Package: segmenTier (via r-universe)

August 26, 2024

Type Package

Title Similarity-Based Segmentation of Multidimensional Signals

Version 0.1.2

Author Rainer Machne, Douglas B. Murray, Peter F. Stadler

URL https://github.com/raim/segmenTier

BugReports https://github.com/raim/segmenTier/issues

Maintainer Rainer Machne <raim@tbi.univie.ac.at>

Description A dynamic programming solution to segmentation based on maximization of arbitrary similarity measures within segments. The general idea, theory and this implementation are described in Machne, Murray & Stadler (2017) <doi:10.1038/s41598-017-12401-8>. In addition to the core algorithm, the package provides time-series processing and clustering functions as described in the publication. These are generally applicable where a `k-means` clustering yields meaningful results, and have been specifically developed for clustering of the Discrete Fourier Transform of periodic gene expression data (`circadian' or `yeast metabolic oscillations'). This clustering approach is outlined in the supplemental material of Machne & Murray (2012) <doi:10.1371/journal.pone.0037906>), and here is used as a basis of segment similarity measures. Notably, the time-series processing and clustering functions can also be used as stand-alone tools, independent of segmentation, e.g., for transcriptome data already mapped to genes.

License GPL (>= 2)

Imports Rcpp (>= 0.12.7)

Suggests flowMerge, flowClust, flowCore, knitr, rmarkdown

LinkingTo Rcpp

Encoding UTF-8

RoxygenNote 6.1.1

VignetteBuilder knitr

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

RemoteUrl https://github.com/raim/segmentier

RemoteRef HEAD

RemoteSha a639754d74898720da9b066324f490449c17a683

Contents

`ash` `asinh` *data transformation*

Description

The asinh transformation, (`ash(x) = log(x + sqrt(x^2+1))`), is an alternative to log transformation that has less (compressing) effects on the extreme values (low and high values), and naturally handles negative numbers and 0. Also see `log_1`.

Usage

```
ash(x)
```

Arguments

x	a numeric vector

`backtrace` *Back-tracing step of the* `segmenTier` *algorithm.*

Description

back-tracing step: collect clustered segments from the scoring matrix `S(i,c)` by back-tracing the position `j=k` which delivered the maximal score at position `i`.

Usage

```
backtrace(S, K, multib, nextmax = FALSE, verb = TRUE)
```

Arguments

S	matrix S, containing the local scores
K	matrix K, containing the position k used for score maximization
multib	if multiple k produce the maximal score, take either the shortest k ("max") or the longest k ("min"); if `multib` is set to "skip" the next unique k will be searched
nextmax	proceed backwards while score is increasing before opening a new segment
verb	print messages

`calculateScore` *segmenTier's core dynamic programming routine in Rcpp*

Description

segmenTier's core dynamic programming routine in Rcpp

Usage

```
calculateScore(seq, C, score, csim, M, Mn, multi = "max")
```

Arguments

seq	the cluster sequence (where clusters at positions k:i are considered). Note, that unlike the R wrapper, clustering numbers here are 0-based, where 0 is the nuisance cluster.
C	the list of clusters, including nuisance cluster '0', see seq
score	the scoring function to be used, one of "ccor" or "icor", an apt similarity matrix must be supplied via option csim
csim	a matrix, providing either the cluster-cluster (scoring function "ccor") or the position-cluster similarity function (scoring function "icor")
M	minimal sequence length; Note, that this is not a strict cut-off but defined as an accumulating penalty that must be "overcome" by good score
Mn	minimal sequence length for nuisance cluster, Mn<M will allow shorter distances between segments
multi	if multiple k are found which return the same maximal score, should the "max" (shorter segment) or "min" (longer segment) be used? This has little effect on real-life large data sets, since the situation will rarely occur. Default is "max".

Details

This is segmenTier's core dynamic programming routine. It constructs the total score matrix S(i,c), based on the passed scoring function ("icor" or "ccor"), and length penalty M. "Nuisance" cluster "0" can have a smaller penalty Mn to allow for shorter distances between "real" segments.

Scoring function "icor" calculates the sum of similarities of data at positions k:i to cluster centers c over all k and i. The similarities are calculated e.g., as a (Pearson) correlation between the data at individual positions and the tested cluster c center.

Scoring function "ccor" calculates the sum of similarities between the clusters at positions k:i to cluster c over all k and i.

Scoring function "ccls" is a special case of "ccor" and is NOT handled here, but is reflected in the cluster similarity matrix csim. It is handled and automatically constructed in the R wrapper segmentClusters, and merely counts the number of clusters in sequence k:i, over all k and i, that are identical to the tested cluster c, and sub-tracts a penalty for the count of non-identical clusters.

