Package ‘gfpop’

February 17, 2021

Type Package
Title Graph-Constrained Functional Pruning Optimal Partitioning
Version 1.0.3
Maintainer Vincent Runge <vincent.runge@univ-evry.fr>
Description Penalized parametric change-point detection by functional pruning dynamic programming algorithm. The successive means are constrained using a graph structure with edges of types null, up, down, std or abs. To each edge we can associate some additional properties: a minimal gap size, a penalty, some robust parameters (K,a). The user can also constrain the inferred means to lie between some minimal and maximal values. Data is modeled by a quadratic cost with possible use of a robust loss, biweight and Huber (see edge parameters K and a). Other losses are also available with log-linear representation or a log-log representation.

License MIT + file LICENSE
Encoding UTF-8
LazyData true
Imports Rcpp (>= 1.0.0)
SystemRequirements C++11
LinkingTo Rcpp
RoxygenNote 6.1.1
Depends R (>= 3.5.0)
NeedsCompilation yes
Author Vincent Runge [aut, cre], Toby Hocking [aut], Guillem Rigaill [aut], Daniel Grose [aut], Gaetano Romano [aut], Fatemeh Afghah [aut], Paul Fearnhead [aut], Michel Koskas [ctb], Arnaud Liehrmann [ctb]

Repository CRAN
Date/Publication 2021-02-17 13:20:02 UTC
dataGenerator

**R topics documented:**

<table>
<thead>
<tr>
<th>R Object</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataGenerator</td>
<td>2</td>
</tr>
<tr>
<td>Edge</td>
<td>3</td>
</tr>
<tr>
<td>gfpop</td>
<td>3</td>
</tr>
<tr>
<td>graph</td>
<td>4</td>
</tr>
<tr>
<td>itergfpop</td>
<td>5</td>
</tr>
<tr>
<td>Node</td>
<td>6</td>
</tr>
<tr>
<td>paperGraph</td>
<td>6</td>
</tr>
<tr>
<td>plot.gfpop</td>
<td>7</td>
</tr>
<tr>
<td>sdDiff</td>
<td>7</td>
</tr>
<tr>
<td>StartEnd</td>
<td>8</td>
</tr>
</tbody>
</table>

**Index**

<table>
<thead>
<tr>
<th>R Object</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataGenerator</td>
<td>9</td>
</tr>
</tbody>
</table>

---

**Description**

Generating data with a given model = changepoint relative positions + parameters + type of cost + (standard deviation + gamma decay)

**Usage**

```r
dataGenerator(n, changepoints, parameters, type = "mean", sigma = 1, gamma = 1, size = 10)
```

**Arguments**

- `n`: number of data points to generate
- `changepoints`: vector of position of the changepoints in (0,1] (last element is always 1).
- `parameters`: vector of means for the consecutive segments (same length as changepoints)
- `type`: a string defining the cost model to use: "mean", "variance", "poisson", "exp", "negbin"
- `sigma`: a positive number = the standard deviation of the data
- `gamma`: a number between 0 and 1 : the coefficient of the exponential decay (by default = 1 for piecewise constant signals)
- `size`: parameter of the rnbinom function

**Value**

- a vector of size n generated by the chosen model

**Examples**

```r
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", decay = 1, gap = 0, penalty = 0, K = Inf, a = 0)

**Arguments**

- **state1**: a string defining the starting state of the edge
- **state2**: a string defining the ending state of the edge
- **type**: a string equal to "null", "std", "up", "down" or "abs"
- **decay**: a nonnegative number to give the strength of the exponential decay into the segment
- **gap**: a nonnegative number to constrain the size of the gap in the change of state
- **penalty**: a nonnegative number. The penalty associated to this state transition
- **K**: a positive number. Threshold for the Biweight robust loss
- **a**: a positive number. Slope for the Huber robust loss

**Value**

a one-row dataframe with 9 variables

**Examples**

Edge("Dw", "Up", "up", gap = 1, penalty = 10, K = 3)

---

**gfpop**

*Graph-Constrained Functional Pruning Optimal Partitioning (gfpop)*

**Description**

Functional pruning optimal partitioning with a graph structure to take into account constraints on consecutive segment parameters. The user has to specify the graph he wants to use (see the graph function) and a type of cost function. This is the main function of the gfpop package.

**Usage**

gfpop(data, mygraph, type = "mean", weights = NULL, testMode = FALSE)
Arguments

data vector of data to segment
mygraph dataframe of class "graph" to constrain the changepoint inference
type a string defining the cost model to use: "mean", "variance", "poisson", "exp", "negbin"
weights vector of weights (positive numbers), same size as data
testMode boolean. False by default. Used to debug the code

Value

a gfpop object = (changepoints, states, forced, parameters, globalCost)

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)
parameters is the vector of successive parameters of each segment
globalCost is a number equal to the total loss: the minimal cost for the optimization problem with all penalty values excluded

Description

Graph creation using component functions "Edge", "StartEnd" and "Node"

Usage

graph(..., type = "empty", decay = 1, gap = 0, penalty = 0, K = Inf, a = 0, all.null.edges = FALSE)

Arguments

...This is a list of edges defined by functions "Edge", "StartEnd" and "Node"
type a string equal to "std", "isotonic", "updown", "relevant". to build a predefined classic graph
decay a nonnegative number to give the strength of the exponential decay into the segment
gap a nonnegative number to constrain the size of the gap in the change of state
penalty a nonnegative number equals to the common penalty to use for all edges
K a positive number. Threshold for the Biweight robust loss
a a positive number. Slope for the Huber robust loss
all.null.edges a boolean. Add null edges to all nodes automatically
**Value**

A dataframe with 9 variables (columns are named "state1", "state2", "type", "parameter", "penalty", "K", "a", "min", "max") with additional "graph" class.

**Examples**

