Package ‘TRONCO’

February 21, 2024

Version 2.34.0
Date 2023-10-03
Title TRONCO, an R package for TRanslational ONCOlogy
Depends R (>= 4.1.0),
Imports bnlearn, Rgraphviz, gtools, parallel, foreach, doParallel,
       iterators, RColorBrewer, circlize, igraph, grid, gridExtra,
       xtable, gtable, scales, R.matlab, grDevices, graphics, stats,
       utils, methods
Suggests BiocGenerics, BiocStyle, testthat, knitr, rWikiPathways,
        magick
Name An R package for the inference of cancer progression models from
       heterogeneous genomic data
Description The TRONCO (TRanslational ONCOlogy) R package collects algorithms to infer
       progression models via the approach of Suppes-Bayes Causal Network, both
       from an ensemble of tumors (cross-sectional samples) and within an individual
       patient (multi-region or single-cell samples). The package provides parallel
       implementation of algorithms that process binary matrices where each row
       represents a tumor sample and each column a single-nucleotide or a structural
       variant driving the progression; a 0/1 value models the absence/presence of
       that alteration in the sample. The tool can import data from plain, MAF or
       GISTIC format files, and can fetch it from the cBioPortal for cancer genomics.
       Functions for data manipulation and visualization are provided, as well as
       functions to import/export such data to other bioinformatics tools for, e.g,
       clustering or detection of mutually exclusive alterations. Inferred models can be
       visualized and tested for their confidence via bootstrap and cross-validation.
       TRONCO is used for the implementation of the Pipeline for Cancer Inference (PICNIC).

Encoding UTF-8
License GPL-3

URL https://sites.google.com/site/troncopackage/
BugReports https://github.com/BIMIB-DISCo/TRONCO
biocViews BiomedicalInformatics, Bayesian, GraphAndNetwork,
           SomaticMutation, NetworkInference, Network, Clustering,
           DataImport, SingleCell, ImmunoOncology
RoxygenNote  7.2.3  
VignetteBuilder  knitr  
git_url  https://git.bioconductor.org/packages/TRONCO  
git_branch  RELEASE_3_18  
git_last_commit  74dffd8  
git_last_commit_date  2023-10-24  
Repository  Bioconductor 3.18  
Date/Publication  2024-02-20  
Author  Marco Antoniotti [ctb],  
        Giulio Caravagna [aut],  
        Luca De Sano [cre, aut] (<https://orcid.org/0000-0002-9618-3774>),  
        Alex Graudenzi [aut],  
        Giancarlo Mauri [ctb],  
        Bud Mishra [ctb],  
        Daniele Ramazzotti [aut] (<https://orcid.org/0000-0002-6087-2666>)  
Maintainer  Luca De Sano <luca.desano@gmail.com>  

R topics documented: 

aCML  .................................................. 5  
AND  .................................................... 5  
annotate.description  ..................................... 6  
annotate.stages  .......................................... 6  
as.adj.matrix  ........................................... 7  
as.alterations  ........................................... 8  
as.bootstrap.scores  ...................................... 8  
as.colors  ................................................ 9  
as.conditional.probs  .................................... 10  
as.confidence  ........................................... 10  
as.description  .......................................... 11  
as.events  ................................................ 12  
as.events.in.patterns  .................................... 12  
as.events.in.sample  ..................................... 13  
as.gene  ................................................ 14  
as.genes  ................................................ 14  
as.genes.in.patterns  ..................................... 15  
as.genotypes  ............................................ 16  
as.hypotheses  .......................................... 16  
as.joint.probs  .......................................... 17  
as.kfold.eloss  ......................................... 18  
as.kfold.posterr  ....................................... 18  
as.kfold.prederr  ....................................... 19  
as.marginal.probs  ..................................... 20  
as.models  ............................................... 21  
as.parameters  ......................................... 21
topics documented:

as.pathway ........................................... 22
as.patterns ........................................... 23
as.samples ........................................... 23
as.selective.advantage.relations ...................... 24
as.stages ............................................. 25
as.types .............................................. 25
as.types.in.patterns .................................. 26
change.color ......................................... 27
consolidate.data ...................................... 27
crc_gistic ............................................ 28
crc_maf ................................................ 29
crc_plain ............................................. 29
delete.event .......................................... 30
delete.gene ........................................... 31
delete.hypothesis ..................................... 31
delete.model .......................................... 32
delete.pattern ........................................ 33
delete.samples ....................................... 33
delete.type ........................................... 34
duplicates ............................................. 34
ebind .................................................. 35
enforce.numeric ..................................... 35
enforce.string ........................................ 36
events.selection ...................................... 36
export.graphml ....................................... 37
export.mutex .......................................... 38
export.nbs.input ..................................... 39
extract.MAF.HuGO.Entrez.map ......................... 39
genes.table.report ................................... 40
has.duplicates ....................................... 41
has.model ............................................ 41
has.stages ............................................ 42
hypothesis.add ....................................... 42
hypothesis.add.group ................................ 43
hypothesis.add.homologous ............................ 44
import.genotypes ..................................... 45
import.GISTIC ........................................ 45
import.MAF .......................................... 46
import.model ........................................ 48
import.mutex.groups ................................ 48
intersect.datasets ................................... 49
is.compliant .......................................... 49
join.events .......................................... 50
join.types ............................................ 51
keysToNames ......................................... 51
maf ................................................... 52
muts .................................................. 53
nameToKey ........................................... 53
<table>
<thead>
<tr>
<th>R topics documented:</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>nevents</td>
<td>54</td>
</tr>
<tr>
<td>ngenes</td>
<td>54</td>
</tr>
<tr>
<td>nhypotheses</td>
<td>55</td>
</tr>
<tr>
<td>npatterns</td>
<td>55</td>
</tr>
<tr>
<td>nsamples</td>
<td>56</td>
</tr>
<tr>
<td>ntypes</td>
<td>56</td>
</tr>
<tr>
<td>oncoprint</td>
<td>57</td>
</tr>
<tr>
<td>oncoprint.cbio</td>
<td>59</td>
</tr>
<tr>
<td>OR</td>
<td>60</td>
</tr>
<tr>
<td>order.frequency</td>
<td>60</td>
</tr>
<tr>
<td>pathway.visualization</td>
<td>61</td>
</tr>
<tr>
<td>pheatmap</td>
<td>61</td>
</tr>
<tr>
<td>rank.recurrents</td>
<td>65</td>
</tr>
<tr>
<td>rename.gene</td>
<td>66</td>
</tr>
<tr>
<td>rename.type</td>
<td>67</td>
</tr>
<tr>
<td>samples.selection</td>
<td>67</td>
</tr>
<tr>
<td>sbind</td>
<td>68</td>
</tr>
<tr>
<td>ssplit</td>
<td>68</td>
</tr>
<tr>
<td>stage</td>
<td>69</td>
</tr>
<tr>
<td>TCGA.map.clinical.data</td>
<td>69</td>
</tr>
<tr>
<td>TCGA.multiple.samples</td>
<td>70</td>
</tr>
<tr>
<td>TCGA.remove.multiple.samples</td>
<td>70</td>
</tr>
<tr>
<td>TCGA.shorten.barcodes</td>
<td>71</td>
</tr>
<tr>
<td>test_dataset</td>
<td>71</td>
</tr>
<tr>
<td>test_dataset_no_hypos</td>
<td>72</td>
</tr>
<tr>
<td>test_model</td>
<td>73</td>
</tr>
<tr>
<td>test_model_kfold</td>
<td>73</td>
</tr>
<tr>
<td>trim</td>
<td>74</td>
</tr>
<tr>
<td>tronco.bootstrap</td>
<td>74</td>
</tr>
<tr>
<td>tronco.caprese</td>
<td>75</td>
</tr>
<tr>
<td>tronco.capri</td>
<td>76</td>
</tr>
<tr>
<td>tronco.chowliu</td>
<td>77</td>
</tr>
<tr>
<td>tronco.edmonds</td>
<td>78</td>
</tr>
<tr>
<td>tronco.gabow</td>
<td>80</td>
</tr>
<tr>
<td>tronco.kfold.eloss</td>
<td>81</td>
</tr>
<tr>
<td>tronco.kfold.posterr</td>
<td>82</td>
</tr>
<tr>
<td>tronco.kfold.predderr</td>
<td>83</td>
</tr>
<tr>
<td>tronco.pattern.plot</td>
<td>84</td>
</tr>
<tr>
<td>tronco.plot</td>
<td>84</td>
</tr>
<tr>
<td>tronco.prim</td>
<td>87</td>
</tr>
<tr>
<td>view</td>
<td>88</td>
</tr>
<tr>
<td>which.samples</td>
<td>88</td>
</tr>
<tr>
<td>XOR</td>
<td>89</td>
</tr>
</tbody>
</table>

**Index** | 90
Atypical chronic myeloid leukemia dataset

**Description**

This file contains a TRONCO compliant dataset

**Usage**

data(aCML)

**Format**

TRONCO compliant dataset

**Value**

A standard TRONCO object

**Author(s)**

Luca De Sano

**Source**

data from http://www.nature.com/ng/journal/v45/n1/full/ng.2495.html

---

**Description**

AND hypothesis

**Usage**

AND(...)

**Arguments**

... Atoms of the co-occurrence pattern given either as labels or as partially lifted vectors.

**Value**

Vector to be added to the lifted genotype resolving the co-occurrence pattern
annotate.description

Description
Annotate a description on the selected dataset

Usage
annotate.description(x, label)

Arguments
x A TRONCO compliant dataset.
label A string

Value
A TRONCO compliant dataset.

Examples
data(test_dataset)
annotate.description(test_dataset, 'new description')

annotate.stages

Description
Annotate stage information on the selected dataset

Usage
annotate.stages(x, stages, match.TCGA.patients = FALSE)

Arguments
x A TRONCO compliant dataset.
stages A list of stages. Rownames must match samples list of x
match.TCGA.patients
    Match using TCGA notations (only first 12 characters)
as.adj.matrix

Value

A TRONCO compliant dataset.

Examples

data(test_dataset)
data(stage)
test_dataset = annotate.stages(test_dataset, stage)
as.stages(test_dataset)

as.adj.matrix

Description

Extract the adjacency matrix of a TRONCO model. The matrix is indexed with colnames/rownames which represent genotype keys - these can be resolved with function keysToNames. It is possible to specify a subset of events to build the matrix, a subset of models if multiple reconstruction have been performed. Also, either the prima facie matrix or the post-regularization matrix can be extracted.

Usage

as.adj.matrix(x, events = as.events(x), models = names(x$model), type = "fit")

Arguments

x A TRONCO model.

events A subset of events as of as.events(x), all by default.

models A subset of reconstructed models, all by default.

type Either the prima facie ("pf") or the post-regularization ("fit") matrix, "fit" by default.

