

Package ‘binsegRcpp’

October 12, 2022

Type Package

Title Efficient Implementation of Binary Segmentation

Version 2022.7.21

Author Toby Dylan Hocking

Maintainer Toby Dylan Hocking <toby.hocking@r-project.org>

Description Standard template library
containers are used to implement an efficient binary segmentation
algorithm, which is log-linear on average and quadratic in the
worst case.

License GPL-3

LinkingTo Rcpp

URL <https://github.com/tdhock/binsegRcpp>

BugReports <https://github.com/tdhock/binsegRcpp/issues>

Imports data.table, Rcpp

Suggests covr, penaltyLearning, directlabels, ggplot2, testthat,
knitr, markdown, neuroblastoma, changepoint, quadprog

VignetteBuilder knitr

RoxygenNote 7.1.2

NeedsCompilation yes

Repository CRAN

Date/Publication 2022-08-07 06:10:10 UTC

R topics documented:

binseg	2
binseg_interface	6
binseg_normal	7
binseg_normal_cv	9
case.colors	11
case.sizes	11

check_sizes	11
coef.binsegRcpp	12
coef.binseg_normal_cv	12
cum_median	13
cum_median_interface	14
depth_first_interface	14
get_complexity	15
get_complexity_best_heuristic_equal_breadth_full	17
get_complexity_best_heuristic_equal_depth_full	18
get_complexity_best_optimal_cost	19
get_complexity_best_optimal_splits	20
get_complexity_best_optimal_tree	20
get_complexity_empirical	21
get_complexity_extreme	22
get_complexity_worst	22
get_distribution_info	23
get_tree_empirical	23
plot.binsegRcpp	24
plot.binseg_normal_cv	24
plot.complexity	25
print.binsegRcpp	25
print.binseg_normal_cv	26
qp.x	26
random_set_vec	27
size_to_splits	30
tree_layout	31

Index 33

binseg	<i>Binary segmentation</i>
--------	----------------------------

Description

Efficient C++ implementation of the classic binary segmentation algorithm for finding changepoints in a sequence of N data. Output includes columns which can be used to compute parameters for a single model in log-linear time, using coef method.

Usage

```
binseg(distribution.str,
      data.vec, max.segments = NULL,
      is.validation.vec = rep(FALSE,
                                length(data.vec)),
      position.vec = seq_along(data.vec),
      weight.vec = rep(1,
                       length(data.vec)),
      min.segment.length = NULL,
      container.str = "multiset")
```

Arguments

distribution.str	String indicating distribution, use <code>get_distribution_info</code> to see possible values.
data.vec	Vector of numeric data to segment.
max.segments	Maximum number of segments to compute, default=NULL which means to compute the largest number possible, given <code>is.validation.vec</code> and <code>min.segment.length</code> . Note that the returned number of segments may be less than this, if there are min segment length constraints.
is.validation.vec	logical vector indicating which data are to be used in validation set, default=all FALSE (no validation set).
position.vec	integer vector of positions at which data are measured, default=1:length(data.vec).
weight.vec	Numeric vector of non-negative weights for each data point.
min.segment.length	Positive integer, minimum number of data points per segment. Default NULL means to use min given <code>distribution.str</code> .
container.str	C++ container to use for storing breakpoints/cost. Most users should leave this at the default "multiset" for efficiency but you could use "list" if you want to study the time complexity of a slower implementation of binary segmentation.

Details

Each iteration involves first computing and storing the best split point on one or two segments, then looking up the segment with the best split so far. The best case time complexity occurs when splits are equal (N data split into two segments of size N/2), and the worst case is when splits are unequal (N data split into one big segment with N-1 data and one small segment with 1 data point). Looking up the segment with the best split so far is a constant $O(1)$ time operation using C++ `multimap`, so $O(K)$ overall for K iterations/segments. Storage of a new best split point/cost involves the `multimap` insert method which is logarithmic time in the size of the `multimap`, overall $O(K \log K)$ for equal splits and $O(K)$ for unequal splits. Computing the cost values, and overall time complexity, depends on the loss. For normal and poisson distributions the best case $O(N \log K)$ time for equal splits and worst case $O(N K)$ time for unequal splits. For l1/laplace distributions the best case is $O(N \log N)$ time for equal splits and worst case is $O(N \log N K)$ time for unequal splits.

Value

list of class `binsegRcpp` with elements `min.segment.length`, `distribution.str`, `param.names`, `subtrain.borders` and `splits`, which is a `data.table` with columns:

segments	number of segments
loss	subtrain loss
validation.loss	validation loss
end	index of last data point per segment
depth	number of splits to reach segment

before params before changepoint
 after params after changepoint
 before.size number of data before changepoint
 after.size number of data after changepoint
 invalidates.index
 index of param invalidated by this split.
 invalidates.after
 indicates if before/after params invalidated by this split.

