage

```
## S4 method for signature 'HumanDHSFilter'
getRegulatoryRegions(obj, encode.table.name,
  chromosome, start, end, score.threshold = 0)
```

Arguments

`obj`	An object of class HumanDHSFilter
`encode.table.name`	A vector of names for Encode tables
`chromosome`	The chromosome of interest
`start`	The starting position
`end`	The ending position
`score.threshold`	A threshold for the score (default = 200)

Value

A data frame containing the regulatory regions for the filter, including the chromosome, start, and end positions, plus the count and score of each region.

See Also

HumanDHSFilter

Examples

```
## Not run:
# Make a filter for "transcription, DNA-templated" and use it to filter candidates
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
targetGene <- "VRK2"
promoter.length <- 1000
genomeName <- "hg38"
db.address <- system.file(package="trena", "extdata")
genome.db.uri   <- paste("sqlite:/", db.address, "vrk2.neighborhood.hg38.gtfAnnotation.db",  sep = "/")
```

```
jaspar.human <- as.list(query(query(MotifDb, "sapiens"),"jaspar2016"))

# Grab regions for VRK2 using shoulder size of 1000
trena <- Trena(genomeName)
tbl.regions <- getProximalPromoter(trena, "VRK2", 1000, 1000)

hd.filter <- HumanDHSFilter(genomeName, pwmMatchPercentageThreshold = 85,
geneInfoDatabase.uri = genome.db.uri, regions = tbl.regions, pfms = jaspar.human)

chrom <- "chr2"
rs13384219.loc <- 57907323
start <- rs13384219.loc - 10
end <- rs13384219.loc + 10

tableNames <- getEncodeRegulatoryTableNames(hd.filter)

getRegulatoryRegions(hd.filter, tableNames[1], chrom, start, end)

## End(Not run)
```

`getRegulatoryTableColumnNames,Trena-method`
Retrieve the column names in the regulatory table for a Trena object

---

Description

Retrieve the column names in the regulatory table for a Trena object

Usage

```
## S4 method for signature 'Trena'
getRegulatoryTableColumnNames(obj)
```

Arguments

`obj`	An object of class Trena

Value

A character vector listing the column names in the Trena object regulatory table

Examples

```
# Create a Trena object and retrieve the column names of the regulatory table
trena <- Trena("mm10")
tbl.cols <- getRegulatoryTableColumnNames(trena)
```

getSequence,MotifMatcher-method
Retrieve the Sequence for a Set of Regions

Description

Given a MotifMatcher object, a table of chromosomal regions, and an optional set of variants, return the sequences as a new column of the table.

Usage

```
## S4 method for signature 'MotifMatcher'
getSequence(obj, tbl.regions,
  variants = NA_character_)
```

Arguments

obj	An object of class MotifMatcher
tbl.regions	A data frame where each row contains a chromosomal region with the fields "chrom", "start", and "end".
variants	A character containing variants to use for the matching (default = NA_character_) The variants should either have the same number of entries as rows in the tbl.regions, or they should not be supplied.

Value

The tbl.regions data frame with an added column containing the sequence for each entry

Examples

```
## Not run:
# Retrieve the sequences for the rs13384219 neighborhood
library(MotifDb)
motifMatcher <- MotifMatcher(genomeName="hg38",
pfms = as.list(query(query(MotifDb, "sapiens"), "jaspar2016")))
tbl.regions <- data.frame(chrom="chr2", start=57907313, end=57907333, stringsAsFactors=FALSE)
x <- findMatchesByChromosomalRegion(motifMatcher, tbl.regions, pwmMatchMinimumAsPercentage=92)

# Retrieve the sequences, but now include a variant
x.mut <- findMatchesByChromosomalRegion(motifMatcher, tbl.regions,
pwmMatchMinimumAsPercentage=92, "rs13384219")

## End(Not run)
```

getSolverNames,EnsembleSolver-method

Retrieve the solver names from an EnsembleSolver object

Description

Retrieve the solver names from an EnsembleSolver object

Usage

```
## S4 method for signature 'EnsembleSolver'
getSolverNames(obj)
```

Arguments

obj	An object of class Solver

Value

The vector of solver names associated with an EnsembleSolver object

Examples

```
# Create a Solver object using the included Alzheimer's data and retrieve the regulators
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
targetGene <- "MEF2C"
candidateRegulators <- setdiff(rownames(mtx.sub), targetGene)
solver <- EnsembleSolver(mtx.sub, targetGene, candidateRegulators,
solverNames = c("lasso","randomForest"))
solver.names <- getSolverNames(solver)
```

getTarget,Solver-method

Retrieve the target gene from a Solver object

Description

Retrieve the target gene from a Solver object

Usage

```
## S4 method for signature 'Solver'
getTarget(obj)
```

Arguments

obj	An object of class Solver

Value

The target gene associated with a Solver object

Examples

```
# Create a Solver object using the included Alzheimer's data and retrieve the target gene
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
targetGene <- "MEF2C"
candidateRegulators <- setdiff(rownames(mtx.sub), targetGene)
solver <- Solver(mtx.sub, targetGene, candidateRegulators)
target <- getTarget(solver)
```

HumanDHSFilter-class *Create a HumanDHSFilter object*

Description

A HumanDHSFilter object allows for filtering based on DNAse hypersensitivity (DHS) data. Its associated `getCandidates` method uses a genome from a BSgenome database (either hg19 or hg38), DNA region specifications, and (variants/pfms,encodetablename, match to filter a list of possible regulators factors to those that match the supplied criteria.