Value

The adjacency matrix of a TRONCO model.

Examples

data(test_model)
as.adj.matrix(test_model)
as.adj.matrix(test_model, events=as.events(test_model)[5:15,])
as.adj.matrix(test_model, events=as.events(test_model)[5:15,], type='pf')
as.alterations

Description
Return a dataset where all events for a gene are merged in a unique event, i.e., a total of gene-level alterations disregarding the event type. Input ‘x’ is checked to be a TRONCO compliant dataset - see is.compliant.

Usage
as.alterations(x, new.type = "Alteration", new.color = "khaki", silent = FALSE)

Arguments
x
A TRONCO compliant dataset.
new.type
The types label of the new event type, 'Alteration' by default.
new.color
The color of the event new.type, default 'khaki'.
silent
A parameter to disable/enable verbose messages.

Value
A TRONCO compliant dataset with alteration profiles.

Examples
data(muts)
as.alterations(muts)

as.bootstrap.scores

Description
Returns a dataframe with all the bootstrap score in a TRONCO model. It is possible to specify a subset of events or models if multiple reconstruction have been performed.

Usage
as.bootstrap.scores(x, events = as.events(x), models = names(x$model))
as.colors

Arguments

x         A TRONCO model.
events    A subset of events as of as.events(x), all by default.
models    A subset of reconstructed models, all by default.

Value

All the bootstrap scores in a TRONCO model

Examples

data(test_model)
as.bootstrap.scores(test_model)
as.bootstrap.scores(test_model, events=as.events(test_model)[5:15,])

---

as.colors         as.colors

Description

Return the colors associated to each type of event in 'x', which should be a TRONCO compliant dataset - see is.compliant.

Usage

as.colors(x)

Arguments

x         A TRONCO compliant dataset.

Value

A named vector of colors.

Examples

data(test_dataset)
as.colors(test_dataset)
as.conditional.probs  

**Description**

Extract the conditional probabilities from a TRONCO model. The return matrix is indexed with rownames which represent genotype keys - these can be resolved with function keysToNames. It is possible to specify a subset of events to build the matrix, a subset of models if multiple reconstruction have been performed. Also, either the observed or fit probabilities can be extracted.

**Usage**

```r
as.conditional.probs(
  x,
  events = as.events(x),
  models = names(x$model),
  type = "observed"
)
```

**Arguments**

- `x` A TRONCO model.
- `events` A subset of events as of `as.events(x)`, all by default.
- `models` A subset of reconstructed models, all by default.
- `type` observed ('observed')

**Details**

```r
# @examples data(test_model) as.conditional.probs(test_model) as.conditional.probs(test_model, 
events=as.events(test_model)[5:15,])
```

**Value**

The conditional probabilities in a TRONCO model.

as.confidence  

**Description**

Return confidence information for a TRONCO model. Available information are: temporal priority (tp), probability raising (pr), hypergeometric test (hg), parametric (pb), non parametric (npb) or statistical (sb) bootstrap, entropy loss (eloss), prediction error (prederr). Confidence is available only once a model has been reconstructed with any of the algorithms implemented in TRONCO. If more than one model has been reconstructed - for instance via multiple regularizations - confidence information is appropriately nested. The requested confidence is specified via vector parameter conf.
as.description

Usage

as.confidence(x, conf, models = names(x$model))

Arguments

x A TRONCO model.
conf A vector with any of 'tp', 'pr', 'hg', 'nbp', 'pb', 'sb', 'eloss', 'prederr' or 'posterr'.
models The name of the models to extract, all by default.

Value

A list of matrices with the event-to-event confidence.

Examples

data(test_model)
as.confidence(test_model, conf='tp')
as.confidence(test_model, conf=c('tp', 'hg'))

as.description

Description

Return the description annotating the dataset, if any. Input 'x' should be a TRONCO compliant dataset - see is.compliant.

Usage

as.description(x)

Arguments

x A TRONCO compliant dataset.

Value

The description annotating the dataset, if any.

Examples

data(test_dataset)
as.description(test_dataset)
as.events

Description
Return all events involving certain genes and of a certain type in 'x', which should be a TRONCO compliant dataset - see is.compliant.

Usage
as.events(x, genes = NA, types = NA, keysToNames = FALSE)

Arguments
x A TRONCO compliant dataset.
genes The genes to consider, if NA all available genes are used.
types The types of events to consider, if NA all available types are used.
keysToNames If TRUE return a list of mnemonic name composed by type + gene

Value
A matrix with 2 columns (event type, gene name) for the events found.

Examples
data(test_dataset)
as.events(test_dataset)
as.events(test_dataset, types='ins_del')
as.events(test_dataset, genes = 'TET2')
as.events(test_dataset, types='Missing')

as.events.in.patterns

Description
Return the list of events present in selected patterns

Usage
as.events.in.patterns(x, patterns = NULL)

Arguments
x A TRONCO compliant dataset.
patterns A list of patterns for which the list will be returned
as.events.in.sample

Value
A list of events present in patterns which constitute CAPRI’s hypotheses

Examples

data(test_dataset)
as.events.in.patterns(test_dataset)
as.events.in.patterns(test_dataset, patterns='XOR_EZH2')

data(test_dataset)
as.events.in.sample(test_dataset, c('patient 1', 'patient 7'))

Description
Return a list of events which are observed in the input samples list

Usage
as.events.in.sample(x, sample)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>A TRONCO compliant dataset</td>
</tr>
<tr>
<td>sample</td>
<td>Vector of sample names</td>
</tr>
</tbody>
</table>

Value
A list of events which are observed in the input samples list

Examples

data(test_dataset)
as.events.in.sample(test_dataset, c('patient 1', 'patient 7'))
as.gene

Description
Return the genotypes for a certain set of genes and type of events. Input 'x' should be a TRONCO compliant dataset - see is.compliant. In this case column names are substituted with events’ types.

Usage
as.gene(x, genes, types = NA)

Arguments
- x: A TRONCO compliant dataset.
- genes: The genes to consider, if NA all available genes are used.
- types: The types of events to consider, if NA all available types are used.

Value
A matrix, subset of as.genotypes(x) with colnames substituted with events’ types.

Examples
data(test_dataset)
as.gene(test_dataset, genes = c("EZH2", "ASXL1"))

as.genes

Description
Return all gene symbols for which a certain type of event exists in 'x', which should be a TRONCO compliant dataset - see is.compliant.

Usage
as.genes(x, types = NA)

Arguments
- x: A TRONCO compliant dataset.
- types: The types of events to consider, if NA all available types are used.
as.genes.in.patterns

Value

A vector of gene symbols for which a certain type of event exists

Examples

data(test_dataset)
as.genes(test_dataset)

as.genes.in.patterns

Description

Return the list of genes present in selected patterns

Usage

as.genes.in.patterns(x, patterns = NULL)

Arguments

x A TRONCO compliant dataset.
patterns A list of patterns for which the list will be returned

Value

A list of genes present in patterns which constitute CAPRI’s hypotheses

Examples

data(test_dataset)
as.genes.in.patterns(test_dataset)
as.genes.in.patterns(test_dataset, patterns='XOR_EZH2')
as.genotypes

Description
Return all genotypes for input 'x', which should be a TRONCO compliant dataset see is.compliant. Function keysToNames can be used to translate colnames to events.

Usage
as.genotypes(x)

Arguments
x A TRONCO compliant dataset.

Value
A TRONCO genotypes matrix.

Examples
data(test_dataset)
as.genotypes(test_dataset)

as.hypotheses

Description
Return the hypotheses in the dataset which constitute CAPRI's hypotheses.

Usage
as.hypotheses(x, cause = NA, effect = NA)

Arguments
x A TRONCO compliant dataset.
cause A list of genes to use as causes
effect A list of genes to use as effects

Value
The hypotheses in the dataset which constitute CAPRI's hypotheses.
as.joint.probs

Examples

data(test_dataset)
as.hypotheses(test_dataset)

Description

Extract the joint probabilities from a TRONCO model. The return matrix is indexed with row-names/colnames which represent genotype keys - these can be resolved with function keysToNames. It is possible to specify a subset of events to build the matrix, a subset of models if multiple reconstruction have been performed. Also, either the observed or fit probabilities can be extracted.

Usage

as.joint.probs(
  x,
  events = as.events(x),
  models = names(x$model),
  type = "observed"
)

Arguments

x  A TRONCO model.

events  A subset of events as of as.events(x), all by default.

models  A subset of reconstructed models, all by default.

type  observed

Value

The joint probabilities in a TRONCO model.

Examples

data(test_model)
as.joint.probs(test_model)
as.joint.probs(test_model, events=as.events(test_model)[5:15,])
as.kfold.eloss

Description

Returns a dataframe with all the average/stdev entropy loss score of a TRONCO model. It is possible to specify models if multiple reconstruction have been performed.

Usage

```r
as.kfold.eloss(x, models = names(x$model), values = FALSE)
```

Arguments

- `x`: A TRONCO model.
- `models`: A subset of reconstructed models, all by default.
- `values`: If you want to see also the values

Value

All the bootstrap scores in a TRONCO model

Examples

```r
data(test_model_kfold)
as.kfold.eloss(test_model_kfold)
as.kfold.eloss(test_model_kfold, models='capri_aic')
```

as.kfold.posterr

Description

Returns a dataframe with all the posterior classification error score in a TRONCO model. It is possible to specify a subset of events or models if multiple reconstruction have been performed.

Usage

```r
as.kfold.posterr(
  x, 
  events = as.events(x), 
  models = names(x$model), 
  values = FALSE, 
  table = FALSE 
)
```
as.kfold.prederr

Arguments

- **x**: A TRONCO model.
- **events**: A subset of events as of `as.events(x)`, all by default.
- **models**: A subset of reconstructed models, all by default.
- **values**: If you want to see also the values
- **table**: Keep the original table (default false)

Value

All the posterior classification error scores in a TRONCO model

Examples

```r
data(test_model_kfold)
data(test_model)
as.kfold.posterr(test_model_kfold)
as.kfold.posterr(test_model_kfold, events=as.events(test_model)[5:15,])
```

Description

Returns a dataframe with all the prediction error score in a TRONCO model. It is possible to specify a subset of events or models if multiple reconstruction have been performed.