Value

Returns the total score matrix S(i,c) and the matrix K(i,c) which stores the position k which delivered the maximal score at position i. This is used in the back-tracing phase.

References

Machne, Murray & Stadler (2017) <doi:10.1038/s41598-017-12401-8>

clusterCor_c *Calculates position-cluster correlations for scoring function "icor".*

Description

Calculates Pearson's product-moment correlation coefficients between rows in `data` and `cluster`, and is used to calculate the position-cluster similarity matrix for the scoring function "icor". This is implemented in Rcpp for calculation speed, using `myPearson` to calculate correlations.

Usage

```
clusterCor_c(data, clusters)
```

Arguments

`data`	original data matrix
`clusters`	cluster centers

Value

Returns a position-cluster correlation matrix as used in scoring function "icor".

clusterTimeseries *Cluster a processed time-series with k-means.*

Description

Performs `kmeans` clustering of a time-series object `tset` provided by `processTimeseries`, and calculates cluster-cluster and cluster-position similarity matrices as required for `segmentClusters`.

Usage

```
clusterTimeseries(tset, K = 16, iter.max = 1e+05, nstart = 100,
  nui.thresh = -Inf, verb = 1)
```

Arguments

`tset`	a "timeseries" object returned by `processTimeseries`
`K`	the number of clusters to be calculated, ie. the argument `centers` of `kmeans`, but here multiple clusterings can be calculated, ie. `K` can be an integer vector. Note that a smaller cluster number is automatically chosen, if the data doesn't have more then K different values.
`iter.max`	the maximum number of iterations allowed in `kmeans`
`nstart`	number of randomized initializations of `kmeans`: "how many random sets should be chosen?"

nui.thresh	threshold correlation of a data point to a cluster center; if below the data point will be added to nuisance cluster 0
verb	level of verbosity, 0: no output, 1: progress messages

Details

This function performs one or more time-series clustering(s) using kmeans, and the output of processTimeseries as input. It further calculates cluster centers, cluster-cluster and cluster-position similarity matrices (Pearson correlation) that will be used by the main function of this package, segmentClusters, to split the cluster association sequence into segments, and assigns each segment to the "winning" input cluster.

The argument K is an integer vector that sets the requested cluster numbers (argument centers in kmeans). However, to avoid errors in batch use, a smaller K is chosen, if the data contains less then K distinct values.

Nuisance Cluster: values that were removed during time-series processing, such as rows that only contain 0 or NA values, will be assigned to the "nuisance cluster" with cluster label "0". Additionally, a minimal correlation to any cluster center can be specified, argument nui.thresh, and positions without any correlation higher then this, will also be assigned to the "nuisance" cluster. Resulting "nuisance segments" will not be shown in the results.

Cluster Sorting and Coloring: additionally the cluster labels in the result object will be sorted by cluster-cluster similarity (see sortClusters) and cluster colors assigned (see colorClusters) for convenient data inspection with the plot methods available for each data processing step (see examples).

Note that the function, in conjunction with processTimeseries, can also be used as a stand-alone tool for time-series clusterings, specifically implementing the strategy of clustering the Discrete Fourier Transform of periodic time-series developed by Machne & Murray (2012) <doi:10.1371/journal.pone.0037906>, and further analyzed in Lehmann et al. (2013) <doi:10.1186/1471-2105-14-133>, such as transcriptome data from circadian or yeast respiratory oscillation systems.

Value

Returns a list of class "clustering" comprising of a matrix of clusterings, lists of cluster centers, cluster-cluster and cluster-position similarity matrices (Pearson correlation) used by segmentClusters, and additional information such as a cluster sorting by similarity and cluster colors that allow to track clusters in plots. A plot method exists that allows to plot clusters aligned to "timeseries" and "segment" plots.

References

Machne & Murray (2012) <doi:10.1371/journal.pone.0037906>, and Lehmann et al. (2013) <doi:10.1186/1471-2105-14-133>

Examples

```
data(primseg436)
## Discrete Fourier Transform of the time-series,
## see ?processTimeseries for details
tset <- processTimeseries(ts=tsd, na2zero=TRUE, use.fft=TRUE,
```

```
                          dft.range=1:7,  dc.trafo="ash", use.snr=TRUE)
## ... and cluster the transformed time-series
cset <- clusterTimeseries(tset)
## plot methods for both returned objects allow aligned plots
par(mfcol=c(3,1))
plot(tset)
plot(cset)
```

`colorClusters` *Assign colors to clusters.*

Description

Takes a clustering set as returned by `clusterTimeseries` and assigns colors to each cluster in each clustering along the "hue" color wheel, as in `scale_colour_hue` in `ggplot2`. If `cset` contains a sorting, this sorting will be used to assign colors along the color wheel, otherwise a sorting will be calculated first, using `sortClusters`.

