Package ‘SPOTMisc’

October 12, 2022

**Type**  Package

**Title**  Misc Extensions for the 'SPOT' Package

**Version**  1.19.52

**Maintainer**  Thomas Bartz-Beielstein &lt;tbb@bartzundbartz.de&gt;

**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.


Bettonvil, B, Kleijnen JPC (1996) &lt;doi:10.1016/S0377-2217(96)00156-7&gt;.

**License**  GPL (>= 2)

**Encoding**  UTF-8

**LazyLoad**  yes

**LazyData**  true

**LazyDataCompression**  xz

**Date**  2022-08-31

**Depends**  R (>= 4.0.0)

**RoxygenNote**  7.2.1

**Imports**  callr, dplyr, ggplot2, GGally, graphics, grDevices, keras, magrittr, mlr, Metrics, plotly, RColorBrewer, reticulate, rlang, rpart.plot, rsample, sensitivity, smoof, SPOT, stats, tensorflow, tfdatasets, utils

**Suggests**  farff, knitr, rmarkdown, rpart, testthat, xgboost

**URL**  https://www.spotseven.de

**VignetteBuilder**  knitr

**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  2022-09-05 15:10:02 UTC  

R topics documented:  

<table>
<thead>
<tr>
<th>Function</th>
<th>Line No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>active2All</td>
<td>4</td>
</tr>
<tr>
<td>dataCensusFull</td>
<td>5</td>
</tr>
<tr>
<td>evalKerasGeneric</td>
<td>5</td>
</tr>
<tr>
<td>evalKerasMnist</td>
<td>6</td>
</tr>
<tr>
<td>evalKerasMnist_0</td>
<td>7</td>
</tr>
<tr>
<td>evalKerasTransferLearning</td>
<td>8</td>
</tr>
<tr>
<td>evalParamCensus</td>
<td>10</td>
</tr>
<tr>
<td>funBBOBCall</td>
<td>11</td>
</tr>
<tr>
<td>funKerasGeneric</td>
<td>12</td>
</tr>
<tr>
<td>funKerasMnist</td>
<td>14</td>
</tr>
<tr>
<td>funKerasMnist_0</td>
<td>16</td>
</tr>
<tr>
<td>funKerasTransferLearning</td>
<td>19</td>
</tr>
<tr>
<td>genCatsDogsData</td>
<td>23</td>
</tr>
<tr>
<td>genericDataPrep</td>
<td>24</td>
</tr>
<tr>
<td>getDataCensus</td>
<td>25</td>
</tr>
<tr>
<td>getGenericTrainValTestData</td>
<td>26</td>
</tr>
<tr>
<td>getIndices</td>
<td>28</td>
</tr>
<tr>
<td>getKerasConf</td>
<td>28</td>
</tr>
<tr>
<td>getMlConfig</td>
<td>30</td>
</tr>
<tr>
<td>getMlrResample</td>
<td>31</td>
</tr>
<tr>
<td>getMlrTask</td>
<td>33</td>
</tr>
<tr>
<td>getMnistData</td>
<td>34</td>
</tr>
<tr>
<td>getModelConf</td>
<td>35</td>
</tr>
<tr>
<td>getObjf</td>
<td>36</td>
</tr>
<tr>
<td>getPredf</td>
<td>37</td>
</tr>
<tr>
<td>getSimpleKerasModel</td>
<td>37</td>
</tr>
<tr>
<td>getVarNames</td>
<td>38</td>
</tr>
<tr>
<td>ggparcoordPrepare</td>
<td>39</td>
</tr>
<tr>
<td>ggplotProgress</td>
<td>40</td>
</tr>
<tr>
<td>int2fact</td>
<td>41</td>
</tr>
<tr>
<td>kerasBuildCompile</td>
<td>42</td>
</tr>
<tr>
<td>kerasCompileResult</td>
<td>42</td>
</tr>
<tr>
<td>kerasEvalPrediction</td>
<td>44</td>
</tr>
<tr>
<td>kerasFit</td>
<td>45</td>
</tr>
<tr>
<td>kerasReturnDummy</td>
<td>46</td>
</tr>
<tr>
<td>makeLearnerFromHyperparameters</td>
<td>47</td>
</tr>
<tr>
<td>mapX2FLAGS</td>
<td>47</td>
</tr>
<tr>
<td>MSE</td>
<td>48</td>
</tr>
</tbody>
</table>
R topics documented:

optimizer_adadelta ........................................... 49
optimizer_adagrad ............................................ 50
optimizer_adam .................................................. 51
optimizer_adamax ............................................... 52
optimizer_nadam ............................................... 53
optimizer_rmsprop ............................................ 54
optimizer_sgd .................................................. 55
plotParallel .................................................... 56
plotSensitivity .................................................. 57
plot_function_surface ....................................... 58
plot_parallel ................................................... 60
plot_sensitivity ............................................... 61
plot_surface .................................................... 62
predDlCensus ................................................... 63
predMlCensus ................................................... 65
prepareComparisonPlot ....................................... 66
prepareProgressPlot .......................................... 67
prepare_data_plot ............................................. 68
prepare_spot_result_plot .................................... 69
printf ........................................................... 69
printFLAGS ....................................................... 70
resDl100 .......................................................... 70
RMSE .............................................................. 71
scorePredictions ............................................... 71
selectKerasActivation ........................................ 72
selectKerasOptimizer ......................................... 72
selectTarget ..................................................... 73
sequentialBifurcation ........................................ 74
spotKeras .......................................................... 75
spotPlot .......................................................... 76
SSE ................................................................. 77
startCensusRun ................................................ 78
startMnistRun .................................................. 81
startXGBCensusRun .......................................... 83
subgroups ......................................................... 86
translate_levels ................................................ 87
trans_10pow ...................................................... 88
trans_10pow_round .......................................... 88
trans_1minus10pow ........................................... 89
trans_2pow ....................................................... 89
trans_2pow_round ............................................. 90
trans_id ............................................................ 91
trans_mult2_round ............................................. 91
trans_odd_round ................................................ 92

Index 93
**active2All**  
*Active to all*

**Description**
Recreates the full set of parameters from the subset of active ones.

**Usage**
```r
active2All(x, a, model)
```

**Arguments**
- `x` subset of parameters
- `a` names of the active parameters
- `model` model (char)

**Value**
y full parameters

**Examples**
```r
model <- "dl"
# indices of active variables
i <- c(1,3)
# names of active variables
a <- getVarNames(model=model, i=i)
x <- getModelConf(model=model)$defaults
# now matrix x has only active variables 1 and 3:
x <- x[1, getIndices(model=model, a=a), drop=FALSE]
# reconstruct the full set of parameters
active2All(x, a, model)
# 2nd example: new values to x (dropout=0.1, units=11):
x <- matrix(c(0.1,11), nrow=1)
a <- c("dropout", "units")
# reconstruct the full set of parameters
active2All(x, a, model)
# matrix
x <- rbind(x, 2*x)
active2All(x, a, model)
```
dataCensusFull

---

**dataCensusFull**  
*Full census data set*

---

### Description

Census KDD Dataset (OpenML ID: 4535). Data frame with 299285 obs. of 42 variables, obtained via `OpenML::getOMLDataSet(data.id=4535, cache.only=cache.only)$data`.

### Usage

`dataCensusFull`

### Format

A data frame with 42 variables.

---

### evalKerasGeneric

**evalKerasGeneric**  
*evalKerasGeneric model building and compile*

---

### Description

Hyperparameter Tuning: Keras Generic Classification Function.

### Usage

`evalKerasGeneric(x = NULL, kerasConf = NULL, specList = NULL)`

### Arguments

- `x`: matrix of transformed hyperparameter values to evaluate with the function. If `NULL`, a simple keras model will be build, which is considered default (see `getSimpleKerasModel`).
- `kerasConf`: List of additional parameters passed to keras as described in `getKerasConf`. If no value is specified, `stop()` is called.
- `specList`: prepared data. See `genericDataPrep`. See `getGenericTrainValTestData`.