Author(s)

Toby Dylan Hocking

Examples

```

x <- c(0.1, 0, 1, 1.1, 0.1, 0)
## Compute full path of binary segmentation models from 1 to 6
## segments.
(models <- binsegRcpp::binseg("mean_norm", x))

## Plot loss values using base graphics.
plot(models)

## Same loss values using ggplot2.
if(require("ggplot2")){
  ggplot()+
    geom_point(aes(
      segments, loss),
      data=models$plits)
}

## Compute data table of segments to plot.
(segs.dt <- coef(models, 2:4))

## Plot data, segments, changepoints.
if(require("ggplot2")){
  ggplot()+
    theme_bw()+
    theme(panel.spacing=grid::unit(0, "lines"))+
    facet_grid(segments ~ ., labeller=label_both)+
    geom_vline(aes(
      xintercept=start.pos),
      color="green",
      data=segs.dt[1<start])+
    geom_segment(aes(
      start.pos, mean,
      xend=end.pos, yend=mean),
      data=segs.dt,
      color="green")+
    xlab("Position/index")+

```

```

      ylab("Data/mean value")+
      geom_point(aes(
        pos, x),
        data=data.frame(x, pos=seq_along(x)))
    }

## Use min.segment.length to constrain segment sizes.
(constrained.models <- binsegRcpp::binseg("mean_norm", x, min.segment.length = 2L))

## Demonstration of model selection using cross-validation in
## simulated data.
seg.mean.vec <- 1:5
data.mean.vec <- rep(seg.mean.vec, each=20)
set.seed(1)
n.data <- length(data.mean.vec)
data.vec <- rnorm(n.data, data.mean.vec, 0.2)
plot(data.vec)

library(data.table)
loss.dt <- data.table(seed=1:10)[, {
  set.seed(seed)
  is.valid <- sample(rep(c(TRUE,FALSE), l=n.data))
  bs.model <- binsegRcpp::binseg("mean_norm", data.vec, is.validation.vec=is.valid)
  bs.model$plits[, data.table(
    segments,
    validation.loss)]
}, by=seed]
loss.stats <- loss.dt[, .(
  mean.valid.loss=mean(validation.loss)
), by=segments]
plot(
  mean.valid.loss ~ segments, loss.stats,
  col=ifelse(
    mean.valid.loss==min(mean.valid.loss),
    "black",
    "red"))

selected.segments <- loss.stats[which.min(mean.valid.loss), segments]
full.model <- binsegRcpp::binseg("mean_norm", data.vec, selected.segments)
(segs.dt <- coef(full.model, selected.segments))
if(require("ggplot2")){
  ggplot()+
    theme_bw()+
    theme(panel.spacing=grid::unit(0, "lines"))+
    geom_vline(aes(
      xintercept=start.pos),
      color="green",
      data=segs.dt[1<start])+
    geom_segment(aes(
      start.pos, mean,
      xend=end.pos, yend=mean),
      data=segs.dt,
      color="green")+

```

```
xlab("Position/index")+
ylab("Data/mean value")+
geom_point(aes(
  pos, data.vec),
  data=data.frame(data.vec, pos=seq_along(data.vec)))
}

## Demo of poisson loss, weights.
data.vec <- c(3,4,10,20)
(fit1 <- binsegRcpp::binseg("poisson", data.vec, weight.vec=c(1,1,1,10)))
coef(fit1, 2L)
(fit2 <- binsegRcpp::binseg("poisson", data.vec, weight.vec=c(1,1,10,1)))
coef(fit2, 2L)
```

binseg_interface *binseg interface*

Description

Low-level interface to binary segmentation algorithm.

Usage

```
binseg_interface(data_vec,
  weight_vec, max_segments,
  min_segment_length,
  distribution_str,
  container_str, is_validation_vec,
  position_vec)
```

Arguments

data_vec
weight_vec
max_segments
min_segment_length

distribution_str

container_str
is_validation_vec

position_vec

Author(s)

Toby Dylan Hocking

binseg_normal	<i>Binary segmentation, normal change in mean</i>
---------------	---

Description

Calls `binseg` to compute a binary segmentation model for change in mean with constant variance, max normal likelihood = min square loss.

Usage

```
binseg_normal(data.vec,  
              max.segments = sum(!is.validation.vec),  
              is.validation.vec = rep(FALSE,  
                                       length(data.vec)),  
              position.vec = seq_along(data.vec))
```

Arguments

<code>data.vec</code>	Vector of numeric data to segment.
<code>max.segments</code>	Maximum number of segments to compute, default=number of FALSE entries in <code>is.validation.vec</code> .
<code>is.validation.vec</code>	logical vector indicating which data are to be used in validation set, default=all FALSE (no validation set).
<code>position.vec</code>	integer vector of positions at which data are measured, default= <code>1:length(data.vec)</code> .