Usage

```
HumanDHSFilter(genomeName, encodeTableName = "wgEncodeRegDnaseClustered",
  pwmMatchPercentageThreshold, geneInfoDatabase.uri, regions,
  variants = NA_character_, pfms, quiet = TRUE)
```

Arguments

`genomeName`	A character string indicating the reference genome; currently, the only accepted strings are "hg38" and "hg19", both of which are human genomes.
`encodeTableName`	(default = "wgEncodeRegDnaseClustered")
`pwmMatchPercentageThreshold`	A numeric from 0-100 to serve as a threshold for a match
`geneInfoDatabase.uri`	An address for a gene database
`regions`	A data frame containing the regions of interest
`variants`	A character vector containing a list of variants
`pfms`	A list of position frequency matrices, often converted from a MotifList object created by a MotifDb query
`quiet`	A logical denoting whether or not the solver should print output

Value

A CandidateFilter class object that filters using Human DHS data

See Also

`getCandidates-HumanDHSFilter`,

Other Solver class objects: `BayesSpikeSolver`, `EnsembleSolver`, `LassoPVSolver`, `LassoSolver`, `PearsonSolver`, `RandomForestSolver`, `RidgeSolver`, `Solver-class`, `SpearmanSolver`, `SqrtLassoSolver`

Examples

```
if(interactive()) { # takes too long in the bioc windows build
  load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
  targetGene <- "VRK2"
  promoter.length <- 1000
  genomeName <- "hg38"
  db.address <- system.file(package="trena", "extdata")
 genome.db.uri   <- paste("sqlite:/", db.address, "vrk2.neighborhood.hg38.gtfAnnotation.db",  sep = "/")

    # Grab regions for VRK2 using shoulder size of 1000
  trena <- Trena(genomeName)
  tbl.regions <- getProximalPromoter(trena, "VRK2", 1000, 1000)

  hd.filter <- HumanDHSFilter(genomeName, pwmMatchPercentageThreshold = 85,
  geneInfoDatabase.uri = genome.db.uri, regions = tbl.regions,
   pfms = as.list(query(query(MotifDb, "sapiens"),"jaspar2016")))
   } # if interactive
```

LassoPVSolver *Create a Solver class object using the LASSO P-Value solver*

Description

Create a Solver class object using the LASSO P-Value solver

Usage

```
LassoPVSolver(mtx.assay = matrix(), targetGene, candidateRegulators,
  quiet = TRUE)
```

Arguments

mtx.assay	An assay matrix of gene expression data
targetGene	A designated target gene that should be part of the mtx.assay data
candidateRegulators	The designated set of transcription factors that could be associated with the target gene
quiet	A logical denoting whether or not the solver should print output

Value

A Solver class object with LASSO P-Value as the solver

See Also

solve.LassoPV, getAssayData

Other Solver class objects: BayesSpikeSolver, EnsembleSolver, HumanDHSFilter-class, LassoSolver, PearsonSolver, RandomForestSolver, RidgeSolver, Solver-class, SpearmanSolver, SqrtLassoSolver

Examples

```
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
target.gene <- "MEF2C"
tfs <- setdiff(rownames(mtx.sub), target.gene)
lassopv.solver <- LassoPVSolver(mtx.sub, target.gene, tfs)
```

LassoPVSolver-class *An S4 class to represent a LASSO P-Value solver*

Description

An S4 class to represent a LASSO P-Value solver

LassoSolver *Create a Solver class object using the LASSO solver*

Description

Create a Solver class object using the LASSO solver

Usage

```
LassoSolver(mtx.assay = matrix(), targetGene, candidateRegulators,
  regulatorWeights = rep(1, length(candidateRegulators)), alpha = 0.9,
  lambda = numeric(0), keep.metrics = FALSE, quiet = TRUE)
```

Arguments

mtx.assay	An assay matrix of gene expression data
targetGene	A designated target gene that should be part of the mtx.assay data
candidateRegulators	The designated set of transcription factors that could be associated with the target gene
regulatorWeights	A set of weights on the transcription factors (default = rep(1, length(tfs)))
alpha	A parameter from 0-1 that determines the proportion of LASSO to ridge used in the elastic net solver, with 0 being fully ridge and 1 being fully LASSO (default = 0.9)
lambda	A tuning parameter that determines the severity of the penalty function imposed on the elastic net regression. If unspecified, lambda will be determined via permutation testing (default = numeric(0)).
keep.metrics	A logical denoting whether or not to keep the initial supplied metrics versus determining new ones
quiet	A logical denoting whether or not the solver should print output

Value

A Solver class object with LASSO as the solver

See Also

solve.Lasso, getAssayData

Other Solver class objects: BayesSpikeSolver, EnsembleSolver, HumanDHSFilter-class, LassoPVSolver, PearsonSolver, RandomForestSolver, RidgeSolver, Solver-class, SpearmanSolver, SqrtLassoSolver

Examples

```
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
target.gene <- "MEF2C"
tfs <- setdiff(rownames(mtx.sub), target.gene)
lasso.solver <- LassoSolver(mtx.sub, target.gene, tfs)
```

LassoSolver-class *Class LassoSolver*

Description

Class LassoSolver

MotifMatcher-class *Create a MotifMatcher object*

Description

A MotifMatcher object is used directly by the HumanDHSFilter class to match motif matrices to where they occur in the supplied genome.

The MotifMatcher class is used to match motif position weight matrices to places where they occur in a given genome. It requries specification of a genome to search in and a list of motifs to search for. Ordinarily this class is primarily used by the HumanDHSFilter, but can alternatively be used to search for motifs in a given genome without any filtering functionality.

