Package 'SPOTMisc'

August 5, 2021

Type Package

Title Misc Extensions for the 'SPOT' Package

Version 1.2.11

Maintainer Thomas Bartz-Beielstein <tbb@bartzundbartz.de>

Description Implements additional models, simulation tools, and interfaces as extensions to 'SPOT'. It provides tools for hyperparameter tuning via 'keras/tensorflow', interfacing 'mlr', for performing Markov chain simulations,
and for sensitivity analysis based on sequential bifurcation methods as described in Bettonvil and Kleijnen (1996).
Furthermore, additional plotting functions for output from 'SPOT' runs are implemented.
Bartz-Beielstein T, Lasarczyk C W G, Preuss M (2005) <doi:10.1109/CEC.2005.1554761>.
Bartz-Beielstein T, Zaefferer M, Rehbach F (2021) <arXiv:1712.04076>.
Bartz-Beielstein T, Rehbach F, Sen A, Zaefferer M <arXiv:2105.14625>.
Bettonvil, B, Kleijnen JPC (1996) <doi:10.1016/S0377-2217(96)00156-7>.

License GPL (>= 2)

Encoding UTF-8

LazyLoad yes

LazyData true

Date 2021-08-04

Depends R (>= 4.0.0)

RoxygenNote 7.1.1

Imports benchmarkme, callr, emoa, ggsci, graphics, grDevices, jsonlite, keras, mlr, plotly, RColorBrewer, rpart.plot, sensitivity, SimInf, smoof, SPOT, stats, utils

Suggests babsim.hospital, knitr, rmarkdown, testthat

URL https://www.spotseven.de

NeedsCompilation no

Author Thomas Bartz-Beielstein [aut, cre] (<https://orcid.org/0000-0002-5938-5158>), Martin Zaefferer [aut] (<https://orcid.org/0000-0003-2372-2092>), Frederik Rehbach [aut] (<https://orcid.org/0000-0003-0922-8629>)

Repository CRAN

Date/Publication 2021-08-05 01:40:02 UTC

R topics documented:

evalKerasMnist *evalKerasMnist*

Description

Hyperparameter Tuning: Keras MNIST Classification Test Function.

Usage

```
evalKerasMnist(x, kerasConf)
```

Arguments

x	matrix of hyperparameter values to evaluate with the function. Rows for points and columns for dimension.
kerasConf	List of additional parameters passed to keras, e.g., `verbose`: Verbosity mode (0 = silent, 1 = progress bar, 2 = one line per epoch), `callbacks`: List of callbacks to be called during training. `validation_split`: Float between 0 and 1. Fraction of the training data to be used as validation data. The model will set apart this fraction of the training data, will not train on it, and will evaluate the loss and any model metrics on this data at the end of each epoch. The validation data is selected from the last samples in the x and y data provided, before shuffling. `validation_data`: Data on which to evaluate the loss and any model metrics at the end of each epoch. The model will not be trained on this data. This could be a list (x_val, y_val) or a list (x_val, y_val, val_sample_weights). validation_data will override validation_split. `shuffle`: Logical (whether to shuffle the training data before each epoch) or string (for "batch"). "batch" is a special option for dealing with the limitations of HDF5 data; it shuffles in batch-sized chunks. Has no effect when steps_per_epoch is not NULL.

Details

Trains a simple deep NN on the MNIST dataset. Standard Code from https://keras.rstudio.com/ Modified by T. Bartz-Beielstein (tbb@bartzundbartz.de)

Value

list with function values (loss, accuracy, and keras model information)

See Also

`getKerasConf`

`funKerasMnist`

Examples

```
### These examples require an activated Python environment as described in
### Bartz-Beielstein, T., Rehbach, F., Sen, A., and Zaefferer, M.:
### Surrogate Model Based Hyperparameter Tuning for Deep Learning with SPOT,
### June 2021. http://arxiv.org/abs/2105.14625.
PYTHON_RETICULATE = FALSE
if(PYTHON_RETICULATE){

library("SPOTMisc")
kerasConf <- getKerasConf()
kerasConf$verbose <- 1
lower <- c(1e-6, 1e-6, 16,0.6, 1e-9, 10, 6,0.4,0.99,1,1e-8)
upper <- c(0.5, 0.5, 512, 1.5, 1e-2, 50, 10,0.999,0.999,10,6e-8)
types <- c("numeric",  "numeric",  "integer",  "numeric",  "numeric",
           "integer",  "integer",  "numeric",  "numeric",  "integer",
           "numeric")

x <- matrix(lower, 1,)
res <- evalKerasMnist(x, kerasConf)
str(res)
### The number of units for all layers can be listed as follows:
res$modelConf$config$layers[,2]$units
}
```

`evalMarkovChain` *evalMarkovChain*

Description

Evaluation function for the optimization of continuous time Markov chains models using `SIR` models.

Usage

```
evalMarkovChain(x, conf)
```

Arguments

`x`	vector of parameter values, i.e., parameters of the MarkovChain model to evaluate with the function. `p` num [0;1] proportion of confirmed cases `beta` num Transmission rate from susceptible to infected. See `SIR`. `gamma` num Recovery rate from infected to recovered. See `SIR`. `CFR` num Case Fatalities Rate
`conf`	a list with entries

`regionData` A data frame with observations of 3 variables:

- `date` Date, format: "2020-01-22" "2020-01-23" "2020-01-24" "2020-01-25" ...
- `confirmed` num 0 0 0 0 0 0 0 0 0 0 ..
- `fatalities` fatalities: num 0 0 0 0 0 0 0 0 0 0 ...

`N` N population size

Details

Performs a SIR model simulation for one specific parameter setting using the `modelMarkovChain` function and evaluates the result from the simulation model output with the real data. The RMSE is used as the performance metric.