Usage

```r
as.kfold.prederr(
  x,
  events = as.events(x),
  models = names(x$model),
  values = FALSE,
  table = FALSE
)
```

Arguments

- **x**: A TRONCO model.
- **events**: A subset of events as of `as.events(x)`, all by default.
- **models**: A subset of reconstructed models, all by default.
- **values**: If you want to see also the values
- **table**: Keep the original table (default false)
Value

All the bootstrap scores in a TRONCO model

Examples

data(test_model_kfold)
as.kfold.prederr(test_model_kfold)
as.kfold.prederr(test_model_kfold, models='capri_aic')

Description

Extract the marginal probabilities from a TRONCO model. The return matrix is indexed with row-names which represent genotype keys - these can be resolved with function keysToNames. It is possible to specify a subset of events to build the matrix, a subset of models if multiple reconstruction have been performed. Also, either the observed or fit probabilities can be extracted.

Usage

as.marginal.probs(
  x,
  events = as.events(x),
  models = names(x$model),
  type = "observed"
)

Arguments

x                 A TRONCO model.

events            A subset of events as of as.events(x), all by default.

models            A subset of reconstructed models, all by default.

type              observed.

Value

The marginal probabilities in a TRONCO model.

Examples

data(test_model)
as.marginal.probs(test_model)
as.marginal.probs(test_model, events=as.events(test_model)[5:15,])
as.models

Description

Extract the models from a reconstructed object.

Usage

as.models(x, models = names(x$model))

Arguments

x A TRONCO model.
models The name of the models to extract, e.g. 'bic', 'aic', 'caprese', all by default.

Value

The models in a reconstructed object.

Examples

data(test_model)
as.models(test_model)

as.parameters

Description

Get parameters of a model

Usage

as.parameters(x)

Arguments

x A TRONCO model.

Value

A list of parameters
Examples

```r
data(test_model)
as.parameters(test_model)
```

---

**Description**

Given a cohort and a pathway, return the cohort with events restricted to genes involved in the pathway. This might contain a new `pathway` genotype with an alteration mark if any of the involved genes are altered.

**Usage**

```r
as.pathway(
  x,
  pathway.genes,
  pathway.name,
  pathway.color = "yellow",
  aggregate.pathway = TRUE,
  silent = FALSE
)
```

**Arguments**

- `x` A TRONCO compliant dataset.
- `pathway.genes` Gene (symbols) involved in the pathway.
- `pathway.name` Pathway name for visualization.
- `pathway.color` Pathway color for visualization.
- `aggregate.pathway` If TRUE drop the events for the genes in the pathway.
- `silent` A parameter to disable/enable verbose messages.

**Value**

Extract the subset of events for genes which are part of a pathway.

**Examples**

```r
data(test_dataset)
p = as.pathway(test_dataset, c('ASXL1', 'TET2'), 'test_pathway')
```
as.patterns

Description
Return the patterns in the dataset which constitute CAPRI's hypotheses.

Usage
as.patterns(x)

Arguments
x A TRONCO compliant dataset.

Value
The patterns in the dataset which constitute CAPRI's hypotheses.

Examples
data(test_dataset)
as.patterns(test_dataset)

as.samples

Description
Return all sample IDs for input 'x', which should be a TRONCO compliant dataset - see is.compliant.

Usage
as.samples(x)

Arguments
x A TRONCO compliant dataset.

Value
A vector of sample IDs

Examples
data(test_dataset)
as.samples(test_dataset)
Description

Returns a dataframe with all the selective advantage relations in a TRONCO model. Confidence is also shown - see `as.confidence`. It is possible to specify a subset of events or models if multiple reconstruction have been performed.

Usage

```r
as.selective.advantage.relations(
  x,
  events = as.events(x),
  models = names(x$model),
  type = "fit"
)
```

Arguments

- `x` A TRONCO model.
- `events` A subset of events as of `as.events(x)`, all by default.
- `models` A subset of reconstructed models, all by default.
- `type` Either Prima Facie (`'pf'`) or fit (`'fit'`) probabilities, `'fit'` by default.

Value

All the selective advantage relations in a TRONCO model

Examples

```r
data(test_model)
as.selective.advantage.relations(test_model)
as.selective.advantage.relations(test_model, events=as.events(test_model)[5:15,])
as.selective.advantage.relations(test_model, events=as.events(test_model)[5:15,], type='pf')
```
as.stages

**Description**

Return the association sample -> stage, if any. Input 'x' should be a TRONCO compliant dataset - see is.compliant.

**Usage**

```
as.stages(x)
```

**Arguments**

- `x`  
  A TRONCO compliant dataset.

**Value**

A matrix with 1 column annotating stages and rownames as sample IDs.

**Examples**

```
data(test_dataset)
data(stage)
test_dataset = annotate.stages(test_dataset, stage)
as.stages(test_dataset)
```

---

as.types

**Description**

Return the types of events for a set of genes which are in 'x', which should be a TRONCO compliant dataset - see is.compliant.

**Usage**

```
as.types(x, genes = NA)
```

**Arguments**

- `x`  
  A TRONCO compliant dataset.
- `genes`  
  A list of genes to consider, if NA all genes are used.
Value

A matrix with 1 column annotating stages and rownames as sample IDs.

Examples

data(test_dataset)
as.types(test_dataset)
as.types(test_dataset, genes='TET2')

Description

Return the list of types present in selected patterns

Usage

as.types.in.patterns(x, patterns = NULL)

Arguments

x A TRONCO compliant dataset.
patterns A list of patterns for which the list will be returned

Value

A list of types present in patterns which constitute CAPRI's hypotheses

Examples

data(test_dataset)
as.types.in.patterns(test_dataset)
as.types.in.patterns(test_dataset, patterns='XOR_EZH2')
Description

Change the color of an event type

Usage

change.color(x, type, new.color)

Arguments

x  A TRONCO compliant dataset.

 type  An event type

 new.color  The new color (either HEX or R Color)

Value

A TRONCO compliant dataset.

Examples

data(test_dataset)
dataset = change.color(test_dataset, 'ins_del', 'red')

Description

Verify if the input data are consolidate, i.e., if there are events with 0 or 1 probability or indistinguishable in terms of observations

Usage

consolidate.data(x, print = FALSE)

Arguments

x  A TRONCO compliant dataset.

 print  A boolean value stating whether to print or not the summary
Value

The list of any 0 probability, 1 probability and indistinguishable.

Examples

data(test_dataset)
consolidate.data(test_dataset)

crc_gistic

GISTIC example data

Description

This dataset contains an example of GISTIC input of a crc cohort of patients

Usage

data(crc_gistic)

Format

GISTIC score

Value

A gistic file

Author(s)

Daniele Ramazzotti

Source

data from http://www.nature.com/nature/journal/v487/n7407/full/nature11252.html
**Description**
This dataset contains an example of MAF input of a crc cohort of patients

**Usage**
```r
data(crc_maf)
```

**Format**
Manual Annotated Format

**Value**
A MAF file

**Author(s)**
Daniele Ramazzotti

**Source**
data from http://www.nature.com/nature/journal/v487/n7407/full/nature11252.html

---

**Description**
This dataset contains an example of plain input of a crc cohort of patients

**Usage**
```r
data(crc_plain)
```

**Format**
plain data

**Value**
A plain input
Author(s)

Daniele Ramazzotti

Source

data from http://www.nature.com/nature/journal/v487/n7407/full/nature11252.html

---

**delete.event**

**Description**

Delete an event from the dataset

**Usage**

```r
delete.event(x, gene, type)
```

**Arguments**

- `x` A TRONCO compliant dataset.
- `gene` The name of the gene to delete.
- `type` The name of the type to delete.

**Value**

A TRONCO compliant dataset.

**Examples**

```r
data(test_dataset)
test_dataset = delete.event(test_dataset, 'TET2', 'ins_del')
```
**delete.gene**

**Description**
Delete a gene

**Usage**
delete.gene(x, gene)

**Arguments**
x  A TRONCO compliant dataset.
gene  The name of the gene to delete.

**Value**
A TRONCO compliant dataset.

**Examples**
data(test_dataset)
test_dataset = delete.gene(test_dataset, 'TET2')

---

**delete.hypothesis**

**Description**
Delete an hypothesis from the dataset based on a selected event. Check if the selected event exist in the dataset and delete his associated hypothesis

**Usage**
delete.hypothesis(x, event = NA, cause = NA, effect = NA)

**Arguments**
x  A TRONCO compliant dataset.
event  Can be an event or pattern name
cause  Can be an event or pattern name
effect  Can be an event or pattern name
Value

A TRONCO compliant dataset.

Examples

data(test_dataset)
delete.hypothesis(test_dataset, event='TET2')
delete.hypothesis(test_dataset, cause='EZH2')
delete.hypothesis(test_dataset, event='XOR_EZH2')

Description

Delete a reconstructed model from the dataset

Usage

delete.model(x)

Arguments

x A TRONCO compliant dataset.

Value

A TRONCO compliant dataset.

Examples

data(test_model)
model = delete.model(test_model)
has.model(model)
### delete.pattern

**Description**
Delete a pattern and every associated hypotheses from the dataset

**Usage**
```r
delete.pattern(x, pattern)
```

**Arguments**
- `x` A TRONCO compliant dataset.
- `pattern` A pattern name

**Value**
A TRONCO compliant dataset.

**Examples**
```r
data(test_dataset)
delete.pattern(test_dataset, pattern='XOR_EZH2')
```

### delete.samples

**Description**
Delete samples from selected dataset

**Usage**
```r
delete.samples(x, samples)
```

**Arguments**
- `x` A TRONCO compliant dataset.
- `samples` An array of samples name

**Value**
A TRONCO compliant dataset.
Examples

data(test_dataset)
dataset = delete.samples(test_dataset, c('patient 1', 'patient 4'))

delete.type

duplicates
duplicates

Description

Delete an event type

Usage

delete.type(x, type)

Arguments

x A TRONCO compliant dataset.
type The name of the type to delete.

Value

A TRONCO compliant dataset.

Examples

data(test_dataset)
test_dataset = delete.type(test_dataset, 'Pattern')

duplicates
duplicates

Description

Return the events duplicated in x, if any. Input 'x' should be a TRONCO compliant dataset - see is.compliant.

Usage

duplicates(x)

Arguments

x A TRONCO compliant dataset.
Value

A subset of \texttt{as.events(x)} with duplicated events.

Examples

\begin{verbatim}
data(test_dataset)
duplicates(test_dataset)
\end{verbatim}

Description

Binds events from one or more datasets, which must be defined over the same set of samples.

Usage

\begin{verbatim}
ebind(..., silent = FALSE)
\end{verbatim}

Arguments

\begin{verbatim}
... the input datasets
silent A parameter to disable/enable verbose messages.
\end{verbatim}

Value

A TRONCO compliant dataset.

Description

Convert the internal representation of genotypes to numeric, if not.

