Package ‘blockcpd’

August 12, 2022

Title Change Point Detection for Multiple Aligned Independent Time Series

Version 1.0.0

Description Implementation of statistical models based on regularized likelihood for offline change point detection on multiple aligned independent time series. It detects changes in parameters for the specified family for the series as group or block. As a reference for the method, see Prates et al. (2021) <arXiv:2111.10187>.

License GPL (>= 2)

Encoding UTF-8

RoxygenNote 7.2.1

Imports Rcpp, graphics, stats

LinkingTo Rcpp, RcppArmadillo

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation yes

Author Lucas Prates [aut, cre] (<`https://orcid.org/0000-0002-6431-8232`>), Florencia Leonardi [aut] (<`https://orcid.org/0000-0002-0299-0680`>)

Maintainer Lucas Prates <`lucasdelprates@gmail.com`>

Repository CRAN

Date/Publication 2022-08-12 11:20:06 UTC

R topics documented:

`check_input` *Checks input from caller*

Description

Performs a sanity check on the inputs from caller. It stops execution and outputs an error message if arguments are not in conformity with caller method.

Usage

```
check_input(caller, args_to_check)
```

Arguments

`caller`	name of the function that called the check_input.
`args_to_check`	list of arguments that will be checked.

`compare_model` *Compare or evaluate model performance with respect to other model or ground truth*

Description

Compares or evaluates model estimated change point set against another model or ground truth. The comparison is made using common metrics to compare clusters. The metrics provided are

- "hausdorff": Hausdorff Distance metric;
- "rand": Rand Index ;
- "symdiff": Symmetric difference metric;
- "jaccard": Jaccard similarity index.

Usage

```
compare_model(model1, model2, ncol = NULL)
```

Arguments

`model1`	The first blockcpd object or list of sorted integers representing the change point set.
`model2`	The second blockcpd object or list of sorted integers representing the change point set.
`ncol`	The number of variables which the model was fitted on. Only needs to be passed if both arguments are change point sets instead of a blockcpd object.

Value

Returns a list containing four metrics:

- "haus" Hausdorff distance;
- "rand" Rand index;
- "symdiff" Symmetric difference;
- "jaccard" Jaccard index.

Examples

```
model1 = fit_blockcpd(c(0, 1, 0, 1), lambda = 0)
model2 = fit_blockcpd(c(0, 1, 0, 1), lambda = Inf)
comparison = compare_model(model1, model2)
# change-point sets can also be passed directly with ncol
compare_model(c(1,2,3,4), c(3), 10)
```

`compute_dynseg` *Block segmentation using dynamical programming*

Description

Computes the exact solution of the regularized loss optimization problem, providing change point locations and the parameters of each blocks. Should be called within fit_blockcpd

Usage

```
compute_dynseg(
  suff_stats,
  family,
  lambda = 1,
  nrow,
  ncol,
  min_block_size = min_block_size,
  max_blocks = ncol - 1,
  pen_func = bic_loss
)
```

Arguments

`suff_stats`	Sufficient statistics to perform change point analysis
`family`	The name of the family used to fit the model
`lambda`	Penalization constant
`nrow`	Number of rows or samples
`ncol`	Number of columns or variables
`min_block_size`	Minimum block size allowed. Default is 0, and the value must be smaller or equal to ncol.
`max_blocks`	Threshold on the number of block segments to fit the model. Set low values for this parameters if having performance issues on large data sets.
`pen_func`	A penalization function defined i integer intervals The function signature should be pen(left_index, right_index, nrow, ncol), where the left_index:right_index is the integer interval, nrow the sample size and ncol the number of variables/columns.

`compute_hausdorff` *Hausdorff distance metric*

Description

Computes the Hausdorff distance between change point sets.

Usage

```
compute_hausdorff(cp1, cp2)
```

Arguments

`cp1`	Change point set for model 1 or true change point set.
`cp2`	Change point set for model 2 or true change point set.