Value

value (log RMSE)

Examples

```
require("SimInf")
data <- preprocessInputData(regionTrain, regionPopulation)
set.seed(123)
data <- data[[1]]
N <- attr(data, "regionPopulation")
# params:  x = (p, beta, gamma, CFR)
x <- c(0.01, 0.01, 0.1, 0.01)
# Simulate only 2 days
conf <- list(regionData = data[1:2, ], N = N)
evalMarkovChain(x = x, conf=conf)
```

`funBBOBCall` *funBBOBCall*

Description

Call (external) BBOB Function. Call the generator `makeBBOBFunction` for the noiseless function set of the real-parameter Black-Box Optimization Benchmarking (BBOB).

Usage

```
funBBOBCall(x, opt = list(), ...)
```

Arguments

x	matrix of points to evaluate with the function. Rows for points and columns for dimension.
opt	list with the following entries
	dimensions [integer(1)] Problem dimension. Integer value between 2 and 40.
	fid [integer(1)] Function identifier. Integer value between 1 and 24.
	iid [integer(1)] Instance identifier. Integer value greater than or equal 1.
...	further arguments

Value

1-column matrix with resulting function values

Examples

```
## Call the first instance of the 2D Sphere function
require("smoof")
require("SPOT")
set.seed(123)
x <- matrix(c(1,2),1,2)
funBBOBCall(x, opt = list(dimensions = 2L, fid = 1L, iid =1L))
spot(x=NULL, funBBOBCall,
       lower = c(-2,-3), upper = c(1,2),
       control=list(funEvals=15),
       opt = list(dimensions = 2L, fid = 1L, iid = 1L ))
```

---

funKerasMnist *funKerasMnist*

Description

Hyperparameter Tuning: Keras MNIST Classification Test Function.

Usage

```
funKerasMnist(x, kConf)
```

Arguments

x	matrix of hyperparameter values to evaluate with the function. Rows for points and columns for dimension.

kConf — List of additional parameters passed to keras, e.g., `verbose`: Verbosity mode (0 = silent, 1 = progress bar, 2 = one line per epoch), `callbacks`: List of callbacks to be called during training. `validation_split`: Float between 0 and 1. Fraction of the training data to be used as validation data. The model will set apart this fraction of the training data, will not train on it, and will evaluate the loss and any model metrics on this data at the end of each epoch. The validation data is selected from the last samples in the x and y data provided, before shuffling. `validation_data`: Data on which to evaluate the loss and any model metrics at the end of each epoch. The model will not be trained on this data. This could be a list (x_val, y_val) or a list (x_val, y_val, val_sample_weights). validation_data will override validation_split. `shuffle`: Logical (whether to shuffle the training data before each epoch) or string (for "batch"). "batch" is a special option for dealing with the limitations of HDF5 data; it shuffles in batch-sized chunks. Has no effect when steps_per_epoch is not NULL.

Details

Trains a simple deep NN on the MNIST dataset. Standard Code from https://keras.rstudio.com/ Modified by T. Bartz-Beielstein (tbb@bartzundbartz.de)

Value

1-column matrix with resulting function values (test loss)

See Also

getKerasConf

Examples

```
### These examples require an activated Python environment as described in
### Bartz-Beielstein, T., Rehbach, F., Sen, A., and Zaefferer, M.:
### Surrogate Model Based Hyperparameter Tuning for Deep Learning with SPOT,
### June 2021. http://arxiv.org/abs/2105.14625.
PYTHON_RETICULATE = FALSE
if(PYTHON_RETICULATE){

library("SPOTMisc")
library("SPOT")
kerasConf <- getKerasConf()
lower <- c(1e-6, 1e-6, 16,0.6, 1e-9, 10, 6,0.4,0.99,1,1e-8)
upper <- c(0.5, 0.5, 512, 1.5, 1e-2, 50, 10,0.999,0.999,10,6e-8)
types <- c("numeric",  "numeric",  "integer",  "numeric",  "numeric",
           "integer",  "integer",  "numeric",  "numeric",  "integer",
           "numeric")

### First example: simple function call:
x <- matrix(lower, 1,)
funKerasMnist(x, kConf = kerasConf)
```

```
### Second example: evaluation of several (three) hyperparameter settings:
xxx <- rbind(x,x,x)
funKerasMnist(xxx, kConf = kerasConf)

### Third example: spot call with extended verbosity:
kerasConf$verbose <- 1
res <- spot(x = NULL,
            fun = funKerasMnist,
            lower = lower,
            upper = upper,
            control = list(funEvals=50,
                        noise = TRUE,
                        types = types,
                        plots = TRUE,
                        progress = TRUE,
                        seedFun = 1,
                        seedSPOT = 1),
                        kConf = kerasConf)
  }
```

---

funMarkovChain *funMarkovChain*

Description

Wrapper function for evalMarkovChain used by spot.

Usage

```
funMarkovChain(x, conf)
```

Arguments

x	vector of parameter values, i.e., parameters of the MarkovChain model to evaluate with the function.
	p num [0;1] proportion of confirmed cases
	beta num Transmission rate from susceptible to infected. See SIR.
	gamma num Recovery rate from infected to recovered. See SIR.
	CFR num Case Fatalities Rate
conf	a list with entries
	regionData A data frame with observations of 3 variables:
	date Date, format: "2020-01-22" "2020-01-23" "2020-01-24" "2020-01-25" ...
	confirmed num 0 0 0 0 0 0 0 0 0 0 ..
	fatalities fatalities: num 0 0 0 0 0 0 0 0 0 0 ...
	N N population size

Details

Optimization of Continuous Time Markov Chains (MarkovChain) models.