Usage

```
colorClusters(cset, colf, ...)
```

Arguments

`cset`	a clustering set as returned by `clusterTimeseries`
`colf`	a function that generates `n` colors
`...`	arguments to color function `colf`

Value

Returns the input "clustering" object with a list of vectors ("colors"), each providing a named vector of colors for each cluster.

`flowclusterTimeseries` *Cluster a processed time-series with* `flowClust` *&* `flowMerge`.

Description

A wrapper for `flowClust`, clustering a time-series object `tset` provided by `processTimeseries`, where specifically the DFT of a time-series and requested data transformation were calculated. This is intended to work in the same way as `clusterTimeseries` but was so far only tested for clustering of the final segment time-series, as previously applied to microarray data from yeast by Machne & Murray (2012) <doi:10.1371/journal.pone.0037906> and from cyanobacteria by Lehmann et al. (2013) <doi:10.1186/1471-2105-14-133>. It could in principle also be used for segmentation, but that has not been extensively tested. `flowClust` implements a model-based clustering approach and

is much slower then kmeans used in clusterTimeseries. Please see option ncpu on how to use parallel mode, which does not work on some installations. However, model-based clustering has the advantage of an intrinsic measure (BIC) to decide on the optimal cluster numbers. Additionally, the clusters can be "merged" to fewer clusters at constant BIC using flowMerge.

Usage

```
flowclusterTimeseries(tset, ncpu = 1, K = 10, selected,
  merge = FALSE, B = 500, tol = 1e-05, lambda = 1, nu = 4,
  nu.est = 0, trans = 1, ...)
```

Arguments

tset	processed time-series as provided by processTimeseries
ncpu	number of cores available for parallel mode of **flowClust**. NOTE: parallel mode of flowClust is often non-functional. Alternatively, you can set options(mc.cores=ncpu) directly.
K	the requested cluster numbers (vector of integers)
selected	a pre-selected cluster number which is then used as a start clustering for flowMerge (if option merge==TRUE)
merge	logical indicating whether cluster merging with flowMerge should be attempted
B	maximal number of EM iterations
tol	tolerance for EM convergence
lambda	initial Box-Cox trafo
nu	degrees of freedom used for the t distribution, Inf for pure Gaussian
nu.est	0: no, 1: non-specific, 2: cluster-specific estimation of nu
trans	0: no, 1: non-specific, 2: cluster-specific estim. of lambda
...	further parameters for flowClust

References

Machne & Murray (2012) <doi:10.1371/journal.pone.0037906>

---

logLik.kmeans *Experimental: AIC/BIC for kmeans*

Description

This function is supposed to provide a log-likelihood method for kmeans results, after Neal Fultz at https://stackoverflow.com/a/33202188 and also featured in the stackoverflow package. Note, that the blogged version on Jan 30, 2019 adds a minus and a division by 2 compared to a linked git version. This idea has not been reviewed, and this function has not been tested extensively; feel free to do so and contribute your results.

Usage

```
## S3 method for class 'kmeans'
logLik(object, ...)
```

Arguments

object	a kmeans result object
...	unused

Details

This is an attempt to reproduce the BIC measure in model-based clustering to decide on an optimal number of clusters. This function will be used for kmeans results objects when passed to BIC and AIC functions from the **stats** package in base R, and BIC and AIC are calculated this way in segmentClusters. It is however not used anywhere at the moment.

---

log_1 *log transformation handling zeros by adding 1*

Description

A conventional approach to handle 0 in log transformation is to simply add 1 to all data, log_1(x) = log(x+1). Also see ash.

Usage

```
log_1(x)
```

Arguments

x	a numeric vector

---

myPearson *Pearson product-moment correlation coefficient*

Description

Incremental calculation of the Pearson correlation coefficient between two vectors for calculation within Rcpp functions clusterCor_c.

Usage

```
myPearson(x, y)
```

Arguments

x	numeric vector
y	numeric vector

Details

Simply calculates Pearson's product-moment correlation between vectors x and y.