Value

List output from `binseg` which represents a binary segmentation model.

Author(s)

Toby Dylan Hocking

Examples

```
x <- c(0.1, 0, 1, 1.1, 0.1, 0)  
## Compute full path of binary segmentation models from 1 to 6  
## segments.  
(models <- binsegRcpp::binseg_normal(x))  
  
## Plot loss values using base graphics.  
plot(models)  
  
## Same loss values using ggplot2.  
if(require("ggplot2")){  
  ggplot()+
```

```

    geom_point(aes(
      segments, loss),
      data=models$splits)
  }

  ## Compute data table of segments to plot.
  (segs.dt <- coef(models, 2:4))

  ## Plot data, segments, changepoints.
  if(require("ggplot2")){
    ggplot()+
      theme_bw()+
      theme(panel.spacing=grid::unit(0, "lines"))+
      facet_grid(segments ~ ., labeller=label_both)+
      geom_vline(aes(
        xintercept=start.pos),
        color="green",
        data=segs.dt[1<start])+
      geom_segment(aes(
        start.pos, mean,
        xend=end.pos, yend=mean),
        data=segs.dt,
        color="green")+
      xlab("Position/index")+
      ylab("Data/mean value")+
      geom_point(aes(
        pos, x),
        data=data.frame(x, pos=seq_along(x)))
  }

  ## Demonstration of model selection using cross-validation in
  ## simulated data.
  seg.mean.vec <- 1:5
  data.mean.vec <- rep(seg.mean.vec, each=20)
  set.seed(1)
  n.data <- length(data.mean.vec)
  data.vec <- rnorm(n.data, data.mean.vec, 0.2)
  plot(data.vec)

  library(data.table)
  loss.dt <- data.table(seed=1:10)[, {
    set.seed(seed)
    is.valid <- sample(rep(c(TRUE,FALSE), 1=n.data))
    bs.model <- binsegRcpp::binseg_normal(data.vec, is.validation.vec=is.valid)
    bs.model$splits[, data.table(
      segments,
      validation.loss)]
  }, by=seed]
  loss.stats <- loss.dt[, .(
    mean.valid.loss=mean(validation.loss)
  ), by=segments]
  plot(
    mean.valid.loss ~ segments, loss.stats,

```



```

col=ifelse(
  mean.valid.loss==min(mean.valid.loss),
  "black",
  "red"))

selected.segments <- loss.stats[which.min(mean.valid.loss), segments]
full.model <- binsegRcpp::binseg_normal(data.vec, selected.segments)
(segs.dt <- coef(full.model, selected.segments))
if(require("ggplot2")){
  ggplot()+
    theme_bw()+
    theme(panel.spacing=grid::unit(0, "lines"))+
    geom_vline(aes(
      xintercept=start.pos),
      color="green",
      data=segs.dt[1<start])+
    geom_segment(aes(
      start.pos, mean,
      xend=end.pos, yend=mean),
      data=segs.dt,
      color="green")+
    xlab("Position/index")+
    ylab("Data/mean value")+
    geom_point(aes(
      pos, data.vec),
      data=data.frame(data.vec, pos=seq_along(data.vec)))
}

```

binseg_normal_cv	<i>Binary segmentation, normal change in mean, cross-validation for model selection</i>
------------------	---

Description

Efficient implementation of binary segmentation for change in mean, with automatic model selection via cross-validation.

Usage

```

binseg_normal_cv(data.vec,
  max.segments = length(data.vec),
  position.vec = seq_along(data.vec),
  n.validation.sets = 100L,
  prop.validation = 0.5)

```

Arguments

data.vec Vector of numeric data to segment.

max.segments Maximum number of segments to compute, default=length(data.vec).
 position.vec integer vector of positions at which data are measured, default=1:length(data.vec).
 n.validation.sets
 Number of validation sets.
 prop.validation
 Proportion of validation set.

Author(s)

