Package 'TReNA'

July 14, 2017

Type Package

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

Version 0.99.10

Date 2017-04-24

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

Maintainer Matthew Richards <mrichard@systemsbiology.org>

Imports RSQLite, lassopv, randomForest, flare, vbsr, foreach, doParallel, RPostgreSQL, methods, DBI, GenomicRanges

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

Suggests RUnit, limma, 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' 'FootprintFilter.R' 'FootprintFinder.R' 'LassoPVSolver.R' 'LassoSolver.R' 'NullFilter.R' 'PearsonSolver.R' 'RandomForestSolver.R' 'RidgeSolver.R' 'SpearmanSolver.R' 'SqrtLassoSolver.R' 'TReNA.R' 'VarianceFilter.R' 'help.R' 'sharedFunctions.R'

RoxygenNote 6.0.1

NeedsCompilation no

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

TReNA Class Objects

The TReNA class is the central piece of the package. It houses the matrix of gene expression data as well as the details of the solver chosen for feature selection. Its main method is solve, which performs the feature selection and returns the resulting coefficients.

Solver Class Objects

The Solver class is a base class used within a TReNA object. A particular Solver object contains the name of the selected solver and dispatches the correct feature selection method when solve is called on the TReNA 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 separate from the aforementioned classes. It is a base class that contains a gene expression matrix and is used to filter the transcription factors in the matrix. Filtering method depends on the filter type chosen; there are currently the following subclasses: FootprintFilter, NullFilter, 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.

See Also

TReNA

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(), quiet = TRUE)
```

Arguments

mtx.assay	An assay matrix of gene expression data
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, LassoPVSolver, LassoSolver, PearsonSolver, RandomForestSolver, RidgeSolver, Solver-class, SpearmanSolver, SqrtLassoSolver

Examples

```
solver <- BayesSpikeSolver()
```

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 TReNA object.

Usage

```
CandidateFilter(mtx.assay = matrix(), quiet = TRUE)

## S4 method for signature 'CandidateFilter'
getFilterAssayData(obj)
```

Arguments

mtx.assay	An assay matrix of gene expression data
quiet	A logical denoting whether or not the CandidateFilter object should print output
obj	An object of a CandidateFilter class

Value

An object of the Candidate filter class

Methods (by generic)

- `getFilterAssayData`: Retrieve the assay matrix of gene expression data

Slots

`mtx.assay` An assay matrix of gene expression data

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

See Also

`getCandidates`, `getFilterAssayData`

Examples

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

# Create a CandidateFilter object using the included Alzheimer's data and retrieve the matrix
load(system.file(package="TReNA", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
my.filter <- CandidateFilter(mtx.sub)
mtx <- getFilterAssayData(my.filter)
```

`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, mapMotifsToTFsMergeIntoTable,FootprintFinder-method

---

`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(), quiet = TRUE)
```

Arguments

`mtx.assay`	An assay matrix of gene expression data
`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, LassoPVSolver, LassoSolver, PearsonSolver, RandomForestSolver, RidgeSolver, Solver-class, SpearmanSolver, SqrtLassoSolver

Examples

```
solver <- EnsembleSolver()
```

`EnsembleSolver-class` *Class EnsembleSolver*

Description

Class EnsembleSolver

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(mtx.assay = matrix(), quiet = TRUE)
```

Arguments

`mtx.assay`	An assay matrix of gene expression data
`quiet`	A logical denoting whether or not the filter should print output

Value

An object of the FootprintFilter class

See Also

`getCandidates-FootprintFilter`, `getFilterAssayData`

Other Filtering Objects: `NullFilter-class`, `VarianceFilter-class`

Examples

```
load(system.file(package="TReNA", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
footprint.filter <- FootprintFilter(mtx.assay = mtx.sub)
```

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,
mapMotifsToTFsMergeIntoTable,FootprintFinder-method

---

getAssayData *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

```
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"))
solver <- Solver(mtx.sub)
mtx <- getAssayData(solver)
```

`getCandidates` *Get candidate genes using a CandidateFilter object*

Description

Get candidate genes using a CandidateFilter object

Usage

```
getCandidates(obj, extraArgs)
```

Arguments

`obj`	An object of a CandidateFilter class
`extraArgs`	A named list of extra arguments corresponding to the chosen filter

Value

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

See Also

Other getCandidate Methods: `getCandidates,FootprintFilter-method`, `getCandidates,NullFilter-method`, `getCandidates,VarianceFilter-method`