Usage

```
MotifMatcher(genomeName, pfms, quiet = TRUE)
```

Arguments

genomeName	A character string identifying an object of type BSgenome. The genome object contains the information for a specific human genome and should be either "hg38" or "hg19". The supplied genome serves as the search space for matching motifs (default = "hg38").
pfms	A list of motif matrices to serve as queries for the target genome. If supplied, these should be created using a MotifList object from the MotifDb package (see example below). If unspecified, the motifs will default to all vertebrates in the JASPAR database (default = list())
quiet	A logical denoting whether or not the MotifMatcher object should print output

Value

An object of the MotifMatcher class

See Also

HumanDHSFilter

Examples

```
# Specify the genome, and motif list to create a MotifMatcher for only human motifs
library(MotifDb)
mm <- MotifMatcher( genomeName="hg38",
pfms = as.list(query(MotifDb, "sapiens")))
```

---

normalizeModel,PCAMax-method

transform a specific column to fit normal distribution

Description

transform a specific column to fit normal distribution

Usage

```
## S4 method for signature 'PCAMax'
normalizeModel(obj, normalizing.max = 10)
```

Arguments

obj	An object of the class PCAMax
normalizing.max	numeric, a maximum value for the normalized distrubtions

Value

a normalized matrix, each column treated separately

---

parseChromLocString

Parse a string containing a chromosome and location on the genome

---

Description

Given a string of the format "chromosome:start-end", pull out the three values contained in the string and return them as a named list

Usage

```
parseChromLocString(chromLocString)
```

Arguments

chromLocString A string of the format "chromosome:start-end"

Value

A named list containing the chromosome, start, and end from the supplied string

Examples

```
# Note: Both examples return :
# list(chrom="chr10", start=118441047, end=118445892)

# Parse a string containing the "chr" prefix for chromosome
chrom.list <- parseChromLocString("chr10:118441047-118445892")

# Parse a string without the "chr" prefix in the chromosome
chrom.list <- parseChromLocString("10:118441047-118445892")
```

---

parseDatabaseUri *Parse a string containing the information for connecting to a database*

Description

Given a string of the format "database_driver://host/database_name", pull out the 3 pieces of information and return them as a named list.

Usage

```
parseDatabaseUri(database.uri)
```

Arguments

database.uri A string of the format "database_driver://host/database_name"

Value

A named list containing the driver, host, and database name from the supplied string

Examples

```
# Parse a URI for a local PostgreSQL database called "gtf"
parsed.URI <- parseDatabaseUri("postgres://localhost/gtf")

# Parse a URI for the included SQLite database "vrk2.neighborhood.hg38.gtfAnnotation.db" in the local documents
db.address <- system.file(package="trena", "extdata")
genome.db.uri   <- paste("sqlite:/", db.address, "vrk2.neighborhood.hg38.gtfAnnotation.db", sep = "/")
parsed.URI <- parseDatabaseUri(genome.db.uri)
```

PCAMax *Class PCAMax*

Description

Class PCAMax

Create a PCAMax object from a data.frame produced by the EnsembleSolver

Usage

```
PCAMax(tbl, tfIdentifierColumnName = "tf.hgnc")
```

Arguments

tbl	a data.frame
tfIdentifierColumnName	a character string identifying the transcription factor column

Value

a PCAMax object

See Also

EnsembleSolver

PearsonSolver *Create a Solver class object using Pearson correlation coefficients as the solver*

Description

Create a Solver class object using Pearson correlation coefficients as the solver

Usage

```
PearsonSolver(mtx.assay = matrix(), targetGene, candidateRegulators,
  quiet = TRUE)
```

Arguments

mtx.assay	An assay matrix of gene expression data
targetGene	A designated target gene that should be part of the mtx.assay data
candidateRegulators	The designated set of transcription factors that could be associated with the target gene
quiet	A logical denoting whether or not the solver should print output

Value

A Solver class object with Pearson correlation coefficients as the solver

See Also

solve.Pearson, getAssayData

Other Solver class objects: BayesSpikeSolver, EnsembleSolver, HumanDHSFilter-class, LassoPVSolver, LassoSolver, RandomForestSolver, RidgeSolver, Solver-class, SpearmanSolver, SqrtLassoSolver

Examples

```
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
target.gene <- "MEF2C"
tfs <- setdiff(rownames(mtx.sub), target.gene)
pearson.solver <- PearsonSolver(mtx.sub, target.gene, tfs)
```

PearsonSolver-class *An S4 class to represent a Pearson solver*

Description

An S4 class to represent a Pearson solver

RandomForestSolver *Create a Solver class object using the Random Forest solver*

Description

Create a Solver class object using the Random Forest solver

Usage

```
RandomForestSolver(mtx.assay = matrix(), targetGene, candidateRegulators,
  regulatorWeights = rep(1, length(candidateRegulators)), quiet = TRUE)
```

Arguments

mtx.assay	An assay matrix of gene expression data
targetGene	A designated target gene that should be part of the mtx.assay data
candidateRegulators	The designated set of transcription factors that could be associated with the target gene
regulatorWeights	A set of weights on the transcription factors (default = rep(1, length(candidateRegulators)))
quiet	A logical denoting whether or not the solver should print output

Value

A Solver class object with Random Forest as the solver

See Also

solve.RandomForest, getAssayData

Other Solver class objects: BayesSpikeSolver, EnsembleSolver, HumanDHSFilter-class, LassoPVSolver, LassoSolver, PearsonSolver, RidgeSolver, Solver-class, SpearmanSolver, SqrtLassoSolver

Examples

```
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
targetGene <- "MEF2C"
candidateRegulators <- setdiff(rownames(mtx.sub), targetGene)
rf.solver <- RandomForestSolver(mtx.sub, targetGene, candidateRegulators)
```

RandomForestSolver-class

Class RandomForestSolver

Description

Class RandomForestSolver

---

rescalePredictorWeights,Solver-method

Rescale the Predictor Weights

Description

Solvers such as LASSO penalize predictors on a scale of 1 (full weight) to infinity (zero weight). With the `rescalePredictorWeights` method, incoming raw values can be scaled between a possibly theoretical minimum and maximum value.