`compute_hierseg` *Block segmentation using hierarchical algorithm*

Description

Uses binary splitting to obtain a greedy solution to the regularized loss optimization problem. Should be called within fit_blockcpd

Usage

```
compute_hierseg(
  suff_stats,
  family,
  lambda = 1,
  nrow,
  ncol,
  pen_func = bic_loss,
  min_block_size = min_block_size,
  max_blocks = NULL
)
```

Arguments

`suff_stats`	Sufficient statistics to perform change point analysis
`family`	The name of the family used to fit the model
`lambda`	Penalization constant
`nrow`	Number of rows or samples
`ncol`	Number of columns or variables
`pen_func`	A penalization function defined i integer intervals The function signature should be pen(left_index, right_index, nrow, ncol), where the left_index:right_index is the integer interval, nrow the sample size and ncol the number of variables/columns.
`min_block_size`	Minimum block size allowed. Default is 0, and the value must be smaller or equal to ncol.
`max_blocks`	Threshold on the number of block segments to fit the model. Set low values for this parameters if having performance issues on large data sets.

`compute_jaccard` *Jaccard's Index metric*

Description

Computes the Jaccard index between two change point detection sets

Usage

```
compute_jaccard(cp1, cp2)
```

Arguments

`cp1`	Change point set for model 1 or true change point set.
`cp2`	Change point set for model 2 or true change point set.

`compute_rand` *Rand Index Function for change point detection*

Description

Computes the rand Index (non-adjusted) for the change point sets. A specific equation for change point detection is used to make the computation faster. Proof of correctness of the equation is given in the dissertation.

Usage

```
compute_rand(cp1, cp2, m)
```

Arguments

`cp1`	Change point set for model 1 or true change point set.
`cp2`	Change point set for model 2 or true change point set.
`m`	The size of the vector array.

`compute_symdiff` *Symmetric difference metric*

Description

Computes the size of the symmetric difference between two change point detection sets

Usage

```
compute_symdiff(cp1, cp2)
```

Arguments

`cp1`	Change point set for model 1 or true change point set.
`cp2`	Change point set for model 2 or true change point set.

`confidence_plot` *Plot to check reported change-points*

Description

Plots the estimates of how likely it is for the model to detect a change at any given point. True change-points should have confidence near $100%$, while non change-points should have a confidence near $0%$. It might also be difficult to detect a true change-point at the given sample size. In this case, it should fluctuate in the middle.

Usage

```
confidence_plot(
  model,
  scale = "percentage",
  index_values = NULL,
  index_variable_name = "Index",
  pkg = "base"
)
```

Arguments

`model`	A `blockcpd` model object.
`scale`	A string describing the scale which the y-scale should is plotted. Possible values are "percentage", "probability" and "frequency".
`index_values`	A numerical vector of size ncol that contains the values of the the variable corresponding to the change points.
`index_variable_name`	Name of the variable segmented.
`pkg`	Graphical package to be used for plotting. Current values are "base".

Value

No return value.

Examples

```
td = rcpd(nrow = 10, ncol = 10)
model = fit_blockcpd(td$data_matrix, bootstrap = TRUE)
confidence_plot(model)
```

`fit_blockcpd` *Fits a blockcpd model*

Description

Fits a blockcpd model to find the best segmentation of the data into blocks. Variables in each block have the same distribution and parameter, and consecutive blocks have different parameters.

Usage

```
fit_blockcpd(
  data_matrix,
  method = "hierseg",
  family = "bernoulli",
  lambda = 1,
  pen_func = bic_loss,
  min_block_size = 1L,
  max_blocks = NULL,
  bootstrap = FALSE,
  bootstrap_samples = 100L,
  bootstrap_progress = FALSE,
  skip_input_check = FALSE
)
```

Arguments

`data_matrix`	Data frame or matrix containing the data set to be segmented. There is no verification if the entries correspond to the model specified by the "family" argument, such as entries different than 0, 1 or NA for the bernoulli family.
`method`	The method that will be used to fit the model. The current implemented models are: • [hierseg] Hierarchical segmentation, also known as binary segmentation; • [dynseg] Dynamical programming segmentation.
`family`	The name of the family to detect changes in parameters. Should be passed as a string. The families currently implemented are: • "bernoulli": The model assumes that data comes from a Bernoulli distribution. For each block, the algorithm estimates the probability paramater. Each entry should be binary. • "normal": The model assumes data comes fro ma Normal distribution with unknown mean and variance. For each block, the algorithms estimates the mean and variance parameter. Each entry should be numeric. • "binaryMarkov": The model assumes that data comes from two states (0, 1) Markov Chain. For each block, the algorithm estimates the 2x2 transition matrix. Each entry should be binary. At the boundary of the blocks, the transition is defined using the parameters of the next (new) block. For

instance, consider a block defined from a to c, followed a block from c + 1 to b (including the extremes). By definition, c is a change point, and the transition from X_c to X_c + 1 is defined by the parameters on c + 1 to b.

