Package 'crossvalidationCP'

July 23, 2025

Title Cross-Validation for Change-Point Regression

Version 1.1

Depends R (>= 3.3.0)

Imports changepoint (>= 2.0), fpopw(>= 1.1), wbs (>= 1.4), stats

Suggests testthat (>= 2.0.0)

Description Implements the cross-validation methodology from Pein and Shah (2021) <doi:10.48550/arXiv.2112.03220>. Can be customised by providing different cross-validation criteria, estimators for the change-point locations and local parameters, and freely chosen folds. Pre-implemented estimators and criteria are available. It also includes our own implementation of the COPPS procedure <doi:10.1214/19-AOS1814>.

License GPL-3

NeedsCompilation no

Author Pein Florian [aut, cre]

Maintainer Pein Florian <f.pein@lancaster.ac.uk>

Repository CRAN

Date/Publication 2023-05-22 18:30:02 UTC

Contents

`crossvalidationCP-package` *Cross-validation for change-point regression*

Description

Implements the cross-validation methodology from *Pein and Shah (2021)*. The approach can be customised by providing cross-validation criteria, estimators for the change-point locations and local parameters, and freely chosen folds. Pre-implemented estimators and criteria are available. It also includes our own implementation of the COPPS procedure *Zou et al. (2020)*. By default, 5-fold cross-validation with ordered folds, absolute error loss, and least squares estimation for estimating the change-point locations is used.

Details

The main function is `crossvalidationCP`. It selects among a list of parameters the one with the smallest cross-validation criterion for a given method. The user can freely choose the folds, the local estimator and the criterion. Several pre-implemented estimators and criteria are available. Estimators have to allow a list of parameters at the same time. One can use `convertSingleParam` to convert a function allowing only a single parameter to a function that allows a list of parameters.

A ssimpler, but more limited access is given by the functions `VfoldCV`, `COPPS`, `CV1` and `CVmod`. `VfoldCV` performs V-fold cross-validation, where the tuning parameter is directly the number of change-points. `COPPS` implements the COPPS procedure *Zou et al. (2020)*, i.e. 2-fold cross-validation with Order-Preserved Sample-Splitting and the tuning parameter being again the number of change-points. `CV1` and `CVmod` do the same, but with absolute error loss and the modified quadratic error loss, see (15) and (16) in *Pein and Shah (2021)*, instead of quadratic error loss.

Note that `COPPS` can be problematic when larger changes occur at odd locations. For a detailed discussion, why standard quadratic error loss can lead to misestimation, see Section 2 in *Pein and Shah (2021)*. By default, we recommend to use absolute error loss and 5-fold cross-validation as offered by `VfoldCV`.

So far only univariate data is supported, but support for multivariate data is planned.

References

Pein, F., and Shah, R. D. (2021) Cross-validation for change-point regression: pitfalls and solutions. *arXiv:2112.03220*.

Zou, C., Wang, G., and Li, R. (2020) Consistent selection of the number of change-points via sample-splitting. *The Annals of Statistics*, **48**(1), 413–439.

See Also

`crossvalidationCP`, estimators, criteria, `convertSingleParam`, `VfoldCV`, `COPPS`, `CV1`, `CVmod`