Usage

\begin{verbatim}
enforce.numeric(x)
\end{verbatim}

Arguments

\begin{verbatim}
x A TRONCO compliant dataset.
\end{verbatim}

Value

Convert the internal representation of genotypes to numeric, if not.
enforce.string

**Description**

Convert the internal representation of genotypes to character, if not.

**Usage**

enforce.string(x)

**Arguments**

- **x**: A TRONCO compliant dataset.

**Value**

Convert the internal representation of genotypes to character, if not.

**Examples**

```r
data(test_dataset)
test_dataset = enforce.string(test_dataset)
```

events.selection

**Description**

select a subset of the input genotypes ‘x’. Selection can be done by frequency and gene symbols.

**Usage**

```r
events.selection(
  x,
  filter.freq = NA,
  filter.in.names = NA,
  filter.out.names = NA,
  silent = FALSE
)
```
export.graphml

Arguments

x             A TRONCO compliant dataset.
filter.freq   [0,1] value which constrains the minimum frequency of selected events
filter.in.names gene symbols which will be included
filter.out.names gene symbols which will NOT be included
silent        A parameter to disable/enable verbose messages.

Value

A TRONCO compliant dataset.

Examples

data(test_dataset)
dataset = events.selection(test_dataset, 0.3)

Description

Create a graphML object which can be imported in cytoscape This function is based on the tronco.plot function

Usage

export.graphml(x, file, ...)

Arguments

x             A TRONCO compliant dataset
file          Where to save the output
...           parameters for tronco.plot

Examples

data(test_model)
export.graphml(test_model, file='text.xml', scale.nodes=0.3)
Description

Create an input file for MUTEX (ref: https://code.google.com/p/mutex/)

Usage

```r
export.mutex(
  x,
  filename = "tronco_to_mutex",
  filepath = "/",
  label.mutation = "SNV",
  label.amplification = list("High-level Gain"),
  label.deletion = list("Homozygous Loss")
)
```

Arguments

- `x` A TRONCO compliant dataset.
- `filename` The name of the file
- `filepath` The path where to save the file
- `label.mutation` The event type to use as mutation
- `label.amplification` The event type to use as amplification (can be a list)
- `label.deletion` The event type to use as amplification (can be a list)

Value

A MUTEX example matrix

Examples

```r
data(crc_gistic)
dataset = import.GISTIC(crc_gistic)
export.mutex(dataset)
```
Description

Create a .mat file which can be used with NBS clustering (ref: http://chianti.ucsd.edu/~mhofree/wordpress/?page_id=26)

Usage

```r
export.nbs.input(x, map_hugo_entrez, file = "tronco_to_nbs.mat")
```

Arguments

- `x`: A TRONCO compliant dataset.
- `map_hugo_entrez`: Hugo_Symbol-Entrez_Gene_Id map
- `file`: output file name

Description

Extract a map Hugo_Symbol -> Entrez_Gene_Id from a MAF input file. If some genes map to ID 0 a warning is raised.

Usage

```r
extract.MAF.HuGO.Entrez.map(file, sep = "\t")
```

Arguments

- `file`: MAF filename
- `sep`: MAF separator, default \"\t\"

Value

A map Hugo_Symbol -> Entrez_Gene_Id.
genes.table.report  genes.table.report

Description

Generate PDF and LaTeX tables

Usage

```
genes.table.report(
  x,
  name, dir = getwd(),
  maxrow = 33,
  font = 10,
  height = 11,
  width = 8.5,
  fill = "lightblue",
  silent = FALSE
)
```

Arguments

- `x`: A TRONCO compliant dataset.
- `name`: filename
- `dir`: working directory
- `maxrow`: maximum number of row per page
- `font`: document fontsize
- `height`: table height
- `width`: table width
- `fill`: fill color
- `silent`: A parameter to disable/enable verbose messages.

Value

LaTeX code
### has.duplicates

**Description**

Return true if there are duplicated events in the TRONCO dataset 'x', which should be a TRONCO compliant dataset - see is.compliant. Events are identified by a gene name, e.g., a HuGO_Symbol, and a type label, e.g., c('SNP', 'KRAS')

**Usage**

```r
has.duplicates(x)
```

**Arguments**

- `x` A TRONCO compliant dataset.

**Value**

TRUE if there are duplicated events in `x`.

**Examples**

```r
data(test_dataset)
has.duplicates(test_dataset)
```

### has.model

**Description**

Return true if there is a reconstructed model in the TRONCO dataset 'x', which should be a TRONCO compliant dataset - see is.compliant.

**Usage**

```r
has.model(x)
```

**Arguments**

- `x` A TRONCO compliant dataset.

**Value**

TRUE if there is a reconstructed model in `x`.
Examples

data(test_dataset)
has.model(test_dataset)

has.stages

Description

Return true if the TRONCO dataset 'x', which should be a TRONCO compliant dataset - see is.compliant - has stage annotations for samples. Some sample stages might be annotated as NA, but not all.

Usage

has.stages(x)

Arguments

x A TRONCO compliant dataset.

Value

TRUE if the TRONCO dataset has stage annotations for samples.

Examples

data(test_dataset)
has.stages(test_dataset)
data(stage)
test_dataset = annotate.stages(test_dataset, stage)
has.stages(test_dataset)

hypothesis.add

Description

Add a new hypothesis by creating a new event and adding it to the compliant genotypes
### Usage

```r
hypothesis.add(
  data,
  pattern.label,
  lifted.pattern,
  pattern.effect = "*",
  pattern.cause = "*
)
```

### Arguments

- **data**: A TRONCO compliant dataset.
- **pattern.label**: Label of the new hypothesis.
- **lifted.pattern**: Vector to be added to the lifted genotype resolving the pattern related to the new hypothesis.
- **pattern.effect**: Possible effects for the pattern.
- **pattern.cause**: Possible causes for the pattern.

### Value

A TRONCO compliant object with the added hypothesis

---

### Description

Add all the hypotheses related to a group of events

### Usage

```r
hypothesis.add.group(
  x,
  FUN,
  group,
  pattern.cause = "*",
  pattern.effect = "*",
  dim.min = 2,
  dim.max = length(group),
  min.prob = 0,
  silent = FALSE
)
```
hypothesis.add.homologous

**Arguments**

- **x**: A TRONCO compliant dataset.
- **FUN**: Type of pattern to be added, e.g., co-occurrence, soft or hard exclusivity.
- **group**: Group of events to be considered.
- **pattern.cause**: Possible causes for the pattern.
- **pattern.effect**: Possible effects for the pattern.
- **dim.min**: Minimum cardinality of the subgroups to be considered.
- **dim.max**: Maximum cardinality of the subgroups to be considered.
- **min.prob**: Minimum probability associated to each valid group.
- **silent**: A parameter to disable/enable verbose messages.

**Value**

A TRONCO compliant object with the added hypotheses

```r
hypothesis.add.homologous
```

**Description**

Add all the hypotheses related to homologous events

**Usage**

```r
hypothesis.add.homologous(
  x,
  pattern.cause = "*",
  pattern.effect = "*",
  genes = as.genes(x),
  silent = FALSE
)
```

**Arguments**

- **x**: A TRONCO compliant dataset.
- **pattern.cause**: Possible causes for the pattern.
- **pattern.effect**: Possible effects for the pattern.
- **genes**: List of genes to be considered as possible homologous. For these genes, all the types of mutations will be considered functionally equivalent.
- **silent**: A parameter to disable/enable verbose messages.

**Value**

A TRONCO compliant object with the added hypotheses
import.genotypes

Description

Import a matrix of 0/1 alterations as a TRONCO compliant dataset. Input "geno" can be either a
dataframe or a file name. In any case the dataframe or the table stored in the file must have a column
for each altered gene and a rows for each sample. Colnames will be used to determine gene names,
if data is loaded from file the first column will be assigned as rownames. For details and examples
regarding the loading functions provided by the package we refer to the Vignette Section 3.

Usage

import.genotypes(geno, event.type = "variant", color = "Darkgreen")

Arguments

genoeither a dataframe or a filename

event.typeAny 1 in "geno" will be interpreted as a an observed alteration labeled with type
"event.type"

colorThis is the color used for visualization of events labeled as of "event.type"

Value

A TRONCO compliant dataset

import.GISTIC

Description

Transform GISTIC scores for CNAs in a TRONCO compliant object. Input can be either a matrix,
with columns for each altered gene and rows for each sample; in this case colnames/rownames must
be provided. If input is a character an attempt to load a table from file is performed. In this case the
input table format should be constitent with TCGA data for focal CNA; there should hence be: one
column for each sample, one row for each gene, a column Hugo_Symbol with every gene name and
a column Entrez_Gene_Id with every gene\'s Entrez ID. A valid GISTIC score should be any value
of: "Homozygous Loss" (-2), "Heterozygous Loss" (-1), "Low-level Gain" (+1), "High-level Gain"
(+2). For details and examples regarding the loading functions provided by the package we refer to
the Vignette Section 3.
Usage

```r
import.GISTIC(
  x,
  filter.genes = NULL,
  filter.samples = NULL,
  silent = FALSE,
  trim = TRUE,
  rna.seq.data = NULL,
  rna.seq.up = NULL,
  rna.seq.down = NULL
)
```

Arguments

- **x** Either a dataframe or a filename
- **filter.genes** A list of genes
- **filter.samples** A list of samples
- **silent** A parameter to disable/enable verbose messages.
- **trim** Remove the events without occurrence
- **rna.seq.data** Either a dataframe or a filename
- **rna.seq.up** TODO
- **rna.seq.down** TODO

Value

A TRONCO compliant representation of the input CNAs.

Examples

```r
data(crc_gistic)
gistic = import.GISTIC(crc_gistic)
```

Description

Import mutation profiles from a Manual Annotation Format (MAF) file. All mutations are aggregated as a unique event type labeled "Mutation" and assigned a color according to the default of function import.genotypes. If this is a TCGA MAF file check for multiple samples per patient is performed and a warning is raised if these occur. Customized MAF files can be imported as well provided that they have columns Hugo_Symbol, Tumor_Sample_Barcodes and Variant_Classification. Custom filters are possible (via filter.fun) to avoid loading the full MAF data. For details and examples regarding the loading functions provided by the package we refer to the Vignette Section 3.
import.MAF

Usage

import.MAF(
    file,
    sep = "\t",
    is.TCGA = TRUE,
    filter.fun = NULL,
    to.TRONCO = TRUE,
    irregular = FALSE,
    paste.to.Hugo_Symbol = NULL,
    merge.mutation.types = TRUE,
    silent = FALSE
)

Arguments

  file      MAF filename
  sep       MAF separator, default \"\t\"
  is.TCGA   TRUE if this MAF is from TCGA; thus its sample codenames can be interpreted
  filter.fun A filter function applied to each row. This is expected to return TRUE/FALSE.
  to.TRONCO If FALSE returns a dataframe with MAF data, not a TRONCO object
  irregular If TRUE seeks only for columns Hugo_Symbol, Tumor_Sample_Barcode and
             Variant_Classification
  paste.to.Hugo_Symbol
             If a list of column names, this will be pasted each Hugo_Symbol to yield names
             such as PHC2.chr1.33116215.33116215
  merge.mutation.types
             If TRUE, all mutations are considered equivalent, regardless of their Variant_Classification
             value. Otherwise no.
  silent    A parameter to disable/enable verbose messages.