---

plot.clustering *Plot method for the "clustering" object.*

Description

plot the clustering object returned by clusterTimeseries

Usage

```
## S3 method for class 'clustering'
plot(x, k, sort = FALSE, xaxis, axes = 1:2,
  pch = 16, ylabh = TRUE, ...)
```

Arguments

x	a "clustering" object as returned by clusterTimeseries
k	a numeric or string vector indicating the clusterings to be plotted; specifically the column numbers or names in the matrix of clusterings in cset$clusters; if missing all columns will be plotted and the calling code must take care of properly assigning par(mfcol) or layout for the plot
sort	if TRUE and the clustering is yet unsorted a cluster sorting will be calculated based on "ccor" cluster-cluster similarity matrix x$Ccc; see sortClusters
xaxis	optionally x-values to use as x-axis (e.g. to reflect absolute chromosomal coordinates)
axes	list of axes to plot, numbers as used as first argument in function axis
pch	argument pch (symbol) for plot
ylabh	plot "clustering" horizontally at y-axis
...	additional arguments to plot, eg. to set point cex

Value

returns the input "clustering" object with (potentially new) cluster sorting and colors as in shown in the plot

plot.segments *Plot method for the "segments" object.*

Description

plot the final segmentation objects returned by segmentClusters and segmentCluster.batch

Usage

```
## S3 method for class 'segments'
plot(x, plot = c("S", "segments"), types, params,
  xaxis, show.fused = FALSE, ...)
```

Arguments

x	a segmentation object as returned by segmentClusters and segmentCluster.batch
plot	string vector indicating which data should be plotted; "segments": plot segments as arrows; "S1" plot the scoring vectors s(i,j,c for all c; "S" plot the derivative of matrix S(i,c) for all c
types	a string vector indicating segment types to plot (a subset of x$ids; defaults to all in x$ids)
params	a named vector of parameter settings used in segmentCluster.batch allows to filter plotted segment types, e.g. params=c(S="icor") will only plot segments where the scoring function (parameter S) "icor" was used.
xaxis	optional x-values to use as x-axis (e.g. to reflect absolute chromosomal coordinates)
show.fused	show the fuse tag as a black x
...	additional arguments forwarded to arrows, eg. to set lwd for for plot="segments", or to matplot for plot="S"

plot.timeseries *Plot method for the "timeseries" object.*

Description

plot the processed time-series object returned from processTimeseries.

Usage

```
## S3 method for class 'timeseries'
plot(x, plot = c("total", "timeseries"), xaxis,
  ylabh = TRUE, ...)
```

Arguments

x	a time-series object as returned by processTimeseries
plot	a string vector indicating the values to be plotted; "total": plot of the total signal, summed over the time-points, and indicating the applied threshold low.thresh; note that the total levels may have been transformed (e.g. by log_1 or ash) depending on the arguments trafo and dc.trafo in processTimeseries; "time-series": plot the complete time-series as a heatmap, where time is plotted bottom-up on the y-axis and segmentation coordinates on the x-axis;
xaxis	x-values to use as x-axis (e.g. to reflect absolute chromosomal coordinates)
ylabh	plot y-axis title horizontally
...	additional arguments to plot of total signal

---

plotdev *Switch between plot devices.*

Description

Switch between plot devices.

Usage

```
plotdev(file.name = "test", type = "png", width = 5, height = 5,
  res = 100)
```

Arguments

file.name	file name without suffix (.png, etc)
type	plot type: png, jpeg, eps, pdf, tiff or svg
width	figure width in inches
height	figure height in inches
res	resolution in ppi (pixels per inch), only for types png, jpeg and tiff

plotSegmentation *Summary plot for the* segmenTier *pipeline.*

Description

Plot all objects from the segmentation pipeline, i.e. the processed time-series, the clustering, the internal scoring matrices and the final segments.

Usage

```
plotSegmentation(tset, cset, sset, split = FALSE, plot.matrix = FALSE,
  mai = c(0.01, 2, 0.01, 0.01), ...)
```

Arguments

tset	a time-series object as returned by processTimeseries
cset	a clusterings object as returned by clusterTimeseries
sset	a segmentation object as returned by segmentClusters and segmentCluster.batch
split	split segment plots by clustering plots
plot.matrix	include the internal scoring matrices in the plot
mai	margins of individual plots, see par
...	further arguments to plot.clustering (cset) and plot.segments (sset). Note: these may conflict and cause errors, but eg. a combination of cex=0.5, lwd=3 works, affecting cluster point size and segment line width, respectively.

print.segments *Print method for segmentation result from* segmentClusters.

Description

Print method for segmentation result from segmentClusters.

Usage

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

Arguments

x	result object returned by function segmentClusters
...	further argument to print.data.frame

`processTimeseries` *Process a time-series for clustering and segmentation.*

Description

Prepares a time-series (time points in columns) for subsequent clustering, and performs requested data transformations, including a Discrete Fourier Transform (DFT) of the time-series, as direct input for the clustering wrapper `clusterTimeseries`. When used for segmentation the row order reflects the order of the data points along which segmentation will occur. The function can also be used as a stand-alone function equipped especially for analysis of oscillatory time-series, including calculation of phases and p-values for all DFT components, and can also be used for Fourier Analysis and subsequent clustering without segmentation.