Examples

```
# call with default parameters:
# 5-fold cross-validation with absolute error loss, least squares estimation,
# and possible parameters being 0 to 5 change-points
Y <- rnorm(100)
(ret <- crossvalidationCP(Y = Y))
# a simpler, but more limited access to it is offered by VfoldCV()
identical(VfoldCV(Y = Y), ret)

# more interesting data and more detailed output
set.seed(1L)
Y <- c(rnorm(50), rnorm(50, 5), rnorm(50), rnorm(50, 5))
VfoldCV(Y = Y, output = "detailed")
# finds the correct change-points at 50, 100, 150
# (plus the start and end points 0 and 200)

# reducing the maximal number of change-points to 2
VfoldCV(Y = Y, Kmax = 2)

# crossvalidationCP is more flexible and allows a list of parameters
# here only 1 or 2 change-points are allowed
crossvalidationCP(Y = Y, param = as.list(1:2))

# reducing the number of folds to 3
ret <- VfoldCV(Y = Y, V = 3L, output = "detailed")
# the same but with explicitly specified folds
identical(crossvalidationCP(Y = Y, folds = list(seq(1, 200, 3), seq(2, 200, 3), seq(3, 200, 3)),
                            output = "detailed"), ret)

# 2-fold cross-validation with Order-Preserved Sample-Splitting
ret <- crossvalidationCP(Y = Y, folds = "COPPS", output = "detailed")

# a simpler access to it is offered by CV1()
identical(CV1(Y = Y, output = "detailed"), ret)

# different criterion: quadratic error loss
ret <- crossvalidationCP(Y = Y, folds = "COPPS", output = "detailed", criterion = criterionL2loss)

# same as COPPS procedure; as offered by COPPS()
identical(COPPS(Y = Y, output = "detailed"), ret)

# COPPS potentially fails to provide a good selection when large changes occur at odd locations
# Example 1 in (Pein and Shah, 2021), see Section 2.2 in this paper for more details
set.seed(1)
exampleY <- rnorm(102, c(rep(10, 46), rep(0, 5), rep(30, 51)))
# misses one change-point
crossvalidationCP(Y = exampleY, folds = "COPPS", criterion = criterionL2loss)

# correct number of change-points when modified criterion (or absolute error loss) is used
(ret <- crossvalidationCP(Y = exampleY, folds = "COPPS", criterion = criterionMod))

# a simpler access to it is offered by CVmod()
```

```
identical(CVmod(Y = exampleY), ret)

# manually given criterion; identical to criterionL1loss()
testCriterion <- function(testset, estset, value = NULL, ...) {
  if (!is.null(value)) {
    return(sum(abs(testset - value)))
  }

  sum(abs(testset - mean(estset)))
}
identical(crossvalidationCP(Y = Y, criterion = testCriterion, output = "detailed"),
          crossvalidationCP(Y = Y, output = "detailed"))

# PELT as a local estimator instead of least squares estimation
# param must contain parameters that are acceptable for the given estimator
crossvalidationCP(Y = Y, estimator = pelt, output = "detailed",
                  param = list("SIC", "MBIC", 3 * log(length(Y))))

# argument minseglen of pelt specified in ...
crossvalidationCP(Y = Y, estimator = pelt, output = "detailed",
                  param = list("SIC", "MBIC", 3 * log(length(Y))), minseglen = 60)
```

`convertSingleParam` *Provides estimators that allows list of parameters*

Description

Converts estimators allowing single parameters to estimators allowing a list of parameters. The resulting function can be passed to the argument `estimator` in the cross-validation functions, see *See Also.*

Usage

```
convertSingleParam(estimator)
```

Arguments

`estimator`	the function to be converted, i.e. a function providing a local estimate. The function must have the arguments `Y`, `param` and `...`, where `Y` will be the observations, and `param` a single parameter of arbitrary type. Hence `lists` can be used when multiple parameter of different types are needed. It has to return either a vector with the estimated change-points or a list containing the named entries `cps` and `value`. In this case `cps` has to be a numeric vector with the estimated change-points as before and `value` has to be a list of length one entry longer than `cps` giving the locally estimated values. An example is given below.

Value

a function that can be passed to the argument `estimator` in the cross-validation functions, see the functions listed in *See Also*

References

Pein, F., and Shah, R. D. (2021) Cross-validation for change-point regression: pitfalls and solutions. *arXiv:2112.03220*.