Toby Dylan Hocking

Examples

```

seg.mean.vec <- 1:5
data.mean.vec <- rep(seg.mean.vec, each=20)
set.seed(1)
n.data <- length(data.mean.vec)
data.vec <- rnorm(n.data, data.mean.vec, 0.2)
plot(data.vec)
(fit <- binsegRcpp::binseg_normal_cv(data.vec))
seg.dt <- coef(fit)
model.color <- "red"
seg.dt[, segments(start.pos, mean, end.pos, mean, col=model.color)]
seg.dt[start>1, abline(v=start.pos, col=model.color)]

## plot method shows number of times selected.
plot(fit)

if(requireNamespace("neuroblastoma")){
  data(neuroblastoma, package="neuroblastoma", envir=environment())
  library(data.table)
  profiles.dt <- data.table(neuroblastoma$profiles)
  one.chrom <- profiles.dt[profile.id=="4" & chromosome=="2"]
  fit <- one.chrom[, binsegRcpp::binseg_normal_cv(
    logratio, position.vec=position)]
  selected.segs <- coef(fit)
  if(require(ggplot2)){
    ggplot()+
      geom_point(aes(
        position, logratio),
        data=one.chrom)+
      geom_segment(aes(
        start.pos, mean,
        xend=end.pos, yend=mean),
        data=selected.segs,
        color=model.color)+
      geom_vline(aes(
        xintercept=start.pos,
        data=selected.segs[start>1],
        color=model.color)
    )
  }
}

```

```
}
```

case.colors	<i>case colors</i>
-------------	--------------------

Description

Character vector giving default colors for cases, ordered from worst to best.

Usage

```
"case.colors"
```

case.sizes	<i>case sizes</i>
------------	-------------------

Description

Numeric vector giving default sizes for cases.

Usage

```
"case.sizes"
```

check_sizes	<i>check sizes</i>
-------------	--------------------

Description

Checks types and values of size inputs.

Usage

```
check_sizes(N.data, min.segment.length,  
            n.segments)
```

Arguments

N.data
min.segment.length

n.segments

Author(s)

Toby Dylan Hocking

coef.binsegRcpp	<i>coef binsegRcpp</i>
-----------------	------------------------

Description

Compute a data table of segment start/end/mean values for all models given by segments.

Usage

```
## S3 method for class 'binsegRcpp'
coef(object,
      segments = 1:min(nrow(object$splits),
                       10), ...)
```

Arguments

object	data.table from binseg .
segments	integer vector, model sizes in number of segments.
...	ignored.

Value

data.table with one row for each segment.

Author(s)

Toby Dylan Hocking

coef.binseg_normal_cv	<i>coef binseg normal cv</i>
-----------------------	------------------------------

Description

Compute a data table of segment start/end/mean values for all models given by segments.

Usage

```
## S3 method for class 'binseg_normal_cv'
coef(object,
      segments = max(nrow(object$splits)),
      ...)
```

Arguments

object	data.table from binseg_normal_cv .
segments	integer vector, model sizes in number of segments. default=number of selected segments.
...	ignored.

Value

data.table with one row for each segment.

Author(s)

Toby Dylan Hocking

cum_median	<i>cum median</i>
------------	-------------------

Description

Efficient log-linear cumulative median.

Usage

```
cum_median(data.vec,  
            weight.vec = rep(1,  
                             length(data.vec)))
```

Arguments

data.vec	Numeric vector of data.
weight.vec	Numeric vector of weights.

Author(s)

Toby Dylan Hocking

cum_median_interface *cum median interface*

Description

Efficient log-linear cumulative median.

Usage

```
cum_median_interface(data_vec,  
                      weight_vec)
```

Arguments

data_vec
weight_vec

Author(s)

Toby Dylan Hocking

depth_first_interface *depth first interface*

Description

Use depth first search to compute a data.frame with one row for each segment, and columns splits and depth, number/depth of candidate splits that need to be computed after splitting that segment.

Usage

```
depth_first_interface(n_data,  
                      min_segment_length)
```

Arguments

n_data
min_segment_length

Author(s)

Toby Dylan Hocking

get_complexity	<i>get_complexity</i>
----------------	-----------------------

Description

Get empirical and extreme split counts, in order to compare the empirical and theoretical time complexity of the binary segmentation algorithm.

Usage

```
get_complexity(models,  
  y.increment = 0.1)
```

Arguments

models	result of binseg .
y.increment	Offset for y column values of totals output table.

Value

List of class "complexity" which has a plot method. Elements include "iterations" which is a data table with one row per model size, and column splits with number of splits to check after computing that model size; "totals" which is a data table with total number of splits for each case.

Author(s)