Value

  A TRONCO compliant representation of the input MAF

Examples

data(maf)
mutations = import.MAF(maf)
mutations = annotate.description(mutations, 'Example MAF')
mutations = TCGA.shorten.barcodes(mutations)
oncoprint(mutations)
**import.model**

**Description**

Add an adjacency matrix as a model to a TRONCO compliant object. Input model can be either a dataframe or a file name.

**Usage**

```r
import.model(tronco_object, model, model.name = "imported_model")
```

**Arguments**

- `tronco_object`: A TRONCO compliant object
- `model`: Either a dataframe or a filename
- `model.name`: Name of the imported model

**Value**

A TRONCO compliant object

---

**import_mutex.groups**

**Description**

Create a list of unique Mutex groups for a given fdr cutoff current Mutex version is Jan 8, 2015 (ref: https://code.google.com/p/mutex/ )

**Usage**

```r
import_mutex.groups(file, fdr = 0.2, display = TRUE)
```

**Arguments**

- `file`: Mutex results ("ranked-groups.txt" file)
- `fdr`: cutoff for fdr
- `display`: print summary table of extracted groups
intersect.datasets

Description

Intersect samples and events of two dataset.

Usage

intersect.datasets(x, y, intersect.genomes = TRUE)

Arguments

x A TRONCO compliant dataset.
y A TRONCO compliant dataset.
intersect.genomes If False -> just samples.

Value

A TRONCO compliant dataset.

Examples

data(test_dataset)

is.compliant

Description

Check if 'x' is compliant with TRONCO's input: that is if it has dataframes x$genotypes, x$annotations, x$types and x$stage (optional).

Usage

is.compliant(
  x,
  err.fun = "[ERR]",
  stage = !(all(is.null(x$stages)) || all(is.na(x$stages)))
)

Arguments

- `x` A TRONCO compliant dataset.
- `err.fun` string which identifies the function which called `is.compliant` stage
- `stage` boolean flag to check `$x$stage` dataframe

Value

on error stops the computation

Examples

```r
data(test_dataset)
is.compliant(test_dataset)
```

join.events

Description

Merge a list of events in an unique event

Usage

```
join.events(x, ..., new.event, new.type, event.color)
```

Arguments

- `x` A TRONCO compliant dataset.
- `...` A list of events to merge
- `new.event` The name of the resultant event
- `new.type` The type of the new event
- `event.color` The color of the new event

Value

A TRONCO compliant dataset.

Examples

```r
data(muts)
dataset = join.events(muts, 'G1', 'G2', new.event='test', new.type='banana', event.color='yellow')
```
**join.types**

**Description**

For an input dataset merge all the events of two or more distinct types (e.g., say that missense and indel mutations are events of a unique "mutation" type)

**Usage**

```
join.types(x, ..., new.type = "new.type", new.color = "khaki", silent = FALSE)
```

**Arguments**

- **x**: A TRONCO compliant dataset.
- **...**: type to merge
- **new.type**: label for the new type to create
- **new.color**: color for the new type to create
- **silent**: A parameter to disable/enable verbose messages.

**Value**

A TRONCO compliant dataset.

**Examples**

```
data(test_dataset_no_hypos)
join.types(test_dataset_no_hypos, 'ins_del', 'missense_point_mutations')
join.types(test_dataset_no_hypos, 'ins_del',
    'missense_point_mutations', new.type='mut', new.color='green')
```

---

**keysToNames**

**Description**

Convert colnames/rownames of a matrix into intelligible event names, e.g., change a key G23 in 'Mutation KRAS'. If a name is not found, the original name is left unchanged.

**Usage**

```
keysToNames(x, matrix)
```
Arguments

- x: A TRONCO compliant dataset.
- matrix: A matrix with colnames/rownames which represent genotypes keys.

Value

The matrix with intelligible colnames/rownames.

Examples

```r
data(test_model)
adj_matrix = as.adj.matrix(test_model, events=as.events(test_model)[5:15,])$capri_bic
keysToNames(test_model, adj_matrix)
```

maf

MAF example data

Description

This dataset contains a standard MAF input for TRONCO

Usage

```r
data(maf)
```

Format

Manual Annotated Format

Value

A standard TRONCO object

Author(s)

Luca De Sano

Source

fake data
**muts**

**Simple mutation dataset**

**Description**
A simple mutation dataset without hypotheses

**Usage**
data(muts)

**Format**
TRONCO compliant dataset

**Value**
A standard TRONCO object

**Author(s)**
Luca De Sano

**Source**
fake data

---

**nameToKey**

**nameToKey**

**Description**
Convert to key an intelligible event names, e.g., change 'Mutation KRAS' in G23. If a name is not found, an error is raised!

**Usage**
nameToKey(x, name)

**Arguments**
- **x**
  A TRONCO compliant dataset.
- **name**
  A intelligible event name

**Value**
A TRONCO dataset key name
### nevents

Return the number of events in the dataset involving a certain gene or type of event.

#### Usage

```r
events(x, genes = NA, types = NA)
```

#### Arguments

- **x**: A TRONCO compliant dataset.
- **genes**: The genes to consider, if NA all available genes are used.
- **types**: The types of events to consider, if NA all available types are used.

#### Value

The number of events in the dataset involving a certain gene or type of event.

#### Examples

```r
data(test_dataset)
events(test_dataset)
```

### ngenes

Return the number of genes in the dataset involving a certain type of event.

#### Usage

```r
ngenes(x, types = NA)
```

#### Arguments

- **x**: A TRONCO compliant dataset.
- **types**: The types of events to consider, if NA all available types are used.

#### Examples

```r
data(test_dataset)
ngenes(test_dataset)
```
**nhypotheses**

**Value**

The number of genes in the dataset involving a certain type of event.

**Examples**

```r
data(test_dataset)
genres(test_dataset)
```

---

<table>
<thead>
<tr>
<th>nhypotheses</th>
<th>Return the number of hypotheses in the dataset</th>
</tr>
</thead>
</table>

**Description**

Return the number of hypotheses in the dataset

**Usage**

```r
nhypotheses(x)
```

**Arguments**

- `x` the dataset.

**Examples**

```r
data(test_dataset)
nhypotheses(test_dataset)
```

---

<table>
<thead>
<tr>
<th>npatterns</th>
<th>Return the number of patterns in the dataset</th>
</tr>
</thead>
</table>

**Description**

Return the number of patterns in the dataset

**Usage**

```r
npatterns(x)
```

**Arguments**

- `x` the dataset.
Examples
   data(test_dataset)
   npatterns(test_dataset)

Description
   Return the number of samples in the dataset.

Usage
   nsamples(x)

Arguments
   x   A TRONCO compliant dataset.

Value
   The number of samples in the dataset.

Examples
   data(test_dataset)
   nsamples(test_dataset)

Description
   Return the number of types in the dataset.

Usage
   ntypes(x)

Arguments
   x   A TRONCO compliant dataset.
Value

The number of types in the dataset.

Examples

```r
data(test_dataset)
ntypes(test_dataset)
```

Description

`oncoPrint` : plot a genotype. For details and examples regarding the visualization through onco-
prints, we refer to the Vignette Section 4.4.

Usage

```r
oncoprint(
  x,
  excl.sort = TRUE,
  samples.cluster = FALSE,
  genes.cluster = FALSE,
  file = NA,
  ann.stage = has.stages(x),
  ann.hits = TRUE,
  stage.color = "YlOrRd",
  hits.color = "Purples",
  null.color = "lightgray",
  border.color = "white",
  text.cex = 1,
  font.column = NA,
  font.row = NA,
  title = as.description(x),
  sample.id = FALSE,
  hide.zeros = FALSE,
  legend = TRUE,
  legend.cex = 0.5,
  cellwidth = NA,
  cellheight = NA,
  group.by.label = FALSE,
  group.by.stage = FALSE,
  group.samples = NA,
  gene.annot = NA,
  gene.annot.color = "Set1",
  show.patterns = FALSE,
)```
annotate.consolidate.events = FALSE,
txt.stats = paste(nsamples(x), " samples
", nevents(x), " events
", ngenes(x), " genes
", npatterns(x), " patterns", sep = ""),
gtable = FALSE,
...
)

Arguments

x A TRONCO compliant dataset
excl.sort Boolean value, if TRUE sorts samples to enhance exclusivity of alterations
samples.cluster Boolean value, if TRUE clusters samples (columns). Default FALSE
genes.cluster Boolean value, if TRUE clusters genes (rows). Default FALSE
file If not NA write to file the Oncoprint, default is NA (just visualization).
ann.stage Boolean value to annotate stage classification, default depends on x
ann.hits Boolean value to annotate the number of events in each sample, default is TRUE
stage.color RColorbrewer palette to color stage annotations. Default is 'YlOrRd'
hits.color RColorbrewer palette to color hits annotations. Default is 'Purples'
null.color Color for the Oncoprint cells with 0s, default is 'lightgray'
border.color Border color for the Oncoprint, default is white' (no border)
text.cex Title and annotations cex, multiplied by font size 7
font.column If NA, half of font.row is used
font.row If NA, max(c(15 * exp(-0.02 * nrow(data)), 2)) is used, where data is the data visualized in the Oncoprint
title Oncoprint title, default is as.name(x) - see as_name
sample.id If TRUE shows samples name (columns). Default is FALSE
hide.zeroses If TRUE trims data - see trim - before plot. Default is FALSE
legend If TRUE shows a legend for the types of events visualized. Default is TRUE
legend.cex Default 0.5; determines legend size if legend = TRUE
cellwidth Default NA, sets autoscale cell width
cellheight Default NA, sets autoscale cell height
group.by.label Sort samples (rows) by event label - usefull when multiple events per gene are available

group.by.stage Default FALSE; sort samples by stage.
group.samples If this samples -> group map is provided, samples are grouped as of groups and sorted according to the number of mutations per sample - usefull when data was clustered
gene.annot Genes’ groups, e.g. list(RAF=c('KRAS','NRAS'), Wnt=c('APC', 'CTNNB1')). Default is NA.
oncoprint.cbio

Description

export input for cbio visualization at http://www.cbioportal.org/public-portal/oncoprinter.jsp

Usage

oncoprint.cbio(
  x,
  file = "oncoprint-cbio.txt",
  hom.del = "Homozygous Loss",
  het.loss = "Heterozygous Loss",
  gain = "Low-level Gain",
  amp = "High-level Gain"
)

Arguments

  x                A TRONCO compliant dataset.
  file             name of the file where to save the output
  hom.del          type of Homozygous Deletion
  het.loss         type of Heterozygous Loss
  gain             type of Gain
  amp              type of Amplification

Value

A file containing instruction for the CBio visualization Tool

Examples

data(crc_gistic)
gistic = import.GISTIC(crc_gistic)
oncoprint.cbio(gistic)
Description
OR hypothesis

Usage
OR(…)

Arguments
… Atoms of the soft exclusive pattern given either as labels or as partially lifted vectors.

Value
Vector to be added to the lifted genotype resolving the soft exclusive pattern

Description
Sort the internal genotypes according to event frequency.

Usage
order.frequency(x, decreasing = TRUE)

Arguments
x A TRONCO compliant dataset.
decreasing Inverse order. Default TRUE

Value
A TRONCO compliant dataset with the internal genotypes sorted according to event frequency.

Examples
data(test_dataset)
order.frequency(test_dataset)
### pathway.visualization

**Description**

Visualise pathways informations

**Usage**