See Also

crossvalidationCP, VfoldCV, COPPS, CV1, CVmod

Examples

```
# wrapper around pelt to demonstrate an estimator that allows a single parameter only
singleParamEstimator <- function(Y, param, minseglen = 1, ...) {
  if (is.numeric(param)) {
   ret <- changepoint::cpt.mean(data = Y, penalty = "Manual", pen.value = param, method = "PELT",
                                minseglen = minseglen)
  } else {
   ret <- changepoint::cpt.mean(data = Y, penalty = param, method = "PELT", minseglen = minseglen)
  }

  list(cps = ret@cpts[-length(ret@cpts)], value = as.list(ret@param.est$mean))
}
# conversion to an estimator that is suitable for crossvalidationCP() etc.
estimatorMultiParam <- convertSingleParam(singleParamEstimator)
crossvalidationCP(rnorm(100), estimator = estimatorMultiParam, param = list("SIC", "MBIC"))
```

COPPS *Cross-validation with Order-Preserved Sample-Splitting*

Description

Tuning parameters are selected by a generalised COPPS procedure. All functions use Order-Preserved Sample-Splitting, meaning that the folds will be the odd and even indexed observations. The three functions differ in which cross-validation criterion they are using. COPPS is the original COPPS procedure *Zou et al. (2020)*, i.e. uses quadratic error loss. CV1 and CVmod use absolute error loss and the modified quadratic error loss, respectively.

Usage

```
COPPS(Y, param = 5L, estimator = leastSquares,
      output = c("param", "fit", "detailed"), ...)
CV1(Y, param = 5L, estimator = leastSquares,
    output = c("param", "fit", "detailed"), ...)
CVmod(Y, param = 5L, estimator = leastSquares,
      output = c("param", "fit", "detailed"), ...)
```

Arguments

Y	the observations, can be any data type that supports the function length and the operator [] and can be passed to estimator and the cross-validation criterion, e.g. a numeric vector or a list. Support for matrices, i.e. for multivariate data, is planned but not implemented so far
param	a list giving the possible tuning parameters. Alternatively, a single integer which will be interpreted as the maximal number of change-points and converted to as.list(0:param)
estimator	a function providing a local estimate. For pre-implemented estimators see estimators. The function must have the arguments Y, param and ..., where Y will be a subset of the observations, and param and ... will be the corresponding arguments of the called function. Note that ... will be passed to estimator and the cross-validation criterion. The return value must be either a list of length length(param) with each entry containing the estimated change-point locations for the given entry in param or a list containing the named entries cps and value. In this case cps has to be a list of the estimated change-points as before and value has to be a list of the locally estimated values for each entry in param, i.e. each list entry has to be a list itself of length one entry longer than the corresponding entry in cps. The function convertSingleParam offers the conversion of an estimator allowing a single parameter into an estimator allowing multiple parameters
output	a string specifying the output, either "param", "fit" or "detailed". For details what they mean see *Value*
...	additional parameters that are passed to estimator and the cross-validation criterion

Value

if output == "param", the selected tuning parameter, i.e. an entry from param. If output == "fit", a list with the entries param, giving the selected tuning parameter, and fit. The named entry fit is a list giving the returned fit obtained by applying estimator to the whole data Y with the selected tuning parameter. The returned value is transformed to a list with an entry cps giving the estimated change-points and, if provided by estimator, an entry value giving the estimated local values. If output == "detailed", the same as for output == "fit", but additionally the entries CP, CVodd, and CVeven giving the calculated cross-validation criteria for all parameter entries. CVodd and CVeven are the criteria when the odd / even observations are in the test set, respectively. CP is the sum of those two.

References

Pein, F., and Shah, R. D. (2021) Cross-validation for change-point regression: pitfalls and solutions. *arXiv:2112.03220*.

Zou, C., Wang, G., and Li, R. (2020) Consistent selection of the number of change-points via sample-splitting. *The Annals of Statistics*, **48**(1), 413–439.