- "exponential": The model assumes that data comes from an Exponential distribution. For each block, the algorithm estimates the scale parameter, that is, the inverse of the rate. Each entry should be numeric and positive.
- "poisson": The model assumes that data comes from a Poisson distribution For each block, the algorithm estimates the rate paramater. Each entry should an positive integer.

`lambda`	The penalization constant. Must be a unique non-negative numeric value.
`pen_func`	Regularization function used for fitting, with default as the BIC. For user specified functions, check the template in the regularization regularization.rd file.
`min_block_size`	Minimum block size allowed. Default is 1, and the value must be smaller or equal to ncol.
`max_blocks`	An integer greater than 0 that specify the maximum number of blocks fitted by the algorithm. It is only used if dynseg is specified in the "method" argument.
`bootstrap`	A flag to decide if bootstrap computations for the estimation of the probability of each index being detected as a change point. It also provides a sample of all the metrics implemented computed with respect to the final change point set estimated.
`bootstrap_samples`	Number of bootstrap samples.
`bootstrap_progress`	Flag for bootstrap progress printing.
`skip_input_check`	Flag indicating if input checking should be skipped.

Value

The function returns a S3 object of the type blockcpd.

- "changepoints" a list containing the set of estimated change points;
- "parameters" a list containing the estimated parameters for each block. In the case of multiple parameters, it provides a list of lists, where each sub list refers to the parameter that names the list;
- "loss" the final loss evaluated on the entire data set for the returned model;
- "neg_loglike" The negative log likelihood of the model;
- "ncp" number of change points estimated;
- "metadata" Arguments passed to fit the model;
- "bootstrap_info" if bootstrap argument is true, this contains a list of the metrics for each bootstrap sample, and contains the estimated probability of each index being detected as a change point;

Examples

```
fit_blockcpd(c(0, 1, 2, 10, 11), family = "normal", lambda = 1) # single series
fit_blockcpd(matrix(c(0, 1, 0, 0, 0, 0, 1, 1), nrow = 2)) # 2 binary series
```

plot.blockcpd *Plot for blockcpd object*

Description

Plots the selected parameters in a blocked fashion.

Usage

```
## S3 method for class 'blockcpd'
plot(
  x,
  ...,
  parameter = NULL,
  index_values = NULL,
  index_variable_name = "Index",
  pkg = "base"
)
```

Arguments

`x`	A fitted blockcpd S3 object provided by the fit_blockcpd function.
`...`	Other parameters
`parameter`	The parameter of the family for which to plot the blocked
`index_values`	A numerical vector of size ncol that contains the values of the the variable corresponding to the change points. For example, if your segmented variable corresponds to a time samples from 0 to 150 sampled each 15 seconds, the model treats these as values from 1 to 11. To plot on the variable scale, pass the argument 'index_values = seq(0, 150, 15)'.
`index_variable_name`	Name of the variable segmented.
`pkg`	Graphical package to be used for plotting. Current values are "base".

Value

No return value.

Examples

```
plot(fit_blockcpd(c(1,2,3, 4), family = "exponential", lambda = 0))
```

`plot.frv` *Plot for graphical selection of the constant*

Description

Plots the output of a frv object. It shows how the number of change-points estimated by the given model vary with the regularization constant lambda. Graphical inspection can be used to choose a proper value for the constant. The suggestion is to pick a value in which the curve starts to "flat-out"

Usage

```
## S3 method for class 'frv'
plot(x, ..., pkg = "base")
```

Arguments

`x`	An object returned from the function select_frv
`...`	Other parameters
`pkg`	Graphical package to be used for plotting. Current values are "base".

Value

No return value.

Examples

```
td = rcpd(nrow = 10, ncol = 10)
frv = select_frv(td$data_matrix)
plot(frv)
```

`rcpd` *Sampler for the CPD Block Model*

Description

Creates a $nrow \times ncol$ matrix with ncp change points. In between change points, the random variables are i.i.d. sampled from the given family and parameters

Usage

```
rcpd(
  nrow = 100,
  ncol = 50,
  ncp = 1,
  family = "bernoulli",
  parameters = NULL,
  changepoints = NULL,
  prob_NA = 0
)
```