Usage

```
## S4 method for signature 'Solver'
rescalePredictorWeights(obj, rawValue.min, rawValue.max,
  rawValues)
```

Arguments

`obj`	An object of the Solver class
`rawValue.min`	The minimum value of the raw expression values
`rawValue.max`	The maximum value of the raw expression values
`rawValues`	A matrix of raw expression values

Value

A matrix of the raw values re-scaled using the minimum and maximum values

Examples

```
# Create a LassoSolver object using the included Alzheimer's data and rescale the predictors
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
targetGene <- "MEF2C"
candidateRegulators <- setdiff(rownames(mtx.sub), targetGene)
ls <- LassoSolver(mtx.sub, targetGene, candidateRegulators)
raw.values <- c(241, 4739, 9854, 22215, 658334)
cooked.values <- rescalePredictorWeights(ls, rawValue.min = 1, rawValue.max = 1000000, raw.values)
```

RidgeSolver *Create a Solver class object using the Ridge solver*

Description

Create a Solver class object using the Ridge solver

Usage

```
RidgeSolver(mtx.assay = matrix(), targetGene, candidateRegulators,
  regulatorWeights = rep(1, length(candidateRegulators)), alpha = 0,
  lambda = numeric(0), keep.metrics = FALSE, quiet = TRUE)
```

Arguments

mtx.assay	An assay matrix of gene expression data
targetGene	A designated target gene that should be part of the mtx.assay data
candidateRegulators	The designated set of transcription factors that could be associated with the target gene
regulatorWeights	A set of weights on the transcription factors (default = rep(1, length(tfs)))
alpha	A parameter from 0-1 that determines the proportion of LASSO to ridge used in the elastic net solver, with 0 being fully ridge and 1 being fully LASSO (default = 0.9)
lambda	A tuning parameter that determines the severity of the penalty function imposed on the elastic net regression. If unspecified, lambda will be determined via permutation testing (default = numeric(0)).
keep.metrics	A logical denoting whether or not to keep the initial supplied metrics versus determining new ones
quiet	A logical denoting whether or not the solver should print output

Value

A Solver class object with Ridge as the solver

See Also

solve.Ridge, getAssayData

Other Solver class objects: BayesSpikeSolver, EnsembleSolver, HumanDHSFilter-class, LassoPVSolver, LassoSolver, PearsonSolver, RandomForestSolver, Solver-class, SpearmanSolver, SqrtLassoSolver

Examples

```
## Not run:
  load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
   target.gene <- "MEF2C"
   tfs <- setdiff(rownames(mtx.sub), target.gene)
   ridge.solver <- RidgeSolver(mtx.sub, target.gene, tfs)

## End(Not run)
```

RidgeSolver-class *Class RidgeSolver*

Description

Class RidgeSolver

---

run *Run a Solver object to select features*

Description

Run a Solver object to select features

Usage

```
run(obj)
```

Arguments

obj	An object of a Solver class

Value

A data frame of candidate regulators and the relation to the target gene

run,BayesSpikeSolver-method
Run the Bayes Spike Solver

Description

Given a TReNA object with Bayes Spike as the solver, use the vbsr function to estimate coefficients for each transcription factor as a predictor of the target gene's expression level.

Usage

```
## S4 method for signature 'BayesSpikeSolver'
run(obj)
```

Arguments

obj	An object of the class BayesSpikeSolver

Value

A data frame containing the coefficients relating the target gene to each transcription factor, plus other fit parameters

See Also

vbsr, BayesSpikeSolver

Other solver methods: run,EnsembleSolver-method, run,LassoPVSolver-method, run,LassoSolver-method, run,PearsonSolver-method, run,RandomForestSolver-method, run,RidgeSolver-method, run,SpearmanSolver run,SqrtLassoSolver-method

Examples

```
## Not run:
# Load included Alzheimer's data, create a TReNA object with Bayes Spike as solver, and solve
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
target.gene <- "MEF2C"
tfs <- setdiff(rownames(mtx.sub), target.gene)
bayes.solver <- BayesSpikeSolver(mtx.sub, target.gene, tfs)
tbl <- run(bayes.solver)

# Solve the same Alzheimer's problem, but this time set the number of random starts to 100
bayes.solver <- BayesSpikeSolver(mtx.sub, target.gene, tfs, nOrderings = 100)
tbl <- run(bayes.solver)

## End(Not run)
```

run,EnsembleSolver-method
Run the Ensemble Solver

Description

Given a TReNA object with Ensemble as the solver and a list of solvers (default = "default.solvers"), estimate coefficients for each transcription factor as a predictor of the target gene's expression level. The final scores for the ensemble method combine all specified solvers to create a composite score for each transcription factor. This method should be called using the solve method on an appropriate TReNA object.