See Also

estimators, criteria, convertSingleParam

Examples

```
# call with default parameters:
# 2-folds cross-validation with ordereded folds, absolute error loss,
# least squares estimation, and possible parameters being 0 to 5 change-points
CV1(Y = rnorm(100))
# the same, but with modified error loss
CVmod(Y = rnorm(100))
# the same, but with quadratic error loss, indentical to COPPS procedure
COPPS(Y = rnorm(100))

# more interesting data and more detailed output
set.seed(1L)
Y <- c(rnorm(50), rnorm(50, 5), rnorm(50), rnorm(50, 5))
CV1(Y = Y, output = "detailed")
# finds the correct change-points at 50, 100, 150
# (plus the start and end points 0 and 200)

# list of parameters, only allowing 1 or 2 change-points
CVmod(Y = Y, param = as.list(1:2))

# COPPS potentially fails to provide a good selection when large changes occur at odd locations
# Example 1 in (Pein and Shah, 2021), see Section 2.2 in this paper for more details
set.seed(1)
exampleY <- rnorm(102, c(rep(10, 46), rep(0, 5), rep(30, 51)))
# misses one change-point
COPPS(Y = exampleY)

# correct number of change-points when modified criterion (or absolute error loss) is used
CVmod(Y = exampleY)

# PELT as a local estimator instead of least squares estimation
# param must contain parameters that are acceptable for the given estimator
CV1(Y = Y, estimator = pelt, output = "detailed", param = list("SIC", "MBIC", 3 * log(length(Y))))

# argument minseglen of pelt specified in ...
CVmod(Y = Y, estimator = pelt, output = "detailed", param = list("SIC", "MBIC", 3 * log(length(Y))),
                  minseglen = 30)
```

`criteria` *Pre-implemented cross-validation criteria*

Description

`criterionL1loss`, `criterionMod` and `criterionL2loss` compute the cross-validation criterion with L1-loss, the modified criterion and the criterion with L2-loss for univariate data, see (15), (16), and (6) in *Pein and Shah (2021)*, respectively. If `value` is given (i.e. `value =! NULL`), then `value` replaces the empirical means. All criteria can be passed to the argument `criterion` in the cross-validation functions, see the functions listed in *See Also*.

Usage

```
criterionL1loss(testset, estset, value = NULL, ...)
criterionMod(testset, estset, value = NULL, ...)
criterionL2loss(testset, estset, value = NULL, ...)
```

Arguments

testset	a numeric vector giving the observations in the test set / fold. For criterionMod, if length(testset) == 1L, NaN will be returned, see *Details*
estset	a numeric vector giving the observations in the estimation set
value	a single numeric giving the local value on the segment or NULL. If NULL the value will be mean(estset)
...	unused

Details

criterionMod requires that the minimal segment length is at least 2. So far the only pre-implemented estimators that allows for such an option are pelt and binseg, where one can specify minseglen in ....

Value

a single numeric

References

Pein, F., and Shah, R. D. (2021) Cross-validation for change-point regression: pitfalls and solutions. *arXiv:2112.03220*.

See Also

crossvalidationCP, VfoldCV, COPPS, CV1, CVmod

Examples

```
# all functions can be called directly, e.g.
Y <- rnorm(100)
criterionL1loss(testset = Y[seq(1, 100, 2)], estset = Y[seq(2, 100, 2)])

# but their main purpose is to serve as the criterion in the cross-validation functions, e.g.
crossvalidationCP(rnorm(100), criterion = criterionL1loss)
```

`crossvalidationCP` *Cross-validation in change-point regression*

Description

Generic function for cross-validation to select tuning parameters in change-point regression. It selects among a list of parameters the one with the smallest cross-validation criterion for a given method. The cross-validation criterion, the estimator, and the the folds can be specified by the user.

