Package 'SRCS'

October 12, 2022

Title Statistical Ranking Color Scheme for Multiple Pairwise Comparisons

Description Implementation of the SRCS method for a color-based visualization of the results of multiple pairwise tests on a large number of problem configurations, proposed in: I.G. del Amo, D.A. Pelta. SRCS: a technique for comparing multiple algorithms under several factors in dynamic optimization problems. In: E. Alba, A. Nakib, P. Siarry (Eds.), Metaheuristics for Dynamic Optimization. Series: Studies in Computational Intelligence 433, Springer, Berlin/Heidelberg, 2012.

Version 1.1

Date 2015-06-30

Author Pablo J. Villacorta <pjvi@decsai.ugr.es>

Maintainer Pablo J. Villacorta <pjvi@decsai.ugr.es>

Imports parallel, graphics, stats, grDevices, methods

Suggests R.rsp

License LGPL (>= 3)

URL http://decsai.ugr.es/~pjvi/r-packages.html

LazyData true

VignetteBuilder R.rsp

NeedsCompilation no

Repository CRAN

Date/Publication 2015-07-02 06:47:17

R topics documented:

ML1 2
MPB 3
MPBall 4
plot.SRCS 5
SRCS 10
SRCScomparison 11
SRCSranks 12

Index 15

ML1 *Performance of 6 different supervised classification algorithms on eight noisy datasets (see references)*

Description

Dataset with the test accuracy of 6 supervised classification algorithms on eight noisy datasets. The way noise is introduced in originally clear datasets can be adjusted according to some parameters such as the noise type (attribute noise versus class noise) and the noise ratio.

Usage

```
data(ML1)
```

Format

A data frame with 52800 observations on the following 6 variables.

`Algorithm` A factor with 6 levels: `1-NN`, `3-NN`, `5-NN`, `C4.5`, `RIPPER`, `SVM` that correspond to 6 different supervised classification algorithms.

`Dataset` A factor with 8 levels: `autos`, `balanced`, `cleveland`, `ecoli`, `ionosphere`, `pima`, `vehicle` corresponding to the names of eight datasets in which noise has been introduced artificially.

`Noise type` A factor with 4 levels: `ATT_GAUS`, `ATT_RAND`, `CLA_PAIR`, `CLA_RAND` that correspond to the type of noise introduced: ATT_* to denote noise added to (a percentage of) the attributes of the instance (either in a gaussian or uniformly random way), and CLA_* to denote noise which modifies the class of (a percentage of) the instances of the dataset (either by any other class at random, as in CLA_RAND, or by replacing the label of only a percentage of the examples of the majority class by the label of the second-majority class as in CLA_PAIR).

`Noise ratio` A real number with the ratio of attributes affected by noise (for `ATT_GAUS` and `ATT_RAND`), or the ratio of examples within the global dataset affected by a class error (for `CLA_PAIR` and `CLA_RAND`).

`Fold` An integer number (between 1 and 25) associated with the repetition of the experiment. Recall that test results were obtained by repeating five independent times a complete 5-fold Cross Validation process.

`Performance` Real number between 0 and 1 with the accuracy (in percentage) of the classifier over the test examples.

Source

J.A. Saez, M.Galar, J.Luengo, F.Herrera, Tackling the Problem of Classification with Noisy Data using Multiple Classifier Systems: Analysis of the Performance and Robustness. Information Sciences, 247 (2013) 1-20.

References

Christopher M. Bishop. Pattern Recognition and Machine Learning. Springer (2006).

Examples

```
data(ML1)
str(ML1)
head(ML1)
```

MPB *Performance of 8 different dynamic optimization algorithms on the Moving Peaks Benchmark (see references)*

Description

Dataset with the performance of several dynamic optimization algorithms in the Moving Peaks Benchmark problem (see the source section). The MPB function can be configured according to some parameters such as the dimension, the change frequency and the severity of changes. The performance measure employed is the average offline error.

Usage

```
data(MPB)
```

Format

A data frame with 220000 observations on the following 5 variables.

`Algorithm` A factor with levels `reactive-cs` `independent-cs` `mqso-both` `mqso-rand` `mqso-change` `mqso agents` `soriga` that correspond to 8 different algorithms for dynamic optimization applied to the Moving Peaks Benchmark function.