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

Description

Get candidate genes using the variance filter

Usage

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

Arguments

obj	An object of class FootprintFilter
extraArgs	A named list containing 5 fields: • "target.gene" A designated target gene that should be part of the mtx.assay data • "genome.db.uri" A connection to a genome database containing footprint information • "project.db.uri" A connection to a project database containing footprint information • "size.upstream" An integer denoting the distance upstream of the target gene to look for footprints • "size.downstream" An integer denoting the distance downstream of the target gene to look for footprints

Value

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

See Also

FootprintFilter

Other getCandidate Methods: getCandidates,NullFilter-method, getCandidates,VarianceFilter-method, getCandidates

Examples

```
# Use footprint filter with the included SQLite database for MEF2C to filter candidates
# in the included Alzheimer's dataset
load(system.file(package="TReNA", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
footprint.filter <- FootprintFilter(mtx.assay = mtx.sub)

target.gene <- "MEF2C"
db.address <- system.file(package="TReNA", "extdata")
genome.db.uri <- paste("sqlite:/",db.address,"genome.sub.db", sep = "/")
project.db.uri <- paste("sqlite:/",db.address,"project.sub.db", sep = "/")

tfs <- getCandidates(footprint.filter, extraArgs = list("target.gene" = target.gene,
"genome.db.uri" = genome.db.uri, "project.db.uri" = project.db.uri,
"size.upstream" = 1000, "size.downstream" = 1000))
```

---

getCandidates,NullFilter-method

Get candidate genes using the null filter

Description

Get candidate genes using the null filter

Usage

```
## S4 method for signature 'NullFilter'
getCandidates(obj, extraArgs = list())
```

Arguments

obj	An object of class NullFilter
extraArgs	An empty list

Value

A vector containing all genes in the assay matrix

See Also

NullFilter

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

Examples

```
# Using the included Alzheimer's data, return all transcription factors as candidates
load(system.file(package="TReNA", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
null.filter <- NullFilter(mtx.assay=mtx.sub)
tfs <- getCandidates(null.filter)
```

---

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, extraArgs)
```

Arguments

obj	An object of class VarianceFilter
extraArgs	A named list containing two fields: • "target.gene" A designated target gene that should be part of the mtx.assay data • "var.size" A user-specified percentage (0-1) of the target gene variance to use as a filter

Value

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

See Also

VarianceFilter

Other getCandidate Methods: getCandidates,FootprintFilter-method, getCandidates,NullFilter-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)

target.gene <- "MEF2C"
tfs <- getCandidates(variance.filter, extraArgs = list("target.gene" = target.gene, "var.size" = 0.5))
```

---

getChromLoc,FootprintFinder-method
Get Chromasome Location

Description

Using the gtf table in the genome database contained in a FootprintFinder object, get the locations of chromasomes 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, mapMotifsToTFsMergeIntoTable,FootprintFinder-method

Examples

```
db.address <- system.file(package="TReNA", "extdata")
genome.db.uri <- paste("sqlite:/",db.address,"genome.sub.db", sep = "/")
project.db.uri <- paste("sqlite:/",db.address,"project.sub.db", sep = "/")
fp <- FootprintFinder(genome.db.uri, project.db.uri)

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

---

getFilterAssayData *Retrieve the assay matrix of gene expression data*

Description

Retrieve the assay matrix of gene expression data

Usage

```
getFilterAssayData(obj)
```

Arguments

obj	An object of a CandidateFilter class

Value

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

Examples

```
# Create a CandidateFilter object using the included Alzheimer's data and retrieve the matrix
load(system.file(package="TReNA", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
my.filter <- CandidateFilter(mtx.sub)
mtx <- getFilterAssayData(my.filter)
```

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, mapMotifsToTFsMergeIntoTable,FootprintFinder-method

Examples

```
db.address <- system.file(package="TReNA", "extdata")
genome.db.uri <- paste("sqlite:/",db.address,"genome.sub.db", sep = "/")
project.db.uri <- paste("sqlite:/",db.address,"project.sub.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, chromasome, 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, mapMotifsToTFsMergeIntoTable,FootprintFinder-m

Examples

```
db.address <- system.file(package="TReNA", "extdata")
genome.db.uri <- paste("sqlite:/",db.address,"genome.sub.db", sep = "/")
project.db.uri <- paste("sqlite:/",db.address,"project.sub.db", sep = "/")
fp <- FootprintFinder(genome.db.uri, project.db.uri)

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

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 chromasome, 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 chromasome 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, mapMotifsToTFsMergeIntoTable,FootprintFinder-m

Examples