Usage

```
## S4 method for signature 'EnsembleSolver'
run(obj)
```

Arguments

obj	An object of class Solver with "ensemble" as the solver string

Details

- concordancea composite score
- pcaMaxa composite of the principal components of the individual solver scores

Value

A data frame containing the scores for all solvers and two composite scores relating the target gene to each transcription factor. The two new scores are:

See Also

EnsembleSolver

Other solver methods: run,BayesSpikeSolver-method, run,LassoPVSolver-method, run,LassoSolver-method, run,PearsonSolver-method, run,RandomForestSolver-method, run,RidgeSolver-method, run,SpearmanSolver run,SqrtLassoSolver-method

Examples

```
## Not run:
# Load included Alzheimer's data, create an Ensemble object with default solvers, and solve
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
target.gene <- "MEF2C"
tfs <- setdiff(rownames(mtx.sub), target.gene)[1:30]
ensemble.solver <- EnsembleSolver(mtx.sub, target.gene, tfs)
tbl <- run(ensemble.solver)

# Solve the same problem, but supply extra arguments that change alpha for LASSO to 0.8 and also
# Change the gene cutoff from 10% to 20%
ensemble.solver <- EnsembleSolver(mtx.sub, target.gene, tfs, geneCutoff = 0.2, alpha.lasso = 0.8)
tbl <- run(ensemble.solver)
```

```
# Solve the original problem with default cutoff and solver parameters, but use only 4 solvers
ensemble.solver <- EnsembleSolver(mtx.sub, target.gene, tfs,
solverNames = c("lasso", "pearson", "ridge"))
tbl <- run(ensemble.solver)

## End(Not run)
```

`run,LassoPVSolver-method`

Run the LASSO P-Value Solver

Description

Given a TReNA object with LASSO P-Value as the solver, use the `lassopv` function to estimate coefficients for each transcription factor as a predictor of the target gene's expression level.

Usage

```
## S4 method for signature 'LassoPVSolver'
run(obj)
```

Arguments

`obj`	An object of class LassoPVSolver

Value

A data frame containing the p-values for each transcription factor pertaining to the target gene plus the Pearson correlations between each transcription factor and the target gene.

See Also

`lassopv`, , `LassoPVSolver`

Other solver methods: `run,BayesSpikeSolver-method`, `run,EnsembleSolver-method`, `run,LassoSolver-method`, `run,PearsonSolver-method`, `run,RandomForestSolver-method`, `run,RidgeSolver-method`, `run,SpearmanSolver` `run,SqrtLassoSolver-method`

Examples

```
# Load included Alzheimer's data, create a TReNA object with Bayes Spike as solver, and solve
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
target.gene <- "MEF2C"
tfs <- setdiff(rownames(mtx.sub), target.gene)
lassopv.solver <- LassoPVSolver(mtx.sub, target.gene, tfs)
tbl <- run(lassopv.solver)
```

run,LassoSolver-method

Run the LASSO Solver

Description

Given a LassoSolver object, use the glmnet function to estimate coefficients for each transcription factor as a predictor of the target gene's expression level.

Usage

```
## S4 method for signature 'LassoSolver'
run(obj)
```

Arguments

obj	An object of class LassoSolver

Value

A data frame containing the coefficients relating the target gene to each transcription factor, plus other fit parameters.

See Also

glmnet,, LassoSolver

Other solver methods: run,BayesSpikeSolver-method, run,EnsembleSolver-method, run,LassoPVSolver-method run,PearsonSolver-method, run,RandomForestSolver-method, run,RidgeSolver-method, run,SpearmanSolver run,SqrtLassoSolver-method

Examples

```
# Load included Alzheimer's data, create a TReNA object with LASSO as solver, and solve
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
target.gene <- "MEF2C"
tfs <- setdiff(rownames(mtx.sub), target.gene)
lasso.solver <- LassoSolver(mtx.sub, target.gene, tfs)
tbl <- run(lasso.solver)
```

run,PearsonSolver-method

Run the Pearson Solver

Description

Given a PearsonSolver object, use the cor function to estimate coefficients for each transcription factor as a predictor of the target gene's expression level.

Usage

```
## S4 method for signature 'PearsonSolver'
run(obj)
```

Arguments

obj	An object of class PearsonSolver

Value

The set of Pearson Correlation Coefficients between each transcription factor and the target gene.

See Also

cor, PearsonSolver

Other solver methods: run,BayesSpikeSolver-method, run,EnsembleSolver-method, run,LassoPVSolver-method
run,LassoSolver-method, run,RandomForestSolver-method, run,RidgeSolver-method, run,SpearmanSolver-m
run,SqrtLassoSolver-method

Examples

```
# Load included Alzheimer's data, create a TReNA object with Bayes Spike as solver, and solve
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
target.gene <- "MEF2C"
tfs <- setdiff(rownames(mtx.sub), target.gene)
pearson.solver <- PearsonSolver(mtx.sub, target.gene, tfs)
tbl <- run(pearson.solver)
```

---

run,RandomForestSolver-method

Run the Random Forest Solver

Description

Given a TReNA object with RandomForest as the solver, use the randomForest function to estimate coefficients for each transcription factor as a predictor of the target gene's expression level.

Usage

```
## S4 method for signature 'RandomForestSolver'
run(obj)
```

Arguments

obj	An object of class TReNA with "randomForest" as the solver string

Value

A data frame containing the IncNodePurity for each candidate regulator. This coefficient estimates the relationship between the candidates and the target gene.