Arguments

`nrow`	Number of rows, or sample size, of the data.
`ncol`	Number of columns of data matrix. It is the number of variables for each sample.
`ncp`	Number of change points. The number of blocks is $ncp + 1$. It is overridden if changepoints is non-NULL.
`family`	The family model to be sampled. The families currently implemented are: • bernoulli: Sample independent Bernoullis with probability parameter of the block • normal: Sample independent Normal with mean and variance specified by the block. • binaryMarkov: Samples a two state Markov Chain process with transition matrix defined by the block. • exponential: Sample independent Exponential with scale parameter defined by the block. • poisson: Sample independent Poisson with rate parameter defined by the block.
`parameters`	List of parameters containing $ncp + 1$ dimensional parameter vectors of each block. If NULL, the parameters are sampled randomly.
`changepoints`	A sorted vector of size ncp containing integers as change point locations. The change points are between 1 and $ncol - 1$. If NULL, the change points are sampled uniformly in $[1, ncol - 1]$.
`prob_NA`	Probability of each entry of being NA. Default is 0.

Value

Returns a list containing 3 elements: #'

- "data_matrix" A matrix containing the data.
- "changepoints" A numeric vector containing the change-point locations
- "parameters" A list whose keys are the parameters names and the values are vectors containing the parameter for each block.

Examples

```
td = rcpd(nrow = 20, ncol = 10) # 20 Bernoulli series of size 10 with 1 change-point
td = rcpd(nrow = 10, ncol = 100, ncp = 5,
          family = "normal") # 10 normal series of size 100 with 5 change-points
td = rcpd(nrow = 1000, ncol = 100, changepoints = c(10, 40, 79)) # choosing change-points locations
td = rcpd(nrow = 100, ncol = 15, ncp = 2, family = "normal",
          parameters = list(mean = c(1, 2, 3), var = c(4, 5, 6))) # choosing parameters
```

---

select_frv *Methodology to aid choosing regularization constant*

Description

Aids in the selection of the penalization constants, possibly providing an automatic optimal value. It analyses how the number of change-points vary with the chosen grid of penalization constant. It applied the First Repeated Value (FRV) methodology to select the regularization constant lambda. It is similar to the Elbow method used in clustering, or the CROPS algorithm in change-point detection. The values of the constant range from 'lambda_left' to 'lambda_right', increasing by 'step'. For each value, the function fit_blockcpd is run with arguments 'model_args'. An automatic suggestion for the penalization, number of change-points and model is given automatically. Optionally, The user can call the plot function to the output of this method so he can use an elbow plot like graphical inspection to select the constant value.

Usage

```
select_frv(
  data_matrix,
  lambda_left = 0,
  lambda_right = 10,
  step = "automatic",
  model_args = list()
)
```

Arguments

`data_matrix`	Data frame or matrix containing the data set to be segmented.
`lambda_left`	Left most value of lambda. Must be non-negative.
`lambda_right`	Right most value of lambda. Must be non-negative and greater than lambda_left.
`step`	Value by which lambda will be increased. Must be greater than 0, The default is 'automatic', which consists of a penalization of 1/sqrt(log(n)), where n is the number of samples (rows).
`model_args`	A list with argument values for the fit_blockcpd function. The list keys must be the arguments names. It must *not* contain the argument 'lambda' or 'data_matrix'.

Value

Returns a frv object containing the suggested values and caller parameters.

toy_regularization *Implements the regularization functions used in the estimation*

Description

The estimator in this package computes the optimum of $-l(C,p)+\lambda*R(leftIndex, rightIndex, nrow, ncol)$, where l is the log likelihood of the family, lambda is the penalization constant and R is the regularization function. The user can create his own regularization function and pass as an argument to fit_blockcpd. It should have four arguments, in the following order: left_index, right_index, nrow and ncol. Each argument is explained in the parameter section. If the function depends on leftIndex and rightIndex, it will be non-homogeneous, which might be interesting in some applications. The package implements some functions as an example, but uses only bic_loss as the default. The algorithm is consistent as long as the the regularization is bounded by a constant.

Usage

```
toy_regularization(left_index, right_index, nrow, ncol)
```

Arguments

`left_index`	First index of the interval
`right_index`	Last index of the interval
`nrow`	Number of rows/signals/series
`ncol`	Number of columns/variables

Examples

```
my_reg <- function(leftIndex, rightIndex, nrow, ncol){
    block_size = (rightIndex - leftIndex + 1)
    return(log(nrow*ncol)*(1/block_size))
}
```

Index