Package 'sisal'

February 15, 2020

Encoding UTF-8

Type Package

Title Sequential Input Selection Algorithm

Version 0.48

Date 2020-02-14

Author Mikko Korpela [aut, cre]

Maintainer Mikko Korpela <mvkorpel@iki.fi>

Depends R (>= 3.1.2)

Imports graphics, grDevices, grid, methods, stats, utils, boot, lattice, mgcv, digest, R.matlab, R.methodsS3

Suggests graph, Rgraphviz, testthat (>= 0.8)

Description Implements the SISAL algorithm by Tikka and Hollmén. It is a sequential backward selection algorithm which uses a linear model in a cross-validation setting. Starting from the full model, one variable at a time is removed based on the regression coefficients. From this set of models, a parsimonious (sparse) model is found by choosing the model with the smallest number of variables among those models where the validation error is smaller than a threshold. Also implements extensions which explore larger parts of the search space and/or use ridge regression instead of ordinary least squares.

License GPL (>= 2)

URL https://github.com/mvkorpel/sisal

BugReports https://github.com/mvkorpel/sisal/issues

LazyData yes

NeedsCompilation no

Repository CRAN

Date/Publication 2020-02-15 21:00:03 UTC

R topics documented:

`sisal-package` *sisal: Sequential input selection algorithm*

Description

Implements the SISAL algorithm by Tikka and Hollmén. It is a sequential backward selection algorithm which uses a linear model in a cross-validation setting. Starting from the full model, one variable at a time is removed based on the regression coefficients. From this set of models, a parsimonious (sparse) model is found by choosing the model with the smallest number of variables among those models where the validation error is smaller than a threshold. Also implements extensions which explore larger parts of the search space and/or use ridge regression instead of ordinary least squares.

Details

Package:	sisal
Depends:	R (>= 3.1.2)
Imports:	graphics, grDevices, grid, methods, stats, utils, boot, lattice, mgcv, digest, R.matlab, R.methodsS3
Suggests:	graph, Rgraphviz, testthat (>= 0.8)
License:	GPL (>= 2)
LazyData:	yes

Index:

	SISAL Object
dynTextGrob	Create Text with Changing Size
laggedData	Create Input Matrix and Output Vector for Time Series Prediction
plot.sisal	Plotting Sequential Input Selection Results
plotSelected.sisal	Plotting Sets of Inputs Produced by Sequential Input Selection
print.sisal	Printing Sequential Input Selection Objects
sisal	Sequential Input Selection Algorithm (SISAL)
sisal-package	sisal: Sequential input selection algorithm in R
sisalData	Download External Datasets for SISAL
sisalTable	Draw Table with Equally Sized Cells
summary.sisal	Summarizing Sequential Input Selection Results
testSisal	Testing the Sequential Input Selection Algorithm
toy.learn	Toy Data for SISAL (Learning Set)
toy.test	Toy Data for SISAL (Test Set)
tsToy.learn	Toy Time Series Data for SISAL (Learning Set)
tsToy.test	Toy Time Series Data for SISAL (Test Set)

Run input selection on your own data with sisal. For demo purposes, use testSisal to run the algorithm on example data sets. After input selection, compute bootstrap MSE in test data with bootMSE.

Author(s)

Mikko Korpela <mvkorpel@iki.fi>

References

Tikka, J. and Hollmén, J. (2008) Sequential input selection algorithm for long-term prediction of time series. *Neurocomputing*, 71(13–15):2604–2615.

---

bootMSE *Bootstrap Estimate of Mean Squared Error Using* SISAL *Object*

Description

Using a linear model produced by sisal, computes a bootstrap estimate of MSE in test data.

Usage

```
bootMSE(object, dataset = NULL, R = 1000,
        inputs = c("L.f", "L.v", "full"),
        method = c("OLS", "magic"), standardize = "inherit",
        stepsAhead = NULL, noiseSd = NULL, verbose = 1, ...)
```

Arguments

object	an object of class "sisal", containing the results of input selection and the corresponding ordinary least squares and ridge regression models. Must be compatible with *dataset*. See 'Details'.
dataset	dataset to work on. A character string, a numeric vector or a list with components "X" and "y". When the default value NULL is used, the function attempts to detect the dataset from attributes of *object*. See 'Details'.
R	the number of bootstrap replicates. Usually a single positive integral number. See boot::boot.
inputs	a character string. Which set of input variables to use. Choices are "L.f" (smallest set with error under threshold), "L.v" (minimum validation error) and "full" (full model). See sisal.
method	a character string. "OLS" for ordinary least squares regression or "magic" for a ridge regression model with an automatically selected regularization parameter. See sisal.
standardize	"inherit" or a logical flag. If TRUE, standardizes the data to zero mean and unit variance. If FALSE, uses original data. If "inherit" (the default), the value of this argument is copied from *object*. This affects the scale of the results.
stepsAhead	If doing time series prediction, this indicates how many steps ahead to predict. A non-negative integral value or NULL. If NULL (the default), the value is copied from an attribute of *object*, put there by testSisal.
noiseSd	standard deviation of the noise to be added to the dependent variable when *dataset* is "toy". The noise is a saved dataset. Thus it is always identical, only scaled by noiseSd. If NULL (the default), the value is copied from *object*.
verbose	verbosity level. A single numeric value. If 0, output is disabled. If greater than 0, shows some information about what the function is doing. Currently there is only one non-zero verbosity level (the default).
...	arguments passed to boot::boot.

Details

Four types of values are supported in *dataset*.

1. Use one of "laser", "poland", "toy" and "tsToy" to work on the test part of a dataset included in or specifically supported by the package. The first two options will load their respective datasets over a network connection. See sisalData, toy.test and tsToy.test.
2. Use a numeric vector to work with time series data. The use of the "laser" and "poland" datasets is recognized. Loading the datasets in advance reduces unnecessary network traffic when doing multiple repeats with the same dataset.
3. Use a list with a numeric matrix "X" and a numeric vector "y" to supply inputs "X" and output "y". This is appropriate when using your own data for something else than time series prediction based on past values of the same time series.
4. Use NULL (the default value) for automatic detection of the dataset. This works if *object* was created with testSisal.

When using time series data, the names of the inputs used in `object` must match the regular expression `"lag\.\d+"`, i.e. `"lag"` followed by a dot and an integer without spaces or any other formatting. This is automatically taken care of by `laggedData` and `testSisal`.

When using other than time series data, the user-supplied *dataset* must contain all the input variables used in the selected linear model (i.e. full model or a subset of inputs) of *object*.

Value

An object of class `"boot"`, as returned by `boot::boot`.

Author(s)

Mikko Korpela

See Also

`boot::boot`, `sisal`, `testSisal`

Examples

```
foo <- testSisal(dataset="toy", Mtimes=10)
bootMSE(foo)
```

---

dynTextGrob *Create Text with Changing Size*

Description

This function creates a text object. When drawn, its size changes automatically according to the space available.

Usage

```
dynTextGrob(label, x = 0.5, y = 0.5, width = 1, height = 1,
            default.units = "npc", just = c(0.5, 0.5),
            hjust = NULL, vjust = NULL, rot = 0, rotJust = TRUE,
            rotHjust = NULL, rotVjust = NULL, resize = TRUE,
            sizingWidth = NULL, sizingHeight = NULL,
            adjustJust = TRUE, takeMeasurements = FALSE,
            name = NULL, gp = gpar(), vp = NULL)
```

Arguments

`label`	a `character` or `expression` vector, or a `list` containing both character strings and mathematical expressions. These are the text items to be drawn.
`x`	a `numeric` vector or `unit` of x locations for the labels.
`y`	a `numeric` vector or `unit` of y locations for the labels.

`width` the space available for the labels in the width direction of the viewport. Used for computing the fontsize.

`height` the space available for the labels in the height direction of the viewport. Used for computing the fontsize.

`default.units` default unit to use when dimensions or locations are unitless numbers. See `unit`.

`just` a `numeric` or `character` vector with one or two elements for setting the same justification for all labels. See `textGrob`.

`hjust` a `numeric` vector for setting horizontal justification of individual labels. If given, overrides `just`.

`vjust` a `numeric` vector for setting vertical justification of individual labels. If given, overrides `just`.

`rot` a `numeric` vector for setting the rotation angle of individual labels in degrees.

