Package ‘gfpop’

March 10, 2022

Type Package
Title Graph-Constrained Functional Pruning Optimal Partitioning
Version 1.1.0
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 defining the nature of the changes. These changes can be unconstrained (type std), up or down constrained (type up and down) or constrained by a minimal size jump (type abs). The type null means that the graph allows us to stay on the same segment. To each edge we can associate some additional properties: a minimal gap size, a penalty, some robust parameters \((K, a)\) for biweight \((K)\) and Huber losses \((K\) and \(a)\). The user can also constrain the inferred means to lie between some minimal and maximal values. Data is modeled by a cost with possible use of a robust loss, biweight and Huber (see edge parameters \(K\) and \(a\)). These costs should have a quadratic, log-linear or a log-log representation.

License MIT + file LICENSE
Encoding UTF-8
Imports Rcpp (>= 1.0.0)
SystemRequirements C++11
LinkingTo Rcpp
RoxygenNote 7.1.2
Suggests devtools, knitr, data.table, testthat, rmarkdown, ggplot2, penaltyLearning
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],
dataGenerator

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

Usage

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

Arguments

n number of data points to generate
changepoints vector of positions 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(500, c(0.3, 0.6, 1), c(1, 2, 3), type = "mean", sigma = 0.5)
dataGenerator(1000, c(0.2, 0.33, 0.5, 1), c(4, 0.2, 3, 0.5), type = "variance")
dataGenerator(800, c(0.4, 0.8, 1), c(15, 5, 8), type = "mean", gamma = 0.95, sigma = 0.4)
dataGenerator(400, c(0.4, 0.9, 1), c(2, 1.5, 3), type = "poisson")
dataGenerator(1000, c(0.44, 0.86, 1), c(0.5, 0.2, 0.4), type = "negbin", size = 3)
```

---

**Description**

Edge creation for gfpop-R-Package graph

**Usage**

```r
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". Default type is "null", the transition to stay on the same segment.
- **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)`
- `Edge(0, 1, "abs", penalty = 2, gap = 1)`
- `Edge(0, 0, "null", penalty = 0, K = 2, a = 1)`
- `Edge("Dw", "Dw", type = "null", decay = 0.997)`

---

### 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. Its result can be plotted using the S3 gfpop function `gfpop::plot()`.

### Usage

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

- **data**: vector of data to segment. For simulation studies, Data can be generated using `gfpop` package function `gfpop::dataGenerator()`.

- **mygraph**: dataframe of class "graph" to constrain the changepoint inference, see `gfpop::graph()`.

- **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.

Details

The constrained optimization problem for n data points takes the following general form:

\[ Q_n = \min(\text{with constraints}) \left( \sum_{t=1}^{n} (\gamma(e[t])(y[t], \mu[t]) + \beta(e[t])) \right) \]

with data points \( y[t] \), edges \( e[t] \), edge-dependent penalties \( \beta(e[t]) \) and cost functions \( \gamma \). The cost function can take three different forms for parameter \( x \) and constants \( (A, B, C) \):

- **quadratic**, with representation \( Ax^2 + Bx + C \) with \( x \) in \( \mathbb{R} \)
- **log-linear**, with representation \( Ax - B \log(x) + C \) with \( x \geq 0 \)
- **log-log**, with representation \(-A \log(x) - B \log(1-x) + C \) with \( 0 \leq x \leq 1 \)

For each optimization problem, we consider a unique cost representation. However, the User can define robustness values (\( K \) and \( a \)) specific to each edge, making the cost function edge-dependent. We give the atomic form of each of the five available types (for one data point of value \( y \) with weight \( w \)):

- **"mean"**: \( A = w, B = -2wy, C = wy^2 \)
- **"variance"**: \( A = wy^2, B = w, C = 0 \)
- **"poisson"**: \( A = w, B = wy, C = 0 \)
- **"exp"**: \( A = wy, B = w, C = 0 \)
- **"negbin"**: \( A = w, B = wy, C = 0 \)

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 (= TRUE) or not (= FALSE).
- **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.
See Also

- `gfpop::dataGenerator()` to generate data for multiple change-point simulations
- `gfpop::graph()` to create graphs complying with the gfpop function
- `gfpop::plot()` to plot the gfpop object and visualize inferred changepoints and parameters

Examples

```r
n <- 1000 # data length
### EXAMPLE 1 ### updown graph + poisson loss
myData <- dataGenerator(n, c(0.1, 0.3, 0.5, 0.8, 1), c(1, 2, 1, 3, 1), type = "poisson")
myGraph <- graph(penalty = 2 * sdDiff(myData)^2 * log(n), type = "updown")
gfpop(data = myData, mygraph = myGraph, type = "poisson")

### EXAMPLE 2 ### relevant graph with min gap = 2 + poisson loss
myData <- dataGenerator(n, c(0.1, 0.3, 0.5, 0.8, 1), c(1, 2, 3, 5, 3), type = "poisson")
myGraph <- graph(type = "relevant", penalty = 2 * log(n), gap = 2)
gfpop(data = myData, mygraph = myGraph, type = "poisson")

### EXAMPLE 3 ### std graph with robust loss + variance loss
myData <- dataGenerator(n, c(0.1, 0.3, 0.5, 0.8, 1), c(1, 5, 1, 5, 1), type = "variance")
outliers <- 5 * rbinom(n, 1, 0.05) - 5 * rbinom(n, 1, 0.05)

### with robust parameter K
myGraph <- graph(type = "std", penalty = 2 * log(n), K = 10)
gfpop(data = myData + outliers, mygraph = myGraph, type = "variance")

### no K
myGraph <- graph(type = "std", penalty = 2 * log(n))
gfpop(data = myData, mygraph = myGraph, type = "variance")

### EXAMPLE 4 ### 3-segment graph with mean (Gaussian) loss
myData <- dataGenerator(n, c(0.12, 0.31, 0.53, 0.88, 1), c(1, 2, 0, 1, 2), type = "mean")
outliers <- 5 * rbinom(n, 1, 0.05) - 5 * rbinom(n, 1, 0.05)
gfpop(data = myData + outliers, mygraph = paperGraph(8, penalty = 2 * log(n)), type = "mean")
```