```r
pathway.visualization(
    x,
    title = paste("Pathways:", paste(names(pathways), collapse = ", ", sep = "")),
    file = NA,
    pathways.color = "Set2",
    aggregate.pathways,
    pathways,
    ...
)
```

**Arguments**

- `x` A TRONCO compliant dataset
- `title` Plot title
- `file` To generate a PDF a filename have to be given
- `pathways.color` A RColorBrewer color palette
- `aggregate.pathways` Boolean parameter
- `pathways` Pathways
- `...` Additional parameters

**Value**

plot information

---

### pheatmap

**Description**

A function to draw clustered heatmaps where one has better control over some graphical parameters such as cell size, etc.

**Function**

A function to draw clustered heatmaps.
Usage

pheatmap(
  mat,
  color = colorRampPalette(rev(brewer.pal(n = 7, name = "RdYlBu")))((100),
  kmeans_k = NA,
  breaks = NA,
  border_color = "grey60",
  cellwidth = NA,
  cellheight = NA,
  scale = "none",
  cluster_rows = TRUE,
  cluster_cols = TRUE,
  clustering_distance_rows = "euclidean",
  clustering_distance_cols = "euclidean",
  clustering_method = "complete",
  cutree_rows = NA,
  cutree_cols = NA,
  treeheight_row = ifelse(cluster_rows, 50, 0),
  treeheight_col = ifelse(cluster_cols, 50, 0),
  legend = TRUE,
  legend_breaks = NA,
  legend_labels = NA,
  annotation_row = NA,
  annotation_col = NA,
  annotation = NA,
  annotation_colors = NA,
  annotation_legend = TRUE,
  drop_levels = TRUE,
  show_rownames = TRUE,
  show_colnames = TRUE,
  main = NA,
  fontsize = 10,
  fontsize_row = fontsize,
  fontsize_col = fontsize,
  display_numbers = FALSE,
  number_format = "%2f",
  number_color = "grey30",
  fontsize_number = 0.8 * fontsize,
  gaps_row = NULL,
  gaps_col = NULL,
  labels_row = NULL,
  labels_col = NULL,
  filename = NA,
  width = NA,
  height = NA,
  silent = FALSE,
  legend.cex = 1,
  txt.stats = NA,
Arguments

mat numeric matrix of the values to be plotted.

color vector of colors used in heatmap.

kmeans_k the number of kmeans clusters to make, if we want to aggregate the rows before drawing heatmap. If NA then the rows are not aggregated.

breaks a sequence of numbers that covers the range of values in mat and is one element longer than color vector. Used for mapping values to colors. Useful, if needed to map certain values to certain colors, to certain values. If value is NA then the breaks are calculated automatically.

border_color color of cell borders on heatmap, use NA if no border should be drawn.

cellwidth individual cell width in points. If left as NA, then the values depend on the size of plotting window.

cellheight individual cell height in points. If left as NA, then the values depend on the size of plotting window.

scale character indicating if the values should be centered and scaled in either the row direction or the column direction, or none. Corresponding values are "row", "column" and "none"

cluster_rows boolean values determining if rows should be clustered.

cluster_cols boolean values determining if columns should be clustered.

clustering_distance_rows distance measure used in clustering rows. Possible values are "correlation" for Pearson correlation and all the distances supported by dist, such as "euclidean", etc. If the value is none of the above it is assumed that a distance matrix is provided.

clustering_distance_cols distance measure used in clustering columns. Possible values the same as for clustering_distance_rows.

clustering_method clustering method used. Accepts the same values as hclust.

cutree_rows number of clusters the rows are divided into, based on the hierarchical clustering (using cutree), if rows are not clustered, the argument is ignored

cutree_cols similar to cutree_rows, but for columns

treeheight_row the height of a tree for rows, if these are clustered. Default value 50 points.

treeheight_col the height of a tree for columns, if these are clustered. Default value 50 points.

legend logical to determine if legend should be drawn or not.

legend_breaks vector of breakpoints for the legend.

legend_labels vector of labels for the legend_breaks.
annotation_row  data frame that specifies the annotations shown on left side of the heatmap. Each row defines the features for a specific row. The rows in the data and in the annotation are matched using corresponding row names. Note that color schemes takes into account if variable is continuous or discrete.

annotation_col  similar to annotation_row, but for columns.

annotation  deprecated parameter that currently sets the annotation_col if it is missing

annotation_colors  list for specifying annotation_row and annotation_col track colors manually. It is possible to define the colors for only some of the features. Check examples for details.

annotation_legend  boolean value showing if the legend for annotation tracks should be drawn.

drop_levels  logical to determine if unused levels are also shown in the legend

show_rownames  boolean specifying if column names are be shown.

show_colnames  boolean specifying if column names are be shown.

main  the title of the plot

fontsize  base fontsize for the plot

fontsize_row  fontsize for rownames (Default: fontsize)

fontsize_col  fontsize for colnames (Default: fontsize)

display_numbers  logical determining if the numeric values are also printed to the cells. If this is a matrix (with same dimensions as original matrix), the contents of the matrix are shown instead of original values.

number_format  format strings (C printf style) of the numbers shown in cells. For example "%.2f" shows 2 decimal places and "%.1e" shows exponential notation (see more in sprintf).

number_color  color of the text

fontsize_number  fontsize of the numbers displayed in cells

gaps_row  vector of row indices that show where to put gaps into heatmap. Used only if the rows are not clustered. See cutree_row to see how to introduce gaps to clustered rows.

gaps_col  similar to gaps_row, but for columns.

labels_row  custom labels for rows that are used instead of rownames.

labels_col  similar to labels_row, but for columns.

filename  file path where to save the picture. Filetype is decided by the extension in the path. Currently following formats are supported: png, pdf, tiff, bmp, jpeg. Even if the plot does not fit into the plotting window, the file size is calculated so that the plot would fit there, unless specified otherwise.

width  manual option for determining the output file width in inches.

height  manual option for determining the output file height in inches.

silent  do not draw the plot (useful when using the gtable output)
legend.cex  Default 0.5; determines legend size if legend = TRUE

txt.stats  By default, shows a summary statistics for shown data (n,m, |G| and |P|)

Details

The function also allows to aggregate the rows using kmeans clustering. This is advisable if number of rows is so big that R cannot handle their hierarchical clustering anymore, roughly more than 1000. Instead of showing all the rows separately one can cluster the rows in advance and show only the cluster centers. The number of clusters can be tuned with parameter kmeans_k.

This is a modified version of the original pheatmap (https://cran.r-project.org/web/packages/pheatmap/index.html) edited in accordance with GPL-2.

Value

Invisibly a list of components

- tree_row the clustering of rows as hclust object
- tree_col the clustering of columns as hclust object
- kmeans the kmeans clustering of rows if parameter kmeans_k was specified

Author(s)

Raivo Kolde <rkolde@gmail.com>

Examples

# Create test matrix
test = matrix(rnorm(200), 20, 10)
test[1:10, seq(1, 10, 2)] = test[1:10, seq(1, 10, 2)] + 3
test[11:20, seq(2, 10, 2)] = test[11:20, seq(2, 10, 2)] + 2
test[15:20, seq(2, 10, 2)] = test[15:20, seq(2, 10, 2)] + 4
colnames(test) = paste("Test", 1:10, sep = "")
rownames(test) = paste("Gene", 1:20, sep = "")

# Draw heatmaps
pheatmap(test)

Description

Return the first n recurrent events
rename.gene

Usage

\texttt{rank.recurrents(x, n)}

Arguments

\begin{itemize}
\item \texttt{x} \hspace{1em} A TRONCO compliant dataset.
\item \texttt{n} \hspace{1em} The number of events to rank
\end{itemize}

Value

the first \texttt{n} recurrent events

Examples

\begin{verbatim}
data(test_dataset)
dataset = rank.recurrents(test_dataset, 10)
\end{verbatim}

rename.gene

Usage

\texttt{rename.gene(x, old.name, new.name)}

Arguments

\begin{itemize}
\item \texttt{x} \hspace{1em} A TRONCO compliant dataset.
\item \texttt{old.name} \hspace{1em} The name of the gene to rename.
\item \texttt{new.name} \hspace{1em} The new name
\end{itemize}

Value

A TRONCO compliant dataset.

Examples

\begin{verbatim}
data(test_dataset)
test_dataset = rename.gene(test_dataset, 'TET2', 'gene x')
\end{verbatim}
rename.type

Description
Rename an event type

Usage
rename.type(x, old.name, new.name)

Arguments
- x: A TRONCO compliant dataset.
- old.name: The type of event to rename.
- new.name: The new name

Value
A TRONCO compliant dataset.

Examples
data(test_dataset)
test_dataset = rename.type(test_dataset, 'ins_del', 'deletion')

samples.selection

Description
Filter a dataset based on selected samples id

Usage
samples.selection(x, samples)

Arguments
- x: A TRONCO compliant dataset.
- samples: A list of samples

Value
A TRONCO compliant dataset.
Examples

data(test_dataset)
dataset = samples.selection(test_dataset, c('patient 1', 'patient 2'))

Description

Binds samples from one or more datasets, which must be defined over the same set of events

Usage

sbind(...)

Arguments

... the input datasets

Value

A TRONCO compliant dataset.

Description

Split cohort (samples) into groups, return either all groups or a specific group.

Usage

ssplit(x, clusters, idx = NA)

Arguments

x A TRONCO compliant dataset.
clusters A list of clusters. Rownames must match samples list of x
idx ID of a specific group present in stages. If NA all groups will be extracted

Value

A TRONCO compliant dataset.
**stage**  

*Stage information for test_dataset*

---

**Description**

This dataset contains stage information for patient in test_dataset

**Usage**

`data(stage)`

**Format**

Vector of stages

**Value**

A list of stages

**Author(s)**

Luca De Sano

**Source**

fake data

---

**TCGA.map.clinical.data**

*TCGA.map.clinical.data*

---

**Description**

Map clinical data from the TCGA format

**Usage**

`TCGA.map.clinical.data(file, sep = "\t", column.samples, column.map)`

**Arguments**

- `file`: A file with the clinical data
- `sep`: file delimiter
- `column.samples`: Required columns
- `column.map`: Map to the required columns
TCGA.remove.multiple.samples

Description
Check if there are multiple sample in x, according to TCGA barcodes naming

Usage
TCGA.remove.multiple.samples(x)

Arguments
x A TRONCO compliant dataset.

Value
A TRONCO compliant dataset

Examples
data(test_dataset)
TCGA.remove.multiple.samples(test_dataset)

TCGA.remove.multiple.samples

Description
If there are multiple sample in x, according to TCGA barcodes naming, remove them

Usage
TCGA.remove.multiple.samples(x)

Arguments
x A TRONCO compliant dataset.

Value
A TRONCO compliant dataset
Examples

