Package: rupturesRcpp (via r-universe)

May 23, 2026

Type Package

Title Object-Oriented Interface for Offline Change-Point Detection

Version 1.0.2

Description A collection of efficient implementations of popular offline change-point detection algorithms, featuring a consistent, object-oriented interface for practical use.

Encoding UTF-8

RoxygenNote 7.3.2

License CC BY 4.0

LinkingTo Rcpp, RcppArmadillo

Imports Rcpp, R6, ggplot2, patchwork, methods

Suggests testthat (>= 3.0.0), reticulate, binsegRcpp

Config/testthat/edition 3

Collate 'costFuncR6.R' 'PeltR6.R' 'WindowR6.R' 'binSegR6.R' 'rupturesRcpp-package.R' 'zzz.R'

NeedsCompilation yes

Author Minh Long Nguyen [aut, cre], Toby Hocking [aut], Charles Truong [aut]

Maintainer Minh Long Nguyen <edelweiss611428@gmail.com>

Repository https://cran.r-universe.dev

Date/Publication 2025-11-23 22:10:02 UTC

RemoteUrl https://github.com/cran/rupturesRcpp

RemoteRef HEAD

RemoteSha fa4123a43d7592431f3476100a95e6bbb4bc1995

Contents

binSeg *Binary Segmentation (*binSeg*)*

Description

An R6 class implementing binary segmentation for offline change-point detection.

Details

Binary segmentation is a classic algorithm for change-point detection that recursively splits the data at locations that minimise the cost function.

binSeg requires a R6 object of class costFunc, which can be created via costFunc$new(). Currently, the following cost functions are supported:

- "L1" and "L2" for (independent) piecewise Gaussian process with **constant variance**
- "SIGMA": for (independent) piecewise Gaussian process with **varying variance**
- "VAR": for piecewise Gaussian vector-regressive process with **constant noise variance**
- "LinearL2": for piecewise linear regression process with **constant noise variance**

See $eval() method for more details on computation of cost.

Some examples are provided below. See the GitHub README for detailed basic usage!

Methods

$new() Initialises a binSeg object.

$describe() Describes the binSeg object.

$fit() Constructs a binSeg module in C++.

$eval() Evaluates the cost of a segment.

$predict() Performs binSeg given a linear penalty value.

$plot() Plots change-point segmentation in ggplot style.

$clone() Clones the R6 object.

Active bindings

minSize Integer. Minimum allowed segment length. Can be accessed or modified via $minSize. Modifying minSize will automatically trigger $fit().

jump Integer. Search grid step size. Can be accessed or modified via $jump. Modifying jump will automatically trigger $fit().

costFunc R6 object of class costFunc. Search grid step size. Can be accessed or modified via $costFunc. Modifying costFunc will automatically trigger $fit().

tsMat Numeric matrix. Input time series matrix of size $n \times p$. Can be accessed or modified via $tsMat. Modifying tsMat will automatically trigger $fit().

covariates Numeric matrix. Input time series matrix having a similar number of observations as tsMat. Can be accessed or modified via $covariates. Modifying covariates will automatically trigger $fit().

Methods

Public methods:

- binSeg$new()
- binSeg$describe()
- binSeg$fit()
- binSeg$eval()
- binSeg$predict()
- binSeg$plot()
- binSeg$clone()

Method `new()`**:** Initialises a `binSeg` object.

Usage:

```
binSeg$new(minSize, jump, costFunc)
```

Arguments:

`minSize` Integer. Minimum allowed segment length. Default: `1L`.

`jump` Integer. Search grid step size: only positions in {k, 2k, ...} are considered. Default: `1L`.

`costFunc` A `R6` object of class `costFunc`. Should be created via `costFunc$new()` to avoid error. Default: `costFunc$new("L2")`.

Returns: Invisibly returns `NULL`.

Method `describe()`**:** Describes a `binSeg` object.

Usage:

```
binSeg$describe(printConfig = FALSE)
```

Arguments:

`printConfig` Logical. Whether to print object configurations. Default: `FALSE`.

Returns: Invisibly returns a list storing at least the following fields:

`minSize` Minimum allowed segment length.

`jump` Search grid step size.

`costFunc` The `costFun` object.

`fitted` Whether or not `$fit()` has been run.

`tsMat` Time series matrix.

`covariates` Covariate matrix (if exists).

`n` Number of observations.

`p` Number of features.

Method `fit()`**:** Constructs a `C++` module for binary segmentation.

Usage:

```
binSeg$fit(tsMat = NULL, covariates = NULL)
```

Arguments:

`tsMat` Numeric matrix. A time series matrix of size $n \times p$ whose rows are observations ordered in time. If `tsMat = NULL`, the method will use the previously assigned `tsMat` (e.g., set via the active binding `$tsMat` or from a prior `$fit(tsMat)`). Default: `NULL`.

`covariates` Numeric matrix. A time series matrix having a similar number of observations as `tsMat`. Required for models involving both dependent and independent variables. If `covariates = NULL` and no prior covariates were set (i.e., `$covariates` is still `NULL`), the model is force-fitted with only an intercept. Default: `NULL`.