---

**graph**

*Graph generation*

**Description**

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

**Usage**

```r
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. See gfpop functions gfpop::Edge(), gfpop::StartEnd() and gfpop::Node()

type a string equal to "std", "isotonic", "updown" or "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

graph(type = "updown", gap = 1.3, penalty = 5)

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))

graph(Edge("1", "2", type = "std"),
    Edge("2", "3", type = "std"),
    Edge("3", "4", type = "std"),
    StartEnd(start = "1", end = "4"),
    all.null.edges = TRUE)
itergfpop

Graph-constrained functional pruning optimal partitioning iterated

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. For simulation studies, Data can be generated using gfpop package function `gfpop::dataGenerator()`

mygraph dataframe of class "graph" to constrain the changepoint inference, see `gfpop::graph()`

type a string defining the cost model to use: "mean", "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)

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 (= TRUE) or not (= FALSE)

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

**Description**

Constrain the range of values to consider at a node

**Usage**

```
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 state, min and max defined (not NA).

**Examples**

```
Node(state = "s0", min = 0, max = 2)
Node(state = 0, min = -1, max = 1)
Node(state = "positive", min = 0)
Node(state = "mu0", min = 0.5, max = 0.5)
```

---

**paperGraph**

*Graphs of our paper in JSS*

**Description**

Graphs of our paper in JSS (Journal of Statistical Software)

**Usage**

```
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

Value

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

Description

Plotting inferred segment parameters (the result of gfpop) and data.

Usage

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

Arguments

- `x` : a gfpop class object
- `...` : Other parameters
- `data` : the data from which we get the gfpop result
- `multiple` : if TRUE we plot data and the model on different graphs. Only with "mean" and "poisson" cost functions (as in that case the parameter values represent the data mean value over each segment) we allow the User to plot signal and data on a single graph.

Value

plot data and the inferred gfpop segments
Examples

```r
n <- 1000  # data length
data <- dataGenerator(n, c(0.3, 0.4, 0.7, 0.95, 1), c(1, 3, 1, -1, 4), "mean", sigma = 3)
myGraph <- graph(type = "relevant", gap = 0.5, penalty = 2 * sdDiff(data) ^ 2 * log(n))
g <- gfpop(data, myGraph, type = "mean")
plot(x = g, data = data, multiple = FALSE)

data <- dataGenerator(n, c(0.4, 0.8, 1), c(1, 1.7, 2.3), "exp")
g <- gfpop(data, graph(type = "isotonic", penalty = 2 * sdDiff(data) ^ 2 * log(n)), type = "exp")
plot(x = g, data = data, multiple = TRUE)

data <- dataGenerator(n, c(0.22, 0.75, 1), c(1.4, 1, 0.8), "poisson")
g <- gfpop(data, paperGraph(8), type = "poisson")
plot(x = g, data = data, multiple = TRUE)
```

Description

sdDiff is a function based on the difference operator (or difference order for HALL method) estimating the time-series standard deviation. The estimation works for time-series generated by Gaussian random variables with constant standard deviation and multiple changes in mean. Three estimators are available:

- **HALL**: the so-called HALL-estimator of order 3. For more details see: *(1990) Asymptotically optimal difference-based estimation of variance in nonparametric regression. Authors: Hall, Peter and Kay, JW and Titterinton, DM. Biometrika, pages 521–528*
- **MAD**: the median absolute deviation estimator computed on \( \frac{\text{diff}(x)}{\sqrt{2}} \) with \( x \) the vector of datapoints
- **SD**: the standard deviation estimator (function sd) computed on \( \frac{\text{diff}(x)}{\sqrt{2}} \) with \( x \) the vector of datapoints

Usage

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

Arguments

- **x**: vector of datapoints
- **method**: Three available methods: "HALL", "MAD" and "SD"

Value

a value equal to the estimated standard deviation
Examples

data <- dataGenerator(300, seq(0.1,1,0.1), sample(10), sigma = 2)
sdDiff(data, method = "HALL")
sdDiff(data, method = "MAD")
sdDiff(data, method = "SD")

StartEnd

Start and End nodes for the graph

Description

Defining the beginning and ending states of a graph

Usage

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 (not NA).

Examples

StartEnd(start = "A", end = c("A","B"))
StartEnd(start = 0)
StartEnd(start = 1, end = 1)
StartEnd(start = "v0", end = "v3")
StartEnd(end = "s0")
Index

dataGenerator, 2

Edge, 3

gfpop, 4
gfpop::dataGenerator(), 5, 6, 8
gfpop::Edge(), 7
gfpop::graph(), 5, 6, 8
gfpop::Node(), 7
gfpop::plot(), 4, 6
gfpop::StartEnd(), 7
graph, 6

itergfpop, 8

Node, 9

paperGraph, 9
plot.gfpop, 10

sdDiff, 11
StartEnd, 12