Value

1-column matrix with resulting function values (RMSE)

Examples

```
data <- preprocessInputData(regionTrain, regionPopulation)
set.seed(123)
data <- data[[1]]
N <- attr(data, "regionPopulation")
# params:  x = (p, beta, gamma, CFR)
x <- matrix(c(0.01, 0.1, 0.01, 0.1),1,4)
conf <- list(regionData = data, N = N)
funMarkovChain(x, conf)
```

`generateMCPrediction` *generateMCPrediction*

Description

Predict results on test data using the tuned MarkovChain mode. Result has the required data format for a submission to the Kaggle COVID-19 challenge.

Usage

```
generateMCPrediction(
  testData,
  models,
  startSimulation = "2020-01-22",
  write = FALSE
)
```

Arguments

`testData`	'data.frame': obs. of 3 variables:
	`ForecastId` int 1 2 3 4 5 6 7 8 9 10 ...
	`Region` Factor w/ 313 levels "Afghanistan/",..: 1 1 1 1 1 1 1 1 1 1 ...
	`Date` Date, format: "2020-04-02" "2020-04-03" "2020-04-04" "2020-04-05" ...
`models`	'data.frame': obs. of 7 variables:
	`p` num [0;1] proportion of confirmed cases
	`beta` num 13.8 13.8 16.3 11.5 29.2 ...

gamma num 13.8 13.8 16.3 11.5 29.2 ...

CFR num 0.14 0.14 0.2319 0.0312 0.0705 ...

cost num 658 256 1207 1091 300 ...

region chr, e.g., "Afghanistan/" "Albania/" "Algeria/" "Andorra/" ...

`startSimulation`
: chr start of the simulation period, e.g., "2020-01-22". startSimulation must be at or before the Date from testData. Simulations can start earlier, because some use R=0. This enables a warm-up period.

`write`
: logical. Default `FALSE`. If `TRUE`, results are written to the file `submit.csv`.

Details

Output from `parseTunedRegionModel` is processed.

Value

returns data.frame with obs. of the following 3 variables:

`ForecastId` int: Forecast Id taken from the `regionTest` data set.

`ConfirmedCases` num: Cumulative number of confirmed cases.

`Fatalities` num: Cumulative number of fatalities.

Examples

```
require("SimInf")
require("SPOT")
data <- preprocessInputData(regionTrain, regionPopulation)
testData <- preprocessTestData(regionTest)
## Select the first region:
testData <- testData[testData$Region==levels(testData$Region)[1], ]
testData$Region <- droplevels(testData$Region)
## Very small number of function evaluations:
n <- 6
res <- lapply(data[1], tuneRegionModel, pops=NULL,
control=list(funEvals=n, designControl=list(size=5), model = buildLM))
parsedList <- parseTunedRegionModel(res)
pred <- generateMCPrediction(testData = testData, models = parsedList$models, write = FALSE)
```

---

getKerasConf *Get keras configuration parameter list*

Description

Configuration list for Keras

Usage

```
getKerasConf()
```

Details

Additional parameters passed to keras, e.g., `verbose`: Verbosity mode (0 = silent, 1 = progress bar, 2 = one line per epoch), `callbacks`: List of callbacks to be called during training. `validation_split`: Float between 0 and 1. Fraction of the training data to be used as validation data. The model will set apart this fraction of the training data, will not train on it, and will evaluate the loss and any model metrics on this data at the end of each epoch. The validation data is selected from the last samples in the x and y data provided, before shuffling. `validation_data`: Data on which to evaluate the loss and any model metrics at the end of each epoch. The model will not be trained on this data. This could be a list (x_val, y_val) or a list (x_val, y_val, val_sample_weights). validation_data will override validation_split. `shuffle`: Logical (whether to shuffle the training data before each epoch) or string (for "batch"). "batch" is a special option for dealing with the limitations of HDF5 data; it shuffles in batch-sized chunks. Has no effect when steps_per_epoch is not NULL.

Value

control list (kerasConf)

`get_objf` *Get objective function*

Description

mlrTools This function receives a configuration for a tuning experiment, and returns an objective function to be tuned via SPOT.

Usage

```
get_objf(config, timeout = 3600)
```

Arguments

`config`	list
`timeout`	integer, time in seconds after which a model (learner) evaluation will be aborted.

Value

an objective function that can be optimized via `spot`

int2fact *Helper function: transform integer to factor*

Description

This function re-codes a factor with pre-specified factor levels, using an integer encoding as input.

Usage

```
int2fact(x, lvls)
```

Arguments

x	an integer vector (that represents factor vector) to be transformed
lvls	the original factor levels used

Value

the same factor, now coded with the original levels

modelMarkovChain *modelMarkovChain*

Description

Modeling continuous time Markov chains (MarkovChain) models using SIR models.

Usage

```
modelMarkovChain(x, days, N, n = 3)
```

Arguments

x	vector of three parameters. Used for parametrizing the MarkovChain model. p num [0;1] proportion of confirmed cases beta num A numeric vector with the transmission rate from susceptible to infected where each node can have a different beta value. The vector must have length 1 or nrow(u0). If the vector has length 1, but the model contains more nodes, the beta value is repeated in all nodes. gamma num A numeric vector with the recovery rate from infected to recovered where each node can have a different gamma value. The vector must have length 1 or nrow(u0). If the vector has length 1, but the model contains more nodes, the beta value is repeated in all nodes.

days	number of simulation steps, usually days (int). It will be used to generate (internally) a vector (length >= 1) of increasing time points where the state of each node is to be returned.
N	population size
n	number of nodes to be evaluated in the SIR model

Details

SIR considers three compartments: S (susceptible), I (infected), and R (recovered). Using the parameter vector x, the population size N, and the number of days (prediction horizon), the SIR model parameters are determined as follows. N denotes the population size. First: S, the number of susceptible in each node, will be calculated as N - I - R, where I is the number of infected in each node, and R is the number of recovered in each node. Then, the data frame 'u0' is set up: `u0 = data.frame(S ,I ,R)`. 'u0' contains the initial number of individuals in each compartment in every node. An integer matrix (Ncomp × Nnodes) is used for storing 'u0' information. The timespan is calculated as `tspan = 1:days`. The SIR is set up and run, using run.