Details: This method constructs a `C++ binSeg` module and sets `private$.fitted` to `TRUE`, enabling the use of `$predict()` and `$eval()`. Some precomputations are performed to allow `$predict()` to run in linear time with respect to the number of data points (see `$predict()` for more details).

Returns: Invisibly returns `NULL`.

Method `eval()`**:** Evaluate the cost of the segment (a,b]

Usage:

```
binSeg$eval(a, b)
```

Arguments:

`a` Integer. Start index of the segment (exclusive). Must satisfy `start < end`.

`b` Integer. End index of the segment (inclusive).

Details: The segment cost is evaluated as follows:

- **L1 cost function**:

$$c_{L_1}(y_{(a+1)\dots b}) := \sum_{t=a+1}^{b} \|y_t - \tilde{y}_{(a+1)\dots b}\|_1$$

  where $\tilde{y}_{(a+1)\dots b}$ is the coordinate-wise median of the segment. If $a \geq b-1$, return 0.

- **L2 cost function**:

$$c_{L_2}(y_{(a+1)\dots b}) := \sum_{t=a+1}^{b} \|y_t - \bar{y}_{(a+1)\dots b}\|_2^2$$

  where $\bar{y}_{(a+1)\dots b}$ is the empirical mean of the segment. If $a \geq b-1$, return 0.

- **SIGMA cost function**:

$$c_{\sum}(y_{(a+1)\dots b}) := (b-a)\log\det\hat{\Sigma}_{(a+1)\dots b}$$

  where $\hat{\Sigma}_{(a+1)\dots b}$ is the empirical covariance matrix of the segment without Bessel's correction. Here, if `addSmallDiag = TRUE`, a small bias `epsilon` is added to the diagonal of estimated covariance matrices to improve numerical stability.

  By default, `addSmallDiag = TRUE` and `epsilon = 1e-6`. In case `addSmallDiag = TRUE`, if the computed determinant of covariance matrix is either 0 (singular) or smaller than `p*log(epsilon)` - the lower bound, return `(b - a)*p*log(epsilon)`, otherwise, output an error message.

- **VAR(r) cost function**:

$$c_{\mathrm{VAR}}(y_{(a+1)\dots b}) := \sum_{t=a+r+1}^{b} \left\| y_t - \sum_{j=1}^{r} \hat{A}_j y_{t-j} \right\|_2^2$$

  where $\hat{A}_j$ are the estimated VAR coefficients, commonly estimated via the OLS criterion. If system is singular, $a - b < p * r + 1$ (i.e., not enough observations), or $a \geq n - p$ (where `n` is the time series length), return 0.

- **"LinearL2"** for piecewise linear regression process with **constant noise variance**

$$c_{\text{LinearL2}}(y_{(a+1):b}) := \sum_{t=a+1}^{b} \|y_t - X_t\hat{\beta}\|_2^2$$

where $\hat{\beta}$ are OLS estimates on segment $(a+1) : b$. If segment is shorter than the minimum number of points needed for OLS, return 0.

Returns: The segment cost.

Method `predict()`**:** Performs `binSeg` given a linear penalty value.

Usage:

```
binSeg$predict(pen = 0)
```

Arguments:

`pen` Numeric. Penalty per change-point. Default: `0`.

Details: The algorithm recursively partitions a time series to detect multiple change-points. At each step, the algorithm identifies the segment that, if split, would result in the greatest reduction in total cost. This process continues until no further splits are possible (e.g., each segment is of minimal length or each breakpoint corresponds to a single data point).
Then, the algorithm selects the "optimal" set of break-points given the linear penalty threshold. Let $[c_1, \ldots, c_k, c_{k+1}]$ denote the set of segment end-points with $c_1 < c_2 < \cdots < c_k < c_{k+1} = n$, where k is the number of detected change-points and n is the total number of data points. and k is the number of change-points. Let $c_{(c_i, c_{i+1}]}$ be the cost of segment $(c_i, c_{i+1}]$. The total penalised cost is then

$$\text{TotalCost} = \sum_{i=1}^{k+1} c_{(c_i, c_{i+1}]} + \lambda \cdot k,$$

where λ is a linear penalty applied per change-point. We then optimise over k to minimise the penalised cost function.
This approach allows detecting multiple change-points in a time series while controlling model complexity through the linear penalty threshold.
In our implementation, the recursive step is carried out during `$fit()`. Therefore, `$predict()` runs in linear time with respect to the number of data points.
Temporary segment end-points are saved to `private$.tmpEndPoints` after `$predict()`, enabling users to call `$plot()` without specifying endpoints manually.

Returns: An integer vector of regime end-points. By design, the last element is the number of observations.

Method `plot()`**:** Plots change-point segmentation

Usage:

```
binSeg$plot(
  d = 1L,
  endPts,
  dimNames,
  main,
  xlab,
  tsWidth = 0.25,
```

```
  tsCol = "#5B9BD5",
  bgCol = c("#A3C4F3", "#FBB1BD"),
  bgAlpha = 0.5,
  ncol = 1L
)
```

Arguments:

`d` Integer vector. Dimensions to plot. Default: `1L`.

`endPts` Integer vector. End points. Default: latest temporary changepoints obtained via `$predict()`.