### Details


### Value

List with function values (training, validation, and test loss/accuracy, and keras model information).
evalKerasMnist

See Also

getakerasConf
funKerasGeneric
fit

evalKerasMnist evalKerasMnist

description

Hyperparameter Tuning: Keras MNIST Classification Test Function.

Usage

evalKerasMnist(x, kerasConf, data)

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 as described in getKerasConf.
Default: kerasConf = getKerasConf()


Details

Modified by T. Bartz-Beielstein (tbb@bartzundbartz.de)

Value

list with function values (training, validation, and test loss/accuracy, and keras model information)

See Also

getKerasConf
funKerasMnist
fit
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,
### PYTHON_RETICULATE <- FALSE
### if(PYTHON_RETICULATE){

```r
library("SPOTMisc")
kerasConf <- getKerasConf()
kerasConf$verbose <- 1
kerasConf$model <- "dl"
cfg <- getModelConf(kerasConf)
x <- matrix(cfg$default, nrow=1)
if (length(cfg$transformations) > 0) { x <- transformX(xNat=x, fn=cfg$transformations)}
res <- evalKerasMnist(x, kerasConf, data = getMnistData(kerasConf))
#
kerasConf$model <- "cnn"
kerasConf$encoding <- "tensor"
cfg <- getModelConf(kerasConf)
x <- matrix(cfg$default, nrow=1)
if (length(cfg$transformations) > 0) { x <- transformX(xNat=x, fn=cfg$transformations)}
res <- evalKerasMnist(x, kerasConf, data = getMnistData(kerasConf))
```

### Description

Hyperparameter Tuning: Keras MNIST Classification Test Function.

### Usage

evalKerasMnist_0(x, kerasConf = getKerasConf(), data = getMnistData())

### 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 as described in `getKerasConf`. Default: `kerasConf = getKerasConf()`.
- **data**: mnist data set. Default: `getMnistData()`.

### Details

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

Description

Hyperparameter Tuning: Keras TransferLearning Test Function.

Usage

evalKerasTransferLearning(x, kerasConf = getKerasConf(), data = NULL)
evalKerasTransferLearning

Arguments

- **x**: matrix of hyperparameter values to evaluate with the function. Rows for points and columns for dimension. "dropout" = x[1], "learning_rate" = x[2], "epochs" = x[3], "beta_1" = x[4], "beta_2" = x[5], "epsilon" = x[6], and "optimizer" = x[7] (type: factor).

- **kerasConf**: List of additional parameters passed to keras as described in `getKerasConf`. Default: kerasConf = getKerasConf().

- **data**: data

Details

Trains a transfer learning model. Standard Code from https://tensorflow.rstudio.com/ Modified by T. Bartz-Beielstein (tbb@bartzundbartz.de)

Value

- list with function values (training, validation, and test loss/accuracy, and keras model information)

See Also

- `getKerasConf`
- `funKerasTransferLearning`
- `funKerasMnist`
- `fit`

Examples

```r
### These examples require an activated Python environment as described in
### PYTHON_RETICULATE <- FALSE
if(PYTHON_RETICULATE){
  library("SPOTMisc")
  lower <- c(1e-6, 1e-6, 1, 0.6, 0.99, 1e-9, 1)
  x <- matrix(lower, 1,
  res <- evalKerasTransferLearning(x,
        kerasConf = getKerasConf()
  )
  str(res)
  ### The number of units for all layers can be listed as follows:
  res$modelConf$config$layers[,2]$units
}
```
evalParamCensus  

evaluate hyperparameter config on census data

Description

evaluate hyperparameter config on census data

Usage

evalParamCensus(
  runNr = "00",
  model = "dl",
  xbest = "xBestOcba",
  k = 30,
  directory = "data",
  target = "age",
  cachedir = "oml.cache",
  task.type = "classif",
  nobs = 10000,
  nfactors = "high",
  nnumericals = "high",
  cardinality = "high",
  prop = 2/3,
  verbosity = 0
)

Arguments

runNr        run number (character)
model        ml/dl model (character)
xbest        best value, e.g., "xBestOcba" or "xbest"
k            number of repeats (integer)
directory    location of the (non-default, e.g., tuned) parameter file
target       "age" or "income_class"
cachedir     cache dir
task.type    task type: "classif" or "regression"
nobs         number of observations
nfactors     factors, e.g., "high"
nnumericals  numericals
cardinality  cardinality
prop         proportion. Default: 2/3
verbosity    verbosity level (0 or 1)
funBBOBCall

Examples

### These examples require an activated Python environment as described in

```r
PYTHON_RETICULATE <- FALSE
if(PYTHON_RETICULATE){
  ## The following code was used to evaluate the results in the book
  ## "Hyperparameter Tuning for Machine and Deep Learning with R - A Practical Guide"
  ## by Bartz, Bartz-Beielstein, Zaefferer, Mersmann:
  ##
  modellist <- list("dl", "cvglmnet", "kknn", "ranger", "rpart", "svm", "xgboost")
  runNr <- list("100", "Default")
  directory <- ".\book\data"
  for (model in modellist){
    for (run in runNr){
      score <- evalParamCensus(model = model,
                              runNr = run,
                              directory = directory,
                              prop = 2/3,
                              k = 30)
      fileName <- paste0(directory, "/", model, run, "Evaluation.RData")
      save(score, file = fileName)
    }
  }
}
```

funBBOBCall

Description

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

Usage

```r
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
funKerasGeneric

Value

1-column matrix with resulting function values

Examples

```r
## 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 ))
```

funKerasGeneric  funKerasGeneric

Description

Hyperparameter Tuning: Generic Classification Objective Function.

Usage