`rotJust` a `logical` vector which affects the justification of individual labels. If an element is `FALSE`, the corresponding label is first justified according to `hjust` (reading direction) and `vjust` (the perpendicular direction), then rotated. This is the way a `textGrob` works. If an element is `TRUE`, the concept is: align the label with the other labels according to `rotHjust` (reading direction) and `rotVjust` (the perpendicular direction), then rotate, and finally justify in the width and height directions of the viewport with `hjust` and `vjust`, respectively.

`rotHjust` a `numeric` vector or `NULL`. When the corresponding element of `rotJust` is `TRUE`, `rotHjust` sets the justification of a label in the reading direction. If `NULL` or an `NA` element is encountered, an automatic value will be computed based on rotation angle (`rot`) and justification along the viewport axes (`just`, `hjust` and `vjust`).

`rotVjust` a `numeric` vector or `NULL`. Set the justification of labels perpendicular to the reading direction when `rotJust` is `TRUE`. See `rotHjust`.

`resize` a `logical` flag. If `TRUE` (the default), the fontsize of the labels will be adjusted according to the space available. If `FALSE`, the size will remain constant, even if the graphical object is drawn in a viewport with a different setting for the `"cex"` graphical parameter.

`sizingWidth` If `resize` is `TRUE`, a `numeric` value given here sets the width of the grob used when calculating fontsize at drawing time. If `NULL` (the default), the size is computed from the actual dimensions of the labels.

`sizingHeight` See `sizingWidth`, only height instead of width.

`adjustJust` A `logical` flag. If `TRUE` (the default), adjustments are made to the justification of the labels instead of passing the justification settings straight to the underlying `textGrob`(s). The justification of labels given in `expression` form will be unified with the justification of `character` labels, meaning that a setting of `vjust = 0` will align the baselines of the labels and `vjust = 1` will align the labels at lineheight, or at a multiple of lineheight in case of multiline `character` labels. The labels will also be shifted so that there is room for descenders.

`takeMeasurements` A `logical` flag. If `TRUE`, only measurements of labels will be returned instead of a graphical object. An example of where this might be useful is when several labels should have the same fontsize but different graphical parameters such

as color, or when the labels should be drawn in different viewports. See the source of `sisalTable`, particularly `makeContent.sisalTable`, for an example. If `FALSE` (the default) a graphical object will be returned.

`name` a `character` string identifier for the graphical object returned by the function. If `NULL` (the default), a name will be assigned automatically.

`gp` graphical parameters. See `gpar`.

`vp` a `"viewport"` object, the name of a viewport object, a `vpPath` object pointing to a viewport or `NULL` (the default). If not `NULL`, this graphical object will be drawn in the given viewport. The name or the path must point to a descendant of the viewport that is current at drawing time. See `current.vpPath`, `current.vpTree`, `downViewport` and `grid.draw`.

Details

The number of labels created is the maximum of the lengths of `x` and `y`. Variables are recycled to that length if necessary.

All labels of one `"dynText"` grob have the same fontsize.

Value

If `takeMeasurements` is `FALSE` (the default), returns a `grob` of `class` `"dynText"`. It can be drawn with `grid.draw`.

If `takeMeasurements` is `TRUE`, returns a `list` containing measurements of the labels.

Author(s)

Mikko Korpela

See Also

See function `textGrob` in package grid.

Examples

```
library(grid)
grid.newpage()
grid.draw(dynTextGrob("Hello", vjust = 0, y = 0))
grid.draw(dynTextGrob(list(expression(y==x^2),
                           "Hello,\ntry resizing me!"),
                      x = rep(1, 2), y = 1, rot = -45,
                      hjust = 1, vjust = 1,
                      rotHjust = c(0, 1), rotVjust = 1))
```

`laggedData` *Create Input Matrix and Output Vector for Time Series Prediction*

Description

Given a time series vector, produces the input matrix and output vector for a time series prediction task. The other parameters are the lags to include and the number of steps ahead to predict.

Usage

```
laggedData(x, lags = 0:9, stepsAhead = 1)
```

Arguments

`x`	an `atomic vector` representing a (uniformly sampled) time series. Any attributes are ignored.
`lags`	which lags to use for prediction. A `vector` of non-negative integral values.
`stepsAhead`	how many steps ahead to predict. A non-negative integral value (`integer` or `numeric`).

Details

The default parameters correspond to predicting one step ahead (position $t+1$) using the ten most recent values (positions $t \ldots t-9$).

Value

A `list` with two components:

`X`	The (`length(`x`) -max(`*lags*`) -`*stepsAhead*) rows by `length(`*lags*`)` columns input `matrix` with the same type as x.
`y`	The output `vector` with `length(`x`) -max(`*lags*`) -`*stepsAhead* elements. Same type as x.

Author(s)

Mikko Korpela

Examples

```
laggedData(1:20)
```

plot.sisal *Plotting Sequential Input Selection Results*

Description

A plot method for class "sisal". Supports 3 plot types: error as a function of the number of variables, search graph, and color key of the search graph.

Usage

```
## S3 method for class 'sisal'
plot(x, which = 1, standardize = "inherit", ...,
     plotArgs = list(list(), list(mai = rep(0.1, 4))),
     xlim = c(x[["d"]], 0), ylim = NULL, ask = TRUE,
     dev.set = !ask, draw.node.labels = TRUE,
     draw.edge.labels = TRUE, draw.selected.labels = TRUE,
     rankdir = c("TB", "LR", "BT", "RL"),
     fillcolor.normal = "deepskyblue",
     fillcolor.pruned = "deeppink",
     fillcolor.selected = "chartreuse",
     fillcolor.levelbest = "gold",
     fillcolor.small = "moccasin", fillcolor.large = "black",
     fillcolor.NA = "white",
     bordercolor.normal = "black",
     bordercolor.special.levelbest = fillcolor.levelbest,
     bordercolor.special.selected = fillcolor.selected,
     color.by.error = FALSE,
     ramp.space = c("Lab", "rgb"), ramp.size = 128,
     error.limits = c(NA_real_, NA_real_),
     category.labels =
         c(normal = gettext("Other", domain="R-sisal"),
           pruned = gettext("Pruned", domain="R-sisal"),
           levelbest = gettext("Best\nin class", domain="R-sisal"),
           selected = gettext("Selected", domain="R-sisal"),
           special.levelbest = gettext("Best\n(no branching)",
                                       domain="R-sisal"),
           special.selected = gettext("Selected\n(no branching)",
                                      domain="R-sisal"),
           shape.normal=gettext("Other", domain="R-sisal"),
           shape.highlighted=gettext("Highlighted", domain="R-sisal")),
     integrate.colorkey = TRUE, colorkey.gap = 0.1,
     colorkey.space = c("right", "bottom", "left", "top"),
     colorkey.title.gp = gpar(fontface = "bold"),
     nodesep = 0.25, ranksep = 0.5,
     graph.attributes = character(0),
     node.attributes = character(0),
     edge.attributes = character(0))
```

Arguments

x	an object of class "sisal".
which	which plots to draw. A numeric vector containing a subset of the following numbers: **1** error vs. number of inputs. **2** search graph. A directed acyclic graph (DAG). **3** node shape and color keys for the search graph. Requires that plot 2 is drawn, too. The default is to draw plot number 1. For drawing plot number 2, Bioconductor packages "graph" and "Rgraphviz" must be installed. Some other arguments of this method only apply to specific plots.
standardize	"inherit" or a logical flag. If TRUE, the error values in plot 1 correspond to standardized data (see *standardize* in sisal). If FALSE, the original scale of the data is used instead. If "inherit" (the default), the value of this argument is copied from *x*.
...	arguments passed to plot and matplot. These are used in all plots where plot or matplot do the actual drawing. For more fine-grained control and passing arguments to other graphical functions, use the plotArgs argument.
plotArgs	arguments passed to graphical functions. A list where *plotArgs*[[*k*]] (if present) are named lists of arguments passed to plot number *k*. See 'Details'.
xlim	the x limits c(x1,x2) of plot 1. A numeric vector. Defaults to showing the whole range, i.e. everything between no input variables at all (except possibly an intercept) and the maximum number of inputs.
ylim	the y limits c(x1,x2) of plot 1. A numeric vector. If NULL (the default), adjusts to the range of y values corresponding to x values delimited by xlim.
ask	a logical flag. If TRUE (the default) and !dev.set, prompts the user before replacing a plot drawn with this function with another one. The user will not be alerted as long as there are free slots in the plot layout (see mfrow and mfcol in par).
dev.set	a logical flag. If TRUE, the function calls dev.set for switching to the next available graphical device when it runs out of free slots in the plot layout. If FALSE, the same graphical device is used for all the plots. The default value is !ask.
draw.node.labels	a logical flag. If TRUE, label the nodes of the search graph plot representing one input variable.
draw.edge.labels	a logical flag. If TRUE, label the edges of the search graph plot with the identity of the removed input variable.
draw.selected.labels	a logical flag. If TRUE, label the nodes of the search graph plot representing the L.v and L.f input variable sets of sisal.
rankdir	the drawing direction of plot number 2 (search graph). A character string, one of "TB" (top to bottom, the default), "LR" (left to right), "BT" (bottom to top), or "RL" (right to left).