```r
UpDownGraph <- graph(type = "updown", gap = 1.3, penalty = 10)
MyGraph <- graph(Edge("Dw","Dw"), Edge("Up","Up"), Edge("Dw","Up","up", gap = 0.5, penalty = 10),
Edge("Up","Dw","down"), StartEnd("Dw","Dw"), Node("Dw",0,1), Node("Up",0,1))
```

**Description**

Functional pruning optimal partitioning with a graph structure to take into account constraints on consecutive segment parameters. This is an iterated version of the main gfpop function using a Birgé Massart like penalty.

**Usage**

```r
itergfpop(data, mygraph, type = "mean", weights = NULL,
iter.max = 100, D.init = 1)
```

**Arguments**

- `data`: vector of data to segment
- `mygraph`: dataframe of class graph to constrain the changepoint inference
- `type`: a string defining the cost model to use: "gauss", "variance", "poisson", "exp", "negbin"
- `weights`: vector of weights (positive numbers), same size as data
- `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, parameters, globalCost, Dvect)

- `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)
- `parameters` is the vector of successive parameters of each segment
- `globalCost` is a number equal to the total loss: the minimal cost for the optimization problem with all penalty values excluded
- `Dvect` is a vector of integers. The successive tested D in the Birgé Massart penalty until convergence
### Node

<table>
<thead>
<tr>
<th>Node Values</th>
</tr>
</thead>
</table>

**Description**

Constrain the range of values to consider at a node

**Usage**

```r
Node(state = NULL, min = -Inf, max = Inf)
```

**Arguments**

- **state**: a string defining the state to constrain
- **min**: minimal value for the inferred parameter
- **max**: maximal value for the inferred parameter

**Value**

a dataframe with 9 variables with only 'state1', 'min' and 'max' defined.

**Examples**

```r
Node(state = "s0", min = 0, max = 2)
```

---

### paperGraph

**Graphs of our paper in jss**

**Description**

Graphs of our paper in jss

**Usage**

```r
paperGraph(nb, penalty = 0, decay = 1, gap = 0, oneValue = 0, K = Inf, a = 0)
```

**Arguments**

- **nb**: the number of the Figure in paper
- **penalty**: the penalty to use for change-point
- **decay**: a nonnegative number to give the strength of the exponential decay into the segment
- **gap**: a nonnegative number to constrain the size of the gap in the change of state
- **oneValue**: the value for parameter when we consider the collective anomalies problem
- **K**: a positive number. Threshold for the Biweight robust loss
- **a**: a positive number. Slope for the Huber robust loss
plot.gfpop

Value

A dataframe with 9 variables (columns are named "state1", "state2", "type", "parameter", "penalty", "K", "a", "min", "max") with additional "graph" class.

Description

Plot the result of the gfpop function with the data-points

Usage

```r
## S3 method for class 'gfpop'
plot(x, ..., data)
```

Arguments

- `x`: A gfpop class object
- `...`: Other parameters
- `data`: The data from which we get the gfpop result

Value

Plot data and the inferred gfpop segments

sdDiff

Description

Three methods to estimate the standard deviation in a time series for change-in-mean problem

Usage

```r
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**

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

---

**Description**

Defining the beginning and ending states of a graph

**Usage**

```r
StartEnd(start = NULL, end = NULL)
```

**Arguments**

- **start**: a vector of states. The beginning nodes for the changepoint inference
- **end**: a vector of states. The ending nodes for the changepoint inference

**Value**

dataframe with 9 variables with only `state1` and `type` = start or end defined.

**Examples**

```r
StartEnd(start = "A", end = c("A","B"))
```
Index

dataGenerator, 2
Edge, 3
gfpop, 3
graph, 4
itergfpop, 5
Node, 6
paperGraph, 6
plot.gfpop, 7
sdDiff, 7
StartEnd, 8