See Also

randomForest, RandomForestSolver

Other solver methods: run,BayesSpikeSolver-method, run,EnsembleSolver-method, run,LassoPVSolver-method
run,LassoSolver-method, run,PearsonSolver-method, run,RidgeSolver-method, run,SpearmanSolver-method
run,SqrtLassoSolver-method

Examples

```
# Load included Alzheimer's data, create a TReNA object with Random Forest as solver, and solve
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
targetGene <- "MEF2C"
candidateRegulators <- setdiff(rownames(mtx.sub), targetGene)
rf.solver <- RandomForestSolver(mtx.sub, targetGene, candidateRegulators)
tbl <- run(rf.solver)
```

---

run,RidgeSolver-method

Run the Ridge Regression Solver

Description

Given a TReNA object with Ridge Regression as the solver, use the glmnet function to estimate coefficients for each transcription factor as a predictor of the target gene's expression level.

Usage

```
## S4 method for signature 'RidgeSolver'
run(obj)
```

Arguments

obj	An object of class RidgeSolver

Value

A data frame containing the coefficients relating the target gene to each transcription factor, plus other fit parameters.

See Also

glmnet, , RidgeSolver

Other solver methods: run,BayesSpikeSolver-method, run,EnsembleSolver-method, run,LassoPVSolver-method
run,LassoSolver-method, run,PearsonSolver-method, run,RandomForestSolver-method, run,SpearmanSolver
run,SqrtLassoSolver-method

Examples

```
# Load included Alzheimer's data, create a TReNA object with Bayes Spike as solver, and solve
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
target.gene <- "MEF2C"
tfs <- setdiff(rownames(mtx.sub), target.gene)
ridge.solver <- RidgeSolver(mtx.sub, target.gene, tfs)
tbl <- run(ridge.solver)
```

run,SpearmanSolver-method

Run the Spearman Solver

Description

Given a TReNA object with Spearman as the solver, use the cor function with method = "spearman" to esimate coefficients for each transcription factor as a predictor of the target gene's expression level.

Usage

```
## S4 method for signature 'SpearmanSolver'
run(obj)
```

Arguments

obj	An object of class SpearmanSolver

Value

The set of Spearman Correlation Coefficients between each transcription factor and the target gene.

See Also

cor, SpearmanSolver

Other solver methods: run,BayesSpikeSolver-method, run,EnsembleSolver-method, run,LassoPVSolver-method run,LassoSolver-method, run,PearsonSolver-method, run,RandomForestSolver-method, run,RidgeSolver-me run,SqrtLassoSolver-method

Examples

```
# Load included Alzheimer's data, create a TReNA object with Bayes Spike as solver, and solve
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
target.gene <- "MEF2C"
tfs <- setdiff(rownames(mtx.sub), target.gene)
spearman.solver <- SpearmanSolver(mtx.sub, target.gene, tfs)
tbl <- run(spearman.solver)
```

run,SqrtLassoSolver-method

Run the Square Root LASSO Solver

Description

Given SqrtLassoSolver object, use the slim function to estimate coefficients for each transcription factor as a predictor of the target gene's expression level.

Usage

```
## S4 method for signature 'SqrtLassoSolver'
run(obj)
```

Arguments

obj	An object of class Solver with "sqrtlasso" as the solver string

Value

A data frame containing the coefficients relating the target gene to each transcription factor, plus other fit parameters.

See Also

slim, SqrtLassoSolver

Other solver methods: run,BayesSpikeSolver-method, run,EnsembleSolver-method, run,LassoPVSolver-method run,LassoSolver-method, run,PearsonSolver-method, run,RandomForestSolver-method, run,RidgeSolver-me run,SpearmanSolver-method

Examples

```
# Load included Alzheimer's data, create a TReNA object with Square Root LASSO as solver,
# and run using a few predictors

## Not run:
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
target.gene <- "MEF2C"

# Designate just 5 predictors and run the solver
tfs <- setdiff(rownames(mtx.sub), target.gene)[1:5]
sqrt.solver <- SqrtLassoSolver(mtx.sub, target.gene, tfs)
tbl <- run(sqrt.solver)

## End(Not run)
```

---

show,BayesSpikeSolver-method

Show the Bayes Spike Solver

Description

Show the Bayes Spike Solver

Usage

```
## S4 method for signature 'BayesSpikeSolver'
show(object)
```

Arguments

object	An object of the class BayesSpikeSolver

Value

A truncated view of the supplied object

Examples

```
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
target.gene <- "MEF2C"
tfs <- setdiff(rownames(mtx.sub), target.gene)
bayes.solver <- BayesSpikeSolver(mtx.sub, target.gene, tfs)
show(bayes.solver)
```

---

show,EnsembleSolver-method

Show the Ensemble Solver

Description

Show the Ensemble Solver

Usage

```
## S4 method for signature 'EnsembleSolver'
show(object)
```

Arguments

object	An object of the class EnsembleSolver

Value

A truncated view of the supplied object

Examples

```
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
target.gene <- "MEF2C"
tfs <- setdiff(rownames(mtx.sub), target.gene)
ensemble.solver <- EnsembleSolver(mtx.sub, target.gene, tfs)
show(ensemble.solver)
```

show,HumanDHSFilter-method *Show the details of a human DHS filter*

Description

Show the details of a human DHS filter

Usage

```
## S4 method for signature 'HumanDHSFilter'
show(object)
```

Arguments

object	An object of class HumanDHSFilter

Value

A list, where one element a character vector of transcription factors that match the GO term and the other is an empty data frame.