`Dim` A numeric vector with the dimension (number of input variables) of the MPB function.

`CF` A numeric vector with the change frequency along the time, i.e. the number of evaluations of the fitness function after which a change of the location of the function maxima happens.

`Severity` A numeric vector with the severity of a change when it occurs.

`OffError` A numeric vector with the performance measure, in this case the offline error computed as the average of the offline errors just before every change.

Source

I.G. del Amo, D.A. Pelta. SRCS: a technique for comparing multiple algorithms under several factors in dynamic optimization problems, in: E. Alba, A. Nakib, P. Siarry (Eds.), Metaheuristics for Dynamic Optimization. Series: Studies in Computational Intelligence 433, Springer, Berlin/Heidelberg, 2012.

Examples

```
data(MPB)
str(MPB)
head(MPB)
```

MPBall — *Performance of 3 different dynamic optimization algorithms on the Moving Peaks Benchmark captured at five time moments of the execution (see references)*

Description

Dataset with the performance of several dynamic optimization algorithms in the Moving Peaks Benchmark problem (see the source section) at five time moments, just before a change. The MPB function can be configured according to some parameters such as the dimension, the change frequency and the severity of changes. The performance measure employed is the average offline error, averaged from the beginning up to each time moment. This dataset serves for illustrating how to compose a video sequence using function `animatedplot`.

Usage

```
data(MPBall)
```

Format

A data frame with 82500 observations on the following variables.

`Algorithm` A factor with levels `reactive-cs` `independent-cs` `mqso-both` `mqso-rand` `mqso-change` `mqso agents` `soriga` that correspond to 8 different algorithms for dynamic optimization applied to the Moving Peaks Benchmark function.

`Dim` A numeric vector with the dimension (number of input variables) of the MPB function.

`CF` A numeric vector with the change frequency along the time, i.e. the number of evaluations of the fitness function after which a change of the location of the function maxima happens.

`Severity` A numeric vector with the severity of a change when it occurs.

OffError_1,OffError_25,OffError_49,OffError_73,OffError_97 A numeric vector with the performance measure, in this case the offline error computed as the average (over the previous changes) of the offline errors just before every change. Each algorithm was allowed to run for 100 slices, but we have selected only 5 moments of that process, i.e. before the first change, the 25th change, the 49th, 73th and 97th change, in order to keep the resulting dataset reasonably small.

Source

I.G. del Amo, D.A. Pelta. SRCS: a technique for comparing multiple algorithms under several factors in dynamic optimization problems, in: E. Alba, A. Nakib, P. Siarry (Eds.), Metaheuristics for Dynamic Optimization. Series: Studies in Computational Intelligence 433, Springer, Berlin/Heidelberg, 2012.

Examples

```
data(MPBall)
str(MPBall)
head(MPBall)
```

plot.SRCS *Heatmap plot of the ranking achieved by a target variable levels after all statistical pairwise comparisons in multi-parameter problem instances.*

Description

plot.SRCS: Function to display a grid of heatmaps representing the statistical ranking of one level of the target factor (usually, the algorithm) vs the rest of levels of the target factor, over several problem configurations characterized by (at most) 3 parameters in addition to the target factor.

animatedplot: Function to generate an animated video consisting of a temporal sequence of grid plots like those generated by plot.SRCS. The function requires software ImageMagick has been installed.

singleplot: Function to display either a single heatmap representing the statistical ranking of one level of the target factor (usually, the algorithm) vs the rest of levels of the target factor, over one single problem configurations defined by a combination of values for the problem configuration parameters.

Usage

```
## S3 method for class 'SRCS'
plot(x, yOuter, xOuter, yInner, xInner, zInner = "rank",
  out.Y.par = list(), out.X.par = list(), inner.X.par = list(),
  inner.Y.par = list(), colorbar.par = list(),
  color.function = heat.colors, heatmaps.per.row = NULL,
  heatmaps.titles = NULL, show.colorbar = TRUE, annotation.lab = NULL,
  heat.cell.par = list(), heat.axes.par = list(),
  colorbar.cell.par = list(), colorbar.axes.par = list(),
  annotation.text.par = list(), ...)

animatedplot(x, filename, path.to.converter, yOuter, xOuter, yInner, xInner,
  zInner, width = 800, height = 800, res = 100, pointsize = 16,
  delay = 30, type = c("png", "jpeg", "bmp", "tiff"), quality = 75,
  compression = c("none", "rle", "lzw", "jpeg", "zip"), annotations = NULL,
  ...)

singleplot(x, yInner, xInner, zInner = "rank", color.function = heat.colors,
  labels.par = list(), colorbar.par = list(), heat.axes.par = list(),
  colorbar.axes.par = list(), haxis = TRUE, vaxis = TRUE, title = "",
  show.colorbar = TRUE, ...)
```