Usage

```
crossvalidationCP(Y, param = 5L, folds = 5L, estimator = leastSquares,
                  criterion = criterionL1loss,
                  output = c("param", "fit", "detailed"), ...)
```

Arguments

`Y`	the observations, can be any data type that supports the function `length` and the operator `[]` and can be passed to `estimator` and `criterion`, e.g. a numeric vector or a list. Support for `matrices`, i.e. for multivariate data, is planned but not implemented so far
`param`	a `list` giving the possible tuning parameters. Alternatively, a single integer which will be interpreted as the maximal number of change-points and converted to `as.list(0:param)`. All values have to be acceptable values for the specified `estimator`
`folds`	either a `list`, a single integer or the string `"COPPS"` specifying the folds. If a `list`, each entry should be an integer vector with values between 1 and `length(Y)` giving the indices of the observations in the fold. A single integer specifies the number of folds and ordered folds are automatically created, i.e. fold `i` will be `seq(i, length(Y), folds)`. `"COPPS"` means that a generalised COPPS procedure *Zou et al. (2020)* will be used, i.e. 2-fold cross-validation with Order-Preserved Sample-Splitting, meaning that the folds will be the odd and even indexed observations. Note that observations will be given in reverse order to the cross-validation criterion when the odd-indexed observations are in the test set. This allows criteria such as the modified criterion, where for the odd-indexed the first and for the even-indexed the last observation is removed
`estimator`	a function providing a local estimate. For pre-implemented estimators see estimators. The function must have the arguments `Y`, `param` and `...`, where `Y` will be a subset of the observations, and `param` and `...` will be the corresponding arguments of the called function. Note that `...` will be passed to `estimator` and `criterion`. The return value must be either a list of length `length(param)` with each entry containing the estimated change-point locations for the given entry in `param` or a list containing the named entries `cps` and `value`. In this case `cps` has to be a list of the estimated change-points as before and `value` has to be a list of the locally estimated values for each entry in `param`, i.e. each list entry has to be a list itself of length one entry longer than the corresponding entry in

cps. The function convertSingleParam offers the conversion of an estimator allowing a single parameter into an estimator allowing multiple parameters

criterion — a function providing the cross-validation criterion. For pre-implemented criteria see criteria. The function must have the arguments testset, estset and value. testset and estset are the observations of one segment that are in the test and estimation set, respectively. value is the local parameter on the segment if provided by estimator, otherwise NULL. Additionally, ... is possible and potentially necessary to absorb arguments, since the argument ... of crossvalidationCP will be passed to estimator and criterion. It must return a single numeric. All return values will be summed accordingly and which.min will be called on the vector to determine the parameter with the smallest criterion, hence some NaN values etc. are allowed

output — a string specifying the output, either "param", "fit" or "detailed". For details what they mean see *Value*

... — additional parameters that are passed to estimator and criterion

Value

if output == "param", the selected tuning parameter, i.e. an entry from param. If output == "fit", a list with the entries param, giving the selected tuning parameter, and fit. The named entry fit is a list giving the returned fit obtained by applying estimator to the whole data Y with the selected tuning parameter. The retured value is transformed to a list with an entry cps giving the estimated change-points and, if provided by estimator, an entry value giving the estimated local values. If output == "detailed", the same as for output == "fit", but additionally an entry CP giving all calculated cross-validation criteria. Those values are summed over all folds

References

Pein, F., and Shah, R. D. (2021) Cross-validation for change-point regression: pitfalls and solutions. *arXiv:2112.03220*.

Zou, C., Wang, G., and Li, R. (2020) Consistent selection of the number of change-points via sample-splitting. *The Annals of Statistics*, **48**(1), 413–439.