```
db.address <- system.file(package="TReNA", "extdata")
genome.db.uri <- paste("sqlite:/",db.address,"genome.sub.db", sep = "/")
project.db.uri <- paste("sqlite:/",db.address,"project.sub.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, mapMotifsToTFsMergeIntoTable,FootprintFinder-m

Examples

```
db.address <- system.file(package="TReNA", "extdata")
genome.db.uri <- paste("sqlite:/",db.address,"genome.sub.db", sep = "/")
project.db.uri <- paste("sqlite:/",db.address,"project.sub.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, mapMotifsToTFsMergeIntoTable,FootprintFinder-m

Examples

```
db.address <- system.file(package="TReNA", "extdata")
genome.db.uri <- paste("sqlite:/",db.address,"genome.sub.db", sep = "/")
project.db.uri <- paste("sqlite:/",db.address,"project.sub.db", sep = "/")
fp <- FootprintFinder(genome.db.uri, project.db.uri)

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

---

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, mapMotifsToTFsMergeIntoTable,FootprintFinder-method

Examples

```
db.address <- system.file(package="TReNA", "extdata")
genome.db.uri <- paste("sqlite:/",db.address,"genome.sub.db", sep = "/")
project.db.uri <- paste("sqlite:/",db.address,"project.sub.db", sep = "/")
fp <- FootprintFinder(genome.db.uri, project.db.uri)

footprints <- getPromoterRegionsAllGenes(fp)
```

---

getSolverName *Get the Solver Name from a TReNA Object*

Description

Get the Solver Name from a TReNA Object

Usage

```
getSolverName(obj)
```

Arguments

obj	An object of class TReNA

Value

The name of the solver subclass object contained by the given TReNA object

Examples

```
# Create a LassoSolver object using the included Alzheimer's data and retrieve the solver name
load(system.file(package="TReNA", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
solver <- TReNA(mtx.sub, solver = "lasso")
mtx <- getSolverName(solver)
```

getSolverObject *Get the Solver Object from a TReNA Object*

Description

Get the Solver Object from a TReNA Object

Usage

```
getSolverObject(obj)
```

Arguments

obj	An object of class TReNA

Value

The Solver object contained by the given TReNA object

Examples

```
# Create a LassoSolver object using the included Alzheimer's data and retrieve the solver object
load(system.file(package="TReNA", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
solver <- TReNA(mtx.sub, solver = "lasso")
mtx <- getSolverObject(solver)
```

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(), quiet = TRUE)
```

Arguments

mtx.assay	An assay matrix of gene expression data
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, LassoSolver, PearsonSolver, RandomForestSolver, RidgeSolver, Solver-class, SpearmanSolver, SqrtLassoSolver

Examples

```
solver <- LassoPVSolver()
```

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(), quiet = TRUE)
```

Arguments

mtx.assay	An assay matrix of gene expression data
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, LassoPVSolver, PearsonSolver, RandomForestSolver, RidgeSolver, Solver-class, SpearmanSolver, SqrtLassoSolver

Examples

```
solver <- LassoSolver()
```

LassoSolver-class *Class LassoSolver*

Description

Class LassoSolver

mapMotifsToTFsMergeIntoTable,FootprintFinder-method *Map Motifs to Transcription Factors and Merge into a Table*

Description

Using the motifsgenes table inside the genome database specified by the FootprintFinder object, return a table mapping each motif to transcription factors

Usage

```
## S4 method for signature 'FootprintFinder'
mapMotifsToTFsMergeIntoTable(obj, tbl)
```

Arguments

obj	An object of class FootprintFinder
tbl	A dataframe of footprints, generally obtained using getFootprintsInRegion or getFootprintsForGene

Value

A data frame containing the motifs from the supplied footprints table and the transcription factors they map to

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, getPromoterRegionsAllGenes,FootprintFinder-method

Examples

```
db.address <- system.file(package="TReNA", "extdata")
genome.db.uri <- paste("sqlite:/",db.address,"genome.sub.db", sep = "/")
project.db.uri <- paste("sqlite:/",db.address,"project.sub.db", sep = "/")
fp <- FootprintFinder(genome.db.uri, project.db.uri)

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

NullFilter-class *Create a NullFilter object*

Description

A NullFilter object allows for "filtering" of the genes in an assay matrix. Its associated `getCandidates` method returns the list of transcription factors included in the assay matrix.

Usage