```r
funKerasGeneric(x, kerasConf = NULL, specList = NULL)
```

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 as described in `getKerasConf`. Default: NULL.

- **specList**
  - prepared data. See `genericDataPrep`.

Details

Trains a simple deep NN on arbitrary data sets. Provides a template that can be used for other networks as well. Standard Code from [https://tensorflow.rstudio.com/](https://tensorflow.rstudio.com/) Modified by T. Bartz-Beielstein (tbb@bartzundbartz.de)

Note: The WARNING "tensorflow:Layers in a Sequential model should only have a single input tensor. Consider rewriting this model with the Functional API" can be safely ignored: in general, Keras encourages its users to use functional models for multi-input layers, but there is nothing wrong with doing so. See: [https://github.com/tensorflow/recommenders/issues/188](https://github.com/tensorflow/recommenders/issues/188).
Value

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

See Also

getKerasConf
evalKerasGeneric
evalKerasGeneric
fit

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,
PYTHON_RETICULATE <- FALSE
if(PYTHON_RETICULATE){

## data preparation
target <- "age"
batch_size <- 32
prop <- 2/3
dfGeneric <- getDataCensus(target = target, nobs = 1000)
data <- getGenericTrainValTestData(dfGeneric = dfGeneric, prop = prop)
specList <- genericDataPrep(data=data, batch_size = batch_size)

## model configuration:
cfg <- getModelConf(list(model="dl"))
x <- matrix(cfg$default, nrow=1)
transformFun <- cfg$transformations
types <- cfg$type
lower <- cfg$lower
upper <- cfg$upper

kerasConf <- getKerasConf()

### First example: simple function call:
message("objectiveFunctionEvaluation(): x before transformX().")
print(x)
if (length(transformFun) > 0) { x <- transformX(xNat=x, fn=transformFun)}
message("objectiveFunctionEvaluation(): x after transformX().")
print(x)
funKerasGeneric(x, kerasConf = kerasConf, specList = specList)

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

### Third example: spot call with extended verbosity:
funKerasMnist

res <- spot(x = NULL,
    fun = funKerasGeneric,
    lower = lower,
    upper = upper,
    control = list(funEvals=50,
        handleNAsMethod = handleNAsMean,
        noise = TRUE,
        types = types,
        plots = TRUE,
        progress = TRUE,
        seedFun = 1,
        seedSPOT = 1,
        transformFun=transformFun),
        kerasConf = kerasConf,
        specList = specList)

funKerasMnist

Description

Hyperparameter Tuning: Keras MNIST Classification Test Function.

Usage

funKerasMnist(x, kerasConf, data)

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 as described in getKerasConf. Default: kerasConf = getKerasConf.

Details

Trains a simple deep NN on the MNIST dataset. Provides a template that can be used for other networks as well. Standard Code from https://tensorflow.rstudio.com/ Modified by T. Bartz-Beielstein (tbb@bartzundbartz.de)

Value

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

See Also
getKerasConf
evalKerasMnist
fit

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,

PYTHON_RETICULATE <- FALSE
if(PYTHON_RETICULATE){
  library("SPOTMisc")
  library("SPOT")
  kerasConf <- getKerasConf()
  ## The following two settings are default:
  kerasConf$encoding = "oneHot"
  kerasConf$model = "dl"
  ## get the data with the correct encoding
  mnist <- getMnistData(kerasConf)
  ## get the model
  cfg <- getModelConf(kerasConf)
  ### First example: simple function call:
  x <- matrix(cfg$default, nrow=1)
  if (length(cfg$transformations) > 0) { x <- transformX(xNat=x, fn=cfg$transformations)}
  funKerasMnist(x, kerasConf = kerasConf, data = mnist)
  ### Use convnet:
  kerasConf <- getKerasConf()
  kerasConf$encoding <- "tensor"
  kerasConf$model <- "cnn"
  mnist <- getMnistData(kerasConf)
  cfg <- getModelConf(kerasConf)
  x <- matrix(cfg$default, nrow=1)
  if (length(cfg$transformations) > 0) { x <- transformX(xNat=x, fn=cfg$transformations)}
  funKerasMnist(x, kerasConf = kerasConf, data = mnist)
  ### Second example: evaluation of several (three) hyperparameter settings:
  x <- matrix(cfg$default, nrow=1)
  if (length(cfg$transformations) > 0) { x <- transformX(xNat=x, fn=cfg$transformations)}
  xxx <- rbind(x,x,x)
  funKerasMnist(xxx, kerasConf = kerasConf, data=mnist)
  ### Third example: spot call (dense network):
  kerasConf <- getKerasConf()
  kerasConf$verbose <- 0
  kerasConf$encoding = "oneHot"
  kerasConf$model = "dl"
funKerasMnist_0

## get the data with the correct encoding
mnist <- getMnistData(kerasConf)
## get the model
cfg <- getModelConf(kerasConf)
## max 32 training epochs
resDl <- spot(x = NULL,
  fun = funKerasMnist,
  lower = cfg$lower,
  upper = cfg$upper,
  control = list(funEvals=15,
    transformFun = cfg$transformations,
    types = cfg$type,
    noise = TRUE,
    plots = TRUE,
    progress = TRUE,
    seedFun = 1,
    seedSPOT = 1),
    kerasConf = kerasConf,
    data = mnist)

### Fourth example: spot call (convnet):
kerasConf <- getKerasConf()
kerasConf$verbose <- 1
kerasConf$encoding <- "tensor"
kerasConf$model <- "cnn"
## get the data with the correct encoding
mnist <- getMnistData(kerasConf)
## get the model
cfg <- getModelConf(kerasConf)
## max 32 training epochs
resCnn <- spot(x = NULL,
  fun = funKerasMnist,
  lower = cfg$lower,
  upper = cfg$upper,
  control = list(funEvals=15,
    transformFun = cfg$transformations,
    types = cfg$type,
    noise = TRUE,
    plots = TRUE,
    progress = TRUE,
    seedFun = 1,
    seedSPOT = 1),
    kerasConf = kerasConf,
    data = mnist)
Description

Hyperparameter Tuning: Keras MNIST Classification Test Function.

Usage

funKerasMnist_0(x, kerasConf, data)

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 as described in getKerasConf. Default: kerasConf = getKerasConf().


Details

Trains a simple deep NN on the MNIST dataset. Provides a template that can be used for other networks as well. Standard Code from https://tensorflow.rstudio.com/ Modified by T. Bartz-Beielstein (tbb@bartzundbartz.de)

Value

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

See Also

getKerasConf
evalKerasMnist
fit

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,

PYTHON_RETICULATE <- FALSE
if(PYTHON_RETICULATE){

library("SPOTMisc")
library("SPOT")
kerasConf <- getKerasConf()

## The following two settings are default:
kerasConf$encoding = "oneHot"
kerasConf$model = "dl"
cfg <- getModelConf(kerasConf$model)
x <- matrix(cfg$default, nrow=1)
transformFun <- cfg$transformations
types <- cfg$type
lower <- cfg$lower
upper <- cfg$upper

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

### Use convnet:
kerasConf$encoding <- "tensor"
kerasConf$model <- "cnn"
funKerasMnist(x, kerasConf = kerasConf)

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

### Third example: spot call (dense network):
kerasConf$verbose <- 1
data <- getMnistData()
res <- spot(x = NULL,
    fun = funKerasMnist,
    lower = lower,
    upper = upper,
    control = list(funEvals=15,
        noise = TRUE,
        types = types,
        plots = TRUE,
        progress = TRUE,
        seedFun = 1,
        seedSPOT = 1),
    kerasConf = kerasConf,
    data = data)

### Fourth example: spot call (convnet):
kerasConf$verbose <- 1
kerasConf$encoding <- "tensor"
kerasConf$model <- "cnn"
data <- getMnistData(kerasConf)
res <- spot(x = NULL,
    fun = funKerasMnist,
    lower = lower,
    upper = upper,
    control = list(funEvals=15,
        noise = TRUE,
        types = types,
        plots = TRUE,
        progress = TRUE,
        seedFun = 1,
funKerasTransferLearning

Description

Hyperparameter Tuning: Keras TransferLearning Test Function.

Usage

funKerasTransferLearning(x, kerasConf = getKerasConf(), data = NULL)

Arguments

x \hspace{1cm} \text{matrix of hyperparameter values to evaluate with the function. Rows for points and columns for dimension.}

kerasConf \hspace{1cm} \text{List of additional parameters passed to keras as described in getKerasConf. Default: kerasConf = getKerasConf().}

data \hspace{1cm} \text{data}

Details

Trains a simple deep NN on the MNIST dataset. Provides a template that can be used for other networks as well. Standard Code from https://tensorflow.rstudio.com/ Modified by T. Bartz-Beielstein (tbb@bartzundbartz.de)

Value

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

See Also

getKerasConf
evalKerasTransferLearning
evalKerasMnist
fit
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,

```r
PYTHON_RETICULATE <- FALSE
if(PYTHON_RETICULATE){
  library("SPOTMisc")
  library("SPOT")
  kerasConf <- getKerasConf()

  # Hyperparameters:
  # "dropout" = x[1],
  # "learning_rate" = x[2],
  # "epochs" = x[3],
  # "beta_1" = x[4],
  # "beta_2" = x[5],
  # "epsilon" = x[6],
  # "optimizer" = x[7]

  lower <- c(1e-6, 1e-6, 2, 0.8, 0.8, 1e-9, 1)
  upper <- c(0.2, 1e-2, 5, 0.99, 0.9999, 1e-3, 2)
  types <- c("numeric", "numeric", "integer", "numeric", "numeric", "integer", "factor")