Toby Dylan Hocking

Examples

```
## Example 1: empirical=worst case.  
data.vec <- rep(0:1, l=8)  
plot(data.vec)  
worst.model <- binsegRcpp::binseg_normal(data.vec)  
worst.counts <- binsegRcpp::get_complexity(worst.model)  
plot(worst.counts)  
  
## Example 2: empirical=best case for full path.  
data.vec <- 1:8  
plot(data.vec)  
full.model <- binsegRcpp::binseg_normal(data.vec)  
full.counts <- binsegRcpp::get_complexity(full.model)  
plot(full.counts)  
  
## Example 3: empirical=best case for all partial paths.  
data.vec <- c(0,3,6,10,21,22,23,24)  
plot(data.vec)  
best.model <- binsegRcpp::binseg_normal(data.vec)
```

```

best.counts <- binsegRcpp::get_complexity(best.model)
plot(best.counts)

## ggplot comparing examples 1-3.
if(require("ggplot2")){
  library(data.table)
  splits.list <- list()
  for(data.type in names(worst.counts)){
    splits.list[[data.type]] <- rbind(
      data.table(data="worst", worst.counts[[data.type]]),
      data.table(data="best always", best.counts[[data.type]]),
      data.table(data="best full", full.counts[[data.type]])
    )
  }
  ggplot()+
    facet_grid(data ~ .)+
    geom_line(aes(
      segments, cum.splits, color=case, size=case),
      data=splits.list$iterations[case!="empirical"])+
    geom_point(aes(
      segments, cum.splits, color=case),
      data=splits.list$iterations[case=="empirical"])+
    scale_color_manual(
      values=binsegRcpp::case.colors,
      breaks=names(binsegRcpp::case.colors))+
    scale_size_manual(
      values=binsegRcpp::case.sizes,
      guide="none")
}

## Example 4: empirical case between best/worst.
data.vec <- rep(c(0,1,10,11),8)
plot(data.vec)
m.model <- binsegRcpp::binseg_normal(data.vec)
m.splits <- binsegRcpp::get_complexity(m.model)
plot(m.splits)

## Example 5: worst case for normal change in mean and variance
## model.
mv.model <- binsegRcpp::binseg("meanvar_norm", data.vec)
mv.splits <- binsegRcpp::get_complexity(mv.model)
plot(mv.splits)

## Compare examples 4-5 using ggplot2.
if(require("ggplot2")){
  library(data.table)
  splits.list <- list()
  for(data.type in names(m.splits)){
    splits.list[[data.type]] <- rbind(
      data.table(model="mean and variance", mv.splits[[data.type]]),
      data.table(model="mean only", m.splits[[data.type]])
    )
  }
  ggplot()+
    facet_grid(model ~ .)+

```



```

    geom_line(aes(
      segments, splits, color=case, size=case),
      data=splits.list$iterations[case!="empirical"])+
    geom_point(aes(
      segments, splits, color=case),
      data=splits.list$iterations[case=="empirical"])+
    geom_text(aes(
      x, y,
      label=label,
      color=case,
      hjust=1,
      data=splits.list$totals)+
    scale_color_manual(
      values=binsegRcpp::case.colors,
      guide="none")+
    scale_size_manual(
      values=binsegRcpp::case.sizes,
      guide="none")
  }

## Compare cumsums.
if(require("ggplot2")){
  library(data.table)
  splits.list <- list()
  for(data.type in names(m.splits)){
    splits.list[[data.type]] <- rbind(
      data.table(model="mean and variance", mv.splits[[data.type]]),
      data.table(model="mean only", m.splits[[data.type]]))
  }
  ggplot()+
  facet_grid(model ~ .)+
  geom_line(aes(
    segments, cum.splits, color=case, size=case),
    data=splits.list$iterations[case!="empirical"])+
  geom_point(aes(
    segments, cum.splits, color=case),
    data=splits.list$iterations[case=="empirical"])+
  scale_color_manual(
    values=binsegRcpp::case.colors,
    breaks=names(binsegRcpp::case.colors))+
  scale_size_manual(
    values=binsegRcpp::case.sizes,
    guide="none")
}

```

get_complexity_best_heuristic_equal_breadth_full

get complexity best heuristic equal breadth full

Description

Compute a fast approximate best case based on equal size splits.

Usage

```
get_complexity_best_heuristic_equal_breadth_full(N.data,  
min.segment.length)
```

Arguments

N.data
min.segment.length

Author(s)

Toby Dylan Hocking

```
get_complexity_best_heuristic_equal_depth_full  
get complexity best heuristic equal depth full
```

Description

Heuristic depth first.

Usage

```
get_complexity_best_heuristic_equal_depth_full(N.data,  
min.segment.length)
```

Arguments

N.data
min.segment.length

Author(s)

Toby Dylan Hocking

```
get_complexity_best_optimal_cost  
    get complexity best optimal cost
```

Description

Dynamic programming for computing lower bound on number of split candidates to compute / best case of binary segmentation. The dynamic programming recursion is on $f(d,s)$ = best number of splits for segment of size s which is split d times. Need to optimize $f(d,s) = g(s) + \min f(d1,s1) + f(d2,s2)$ over $s1,d1$ given that $s1+s2=s$, $d1+d2+1=d$, and $g(s)$ is the number of splits for segment of size s .

Usage

```
get_complexity_best_optimal_cost(N.data,  
    min.segment.length = 1L,  
    n.segments = NULL)
```

Arguments

`N.data` positive integer number of data.
`min.segment.length`
 positive integer min segment length.
`n.segments` positive integer number of segments.