Data are taken from the `regionTrain` and `regionPopulation` data sets that were combined using the preprocessInputData function. `regionTrain` and `regionPopulation` are stored in the `babsim.data` package.

Value

data.frame of days obs. of 4 variables:

`t` num 0 1 2 3 4 5 6 7 8 9 ... (timesteps)

`X1` num 1704 1490 1275 1069 880 ... (susceptible)

`X2` num 1000 1178 1351 1509 1646 ... (infected)

`X3` num num 0 36.3 78.5 126.2 178.8 ... (recovered)

Examples

```
require("SimInf")
data <- preprocessInputData(regionTrain, regionPopulation)
regionData <- data[[1]]
N <- attr(regionData, "regionPopulation")
# N_curr <- max(regionData$confirmed)
p <- 0.01
beta <- 0.1
gamma <- 0.01
# parameter vector for the SIR model: (p, beta, gamma)
x <- c(p, beta, gamma)
# Every row in the data represents one day:
days <- nrow(regionData)
modelMarkovChain(x = x, days = days, N = N)
```

parseTunedRegionModel *parseTunedRegionModel*

Description

Parse results from the tuneRegionModel function, i.e., results from a spot run on funMarkovChain

Usage

```
parseTunedRegionModel(xList)
```

Arguments

xList	list of results from spot run

Value

returns the following list of 3:

models data.frame with obs. of 7 variables:

- p num, e.g., 27373033
- beta num
- gamma num
- CFR num
- cost num
- region chr, e.g., "Afghanistan/"
- regionPopulation num population

pops list of x values for countries, i.e., spot population generated at each generation, e.g., Afghanistan/: num [1:6,1:4] 32039478 28078906 23529925 11257083 9883189 .... Here, 6 function evaluations were performed and the search space is 4-dim.

y function values for pops

Examples

```
require("SimInf")
require("SPOT")
data <- preprocessInputData(regionTrain, regionPopulation)
resList <- lapply(data[1], tuneRegionModel, pops=NULL, control=list(funEvals=6,
designControl=list(size=5), model = buildLM))
parsedList <- parseTunedRegionModel(resList)
```

plotPrediction *plotPrediction*

Description

plot predictions countries/regions by index

Usage

```
plotPrediction(regionDf, countryIndex = 1, ylog = FALSE)
```

Arguments

regionDf	A list containing a representation of the data.
countryIndex	num Index
ylog	logical plot log y axis (log = y)

Details

Data are taken from the regionTrain and regionPopulation data sets that were combined using the preprocessInputData function. regionTrain and regionPopulation are stored in the babsim.data package.

Value

A plot

Examples

```
require(SPOT)
data <- preprocessInputData(regionTrain, regionPopulation)
testData <- preprocessTestData(regionTest)
# Select the first region:
testData <- testData[testData$Region==levels(testData$Region)[1], ]
testData$Region <- droplevels(testData$Region)
# Very small number of function evaluations:
n <- 6
res <- lapply(data[1], tuneRegionModel, pops=NULL,
              control=list(funEvals=n, designControl=list(size=5), model = buildLM))
parsedList <- parseTunedRegionModel(res)
pred <- generateMCPrediction(testData = testData, models = parsedList$models, write = FALSE)
quickPredict <- cbind(pred, testData$Date, testData$Region)
names(quickPredict) <- c("ForecastID", "confirmed", "fatalities", "date", "region")
p <- plotPrediction(quickPredict, 1)
```

plotRegion *plotRegion*

Description

Plot confirmed cases and fatalities (cumulative) for one country/region by index.

Usage

```
plotRegion(regionList, countryIndex = 1)
```

Arguments

regionList	A list containing a representation of the data.
countryIndex	num Index

Details

Data are taken from the regionTrain and regionPopulation data sets that were combined using the preprocessInputData function. regionTrain and regionPopulation are stored in the babsim.data package.

Value

A plot

See Also

plotRegionByName.

Examples

```
regionList <- preprocessInputData(regionTrain, regionPopulation)
p <- plotRegion(regionList = regionList, countryIndex = 1)
```

plotRegionByName *plotRegionByName*

Description

Plot confirmed cases and fatalities (cumulative) for one country/region by region name.

Usage

```
plotRegionByName(regionList, country = "Germany")
```

Arguments

regionList	A list containing a representation of the data.
country	Name of a country from the list dataList

Details

Data are taken from the regionTrain and regionPopulation data sets that were combined using the preprocessInputData function. regionTrain and regionPopulation are stored in the babsim.data package.

Value

A plot

See Also

plotRegion.

Examples

```
regionList <- preprocessInputData(regionTrain, regionPopulation)
p <- plotRegionByName(regionList = regionList, country = "Germany")
```

---

plotSIRModel *plotSIRModel*

Description

Plot of continuous time Markov chains (MarkovChain) SIR models.

Usage

```
plotSIRModel(x, days, N, n = 3, logy = FALSE)
```

Arguments

x	vector of four parameters. Used for parametrizing the MarkovChain model and calulation of fatalities. p num [0;1] proportion of confirmed cases beta num A numeric vector with the transmission rate from susceptible to infected where each node can have a different beta value. The vector must have length 1 or nrow(u0). If the vector has length 1, but the model contains more nodes, the beta value is repeated in all nodes. gamma num A numeric vector with the recovery rate from infected to recovered where each node can have a different gamma value. The vector must have length 1 or nrow(u0). If the vector has length 1, but the model contains more nodes, the beta value is repeated in all nodes.