See Also

estimators, criteria, convertSingleParam, VfoldCV, COPPS, CV1, CVmod

Examples

```
# call with default parameters:
# 5-fold cross-validation with absolute error loss, least squares estimation,
# and possible parameters being 0 to 5 change-points
# a simpler access to it is offered by VfoldCV()
crossvalidationCP(Y = rnorm(100))

# more interesting data and more detailed output
set.seed(1L)
Y <- c(rnorm(50), rnorm(50, 5), rnorm(50), rnorm(50, 5))
crossvalidationCP(Y = Y, output = "detailed")
```

```
# finds the correct change-points at 50, 100, 150
# (plus the start and end points 0 and 200)

# list of parameters, only allowing 1 or 2 change-points
crossvalidationCP(Y = Y, param = as.list(1:2))

# reducing the number of folds to 3
ret <- crossvalidationCP(Y = Y, folds = 3L, output = "detailed")
# the same but with explicitly specified folds
identical(crossvalidationCP(Y = Y, folds = list(seq(1, 200, 3), seq(2, 200, 3), seq(3, 200, 3)),
                            output = "detailed"), ret)

# 2-fold cross-validation with Order-Preserved Sample-Splitting
ret <- crossvalidationCP(Y = Y, folds = "COPPS", output = "detailed")

# a simpler access to it is offered by CV1()
identical(CV1(Y = Y, output = "detailed"), ret)

# different criterion: quadratic error loss
ret <- crossvalidationCP(Y = Y, folds = "COPPS", output = "detailed", criterion = criterionL2loss)

# same as COPPS procedure; as offered by COPPS()
identical(COPPS(Y = Y, output = "detailed"), ret)

# COPPS potentially fails to provide a good selection when large changes occur at odd locations
# Example 1 in (Pein and Shah, 2021), see Section 2.2 in this paper for more details
set.seed(1)
exampleY <- rnorm(102, c(rep(10, 46), rep(0, 5), rep(30, 51)))
# misses one change-point
crossvalidationCP(Y = exampleY, folds = "COPPS", criterion = criterionL2loss)

# correct number of change-points when modified criterion (or absolute error loss) is used
(ret <- crossvalidationCP(Y = exampleY, folds = "COPPS", criterion = criterionMod))

# a simpler access to it is offered by CVmod()
identical(CVmod(Y = exampleY), ret)

# manually given criterion; identical to criterionL1loss()
testCriterion <- function(testset, estset, value = NULL, ...) {
  if (!is.null(value)) {
    return(sum(abs(testset - value)))
  }

  sum(abs(testset - mean(estset)))
}
identical(crossvalidationCP(Y = Y, criterion = testCriterion, output = "detailed"),
          crossvalidationCP(Y = Y, output = "detailed"))

# PELT as a local estimator instead of least squares estimation
# param must contain parameters that are acceptable for the given estimator
crossvalidationCP(Y = Y, estimator = pelt, output = "detailed",
                  param = list("SIC", "MBIC", 3 * log(length(Y))))
```

```
# argument minseglen of pelt specified in ...
crossvalidationCP(Y = Y, estimator = pelt, output = "detailed",
                  param = list("SIC", "MBIC", 3 * log(length(Y))), minseglen = 60)
```

`estimators` *Pre-implemented estimators*

Description

Pre-implemented change-point estimators that can be passed to the argument `estimator` in the cross-validation functions, see the functions listed in *See Also*.

Usage

```
leastSquares(Y, param, ...)
pelt(Y, param, ...)
binseg(Y, param, ...)
wbs(Y, param, ...)
```

Arguments

`Y`	a numeric vector giving the observations
`param`	a `list` giving the possible tuning parameters. See *Details* to see which tuning parameters are allowed for which function
`...`	additional arguments, see *Details* to see which arguments are allowed for which function

Details

`leastSquares` implements least squares estimation by using the segment neighbourhoods algorithm with functional pruning from *Rigaill (20015)*, see also *Auger and Lawrence (1989)* for the original segment neighbourhoods algorithm. It calls `Fpsn`. Each list entry in `param` has to be a single integer giving the number of change-points.

`optimalPartitioning` is outdated. It will give the same results as `leastSquares`, but is slower. It is part of the package for backwards compatibility only.