`fillcolor.normal`
: fill color for normal nodes in plot number 2.

`fillcolor.pruned`
: fill color for pruned (unevaluated) nodes in plot 2. If *color.by.error* is `TRUE`, this color is used as the border color.

`fillcolor.selected`
: fill color for nodes representing the L.v and L.f input variable sets of `sisal` in plot 2. If *color.by.error* is `TRUE`, this color is used as the border color.

`fillcolor.levelbest`
: fill color for nodes with the smallest validation error using a given number of input variables in plot 2. If *color.by.error* is `TRUE`, this color is used as the border color.

`fillcolor.small`
: if `color.by.error` is `TRUE`, fill color for nodes with small validation error in plot 2.

`fillcolor.large`
: if `color.by.error` is `TRUE`, fill color for nodes with large validation error in plot 2.

`fillcolor.NA` if `color.by.error` is `TRUE`, fill color for pruned (unevaluated) nodes in plot 2.

`bordercolor.normal`
: border color for normal nodes in plot 2.

`bordercolor.special.levelbest`
: border color for special nodes in plot 2. If branching (*hbranches* > 1) reduces validation error with a given number of input variables, the "no branching" node is marked with this border color. If *pruning.keep.best* is `FALSE`, the comparison may not be possible for all sizes of the input variable set.

`bordercolor.special.selected`
: border color for another kind of special nodes in plot 2. The "no branching" L.v or L.f node, if different from the corresponding node in the solution where branching is allowed, is marked with this border color. If *pruning.keep.best* is `FALSE`, these alternative L.v and L.f nodes may not be defined, in which case the special color will not be used. If *color.by.error* is `TRUE`, this border color is also used to mark nodes that would be marked with *fillcolor.selected* in the case where *color.by.error* is `FALSE`.

`color.by.error` a `logical` flag. If `TRUE` nodes in plot 2 are colored using a color gradient between `fillcolor.small` and `fillcolor.large` according to the validation error in the node. If `FALSE`, the nodes are colored by category (normal, pruned, selected, levelbest).

`ramp.space` color space to be used in plots number 2 and 3 if `color.by.error` is `TRUE`. Either `"Lab"` (the default) or `"rgb"`. See `colorRamp`.

`ramp.size` the number of colors to be used in the color gradient of plot number 3 if `color.by.error` is `TRUE`. See `colorRampPalette`.

`error.limits` a `numeric vector` giving the minimum (first value) and maximum (second value) validation error. These are used as the endpoints of the color gradient used in plots number 2 and 3 if `color.by.error` is `TRUE`.

`category.labels`
: text labels to be used in plot number 3 if `color.by.error` is `FALSE`. A `character vector` with elements named `"normal"`, `"pruned"`, `"levelbest"` and `"selected"`. See the corresponding arguments with the name prefix `"fillcolor"`. The vector must also have elements named `"special.levelbest"` and `"special.selected"`. See the corresponding arguments with the name prefix `"bordercolor"`. The final required elements are `"shape.normal"` and `"shape.highlighted"`, which correspond to rectangular and circular nodes, respectively. Circular shape highlights nodes that have the lowest validation error considering the number of inputs used. Also highlighted is each node with the lowest validation error per number of variables but without using branches, if available and different from the unrestricted best node.

`integrate.colorkey`
: a `logical` flag. If `TRUE`, plots 2 (graph) and 3 (color and shape key for the graph) will be integrated if possible. This involves a version requirement on the `"Rgraphviz"` package. If `FALSE` or the version requirement is not met, the plots will be drawn separately.

`colorkey.gap`
: a `numeric` value giving the space (in inches) between the graph and the color key when plot 2 and 3 are integrated (*integrate.colorkey*).

`colorkey.space`
: location of the color and shape key (plot 3) relative to the graph (plot 2). One of `"bottom"`, `"right"`, `"top"` and `"left"`.

`colorkey.title.gp`
: graphical parameters for the titles in plot 3. See `gpar`.

`nodesep`
: a Graphviz attribute giving the minimum space in inches between adjacent nodes representing the same number of input variables. This `numeric` value applies to plot number 2.

`ranksep`
: a Graphviz attribute giving the minimum space in inches between adjacent rows or columns of nodes, where a row or column consists of nodes representing the same number of input variables. This `numeric` value applies to plot number 2.

`graph.attributes`
: a named `character vector` of extra Graphviz graph attributes. Applies to plot number 2.

`node.attributes`
: a named `character vector` of extra Graphviz node attributes. Applies to plot number 2.

`edge.attributes`
: a named `character vector` of extra Graphviz edge attributes. Applies to plot number 2.

Details

In argument `plotArgs`, *plotArgs*`[[1]]` is passed to `matplot`, *plotArgs*`[[2]]` to the plot method for class `"Ragraph"`, and *plotArgs*`[[3]]` to `draw.colorkey$key`.

For possible color values, see `col2rgb`.

Value

When `2 %in% which`, the function invisibly returns a graph of class "graphNEL" representing the search graph of a run of `sisal`. Otherwise `NULL`.

Author(s)

Mikko Korpela

References

For information about graph, node and edge attributes for plot number 2, see the Graphviz web site: http://www.graphviz.org/.

See Also

`sisal`

Examples

```
library(graphics)
foo <- testSisal(dataset="toy", Mtimes=10)
## Plotting the search graph requires "Rgraphviz" and "graph"
if (requireNamespace("Rgraphviz", quietly=TRUE) &&
    requireNamespace("graph", quietly=TRUE)) {
    plot(foo, which=2)
}
## Default output is a mean squared error plot
plot(foo)
```

---

plotSelected.sisal *Plotting Sets of Inputs Produced by Sequential Input Selection*

Description

Draws a table depicting the inputs selected by a number of `sisal` runs, one row for each run.

Usage

```
## S3 method for class 'sisal'
plotSelected(x, useAllNames = TRUE,
             pickIntPart = FALSE, intTransform = function(x) x,
             formatCArgs = list(), xLabels = 1, yLabels = NULL,
             L.f.color = "black", L.v.color = "grey50",
             other.color = "white", naFill = other.color,
             naStripes = L.v.color, selectedLabels = TRUE,
             otherLabels = FALSE,
             labelPar = gpar(fontface = 1, fontsize = 20, cex = 0.35),
             nestedPar = gpar(fontface = 3),
```

```
              ranking = c("pairwise", "nested"), tableArgs = list(),
              ...)

## S3 method for class 'list'
plotSelected(x, ...)
```

Arguments

x	an object of class "sisal" or a list of such objects giving the results of input selection.
useAllNames	a logical flag. If TRUE, collects the names of input variables from all elements of a list *x* or from the single "sisal" object. Each unique name is represented by one column in the table. If FALSE, all elements of *x* are assumed to have the same set of input variables in the same order.
pickIntPart	a logical vector. If *pickIntPart*[*k*] is TRUE, the input names collected from *x*[[*k*]] (*x* is a list) or from *x* (*x* is a single "sisal" object and *k* == 1) are filtered so that any name containing an integer part is converted to that integer (the remaining part is dropped). If the length of the vector and the number of rows in the table differ, the values of the vector are recycled.
intTransform	a function that transforms integral valued input names to another integer. Used if and only if the relevant element of *pickIntPart* is TRUE. The function must accept a numeric vector argument and return a numeric vector. The default value is an identity function.
formatCArgs	a named list of arguments to formatC. If the relevant element of *pickIntPart* is TRUE, the integral valued column names are formatted with formatC using these arguments. For example, it is possible to add a sign with list(flag = "+").
xLabels	a numeric value, character vector or list affecting the column labels in the table. If useAllNames is TRUE, a named list or character vector can be used to rename inputs. In this case, the names in the vector must contain all the input names gathered from x. The new names (display names) are taken from the values in the vector, indexed with the names from x. If useAllNames is TRUE, a numeric value has no effect. If useAllNames is FALSE, a numeric value is an index to x indicating the object to be used when collecting input names. An unnamed list or character vector of column names can also be used when useAllNames is FALSE.
yLabels	a character vector or list giving the row labels in the table. NULL (the default) means no labels.
L.f.color	fill color for table cells representing an input variable in the *L.f* set.
L.v.color	fill color for table cells representing an input variable in the *L.v* set.
other.color	fill color for table cells representing an input variable outside both *L.f* and *L.v*.
naFill	background color for table cells representing a missing input variable.
naStripes	stripe color for table cells representing a missing input variable.
selectedLabels	a logical flag. If TRUE (the default), draw labels on table cells representing input variables in the *L.f* or *L.v* sets. The label shows the importance rank of the variable. See 'Details'.