`dimNames` Character vector. Feature names matching length of `d`. Defaults to `"X1", "X2", ...`.

`main` Character. Main title. Defaults to `"binSeg: d = ..."`.

`xlab` Character. X-axis label. Default: `"Time"`.

`tsWidth` Numeric. Line width for time series and segments. Default: `0.25`.

`tsCol` Character. Time series color. Default: `"#5B9BD5"`.

`bgCol` Character vector. Segment colors, recycled to length of `endPts`. Default: `c("#A3C4F3", "#FBB1BD")`.

`bgAlpha` Numeric. Background transparency. Default: `0.5`.

`ncol` Integer. Number of columns in facet layout. Default: `1L`.

Details: Plots change-point segmentation results. Based on `ggplot2`. Multiple plots can easily be horizontally and vertically stacked using `patchwork`'s operators `/` and `|`, respectively.

Returns: An object of classes `gg` and `ggplot`.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
binSeg$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Minh Long Nguyen <edelweiss611428@gmail.com>
Toby Dylan Hocking <toby.hocking@r-project.org>
Charles Truong <ctruong@ens-paris-saclay.fr>

References

Truong, C., Oudre, L., & Vayatis, N. (2020). Selective review of offline change point detection methods. Signal Processing, 167, 107299.

Hocking, T. D. (2024). Finite Sample Complexity Analysis of Binary Segmentation. arXiv preprint arXiv:2410.08654.

Examples

```
## L2 example
set.seed(1121)
signals = as.matrix(c(rnorm(100,0,1),
```

```
                      rnorm(100,5,1)))
# Default L2 cost function
binSegObj = binSeg$new(minSize = 1L, jump = 1L)
binSegObj$fit(signals)
binSegObj$predict(pen = 100)
binSegObj$plot()

## SIGMA example
set.seed(111)
signals = as.matrix(c(rnorm(100,-5,1),
                      rnorm(100,-5,10),
                      rnorm(100,-5,1)))
# L2 cost function
binSegObj = binSeg$new(minSize = 1L, jump = 1L)
binSegObj$fit(signals)
# We choose pen = 50.
binSegObj$predict(pen = 50)
binSegObj$plot()

# The standard L2 cost function is not suitable.
# Use the SIGMA cost function.
binSegObj$costFunc = costFunc$new(costFunc = "SIGMA")
binSegObj$predict(pen = 50)
binSegObj$plot()
```

---

costFunc costFunc *class*

Description

An R6 class specifying a cost function

Details

Creates an instance of costFunc R6 class, used in initialisation of change-point detection modules. Currently supports the following cost functions:

- **L1 cost function**:

$$c_{L_1}(y_{(a+1)\ldots b}) := \sum_{t=a+1}^{b} \|y_t - \tilde{y}_{(a+1)\ldots b}\|_1$$

  where $\tilde{y}_{(a+1)\ldots b}$ is the coordinate-wise median of the segment. If $a \geq b-1$, return 0.

- **L2 cost function**:

$$c_{L_2}(y_{(a+1)\ldots b}) := \sum_{t=a+1}^{b} \|y_t - \bar{y}_{(a+1)\ldots b}\|_2^2$$

  where $\bar{y}_{(a+1)\ldots b}$ is the empirical mean of the segment. If $a \geq b-1$, return 0.

- **SIGMA cost function**:

$$c_{\sum}(y_{(a+1)\dots b}) := (b-a)\log\det\hat{\Sigma}_{(a+1)\dots b}$$

where $\hat{\Sigma}_{(a+1)\dots b}$ is the empirical covariance matrix of the segment without Bessel's correction. Here, if `addSmallDiag = TRUE`, a small bias `epsilon` is added to the diagonal of estimated covariance matrices to improve numerical stability.

By default, `addSmallDiag = TRUE` and `epsilon = 1e-6`. In case `addSmallDiag = TRUE`, if the computed determinant of covariance matrix is either 0 (singular) or smaller than `p*log(epsilon)` - the lower bound, return `(b - a)*p*log(epsilon)`, otherwise, output an error message.

- **VAR(r) cost function**:

$$c_{\mathrm{VAR}}(y_{(a+1)\dots b}) := \sum_{t=a+r+1}^{b} \left\| y_t - \sum_{j=1}^{r} \hat{A}_j y_{t-j} \right\|_2^2$$

where $\hat{A}_j$ are the estimated VAR coefficients, commonly estimated via the OLS criterion. If system is singular, $a - b < p * r + 1$ (i.e., not enough observations), or $a \geq n - p$ (where `n` is the time series length), return 0.

- **"LinearL2"** for piecewise linear regression process with **constant noise variance**

$$c_{\mathrm{LinearL2}}(y_{(a+1):b}) := \sum_{t=a+1}^{b} \|y_t - X_t\hat{\beta}\|_2^2$$

where $\hat{\beta}$ are OLS estimates on segment $(a+1) : b$. If segment is shorter than the minimum number of points needed for OLS, return 0.

If active binding `$costFunc` is modified (via assignment operator), the default parameters will be used.

Methods

`$new()` Initialises a `costFunc` object.