	CFR num [0;1] proportion of fatalities
days	number of simulation steps, usually days (int). It will be used to generate (internally) a vector (length >= 1) of increasing time points where the state of each node is to be returned.
N	population size
n	number of nodes to be evaluated in the SIR model
logy	logical Plot logarithmic y axis. Default: FALSE

Details

SIR considers three compartments: S (susceptible), I (infected), and R (recovered). The timespan is calculated as tspan = 1:days.

Value

plot

Examples

```
require("SimInf")
# Result from \code{\link{parseTunedRegionModel}}, e.g., deModels:
# parameters: x = c(deModels$p, deModels$beta, deModels$gamma, deModels$CFR)
x = c(1e-05, 0.216764668858674, 0.204440265426977, 0.100982384347174)
plotSIRModel(x, days=1000, N = 83783945, n=10, logy=TRUE)
```

---

plot_function_surface *Surface plot*

Description

A (filled) contour plot or perspective plot of a function, interactive via plotly.

Usage

```
plot_function_surface(
  f = function(x) {     rowSums(x^2) },
  lower = c(0, 0),
  upper = c(1, 1),
  type = "filled.contour",
  s = 100,
  xlab = "x1",
  ylab = "x2",
  zlab = "y",
  color.palette = terrain.colors,
  title = " ",
```

```
  levels = NULL,
  points1,
  points2,
  pch1 = 20,
  pch2 = 8,
  lwd1 = 1,
  lwd2 = 1,
  cex1 = 1,
  cex2 = 1,
  col1 = "blue",
  col2 = "red",
  ...
)
```

Arguments

f	function to be plotted. The function should either be able to take two vectors or one matrix specifying sample locations. i.e. `z=f(X)` or `z=f(x2,x1)` where Z is a two column matrix containing the sample locations `x1` and `x2`.
lower	boundary for x1 and x2 (defaults to `c(0,0)`).
upper	boundary (defaults to `c(1,1)`).
type	string describing the type of the plot: `"filled.contour"` (default), `"contour"`, `"persp"` (perspective), or `"persp3d"` plot. Note that "persp3d" is based on the plotly package and will work in RStudio, but not in the standard RGui.
s	number of samples along each dimension. e.g. `f` will be evaluated `s^2` times.
xlab	lable of first axis
ylab	lable of second axis
zlab	lable of third axis
color.palette	colors used, default is `terrain.color`
title	of the plot
levels	number of levels for the plotted function value. Will be set automatically with default NULL.. (contour plots only)
points1	can be omitted, but if given the points in this matrix are added to the plot in form of dots. Contour plots and persp3d only. Contour plots expect matrix with two columns for coordinates. 3Dperspective expects matrix with three columns, third column giving the corresponding observed value of the plotted function.
points2	can be omitted, but if given the points in this matrix are added to the plot in form of crosses. Contour plots and persp3d only. Contour plots expect matrix with two columns for coordinates. 3Dperspective expects matrix with three columns, third column giving the corresponding observed value of the plotted function.
pch1	pch (symbol) setting for points1 (default: 20). (contour plots only)
pch2	pch (symbol) setting for points2 (default: 8). (contour plots only)
lwd1	line width for points1 (default: 1). (contour plots only)
lwd2	line width for points2 (default: 1). (contour plots only)

cex1	cex for points1 (default: 1). (contour plots only)
cex2	cex for points2 (default: 1). (contour plots only)
col1	color for points1 (default: "black"). (contour plots only)
col2	color for points2 (default: "black"). (contour plots only)
...	additional parameters passed to contour or filled.contour

Value

plotly visualization (based on plot_ly)

---

plot_parallel *Parallel coordinate plot of a data set*

Description

mlrTools

Usage

```
plot_parallel(
  object,
  yrange = NULL,
  yvar = 1,
  xlab = paste("x", 1:ncol(x), sep = ""),
  ylab = "y",
  ...
)
```

Arguments

object	the result list returned by spot, importantly including a modelFit, and the data x, y.
yrange	a two-element vector that specifies the range of y values to consider (data outside of that range will be excluded).
yvar	integer which specifies the variable that is displayed on the color scale. yvar==1 (default) means that the y-variable is shown (tuned measure). Larger integers mean that respective columns from logInfo are used (i.e., yvar specifies the respective column number, starting with 2 for the first logged value).
xlab	a vector of characters, giving the labels for each of the two independent variables.
ylab	character, the value of the dependent variable predicted by the corresponding model.
...	additional parameters (currently unused).

Value

plotly parallel coordinate plot ('parcoords') visualization (based on plot_ly)

See Also

plotFunction, plotData

`plot_pareto` *Pareto front (as well as non-optimal solutions)*

Description

mlrTools

Usage

```
plot_pareto(object, xvar = 1, yvar = 2, xlab = NULL, ylab = NULL, ...)
```

Arguments

`object`	the result list returned by spot, importantly including a `modelFit`, and the data `x`, `y`.
`xvar`	integer which specifies the variable that is displayed on the x axis. xvar=1 (default) means that the y-variable is shown (tuned measure). Larger integers mean that respective columns from logInfo are used (then, ylog specifies the respective column number, starting with 2 for the first logged value).
`yvar`	integer which specifies the variable that is displayed on the color scale. yvar==1 (default) means that the y-variable is shown (tuned measure). Larger integers mean that respective columns from logInfo are used (then, ylog specifies the respective column number, starting with 2 for the first logged value).
`xlab`	character, giving the label for the x-axis of the plot.
`ylab`	character, giving the label for the y-axis of the plot.
`...`	additional parameters (currently unused). By default, it is assumed that all variables that are plotted (yvar,xvar) are minimized. If this is not true, assign a negative sign to the respective integer of the xvar and yvar arguments.