`pelt` implements PELT *(Killick et al., 2012)*, i.e. penalised maximum likelihood estimation computed by a pruned dynamic program. For each list entry in `param` it calls `cpt.mean` with `method = "PELT"` and `penalty = param[[i]]` or when `param[[i]]` is a numeric with `penalty = "Manual"` and `pen.value = param[[i]]`. Hence, each entry in `param` must be a single numeric or an argument that can be passed to `penalty`. Additionally `minseglen` can be specified in `...`, by default `minseglen = 1`.

`binseg` implements binary segmentation *(Vostrikova, 1981)*. The call is the same as for `pelt`, but with `method = "BinSeg"`. Additionally, the maximal number of change-points `Q` can be specified in `...`, by default `Q = 5`. Alternatively, each list entry of `param` can be a list itself containing the named entries `penalty` and `Q`. Note that this estimator differs from binary segmentation in *Zou et al. (2020)*, it requires a penalty instead of a given number of change-points. Warnings that `Q` is chosen

too small are suppressed when `Q` is given in `param`, but not when it is a global parameter specified in `...` or `Q = 5` by default.

`wbs` implements wild binary segmentation *(Fryzlewicz, 2014)*. It calls `changepoints` with `th.const = param`, hence `param` has to be a list of positive scalars. Additionally, `...` will be passed.

Value

For `leastSquares` and `wbs` a list of length `length(param)` with each entry containing the estimated change-point locations for the given entry in `param`. For the other functions a list containing the named entries `cps` and `value`, with `cps` a list of the estimated change-points as before and `value` a list of the locally estimated values for each entry in `param`, i.e. each list entry is a list itself of length one entry longer than the corresponding entry in `cps`.

References

Pein, F., and Shah, R. D. (2021) Cross-validation for change-point regression: pitfalls and solutions. *arXiv:2112.03220*.

Rigaill, G. (2015) A pruned dynamic programming algorithm to recover the best segmentations with 1 to Kmax change-points. *Journal de la Societe Francaise de Statistique* **156**(4), 180–205.

Auger, I. E., Lawrence, C. E. (1989) Algorithms for the Optimal Identification of Segment Neighborhoods. *Bulletin of Mathematical Biology*, **51**(1), 39–54.

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

Vostrikova, L. Y. (1981). Detecting 'disorder' in multidimensional random processes. *Soviet Mathematics Doklady*, **24**, 55–59.

Fryzlewicz, P. (2014) Wild binary segmentation for multiple change-point detection. *The Annals of Statistics*, **42**(6), 2243–2281.

Zou, C., Wang, G., and Li, R. (2020). Consistent selection of the number of change-points via sample-splitting. *The Annals of Statistics*, **48**(1), 413–439.

See Also

`crossvalidationCP`, `VfoldCV`, `COPPS`, `CV1`, `CVmod`

Examples

```
# all functions can be called directly, e.g.
leastSquares(Y = rnorm(100), param = 2)

# but their main purpose is to serve as a local estimator in the cross-validation functions, e.g.
crossvalidationCP(rnorm(100), estimator = leastSquares)

# param must contain values that are suitable for the given estimator
crossvalidationCP(rnorm(100), estimator = pelt, param = list("SIC", "MBIC"))
```

VfoldCV *V-fold cross-validation*

Description

Selects the number of change-points by minimizing a V-fold cross-validation criterion. The criterion, the estimator, and the number of folds can be specified by the user.

Usage

```
VfoldCV(Y, V = 5L, Kmax = 8L, adaptiveKmax = TRUE, tolKmax = 3L, estimator = leastSquares,
       criterion = criterionL1loss, output = c("param", "fit", "detailed"), ...)
```

