Package ‘gfpop’

February 14, 2019

Type Package

Title Graph-constrained Functional Pruning Optimal Partitioning

Version 0.1.2

Author Vincent Runge

Maintainer Vincent Runge <vincent.runge@univ-evry.fr>

Description Penalized parametric changepoint detection by functional pruning dynamic programming algorithm. The successive means can be constrained using a graph structure with edge of type null, up, down, std, absInf or absSup. To each edge we can use an additional nonnegative parameter allowing us to force a minimal gap between two successive means. The user can also constraint the infered means to lie between some minimal and maximal values. Data is modelized by a quadratic cost with possible use of a robust loss, biweight and Huber. In a next version of this package, other parametric losses will be available (L1, Poisson, binomial).

License GPL-3

Encoding UTF-8

LazyData true

Imports Rcpp (>= 0.12.18)

LinkingTo Rcpp

RoxygenNote 6.1.0

Suggests knitr,
rmarkdown

VignetteBuilder knitr

R topics documented:

`dataGenerator` *Gaussian data Generator*

Description

Generating data with given model = changepoint relative position + means + standard deviation

Usage

```
dataGenerator(n, changepoints, means, sigma = 1)
```

Arguments

`n`	number of data to generate
`changepoints`	vector of position of the changepoint in (0,1] (last element is always 1).
`means`	vector of means for the consecutive segments (same length as changepoints)
`sigma`	a positive number = the standard deviation of the data

Value

a vector of size n generated by the chosen model

Examples

```
dataGenerator(100, c(0.3, 0.6, 1), c(1, 2, 3))
```

`edge` *Edge generation*

Description

Edge creation for graph

Usage

```
edge(state1, state2, type = "null", penalty = 0, parameter = 0)
```

Arguments

`state1`	a nonnegative integer defining the starting state of the edge
`state2`	a nonnegative integer defining the ending state of the edge
`type`	a string equal to "null", "std", "up", "down", "absInf" or "absSup"
`penalty`	a nonnegative number. The penality associated to this state transition
`parameter`	a nonnegative number to constraint the size of the gap in the change of state

Value

a dataframe with five components equal to the five parameters

Examples

```
edge(0, 1, "up", 10, 1)
```

`gfpop` *Graph-constrained functional pruning optimal partitioning*

Description

Graph-contstrained functional pruning optimal partitionning

Usage

```
gfpop(vectData = c(0), vectWeight = c(0), mygraph, type = "gauss",
  K = Inf, a = 0, min = -Inf, max = Inf)
```

Arguments

`vectData`	vector of data to segment
`vectWeight`	vector of weights (positive numbers) same size as vectData
`mygraph`	dataframe of class graph to constraint the changepoint dynamic programming algorithm
`type`	a string defining the type of cost to use. "gauss", "poisson" or "binomial"
`K`	a positive number. Threshold for the Biweight robust loss
`a`	a positive number. Slope for the Huber robust loss
`min`	minimal bound for the infered means
`max`	maximal bound for the infered means

Value

a gfpop object = (changepoints, states, forced, means). 'changepoints' is the vector of changepoints (we give the last element of each segment). 'states' is the vector giving the state of each segment 'forced' is the vector specifying whether the constraints of the graph are active (=1) or not (=0) 'means' is the vector of successive means of each segment 'cost' is a number equal to the global cost of the graph-constrained segmentation graph *Graph generation*

Description

Graph creation

Usage

```
graph(..., penalty = 0, type = "empty")
```

Arguments

...	This is a list of edges definied by functions edge and StartEnd
penalty	a nonnegative number equals to the common penalty to use for all edges
type	a string equal to "std", "isotonic", "updown", "infsup". to build a predefined classic graph

Value

a dataframe with edges in rows (columns are named "state1", "state2", "type", "penalty", "parameter") with additional "graph" class.

Examples

```
UpDownGraph <- graph(penalty = 10, type = "updown")
MyGraph <- graph(edge(0,0), edge(1,1), edge(0,1,"up",10), edge(1,0,"down",0), StartEnd(0,0))
```

itergfpop *Graph-constrained functional pruning optimal partitioning iterated*

Description

Graph-contstrained functional pruning optimal partitionning iterated with a Birgé Massart like penalty

Usage

```
itergfpop(vectData = c(0), vectWeight = c(0), mygraph,
  type = "gauss", K = Inf, a = 0, min = -Inf, max = Inf,
  iter.max = 100, D.init = 1)
```

Arguments

`vectData`	vector of data to segment
`vectWeight`	vector of weights (positive numbers) same size as vectData
`mygraph`	dataframe of class graph to constraint the changepoint dynamic programming algorithm
`type`	a string defining the type of cost to use. "gauss", "poisson" or "binomial"
`K`	a positive number. Threshold for the Biweight robust loss
`a`	a positive number. Slope for the Huber robust loss
`min`	minimal bound for the infered means
`max`	maximal bound for the infered means
`iter.max`	maximal number of iteration of the gfpop function
`D.init`	initialisation of the number of segments

Value

a gfpop object = (changepoints, states, forced, means). 'changepoints' is the vector of changepoints (we give the last element of each segment). 'states' is the vector giving the state of each segment 'forced' is the vector specifying whether the constraints of the graph are active (=1) or not (=0) 'means' is the vector of successive means of each segment 'cost' is a number equal to the global cost of the graph-constrained segmentation 'Dvect' is a vector of integers. The successive tested D in the Birgé Massart penalty until convergence

---

`sdDiff` *sdDiff*

Description

Estimation of the standard deviation

Usage

```
sdDiff(x, method = "HALL")
```

Arguments

`x`	vector of datapoint
`method`	Three available methods: "HALL", "MAD" and "SD"

Value

a value equal to the estimated standard deviation

Examples

```
data <- dataGenerator(100, c(0.3, 0.6, 1), c(1, 2, 3), 2)
sdDiff(data)
```

`StartEnd` *Constraint starting and ending states to a graph*

Description

Adding constraints on the starting and ending states to a graph

Usage

```
StartEnd(start = -1, end = -1)
```

Arguments

`start`	a nonnegative integer. The first vertex in the changepoint inference
`end`	a nonnegative integer. The end vertex in the changepoint inference

Value

a dataframe with five components (as for edge) with only state1 and type = start or end defined.

Examples

```
StartEnd(start = 0, end = 1)
```

Index