Arguments

x An SRCS object containing columns for the names of the problem parameters (including the algorithm) and the rank obtained by that algorithm when compared with the rest over the same problem configuration. Typically this is the object returned by a call to SRCSranks.

`yOuter,xOuter` Names of the variables in `x` that will be placed vertically (in the left-most part) and horizontally (on the top), respectively. Each level of `yOuter` (resp. `xOuter`) corresponds to a complete row (complete column) of heatmaps in the grid.

`yInner,xInner` Names of the variables in `x` that will be placed on the Y axis and on the X axis of every heatmap, respectively. Each level of `yInner` (resp. `xInner`) corresponds to a row (column) inside a heatmap.

`zInner` Name of the variable in `x` that will be represented with a color code inside every heatmap. Usually corresponds to the ranking column of `x`, which will most often contain integer values (negatives are allowed). When the SRCS object being plotted has been returned by a call to `SRCSranks`, this column is called "rank". For `animatedplot`, it should be a vector of strings containing the names of the ranks columns that will be depicted, each at a time, sorted by time instant (from the earliest to the most recent).

`out.Y.par,out.X.par`

A tagged list with parameters to configure how the labels of the outer Y and X variables and their levels are displayed. Valid parameters and their default values are as follows:

- `lab = TRUE` Label with the name of the variable. Will be displayed vertically for the outer Y variable and horizontally for the outer X variable. Valid values: FALSE for no label; NULL or TRUE for the name of the outer variable; and any string for a specific label. Defaults to TRUE.
- `lab.width = lcm(1)` Width of the left-most column (for the outer Y variable) or top row (for the outer X variable) containing the name of the variable.
- `lab.textpar = list(cex = 1.6)` Rest of parameters that will be passed to `text` to display this label. Parameter `cex` (text magnification factor) will be set 1.6 by default when not provided.
- `levels.lab = TRUE` Whether a label should be displayed (or not) for every level of the variable.
- `levels.lab.width = lcm(1)` Width of the row or column containing the levels of the variable.
- `levels.lab.textpar = list(cex = 1.4)` Tagged list with more parameters that will be passed directly to `text` to display this label. Parameter `cex` (text magnification factor) will be set 1.4 by default when not provided. NOTE: if present, the value of parameter `str` will always be overwritten by 0 (horizontal text) for the outer X, and 90 (vertical text) for the outer Y variable.
- `lab.bg = NULL` Background color of the rectangle where the label is placed. Default is transparent. No additional checks will be done concerning the validity of this parameter.
- `levels.bg = NULL` Background color of the rectangle where the levels of the label are placed. Default is transparent. No additional checks will be done concerning the validity of this parameter.
- `lab.border = NULL` Border color of the rectangle where the label is placed. Defaults to NULL (no line). No additional checks will be done concerning the validity of this parameter.

- `levels.border = NULL` Line color of the rectangle border where the levels of this label are placed. Defaults to NULL (no line). No additional checks will be done concerning the validity of this parameter.

`inner.X.par,inner.Y.par`

A tagged list with parameters to configure how the labels of the innter Y and X variables and their levels are displayed. Valid parameters and their default values are the following:

- `lab = TRUE` Inner label to be shown. Valid values are FALSE for no label, NULL or TRUE for the name of variable passed as argument to `plot.SRCS`, or any string for a specific label. Defaults to TRUE.
- `lab.width = lcm(1)` Width of the optional space for the label of the inner Y variable. The label will be repeated along the rows of the left-most column of heatmaps.
- `lab.textpar = list(cex = 1)` Rest of parameters passed to `text` to display this label.
- `levels.loc = c("bottom", "left", "all", "none")` Location of the inner level labels: only in heatmaps of the left-most column or the bottom row, or in every heatmap of the plot, or none. Defaults to "bottom" for the inner X variable and "left" for the inner Y variable. When levels.loc is set to "none", the value of params[["levels.at"]] is ignored.
- `levels.at = NULL` Levels of the inner variable where the label will be shown. Defaults to all the levels. They can be provided in any order, since the order in which they will be displayed only depends on the order defined by the levels argument when that factor column of the data was created.
- `levels.las = 1` Orientation of the level labels of this variable, defined as in `axis`. 1 for horizontal, 2 for vertical.