otherLabels	a logical flag. If TRUE, draw labels on table cells representing input variables not included the *L.f* or *L.v* sets. The label shows the importance rank of the variable. The default value is FALSE. See 'Details'.
labelPar	graphical parameters for labels of table cells.
nestedPar	graphical parameters for labels on rows that represent input selection runs where the best nodes of each size are all nested. See 'Details'. Only used if *ranking* includes "nested". These take precedence over values set in *labelPar*.
ranking	which input ranking method(s) to use. A character vector containing one or both of "pairwise" and "nested". Abbreviated versions can be used. See 'Details' for a description of the ranking methods. If both rankings are requested by the user and exist, they are both written on the label, but only where the ranks differ. The first element indicates the preferred primary ranking method, and any differing ranks produced by a possible secondary ranking method are presented in parentheses after the rank indicated by the primary method. The default is to use both methods when possible, preferring the always available "pairwise" method.
tableArgs	a named list of arguments passed to sisalTable. This can also be used when arguments of sisalTable and the "sisal" method of plotSelected have the same name.
...	In the "sisal" method, arguments passed to sisalTable. In the "list" method, arguments passed to the next method, determined by the class of the first element in the list.

Details

Currently the "sisal" and "list" methods are the only methods for the generic function plotSelected defined by the sisal package.

Mathematical annotation can be used in text. See plotmath. If the same input is in both the *L.f* and the *L.v* sets, *L.f.color* and *L.v.color* are mixed in alternating stripes. See col2rgb for a description of possible color values.

The importance rank of input variables is determined using one or both of the following two methods (see *ranking*):

"nested" This method requires that all the nodes with the smallest validation error among the nodes with the same number of input variables are nested. Let's imagine a path through the incrementally smaller best nodes (not necessarily a path in the search graph) where the edges are labeled with the ID of the input removed in order to create the smaller model. In this ranking method, the remaining input variable gets rank 1. Traversing the path in the reverse direction and printing the edge labels produces the rest of the input variables from smaller rank to larger. If hbranches = 1 in sisal, the models are always nested and the method agrees with "pairwise".

"pairwise" This is Copeland's pairwise aggregation method. It can be used in all cases, unlike "nested". The score of an input variable is the number of pairwise victories minus the number of pairwise defeats when compared with other inputs. The inputs are ranked by their score. The method may result in ties. Tied nodes are ranked according to *ties.method* = "min" in rank.

The pairwise comparisons are performed in the following way: In `sisal`, at each stage of the search, input variables are ordered and inputs are removed starting from one or more (when *hbranches* > 1) of the worst ones according to that order. A record, let's say $C[A, B]$, is kept of each pair of inputs (A, B) in order to keep track of how many times A was better than B. Let L be the set of inputs to remove at the current stage of the search in one of the branches and M the set of remaining inputs. Then, $C[A, B]$ is incremented by one for all A in M and B in L, but also for all A in L and B in L such that A is better than B according to the order used for picking the inputs to remove. A gets a pairwise victory over B if $C[A, B] > C[B, A]$.

For information on setting graphical parameters (*labelPar*, *nestedPar*), see `gpar`.

Value

The function is usually called for the side effect (a plot is drawn), but it also returns a `grob` representation of the plot.

Author(s)

Mikko Korpela

References

Pomerol, J.-C. and Barba-Romero, S. (2000) *Multicriterion decision in management: principles and practice*. Springer. p. 122. ISBN: 0-7923-7756-7.

See Also

`sisal`, `sisalTable`, plotmath, `gpar`

Examples

```
library(grDevices)
library(grid)
toy1.2 <- list(testSisal(Mtimes=10, stepsAhead=1, dataset="tsToy"),
               testSisal(Mtimes=10, stepsAhead=2, dataset="tsToy"))
## Resizing enabled:
## - mathematical expressions in titles
## - extracting the integer part of input variable names
grid.newpage()
plotSelected(toy1.2, yLabels = c("+1", "+2"),
             main = "Toy time series",
             xlab = expression(paste("input variables ",
                                     italic(y[t+l]))),
             ylab = expression(paste("output ", italic(y[t+k]))),
             pickIntPart = TRUE, intTransform = function(x) -x)
## Fixed size plot:
## - some graphical parameters adjusted
## - cex in labelPar adjusts the space around the text in table cells
## - new device the same size as the plot
grb <- plotSelected(toy1.2, resizeText = FALSE, resizeTable = FALSE,
```

```
                    axesPar = gpar(fontsize = 11, col = "red"),
                    labelPar = gpar(fontsize = 14/0.25, cex = 0.25),
                    fg = "wheat", outerRect = FALSE,
                    linePar = gpar(lty = "dashed"),
                    xAxisRot = 45, just = c("left", "top"),
                    tableArgs = list(x = 0, y = 1), draw = FALSE)
devWidth <- convertWidth(grobWidth(grb), unitTo = "inches",
                         valueOnly = TRUE)
devHeight <- convertHeight(grobHeight(grb), unitTo = "inches",
                           valueOnly = TRUE)
dev.new(width = devWidth, height = devHeight, units = "in", res = 72)
grid.draw(grb)
if (interactive()) {
    dev.set(dev.prev())
} else {
    dev.off()
}
```

`print.sisal` *Printing Sequential Input Selection Objects*

Description

Prints information contained in a sequential input selection object.

Usage

```
## S3 method for class 'sisal'
print(x, max.warn = 10, ...)
```

Arguments

`x`	an object of class `"sisal"`.
`max.warn`	a `numeric` value giving the maximum number of warnings to show. See *max.warn* in `sisal`.
`...`	additional arguments passed to other `print` methods.

Details

The following information is printed:

- Parameter values used in the `sisal` call
- Data dimensions
- Names of the input variables, if available
- Selected inputs, *L.v* (smallest validation error)
- Selected inputs, *L.f* (result within error margin)
- Whether *L.f* is a subset of *L.v* (nested model) or not

- The removal order and / or rank of the input variables (see `plotSelected.sisal`)
- The stages of search (if any) at which branching reduced validation error compared to a `hbranches = 1` solution. Not printed if branching was not used or if it is possible that the search did not proceed through every set of variables on the `hbranches = 1` path, i.e. if `pruning.keep.best` was `FALSE`. One must note that these results, like many others, are subject to randomness. Thus the results may differ between successive runs of `sisal`.
- Any warnings produced by the `sisal` run (see *max.warn*)

Value

Invisibly returns *x*.

Author(s)

Mikko Korpela

See Also

More information can be obtained with `summary.sisal`.

Examples

```
foo <- testSisal(dataset="toy", nData = 200, Mtimes = 10,
                 noiseSd = 0.5, verbose = 0)
print(foo)
```

`sisal` *Sequential Input Selection Algorithm (*SISAL*)*

Description

Identifies relevant inputs using a backward selection type algorithm with optional branching. Choices are made by assessing linear models estimated with ordinary least squares or ridge regression in a cross-validation setting.