Usage

```
processTimeseries(ts, na2zero = FALSE, trafo = "raw",
  use.fft = FALSE, dc.trafo = "raw", dft.range, perm = 0,
  use.snr = FALSE, lambda = 1, low.thresh = -Inf, smooth.space = 1,
  smooth.time = 1, circular.time = FALSE, verb = 0)
```

Arguments

`ts`	a time-series as a matrix, where columns are the time points and rows are ordered measurements, e.g., genomic positions for transcriptome data
`na2zero`	interpret NA values as 0
`trafo`	prior data transformation, pass any function name, e.g., "log", or the package functions "ash" (asinh: `ash(x) = log(x + sqrt(x^2+1))`) or "log_1" (`log(ts+1)`)
`use.fft`	use the Discrete Fourier Transform of the data
`dc.trafo`	data transformation for the first (DC) component of the DFT, pass any function name, e.g., "log", or the package functions "ash" (asinh: `ash(x) = log(x + sqrt(x^2+1))`) or "log_1" (`log(x+1)`).
`dft.range`	a vector of integers, giving the components of the Discrete Fourier Transform to be used where 1 is the first component (DC) corresponding to the total signal (sum over all time points), and 2:n are the higher components corresponding to 2:n full cycles in the data
`perm`	number of permutations of the data set, to obtain p-values for the oscillation
`use.snr`	use a scaled amplitude, where each component of the Discrete Fourier Transform is divided by the mean of all other components (without the first or DC component), a normalization that can be interpreted to reflect a signal-to-noise ratio (SNR)
`lambda`	parameter lambda for Box-Cox transformation of DFT amplitudes (experimental; not tested)
`low.thresh`	use this threshold to cut-off data, which will be added to the absent/nuisance cluster later

smooth.space	integer, if set a moving average is calculated for each time-point between adjacent data points using stats package's smooth with option span=smooth.space
smooth.time	integer, if set the time-series will be smoothed using stats package's filter to calculate a moving average with span smooth.time and smoothEnds to extrapolate smoothed first and last time-points (again using span smooth.time)
circular.time	logical value indicating whether time can be treated as circular in smoothing via option smooth.time
verb	level of verbosity, 0: no output, 1: progress messages

Details

This function exemplifies the processing of an oscillatory transcriptome time-series data as used in the establishment of this algorithm and the demo segment_data. As suggested by Machne & Murray (PLoS ONE 2012) and Lehmann et al. (BMC Bioinformatics 2014) a Discrete Fourier Transform of time-series data allows to cluster time-series by their change pattern.

Note that NA values are here interpreted as 0. Please take care of NA values yourself, if you do not want this behavior.

Rows consisting only of 0 (or NA) values, or with a total signal (sum over all time points) below the value passed in argument low.thresh, are detected, result in NA values in the transformed data, and will be assigned to the "nuisance" cluster in clusterTimeseries.

Discrete Fourier Transform (DFT): if requested (option use.fft=TRUE), a DFT will be applied using base R's mvfft function and reporting all or only requested (option dft.range) DFT components, where the first, or DC ("direct current") component, equals the total signal (sum over all points) and other components are numbered 1:n, reflecting the number of full cycles in the time-series. Values are reported as complex numbers, from which both amplitude and phase can be calculated. All returned DFT components will be used by clusterTimeseries.

Additional Transformations: data can be transformed prior to DFT (options trafo, smooth.time, smooth.space), or after DFT (options use.snr and dc.trafo). It is recommended to use the amplitude scaling (a signal-to-noise ratio transformation, see option documentation). The separate transformation of the DC component allows to de-emphasize the total signal in subsequent clustering & segmentation. Additionally, but not tested in the context of segmentation, a Box-Cox transformation of the DFT can be performed (option lambda). This transformation proofed useful in DFT-based clustering with the model-based clustering algorithm in package **flowClust**, and is available here for further tests with k-means clustering.

Phase, Amplitude and Permutation Analysis: this time-series processing and subsequent clustering can also be used without segmentation, eg. for conventional microarray data or RNA-seq data already mapped to genes. The option perm allows to perform a permutation test (perm times) and adds a matrix of empirical p-values for all DFT components to the results object, ie. the fraction of perm where amplitude was higher then the amplitude of the randomized time-series. Phases and amplitudes can be derived from the complex numbers in matrix "dft" of the result object.

Value

Returns a list of class "timeseries" which comprises of the transformed time-series and additional information, such as the total signal, and positions of rows with only NA/0 values. Note that NA values are interpreted as 0.

