# Package 'TRONCO'

October 9, 2015

**Version** 1.0.0

**Date** 2014-09-22

**Title** TRONCO, a package for TRanslational ONCOlogy

**Author** Marco Antoniotti, Giulio Caravagna,
Alex Graudenzi, Ilya Korsunsky,
Mattia Longoni, Loes Olde Loohuis,
Giancarlo Mauri, Bud Mishra, Daniele Ramazzotti

**Maintainer** Giulio Caravagna <`giulio.caravagna@disco.unimib.it`>,
Alex Graudenzi <`alex.graudenzi@disco.unimib.it`>,
Daniele Ramazzotti <`daniele.ramazzotti@disco.unimib.it`>

**Depends** R (>= 2.10), methods, Rgraphviz, lattice, graph

**Description** Genotype-level cancer progression models describe the ordering of accumulating mutations, e.g., somatic mutations / copy number variations, during cancer development. These graphical models help understand the causal structure involving events promoting cancer progression, possibly predicting complex patterns characterising genomic progression of a cancer. Reconstructed models can be used to better characterise genotype-phenotype relation, and suggest novel targets for therapy design. TRONCO (TRanslational ONCOlogy) is a R package aimed at collecting state-of-the-art algorithms to infer progression models from cross-sectional data, i.e., data collected from independent patients which does not necessarily incorporate any evident temporal information. These algorithms require a binary input matrix where: (i) each row represents a patient genome, (ii) each column an event relevant to the progression (a priori selected) and a 0/1 value models the absence/presence of a certain mutation in a certain patient. The current first version of TRONCO implements the CAPRESE algorithm (Cancer PRogression Extraction with Single Edges) to infer possible progression models arranged as trees; cfr. Inferring tree causal models of cancer progression with probability raising, L. Olde Loohuis, G. Caravagna, A. Graudenzi, D. Ramazzotti, G. Mauri, M. Antoniotti and B. Mishra. PLoS One, to appear. This vignette shows how to use TRONCO to infer a tree model of ovarian cancer progression from CGH data of copy number alterations (classified as gains or losses over chromosome's arms). The dataset used is available in the SKY/M-FISH database.

**License** EPL (>= 1.0)

**URL** http://bimib.disco.unimib.it

**biocViews** Cancer

**Suggests** RUnit, BiocGenerics

**NeedsCompilation** no

## R topics documented:

---

`confidence` *provides various kinds of confidence measures for an inferred progression model*

---

### Description

A set of functions to visualise and compare the probability of each event in the progression model, as well as their joint and conditional distributions. These can be evaluated both in the data (observed probabilities) and in the reconstructed model (fitted probabilities).

### Usage

```
confidence.data.joint(topology)

confidence.fit.joint(topology)

confidence.data.single(topology)

confidence.fit.single(topology)

confidence.data.conditional(topology)
confidence.fit.conditional(topology)

confidence.single(topology)

confidence.joint(topology)

confidence.conditional(topology)
```

## Arguments

| | |
|---|---|
| `topology` | A topology returned by the reconstruction algorithm |

## Details

`confidence.data.joint` plot the pairwise observed joint probability of the events

`confidence.fit.joint` plot the pairwise fitted joint probability of the events

`confidence.data.single` plot the observed probability of each event

`confidence.fit.single` plot the fitted probability of each event

`confidence.data.conditional` plot the pairwise observed conditional probability of the events

`confidence.fit.conditional` plot the pairwise fitted conditional probability of the events

`confidence.single` plot the difference between the observed and fitted probability of each event

`confidence.joint` plot the pairwise difference between the observed and fitted joint probability of the events

`confidence.conditional` plot the pairwise difference between the observed and fitted conditional probability of the events

---

`data.load` *load a dataset (binary matrix) from a file or a preloaded dataset.*

---

## Description

`data.load` sets a global data frame ’data.values’ that contains the dataset loaded from an input file.

## Usage

```
data.load(data.input)
```

## Arguments

| | |
|---|---|
| `data.input` | The input file path. or a dataset loaded by `data` function |

## Details

`data.load` loads a dataset from disk and associates all columns in the dataset to a specified event. Thus, types and events must be specified before calling this function to ensure a consistency check is performed on the input dataset (see `types.load`, `types.add`, `events.load`, `events.add` to load/add types/events).