Usage

```
sisal(X, y, Mtimes = 100, kfold = 10, hbranches = 1,
      max.width = hbranches^2, q = 0.165, standardize = TRUE,
      pruning.criterion = c("round robin", "random nodes",
                            "random edges", "greedy"),
      pruning.keep.best = TRUE, pruning.reverse = FALSE,
      verbose = 1, use.ridge = FALSE,
      max.warn = getOption("nwarnings"), sp = -1, ...)
```

Arguments

X	a numeric matrix where each column is a predictor (independent variable) and each row is an observation (data point)
y	a numeric vector containing a sample of the response (dependent) variable, in the same order as the rows of X
Mtimes	the number of times the cross-validation is repeated, i.e. the number of predictions made for each data point. An integral value (numeric or integer).
kfold	the number of approximately equally sized parts used for partitioning the data on each cross-validation round. An integral value (numeric or integer).
hbranches	the number of branches to take when removing a variable from the model. In Tikka and Hollmén (2008), the algorithm always removes the "weakest" variable (*hbranches* equals 1, also the default here). By using a value larger than 1, the algorithm creates branches in the search graph by removing each of the *hbranches* "weakest" variables, one at a time. The number of branches created is naturally limited by the number of variables remaining in the model at that point. See also *max.width*.
max.width	the maximum number of nodes with a given number of variables allowed in the search graph. The same limit is used for all search levels. An integral value (numeric or integer). See *pruning.criterion* and *pruning.keep.best*.
q	a numeric value between 0 and 0.5 (endpoints excluded) defining the quantiles 1-q and q. The difference of these sample quantiles is used as the width of the sampling distribution (a measure of uncertainty) of each coefficient in a linear model. The default value 0.165 is the same as used by Tikka and Hollmén (2008). In case of a normally distributed parameter, the width is approximately twice the standard deviation (one standard deviation on both sides of the mean).
standardize	a logical flag. If TRUE, standardizes the data to zero mean and unit variance. If FALSE, uses original data. This affects the scale of the results. If *use.ridge* is TRUE, this should be set to TRUE or the search graph and the sets of selected variables could be affected.
pruning.criterion	a character string. Options are "round robin", "random nodes", "random edges" and "greedy". Abbreviations are allowed. This affects how the search tree is pruned if the number of nodes to explore is about to exceed *max.width*. One of the following methods is used to select *max.width* nodes for the next level of search. If "round robin", the nodes of the current level (i variables) take turns selecting nodes for the next level (i-1 variables). The turns are taken in order of increasing validation error. Each parent node chooses children according to the order described in 'Details'. If a duplicate choice would be made, the turn is skipped. If "random nodes", random nodes are selected with uniform probability. If "random edges", random nodes are selected, with the probability of a node directly proportional to the number of edges leading to it. If "greedy", a method similar to "round robin" is used, but with the (virtual) looping order of parents and children swapped. Whereas the outer loop in "round robin" operates over children and the inner loop over parents, the outer

loop in "greedy" operates over parents and the inner loop over children. That is, a "greedy" parent node selects all its children before passing on the turn to the next parent.

pruning.keep.best
: a logical flag. If TRUE, the nodes that would also be present in the *hbranches* = 1 case are immune to pruning. If FALSE, the result may underperform the original Tikka and Hollmén (2008) solution in terms of (the lowest) validation error as function of the number of inputs.

pruning.reverse
: a logical flag. If TRUE, all the methods described in *pruning.criterion* except "random nodes" use reverse orders or inverse probabilities. The default is FALSE.

verbose
: a numeric or integer verbosity level from 0 (no output) to 5 (all possible diagnostics).

use.ridge
: a logical flag. If TRUE, the function uses ridge regression with automatic selection of the regularization (smoothing) parameter.

max.warn
: a numeric value giving the maximum number of warnings to store in the returned object. If more warnings are given, their total number is still recorded in the object.

sp
: a numeric value passed to magic if *use.ridge* is TRUE. Initial value of the regularization parameter. If negative (the default), initialization is automatic.

...
: additional arguments passed to magic if *use.ridge* is TRUE. It is an error to supply arguments named "S" or "off".

Details

When choosing which variable to drop from the model, the importance of a variable is measured by looking at two variables derived from the sampling distribution of its coefficient in the linear models of the repeated cross-validation runs:

1. absolute value of the median and
2. width of the distribution (see q).

The importance of an input variable is the ratio of the median to the width: *hbranches* variables with the smallest ratios are dropped, one variable in each branch. See *max.width* and *pruning.criterion*.

The main results of the function are described here. More details are available in 'Value'.

The function returns two sets of inputs variables:

L.v set corresponding to the smallest validation error.

L.f smallest set where validation error is close to the smallest error. The margin is the standard deviation of the training error measured in the node of the smallest validation error.

The mean of mean squared errors in the **tr**aining and **v**alidation sets are also returned (*E.tr*, *E.v*). For the training set, the standard deviation of MSEs (*s.tr*) is also returned. The length of these vectors is the number of variables in X. The i:th element in each of the vectors corresponds to the best model with i input variables, where goodness is measured by the mean MSE in the validation set.

Linear models fitted to the whole data set are also returned. Both ordinary least square regression (*lm.L.f*, *lm.L.v*, *lm.full*) and ridge regression models (*magic.L.f*, *magic.L.v*, *magic.full*) are computed, irrespective of the *use.ridge* setting. Both fitting methods are used for the *L.f* set of variables, the *L.v* set and the full set (all variables).

Value

A `list` with `class "sisal"`. The items are:

`L.f` a `numeric` vector containing indices to columns of X. See 'Details'.

`L.v` a `numeric` index vector like *L.f*. See 'Details'.

`E.tr` a `numeric` vector of length $d + 1$. See 'Details'.

`s.tr` a `numeric` vector of length $d + 1$. See 'Details'.

`E.v` a `numeric` vector of length $d + 1$. See 'Details'.

`L.f.nobranch` a `numeric` vector or `NULL`. Like *L.f* but for the "no branching" solution. `NULL` if branching is not used or if some elements of *branching.useful* are missing.

`L.v.nobranch` like *L.f.nobranch* but related to *L.v*.

`E.tr.nobranch` a `numeric` vector or `NULL`. Like *E.tr* but for the "no branching" solution. `NULL` when *branching.useful* is `NULL`. An element is missing when the corresponding element of *branching.useful* is missing.

`s.tr.nobranch` like *E.tr.nobranch* but related to *s.tr*.

`E.v.nobranch` like *E.tr.nobranch* but related to *E.v*.

`n.evaluated` a `numeric` vector of length $d + 1$. The number of nodes evaluated for each model size, indexed by the number of variables used plus one.

`edges` a `list` of directed edges between nodes in the search graph. There is an edge from node A to node B if and only if B was a candidate for a new node to be evaluated, resulting from removing one variable in A. The i:th element of the list contains edges directed away from the node represented by the i:th element of *vertices*. Each element is a list with one element, `"edges"`, which is a `numeric` vector of indices to *vertices*, pointing to the nodes towards which the edges are directed. There are no edges directed away from pruned nodes or nodes representing a single variable.

`vertices` a `character` vector the same size as *edges*. Contains the names of the nodes in the search graph. Each name contains the indices of the variables included in the set in question, separated by dots.

`vertices.logical`
a `logical matrix` containing an alternative representation of *vertices*. Number of rows is the length of *vertices* and number of columns is d. The i:th column indicates whether the i:th input variable is present in a given node. The row index and the index to *vertices* are equivalent.

`vertex.data` A `data.frame` with information about each node in the search graph (missing information means pruned node). The rows correspond to items in *vertices*. The columns are:

E.tr mean of MSEs, training.

s.tr standard deviation (n-1) of MSEs, training.

E.v mean of MSEs, validation.

E.v.level.rank rank of the node among all the evaluated (non-pruned) nodes with the same number of variables, in terms of validation error. Smallest error is rank 1.

n.rank.deficient number of rank deficient linear models. This problem arises when the number of input variables is large compared to the number of observations and *use.ridge* is FALSE.

n.NA.models number of models that could not be estimated due to lack of any samples

n.inputs number of input variables used in the model represented by the node.

min.branches the smallest branching factor large enough for producing the node. This is a number k between 1 and *hbranches*. The value for the root node (all input variables) is 1. The value for other nodes is the minimum of the set of values suggested by its parents. The value suggested by an individual parent is the *min.branches* value of the parent itself or the ranking of the child in terms of increasing importance of the removed variable (see 'Details'), whichever is larger. For example, when *pruning.keep.best* is TRUE, the *hbranches* = 1 search path can be followed by looking for nodes where *min.branches* is 1.

`var.names` names of the variables (column names of X).

`n` number of observations in the (X, y) data.

`d` number of variables (columns) in X.

`n.missing` number of samples where either y or all variables of X are missing.

`n.clean` number of complete samples in the data set X, y.

`lm.L.f` lm model fitted to $L.f$ variables.

`lm.L.v` lm model fitted to $L.v$ variables.

`lm.full` lm model fitted to all variables.

`magic.L.f` magic model fitted to $L.f$ variables.

`magic.L.v` magic model fitted to $L.v$ variables.

`magic.full` magic model fitted to all variables.

`mean.y` mean of y.

`sd.y` standard deviation (denominator n -1) of y.

`zeroRange.y` a logical value indicating whether all non-missing elements of y are equal, with some numeric tolerance.

`mean.X` column means of X.

`sd.X` standard deviation (denominator n -1) of each column in X.