```r
data(test_dataset)
TCGA.remove.multiple.samples(test_dataset)
```

---

**Description**

Keep only the first 12 character of samples barcode if there are no duplicates

**Usage**

```r
TCGA.shorten.barcodes(x)
```

**Arguments**

|x| A TRONCO compliant dataset.

**Value**

A TRONCO compliant dataset

**Examples**

```r
data(test_dataset)
TCGA.shorten.barcodes(test_dataset)
```

---

**test_dataset**

*A complete dataset with hypotheses*

---

**Description**

This dataset contains a complete test dataset

**Usage**

```r
data(test_dataset)
```

**Format**

TRONCO compliant dataset
Value

A standard TRONCO object

Author(s)

Luca De Sano

Source

fake data

test_dataset_no_hypos  A complete dataset

Description

This dataset contains a complete test dataset

Usage

data(test_dataset_no_hypos)

Format

TRONCO compliant dataset

Value

A standard TRONCO object

Author(s)

Luca De Sano

Source

fake data
**test_model**  

**Description**  
This dataset contains a model reconstructed with CAPRI

**Usage**  
data(test_model)

**Format**  
TRONCO compliant dataset

**Value**  
A standard TRONCO object

**Author(s)**  
Luca De Sano

**Source**  
fake data

**test_model_kfold**  

**Description**  
This dataset contains a model reconstructed with CAPRI

**Usage**  
data(test_model_kfold)

**Format**  
TRONCO compliant dataset

**Value**  
A standard TRONCO object
Author(s)
Luca De Sano

Source
fake data

trim

Description
Deletes all events which have frequency 0 in the dataset.

Usage
trim(x)

Arguments
x A TRONCO compliant dataset.

Value
A TRONCO compliant dataset.

Examples
data(test_dataset)
test_dataset = trim(test_dataset)

tronco.bootstrap

Description
Bootstrap a reconstructed progression model. For details and examples regarding the statistical
assessment of an inferred model, we refer to the Vignette Section 7.

Usage
tronco.bootstrap(
    reconstruction,
    type = "non-parametric",
    nboot = 100,
    cores.ratio = 1,
    silent = FALSE
)
tronco.caprese

**Arguments**

- `reconstruction` The output of tronco.capri or tronco.caprese
- `type` Parameter to define the type of sampling to be performed, e.g., non-parametric for uniform sampling.
- `nboot` Number of bootstrap sampling to be performed when estimating the model confidence.
- `cores.ratio` Percentage of cores to use coresRate * (numCores - 1)
- `silent` A parameter to disable/enable verbose messages.

**Value**

A TRONCO compliant object with reconstructed model

**Examples**

```r
data(test_model)
boot = tronco.bootstrap(test_model, nboot = 1, cores.ratio = 0)
```

---

**tronco.caprese**

**tronco caprese**

**Description**

Reconstruct a progression model using CAPRESE algorithm. For details and examples regarding the inference process and on the algorithm implemented in the package, we refer to the Vignette Section 6.

**Usage**

tronco.caprese(data, lambda = 0.5, silent = FALSE, epos = 0, eneg = 0)

**Arguments**

- `data` A TRONCO compliant dataset.
- `lambda` Coefficient to combine the raw estimate with a correction factor into a shrinkage estimator.
- `silent` A parameter to disable/enable verbose messages.
- `epos` Error rate of false positive errors.
- `eneg` Error rate of false negative errors.

**Value**

A TRONCO compliant object with reconstructed model
Examples

data(test_dataset_no_hypos)
recon = tronco.caprese(test_dataset_no_hypos)

---

tronco.capri    tronco capri

Description

Reconstruct a progression model using CAPRI algorithm. For details and examples regarding the inference process and on the algorithm implemented in the package, we refer to the Vignette Section 6.

Usage

tronco.capri(
    data,
    command = "hc",
    regularization = c("bic", "aic"),
    do.boot = TRUE,
    nboot = 100,
    pvalue = 0.05,
    min.boot = 3,
    min.stat = TRUE,
    boot.seed = NULL,
    silent = FALSE,
    epos = 0,
    eneg = 0,
    restart = 100
)

Arguments

data         A TRONCO compliant dataset.
command      Parameter to define to heuristic search to be performed. Hill Climbing and Tabu search are currently available.
regularization Select the regularization for the likelihood estimation, e.g., BIC, AIC.
do.boot      A parameter to disable/enable the estimation of the error rates given the reconstructed model.
nboot        Number of bootstrap sampling (with rejection) to be performed when estimating the selective advantage scores.
pvalue       Pvalue to accept/reject the valid selective advantage relations.
min.boot     Minimum number of bootstrap sampling to be performed.
min.stat A parameter to disable/enable the minimum number of bootstrap sampling required besides nboot if any sampling is rejected.

boot.seed Initial seed for the bootstrap random sampling.

silent A parameter to disable/enable verbose messages.

epos Error rate of false positive errors.

eneg Error rate of false negative errors.

restart An integer, the number of random restarts.

Value

A TRONCO compliant object with reconstructed model

Examples

data(test_dataset)
recon = tronco.capri(test_dataset, nboot = 1)

data tronco.chowliu Tronco Chow Liu

Description

Reconstruct a progression model using Chow Liu algorithm combined with probabilistic causation. For details and examples regarding the inference process and on the algorithm implemented in the package, we refer to the Vignette Section 6.

Usage

tronco.chowliu(
    data,
    regularization = c("bic", "aic"),
    do.boot = TRUE,
    nboot = 100,
    pvalue = 0.05,
    min.boot = 3,
    min.stat = TRUE,
    boot.seed = NULL,
    silent = FALSE,
    epos = 0,
    eneg = 0
)
Arguments

data A TRONCO compliant dataset.
regularization Select the regularization for the likelihood estimation, e.g., BIC, AIC.
do.boot A parameter to disable/enable the estimation of the error rates given the reconstructed model.
nboot Number of bootstrap sampling (with rejection) to be performed when estimating the selective advantage scores.
pvalue Pvalue to accept/reject the valid selective advantage relations.
min.boot Minimum number of bootstrap sampling to be performed.
min.stat A parameter to disable/enable the minimum number of bootstrap sampling required besides nboot if any sampling is rejected.
boot.seed Initial seed for the bootstrap random sampling.
silent A parameter to disable/enable verbose messages.
epos Error rate of false positive errors.
eneg Error rate of false negative errors.

Value

A TRONCO compliant object with reconstructed model

Examples

data(test_dataset_no_hypos)
recon = tronco.chowliu(test_dataset_no_hypos, nboot = 1)

Description

Reconstruct a progression model using Edmonds algorithm combined with probabilistic causation. For details and examples regarding the inference process and on the algorithm implemented in the package, we refer to the Vignette Section 6.

Usage

tronco.edmonds(
data,
regularization = "no_reg",
score = "pmi",
do.boot = TRUE,
nboot = 100,
pvalue = 0.05,
data (test_dataset_no_hypos)
recon = tronco.edmonds(test_dataset_no_hypos, nboot = 1)
tronco.gabow

Description

Reconstruct a progression model using Gabow algorithm combined with probabilistic causation. For details and examples regarding the inference process and on the algorithm implemented in the package, we refer to the Vignette Section 6.

Usage