References

Machne & Murray (2012) <doi:10.1371/journal.pone.0037906>, and Lehmann et al. (2013) <doi:10.1186/1471-2105-14-133>

Examples

```
data(primseg436)
## The input data is a matrix with time points in columns
## and a 1D order, here 7624 genome positions, is reflected in rows,
## if the time-series should be segmented.
nrow(tsd)
## Time-series processing prepares the data for clustering,
## the example data is periodic, and we will cluster its Discrete Fourier
## Transform (DFT) rather then the original data. Specifically we will
## only use components 1 to 7 of the DFT (dft.range) and also apply
## a signal/noise ratio normalization, where each component is
## divided by the mean of all other components. To de-emphasize
## total levels the first component (DC for "direct current") of the
## DFT will be separately arcsinh transformed. This peculiar combination
## proofed best for our data:
tset <- processTimeseries(ts=tsd, na2zero=TRUE, use.fft=TRUE,
                          dft.range=1:7, dc.trafo="ash", use.snr=TRUE)
## a plot method exists for the returned time-series class:
par(mfcol=c(2,1))
plot(tset)
```

---

segmentCluster.batch *Batch wrapper for* segmentClusters.

Description

A high-level wrapper for multiple runs of segmentation by segmentClusters for multiple clusterings and/or multiple segmentation parameters. It additionally allows to tag adjacent segments to be potentially fused due to similarity of their clusters.

Usage

```
segmentCluster.batch(cset, varySettings = setVarySettings(),
  fuse.threshold = 0.2, rm.nui = TRUE, type.name, short.name = TRUE,
  id, save.matrix = FALSE, verb = 1)
```

Arguments

cset	a clustering set as returned by clusterTimeseries
varySettings	list of settings where each entry can be a vector; the function will construct a matrix of all possible combinations of parameter values in this list, call segmentClusters for each, and report a matrix of segments where the segment 'type' is constructed from the varied parameters; see option short.name. A varySettings list with all required (default) parameters can be obtained via function setVarySettings.

fuse.threshold	if adjacent segments are associated with clusters the centers of which have a Pearson correlation >fuse.threshold the field "fuse" will be set to 1 for the second segments (top-to-bottom as reported)
rm.nui	remove nuisance cluster segments from final results
type.name	vector of strings selecting the parameters which will be used as segment types. Note, that all parameters that are actually varied will be automatically added (if missing). The list can include parameters from time-series processing found in the "clustering" object cset as cset$tids.
short.name	default type name construction; if TRUE (default) parameters that are not varied will not be part of the segment type and ID. This argument has no effect if argument type.name is set.
id	if set, the default segment IDs, constructed from numbered segment types, are replaced by this
save.matrix	store the total score matrix S(i,c) and the backtracing matrix K(i,c); useful in testing stage or for debugging or illustration of the algorithm; TODO: save.matrix is currently not implemented, since batch function returns a matrix only
verb	level of verbosity, 0: no output, 1: progress messages

Details

This is a high-level wrapper for segmentClusters which allows segmentation over multiple clusterings as provided by the function clusterTimeseries and over multiple segmentation parameters. Each parameter in the list varySettings can be a vector and ALL combinations of the passed parameter values will be used for one run of segmentClusters. The resulting segment table, list item "segments" of the returned object, is a data.frame with additional columns "ID" and "type", automatically generated strings indicating the used parameters (each "type" reflects one parameter set), and "colors", indicating the automatically generated color of the assigned cluster label.

Value

Returns an object of class "segments", just as its base function segmentClusters, but the main segment table, list item "segments", is a data.frame with additional columns "ID" and "type", automatically generated strings indicating the used parameters (each "type" reflects one parameter set), and "colors", indicating the automatically generated color of the assigned cluster label.

Examples

```
# load example data, an RNA-seq time-series data from a short genomic
# region of budding yeast
data(primseg436)

# 1) Fourier-transform time series:
tset <- processTimeseries(ts=tsd, na2zero=TRUE, use.fft=TRUE,
                          dft.range=1:7, dc.trafo="ash", use.snr=TRUE)

# 2) cluster time-series several times into K=12 clusters:
cset <- clusterTimeseries(tset, K=c(12,12,12))
```

```
# 3) choose parameter ranges, here only E is varied
vary <- setVarySettings(M=100, E=c(1,3), nui=3, S="icor", Mn=20)

# 4) ... segment ALL using the batch function:
## Not run:  ## NOTE: takes too long for CRAN example timing restrictions
segments <- segmentCluster.batch(cset=cset, varySettings=vary)

# 5) inspect results:
print(segments)
plotSegmentation(tset, cset, segments)

# 6) and get segment border table. Note that the table has
#    additional columns "ID" and "type", indicating the used parameters,
#    and "color" providing the color of the cluster the segment was
#    assigned to. This allows to track segments in the inspection plots.
sgtable <- segments$segments

## End(Not run)
```

`segmentClusters` *Run the* `segmenTier` *algorithm.*

Description

segmenTier's main wrapper interface, calculates segments from a clustering sequence. This will run the segmentation algorithm once for the indicated parameters. The function `segmentCluster.batch` allows for multiple runs over different parameters or input-clusterings.