`zeroRange.X` a logical vector. Like *zeroRange.y* but for each column of X.

`constant.X` a logical vector where the i:th value indicates whether the i:th column of X has a (nearly) constant, non-zero value (NA values allowed).

`params` a named list containing the values used for most of the parameter-like formal arguments of the function, and also anything in .... The names are the names of the parameters.

`pairwise.points`
: a `numeric` square `matrix` with d rows and columns. The count in row i, column j indicates the number of times that variable i was better than variable j. See 'Details' in `plotSelected.sisal`.

`pairwise.wins`
: a `logical` square `matrix` with d rows and columns. A `TRUE` value in row i, column j indicates that i is more important than variable j. Derived from *pairwise.points*.

`pairwise.preferences`
: a `numeric` vector with d elements. Number of wins minus number of losses (when another variable wins) per variable. Derived from *pairwise.wins*.

`pairwise.rank`
: an `integer` vector of ranks according to Copeland's pairwise aggregation method. Element number i is the rank of variable (column) number i in X. Derived from *pairwise.preferences*. See 'Details' in `plotSelected.sisal`.

`path.length`
: a `numeric` vector of path lengths. Consider a path starting from the full model and continuing through incrementally smaller models, each with the smallest validation error among the nodes with that number of variables. However, the path is broken at each point where the model with one less variable cannot be constructed by removing one variable from the bigger model (is not nested). The vector contains the lengths of the pieces. Its length is the number of breaks plus one.

`nested.path`
: a `numeric` vector containing the indices (column numbers) of the input variables in their removal order on the "nested path". The first element is the index of the variable that was removed first. The remaining variable is the last element. If the path does not exist, this is `NULL`. See 'Details' in `plotSelected.sisal`.

`nested.rank`
: an `integer` vector of ranks determined by *nested.path*. Element number i is the rank of variable (column) number i in X. `NULL` if *nested.path* is `NULL`. See 'Details' in `plotSelected.sisal`.

`branching.useful`
: If branching is enabled (*hbranches* > 1), this is a `logical` vector of length d. If the i:th element is `TRUE`, branching improved the best model with i variables in terms of validation error. The result is `NA` if a comparison is not possible (may happen if *pruning.keep.best* is `FALSE`). If branching is not used, this is `NULL`.

`warnings`
: warnings stored. A `list` of objects that evaluate to a `character` string.

`n.warn`
: number of warnings produced. May be higher than number of warnings stored.

Author(s)

Mikko Korpela

References

Tikka, J. and Hollmén, J. (2008) Sequential input selection algorithm for long-term prediction of time series. *Neurocomputing*, 71(13–15):2604–2615.

See Also

See magic for information about the algorithm used for estimating the regularization parameter and the corresponding linear model when *use.magic* is TRUE.

See summary.sisal for how to extract information from the returned object.

Examples

```
library(stats)
set.seed(123)
X <- cbind(sine=sin((1:100)/5),
           linear=seq(from=-1, to=1, length.out=100),
           matrix(rnorm(800), 100, 8,
                  dimnames=list(NULL, paste("random", 1:8, sep="."))))
y <- drop(X %*% c(3, 10, 1, rep(0, 7)) + rnorm(100))
foo <- sisal(X, y, Mtimes=10, kfold=5)
print(foo)           # selected inputs "L.v" are same as
summary(foo$lm.full) # significant coefficients of full model
```

---

sisalData *Download External Datasets for* SISAL

Description

Loads external datasets for testing with SISAL. Choices are laser generated data and Poland electricity load data.

Usage

```
sisalData(dataset = c("poland", "laser", "laser.cont"), verify = TRUE)
```

Arguments

dataset	A character string: "poland" (default), "laser" or "laser.cont" (see 'Note').
verify	A logical flag. If TRUE, verifies the integrity of the downloaded data by computing a checksum and comparing it to a pre-computed value.

Details

The laser generated data come in two parts, "laser" and "laser.cont". The Poland electricity load data is also divided in two parts, but they are both returned with dataset="poland".

This function requires an Internet connection. The download may fail due to a problem such as the remote server being unavailable.

Value

With option *dataset*="laser", returns an integer vector of length 1000.

With option *dataset*="laser.cont", returns an integer vector of length 9093.

With option *dataset*="poland", returns a list with two numeric vectors:

learn	1400 values
test	201 values

Note

Checked on 2020-02-14, the Santa Fe datasets are no longer available at their previous location. Attempting to download them with this function will result in an error.

Author(s)

Mikko Korpela

References

The Santa Fe Time Series Competition Data / Data Set A: Laser generated data. Availability unknown (2020-02-14).

Environmental and Industrial Machine Learning Group / Datasets / Poland Electricity Load. http://research.ics.aalto.fi/eiml/datasets.shtml. URL accessed on 2020-02-14.

See Also

testSisal

Examples

```
## Not run:
foo <- sisalData("poland")
length(foo$learn) # 1400
length(foo$test)  # 201
## End(Not run)
```

---

sisalTable *Draw Table with Equally Sized Cells*

Description

Draws a resizable or fixed-size table with equally sized cells. Main title, axis (tick) labels and axis titles (left, bottom) are optional. Cells can have individual background and text colors and stripes.

Usage

```
sisalTable(labels = matrix(seq_len(12), 3, 4),
           nRows = NROW(labels), nCols = NCOL(labels),
           bg = sample(colors(), nRows * nCols, replace = TRUE),
           stripeCol = NULL, fg = NULL, naFill = "white",
           naStripes = "grey50", main = NULL, xlab = NULL,
           ylab = NULL, xAxisLabels = NULL, yAxisLabels = NULL,
           draw = TRUE, outerRect = TRUE, innerLines = TRUE,
           nStripes = 7, stripeRot = 45, stripeWidth = 0.2,
           stripeScale = 0.95, resizeText = TRUE,
           resizeTable = TRUE, resizeMain = resizeText,
           resizeLab = resizeText, resizeAxes = resizeText,
           resizeLabels = resizeTable && resizeText,
           x = unit(0.5, "npc"), y = unit(0.5, "npc"),
           width = unit(0.97, "npc"), height = unit(0.97, "npc"),
           default.units = "npc", just = "center",
           clip = "inherit", xAxisRot = 0, yAxisRot = 0,
           xAxisJust = c(0.5, 1), xAxisX = 0.5, xAxisY = 1,
           yAxisJust = c(1, 0.5), yAxisX = 1, yAxisY = 0.5,
           mainMargin = if (resizeMain) 0.15 else unit(8, "points"),
           xlabMargin = if (resizeLab) 0.1 else unit(5, "points"),
           ylabMargin = if (resizeLab) 0.1 else unit(5, "points"),
           axesMargin = if (resizeAxes) 0.1 else unit(5, "points"),
           axesSize = 0.8, forceAxesSize = FALSE,
           mainSize = 1, xlabSize = 1, ylabSize = 1,
           mainPar = gpar(fontface = "bold", fontsize = 14),
           labPar = gpar(fontface = "plain", fontsize = 14),
           labelPars = gpar(fontsize = 20, cex = 0.6),
           axesPar = gpar(fontsize = 10),
           rectPar = gpar(), linePar = gpar(),
           name = NULL, gp = NULL, vp = NULL)
```

Arguments

labels	the labels to use in the table cells. A list or an atomic vector containing something that can be displayed as text, e.g. character values. One element is used for each cell. If the object has a "dim" attribute (matrix, array), it is used for determining the number of rows and columns in the table. NA means no text.
nRows	the number of rows in the table. A positive integral number.
nCols	the number of columns in the table. A positive integral number.
bg	the background colors of the table cells. One element is used for each cell.
stripeCol	an optional vector of colors. If used, indicates the color of stripes to be painted on top of the background color in each table cell. One element is used for each table cell. NA means no stripes.
fg	the text colors of the table cells. One element is used for each cell. If NULL (the default), black or white text is used so that the contrast between foreground and

background is maximized.

naFill — background color to use when the label of a table cell is NA. This is a single color value.

naStripes — table cells with an NA label are indicated with stripes. This is the color of the stripes, a single color value. The stripes can be hidden by using a value identical with that of *naFill*.

main — the main title of the plot.

xlab — a title for the x axis.

ylab — a title for the y axis.

xAxisLabels — a label for each column of the table.

yAxisLabels — a label for each row of the table.

draw — a logical flag indicating whether to draw the table. If FALSE, no drawing is done.

outerRect — a logical flag indicating whether a rectangle will be drawn around the table.

innerLines — a logical flag indicating whether line segments will be drawn between the table cells.