Value

plotly visualization (type 'scatter'), based on plot_ly

See Also

plotFunction, plotData

`plot_sensitivity` *Sensitivity plot of a model*

Description

mlrTools

Usage

```
plot_sensitivity(
  object,
  s = 100,
  xlab = paste("x", 1:ncol(object$x), sep = ""),
  ylab = "y",
  type = "best",
  agg.sample = 100,
  agg.fun = mean,
  ...
)
```

Arguments

`object`	the result list returned by `spot`, importantly including a `modelFit`, and the data `x`, `y`.
`s`	number of samples along each dimension.
`xlab`	a vector of characters, giving the labels for each of the two independent variables.
`ylab`	character, the value of the dependent variable predicted by the corresponding model.
`type`	string describing the type of the plot: `"best"` (default) shows sensitivity around optimum, `"contour"`, `"persp"` (perspective), or `"persp3d"` plot. Note that "persp3d" is based on the plotly package and will work in RStudio, but not in the standard RGui.
`agg.sample`	number of samples for aggregation type (type="agg").
`agg.fun`	function for aggregation (type="agg").
`...`	additional parameters (currently unused).

Value

plotly visualization (based on `plot_ly`)

See Also

`plotFunction`, `plotData`

plot_surface *Surface plot of a model*

Description

A (filled) contour or perspective plot of a fitted model.

Usage

```
plot_surface(
  object,
  which = if (ncol(object$x) > 1 & tolower(type) != "singledim") {    1:2 } else {
    1 },
  constant = object$x[which.min(unlist(object$y)), ],
  xlab = paste("x", which, sep = ""),
  ylab = "y",
  type = "filled.contour",
  ...
)
```

Arguments

object	the result list returned by spot, importantly including a modelFit, and the data x, y.
which	a vector with two elements, each an integer giving the two independent variables of the plot (the integers are indices of the respective data set).
constant	a numeric vector that states for each variable a constant value that it will take on if it is not varied in the plot. This affects the parameters not selected by the which parameter. By default, this will be fixed to the best known solution, i.e., the one with minimal y-value, according to which.min(object$y). The length of this numeric vector should be the same as the number of columns in object$x
xlab	a vector of characters, giving the labels for each of the two independent variables.
ylab	character, the value of the dependent variable predicted by the corresponding model.
type	string describing the type of the plot: "filled.contour" (default), "contour", "persp" (perspective), or "persp3d" plot. Note that "persp3d" is based on the plotly package and will work in RStudio, but not in the standard RGui.
...	additional parameters passed to the contour or filled.contour function.

Value

plotly visualization (based on plot_ly)

`prepare_data_plot` *Prepare data for plots*

Description

mlrTools

Usage

```
prepare_data_plot(
  model = buildRanger,
  modelControl = list(),
  x,
  namesx = paste("x", 1:ncol(x), sep = ""),
  y,
  namesy = "y",
  log = NULL,
  nameslog = NULL
)
```

Arguments

`model`	a function that can be used to build a model based on the data, e.g. : `buildRanger` or `buildKriging`. Default is `buildRanger`, since it is fast and robust.
`modelControl`	a list of control settings for the respective model. Default is an empty list (use default model controls).
`x`	a matrix of x-values to be plotted (i.e., columns are the independent variables, rows are samples). Should have same number of rows as y and log.
`namesx`	character vector, printable names for the x data. Should have same length as x has columns. Default is x1, x2, ...
`y`	a one-column matrix of y-values to be plotted (dependent variable). Should have same number of rows as x and log.
`namesy`	character, giving a printable name for y. Default is "y".
`log`	matrix, a data set providing (optional) additional dependent variables (but these are not modeled). Should have same number of rows as y and x.
`nameslog`	character vector, printable names for the log data. Should have same length as log has columns. Default is NULL (no names).

Value

list with plotting data and information

`prepare_spot_result_plot` *Prepare data (results from a tuning run) for plots*

Description

Preparation ot the list elements used for plotting.

Usage

```
prepare_spot_result_plot(data, model = buildRanger, modelControl = list(), ...)
```

Arguments

`data`	a list containing the various data, e.g., as produced by a `spot` call.
`model`	a function that can be used to build a model based on the data, e.g. : `buildRanger` or `buildKriging`. Default is `buildRanger`, since it is fast and robust.
`modelControl`	a list of control settings for the respective model. Default is an empty list (use default model controls).
`...`	additional parameters passed to `prepare_data_plot`: `namesx`, `namesy`, `nameslog` character vectors providing names for x, y and logInfo data. Se

Value

list with plotting data and information generated with `prepare_data_plot`

`preprocessCdeInputData` *preprocessCdeInputData*

Description

Prepare Ced Data for SIR modeling

Usage

```
preprocessCdeInputData(cdeData)
```

Arguments

`cdeData`	data frame

Details

Resulting data set can be used as input for the COVID-19 simulations of the `tuneRegionModel` function.

Value

A large list containing the following information (3 lists) for each region:

`date` Date

`confirmed` num Number of confirmed cases (cumulative)

`fatalities` num Number of fatalities (cumulative)

`preprocessCdeTestData` *preprocessCdeTestData*

Description

Process Corona Data Explorer (CDE) Data. Rename variables and factorize location information.

Usage

```
preprocessCdeTestData(testData)
```

Arguments

`testData`	data frame with location information:
	`ForecastId` int Identifier 1 2 3 ...
	`Province_State` chr Province/State
	`Country_Region` chr Country/Region, e.g., "Afghanistan" ...
	`Date` Date, format: "2020-04-02" ...

Details

The variable `location` is renamed to `Region` and converted to a factor variable.

Value

A data frame that can be processed by `generateMCPrediction` to predict results on test data using the tuned `{link{modelMarkovChain}` model. Data.frame with n obs. of 3 variables:

`ForecastId` int Identifier 1 2 3 ...

`Region` Factor w/ m levels, e.g., "Afghanistan/",..: 1 1 1 1 1 1 1 1 1 1 ...