`colorbar.par` Tagged list to configure the aspect of the colorbar legend displayed on the right part of the figure:

- `levels.at = NULL` String vector: Levels at which the Y axis ticks of the colorbar will be shown. By default, three levels will be labeled: 0, the `min(x[[zInner]])` and `max(x[[zInner]])`.
- `hlines = TRUE` Logical: whether black horizontal lines should be displayed in the colorbar to separate the colors. Defaults to TRUE.

`color.function` A custom function that receives one argument (number of colors to be generated, (maxrank - minrank + 1) in our case) and returns a vector of that length with the hexadecimal codes of the colors to be used in the heatmaps, see `heat.colors` or `terrain.colors` for instance. Defaults to the `heat.colors` function.

`heatmaps.per.row`

Maximum number of heatmaps displayed in a row of the grid. Useful when variable `xOuter` has too many levels so they can be splitted in two or more sub-rows of heatmaps, with all the sub-rows corresponding to a single level of the `yOuter` variable.

`heatmaps.titles`

A vector of the same length as the total number of heatmaps, i.e. unique(x[[yOuter]]) * unique(x[[xOuter]]), containing the titles to be displayed on top of each heatmap. The elements of the vector will be associated to the heatmaps of the grid from left to right and from top to bottom.

`show.colorbar` Logical: whether a colorbar legend will be shown on the right of the figure (one for each row of heatmaps) or not. Defaults to TRUE

`annotation.lab` String with the annotation title that will be displayer on the top left corner. Defaults to NULL, indicating no annotation will be shown.

`heat.cell.par` Tagged list that will be passed to `par` just before displaying each heatmap. This way expert users can configure exactly the appearance of the heatmaps. No additional checks will be done concerning the validity of this list.

`heat.axes.par` Tagged list that will be passed to `axis` when creating the heatmap axes. No additional validity checks are done. The values of the arguments `side, at, labels` will always be replaced by suitable ones according to `inner.X.par[["levels.at"]]` or `inner.Y.par[["levels.at"]]`.

`colorbar.cell.par`
Tagged list that will be passed to `par` just before showing each colorbar. No additional validity checks are done.

`colorbar.axes.par`
Tagged list that will be passed to `axis` to draw the axes of the colorbar. No additional validity checks are done.

`annotation.text.par`
Tagged list that will be passed to `text` to show an additional title on the top left corner. No additional validity checks are done.

`...` (In `animatedplot`): Rest of optional parameters that will be passed to `plot.SRCS` to plot every frame. (In `singleplot`): A number of named arguments of the form `variable = value`, where `variable` is a column in `x`, for subsetting `x` in a way that there exists exactly one occurrence of all the levels of `zInner` for each combination of yInner,xInner.

`filename` Name of the output video file, including the extension. It is strongly recommended that the name ends in ".gif" to preserve most of image quality.

`path.to.converter`
String with the full path to the converter program delivered with ImageMagick, e.g. "C:/Program Files/ImageMagick-<version>/convert.exe"

`width,height` Width and height, in pixels, of the result video. Both default to 800

`res` Nominal resolution (in ppi) of the output video. Used to set text size and line widths. Defaults to 100. See `png`, `jpeg`, `bmp`, `tiff`.

`pointsize` Point size argument to be passed to the functions that print to image. Defaults to 16.

`delay` Time delay (in 1/100th of a second) spent in each of the images that compose the video. Defaults to 30, i.e. 0.3 seconds.

`type` The type of image file generated for each frame. The image files will be then joined together into a video. Should be one of `"png"`, `"jpeg"`, `"bmp"`, `"tiff"`.

`quality` The quality of the images, in a scale from 1 to 100. The less the quality, the more the compression and the smaller the file size.