nStripes — a positive integral number giving the number of stripes to be drawn in table cells. Only applies to those cells where stripes are used, i.e. when the relevant element of *label* is NA or *stripeCol* is not NA. The stripes are spaced evenly. Defaults to 7.

stripeRot — an integral number giving the rotation angle (degrees, counterclockwise) of the stripes used in table cells. Defaults to 45 which means diagonal stripes parallel to a line segment between the lower left corner and the upper right corner of the cell. Value 0 means horizontal and 90 vertical stripes.

stripeWidth — a numerical value giving the width of the stripes used in cells as a proportion of the available width. Values between 0 and 1 are allowed, excluding the endpoints. Defaults to 0.2.

stripeScale — a numerical value indicating the proportion of the area of a table cell to be used for the stripe pattern. The pattern is always centered, and the possible empty space is left on the borders of the cell. Values between 0 and 1 are allowed, including the endpoints. Defaults to 0.95.

resizeText — a logical flag indicating whether to use dynamic text size. This is only used as the default value of *resizeMain*, *resizeLab*, *resizeLabels* and *resizeAxes*. Defaults to TRUE.

resizeTable — a logical flag indicating whether the size of the table will depend on the size of the main viewport, which itself may be static or depend on the size of the graphical device. Defaults to TRUE. See 'Details'.

resizeMain — a logical flag indicating whether the main title will be resizable.

resizeLab — a logical flag indicating whether the the x axis and y axis titles will be resizable.

resizeLabels — a logical flag indicating whether the labels used in the table cells will be resizable.

resizeAxes	a logical flag indicating whether the row and column labels will be resizable.
x	a numeric vector or unit object of length one specifying the x location of the graphical object.
y	a numeric vector or unit object of length one specifying the y location of the graphical object.
width	a numeric vector or unit object of length one specifying the width of the graphical object. See 'Details'.
height	a numeric vector or unit object of length one specifying the height of the graphical object. See 'Details'.
default.units	a character string indicating the unit to use for numeric values of *x*, *y*, *width* and *height*.
just	a character or numeric vector of one or two values specifying the justification of the graphical object relative to its (x, y) location. See viewport.
clip	a character string specifying what to do if the graphical object overflows the viewport reserved for it. See 'Details'.
xAxisRot	a numeric value giving the rotation angle of the column labels in degrees.
yAxisRot	a numeric value giving the rotation angle of the row labels in degrees.
xAxisJust	justification setting for column labels. A numeric or character vector. Rotation (if any) will be done *before* justification. See *just* in textGrob for possible values.
xAxisX	x location of column labels relative to the space allocated for them. A numeric value where 0 means left and 1 right.
xAxisY	y location of column labels relative to the space allocated for them. A numeric value where 0 means bottom and 1 top.
yAxisJust	justification setting for row labels. A numeric or character vector. See *xAxisJust*.
yAxisX	x location of row labels relative to the space allocated for them. A numeric value where 0 means left and 1 right.
yAxisY	y location of row labels relative to the space allocated for them. A numeric value where 0 means bottom and 1 top.
mainMargin	size of the margin between the main title and the table.
xlabMargin	size of the margin between the x axis title and the next graphical object towards the table.
ylabMargin	size of the margin between the y axis title and the next graphical object towards the table.
axesMargin	size of the margin between the row or column labels and the table.
axesSize	a positive numeric value specifying the desired ratio of fontsize in row and column labels to fontsize in table cells.
forceAxesSize	a logical flag. If TRUE, the function will reduce the size of text in table cells if it is necessary in order to achieve the desired *axesSize*.
mainSize	scale factor for fontsize of main title. A positive numeric value. Only effective when *resizeMain* is TRUE.

xlabSize	scale factor for fontsize of x axis title. A positive numeric value. Only effective when *resizeLab* is TRUE.
ylabSize	scale factor for fontsize of y axis title. A positive numeric value. Only effective when *resizeLab* is TRUE.
mainPar	graphical parameters for the main title.
labPar	graphical parameters for x and y axis titles.
labelPars	graphical parameters for labels used in table cells. Can also be a list, one element for each table cell, recycled if necessary.
axesPar	graphical parameters for row and column labels.
rectPar	graphical parameters for the rectangle around the table.
linePar	graphical parameters for the line segments between table cells.
name	a character string identifier for the graphical object returned by the function. If NULL (the default), a name will be assigned automatically.
gp	graphical parameters for the whole object.
vp	a "viewport" object, the name of a viewport object, a vpPath object pointing to a viewport or NULL (the default). If not NULL, this graphical object will be drawn in the given viewport. The name or the path must point to a descendant of the current viewport. See current.vpPath, current.vpTree, downViewport and grid.draw.

Details

This function was written to be used with plotSelected but it should be generic enough to be useful for other purposes, too.

The color and text vectors (including matrices and arrays) pointing to table cells (*labels*, *bg*, *stripeCol*, *fg*) are interpreted in column-major order, like linear indexing of a matrix. Each data.frame argument is collapsed to a list by combining its columns. Finally, values are recycled if needed, also in *xAxisLabels* and *yAxisLabels*.

For possible color values, see col2rgb.

In the various text objects, mathematical annotation (see plotmath) is supported in addition to character values.

For information on setting graphical parameters (*gp*, *mainPar*, *labPar*, ...), see gpar.

The graphical object returned is a gTree which contains a gList of graphical objects and a vpTree of viewports. The child viewports are placed inside the parent using a grid.layout. The size of the whole object is the size of the parent viewport. It will be fixed or depend on the space available to it:

- If all graphical elements are non-resizable (but *resizeLabels* can be TRUE), a suitable fixed size will be computed.
- Otherwise, the size is determined by *width* and *height*. However, if there are non-resizable elements, the graphical object may be larger than that.

The graphical object will not use any excess space. In other words, the width and height reported by grobWidth and grobHeight are tight. It is possible that some parts of the plot may overflow their assigned space and the bounds computed for the whole graphical object. Examples include using large fixed-size text elements or large values of the gpar graphical parameter "cex". Clipping can be adjusted through *clip*.

If *resizeAxes* is TRUE, *axesMargin* must be a non-negative numeric value giving the size of the margin as a proportion of the side length of a table cell. If *resizeAxes* is FALSE, *axesMargin* can also be a unit object. The arguments *mainMargin* and *labMargin* are analogous to *axesMargin*.

Value

The function is usually called for the side effect (a plot is drawn), but it also returns a grob representation of the plot. The returned object is a custom gTree of class "sisalTable".

Author(s)

Mikko Korpela

Examples

```
library(grDevices)
library(grid)
## Default: 3 by 4 table with labels 1:12 and random background colors
grid.newpage()
sisalTable()

## Four examples in a grid layout
rowCol <- c(1, 18, 2, 18, 1)
lo <- grid.layout(nrow = 5, ncol = 5,
                  widths = rowCol, heights = rowCol)
grid.newpage()
pushViewport(viewport(layout = lo, name = "bgLayout"))
grid.rect(gp=gpar(fill="grey75", col="grey75"))

rNames <- c("topmargin", "top", "hspace", "bottom", "bottommargin")
cNames <- c("leftmargin", "left", "vspace", "right", "rightmargin")
for (Row in c(2, 4)) {
    for (Col in c(2, 4)) {
        pushViewport(viewport(layout.pos.row = Row,
                              layout.pos.col = Col,
                              name = paste(rNames[Row],
                                           cNames[Col], sep="")))
        grid.rect(gp=gpar(fill="cadetblue"))
        upViewport(1)
    }
}

colors1Vec <- terrain.colors(12)
colors1Mat <- matrix(colors1Vec, 3, 4)
labels1Vec <- sample(c(letters, LETTERS), 12)
labels1Mat <- matrix(labels1Vec, 3, 4)
```

```
## Column vector, aligned with the right side of the viewport
longText <- rep("", 12)
longText[3] <- "a longish piece of text"
longText[9] <- "and some more"
sisalTable(labels1Vec, bg = colors1Vec, vp = "topleft",
           x = 1, just = "right",
           yAxisLabels = longText, xAxisLabels = "Boo")

## Matrix, zero margin
downViewport("topright")
sisalTable(labels1Mat, bg = colors1Mat,
           width = 1, height = 1, name = "trPlot",
           xAxisLabels = 1:4, yAxisLabels = LETTERS[1:3])
grid.rect(width = grobWidth("trPlot"), height = grobHeight("trPlot"),
          gp = gpar(lty="dashed", col = "white", lwd = 2))
upViewport(1)

## Transpose of matrix, width and height 0.75 "npc" units
downViewport("bottomleft")
sisalTable(t(labels1Mat), bg = t(colors1Mat),
           width = 0.75, height = 0.75, name = "blPlot",
           yAxisLabels = 1:4, xAxisLabels = LETTERS[1:3])
grid.rect(width = grobWidth("blPlot"), height = grobHeight("blPlot"),
          gp = gpar(lty="dashed", col = "white", lwd = 2))
upViewport(1)

## ?plotmath, some cells with no background color
labels2 <- expression(x^{y+x}, sqrt(x), bolditalic(x), NA)
bgCol <- c(rep("white", 3), NA)
sisalTable(labels2, nRows=3, nCols=5, bg = bgCol, naFill = NA,
           naStripes = "darkmagenta", vp="bottomright",
           main = "plotmath text")
```

---

summary.sisal *Summarizing Sequential Input Selection Results*

Description

summary method for class "sisal"

Usage

```
## S3 method for class 'sisal'
summary(object, ...)
## S3 method for class 'summary.sisal'
print(x, ...)
```

Arguments

`object`	an object of class `"sisal"`.
`x`	an object of class `"summary.sisal"`.
`...`	arguments passed to/from other methods.