Value

data table with one row for each $f(d,s)$ value computed.

Author(s)

Toby Dylan Hocking

Examples

```
binsegRcpp::get_complexity_best_optimal_cost(  
  N.data = 19L,  
  min.segment.length = 3L,  
  n.segments = 4L)
```

`get_complexity_best_optimal_splits`
get complexity best optimal splits

Description

Convert output of `get_complexity_best_optimal_tree` to counts of candidate splits that need to be considered at each iteration.

Usage

```
get_complexity_best_optimal_splits(node.dt,  
  min.segment.length)
```

Arguments

`node.dt`
`min.segment.length`

Value

Data table with one row for each segment.

Author(s)

Toby Dylan Hocking

`get_complexity_best_optimal_tree`
get complexity best optimal tree

Description

decoding.

Usage

```
get_complexity_best_optimal_tree(f.dt)
```

Arguments

`f.dt`

Value

Data table with one row for each node in the tree.

Author(s)

Toby Dylan Hocking

Examples

```
N.data <- 19L
min.seg.len <- 3L
max.segments <- 4L
cost.dt <- binsegRcpp::get_complexity_best_optimal_cost(
  N.data, min.seg.len, max.segments)
binsegRcpp::get_complexity_best_optimal_tree(cost.dt)
```

```
get_complexity_empirical
      get complexity empirical
```

Description

Get empirical split counts. This is a sub-routine of [get_complexity](#), which should typically be used instead.

Usage

```
get_complexity_empirical(model.dt,
  min.segment.length = 1L)
```

Arguments

`model.dt` splits data table from [binseg](#) result list.
`min.segment.length` Minimum segment length, positive integer.

Value

data.table with one row per model size, and column splits with number of splits to check after computing that model size.

Author(s)

Toby Dylan Hocking

```
get_complexity_extreme
    get complexity extreme
```

Description

Compute best and worst case number of splits.

Usage

```
get_complexity_extreme(N.data,
    min.segment.length = 1L,
    n.segments = NULL)
```

Arguments

N.data number of data to segment, positive integer.
min.segment.length minimum segment length, positive integer.
n.segments number of segments, positive integer.

Value

data.table with one row per model size, and column splits with number of splits to check after computing that model size. Column case has values best (equal segment sizes, min splits to check) and worst (unequal segment sizes, max splits to check).

Author(s)

Toby Dylan Hocking

```
get_complexity_worst    get complexity worst
```

Description

Get full sequence of splits which results in worst case time complexity.

Usage

```
get_complexity_worst(N.data,
    min.segment.length)
```

Arguments

N.data
min.segment.length

Author(s)

Toby Dylan Hocking

`get_distribution_info` *get distribution info*

Description

Compute a data.frame with one row for each distribution implemented in the C++ code, and columns distribution.str, parameters, description.

Usage

`get_distribution_info()`

Author(s)

Toby Dylan Hocking

`get_tree_empirical` *get tree empirical*

Description

Compute tree for empirical binary segmentation model.

Usage

`get_tree_empirical(fit)`

Arguments

fit

Author(s)

Toby Dylan Hocking

plot.binsegRcpp *plot binsegRcpp*

Description

Plot loss values from binary segmentation.

Usage

```
## S3 method for class 'binsegRcpp'  
plot(x, ...)
```

Arguments

x data.table from [binseg](#).
... ignored.

Author(s)

Toby Dylan Hocking

plot.binseg_normal_cv *plot binseg normal cv*

Description

Plot loss values from binary segmentation.

Usage

```
## S3 method for class 'binseg_normal_cv'  
plot(x,  
      ...)
```

Arguments

x data.table from [binseg_normal_cv](#).
... ignored.

Author(s)

Toby Dylan Hocking

plot.complexity	<i>plot complexity</i>
-----------------	------------------------

Description

Plot comparing empirical number of splits to best/worst case.

Usage

```
## S3 method for class 'complexity'  
plot(x, ...)
```

Arguments

x	data.table from get_complexity .
...	ignored.

Author(s)

Toby Dylan Hocking

print.binsegRcpp	<i>print binsegRcpp</i>
------------------	-------------------------

Description

Print method for binsegRcpp.

Usage

```
## S3 method for class 'binsegRcpp'  
print(x, ...)
```

Arguments

x	data.table from binseg .
...	ignored.

Author(s)

Toby Dylan Hocking

```
print.binseg_normal_cv  
      print binseg normal cv
```

Description

Print method for [binseg_normal_cv](#).

Usage

```
## S3 method for class 'binseg_normal_cv'  
print(x,  
      ...)
```

Arguments

x	data.table from binseg_normal_cv .
...	ignored.

Author(s)

Toby Dylan Hocking

qp.x	<i>qp.x</i>
------	-------------

Description

Solve quadratic program to find x positions.