---

`events` *Events collection for Ovarian cancer CGH data*

---

## Description

This example contains a collection of events associeted to the Ovarian cancer CGH dataset

## Format

An example with 7 events

---

`events.add` *add a new event (e.g., a missense point mutation for EGFR)*

---

## Description

`events.add` sets a global data frame 'events' that contains all the events defined. Events can be added and refined incrementally, in any order.

## Usage

```
events.add(event.name, type.name, column.number = NA)
```

## Arguments

| | |
|---|---|
| `event.name` | The event label(e.g., 'EGFR') . All event labels are strings. |
| `type.name` | The type name of this event (e.g., 'missense point'). Type names must refer to types loaded before adding an event, a consistency check raises an error if the type name is unknown. |
| `column.number` | The dataset column to which this event is associated. Column number must be an integer positive value. |

## Details

`events.add` allows to define one event at a time. If the event was previously defined, its definition is updated to keep track of its last definition. A consistency check is performed to ensure that the type of defined event is valid. Thus, types must be defined before events are loaded (see `types.add`, `types.load`).

## Examples

```
types.add("gain", "red")
events.add("8q+", "gain", 1)
```

---

`events.load` *load a set of events from file*

---

## Description

`events.load` sets a global data frame ’events’ that contains all event definitions found in a specified file or dataset to be validated. This is a way to automatise calls to function `events.add` for a bunch of events.

## Usage

```
events.load(data.input)
```

## Arguments

| | |
|---|---|
| `data.input` | The input file path or a dataset to be validated. |

## Details

`events.load` load a set of events from a given file. The input file must be structured as a CSV file, where each event is defined on a separate line in the format: eventName, typeName, columnNumber.

## See Also

`events.add`

---

`ov.cgh` *Ovarian cancer CGH data*

---

## Description

This is a data set obtained using the comparative genomic hybridization technique (CGH) on samples from papillary serous cystadenocarcinoma of the ovary. Only the seven most commonly occurring events are given.

## Format

A data frame with 87 observations on 7 variables.

### Details

The CGH technique uses fluorescent staining to detect abnormal (increased or decreased) number of DNA copies. Often the results are reported as a gain or loss on a certain arm, without further distinction for specific regions. It is common to denote a change in DNA copy number on a specific chromosome arm by prefixing a "-" sign for decrease and a "+" for increase. Thus, say, -3q denotes abnormally low DNA copy number on the q arm of the 3rd chromosome.

### Source

http://www.ncbi.nlm.nih.gov/sky/

---

## `reset` *reset*

---

### Description

A set of functions to reset events, types and data.values variables

### Usage

```
reset.events()

reset.types()

reset()
```

### Details

`reset.events` Resets the events variable

`reset.types()` Resets the types variable

`reset()` Resets types, events and data.values variables

### Examples

```
reset.events()
reset.types()
reset()
```

## `tronco.bootstrap` *perform bootstrap algorithm*

### Description

`tronco.bootstrap` perform parametric and non-parametric bootstrap algorithms

### Usage

```
tronco.bootstrap(topology, lambda = 0.5, type = "non-parametric",
  nboot = 1000)
```

### Arguments

| | |
|---|---|
| `topology` | A topology returned by a reconstruction algorithm |
| `lambda` | A lambda value, default is 0.5 |
| `type` | The type of bootstrap performed, parametric and non parametric types are available. To specify wich type of bootstrap run type must be "parametric" or "non-parametric". |
| `nboot` | Samplig value. The grater will be the nboot value the logehr time the entire process will take to complete the computing |

### Value

A topology object with bootstrap informations added

## `tronco.bootstrap.show` *show bootstrapping results*

### Description

`tronco.bootstrap.show` show bootstrapping results. Requires that you already executed tronco.bootstrap

### Usage

```
tronco.bootstrap.show(topology)
```

### Arguments

| | |
|---|---|
| `topology` | A topology returned by a reconstruction algorithm |

## `tronco.caprese` *runs CAPRESE algorithm*

### Description

`tronco.caprese` executes the CAPRESE algorithm on the dataset `data.values` specified.