  ## First Example: spot call with extended verbosity. Default objective
  ## "validationLoss", i.e., validation loss, is used. only 20 function
  ## evaluations (for testing).
  kerasConf$verbose <- 1
  res <- spot(x = NULL,
              fun = funKerasTransferLearning,
              lower = lower,
              upper = upper,
              control = list(funEvals=20,
                              model=buildKriging,
                              noise = TRUE,
                              types = types,
                              optimizer=optimDE,
                              plots = TRUE,
                              progress = TRUE,
                              seedFun = 1,
                              seedSPOT = 1,
                              kerasConf = kerasConf)
  save(res, file = paste0("resKerasTransferLearning", as.numeric(Sys.time()),".RData"))
```

## Example: resKerasTransferLearning04
## Default objective function "validationLoss", i.e.,
## training loss
funKerasTransferLearning

library("SPOTMisc")
library("SPOT")
kerasConf <- getKerasConf()

# Hyperparameters:
# "dropout" = x[1],
# "learning_rate" = x[2],
# "epochs" = x[3],
# "beta_1" = x[4],
# "beta_2" = x[5],
# "epsilon" = x[6],
# "optimizer" = x[7]

lower <- c(1e-6, 1e-6, 2, 0.8, 0.8, 1e-9, 1)
upper <- c(0.2, 1e-2, 5, 0.99, 0.9999, 1e-3, 2)
types <- c("numeric", "numeric", "integer", "numeric", "numeric", "integer", "factor")

res <- spot(x = NULL,
    fun = funKerasTransferLearning,
    lower = lower,
    upper = upper,
    control = list(funEvals=100,
        model=buildKriging,
        noise = TRUE,
        types = types,
        optimizer=optimDE,
        plots = FALSE,
        progress = TRUE,
        seedFun = 1,
        seedSPOT = 1,
        kerasConf = kerasConf))

save(res,file = paste0("resKerasTransferLearningValidationLoss04",
    as.numeric(Sys.time()),".RData"))

## Example: resKerasTransferLearning05
## objective function "negValidationAccuracy", i.e.,
## negative validation accuracy

library("SPOTMisc")
library("SPOT")
kerasConf <- getKerasConf()

# Hyperparameters:
# "dropout" = x[1],
# "learning_rate" = x[2],
# "epochs" = x[3],
# "beta_1" = x[4],
# "beta_2" = x[5],
# "epsilon" = x[6],
# "optimizer" = x[7]
funKerasTransferLearning

lower <- c(1e-6, 1e-6, 2, 0.8, 0.8, 1e-9, 1)
upper <- c(0.2, 1e-2, 5, 0.99, 0.9999, 1e-3, 2)
types <- c("numeric", "numeric", "integer", "numeric", "numeric", "integer", "factor")

kerasConf$returnValue <- "negValidationAccuracy"
res <- spot(x = NULL,
  fun = funKerasTransferLearning,
  lower = lower,
  upper = upper,
  control = list(funEvals=100,
    model=buildKriging,
    noise = TRUE,
    types = types,
    optimizer=optimDE,
    plots = FALSE,
    progress = TRUE,
    seedFun = 1,
    seedSPOT = 1,
    kerasConf = kerasConf))
save(res, file = paste0("resKerasTransferLearningNegValidationAccuracy05",
  as.numeric(Sys.time()), ".RData"))

## Example: resKerasTransferLearning06
## objective function "trainingLoss", i.e.,
## training loss

library("SPOTMisc")
library("SPOT")
kerasConf <- getKerasConf()

# Hyperparameters:
# "dropout" = x[1],
# "learning_rate" = x[2],
# "epochs" = x[3],
# "beta_1" = x[4],
# "beta_2" = x[5],
# "epsilon" = x[6],
# "optimizer" = x[7]

lower <- c(1e-6, 1e-6, 2, 0.8, 0.8, 1e-9, 1)
upper <- c(0.2, 1e-2, 5, 0.99, 0.9999, 1e-3, 2)
types <- c("numeric", "numeric", "integer", "numeric", "numeric", "integer", "factor")

kerasConf$returnValue <- "trainingLoss"
res <- spot(x = NULL,
  fun = funKerasTransferLearning,
  lower = lower,
  upper = upper,
  control = list(funEvals=100,
    model=buildKriging,
genCatsDogsData

noise = TRUE,
types = types,
optimizer=optimDE,
plots = FALSE,
progress = TRUE,
seedFun = 1,
seedSPOT = 1,
kerasConf = kerasConf)

) save(res, file = paste0("resKerasTransferLearningTrainingLoss06",
as.numeric(Sys.time()),".RData"))

}

Description

Generate data for funKerasTransferLearning

Usage

    genCatsDogsData(kerasConf = getKerasConf())

Arguments

    kerasConf  kerass configuration. Default: kerasConf = getKerasConf

Details

    Standard Data from https://tensorflow.rstudio.com/ Modified by T. Bartz-Beielstein (tbb@bartzundbartz.de)

Value

    list with test, validation, and test data
genericDataPrep

Create an input pipeline using tfdatasets

Description

Create an input pipeline using tfdatasets

Usage

genericDataPrep(
  data,
  batch_size = 32,
  minLevelSizeEmbedding = 100,
  embeddingDim = NULL
)

Arguments

data data. List, e.g., df$trainCensus, df$testGeneric, and df$valCensus data)
batch_size batch size. Default: 32
minLevelSizeEmbedding integer. Embedding will be used for factor variables with more than minLevelSizeEmbedding levels. Default: 100.
embeddingDim integer. Dimension used for embedding. Default: floor(log(minLevelSizeEmbedding)).

Value

a fitted FeatureSpec object and the hold-out testGeneric (=data$testGeneric). This is returned as the following list.

train_ds_generic train
val_ds_generic validation
test_ds_generic test
specGeneric_prep feature spec object
testGeneric data$testGeneric

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,
PYTHON_RETICULATE <- FALSE
if(PYTHON_RETICULATE){
target <- "age"
getDataCensus <- function(target = NULL, nobs = 1000, cachedir = "oml.cache",
                        prop = 2/3, cache.only = FALSE) {
  dfCensus <- getDataCensus(task = target, nobs = nobs, cachedir = cachedir, cache.only = FALSE)
  data <- getGenericTrainValTestData(dfGeneric = dfCensus, prop = prop)
  specList <- genericDataPrep(data = data, batch_size = batch_size)
  ## Call iterator:
  require(magrittr)
  specList$train_ds_generic %>%
    reticulate::as_iterator() %>%
    reticulate::iter_next()
}

### Description

This function downloads (or loads from cache folder) the Census KDD Dataset (OpenML ID: 4535). If requested, data set is changed w.r.t the number of observations, number of numerical/categorical feature, the cardinality of the categorical features, and the task type (regr. or classif).

### Usage

```r
getDataCensus(
  task.type = "classif",
  nobs = 50000,
  nfactors = "high",
  nnumericals = "high",
  cardinality = "high",
  data.seed = 1,
  cachedir = "oml.cache",
  target = NULL,
  cache.only = FALSE
)
```

### Arguments

- **task.type** character, either "classif" or "regr".
- **nobs** integer, number of observations uniformly sampled from the full data set.
- **nfactors** character, controls the number of factors (categorical features) to use. Can be "low", "med", "high", or "full" (full corresponds to original data set).
- **nnumericals** character, controls the number of numerical features to use. Can be "low", "med", "high", or "full" (full corresponds to original data set).
cardinality character, controls the number of factor levels (categories) for the categorical features. Can be "low", "med", "high" (high corresponds to original data set).

data.seed integer, this will be used via set.seed() to make the random subsampling reproducible. Will not have an effect if all observations are used.
cachedir character. The cache directory, e.g., "oml.cache". Default: "oml.cache".
target character "age" or "income_class". If target = age, the numerical variable age is converted to a factor: age<-as.factor(age<40)
cache.only logical. Only try to retrieve the object from cache. Will result in error if the object is not found. Default is TRUE.

Value
census data set

Examples