Usage

```
segmentClusters(seq, k = 1, csim, E = 1, S = "ccor", M = 175,
  Mn = 20, a = -2, nui = 1, nextmax = TRUE, multi = "max",
  multib = "max", rm.nui = TRUE, save.matrix = FALSE, verb = 1)
```

Arguments

`seq`	Either an integer vector of cluster labels, or a structure of class 'clustering' as returned by `clusterTimeseries`. The only strict requirement for the first option is that nuisance clusters (which will be treated specially during the dynamic programming routine) have to be '0' (zero).
`k`	if argument `seq` is of class 'clustering' the kth clustering will be used; defaults to 1
`csim`	The cluster-cluster or position-cluster similarity matrix for scoring functions "ccor" and "icor" (option `S`), respectively. If `seq` is of class 'clustering' `csim` is optional and will override the similarity matrices in `seq`. If argument `seq` is a simple vector of cluster labels and the scoring function is "icor" or "ccor",

	an appropriate matrix `csim` MUST be provided. Finally, for scoring function "ccls" the argument `csim` will be ignored and the matrix is instead automatically constructed from argument `a`, and using argument `nui` for the nuisance cluster.
`E`	exponent to scale similarity matrices
`S`	the scoring function to be used: "ccor", "icor" or "ccls"
`M`	segment length penalty. Note, that this is not a strict cut-off but defined as a penalty that must be "overcome" by good score.
`Mn`	segment length penalty for nuisance cluster. Mn<M will allow shorter distances between "real" segments; only used in scoring functions "ccor" and "icor"
`a`	a cluster "dissimilarity" only used for pure cluster-based scoring w/o cluster similarity measures in scoring function "ccls".
`nui`	the similarity score to be used for nuisance clusters in the cluster similarity matrices
`nextmax`	go backwards while score is increasing before opening a new segment, default is TRUE
`multi`	handling of multiple k with max. score in forward phase, either "min" (default) or "max"
`multib`	handling of multiple k with max. score in back-trace phase, either "min" (default), "max" or "skip"
`rm.nui`	remove nuisance cluster segments from final results
`save.matrix`	store the total score matrix `S(i,c)` and the backtracing matrix `K(i,c)`; useful in testing stage or for debugging or illustration of the algorithm;
`verb`	level of verbosity, 0: no output, 1: progress messages

Details

This is the main R wrapper function for the 'segmenTier' segmentation algorithm. It takes an ordered sequence of cluster labels and returns segments of consistent clusterings, where cluster-cluster or cluster-position similarities are maximal. Its main input (argument `seq`) is either a "clustering" object returned by `clusterTimeseries` (scenario I), or an integer vector of cluster labels (scenario II) or. The function then runs the dynamic programming algorithm (`calculateScore`) for a selected scoring function and an according cluster similarity matrix, followed by the back-tracing step (`backtrace`) to find segment borders.

The main result, list item "segments" of the returned object, is a 3-column matrix, where column 1 is the cluster assignment and columns 2 and 3 are start and end indices of the segments. For the batch function `segmentCluster.batch`, the "segments" item is a `data.frame` contain additional information, see ?segmentCluster.batch.

As shown in the publication, the parameters `M`, `E` and `nui` have the strongest impact on resulting segment borders. Other parameters can be fine-tuned but had little impact on our test data set.

In the default and tested scenario I, when the input is an object of class "clustering" produced by `clusterTimeseries`, the cluster-cluster and cluster-position similarity matrices are already provided by this object.

In the second scenario II for custom use, argument `seq` can be a simple clustering vector, where a nuisance cluster must be indicated by cluster label "0" (zero). The cluster-cluster or cluster-position

similarities MUST be provided (argument csim) for scoring functions "ccor" and "icor", respectively. For the simplest scoring function "ccls", a uniform cluster similarity matrix is constructed from arguments a and nui, with cluster self-similarities of 1, "dissimilarities" between different clusters using argument a<0, and nuisance cluster self-similarity of -a.

The function returns a list (class "segments") comprising of the main result (list item "segments"), and "warnings" from the dynamic programming and backtracing phases, the used similarity matrix csim, extended by the nuisance cluster; and optionally (see option save.matrix) the scoring vectors S1(i,c), the total score matrix S(i,c) and the backtracing matrix K(i,c) for analysis of algorithm performance for novel data sets. Additional convenience data is reported, such as cluster colors and sortings if argument seq was of class 'clustering'. These allow for convenient inspection of all data processing steps with the plot methods. A plot method exists that allows to plot segments aligned to "timeseries" and "clustering" plots.