### Usage

```
tronco.caprese(dataset, lambda = 0.5, verbose = FALSE)
```

### Arguments

| | |
|---|---|
| `dataset` | The input dataset. Type: dataframe. The dataset given as input is the data.values data frame loaded by the `data` function. |
| `lambda` | the real positive value of the shrinkage coefficient, required to range in [0, 1]. Its default value is 0.5, if unspecified. |
| `verbose` | execute CAPRESE algorithm with verbose output to screen. Type: boolean, dafault: FALSE. |

### Details

`tronco.caprese` executes the reconstruction of the topology, and computesg all the confidence measures defined in `confidence`.

### Value

an object containing the reconstructed topology and confidence values.

### See Also

`data`

## `tronco.plot` *plot a progression model*

### Description

`tronco.plot` plots a progression model from a recostructed `topology`.

### Usage

```
tronco.plot(topology, title = paste("Progression model", topology@algorithm,
  sep = " "), title.color = "black", confidence = FALSE, legend = TRUE,
  legend.title = "Legend", legend.columns = 1, legend.inline = FALSE,
  legend.pos = "bottomright", legend.coeff = 1, label.coeff = 1,
  label.color = "black", label.edge.size = 12)
```

## Arguments

| | |
|---|---|
| `topology` | A topology returned by a reconstruction algorithm |
| `title` | plot Plot title (default "Progression model x", x reconstruction algorithm) |
| `title.color` | color title (default "black") |
| `confidence` | bool; plot edges according to confidence (default is f) |
| `legend` | bool; show/hide the legend (default is t) |
| `legend.title` | string; legend title (default is "Legend") |
| `legend.columns` | int; use 1 or 2 columns to plot the legend (default is 1) |
| `legend.inline` | bool; print inline legend (default is f) |
| `legend.pos` | string; legend positioning, available keywords "topleft", "topright", "bottom-left" and "bottomright" (default is "bottomright") |
| `legend.coeff` | double; size of the types label in the legend (default is 1) |
| `label.coeff` | double; size of the events label (default is 1) |
| `label.color` | color events label (default "black") |
| `label.edge.size` | double; size of the confidence label, when used (default is 12) |

## Examples

```
## Not run:
types.load("data/types.txt")
events.load("data/events.txt")
data.load("data/CGH.txt")
topology <- tronco.caprese(data.values)
tronco.plot(topology, legend.pos = "topleft", legend = TRUE, confidence = TRUE,
legend.col = 1, legend.coeff = 0.7, label.edge.size = 10, label.coeff = 0.7)

## End(Not run)
```

---

`types` *Types collection for Ovarian cancer CGH data*

---

## Description

This example contains a collection of types associeted to the Ovarian cancer CGH dataset

## Format

An example with 2 types

## `types.add` *add a new type of event (e.g., missense point mutation)*

### Description

`types.add` sets a global data frame 'types' that contains all types defined. Types can be added and refined incrementally, in any order.

### Usage

```
types.add(type.name, color.name)
```

### Arguments

| | |
|---|---|
| `type.name` | The type label. All type labels are strings. |
| `color.name` | The type color. All R's color definitions are allowed. |

### Details

`types.add` defines a type of event considered at a time. If the type was previously defined, its definition is updated to keep track of its last definition. A consistency check is performed to ensure that the type is valid. Types must be defined before events are loaded.

### Examples

```
types.add("gain", "red")
```

## `types.load` *load a set of types from file*

### Description

`types.load` sets a global data frame 'types' that contains all type definitions found in a specified file or dataset to be validated.

### Usage

```
types.load(data.input)
```

### Arguments

| | |
|---|---|
| `data.input` | The input file path or a dataset to be validated. |

## Details

`types.load` allows to load type definitions from a given file path. The file which contains all the definitions must be structured as a csv file. All definitions are couple of values type name and color name as shown below:

typeName, colorName ... , ...

## See Also

`types.add`

# Index