`$pass()` Describes the `costFunc` object.

`$clone()` Clones the `costFunc` object.

Active bindings

`costFunc` Character. Cost function. Can be accessed or modified via `$costFunc`. If `costFunc` is modified and required parameters are missing, the default parameters are used.

`pVAR` Integer. Vector autoregressive order. Can be accessed or modified via `$pVAR`.

`addSmallDiag` Logical. Whether to add a bias value to the diagonal of estimated covariance matrices to stabilise matrix operations. Can be accessed or modified via `$addSmallDiag`.

`epsilon` Double. A bias value added to the diagonal of estimated covariance matrices to stabilise matrix operations. Can be accessed or modified via `$epsilon`.

`intercept` Logical. Whether to include the intercept in regression problems. Can be accessed or modified via `$intercept`.

Methods

Public methods:

- costFunc$new()
- costFunc$pass()
- costFunc$clone()

Method `new()`**:** Initialises a `costFunc` object.

Usage:

```
costFunc$new(costFunc, ...)
```

Arguments:

`costFunc` Character. Cost function. Supported values include `"L2"`, `"VAR"`, and `"SIGMA"`. Default: `L2`.

`...` Optional named parameters required by specific cost functions.
If any required parameters are missing or null, default values will be used.
For `"L1"` and `"L2"`, there is no extra parameter.
For `"SIGMA"`, supported parameters are:

`addSmallDiag` Logical. If `TRUE`, add a small value to the diagonal of estimated covariance matrices to stabilise matrix operations. Default: `TRUE`.

`epsilon` Double. If `addSmallDiag = TRUE`, a small positive value added to the diagonal of estimated covariance matrices to stabilise matrix operations. Default: `1e-6`.

For `"VAR"`, `pVAR` is required:

`pVAR` Integer. Vector autoregressive order. Must be a positive integer. Default: `1L`.

For `"LinearL2"`, `intercept` is required:

`intercept` Logical. Whether to include the intercept in regression problems. Default: `TRUE`.

Method `pass()`**:** Returns a list of configuration parameters to initialise `detection` modules.

Usage:

```
costFunc$pass()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
costFunc$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Minh Long Nguyen <edelweiss611428@gmail.com>

Examples

```
## L2 costFunc (default)
costFuncObj = costFunc$new()
costFuncObj$pass()
## SIGMA costFunc
costFuncObj = costFunc$new(costFunc = "SIGMA")
costFuncObj$pass()
# Modify active bindings
costFuncObj$epsilon = 10^-5
costFuncObj$pass()
costFuncObj$costFunc = "VAR"
costFuncObj$pass()
```

PELT *Pruned Exact Linear Time (PELT)*

Description

An R6 class implementing the PELT algorithm for offline change-point detection.

Details

PELT (Pruned Exact Linear Time) is an efficient algorithm for change point detection that prunes the search space to achieve optimal segmentation in linear time under certain conditions.

PELT requires a R6 object of class costFunc, which can be created via costFunc$new(). Currently, the following cost functions are supported:

- "L1" and "L2" for (independent) piecewise Gaussian process with **constant variance**
- "SIGMA": for (independent) piecewise Gaussian process with **varying variance**
- "VAR": for piecewise Gaussian vector-regressive process with **constant noise variance**
- "LinearL2": for piecewise linear regression process with **constant noise variance**

See $eval() method for more details on computation of cost.

Some examples are provided below. See the GitHub README for detailed basic usage!

Methods

$new() Initialises a PELT object.

$describe() Describes the PELT object.

$fit() Constructs a PELT module in C++.

$eval() Evaluates the cost of a segment.

$predict() Performs PELT given a linear penalty value.

$plot() Plots change-point segmentation in ggplot style.

$clone() Clones the R6 object.

Active bindings

`minSize` Integer. Minimum allowed segment length. Can be accessed or modified via `$minSize`. Modifying `minSize` will automatically trigger `$fit()`.

`jump` Integer. Search grid step size. Can be accessed or modified via `$jump`. Modifying `jump` will automatically trigger `$fit()`.

`costFunc` `R6` object of class `costFunc`. Search grid step size. Can be accessed or modified via `$costFunc`. Modifying `costFunc` will automatically trigger `$fit()`.

`tsMat` Numeric matrix. Input time series matrix of size $n \times p$. Can be accessed or modified via `$tsMat`. Modifying `tsMat` will automatically trigger `$fit()`.

`covariates` Numeric matrix. Input time series matrix having a similar number of observations as `tsMat`. Can be accessed or modified via `$covariates`. Modifying `covariates` will automatically trigger `$fit()`.

Methods

Public methods:

- `PELT$new()`
- `PELT$describe()`
- `PELT$fit()`
- `PELT$eval()`
- `PELT$predict()`
- `PELT$plot()`
- `PELT$clone()`

Method `new()`**:** Initialises a `PELT` object.

Usage:

```
PELT$new(minSize, jump, costFunc)
```

Arguments:

`minSize` Integer. Minimum allowed segment length. Default: `1L`.

`jump` Integer. Search grid step size: only positions in {k, 2k, ...} are considered. Default: `1L`.

`costFunc` A `R6` object of class `costFunc`. Should be created via `costFunc$new()` to avoid error. Default: `costFunc$new("L2")`.

Returns: Invisibly returns `NULL`.

Method `describe()`**:** Describes a `PELT` object.

Usage:

```
PELT$describe(printConfig = FALSE)
```

Arguments:

`printConfig` Logical. Whether to print object configurations. Default: `FALSE`.

Returns: Invisibly returns a list storing at least the following fields:

`minSize` Minimum allowed segment length.

`jump` Search grid step size.

`costFunc` The `costFun` object.

`fitted` Whether or not `$fit()` has been run.

`tsMat` Time series matrix.

`covariates` Covariate matrix (if exists).

`n` Number of observations.

`p` Number of features.

Method `fit()`**:** Constructs a `C++` module for `PELT`.

Usage:

```
PELT$fit(tsMat = NULL, covariates = NULL)
```

Arguments:

`tsMat` Numeric matrix. A time series matrix of size $n \times p$ whose rows are observations ordered in time. If `tsMat = NULL`, the method will use the previously assigned `tsMat` (e.g., set via the active binding `$tsMat` or from a prior `$fit(tsMat)`). Default: `NULL`.

`covariates` Numeric matrix. A time series matrix having a similar number of observations as `tsMat`. Required for models involving both dependent and independent variables. If `covariates = NULL` and no prior covariates were set (i.e., `$covariates` is still `NULL`), the model is force-fitted with only an intercept. Default: `NULL`.

Details: This method constructs a `C++ PELT` module and sets `private$.fitted` to `TRUE`, enabling the use of `$predict()` and `$eval()`.

Returns: Invisibly returns `NULL`.

Method `eval()`**:** Evaluate the cost of the segment (a,b]

Usage:

```
PELT$eval(a, b)
```

Arguments:

`a` Integer. Start index of the segment (exclusive). Must satisfy `start < end`.

`b` Integer. End index of the segment (inclusive).

Details: The segment cost is evaluated as follows:

- **L1 cost function**:

$$c_{L_1}(y_{(a+1)\dots b}) := \sum_{t=a+1}^{b} \|y_t - \tilde{y}_{(a+1)\dots b}\|_1$$

  where $\tilde{y}_{(a+1)\dots b}$ is the coordinate-wise median of the segment. If $a \geq b-1$, return 0.

- **L2 cost function**:

$$c_{L_2}(y_{(a+1)\dots b}) := \sum_{t=a+1}^{b} \|y_t - \bar{y}_{(a+1)\dots b}\|_2^2$$

  where $\bar{y}_{(a+1)\dots b}$ is the empirical mean of the segment. If $a \geq b-1$, return 0.

- **SIGMA cost function**:

$$c_{\sum}(y_{(a+1)\dots b}) := (b-a)\log\det\hat{\Sigma}_{(a+1)\dots b}$$

  where $\hat{\Sigma}_{(a+1)\dots b}$ is the empirical covariance matrix of the segment without Bessel's correction. Here, if `addSmallDiag = TRUE`, a small bias `epsilon` is added to the diagonal of

estimated covariance matrices to improve numerical stability.

By default, `addSmallDiag = TRUE` and `epsilon = 1e-6`. In case `addSmallDiag = TRUE`, if the computed determinant of covariance matrix is either 0 (singular) or smaller than `p*log(epsilon)` - the lower bound, return `(b - a)*p*log(epsilon)`, otherwise, output an error message.

- **VAR(r) cost function**:

$$c_{\text{VAR}}(y_{(a+1)\dots b}) := \sum_{t=a+r+1}^{b} \left\| y_t - \sum_{j=1}^{r} \hat{A}_j y_{t-j} \right\|_2^2$$

where $\hat{A}_j$ are the estimated VAR coefficients, commonly estimated via the OLS criterion. If system is singular, $a - b < p * r + 1$ (i.e., not enough observations), or $a \geq n - p$ (where `n` is the time series length), return 0.

"LinearL2" for piecewise linear regression process with **constant noise variance**

$$c_{\text{LinearL2}}(y_{(a+1):b}) := \sum_{t=a+1}^{b} \|y_t - X_t \hat{\beta}\|_2^2$$

where $\hat{\beta}$ are OLS estimates on segment $(a + 1) : b$. If segment is shorter than the minimum number of points needed for OLS, return 0.

Returns: The segment cost.

Method `predict()`**:** Performs `PELT` given a linear penalty value.

Usage:

```
PELT$predict(pen = 0)
```

Arguments:

`pen` Numeric. Penalty per change-point. Default: `0`.

Details: The `PELT` algorithm detects multiple change-points by finding the set of break-points that globally minimises a penalised cost function. `PELT` uses dynamic programming combined with a pruning rule to reduce the number of candidate change-points, achieving efficient computation.
Let $[c_1, \dots, c_k, c_{k+1}]$ denote the set of segment end-points with $c_1 < c_2 < \cdots < c_k < c_{k+1} = n$, where k is the number of detected change-points and n is the total number of data points. Let $c_{(c_i, c_{i+1}]}$ be the cost of segment $(c_i, c_{i+1}]$. The total penalised cost is

$$\text{TotalCost} = \sum_{i=1}^{k+1} c_{(c_i, c_{i+1}]} + \lambda \cdot k,$$

where λ is a linear penalty applied per change-point. `PELT` finds the set of endpoints that minimises this cost exactly.
The pruning step eliminates candidate change-points that cannot lead to an optimal solution, allowing `PELT` to run in linear time with respect to the number of data points.
Temporary segment end-points are saved to `private$.tmpEndPoints` after `$predict()`, enabling users to call `$plot()` without specifying endpoints manually.

Returns: An integer vector of regime end-points. By design, the last element is the number of observations.

Method `plot()`**:** Plots change-point segmentation

Usage:

```
PELT$plot(
  d = 1L,
  endPts,
  dimNames,
  main,
  xlab,
  tsWidth = 0.25,
  tsCol = "#5B9BD5",
  bgCol = c("#A3C4F3", "#FBB1BD"),
  bgAlpha = 0.5,
  ncol = 1L
)
```

Arguments:

`d` Integer vector. Dimensions to plot. Default: `1L`.

`endPts` Integer vector. End points. Default: latest temporary changepoints obtained via `$predict()`.

`dimNames` Character vector. Feature names matching length of `d`. Defaults to `"X1", "X2", ...`.

`main` Character. Main title. Defaults to `"PELT: d = ..."`.

`xlab` Character. X-axis label. Default: `"Time"`.

`tsWidth` Numeric. Line width for time series and segments. Default: `0.25`.

`tsCol` Character. Time series color. Default: `"#5B9BD5"`.

`bgCol` Character vector. Segment colors, recycled to length of `endPts`. Default: `c("#A3C4F3", "#FBB1BD")`.

`bgAlpha` Numeric. Background transparency. Default: `0.5`.

`ncol` Integer. Number of columns in facet layout. Default: `1L`.

Details: Plots change-point segmentation results. Based on `ggplot2`. Multiple plots can easily be horizontally and vertically stacked using `patchwork`'s operators `/` and `|`, respectively.

Returns: An object of classes `gg` and `ggplot`.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PELT$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Minh Long Nguyen <edelweiss611428@gmail.com>
Toby Dylan Hocking <toby.hocking@r-project.org>
Charles Truong <ctruong@ens-paris-saclay.fr>

References

Truong, C., Oudre, L., & Vayatis, N. (2020). Selective review of offline change point detection methods. Signal Processing, 167, 107299.

Killick, R., Fearnhead, P., & Eckley, I. A. (2012). Optimal detection of change points with a linear computational cost. Journal of the American Statistical Association, 107(500), 1590-1598.

Examples

```
## L2 example
set.seed(1121)
signals = as.matrix(c(rnorm(100,0,1),
                      rnorm(100,5,1)))
# Default L2 cost function
PELTObj = PELT$new(minSize = 1L, jump = 1L)
PELTObj$fit(signals)
PELTObj$predict(pen = 100)
PELTObj$plot()

## SIGMA example
set.seed(111)
signals = as.matrix(c(rnorm(100,-5,1),
                      rnorm(100,-5,10),
                      rnorm(100,-5,1)))
# L2 cost function
PELTObj = PELT$new(minSize = 1L, jump = 1L)
PELTObj$fit(signals)
# We choose pen = 50.
PELTObj$predict(pen = 50)
PELTObj$plot()

# The standard L2 cost function is not suitable.
# Use the SIGMA cost function.
PELTObj$costFunc = costFunc$new(costFunc = "SIGMA")
PELTObj$predict(pen = 50)
PELTObj$plot()
```

Window *Slicing Window (*Window*)*

Description

An R6 class implementing slicing window for offline change-point detection.

Details

Slicing window is a scalable, linear-time change-point detection algorithm that selects breakpoints based on local gains computed over sliding windows.

Currently supports the following cost functions:

- "L1" and "L2" for (independent) piecewise Gaussian process with **constant variance**
- "SIGMA": for (independent) piecewise Gaussian process with **varying variance**
- "VAR": for piecewise Gaussian vector-regressive process with **constant noise variance**
- "LinearL2": for piecewise linear regression process with **constant noise variance**

Window requires a R6 object of class costFunc, which can be created via costFunc$new(). Currently, the following cost functions are supported:

- "L1" and "L2" for (independent) piecewise Gaussian process with **constant variance**
- "SIGMA": for (independent) piecewise Gaussian process with **varying variance**
- "VAR": for piecewise Gaussian vector-regressive process with **constant noise variance**
- "LinearL2": for piecewise linear regression process with **constant noise variance**

See $eval() method for more details on computation of cost.

Some examples are provided below. See the GitHub README for detailed basic usage!

Methods

$new() Initialises a Window object.

$describe() Describes the Window object.

$fit() Constructs a Window module in C++.

$eval() Evaluates the cost of a segment.

$predict() Performs Window given a linear penalty value.

$plot() Plots change-point segmentation in ggplot style.

$clone() Clones the R6 object.

Active bindings

minSize Integer. Minimum allowed segment length. Can be accessed or modified via $minSize. Modifying minSize will automatically trigger $fit().

radius Integer. Window radius. Can be accessed or modified via $radius. Modifying radius will automatically trigger $fit().

jump Integer. Search grid step size. Can be accessed or modified via $jump. Modifying jump will automatically trigger $fit().

costFunc R6 object of class costFunc. Search grid step size. Can be accessed or modified via $costFunc. Modifying costFunc will automatically trigger $fit().

tsMat Numeric matrix. Input time series matrix of size $n \times p$. Can be accessed or modified via $tsMat. Modifying tsMat will automatically trigger $fit().

covariates Numeric matrix. Input time series matrix having a similar number of observations as tsMat. Can be accessed or modified via $covariates. Modifying covariates will automatically trigger $fit().

Methods

Public methods:

- `Window$new()`
- `Window$describe()`
- `Window$fit()`
- `Window$eval()`
- `Window$predict()`
- `Window$plot()`
- `Window$clone()`

Method `new()`**:** Initialises a `Window` object.

Usage:

```
Window$new(minSize, jump, radius, costFunc)
```

Arguments:

`minSize` Integer. Minimum allowed segment length. Default: `1L`.

`jump` Integer. Search grid step size: only positions in {k, 2k, ...} are considered. Default: `1L`.

`radius` Integer. Radius of each sliding window. Default: `1L`.

`costFunc` A `R6` object of class `costFunc`. Should be created via `costFunc$new()` to avoid error. Default: `costFunc$new("L2")`.

Returns: Invisibly returns `NULL`.

Method `describe()`**:** Describes a `Window` object.

Usage:

```
Window$describe(printConfig = FALSE)
```

Arguments:

`printConfig` Logical. Whether to print object configurations. Default: `FALSE`.

Returns: Invisibly returns a list storing at least the following fields:

`minSize` Minimum allowed segment length.

`jump` Search grid step size.

`radius` Radius of each sliding window.

`costFunc` The `costFun` object.

`fitted` Whether or not `$fit()` has been run.

`tsMat` Time series matrix.

`covariates` Covariate matrix (if exists).

`n` Number of observations.

`p` Number of features.

Method `fit()`**:** Constructs a `C++` module for `Window`.

Usage:

```
Window$fit(tsMat = NULL, covariates = NULL)
```

Arguments:

`tsMat` Numeric matrix. A time series matrix of size $n \times p$ whose rows are observations ordered in time. If `tsMat = NULL`, the method will use the previously assigned `tsMat` (e.g., set via the active binding `$tsMat` or from a prior `$fit(tsMat)`). Default: `NULL`.

`covariates` Numeric matrix. A time series matrix having a similar number of observations as `tsMat`. Required for models involving both dependent and independent variables. If `covariates = NULL` and no prior covariates were set (i.e., `$covariates` is still `NULL`), the model is force-fitted with only an intercept. Default: `NULL`..

Details: This method constructs a `C++ Window` module and sets `private$.fitted` to `TRUE`, enabling the use of `$predict()` and `$eval()`. Some precomputations are performed to allow `$predict()` to run in linear time with respect to the number of local change-points (see `$predict()` for more details).

Returns: Invisibly returns `NULL`.

Method `eval()`**:** Evaluate the cost of the segment (a,b]

Usage:

```
Window$eval(a, b)
```

Arguments:

`a` Integer. Start index of the segment (exclusive). Must satisfy `start < end`.

`b` Integer. End index of the segment (inclusive).

Details: The segment cost is evaluated as follows:

- **L1 cost function**:

$$c_{L_1}(y_{(a+1)\dots b}) := \sum_{t=a+1}^{b} \|y_t - \tilde{y}_{(a+1)\dots b}\|_1$$

  where $\tilde{y}_{(a+1)\dots b}$ is the coordinate-wise median of the segment. If $a \geq b-1$, return 0.

- **L2 cost function**:

$$c_{L_2}(y_{(a+1)\dots b}) := \sum_{t=a+1}^{b} \|y_t - \bar{y}_{(a+1)\dots b}\|_2^2$$

  where $\bar{y}_{(a+1)\dots b}$ is the empirical mean of the segment. If $a \geq b-1$, return 0.

- **SIGMA cost function**:

$$c_{\sum}(y_{(a+1)\dots b}) := (b-a)\log\det\hat{\Sigma}_{(a+1)\dots b}$$

  where $\hat{\Sigma}_{(a+1)\dots b}$ is the empirical covariance matrix of the segment without Bessel's correction. Here, if `addSmallDiag = TRUE`, a small bias `epsilon` is added to the diagonal of estimated covariance matrices to improve numerical stability.

  By default, `addSmallDiag = TRUE` and `epsilon = 1e-6`. In case `addSmallDiag = TRUE`, if the computed determinant of covariance matrix is either 0 (singular) or smaller than `p*log(epsilon)` - the lower bound, return `(b - a)*p*log(epsilon)`, otherwise, output an error message.

- **VAR(r) cost function**:

$$c_{\mathrm{VAR}}(y_{(a+1)\dots b}) := \sum_{t=a+r+1}^{b} \left\| y_t - \sum_{j=1}^{r} \hat{A}_j y_{t-j} \right\|_2^2$$

where $\hat{A}_j$ are the estimated VAR coefficients, commonly estimated via the OLS criterion. If system is singular, $a - b < p * r + 1$ (i.e., not enough observations), or $a \geq n - p$ (where `n` is the time series length), return 0.

- **"LinearL2"** for piecewise linear regression process with **constant noise variance**

$$c_{\text{LinearL2}}(y_{(a+1):b}) := \sum_{t=a+1}^{b} \|y_t - X_t\hat{\beta}\|_2^2$$

where $\hat{\beta}$ are OLS estimates on segment $(a+1) : b$. If segment is shorter than the minimum number of points needed for OLS, return 0.

Returns: The segment cost.

Method `predict()`**:** Performs `Window` given a linear penalty value.

Usage:

```
Window$predict(pen = 0)
```

Arguments:

`pen` Numeric. Penalty per change-point. Default: `0`.

Details: The algorithm scans the data with a fixed-size window to detect candidate local change-points (lcps) if the gains of its k_{thresh} neighbors to the left and right are all smaller than its gain, where k_{thresh} is defined as

$$k_{\text{thresh}} = \max\left(\frac{\max(2\text{radius}, 2 \cdot \text{minSize})}{2 \cdot \text{jump}}, 1\right)$$

After candidate local change-points and computing the local gains, the algorithm selects the "optimal" set of break-points given the linear penalty threshold. Let G_i denote the local gain for candidate change-point i, for $i = 1, \ldots, n_{\text{lcps}}$. The local gains are ordered such that $G_1 \geq G_2 \geq \cdots \geq G_{n_{\text{lcps}}}$. Note that it is possible that no local change-points are detected, for example if the window size is too large.
The total cost for the selected k change-points is then calculated as

$$\text{TotalCost} = -\sum_{i=1}^{k} G_i + \lambda \cdot k,$$

where λ is a linear penalty applied per change-point. We then optimise over k to minimise the penalised cost function.
This approach allows detecting multiple change-points in a time series while controlling model complexity through the linear penalty threshold.
In our implementation, scanning the data to detect candidate local change-points and computing their corresponding local gains is already performed in `$fit()`. Therefore, `$predict()` runs in linear time with respect to the number of local change-points.
Temporary segment end-points are saved to `private$.tmpEndPoints` after `$predict()`, enabling users to call `$plot()` without specifying endpoints manually.

Returns: An integer vector of regime end-points. By design, the last element is the number of observations.

Method `plot()`**:** Plots change-point segmentation

Usage:

```
Window$plot(
  d = 1L,
  endPts,
  dimNames,
  main,
  xlab,
  tsWidth = 0.25,
  tsCol = "#5B9BD5",
  bgCol = c("#A3C4F3", "#FBB1BD"),
  bgAlpha = 0.5,
  ncol = 1L
)
```

Arguments:

`d` Integer vector. Dimensions to plot. Default: `1L`.

`endPts` Integer vector. End points. Default: latest temporary changepoints obtained via `$predict()`.

`dimNames` Character vector. Feature names matching length of `d`. Defaults to `"X1", "X2", ...`.

`main` Character. Main title. Defaults to `"Window: d = ..."`.

`xlab` Character. X-axis label. Default: `"Time"`.

`tsWidth` Numeric. Line width for time series and segments. Default: `0.25`.

`tsCol` Character. Time series color. Default: `"#5B9BD5"`.

`bgCol` Character vector. Segment colors, recycled to length of `endPts`. Default: `c("#A3C4F3", "#FBB1BD")`.

`bgAlpha` Numeric. Background transparency. Default: `0.5`.

`ncol` Integer. Number of columns in facet layout. Default: `1L`.

Details: Plots change-point segmentation results. Based on `ggplot2`. Multiple plots can easily be horizontally and vertically stacked using `patchwork`'s operators `/` and `|`, respectively.

Returns: An object of classes `gg` and `ggplot`.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Window$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Minh Long Nguyen <edelweiss611428@gmail.com>
Toby Dylan Hocking <toby.hocking@r-project.org>
Charles Truong <ctruong@ens-paris-saclay.fr>

References

Truong, C., Oudre, L., & Vayatis, N. (2020). Selective review of offline change point detection methods. Signal Processing, 167, 107299.

Examples

```
## L2 example
set.seed(1121)
signals = as.matrix(c(rnorm(100,0,1),
                      rnorm(100,5,1)))
# Default L2 cost function
WindowObj = Window$new(minSize = 1L, jump = 1L)
WindowObj$fit(signals)
WindowObj$predict(pen = 100)
WindowObj$plot()

## SIGMA example
set.seed(111)
signals = as.matrix(c(rnorm(100,-5,1),
                      rnorm(100,-5,10),
                      rnorm(100,-5,1)))
# L2 cost function
WindowObj = Window$new(minSize = 1L, jump = 1L)
WindowObj$fit(signals)
# We choose pen = 50.
WindowObj$predict(pen = 50)
WindowObj$plot()

# The standard L2 cost function is not suitable.
# Use the SIGMA cost function.
WindowObj$costFunc = costFunc$new(costFunc = "SIGMA")
WindowObj$predict(pen = 50)
WindowObj$plot()
```

Index