```
NullFilter(mtx.assay = matrix(), quiet = TRUE)
```

Arguments

`mtx.assay`	An assay matrix of gene expression data
`quiet`	A logical denoting whether or not the filter should print output

Value

A CandidateFilter class object with null as the filtering method

An object of the NullFilter class

See Also

`getCandidates-NullFilter`, `getFilterAssayData`

Other Filtering Objects: `FootprintFilter-class`, `VarianceFilter-class`

Examples

```
load(system.file(package="TReNA", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
null.filter <- NullFilter(mtx.assay = mtx.sub)
```

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(), quiet = TRUE)
```

Arguments

`mtx.assay`	An assay matrix of gene expression data
`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, LassoPVSolver, LassoSolver, RandomForestSolver, RidgeSolver, Solver-class, SpearmanSolver, SqrtLassoSolver

Examples

```
solver <- PearsonSolver()
```

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(), quiet = TRUE)
```

Arguments

mtx.assay	An assay matrix of gene expression data
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, LassoPVSolver, LassoSolver, PearsonSolver, RidgeSolver, Solver-class, SpearmanSolver, SqrtLassoSolver

Examples

```
solver <- RandomForestSolver()
```

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"))
ls <- LassoSolver(mtx.sub)
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(), quiet = TRUE)
```

Arguments

mtx.assay	An assay matrix of gene expression data
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, LassoPVSolver, LassoSolver, PearsonSolver, RandomForestSolver, Solver-class, SpearmanSolver, SqrtLassoSolver

Examples

```
solver <- RidgeSolver()
```

RidgeSolver-class *Class RidgeSolver*

Description

Class RidgeSolver

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. This method should be called using the solve method on an appropriate TReNA object.

Usage

```
## S4 method for signature 'BayesSpikeSolver'
run(obj, target.gene, tfs, tf.weights = rep(1,
  length(tfs)), extraArgs = list())
```

Arguments

obj	An object of the class Solver with "bayesSpike" as the solver string
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)))
extraArgs	Modifiers to the Bayes Spike solver; this includes n_orderings, the number of random starts used by the solver

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, solve,TReNA-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"))
trena <- TReNA(mtx.assay = mtx.sub, solver = "bayesSpike")
target.gene <- "MEF2C"
tfs <- setdiff(rownames(mtx.sub), target.gene)
tbl <- solve(trena, target.gene, tfs)

# Solve the same Alzheimer's problem, but this time set the number of random starts to 100
tbl <- solve(trena, target.gene, tfs, extraArgs = list("n_orderings" = 100))
```

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, target.gene, tfs, tf.weights = rep(1,
  length(tfs)), extraArgs = list())
```

Arguments

obj	An object of class Solver with "ensemble" as the solver string
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)))
extraArgs	Modifiers to the Ensemble solver, including "solver.list", "gene.cutoff", and solver-named arguments denoting extraArgs that correspond to a given solver (e.g. "lasso")

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:

- "concordance": a composite score created similarly to "extreme_score", but with each solver's score scaled using *atan(x)*. This score scales from 0-1
- "pcaMax": a composite score created using the root mean square of the principal components of the individual solver scores

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, solve,TReNA-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"))
trena <- TReNA(mtx.assay = mtx.sub, solver = "ensemble")
target.gene <- "MEF2C"
tfs <- setdiff(rownames(mtx.sub), target.gene)
tbl <- solve(trena, target.gene, tfs)

# 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%
tbl <- solve(trena, target.gene, tfs, extraArgs = list("gene.cutoff" = 0.2, "lasso" = list("alpha" = 0.8)))

# Solve the original problem with default cutoff and solver parameters, but use only 4 solvers
tbl <- solve(trena, target.gene, tfs, extraArgs = list("solver.list" = c("lasso", "randomForest", "pearson",
```

---

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. This method should be called using the solve method on an appropriate TReNA object.

Usage

```
## S4 method for signature 'LassoPVSolver'
run(obj, target.gene, tfs, tf.weights = rep(1,
  length(tfs)), extraArgs = list())
```

Arguments

obj	An object of class Solver with "lassopv" as the solver string
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)))
extraArgs	Modifiers to the Lasso P-Value solver

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, solve,TReNA-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"))
trena <- TReNA(mtx.assay = mtx.sub, solver = "lassopv")
target.gene <- "MEF2C"
tfs <- setdiff(rownames(mtx.sub), target.gene)
tbl <- solve(trena, target.gene, tfs)
```

---

run,LassoSolver-method

Run the LASSO Solver

Description

Given a TReNA object with LASSO as the solver, use the glmnet function to estimate coefficients for each transcription factor as a predictor of the target gene's expression level. This method should be called using the solve method on an appropriate TReNA object.