Value

Returns a list (class "segments") containing the main result (list item "segments"), and additional information (see 'Details'). A plot method exists that allows to plot clusters aligned to time-series and segmentation plots.

References

Machne, Murray & Stadler (2017) <doi:10.1038/s41598-017-12401-8>

Examples

```
# load example data, an RNA-seq time-series data from a short genomic region
# of budding yeast
data(primseg436)

# 1) Fourier-transform time series:
## NOTE: reducing official example data set to stay within
## CRAN example timing restrictions with segmentation below
tset <- processTimeseries(ts=tsd[2500:6500,], na2zero=TRUE, use.fft=TRUE,
                          dft.range=1:7, dc.trafo="ash", use.snr=TRUE)

# 2) cluster time-series into K=12 clusters:
cset <- clusterTimeseries(tset, K=12)

# 3) ... segment it; this takes a few seconds:
segments <- segmentClusters(seq=cset, M=100, E=2, nui=3, S="icor")

# 4) inspect results:
print(segments)
plotSegmentation(tset, cset, segments, cex=.5, lwd=3)

# 5) and get segment border table for further processing:
sgtable <- segments$segments
```

segmenTier *segmenTier : cluster-based segmentation from a sequential clustering*

Description

segmenTier : cluster-based segmentation from a sequential clustering

Dependencies

The package strictly depends only on Rcpp. All other dependencies are usually present in a basic installation (stats, graphics, grDevices)).

Author(s)

Rainer Machne <raim@tbi.univie.ac.at>, Douglas B. Murray, Peter F. Stadler <studla@bioinf.uni-leipzig.de>

References

Machne, Murray & Stadler (2017) <doi:10.1038/s41598-017-12401-8>, Machne & Murray (2012) <doi:10.1371/journal.pone.0037906>, and Lehmann et al. (2013) <doi:10.1186/1471-2105-14-133>

setVarySettings *Parameters for* segmentCluster.batch.

Description

Generates the parameter list (varySettings) for segmentCluster.batch, using defaults for all parameters not passed.

Usage

```
setVarySettings(E = c(1, 3), S = "ccor", M = 100, Mn = 100,
  a = -2, nui = c(1, 3), nextmax = TRUE, multi = "max",
  multib = "max")
```

Arguments

E	exponent to scale similarity matrices
S	the scoring function to be used: "ccor", "icor" or "ccls"
M	segment length penalty. Note, that this is not a strict cut-off but defined as a penalty that must be "overcome" by good score.
Mn	segment length penalty for nuisance cluster. Mn<M will allow shorter distances between "real" segments; only used in scoring functions "ccor" and "icor"

a	a cluster "dissimilarity" only used for pure cluster-based scoring w/o cluster similarity measures in scoring function "ccls".
nui	the similarity score to be used for nuisance clusters in the cluster similarity matrices
nextmax	go backwards while score is increasing before opening a new segment, default is TRUE
multi	handling of multiple k with max. score in forward phase, either "min" (default) or "max"
multib	handling of multiple k with max. score in back-trace phase, either "min" (default), "max" or "skip"

Value

Returns a parameter settings structure that can be used in the batch function segmentCluster.batch.

---

sortClusters *Sort clusters by similarity.*

Description

Takes a "clustering" object as returned by clusterTimeseries and uses the cluster-cluster similarity matrix, item Ccc, to sort clusters by their similarity, starting with the cluster labeled '1'; the next cluster is the first cluster (lowest cluster label) with the highest similarity to cluster '1', and proceeding from there. The final sorting is added as item "sorting" to the cset object and returned. This sorting is subsequently used to select cluster colors and in the plot method. This simply allows for more informative plots of the clustering underlying a segmentation but has no consequence on segmentation itself.

Usage

```
sortClusters(cset, sort = TRUE, verb = 0)
```

Arguments

cset	a clustering set as returned by clusterTimeseries
sort	if set to FALSE the clusters will be sorted merely numerically
verb	level of verbosity, 0: no output, 1: progress messages

Value

Returns the input "clustering" object with a list of vectors (named "sorting"), each providing a similarity-based sorting of cluster labels.

tsd *Transcriptome time-series from budding yeast.*

Description

Transcriptome time-series data from a region encompassing four genes and a regulatory upstream non-coding RNA in budding yeast. The data set is described in more detail in the publication Machne, Murray & Stadler (2017) <doi:10.1038/s41598-017-12401-8>.

Index