`Date` Date, format: "2020-04-02" ...

preprocessInputData *preprocessInputData*

Description

Combine information from the `regionTrain` and the `regionPopulation` data sets.

Usage

```
preprocessInputData(trainData, populationData)
```

Arguments

`trainData`	data frame
`populationData`	data frame

Details

Resulting data set can be used as input for the COVID-19 simulations of the `tuneRegionModel` function.

Value

A large list (313 elements, 1.3 MB) containing the following information (3 lists) for each of the 313 regions:

`date` Date

`confirmed` num Number of confirmed cases (cumulative)

`fatalities` num Number of fatalities (cumulative)

Examples

```
x <- preprocessInputData(regionTrain, regionPopulation)
# Plot confirmed cases from the first country (Afghanistan):
p <- plot(x[[1]]$date, x[[1]]$confirmed)
```

preprocessTestData *preprocessTestData*

Description

Combine Province/State and Country/Region information.

Usage

```
preprocessTestData(testData)
```

Arguments

testData	data frame with n obs. of 4 variables:
	ForecastId int Identifier 1 2 3 ...
	Province_State chr Province/State
	Country_Region chr Country/Region, e.g., "Afghanistan" ...
	Date Date, format: "2020-04-02" ...

Details

Locality information is merged with the paste command as follows: paste(testData$Country_Region,testData$Provir
4-dim data is reduced to 3-dim data.

Value

A data frame that can be processed by generateMCPrediction to predict results on test data using the tuned {link{modelMarkovChain} model. Data.frame with n obs. of 3 variables:

ForecastId int Identifier 1 2 3 ...

Region Factor w/ m levels, e.g., "Afghanistan/",..: 1 1 1 1 1 1 1 1 1 1 ...

Date Date, format: "2020-04-02" ...

Examples

```
testData <- preprocessTestData(regionTest)
```

regionPopulation *Region Population Data*

Description

A data set of populations.

Usage

```
regionPopulation
```

Format

A data frame with 313 rows and 2 columns:

regionName chr: Country. ("Afghanistan/"–""Zimbabwe/")

population int: Population size. (825–1366417754)

Source

<http://owos.gm.fh-koeln.de:8055/bartz/bartz-data/blob/cd4aad9a6640f7c36caa53c898a78568e34374e8/Covid-19/regionPopulation.csv>

See Also

`regionTrain` and `regionTest`.

`cde20200813` and `DEcde20200813`, which are part of the `babsim.data` package, because these are relatively large data sets.

regionTest *Region Test Data*

Description

A data set of COVID-19 cases. Test data for predictions. From "2020-04-02" until "2020-05-14". Time difference of 42 days.

Usage

```
regionTest
```

Format

A data frame with 13459 rows and 4 columns:

ForecastId int id. (1–13459)

Province_State Province State. (""–"Zhejiang"). No NAs, but empty char.

Country_Region Country. ("Afghanistan"–"Zimbabwe")

Date chr: yyyy-mm-dd. ("2020-04-02"–"2020-05-14")

Source

<http://owos.gm.fh-koeln.de:8055/bartz/bartz-data/blob/412e9cf647a6fce875bd49ccddcd5aea8aeb4246/Covid-19/regionTest.csv>

See Also

`regionTrain` and `regionPopulation`.

`cde20200813` and `DEcde20200813`, which are part of the `babsim.data` package, because these are relatively large data sets.

`regionTrain` *Region Train Data*

Description

A data set of COVID-19 cases from "2020-01-22" until "2020-05-03". Time difference of 102 days.

Usage

```
regionTrain
```

Format

A data frame with 32239 rows and 6 columns:

Id id

Province_State Province State ("Alabama"–"Zhejiang"). Also NA

Country_Region Country ("Afghanistan"–"Zimbabwe")

Date chr: yyyy-mm-dd. ("2020-01-22"–"2020-05-03")

ConfirmedCases num

Fatalities num

Source

<http://owos.gm.fh-koeln.de:8055/bartz/bartz-data/blob/cd4aad9a6640f7c36caa53c898a78568e34374e8/Covid-19/regionTrain.csv>

See Also

regionTest and regionPopulation.

cde20200813 and DEcde20200813, which are part of the babsim.data package, because these are relatively large data sets.

---

resTuneRegionModel *Tuned Region Model Data*

Description

A data set resulting from a 'spot' tuning run.

Usage

```
resTuneRegionModel
```

Format

A list of 7 entries:

regionName e.g., "Afghanistan/": List of 7

- xbest Parameters of the best found solution (matrix).
- ybest Objective function value of the best found solution (matrix).
- x Archive of all evaluation parameters (matrix).
- y Archive of the respective objective function values (matrix).
- count Number of performed objective function evaluations.
- msg Message specifying the reason of termination.
- modelFit The fit of the last build model, i.e., an object returned by the last call to the function specified by control$model.

Details

Result of a SPOT::tuneRegionModel run. The data (list of 1) was generated as follows: data <-preprocessInputData(regionTrain,regionPopulation); resTuneRegionModel <-lapply(data[1],tuneRegionM = buildLM)). To accelerate testing, this list is used internally.

Source

<http://owos.gm.fh-koeln.de:8055/bartz/bartz-data/blob/cd4aad9a6640f7c36caa53c898a78568e34374e8/Covid-19/regionTrain.csv>

See Also

regionTrain regionTest regionPopulation

`sequentialBifurcation` *Sequential Bifurcation*

Description

`sequentialBifurcation` is a wrapper function to `sb` from the `sensitivity` package.