See Also

HumanDHSFilter

Examples

```
## Not run:
   # Make a filter and show it
  # load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
   targetGene <- "VRK2"
   promoter.length <- 1000
   genomeName <- "hg38"
   db.address <- system.file(package="trena", "extdata")
  genome.db.uri   <- paste("sqlite:/", db.address, "vrk2.neighborhood.hg38.gtfAnnotation.db",  sep = "/")
   jaspar.human <- as.list(query(query(MotifDb, "sapiens"),"jaspar2016"))
      # Grab regions for VRK2 using shoulder size of 1000
   trena <- Trena(genomeName)
   tbl.regions <- getProximalPromoter(trena, "VRK2", 1000, 1000)
   hd.filter <- HumanDHSFilter(genomeName, pwmMatchPercentageThreshold = 85,
   geneInfoDatabase.uri = genome.db.uri, regions = tbl.regions, pfms = jaspar.human)
   show(hd.filter)

## End(Not run)
```

show,LassoPVSolver-method
Show the Lasso PV Solver

Description

Show the Lasso PV Solver

Usage

```
## S4 method for signature 'LassoPVSolver'
show(object)
```

Arguments

object	An object of the class LassoPVSolver

Value

A truncated view of the supplied object

Examples

```
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
target.gene <- "MEF2C"
tfs <- setdiff(rownames(mtx.sub), target.gene)
lassopv.solver <- LassoPVSolver(mtx.sub, target.gene, tfs)
show(lassopv.solver)
```

show,LassoSolver-method
Show the Lasso Solver

Description

Show the Lasso Solver

Usage

```
## S4 method for signature 'LassoSolver'
show(object)
```

Arguments

object	An object of the class LassoSolver

Value

A truncated view of the supplied object

Examples

```
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
target.gene <- "MEF2C"
tfs <- setdiff(rownames(mtx.sub), target.gene)
lasso.solver <- LassoSolver(mtx.sub, target.gene, tfs)
show(lasso.solver)
```

---

show,MotifMatcher-method

Show a MotifMatcher object

Description

Show a MotifMatcher object

Usage

```
## S4 method for signature 'MotifMatcher'
show(object)
```

Arguments

object	An object of the class MotifMatcher

Value

A truncated view of the supplied object

Examples

```
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
target.gene <- "MEF2C"
tfs <- setdiff(rownames(mtx.sub), target.gene)
lassopv.solver <- LassoPVSolver(mtx.sub, target.gene, tfs)
show(lassopv.solver)
```

---

show,PearsonSolver-method

Show the Pearson Solver

Description

Show the Pearson Solver

Usage

```
## S4 method for signature 'PearsonSolver'
show(object)
```

Arguments

object	An object of the class PearsonSolver

Value

A truncated view of the supplied object

Examples

```
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
target.gene <- "MEF2C"
tfs <- setdiff(rownames(mtx.sub), target.gene)
pearson.solver <- PearsonSolver(mtx.sub, target.gene, tfs)
show(pearson.solver)
```

---

show,RandomForestSolver-method

Show the Random Forest Solver

Description

Show the Random Forest Solver

Usage

```
## S4 method for signature 'RandomForestSolver'
show(object)
```

Arguments

object	An object of the class RandomForestSolver

Value

A truncated view of the supplied object

Examples

```
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
target.gene <- "MEF2C"
tfs <- setdiff(rownames(mtx.sub), target.gene)
rf.solver <- RandomForestSolver(mtx.sub, target.gene, tfs)
show(rf.solver)
```

---

show,RidgeSolver-method

Show the Ridge Solver

Description

Show the Ridge Solver

Usage

```
## S4 method for signature 'RidgeSolver'
show(object)
```

Arguments

object	An object of the class RidgeSolver

Value

A truncated view of the supplied object

Examples

```
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
target.gene <- "MEF2C"
tfs <- setdiff(rownames(mtx.sub), target.gene)
ridge.solver <- RidgeSolver(mtx.sub, target.gene, tfs)
show(ridge.solver)
```

---

show,SpearmanSolver-method

Show the Spearman Solver

Description

Show the Spearman Solver

Usage

```
## S4 method for signature 'SpearmanSolver'
show(object)
```

Arguments

object	An object of the class SpearmanSolver

Value

A truncated view of the supplied object

Examples

```
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
target.gene <- "MEF2C"
tfs <- setdiff(rownames(mtx.sub), target.gene)
spearman.solver <- SpearmanSolver(mtx.sub, target.gene, tfs)
show(spearman.solver)
```

---

show,SqrtLassoSolver-method *Show the Square Root Lasso Solver*

Description

Show the Square Root Lasso Solver

Usage

```
## S4 method for signature 'SqrtLassoSolver'
show(object)
```

Arguments

object	An object of the class SqrtLassoSolver

Value

A truncated view of the supplied object

Examples

```
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
target.gene <- "MEF2C"
tfs <- setdiff(rownames(mtx.sub), target.gene)
sqrt.solver <- SqrtLassoSolver(mtx.sub, target.gene, tfs)
show(sqrt.solver)
```

Solver-class *Define an object of class Solver*

Description

The Solver class is a base class that governs the different solvers available in trena. It is rarely called by itself; rather, interaction with a particular solver object is achieved using a specific solver type.