Usage

```
qp.x(target, y.up, y.lo)
```

Arguments

target
y.up
y.lo

Author(s)

Toby Dylan Hocking

random_set_vec	<i>random set vec</i>
----------------	-----------------------

Description

Random set assignment.

Usage

```
random_set_vec(N, props.vec)
```

Arguments

`N` integer, size of output vector.
`props.vec` numeric vector of set proportions (must sum to one), with set names.

Value

Random vector of `N` set names.

Author(s)

Toby Dylan Hocking

Examples

```
library(data.table)
library(ggplot2)
library(binsegRcpp)
tvt.props <- c(test=0.19, train=0.67, validation=0.14)
tvt.N <- 1234567L
system.time({
  tvt.vec <- random_set_vec(tvt.N, tvt.props)
})
table(tvt.vec, useNA="ifany")/tvt.N

random_set_vec(6L, c(train=2/3, test=1/3))
random_set_vec(5L, c(train=2/3, test=1/3))
random_set_vec(4L, c(train=2/3, test=1/3))
random_set_vec(3L, c(train=2/3, test=1/3))

test.rev <- function(N, prop.vec, expected.vec){
  result <- list()
  for(fun.name in c("identity", "rev")){
    fun <- get(fun.name)
    ctab <- table(random_set_vec(N, fun(prop.vec)))
    result[[fun.name]] <- ctab
  }
}
```

```

    result$same <- sapply(
      result, function(tab) identical(as.numeric(tab), expected.vec))
    result
  }
test.rev(4L, c(test=1/3, train=2/3), c(1, 3))
table(random_set_vec(3L, c(test=0.5, train=0.5)))
table(random_set_vec(3L, c(train=0.5, test=0.5)))
test.rev(3L, c(test=0.4, train=0.6), c(1, 2))
test.rev(3L, c(test=0.49, train=0.51), c(1, 2))
test.rev(3L, c(test=0.6, train=0.4), c(2, 1))
## 2 is optimal after prob=2/3.
test.rev(2L, c(test=0.6, train=0.4), c(1, 1))
test.rev(2L, c(test=0.7, train=0.3), c(2))

## visualize the likelihood as a function of the proportion of
## success.
test.prop <- seq(0, 1, by=0.01)
prob.dt.list <- list()
n.total <- 2
for(n.test in 0:n.total){
  prob.dt.list[[paste(n.test)]] <- data.table(
    n.test,
    test.prop,
    prob=dbinom(n.test, n.total, test.prop))
}
prob.dt <- do.call(rbind, prob.dt.list)
thresh.dt <- data.table(thresh=(1:2)/3)
gg <- ggplot()+
  geom_vline(aes(xintercept=thresh), data=thresh.dt)+
  geom_line(aes(
    test.prop, prob, color=n.test, group=n.test),
    data=prob.dt)
if(requireNamespace("directlabels")){
  directlabels::direct.label(gg, "last.polygons")
}else{
  gg
}

## visualize the binomial likelihood as a function of number of
## successes, for a given probability of success.
n.total <- 43
n.success <- 0:n.total
p.success <- 0.6
lik.dt <- data.table(
  n.success,
  prob=dbinom(n.success, n.total, p.success))
ggplot()+
  geom_point(aes(
    n.success, prob),
    data=lik.dt)+
  geom_vline(xintercept=(n.total+1)*p.success)

## visualize the multinomial likelihood as a function of number of

```

```

## successes, for a given probability of success.
n.total <- 43
prob.vec <- c(train=0.6, validation=0.3, test=0.1)
train.dt <- data.table(train=0:n.total)
grid.dt <- train.dt[, data.table(
  validation=0:(n.total-train)), by=train]
grid.dt[, prob := dmultinom(
  c(train, validation, n.total-train-validation),
  n.total,
  prob.vec),
  by=(train, validation)]

train.bound <- (n.total+1)*prob.vec[["train"]]
validation.bound <- (n.total+1)*prob.vec[["validation"]]
guess.dt <- data.table(
  train=floor(train.bound),
  validation=floor(validation.bound))
max.dt <- grid.dt[which.max(prob)]#same
max.dt[, test := n.total-train-validation]

ggplot()+
  geom_tile(aes(
    train, validation, fill=prob),
    data=grid.dt)+
  scale_fill_gradient(low="white", high="red")+
  theme_bw()+
  geom_vline(
    xintercept=train.bound)+
  geom_hline(
    yintercept=validation.bound)+
  geom_point(aes(
    train, validation),
    shape=1,
    data=guess.dt)+
  coord_equal()

## visualize what happens when we start obs.seq variable above at 1
## or 0. starting at 0 is problematic e.g. 99% train/1% test with
## N=2 observations should return 2 train/0 test (and does when
## obs.seq starts with 1, but does NOT when obs.seq starts with 0).
random_set_vec(2L, c(train=0.99, test=0.01))
obs.dt.list <- list()
cum.dt.list <- list()
for(tvt.N in 2:4){
  obs.dt.list[[paste(tvt.N)]] <- data.table(tvt.N, rbind(
    data.table(start=0, obs=seq(0, tvt.N, l=tvt.N)),
    data.table(start=1, obs=seq(1, tvt.N, l=tvt.N))))
  not.round <- data.table(
    set=c("train", "test"),
    cum.thresh=tvt.N*c((tvt.N-2)/(tvt.N-1), 1))
  cum.dt.list[[paste(tvt.N)]] <- data.table(tvt.N, rbind(
    data.table(round=FALSE, not.round),
    not.round[, .(round=TRUE, set, cum.thresh=round(cum.thresh))]))
}