Usage

```
sequentialBifurcation(
  fun,
  lower,
  upper,
  k,
  interaction = FALSE,
  verbosity = 0,
  ...
)
```

Arguments

`fun`	function
`lower`	bound of natural variables. Determines the number of parameters (variables).
`upper`	bound of natural variables
`k`	`integer` bifurcations. Must be smaller than the number of parameters.
`interaction`	`logical` TRUE if two-factor interactions should be considered. Default is `FALSE`.
`verbosity`	`integer`. If larger than zero, the designs are shown.
`...`	optional parameters passed to fun

Details

The model without interaction is `Y = beta_0 + sum_{i=1}^p beta_i X_i`, while the model with two-factor interactions is `Y = beta_0 + sum_{i=1}^p beta_i X_i + sum_{1 <= i < j <= p} gamma_{ij} X_i X_j`. In both cases, the factors are assumed to be uniformly distributed on [-1,1]. This is a difference with Bettonvil et al. where the factors vary across [0,1] in the former case, while [-1,1] in the latter. Another difference with Bettonvil et al. is that in the current implementation, the groups are splitted right in the middle.

Value

sa `list` with sensitivity information (effects) for subgroups.

References

B. Bettonvil and J. P. C. Kleijnen, 1996, Searching for important factors in simulation models with many factors: sequential bifurcations, European Journal of Operational Research, 96, 180–194.

subgroups *Return effects for each subgroup*

Description

subgroups: returns the table the effects per groups. Code based on the sbgroups function written by Gilles Pujol for the function sb in the sensitivity package.

Usage

```
subgroups(x)
```

Arguments

x	data

Value

data frame with group names and effects

Examples

```
require("SPOT")
require("RColorBrewer")
set.seed(2)
# Interesting for larger n:
n <- 2
lower <- c(-0.1, rep(-10,n))
upper <- c(0.1, rep(10,n))

# Model-based optimization
res <- spot(,funSphere,
             lower, upper,
             control=list(funEvals=30,
                          optimizer = optimNLOPTR))

# Use the surrogate model for prediction
predictFunKriging <- function(x){
      predict(object = res$modelFit, x)
      }

# Determine sensitivity
sens <- sequentialBifurcation(predictFunKriging,
                              lower, upper,
                              k=n+1, interaction = TRUE, verbosity = 0)

# Extract group information (variable effects) from sensitivity analysis
ps <- subgroups(sens)
colors <- brewer.pal(12, "Set3")
```

```
barplot(ps$effect, names.arg=ps$group, col= colors)
```

`translate_levels` *Helper function: translate levels*

Description

Translate existing levels of a factor into new levels.

Usage

```
translate_levels(x, translations)
```

Arguments

`x`	a factor vector to be translated
`translations`	a named list that specifies the translation: `list(newlevel=c(oldlevel1,oldlevel2,etc))`.

Value

translated factor

`trans_10pow` *10 power x transformation*

Description

Parameter values can be translated, e.g., to base 10.

Usage

```
trans_10pow(x)
```

Arguments

`x`	input

Value

`10^x`

trans_10pow_round *10 power x transformation with round*

Description

Parameter values can be translated, e.g., to base 10 as implemented in trans_10pow. trans_10pow_round implements the transformation x -> round(2^x).

Usage

```
trans_10pow_round(x)
```

Arguments

x	input

Value

round(10^x)

trans_2pow *2 power x transformation*

Description

Parameter values can be translated, e.g., to base 10 as implemented in trans_10pow. trans_2pow implements the transformation x -> 2^x.

Usage

```
trans_2pow(x)
```

Arguments

x	input

Value

2^x

trans_2pow_round *2 power x transformation with round*

Description

Parameter values can be translated, e.g., to base 10 as implemented in trans_10pow. trans_2pow_round implements the transformation x -> round(2^x).

Usage

```
trans_2pow_round(x)
```

Arguments

x	input

Value

round(2^x)

trans_id *Identity transformation*

Description

Parameter values can be translated, e.g., to base 10 as implemented in trans_10pow. trans_id implements the identity (transformation), i.e., x is mapped to x.

Usage

```
trans_id(x)
```

Arguments

x	input

Value

x

tuneRegionModel *tuneRegionModel*

Description

Perform a spot run on funMarkovChain with region data. Results can be postprocessed with the function parseTunedRegionModel to extract model and parameter information.

Usage

```
tuneRegionModel(
  regionData,
  pops = NULL,
  lower = NULL,
  upper = NULL,
  control = list()
)
```

Arguments

regionData	is a data.frame with observations of 3 variables: data Date, format: "2020-01-22" "2020-01-23" "2020-01-24" "2020-01-25" ... confirmed num 0 0 0 0 0 0 0 0 0 0 .. fatalities fatalities: num 0 0 0 0 0 0 0 0 0 0 ... and attributes - attr(*, "regionName")= chr "Afghanistan/" - attr(*, "regionPopulation")= int 38041754
pops	evaluated populations
lower	lower bounds for spot optimization, @seealso Link{spot}
upper	upper bounds for spot optimization, @seealso Link{spot}
control	spot control list, see controlSpot

Details

Note: the default number of function evaluations is very low.

Value

This function returns a list with:

regionName e.g., "Afghanistan/": List of 7

- xbest Parameters of the best found solution (matrix).
- ybest Objective function value of the best found solution (matrix).
- x Archive of all evaluation parameters (matrix).
- y Archive of the respective objective function values (matrix).
- count Number of performed objective function evaluations.

`msg` Message specifying the reason of termination.

`modelFit` The fit of the last build model, i.e., an object returned by the last call to the function specified by `control$model`.

Examples

```
require("SimInf")
require("SPOT")
data <- preprocessInputData(regionTrain, regionPopulation)
a <- c(0.01,  0.001, 0.001,   0.001)
b <- c(0.1,  0.01, 0.01,   0.01)
lapply(data[1], tuneRegionModel, pops=NULL, lower = a, upper = b,
control=list(funEvals=6,
designControl=list(size=5), model = buildLM))
```

Index