```r
## Example downloads OpenML data, might take some time:
task.type <- "classif"
nobs <- 1e4 # max: 229285
data.seed <- 1
nfactors <- "full"
nnumericals <- "low"
cardinality <- "med"
censusData <- getDataCensus(
  task.type = task.type,
  nobs = nobs,
  nfactors = nfactors,
  nnumericals = nnumericals,
  cardinality = cardinality,
  data.seed = data.seed,
  cachedir = "oml.cache",
  target="age")
```

getGenericTrainValTestData
genericTrainTestValData

description
genericTrainTestValData

Usage

genericTrainTestValData(dfGeneric = NULL, prop = 0.5)
getGenericTrainValTestData

Arguments

- **dfGeneric**: data, e.g., obtained with `getDataCensus`. Default: NULL.
- **prop**: vector. proportion between train / test and train/val. Default: 2/3. If one value is given, the same proportion will be used for both split. Otherwise, the first entry is used for the test/training split and the second value for the training/validation split. If the second value is 1, the validation set is empty. Given prop = (p1, p2), the data will be partitioned as shown in the following two steps:
  - **Step 1**: train1 = p1*data and test = (1-p1)*data
  - **Step 2**: train2 = p2*train1 = p2*p1*data and val = (1-p2)*train1 = (1-p2)*p1*data

Value

list with training, validation and test data: trainCensus, valCensus, testCensus.

Note

If p2=1, no validation data will be generated.

See Also

- `getKerasConf`
- `funKerasGeneric`
- `getDataCensus`

Examples

```r
### These examples require an activated Python environment as described in
PYTHON_RETICULATE <- FALSE
if(PYTHON_RETICULATE){
  task.type <- "classif"
  nobs <- 1e4
  nfactors <- "high"
  nnumericals <- "high"
  cardinality <- "high"
  data.seed <- 1
  cachedir <- "oml.cache"
  target = "age"
  prop <- 2/3
  dfCensus <- getDataCensus(task.type = task.type,
                           nobs = nobs,
                           nfactors = nfactors,
                           nnumericals = nnumericals,
                           cardinality = cardinality,
                           data.seed = data.seed,
                           cachedir = cachedir,
                           target = target)
  census <- getGenericTrainValTestData(dfGeneric=dfCensus,
                                       prop = prop)
}
## train data size is 2/3*2/3*10000:
dim(census$trainGeneric)
}

---

### getIndices

*Get indices (positions) of variable names*

**Description**

Get indices (positions) of variable names

**Usage**

`getIndices(model, a)`

**Arguments**

- `model` from `getModelConf`, e.g., "dl".
- `a` name of variables

**Value**

indices of variable names.

**Examples**

`getIndices(model="dl", 
a = c("dropout", "units"))`

---

### getKerasConf

*Get keras configuration parameter list*

**Description**

Configuration list for keras’s `fit` function.

**Usage**

`getKerasConf()`
Details

Additional parameters passed to keras, e.g.,

activation: character. Activation function in the last layer. Default: "sigmoid".

active: vector of active variables, e.g., c(1,10) specifies that the first and tenth variable will be
considered by spot.

callbacks: List of callbacks to be called during training. Default: list().
clearSession: logical. Whether to call k_clear_session or not at the end of keras modelling.
                  Default: FALSE.
encoding: character. Encoding used during data preparation, e.g., by getMnistData. Default:
                  "oneHot".
loss: character. Loss function for compile. Default: "loss_binary_crossentropy".
model: model specified via getModelConf. Default: "dl".
nClasses: Number of classes in (multi-class) classification. Specifies the number of units in the
          last layer (before softmax). Default: 1 (binary classification).
resDummy: logical. If TRUE, generate dummy (mock up) result for testing. If FALSE, run keras and
          tf evaluations. Default: FALSE.
returnValue: Return value. Can be one of "trainingLoss", "negTrainingAccuracy", "validationLoss",
              "negValidationAccuracy", "testLoss", or "negTestAccuracy".
returnObject: Return object. Can be one of "evaluation", "model", "pred". Default: "evaluation".
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. Default: FALSE.
testData: Test Data on which to evaluate the loss and any model metrics at the end of the optimi-
          zation using evaluate().
tfDevice: Tensorflow device. CPU/GPU allocation. Passed to tensorflow via tf$device(kerasConf$tfDevice).
          Default: "/cpu:0" (use CPU only).
trainData: Train Data on which to evaluate the loss and any model metrics at the end of each
          epoch.
validationData: Validation Data on which to evaluate the loss and any model metrics at the end of
                 each epoch.
validation_data (deprecated, see validationData): 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. Default: NULL.
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. Default:
          0.2.
verbose: Verbosity mode (0 = silent, 1 = progress bar, 2 = one line per epoch). Default: 0.
Value

kerasConf list with configuration parameters.

See Also

evalKerasMnist
funKerasMnist
fit

getMlConfig  
get ml config for keras on census

Description

gem ml config for keras on census

Usage

gemlConfig(
  target,
  model,
  data,
  task.type,
  nobs,
  nfactors,
  nnumericals,
  cardinality,
  data.seed,
  prop
)

Arguments

target character "age" or "income_class"
model character model name, e.g., "dl"
data data, e.g., from getDataCensus
task.type "classif" (character)
nobs number of observations (numerical), max 229285. Default: 1e4
nfactors (character), e.g., "high"
nnumericals (character), e.g., "high"
cardinality (character), e.g., "high"
data.seed (numerical) seed
prop (numerical) split proportion (train, vals,test)
getMlrResample

Value

cfg (list)

Examples

```r
# These examples require an activated Python environment as described in
# Model Based Hyperparameter Tuning for Deep Learning with SPOT,

if(!PYTHON_RETICULATE) {
  target <- "age"
  task.type <- "classif"
  nobs <- 1e2
  nfactors <- "high"
  nnumericals <- "high"
  cardinality <- "high"
  data.seed <- 1
  cachedir <- "oml.cache"
  model <- "ranger"

  dfCensus <- getDataCensus(
    task.type = task.type,
    nobs = nobs,
    nfactors = nfactors,
    nnumericals = nnumericals,
    cardinality = cardinality,
    data.seed = data.seed,
    cachedir = cachedir,
    target = target)

  cfg <- getMlConfig(
    target = target,
    model = model,
    data = dfCensus,
    task.type = task.type,
    nobs = nobs,
    nfactors = nfactors,
    nnumericals = nnumericals,
    cardinality = cardinality,
    data.seed = data.seed,
    prop = 2/3)
}
```

generateFixedInstance
Description

Determines test/train split and applies makeFixedHoldoutInstance

Usage

getMlrResample(task, dataset, data.seed = 1, prop = NULL)

Arguments

task mlr task
dataset e.g., census data set
data.seed seed
prop proportion, e.g., 2/3 take 2/3 of the data for training and 1/3 for test

Value

list: an mlr resample generated with makeFixedHoldoutInstance

See Also

getMlrTask

Examples

## Example downloads OpenML data, might take some time:
dataset <- getDataCensus(  
task.type="classif",  
nobs = 1e3,  
nfactors = "high",  
nnumericals = "high",  
cardinality = "high",  
data.seed=1,  
cachedir = "oml.cache",  
target = "age")

taskdata <- getMlrTask(dataset,  
task.type = "classif",  
data.seed = 1)

rsmpl <- getMlrResample(task = taskdata,  
dataset = dataset,  
data.seed = 1,  
prop = 2/3)
getMlrTask

Generate an mlr task from Census KDD data set (+variation)

Description

Prepares the Census data set for mlr. Performs imputation via: factor = imputeMode(), integer = imputeMedian(), numeric = imputeMean()

Usage

getMlrTask(dataset, task.type = "classif", data.seed = 1)

Arguments

dataset  census data set

task.type character, either "classif" or "regr".

data.seed seed

Value

an mlr task with the respective data set. Generated with makeClassifTask or makeRegrTask for classification and regression respectively.