```

```
}
cum.dt <- do.call(rbind, cum.dt.list)
obs.dt <- do.call(rbind, obs.dt.list)
ggplot()+
  theme_bw()+
  theme(panel.spacing=grid::unit(0, "lines"))+
  facet_grid(tvt.N ~ .)+
  geom_point(aes(
    obs, start),
    data=obs.dt)+
  geom_vline(aes(
    xintercept=cum.thresh, color=round, linetype=round),
    data=cum.dt)
```

size_to_splits

size to splits

Description

Convert segment size to number of splits which must be computed during the optimization.

Usage

```
size_to_splits(size,
  min.segment.length)
```

Arguments

size Segment size, positive integer.
min.segment.length Minimum segment length, positive integer.

Value

Number of splits, integer.

Author(s)

Toby Dylan Hocking

tree_layout	<i>tree layout</i>
-------------	--------------------

Description

Compute x,y coordinates for graphing a tree.

Usage

```
tree_layout(node.dt,  
            space = 0.5)
```

Arguments

node.dt
space

Author(s)

Toby Dylan Hocking

Examples

```
N.data <- 29L  
min.seg.len <- 3L  
max.segments <- 5L  
cost.dt <- binsegRcpp::get_complexity_best_optimal_cost(  
  N.data, min.seg.len, max.segments)  
set.seed(1)  
data.vec <- rnorm(N.data)  
fit <- binsegRcpp::binseg_normal(data.vec, max.segments)  
tree.list <- list(  
  best=binsegRcpp::get_complexity_best_optimal_tree(cost.dt),  
  empirical=binsegRcpp::get_tree_empirical(fit))  
library(data.table)  
tree.dt <- data.table(type=names(tree.list))[, {  
  binsegRcpp::tree_layout(tree.list[[type]])  
}, by=type]  
total.dt <- tree.dt[, .(  
  candidate.splits=sum(binsegRcpp::size_to_splits(size, min.seg.len))  
), by=type]  
join.dt <- total.dt[tree.dt, on="type"]  
if(require(ggplot2)){  
  ggplot()+  
    facet_grid(. ~ type + candidate.splits, labeller=label_both)+  
    geom_segment(aes(  
      x, depth,  
      xend=parent.x, yend=parent.depth),
```

```
    data=join.dt)+  
  geom_label(aes(  
    x, depth, label=size),  
    data=join.dt)+  
  scale_y_reverse()  
}
```


Index

binseg, [2](#), [7](#), [12](#), [15](#), [21](#), [24](#), [25](#)
binseg_interface, [6](#)
binseg_normal, [7](#)
binseg_normal_cv, [9](#), [13](#), [24](#), [26](#)
binsegRcpp (binseg), [2](#)

case.colors, [11](#)
case.sizes, [11](#)
check_sizes, [11](#)
coef.binseg_normal_cv, [12](#)
coef.binsegRcpp, [12](#)
cum_median, [13](#)
cum_median_interface, [14](#)

depth_first_interface, [14](#)

get_complexity, [15](#), [21](#), [25](#)
get_complexity_best_heuristic_equal_breadth_full,
[17](#)
get_complexity_best_heuristic_equal_depth_full,
[18](#)
get_complexity_best_optimal_cost, [19](#)
get_complexity_best_optimal_splits, [20](#)
get_complexity_best_optimal_tree, [20](#),
[20](#)
get_complexity_empirical, [21](#)
get_complexity_extreme, [22](#)
get_complexity_worst, [22](#)
get_distribution_info, [3](#), [23](#)
get_tree_empirical, [23](#)

plot.binseg_normal_cv, [24](#)
plot.binsegRcpp, [24](#)
plot.complexity, [25](#)
print.binseg_normal_cv, [26](#)
print.binsegRcpp, [25](#)

qp.x, [26](#)

random_set_vec, [27](#)

size_to_splits, [30](#)
tree_layout, [31](#)