Usage

```
## S4 method for signature 'LassoSolver'
run(obj, target.gene, tfs, tf.weights = rep(1,
  length(tfs)), extraArgs = list())
```

Arguments

obj	An object of class Solver with "lasso" as the solver string
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)))
extraArgs	Modifiers to the LASSO solver

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, solve,TReNA-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"))
trena <- TReNA(mtx.assay = mtx.sub, solver = "lasso")
target.gene <- "MEF2C"
tfs <- setdiff(rownames(mtx.sub), target.gene)
tbl <- solve(trena, target.gene, tfs)
```

`run,PearsonSolver-method`

Run the Pearson Solver

Description

Given a TReNA object with Pearson as the solver, use the `cor` function to estimate coefficients for each transcription factor as a perdictor of the target gene's expression level. This method should be called using the `solve` method on an appropriate TReNA object.

Usage

```
## S4 method for signature 'PearsonSolver'
run(obj, target.gene, tfs, tf.weights = rep(1,
  length(tfs)), extraArgs = list())
```

Arguments

`obj`	An object of class Solver with "pearson" as the solver string
`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)))
`extraArgs`	Modifiers to the Pearson solver

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`, `solve,TReNA-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"))
trena <- TReNA(mtx.assay = mtx.sub, solver = "pearson")
target.gene <- "MEF2C"
tfs <- setdiff(rownames(mtx.sub), target.gene)
tbl <- solve(trena, target.gene, tfs)
```

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. This method should be called using the solve method on an appropriate TReNA object.

Usage

```
## S4 method for signature 'RandomForestSolver'
run(obj, target.gene, tfs, tf.weights = rep(1,
  length(tfs)), extraArgs = list())
```

Arguments

obj	An object of class Solver with "randomForest" as the solver string
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)))
extraArgs	Modifiers to the Random Forest solver

Value

A list containing various parameters of the Random Forest fit.

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, solve,TReNA-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"))
trena <- TReNA(mtx.assay = mtx.sub, solver = "randomForest")
target.gene <- "MEF2C"
tfs <- setdiff(rownames(mtx.sub), target.gene)
tbl <- solve(trena, target.gene, tfs)
```

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. This method should be called using the solve method on an appropriate TReNA object.

Usage

```
## S4 method for signature 'RidgeSolver'
run(obj, target.gene, tfs, tf.weights = rep(1,
  length(tfs)), extraArgs = list())
```

Arguments

obj	An object of class Solver with "ridge" as the solver string
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)))
extraArgs	Modifiers to the Ridge Regression solver

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, solve,TReNA-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"))
trena <- TReNA(mtx.assay = mtx.sub, solver = "ridge")
target.gene <- "MEF2C"
tfs <- setdiff(rownames(mtx.sub), target.gene)
tbl <- solve(trena, target.gene, tfs)
```

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. This method should be called using the solve method on an appropriate TReNA object.

Usage

```
## S4 method for signature 'SpearmanSolver'
run(obj, target.gene, tfs, tf.weights = rep(1,
  length(tfs)), extraArgs = list())
```

Arguments

obj	An object of class Solver with "spearman" as the solver string
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)))
extraArgs	Modifiers to the Spearman solver

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, solve,TReNA-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"))
trena <- TReNA(mtx.assay = mtx.sub, solver = "pearson")
target.gene <- "MEF2C"
tfs <- setdiff(rownames(mtx.sub), target.gene)
tbl <- solve(trena, target.gene, tfs)
```

run,SqrtLassoSolver-method
Run the Square Root LASSO Solver

Description

Given a TReNA object with Square Root LASSO as the solver, use the slim function to estimate coefficients for each transcription factor as a predictor of the target gene's expression level. This method should be called using the solve method on an appropriate TReNA object.

Usage

```
## S4 method for signature 'SqrtLassoSolver'
run(obj, target.gene, tfs, tf.weights = rep(1,
  length(tfs)), extraArgs = list())
```

Arguments

obj	An object of class Solver with "sqrtlasso" as the solver string
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)))
extraArgs	Modifiers to the Square Root LASSO solver

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, solve,TReNA-method

Examples