`compression` (For TIFF format only) Used to indicate the kind of compression. Must be one of `"none"`, `"rle"`, `"lzw"`, `"jpeg"`. Ignored if `type` is not `"tiff"`.

annotations — Vector of strings with the annotation label of every image of the video. Should have the same length as zInner. Defaults to NULL (no annotations).

labels.par — Tagged list to configure how the labels will be displayed:

- xlab = TRUE Label with the name of the variable for the X axis. Will be displayed horizontally. Valid values: FALSE for no label; NULL or TRUE for the name of the outer variable; and any string for a specific label. Defaults to TRUE.
- ylab = TRUE Analogous for the Y axis.
- xlevels.at = NULL Levels of the X axis variable where the label will be shown. Defaults to all the levels. The levels can be provided in any order, since the order in which they will be depicted only depends on the original order defined when the corresponding factor column of the data was created.
- ylevels.at = NULL Analogous for Y axis variable.

haxis,vaxis — Whether the X and the Y axes should be displayed or not. Defaults to TRUE for both.

title — Title of the plot.

Details

plot.SRCS plots a grid with the results over all problem configurations, and should be applied to the object returned by SRCSranks with only one performance column.

singleplot is used for plotting only one heatmap for a subset of problem configurations in which the outer X and Y parameters take a fixed value, and should be applied to the object returned by SRCScomparison.

animatedplot creates a video from a sequence of plots, intended to show the temporal evolution of the ranking over time. It should be applied only to the object returned by SRCSranks when the performance argument passed to it was a vector of strings, each of them being the performance column of the data at a given time instant.

Note

The function uses the base graphics system.

See Also

text, par, axis, SRCSranks, animatedplot, singleplot, brewer.pal, RGB

Examples

```
# Example from a Machine Learning problem with noisy data
ranks = SRCSranks(ML1, params = c("Dataset", "Noise type", "Noise ratio"),
  target = "Algorithm", performance="Performance", maximize = TRUE, ncores = 2,
  paired = TRUE, pairing.col = "Fold");
singleplot(ranks, yInner = "Noise type",
   xInner = "Noise ratio", Algorithm = "C4.5", Dataset = "glass")
plot(x = ranks, yOuter = "Dataset", xOuter = "Algorithm", yInner = "Noise type",
  xInner = "Noise ratio", out.X.par = list(levels.lab.textpar =
```

```
  list(col = "white"), levels.bg = "black", levels.border = "white"),
  out.Y.par = list(levels.bg = "gray"), colorbar.axes.par = list(cex.axis = 0.8),
  show.colorbar = TRUE)
SRCScomparison(ranks, "Algorithm", Dataset = "automobile", `Noise type` = "ATT_GAUS",
  `Noise ratio`= 10, pvalues = FALSE)
# ------------------------------------------------
## Not run:
mat = matrix(NA, nrow = nrow(MPBall), ncol = ncol(MPBall))
# First, take the average of the previous performance columns up to each change point
for(j in 6:ncol(MPBall)){
  mat[,j] = rowSums(MPBall[,5:j])/(j-5+1)
}
MPBall[,6:ncol(MPBall)] = mat[,6:ncol(MPBall)]

ranksall = SRCSranks(MPBall, params = c("Dim", "CF", "Severity"), target="Algorithm",
   test = "tukeyHSD", performance=paste("OffError", seq(from=1, to = 100, by = 24),
   sep = "_"), maximize = FALSE, ncores = 2)

# Adjust argument path.to.converter to point to ImageMagick convert utility
animatedplot(x = ranksall, filename = "MPBconv_reduced.gif",
             path.to.converter = "C:/Program Files/ImageMagick-6.8.8-Q8/convert.exe",
             yOuter = "Algorithm", xOuter = "Dim", yInner = "CF", xInner = "Severity",
             zInner = paste0("rank",1:5), delay = 30,
             annotations = paste0("At change ",seq.int(from = 1, to = 100, by = 24)),
             inner.Y.par = list(levels.at = c("40", "200", "400", "600", "800", "1000"),
              lab = "Change\nfrequency", levels.loc = "left"),
             heat.cell.par = list(pty = "s"),
             inner.X.par = list(levels.at = c("2", "8", "14")),
             out.Y.par = list(levels.lab.textpar = list(cex = 1, col = "white"),
              levels.bg = "black", levels.border = "white"),
             out.X.par = list(lab = "Dimension", levels.bg = "gray"),
             colorbar.par = list(levels.at = c("-2", "0", "2")),
             colorbar.axes.par = list(cex.axis = 0.8),
             show.colorbar = TRUE, height = 500
            )
# The full dataset (20 MB) can be downloaded from
# http://decsai.ugr.es/~pjvi/SRCSfiles/MPBall.RData
# (the average must still be computed before plotting, just as in the example above)
# Check the script in http://decsai.ugr.es/~pjvi/SRCSfiles/DOPvideoScript.R

## End(Not run)
```