See Also

getDataCensus

Examples

```r
## Example downloads OpenML data, might take some time:
x <- getDataCensus(
  task.type="classif",
  nobs = 1e3,
  nfactors = "high",
  nnumericals = "high",
  cardinality = "high",
  data.seed=1,
  cachedir = "oml.cache",
  target = "age")

taskdata <- getMlrTask(
  dataset = x,
  task.type = "classif",
  data.seed = 1)
```
getMnistData

Description

Based on the setting `kerasConf$encoding` either one-hot encoded data or tensor-shaped data are returned. The labels are converted to binary class matrices using the function `to_categorical`.

Usage

getMnistData(kerasConf)

Arguments

- `kerasConf`: List of additional parameters passed to keras as described in `getKerasConf`. Default: NULL.

Value

list with training and test data, i.e., list(x_train, x_test, y_train, y_test).

See Also

- `getKerasConf`
- `funKerasMnist`

Examples

```r
### 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,

PYTHON_RETICULATE <- FALSE
if(PYTHON_RETICULATE){
  library("SPOTMisc")
  kerasConf <- getKerasConf()
  kerasConf$encoding <- "oneHot" # default
  mnist <- getMnistData(kerasConf)
  # lots of zeros, but there are also some nonzero (greyscale) values, e.g.:
  mnist$x_train[1,150:160]
  str(mnist$x_train[1,])
  # y-labels are one-hot encoded. The first entry represents "5"
  mnist$y_train[1,]
  ##
  kerasConf$encoding <- "tensor"
  mnist <- getMnistData(kerasConf)
  ## 28x28:
```
getModelConf

getModelConf

Get model configuration

Description

Configure machine and deep learning models

Usage

getModelConf(
  modelArgs = NULL,
  model,
  task.type = NULL,
  nFeatures = NULL,
  active = NULL
)

Arguments

modelArgs list with information about model, active variables etc. Note: argList will replace the other arguments. Use argList$model instead of model etc.

model machine or deep learning model (character). One of the following:
  "cvglmnet" glm net.
  "kknn" nearest neighbour.
  "ranger" random forest.
  "rpart" recursive partitioning and regression trees, rpart
  "svm" support vector machines.
  "xgboost" gradient boosting, xgb.train.
  "dl" deep learning: dense network.
  "cnn" deep learning: convolutionary network.

task.type character, either "classif" or "regr".
nFeatures number of features, e.g., sum(task$task.desc$n.feat)
active vector of activated tunepars, e.g., c("minsplit", "maxdepth") for model "rpart"

Value

Returns returns a list of the machine learning model configuration and corresponding hyperparameters:

learner character: combination of task.type and model name.
lower vector of lower bounds.
upper vector of upper bounds.
fixpars list of fixed parameters.
factorlevels list of factor levels.
transformations vector of transformations.
dummy logical. Use dummy encoding, e.g., \texttt{xgb.train}
relpars list of relative hyperparameters.

Examples

```r
# Get hyperparameter names and their defaults for fitting a
# (recursive partitioning and regression trees) model:
modelArgs <- list(model = "rpart")
cfg <- getModelConf(modelArgs)
cfg$tunepars
cfg$defaults
## do not use anymore:
cfg <- getModelConf(model="rpart")
cfg$tunepars
cfg$defaults
modelArgs <- list(model="rpart", active = c("minsplit", "maxdepth"))
cfgAct <- getModelConf(modelArgs)
cfgAct$tunepars
cfgAct$defaults
```

---

**getObjf**

Get objective function for mlr

Description

mlrTools This function receives a configuration for a tuning experiment, and returns an objective function to be tuned via SPOT. It basically provides the result from a call to \texttt{resample}: \texttt{resample(lrn, task, resample, measures = measures, show.info = FALSE)}, with measures defined as \texttt{mmce} for classification and \texttt{rmse} for regression, \texttt{timeboth}, \texttt{timetrain}, and \texttt{timepredict}.

Usage

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

Arguments

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

**Details**

Parameter names are set, parameters are transformed to the actual parameter scale, integer levels are converted to factor levels for categorical parameters, and parameters are set in relation to other parameters.

**Value**

an objective function that can be optimized via **spot**.

---

**getPredf**

Get predictions from mlr

**Description**

mlrTools This function receives a configuration for a tuning experiment, and returns predicted values.

**Usage**

getPredf(config, timeout = 3600)

**Arguments**

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

**Value**

an prediction function that can be called via **spot**. It basically provides the result from a call to **resample**: resample(lrn, task, resample, measures = measures, show.info = FALSE).

---

**getSimpleKerasModel**

getSimpleKerasModel

**Description**

build, compile, and train a simple model (for testing)

**Usage**

getSimpleKerasModel(specList, kerasConf = getKerasConf())

**Arguments**

- **specList**: spec
- **kerasConf**: keras configuration. Default: return value from **getKerasConf**.
### getVarNames

Get variable names or subsets of variable names

#### Description

Get variable names or subsets of variable names

#### Usage

```r
getVarNames(model, i = "all")
```

#### Arguments

- **model**: from `getModelConf`, e.g., "dl".
- **i**: index for selecting subsets. Default is "all".

#### Value

vector of variable names. Returns NA if wrong indices are selected.
Examples

# Default is return all:
getVarNames(model="dl")
getVarNames(model="dl", i=3)
getVarNames(model="dl", i=c(1,3,5))
# var name does not exits, so return NA
getVarNames(model="dl", i=c(100))

Description

Build data frame for ggparcoord (parallel plot)

Usage

```r
ggparcoordPrepare(
  x,
  y,
  xlab = NULL,
  ylab = NULL,
  probs = seq(0.25, 0.75, 0.25),
  yrange = NULL
)
```

Arguments

- `x`: elements x, e.g., result from a `spot` run.
- `y`: associated function values
- `xlab`: character, the value of the independent variable
- `ylab`: character, the value of the dependent variable predicted by the corresponding model.
- `probs`: quantile probabilities. Default: `seq(0, 1, 0.25)`
- `yrange`: y interval

Value

data frame for `ggparcoord`
Examples

```r
require(SPOT)
require(GGally)
n <- 4 # param
k <- 50 # samples
x <- designUniformRandom(rep(0,n), rep(1,n), control=list(size=k))
y <- matrix(0, nrow=k,ncol=1)
y <- funSphere(x)
result <- list(x=x, y=y)
df <- ggparcoordPrepare(x=result$x,
                         y=result$y,
                         xlab=result$control$parNames,
                         probs = c(0.25, 0.5, 0.75))
                          #probs = c(0.1, 0.9)) )#seq(0.25, 1, 0.25))
m <- ncol(df)
splineFactor <- max(1, floor(2*m))
ggparcoord(data=df, columns = 1:(m-2), groupColumn = m,
scale = "uniminmax", boxplot = FALSE, alphaLines = 0.2,showPoints = TRUE)
##
require(SPOT)
require(GGally)
result <- spot(x=NULL,
               fun=funSphere,
               lower=rep(-1,3),
               upper= rep(1,3),
               control=list(funEvals=20,
                             model=buildKriging,
                             modelControl=list(target="y")))
df <- ggparcoordPrepare(x=result$x,
                         y=result$y,
                         xlab=result$control$parNames,
                         probs = c(0.25, 0.5, 0.75))
                          #c(0.9,0.95) )#seq(0.25, 1, 0.25))
m <- ncol(df)
splineFactor <- max(1, floor(2*m))
ggparcoord(data=df, columns = 1:(m-2), groupColumn = m,
splineFactor = splineFactor, scale = "uniminmax",
boxplot = FALSE, alphaLines = 0.2,showPoints = TRUE, scaleSummary = "median")
```

Description

simple progress plot
Usage

```r
ggplotProgress(
  dfRun,
  xlabel = "function evaluations",
  ylabel = "MMCE",
  aspectRatio = 2,
  scalesFreeFixed = "free_y",
  nColumns = 3
)
```