Arguments

Y	the observations, can be any data type that supports the function length and the operator [] and can be passed to estimator and criterion, e.g. a numeric vector or a list. Support for matrices, i.e. for multivariate data, is planned but not implemented so far
V	a single integer giving the number of folds. Ordered folds will automatically be created, i.e. fold i will be seq(i, length(Y), folds)
Kmax	a single integer giving maximal number of change-points
adaptiveKmax	a single logical indicating whether Kmax should be chosen adaptively. If true Kmax will be double if the estimated number of change-points is not at least Kmax - tolKmax
tolKmax	a single integer specifiying how much the estimated number of change-points have to be smaller than Kmax
estimator	a function providing a local estimate. For pre-implemented estimators see estimators. The function must have the arguments Y, param and ..., where Y will be a subset of the observations, param will be list(0:Kmax), and ... will be the argument ... of VfoldCV. Note that ... will be passed to estimator and criterion. The return value must be either a list of length length(param) with each entry containing the estimated change-point locations for the given entry in param or a list containing the named entries cps and value. In this case cps has to be a list of the estimated change-points as before and value has to be a list of the locally estimated values for each entry in param, i.e. each list entry has to be a list itself of length one entry longer than the corresponding entry in cps. The function convertSingleParam offers the conversion of an estimator allowing a single parameter into an estimator allowing multiple parameters. From the currently pre-implemented estimators only leastSquares accepts param == list(0:Kmax). Estimators that allow param to differ from list(0:Kmax) can be used in crossvalidationCP
criterion	a function providing the cross-validation criterion. For pre-implemented criteria see criteria. The function must have the arguments testset, estset and value. testset and estset are the observations of one segment that are in the test

and estimation set, respectively. `value` is the local parameter on the segment if provided by `estimator`, otherwise `NULL`. Additionally, `...` is possible and potentially necessary to absorb arguments, since the argument `...` of `VfoldCV` will be passed to `estimator` and `criterion`. It must return a single numeric. All return values will be summed accordingly and `which.min` will be called on the vector to determine the parameter with the smallest criterion. Hence some `NaN` values etc. are allowed

`output` a string specifying the output, either `"param"`, `"fit"` or `"detailed"`. For details what they mean see *Value*

`...` additional parameters that are passed to `estimator` and `criterion`

Value

if `output == "param"`, the selected number of change-points, i.e. an integer between `0` and `Kmax`. If `output == "fit"`, a list with the entries `param`, giving the selected number of change-points, and `fit`. The named entry `fit` is a list giving the returned fit obtained by applying `estimator` to the whole data `Y` with the selected tuning parameter. The returned value is transformed to a list with an entry `cps` giving the estimated change-points and, if provided by `estimator`, an entry `value` giving the estimated local values. If `output == "detailed"`, the same as for `output == "fit"`, but additionally an entry `CP` giving all calculated cross-validation criteria. Those values are summed over all folds

References

Pein, F., and Shah, R. D. (2021) Cross-validation for change-point regression: pitfalls and solutions. *arXiv:2112.03220*.

See Also

estimators, criteria, `convertSingleParam`

Examples

```
# call with default parameters:
# 5-fold cross-validation with absolute error loss, least squares estimation,
# and 0 to 5 change-points
VfoldCV(Y = rnorm(100))

# more interesting data and more detailed output
set.seed(1L)
Y <- c(rnorm(50), rnorm(50, 5), rnorm(50), rnorm(50, 5))
VfoldCV(Y = Y, output = "detailed")
# finds the correct change-points at 50, 100, 150
# (plus the start and end points 0 and 200)

# reducing the number of folds to 3
VfoldCV(Y = Y, V = 3L, output = "detailed")

# reducing the maximal number of change-points to 2
VfoldCV(Y = Y, Kmax = 2)
```

```
# different criterion: modified error loss
VfoldCV(Y = Y, output = "detailed", criterion = criterionMod)

# manually given criterion; identical to criterionL1loss()
testCriterion <- function(testset, estset, value = NULL, ...) {
  if (!is.null(value)) {
    return(sum(abs(testset - value)))
  }

  sum(abs(testset - mean(estset)))
}
identical(VfoldCV(Y = Y, criterion = testCriterion, output = "detailed"),
          VfoldCV(Y = Y, output = "detailed"))
```

Index