Usage

```
Solver(mtx.assay = matrix(), targetGene, candidateRegulators,
  quiet = TRUE)
```

Arguments

mtx.assay	An assay matrix of gene expression data
targetGene	A designated target gene that should be part of the mtx.assay data
candidateRegulators	The designated set of transcription factors that could be associated
quiet	A logical indicating whether or not the Solver object should print output

Value

An object of the Solver class

See Also

getAssayData, getTarget, getRegulators

Other Solver class objects: BayesSpikeSolver, EnsembleSolver, HumanDHSFilter-class, LassoPVSolver, LassoSolver, PearsonSolver, RandomForestSolver, RidgeSolver, SpearmanSolver, SqrtLassoSolver

Examples

```
#' # Create a Solver object using the included Alzheimer's data
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
targetGene <- "MEF2C"
candidateRegulators <- setdiff(rownames(mtx.sub), targetGene)
solver <- Solver(mtx.sub, targetGene, candidateRegulators) # Create a simple Solver object with default options
```

SpearmanSolver *Create a Solver class object using Spearman correlation coefficients as the solver*

Description

Create a Solver class object using Spearman correlation coefficients as the solver

Usage

```
SpearmanSolver(mtx.assay = matrix(), targetGene, candidateRegulators,
  quiet = TRUE)
```

Arguments

mtx.assay	An assay matrix of gene expression data
targetGene	A designated target gene that should be part of the mtx.assay data
candidateRegulators	The designated set of transcription factors that could be associated with the target gene
quiet	A logical denoting whether or not the solver should print output

Value

A Solver class object with Spearman correlation coefficients as the solver

See Also

solve.Spearman, getAssayData

Other Solver class objects: BayesSpikeSolver, EnsembleSolver, HumanDHSFilter-class, LassoPVSolver, LassoSolver, PearsonSolver, RandomForestSolver, RidgeSolver, Solver-class, SqrtLassoSolver

Examples

```
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
target.gene <- "MEF2C"
tfs <- setdiff(rownames(mtx.sub), target.gene)
spearman.solver <- SpearmanSolver(mtx.sub, target.gene, tfs)
```

SpearmanSolver-class *An S4 class to represent a Spearman solver*

Description

An S4 class to represent a Spearman solver

SqrtLassoSolver *Create a Solver class object using the Square Root LASSO solver*

Description

Create a Solver class object using the Square Root LASSO solver

Usage

```
SqrtLassoSolver(mtx.assay = matrix(), targetGene, candidateRegulators,
  regulatorWeights = rep(1, length(candidateRegulators)),
  lambda = numeric(0), nCores = 4, quiet = TRUE)
```

Arguments

mtx.assay	An assay matrix of gene expression data
targetGene	A designated target gene that should be part of the mtx.assay data
candidateRegulators	The designated set of transcription factors that could be associated with the target gene
regulatorWeights	A set of weights on the transcription factors (default = rep(1, length(tfs)))
lambda	A tuning parameter that determines the severity of the penalty function imposed on the elastic net regression. If unspecified, lambda will be determined via permutation testing (default = numeric(0)).
nCores	An integer specifying the number of computational cores to devote to this square root LASSO solver. This solver is generally quite slow and is greatly sped up when using multiple cores (default = 4)
quiet	A logical denoting whether or not the solver should print output

Value

A Solver class object with Square Root LASSO as the solver

See Also

solve.SqrtLasso, getAssayData

Other Solver class objects: BayesSpikeSolver, EnsembleSolver, HumanDHSFilter-class, LassoPVSolver, LassoSolver, PearsonSolver, RandomForestSolver, RidgeSolver, Solver-class, SpearmanSolver

Examples

```
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
target.gene <- "MEF2C"
tfs <- setdiff(rownames(mtx.sub), target.gene)
sqrt.solver <- SqrtLassoSolver(mtx.sub, target.gene, tfs)
```

SqrtLassoSolver-class *An S4 class to represent a Square Root LASSO solver*

Description

An S4 class to represent a Square Root LASSO solver

Trena-class *Define an object of class Trena*

Description

The Trena class provides a convenient wrapper for the most commonly used filters and solver in the trena package. Given a particular genome (one of c("hg38","mm10"), the Trena class provides methods to retrieve information about possible regulators for a target gene, assess the effects of SNPs, and create gene models using the flexible EnsembleSolver class.

Usage

```
Trena(genomeName, quiet = TRUE)
```

Arguments

genomeName	A string indicating the genome used by the Trena object. Currently, only human and mouse ("hg38","mm10") are supported
quiet	A logical indicating whether or not the Trena object should print output

Value

An object of the Trena class

See Also

getRegulatoryChromosomalRegions, getRegulatoryTableColumnNames, getGeneModelTableColumnNames, createGeneModelFromRegulatoryRegions, createGeneModelFromTfList

Examples

```
# Create a Trena object using the human hg38 genome
trena <- Trena("hg38")
```

VarianceFilter-class *Create a VarianceFilter object*

Description

A VarianceFilter object allows for filtering based on the variance of a target gene in relation to other genes in the assay matrix. Using its associated `getCandidates` method, a VarianceFilter object can be used to filter a list of possible transcription factors to those within a given range of the variance of a supplied target gene.

Usage

```
VarianceFilter(mtx.assay = matrix(), targetGene, varSize = 0.5,
  quiet = TRUE)
```

Arguments

`mtx.assay`	An assay matrix of gene expression data
`targetGene`	A designated target gene that must be part of the mtx.assay data
`varSize`	A user-specified fraction (0-1) of the target gene variance to use as a filter
`quiet`	A logical denoting whether or not the solver should print output

Value

A CandidateFilter class object with variance as the filtering method

An object of the VarianceFilter class

See Also

getCandidates-VarianceFilter

Other Filtering Objects: FootprintFilter-class

Examples

```
load(system.file(package="trena", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
variance.filter <- VarianceFilter(mtx.assay = mtx.sub, targetGene = "MEF2C")
```

Index