Arguments

- `dfRun` data frame, e.g., result from `prepareProgressPlot`.
- `xlabel` x label
- `ylabel` y label
- `aspectRatio` aspect.ratio
- `scalesFreeFixed` "free_x", "free_y" or "fixed"
- `nColumns` number of columns

Value

- `p ggplot`

---

`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

```r
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
kerasBuildCompile evaluKerasGeneric model building and compile

Description

Hyperparameter Tuning: Keras Generic Classification Function.

Usage

kerasBuildCompile(FLAGS, kerasConf, specList)

Arguments

FLAGS flags. list of hyperparameter values. If NULL, a simple keras model will be build, which is considered default (see getSimpleKerasModel).
kerasConf List of additional parameters passed to keras as described in getKerasConf. Default: kerasConf = getKerasConf().
specList prepared data. See genericDataPrep. See getGenericTrainValTestData.

Details

Trains a simple deep NN on a generic data set. Standard Code from https://tensorflow.rstudio.com/ Modified by T. Bartz-Beielstein (tbb@bartzundbartz.de)

Value

list with function values (training, validation, and test loss/accuracy, and keras model information)

See Also

genericDataPrep
funKerasGeneric
fit

kerasCompileResult Generate result from keras run

Description

Compile a matrix with training, validation, and test results

Usage

kerasCompileResult(y, kerasConf)
Arguments

y (1x6)-dim matrix with the following entries: trainingLoss, negTrainingAccuracy, validationLoss, negValidationAccuracy, testLoss, and negTestAccuracy.

kerasConf keras configuration generated with getKerasConf

Details

All values should be minimized: accuracies will be negative. The (1x7)-dim result matrix has the following entries

returnValue: Metric used for optimization. Default: "validationLoss".
trainingLoss: training loss.
negTrainingAccuracy: negative training accuracy.
validationLoss: validation loss.
negValidationAccuracy: negative validation accuracy.
testLoss: test loss.
negTestAccuracy: negative test accuracy.

Value

result matrix

See Also

evalKerasMnist
gfunKerasMnist

Examples

x <- 1
testLoss <- x
negTestAccuracy <- 1-x
validationLoss <- x/2
negValidationAccuracy <- 1-x/2
trainingLoss <- x/3
negTrainingAccuracy <- 1-x/3
y <- matrix(c(trainingLoss, negTrainingAccuracy, validationLoss, negValidationAccuracy, testLoss, negTestAccuracy), 1,6)
kerasConf <- list()
kerasConf$returnValue <- "testLoss"
sum(kerasCompileResult(y, kerasConf)) == 4
kerasConf$returnValue <- "negTestAccuracy"
sum(kerasCompileResult(y, kerasConf)) == 3
kerasConf$returnValue <- "validationLoss"
sum(kerasCompileResult(y, kerasConf))*2 == 7
kerasConf$returnValue <- "negValidationAccuracy"
sum(kerasCompileResult(y, kerasConf))*2 == 7
kerasConf$returnValue <- "trainingLoss"
sum(kerasCompileResult(y, kerasConf)) * 3 == 10
kerasConf$returnValue <- "negTrainingAccuracy"
sum(kerasCompileResult(y, kerasConf)) * 3 == 11

kerasEvalPrediction  Evaluate keras prediction

Description
Evaluates prediction from keras model using several metrics based on training, validation and test data

Usage
kerasEvalPrediction(pred, testScore = c(NA, NA), specList, metrics, kerasConf)

Arguments
- pred: prediction from keras predict
- testScore: additional score values
- specList: spec with target
- metrics: keras metrics (history)
- kerasConf: keras config

Examples
### These examples require an activated Python environment as described in
PYTHON_RETICULATE <- FALSE
if(PYTHON_RETICULATE){
library(tfdatasets)
library(keras)
target <- "age"
batch_size <- 32
prop <- 2/3
dfCensus <- getDataCensus(nobs=1000,
                          target = target)
data <- getGenericTrainValTestData(dfGeneric = dfCensus,
                                     prop = prop)
specList <- genericDataPrep(data=data, batch_size = batch_size)
# spec test data has 334 elements:
str(specList$testGeneric$target)
# simulate test:
pred <- runif(length(specList$testGeneric$target))
kerasConf <- getKerasConf()
simpleModel <- getSimpleKerasModel(specList=specList,  
    kerasConf=kerasConf)
FLAGS <- list(epochs=16)
y <- kerasFit(model=simpleModel,  
    specList = specList,  
    FLAGS = FLAGS,  
    kerasConf = kerasConf)
simpeModel <- y$model
history <- y$history
# evaluate on test data
pred <- predict(simpleModel, specList$testGeneric)
## in use keras evaluation (test error):
testScore <-
    keras::evaluate(simpleModel,  
        tfdatasets::dataset_use_spec(dataset=specList$test_ds_generic,  
            spec=specList$specGeneric_prep),  
        verbose = kerasConf$verbose)
kerasEvalPrediction(pred=pred,  
    testScore = testScore,  
    specList = specList,  
    metrics = history$metrics,  
    kerasConf = kerasConf  
)

### kerasFit

**kerasFit**  

**Description**  

Hyperparameter Tuning: Keras Generic Classification Function.

**Usage**  

kerasFit(model, specList, FLAGS, kerasConf)

**Arguments**  

- **model**: model If NULL, a simple keras model will be build, which is considered default (see getSimpleKerasModel).
- **specList**: prepared data. See genericDataPrep. See getGenericTrainValTestData.
- **FLAGS**: flags, see also mapX2FLAGS
- **kerasConf**: List of additional parameters passed to keras as described in getKerasConf. Default: kerasConf = getKerasConf().
kerasReturnDummy

Details

Trains a simple deep NN on a generic data set. Standard Code from https://tensorflow.rstudio.com/ Modified by T. Bartz-Beielstein (tbb@bartzundbartz.de)

Value

list with function values (training, validation, and test loss/accuracy, and keras model information)

See Also

getKerasConf
funKerasGeneric
fit

Description

Return dummy values

Usage

kerasReturnDummy(kerasConf)

Arguments

kerasConf keras configuration list

Value

y row matrix of random (uniformly distributed) return values

Examples

kerasConf <- getKerasConf()
kerasReturnDummy(kerasConf)
**makeLearnerFromHyperparameters**

*Make mlr learner from conf and hyperparameter vector*

**Description**
calls makelearner

**Usage**

```r
makeLearnerFromHyperparameters(x = NULL, cfg = NULL)
```

**Arguments**

- `x` hyperparameter vector
- `cfg` configuration list

**Value**

mlr learner

---

**mapX2FLAGS**

*Map x parameters to a list of named values*

**Description**

numerical parameters are mapped to their meanings, e.g., `x[1]` to "dropout rate".

**Usage**

```r
mapX2FLAGS(x, model = "dl")
```

**Arguments**

- `x` matrix input values.
- `model` (char) network type, e.g., "cnn" or "dl". Default: "dl"
Details

For a "dl" network, the parameter vector x is mapped to the following FLAGS:

- x[1]: dropout dropout rate first layer.
- x[2]: dropoutfac dropout factor (multiplier).
- x[3]: units number of units in the first layer.
- x[4]: unitsfact units factor (multiplier).
- x[5]: learning_rate learning rate for optimizer. See, e.g.: link{optimizer_sgd}
- x[6]: epochs number of training epochs.
- x[7]: beta_1 The exponential decay rate for the 1st moment estimates. float, 0 < beta < 1. Generally close to 1.
- x[8]: beta_2 The exponential decay rate for the 2nd moment estimates. float, 0 < beta < 1. Generally close to 1.
- x[9]: layers number of layers.
- x[10]: epsilon float >= 0. Fuzz factor. If NULL, defaults to k_epsilon().

Value

FLAGS named list (parameter names as specified in getModelConf), e.g., for "dl": dropout, dropoutfac, units, unitsfact, learning_rate, epochs, beta_1, beta_2, layers, epsilon, optimizer

Examples