SRCS *R package implementing the Statistical Ranking Color Scheme for visualizing the results of multiple parameterized pairwise comparisons.*

Description

An R implementation of SRCS: Statistical Ranking Color Scheme for visualizing the results of multiple pairwise comparisons in many problem configurations at the same time, each defined by

at most 3 additional parameters. For each problem configuration, this technique ranks every level of the target value according to the performance in relation to how other levels perform on the same problem configuration. Ranks are assigned according to statistical performance comparisons. Then, a color is associated to each rank so it can be easily visualized and interpreted.

References

I.G. del Amo, D.A.Pelta. SRCS: a technique for comparing multiple algorithms under several factors in dynamic optimization problems, in: E. Alba, A. Nakib, P. Siarry (Eds.), Metaheuristics for Dynamic Optimization. Series: Studies in Computational Intelligence 433, Springer, Berlin/Heidelberg, 2012.

`SRCScomparison` *Compares the performance of two algorithms for a single problem configuration specified by the user.*

Description

Compares the performance of two algorithms for a single problem configuration specified by the user.

Usage

```
SRCScomparison(rankdata, target, alpha = 0.05, pvalues = FALSE, ...)
```

Arguments

`rankdata`	The ranks data frame obtained by a previous call to `SRCSranks`.
`target`	Name of the target column in `rframe` that separates the levels to be compared, probably "Algorithm" or similar.
`alpha`	Significance threshold to consider two set of measurements coming from two algorithms as statistically significant
`pvalues`	Boolean. TRUE indicates that the pairwise comparison table should contain p-values. FALSE means only ">","<" or "=" (the latter for non-significant difference) will be displayed in the table. Defaults to FALSE.
`...`	The rest of the columns in `rframe` and the values to fully specify a single problem configuration for which algorithms will be compared. Must be indicated as named arguments, like in "severity" = 4.

Value

A square matrix of the same dimension as algorithms found in the data. An entry i,j contains either the p-value of the Wilcoxon test between algorithms i and j (if `pvalues` was set to TRUE), or the qualitative result (">", "<" or "=") of the statistical comparison (if `pvalues` was set to FALSE).

See Also

`SRCSranks`, `plot.SRCS` for a full working example of SRCScomparison.

SRCSranks *Computes the ranks of all the algorithms from their (repeated) results measurements after grouping them by several factors combined simultaneosly.*

Description

Computes the ranks of all the algorithms from their (repeated) results measurements after grouping them by several factors combined simultaneosly.

Usage

```
SRCSranks(data, params, target, performance, pairing.col = NULL,
  test = c("wilcoxon", "t", "tukeyHSD", "custom"), fun = NULL,
  correction = p.adjust.methods, alpha = 0.05, maximize = TRUE,
  ncores = 1, paired = FALSE, ...)
```

Arguments

`data`	A dataframe object containing (at least) two columns for the target factor and the performance measure Additional columns are aimed at grouping the problem configuration by (at most) 3 different factors.
`params`	A vector with the column names in `data` that define a problem configuration. If not already factor objects, those columns will be converted to factors inside the function (note this does not alter the ordering of the levels in case it was explicitly set before the call). Although an arbitrary number of columns can be passed, if the user intends to plot the ranks computed by this function, at most three columns should be passed.
`target`	Name of the target column of `data`. For each combination of the values of `params`, the ranks are obtained by comparing the repeated measurements of `performance` associated to each level of the `target` column.
`performance`	Name of the column of `data` containing the repeated performance measurements. If given a vector of strings, then a separate ranking will be computed for each of the elements, and no p-values, mean or stdev columns will be returned, just the rankings together with the factors to indicate which problem configuration corresponds to the rank.
`pairing.col`	Name of the column which links together the paired samples, in case we have set `paired = TRUE`. Otherwise, this argument will be ignored.
`test`	The statistical test to be performed to compare the performance of every level of the target variable at each problem configuration.
`fun`	Function performing a custom statistical test, if `test = "custom"`; otherwise, this argument is ignored. The function must receive exactly two vectors (the first is a vector of real numbers and the second is a factor with the level to which each real number corresponds) and must return a `pairwise.htest` object with a `p.value` field. This must be an (N-1)x(N-1) lower-triangular matrix, with