```
# Load included Alzheimer's data, create a TReNA object with Square Root LASSO as solver, and solve
# Use 4 cores with the extraArgs argument
load(system.file(package="TReNA", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
trena <- TReNA(mtx.assay = mtx.sub, solver = "sqrtlasso")
target.gene <- "MEF2C"
tfs <- setdiff(rownames(mtx.sub), target.gene)
tbl <- solve(trena, target.gene, tfs, extraArgs = list("num.cores" = 4))
```

`solve,TReNA-method` *Solve a TReNA Object*

Description

A TReNA object contains an assay matrix with expression data for genes of interest and a string representing the chosen solver. The `solve` method runs the specified solver given a target gene and a designated set of transcription factors, returning a list of parameters that quantify the relationship between the transcription factors and the target gene.

Usage

```
## S4 method for signature 'TReNA'
solve(obj, target.gene, tfs, tf.weights = rep(1,
  length(tfs)), extraArgs = list())
```

Arguments

`obj`	An object of class TReNA
`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)))
`extraArgs`	Modifiers to the solver

Value

A data frame containing coefficients relating the target gene to each transcription factor

See Also

`TReNA`

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`, `run,SqrtLassoSolver-method`

Examples

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

Solver-class *Define an object of class Solver*

Description

The Solver class is a generic class that governs the different solvers available in TReNA. A Solver class object is constructed during creation of a TReNA object and resides within the TReNA object. It is rarely called by itself; rather, interaction with a particular solver object is achieved using the solve method on a TReNA object.

Usage

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

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

Arguments

mtx.assay	An assay matrix of gene expression data
quiet	A logical indicating whether or not the Solver object should print output
obj	An object of class Solver

Value

An object of the Solver class

Methods (by generic)

- getAssayData: Retrieve the assay matrix of gene expression data

See Also

getAssayData, TReNA, solve

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

Examples

```
# Create a simple Solver object with default options
mtx <- matrix(rnorm(10000), nrow = 100)
solver <- Solver(mtx)


# 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"))
solver <- Solver(mtx.sub)
mtx <- getAssayData(solver)
```

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(), quiet = TRUE)
```

Arguments

mtx.assay	An assay matrix of gene expression data
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, LassoPVSolver, LassoSolver, PearsonSolver, RandomForestSolver, RidgeSolver, Solver-class, SqrtLassoSolver

Examples

```
solver <- SpearmanSolver()
```

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(), quiet = TRUE)
```

Arguments

mtx.assay	An assay matrix of gene expression data
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, LassoPVSolver, LassoSolver, PearsonSolver, RandomForestSolver, RidgeSolver, Solver-class, SpearmanSolver

Examples

```
solver <- SqrtLassoSolver()
```

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 *Class TReNA*

Description

Class TReNA defines a TReNA object and contains an assay matrix, which contains expression data over a set of samples for a group of genes, and a string representing the name of a chosen solver.

Usage

```
TReNA(mtx.assay = matrix(), solver = "lasso", quiet = TRUE)

## S4 method for signature 'TReNA'
getSolverName(obj)

## S4 method for signature 'TReNA'
getSolverObject(obj)
```

Arguments

mtx.assay	An assay matrix of gene expression data
solver	A string matching the designated solver for relating a target gene to transcription factors. TReNA currently supports 9 solver strings (default = "lasso"): • "lasso" • "ridge" • "randomForest" • "bayesSpike" • "sqrtlasso" • "lassopv" • "pearson" • "spearman" • "ensemble"
quiet	A logical denoting whether or not the TReNA object should print output
obj	An object of class TReNA

Value

An object of the TReNA class

Methods (by generic)

- getSolverName: Retrieve the name of the solver specified in a TReNA object
- getSolverObject: Retrieve the Solver object contained in a TReNA object

See Also

solve, Solver, getSolverName, getSolverObject

Examples

```
# Create a LassoSolver object using the included Alzheimer's data and retrieve the solver name
load(system.file(package="TReNA", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
solver <- TReNA(mtx.sub, solver = "lasso")
mtx <- getSolverName(solver)

# Create a LassoSolver object using the included Alzheimer's data and retrieve the solver object
load(system.file(package="TReNA", "extdata/ampAD.154genes.mef2cTFs.278samples.RData"))
solver <- TReNA(mtx.sub, solver = "lasso")
mtx <- getSolverObject(solver)
```

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(), quiet = TRUE)
```

Arguments

`mtx.assay`	An assay matrix of gene expression data
`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`, `getFilterAssayData`

Other Filtering Objects: `FootprintFilter-class`, `NullFilter-class`

Examples

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

Index