```
tronco.gabow(
  data,
  regularization = "no_reg",
  score = "pmi",
  do.boot = TRUE,
  nboot = 100,
  pvalue = 0.05,
  min.boot = 3,
  min.stat = TRUE,
  boot.seed = NULL,
  silent = FALSE,
  epos = 0,
  eneg = 0,
  do.raising = TRUE
)
```

Arguments

data A TRONCO compliant dataset.
regularization Select the regularization for the likelihood estimation, e.g., BIC, AIC.
score Select the score for the estimation of the best tree, e.g., pointwise mutual information (pmi), conditional entropy (entropy).
do.boot A parameter to disable/enable the estimation of the error rates given the reconstructed model.
nboot Number of bootstrap sampling (with rejection) to be performed when estimating the selective advantage scores.
pvalue Pvalue to accept/reject the valid selective advantage relations.
min.boot Minimum number of bootstrap sampling to be performed.
min.stat A parameter to disable/enable the minimum number of bootstrap sampling required besides nboot if any sampling is rejected.
boot.seed Initial seed for the bootstrap random sampling.
silent A parameter to disable/enable verbose messages.
epos Error rate of false positive errors.
eneg Error rate of false negative errors.
do.raising Whether to use or not the raising condition as a prior.
Value

A TRONCO compliant object with reconstructed model

Examples

data(test_dataset_no_hypos)
recon = tronco.gabow(test_dataset_no_hypos, nboot = 1)

Description

Perform a k-fold cross-validation using the function bn.cv to estimate the entropy loss. For details and examples regarding the statistical assessment of an inferred model, we refer to the Vignette Section 7.

Usage

tronco.kfold.eloss(
  x,
  models = names(as.models(x)),
  runs = 10,
  k = 10,
  silent = FALSE
)

Arguments

x A reconstructed model (the output of tronco.capri or tronco.caprese)
models The names of the selected regularizers (bic, aic or caprese)
runs a positive integer number, the number of times cross-validation will be run
k a positive integer number, the number of groups into which the data will be split
silent A parameter to disable/enable verbose messages.

Examples

data(test_model)
tronco.kfold.eloss(test_model, k = 2, runs = 2)
tronco.kfold.posterr

For details and examples regarding the statistical assessment of an inferred model, we refer to the Vignette Section 7.

Description

Perform a k-fold cross-validation using the function bn.cv and scan every node to estimate its posterior classification error.

Usage

tronco.kfold.posterr(
  x,
  models = names(as.models(x)),
  events = as.events(x),
  runs = 10,
  k = 10,
  cores.ratio = 1,
  silent = FALSE
)

Arguments

x  A reconstructed model (the output of tronco.capri)
models The names of the selected regularizers (bic, aic or caprese)
events a list of event
runs a positive integer number, the number of times cross-validation will be run
k a positive integer number, the number of groups into which the data will be split
cores.ratio Percentage of cores to use. coresRate * (numCores - 1)
silent A parameter to disable/enable verbose messages.

Examples

data(test_model)
tronco.kfold.posterr(test_model, k = 2, runs = 2, cores.ratio = 0)
Description

Perform a k-fold cross-validation using the function bn.cv and scan every node to estimate its prediction error. For details and examples regarding the statistical assessment of an inferred model, we refer to the Vignette Section 7.

Usage

tronco.kfold.prederr(
  x,
  models = names(as.models(x)),
  events = as.events(x),
  runs = 10,
  k = 10,
  cores.ratio = 0,
  silent = FALSE
)

Arguments

x  A reconstructed model (the output of tronco.capri)
models The names of the selected regularizers (bic, aic or caprese)
events a list of event
runs a positive integer number, the number of times cross-validation will be run
k a positive integer number, the number of groups into which the data will be split
cores.ratio Percentage of cores to use. coresRate * (numCores - 1)
silent A parameter to disable/enable verbose messages.

Examples

data(test_model)
tronco.kfold.prederr(test_model, k = 2, runs = 2, cores.ratio = 0)
Description

tronco.pattern.plot: plot a genotype

Usage

tronco.pattern.plot(
  x,
  group = as.events(x),
  to,
  gap.cex = 1,
  legend.cex = 1,
  label.cex = 1,
  title = paste(to[1], to[2]),
  mode = "barplot"
)

Arguments

x A TRONCO compliant dataset
group A list of events (see as.events() for details)
to A target event
gap.cex cex parameter for gap
legend.cex cex parameter for legend
label.cex cex parameter for label
title title
mode can be ‘circos’ or ‘barplot’

Description

Plots a progression model from a reconstructed dataset. For details and examples regarding the visualization of an inferred model, we refer to the Vignette Section 7.
Usage

```r
tronco.plot(
  x,
  models = names(x$model),
  fontsize = NA,
  height = 2,
  width = 3,
  height.logic = 1,
  pf = FALSE,
  disconnected = FALSE,
  scale.nodes = NA,
  title = as.description(x),
  confidence = NA,
  p.min = 0.05,
  legend = TRUE,
  legend.cex = 1,
  edge.cex = 1,
  label.edge.size = NA,
  expand = TRUE,
  genes = NULL,
  relations.filter = NA,
  edge.color = "black",
  pathways.color = "Set1",
  file = NA,
  legend.pos = "bottom",
  pathways = NULL,
  lwd = 3,
  samples.annotation = NA,
  export.igraph = FALSE,
  create.new.dev = TRUE,
  ...
)
```

Arguments

- **x**: A reconstructed model (the output of the inference by a tronco function)
- **models**: A vector containing the names of the algorithms used (caprese, capri_bic, etc)
- **fontsize**: For node names. Default NA for automatic rescaling
- **height**: Proportion node height - node width. Default height 2
- **width**: Proportion node height - node width. Default width 2
- **height.logic**: Height of logical nodes. Default 1
- **pf**: Should I print Prima Facie? Default False
- **disconnected**: Should I print disconnected nodes? Default False
- **scale.nodes**: Node scaling coefficient (based on node frequency). Default NA (autoscale)
- **title**: Title of the plot. Default as.description(x)
confidence Should I add confidence informations? No if NA
p.min p-value cutoff. Default automatic
legend Should I visualise the legend?
legend.cex CEX value for legend. Default 1.0
edge.cex CEX value for edge labels. Default 1.0
label.edge.size Size of edge labels. Default NA for automatic rescaling
expand Should I expand hypotheses? Default TRUE
genes Visualise only genes in this list. Default NULL, visualise all.
relations.filter Filter relations to dispaly according to this functions. Default NA
edge.color Edge color. Default 'black'
pathways.color RColorBrewer colorser for patways. Default 'Set1'.
file String containing filename for PDF output. If NA no PDF output will be provided
legend.pos Legend position. Default 'bottom',
pathways A vector containing pathways information as described in as.patterns()
lwd Edge base lwd. Default 3
samples.annotation = List of samples to search for events in model
export.igraph If TRUE export the generated igraph object
create.new.dev If TRUE create a new graphical device when calling trono.plot. Set this to FALSE, e.g., if you do not wish to create a new device when executing the command with export.igraph = TRUE
... Additional arguments for RGraphviz plot function

Value
Information about the reconstructed model

Examples

data(test_model)
tronco.plot(test_model)
Description

Reconstruct a progression model using Prim algorithm combined with probabilistic causation. For details and examples regarding the inference process and on the algorithm implemented in the package, we refer to the Vignette Section 6.

Usage

```r
tronco.prim(
  data,  
  regularization = "no_reg",  
  do.boot = TRUE,  
  nboot = 100,  
  pvalue = 0.05,  
  min.boot = 3,  
  min.stat = TRUE,  
  boot.seed = NULL,  
  silent = FALSE,  
  epos = 0,  
  eneg = 0  
)
```

Arguments

data A TRONCO compliant dataset.
regularization Select the regularization for the likelihood estimation, e.g., BIC, AIC.
do.boot A parameter to disable/enable the estimation of the error rates given the reconstructed model.
nboot Number of bootstrap sampling (with rejection) to be performed when estimating the selective advantage scores.
pvalue P-value to accept/reject the valid selective advantage relations.
min.boot Minimum number of bootstrap sampling to be performed.
min.stat A parameter to disable/enable the minimum number of bootstrap sampling required besides nboot if any sampling is rejected.
boot.seed Initial seed for the bootstrap random sampling.
silent A parameter to disable/enable verbose messages.
epos Error rate of false positive errors.
eneg Error rate of false negative errors.

Value

A TRONCO compliant object with reconstructed model
Examples

data(test_dataset_no_hypos)
recon = tronco.prim(test_dataset_no_hypos, nboot = 1)

view

view

Description

Print to console a short report of a dataset 'x', which should be a TRONCO compliant dataset - see is.compliant.

Usage

view(x, view = 5)

Arguments

x
view

A TRONCO compliant dataset.
The first view events are shown via head.

Examples

data(test_dataset)
view(test_dataset)

which.samples

which.samples

Description

Return a list of samples with specified alteration

Usage

which.samples(x, gene, type, neg = FALSE)

Arguments

x

gene
type

A TRONCO compliant dataset.
A list of gene names
A list of types

neg

If FALSE return the list, if TRUE return as.samples() - list
**Value**

A list of sample

**Examples**

```r
data(test_dataset)
which.samples(test_dataset, 'TET2', 'ins_del')
which.samples(test_dataset, 'TET2', 'ins_del', neg=TRUE)
```

**Description**

XOR hypothesis

**Usage**

`XOR(...)`

**Arguments**

... Atoms of the hard exclusive pattern given either as labels or as partially lifted vectors.

**Value**

Vector to be added to the lifted genotype resolving the hard exclusive pattern
Index

aCML, 5
AND, 5
annotate.description, 6
annotate.stages, 6
as.adj.matrix, 7
as.alterations, 8
as.bootstrap.scores, 8
as.colors, 9
as.conditional.probs, 10
as.confidence, 10
as.description, 11
as.events, 12
as.events.in.patterns, 12
as.events.in.sample, 13
as.gene, 14
as.genes, 14
as.genes.in.patterns, 15
as.genotypes, 16
as.hypotheses, 16
as.joint.probs, 17
as.kfold.eloss, 18
as.kfold.posterr, 18
as.kfold.prederr, 19
as.marginal.probs, 20
as.models, 21
as.parameters, 21
as.pathway, 22
as.patterns, 23
as.samples, 23
as.selective.advantage.relations, 24
as.stages, 25
as.types, 25
as.types.in.patterns, 26
change.color, 27
consolidate.data, 27
crc_gistic, 28
crc_maf, 29
crc_plain, 29
delete.event, 30
delete.gene, 31
delete.hypothesis, 31
delete.model, 32
delete.pattern, 33
delete.samples, 33
delete.type, 34
dist, 63
duplicates, 34
ebind, 35
enforce.numeric, 35
enforce.string, 36
events.selection, 36
export.graphml, 37
export.mutex, 38
export.nbs.input, 39
extract.MAF.HuGO.Entrez.map, 39
genes.table.report, 40
gpar, 65
ggrid.text, 65
has.duplicates, 41
has.model, 41
has.stages, 42
hclust, 63, 65
hypothesis.add, 42
hypothesis.add.group, 43
hypothesis.add.homologous, 44
import.genotypes, 45
import.GISTIC, 45
import.MAF, 46
import.model, 48
import.mutex.groups, 48
intersect.datasets, 49
is.compliant, 49
join.events, 50
join.types, 51
keysToNames, 51
maf, 52
muts, 53
nameToKey, 53
nevents, 54
ngenes, 54
nhypotheses, 55
npatterns, 55
nsamples, 56
ntypes, 56
oncoprint, 57
oncoprint.cbio, 59
OR, 60
order.frequency, 60
pathway.visualization, 61
pheatmap, 61
rank.recurrents, 65
rename.gene, 66
rename.type, 67
samples.selection, 67
sbind, 68
sprintf, 64
ssplit, 68
stage, 69
TCGA.map.clinical.data, 69
TCGA.multiple.samples, 70
TCGA.remove.multiple.samples, 70
TCGA.shorten.barcodes, 71
test_dataset, 71
test_dataset_no_hypos, 72
test_model, 73
test_model_kfold, 73
trim, 74
tronco.bootstrap, 74
tronco.caprese, 75
tronco.capri, 76
tronco.chowliu, 77
tronco.edmonds, 78
tronco.gabow, 80
tronco.kfold.eloss, 81
tronco.kfold.posterr, 82
tronco.kfold.prederr, 83
tronco.pattern.plot, 84
tronco.plot, 84
tronco.prim, 87
view, 88
which.samples, 88
XOR, 89