Details

The functions compute and print summaries (`summary.lm`) of the ordinary least squares regression models stored in the *object* and some additional information.

Value

The function `summary.sisal` returns a `list` with `class "summary.sisal"`, currently containing:

`summ.full`	summary of the full model. An object of class `"summary.lm"`.
`summ.L.v`	summary of the *L.v* model. An object of class `"summary.lm"`.
`summ.L.f`	summary of the *L.f* model. An object of class `"summary.lm"`.
`error.df`	a `data.frame` containing information on the best variable sets with a given number of variables, with the following columns (copied from *object*):

n.inputs number of inputs (row label).

E.tr mean training MSE.

s.tr standard deviation of training MSE.

E.v mean validation MSE.

L.f.flag `logical vector` where the location of `TRUE` points the smallest variable set with *thr.flag* `TRUE`.

L.v.flag `logical vector` where the location of `TRUE` points the variable set with the smallest validation error.

thr.flag `logical vector` where `TRUE` means that error is at most *E.v*`[`*L.v.flag*`]` + *s.tr*`[`*L.v.flag*`]`.

The function `print.summary.sisal` invisibly returns *x*.

Author(s)

Mikko Korpela

See Also

`sisal`, `print.sisal`

Examples

```
foo <- testSisal(dataset="toy", Mtimes=10, hbranches=2)
summary(foo)
```

testSisal *Testing the Sequential Input Selection Algorithm*

Description

Tests sisal with example datasets or time series data. The function uses the training part of an example dataset or user-supplied numeric data interpreted as a time series.

Usage

```
testSisal(dataset = c("tsToy", "laser", "poland", "toy"), nData = Inf,
          FUN = "sisal", lags = NULL, stepsAhead = 1,
          noiseSd = 0.2, verbose = 1, ...)
```

Arguments

dataset	the dataset to use. A numeric vector containing time series data or one of "tsToy" (the default), "laser", "poland" and "toy".
nData	a numeric value containing the number of observations to use. If larger than the number of observations in the dataset, all of the data will be used (the default).
FUN	which function to call. By default, acts as a front end to sisal. This can be any function that accepts arguments named "X", "y" and "verbose". See match.fun for legal values.
lags	a numeric or integer vector. When using time series data (*dataset* is numeric, "laser", "poland" or "tsToy"), the function creates lagged versions of the time series to be used as input variables in sisal. The lags are specified here. These are non-negative integral values where 0 means the latest observation, 1 is the previous observation etc. The default values for "laser", "poland" and "tsToy" are 0:19, 0:14 and 0:9, respectively.
stepsAhead	an integral value specifying how many steps ahead to predict in a time series setting. The default is 1.
noiseSd	standard deviation of noise to be used with the "toy" *dataset*. The base noise is always the same (stored with the dataset) and only scaled to match this setting.
verbose	a numeric or integer verbosity level. This function only has two verbosity levels (0 and larger than 0), but the value is also propagated to *FUN*.
...	arguments passed to *FUN*.

Details

The function recognizes if a numeric *dataset* is the "laser" or "poland" dataset. In case repeated experiments will be performed on those datasets, it is best to explicitly fetch them with sisalData before using this function. Doing so reduces the amount of network traffic and makes offline work possible.

Value

The value returned by function *FUN*, when called with the given *dataset* (processed by this function) and parameters. See the help page of the relevant function, e.g. sisal.

Author(s)

Mikko Korpela

See Also

See sisalData, toy.learn and tsToy.learn for documentation on the datasets.

The performance of the models returned by this functions can be evaluated using bootMSE, which uses a separate test part of the dataset.

Examples

```
foo <- testSisal(dataset="toy", hbranches=2, max.width=2, Mtimes=5,
                 use.ridge=TRUE)
print(foo)
names(foo)
```

---

toy.learn *Toy Data for* SISAL *(Learning Set)*

Description

Numeric matrix with independent and dependent variables and noise

Usage

```
toy.learn
```

Format

The format is:

```
num [1:1000, 1:12] -0.62067 1.36985 0.00122 0.75527 -1.82271 ...
- attr(*, "dimnames")=List of 2
 ..$ : NULL
 ..$ : chr [1:12] "y" "noise" "X1" "X2" ...
```

Details

This is the learning set of the toy data, i.e. 1000 rows of the whole 1500 row dataset.

Columns `"X1"`, `"X2"`, ..., `"X10"` were generated with `rnorm` to follow a standard normal distribution.

Column `"y"` is a linear combination of `"X1"`, `"X2"`, `"X3"`, coefficients `(1:3)/sqrt(sum((1:3)^2))`, yielding a theoretical standard normal distribution.

Column `"noise"` was also generated from the standard normal distribution.

Use `file.show(system.file("toyDataSrc","sisalToy.R",package="sisal"))` to view the script that generated the data.

See Also

`toy.test`, `testSisal`

Examples

```
library(graphics)
plot(as.data.frame(toy.learn))
```

---

toy.test *Toy Data for* SISAL *(Test Set)*

Description

Numeric matrix with independent and dependent variables and noise

Usage

```
toy.test
```

Format

The format is:

```
 num [1:500, 1:12] -0.543 -0.881 0.115 0.461 -0.173 ...
 - attr(*, "dimnames")=List of 2
  ..$ : NULL
  ..$ : chr [1:12] "y" "noise" "X1" "X2" ...
```

Details

This is the test set of the toy data, i.e. 500 rows of the whole 1500 row dataset.

For other details, see `toy.learn`.

See Also

`toy.learn`, `bootMSE`

Examples

```
library(graphics)
plot(as.data.frame(toy.test))
```

`tsToy.learn` *Toy Time Series Data for* SISAL *(Learning Set)*

Description

Numeric vector with autoregressive (AR) time series data

Usage

```
tsToy.learn
```

Format

The format is:

```
 num [1:1000] 0.7529 -0.2576 0.441 0.8473 0.0164 ...
```

Details

This is the learning set of the toy time series data, i.e. the first 1000 of the total 3000 observations.

The data follow a second order AR model. The first order coefficient is `-0.5` and the second order coefficient `0.3`. The autocovariances for lags `0` to `4` are `c(1.0,-0.71,0.66,-0.54,0.47)` (theoretical values, two significant digits).

Use `file.show(system.file("toyDataSrc","sisalToyTs.R",package="sisal"))` to view the script that generated the data.

See Also

`tsToy.test`, `testSisal`

Examples

```
library(graphics)
library(stats)
plot(tsToy.learn)
acf(tsToy.learn)
```

tsToy.test *Toy Time Series Data for* SISAL *(Test Set)*

Description

Numeric vector with autoregressive (AR) time series data

Usage

```
tsToy.test
```

Format

The format is:

```
 num [1:2000] 0.583 -0.71 -1.172 1.067 -0.719 ...
```

Details

This is the test set of the toy time series data, i.e. the last 2000 of the total 3000 observations.

The data follow a second order AR model. The first order coefficient is `-0.5` and the second order coefficient `0.3`.

Use `file.show(system.file("toyDataSrc","sisalToyTs.R",package="sisal"))` to view the script that generated the data.

See Also

`tsToy.learn`, `bootMSE`

Examples

```
library(graphics)
library(stats)
plot(tsToy.test)
acf(tsToy.test, type="partial")
```

Index