exactly the same structure as those returned in the p.value field by a call to pairwise.wilcox.test or pairwise.t.test.

correction — The p-value adjust method. Must be one of "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none" (defaults to "holm"). This parameter will be ignored if test = "tukeyHSD" as Tukey HSD incorporates its own correction procedure.

alpha — Significance threshold for pairwise comparisons. Defaults to 0.05.

maximize — Boolean indicating whether the higher the performance measure, the better (default), or vice-versa.

ncores — Number of physical CPUs available for computations. If ncores > 1, parallelization is achieved through the parallel package and is applied to the computation of ranks for more than one problem configuration at the same time. Defaults to 1 (sequential).

paired — Boolean indicating whether samples in the same problem configuration, which only differ in the target value, and in the same relative position (row) within their respective target values are paired or not. Defaults to FALSE. This should be set to TRUE, for instance, in Machine Learning problems in which, for a fixed problem configuration, the target variable (usually the algorithms being compared) is associated to a number of samples (results) coming from the Cross Validation process. If a K-fold CV is being done, then we would have, for a given problem configuration, K rows for each of the algorithms being compared, all of them identical in all the columns except for the performance column. In that case, the performance of the i-th row (1 <= i <= K) of all of those batches (groups of K rows) for that fixed problem configuration would be related, hence every pairwise comparison should take into account paired samples.

... — Further arguments to be passed to the function fun that is called for every pairwise comparison.

Value

If length(performance) equals 1, an object of classes c("SRCS", "data.frame") with the following columns: - A set of columns with the same names as the params and target arguments. - Two columns called "mean" and "sd" containing the mean of the repeated peformance measurements for each problem configuration and the standard deviation. - One column named "rank" with the actual rank of each level of the target variable within that problem configuration. The lower the rank, the better the algorithm. - |target| additional columns containing the p-values resulting of the comparison between the algorithm and the rest for the same problem configuration, where |target| is the number of levels of the target variable.

If length(performance) > 1 (let P = length(performance) for the explanation that follows), an object of classes c("SRCS","data.frame") with the following columns: - A set of columns with the same names as the params and target arguments. - One column per element of the performance vector, named "rank1", ..., "rankP", containing, for each performance measure, the rank of each level of the target variable within that problem configuration for that performance measure. The higher the rank, the better the algorithm.

Note

Although it has no effect on the results of SRCSranks, the user should preferably have set the order of the factor levels explicitly by calling function levels before calling this function, specially if he intends to subsequently apply plot to the results, because the level order does affect the way graphics are arranged in the plot.

See Also

plot.SRCS for a full working example of SRCSranks and plotting facilities. Also pairwise.wilcox.test, t.test, pairwise.t.test, TukeyHSD, p.adjust.methods

Index

∗ **datasets**
ML1, 2
MPB, 3
MPBall, 4

animatedplot, *4*, *6*, *9*
animatedplot *(*plot.SRCS*)*, 5
axis, *7–9*

bmp, *8*
brewer.pal, *9*

jpeg, *8*

ML1, 2
MPB, 3
MPBall, 4

p.adjust.methods, *14*
pairwise.t.test, *13, 14*
pairwise.wilcox.test, *13, 14*
par, *8, 9*
parallel, *13*
plot, *14*
plot.SRCS, *5*, 5, *7, 8*, *11*, *14*
png, *8*

RGB, *9*

singleplot, *9*
singleplot *(*plot.SRCS*)*, 5
SRCS, 10
SRCS-package *(*SRCS*)*, 10
SRCScomparison, *9*, 11
SRCSranks, *5*, *9*, *11*, 12

t.test, *14*
text, *6–9*
tiff, *8*
TukeyHSD, *14*