```r
## First example: dense neural net
x <- getModelConf(list(model="dl"))$defaults
mapX2FLAGS(x=x, model = "dl")
## Second example: convnet
x <- getModelConf(list(model="cnn"))$defaults
mapX2FLAGS(x=x, model = "cnn")
```

---

**MSE**

*mean squared errors*

Description

mean squared errors

Usage

`MSE(y, yhat)`
optimizer_adadelta

Arguments

<table>
<thead>
<tr>
<th>y</th>
<th>actual value</th>
</tr>
</thead>
<tbody>
<tr>
<td>yhat</td>
<td>predicted value</td>
</tr>
</tbody>
</table>

Value

mean squared errors

---

**optimizer_adadelta**  
**Adadelta optimizer.**

Description


Usage

```r
optimizer_adadelta(
  learning_rate = 0,
  rho = 0.95,
  epsilon = NULL,
  decay = 0,
  clipnorm = NULL,
  clipvalue = NULL,
  ...
)
```

Arguments

<table>
<thead>
<tr>
<th>learning_rate</th>
<th>float ≥ 0. Learning rate.</th>
</tr>
</thead>
<tbody>
<tr>
<td>rho</td>
<td>float ≥ 0. Decay factor.</td>
</tr>
<tr>
<td>epsilon</td>
<td>float ≥ 0. Fuzz factor. If ‘NULL’, defaults to ‘k_epsilon()’.</td>
</tr>
<tr>
<td>decay</td>
<td>float ≥ 0. Learning rate decay over each update.</td>
</tr>
<tr>
<td>clipnorm</td>
<td>Gradients will be clipped when their L2 norm exceeds this value.</td>
</tr>
<tr>
<td>clipvalue</td>
<td>Gradients will be clipped when their absolute value exceeds this value.</td>
</tr>
<tr>
<td>...</td>
<td>Unused, present only for backwards compatibility</td>
</tr>
</tbody>
</table>

Note

To enable compatibility with the ranges of the learning rates of the other optimizers, the learning rate `learning_rate` is internally mapped to `1 - learning_rate`. That is, a learning rate of 0 will be mapped to 1 (which is the default.) It is recommended to leave the parameters of this optimizer at their default values.
optimizer_adagrad

See Also

Other optimizers: optimizer_adagrad(), optimizer_adamax(), optimizer_adam(), optimizer_nadam(), optimizer_rmsprop(), optimizer_sgd()

optimizer_adagrad  Adagrad optimizer

Description

Adagrad optimizer as described in [Adaptive Subgradient Methods for Online Learning and Stochastic Optimization](https://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf).

Usage

optimizer_adagrad(
  learning_rate = 0.01,
  epsilon = NULL,
  decay = 0,
  clipnorm = NULL,
  clipvalue = NULL,
  ...
)

Arguments

learning_rate  float >= 0. Learning rate.
epsilon  float >= 0. Fuzz factor. If ‘NULL’, defaults to ‘k_epsilon()’.
decay  float >= 0. Learning rate decay over each update.
clipnorm  Gradients will be clipped when their L2 norm exceeds this value.
clipvalue  Gradients will be clipped when their absolute value exceeds this value.
...  Unused, present only for backwards compatibility

Note

To enable compatibility with the ranges of the learning rates of the other optimizers, the learning rate learning_rate is internally mapped to 10 * learning_rate. That is, a learning rate of 0.001 will be mapped to 0.01 (which is the default.)

See Also

Other optimizers: optimizer_adadelta(), optimizer_adamax(), optimizer_adam(), optimizer_nadam(), optimizer_rmsprop(), optimizer_sgd()
optimizer_adam

Adam optimizer

Description


Usage

```r
optimizer_adam(
  learning_rate = 0.001,
  beta_1 = 0.9,
  beta_2 = 0.999,
  epsilon = NULL,
  decay = 0,
  amsgrad = FALSE,
  clipnorm = NULL,
  clipvalue = NULL,
  ...
)
```

Arguments

- **learning_rate**: float >= 0. Learning rate.
- **beta_1**: The exponential decay rate for the 1st moment estimates. float, 0 < beta < 1. Generally close to 1.
- **beta_2**: The exponential decay rate for the 2nd moment estimates. float, 0 < beta < 1. Generally close to 1.
- **epsilon**: float >= 0. Fuzz factor. If ‘NULL’, defaults to ‘k_epsilon()’.
- **decay**: float >= 0. Learning rate decay over each update.
- ** AMSGrad**: Whether to apply the AMSGrad variant of this algorithm from the paper "On the Convergence of Adam and Beyond".
- **clipnorm**: Gradients will be clipped when their L2 norm exceeds this value.
- **clipvalue**: Gradients will be clipped when their absolute value exceeds this value.
- **...**: Unused, present only for backwards compatibility

References


Note

Default parameters follow those provided in the original paper.
optimizer_adamax

**See Also**
Other optimizers: optimizer_adadelta(), optimizer_adagrad(), optimizer_adamax(), optimizer_nadam(), optimizer_rmsprop(), optimizer_sgd()

---

**optimizer_adamax**  
*Adamax optimizer*

**Description**
Adamax optimizer from Section 7 of the [Adam paper](https://arxiv.org/abs/1412.6980v8). It is a variant of Adam based on the infinity norm.

**Usage**
```
optimizer_adamax(
    learning_rate = 0.002,
    beta_1 = 0.9,
    beta_2 = 0.999,
    epsilon = NULL,
    decay = 0,
    clipnorm = NULL,
    clipvalue = NULL,
    ...
)
```

**Arguments**
- **learning_rate** float >= 0. Learning rate.
- **beta_1** The exponential decay rate for the 1st moment estimates. float, 0 < beta < 1. Generally close to 1.
- **beta_2** The exponential decay rate for the 2nd moment estimates. float, 0 < beta < 1. Generally close to 1.
- **epsilon** float >= 0. Fuzz factor. If 'NULL', defaults to 'k_epsilon()'.
- **decay** float >= 0. Learning rate decay over each update.
- **clipnorm** Gradients will be clipped when their L2 norm exceeds this value.
- **clipvalue** Gradients will be clipped when their absolute value exceeds this value.
- **...** Unused, present only for backwards compatibility

**Note**
To enable compatibility with the ranges of the learning rates of the other optimizers, the learning rate `learning_rate` is internally mapped to 2 * `learning_rate`. That is, a learning rate of 0.001 will be mapped to 0.002 (which is the default.)
optimizer_nadam

See Also

Other optimizers: optimizer_adadelta(), optimizer_adagrad(), optimizer_adam(), optimizer_nadam(), optimizer_rmsprop(), optimizer_sgd()

---

**optimizer_nadam**  
*Nesterov Adam optimizer*

Description

Much like Adam is essentially RMSprop with momentum, Nadam is Adam RMSprop with Nesterov momentum.

Usage

```r
optimizer_nadam(  
  learning_rate = 0.002,  
  beta_1 = 0.9,  
  beta_2 = 0.999,  
  epsilon = NULL,  
  schedule_decay = 0.004,  
  clipnorm = NULL,  
  clipvalue = NULL,  
  ...  
)
```

Arguments

- **learning_rate**: float > 0. Learning rate.
- **beta_1**: The exponential decay rate for the 1st moment estimates. float, 0 < beta < 1. Generally close to 1.
- **beta_2**: The exponential decay rate for the 2nd moment estimates. float, 0 < beta < 1. Generally close to 1.
- **epsilon**: float > 0. Fuzz factor. If `NULL`, defaults to `k_epsilon()`.
- **schedule_decay**: Schedule decay.
- **clipnorm**: Gradients will be clipped when their L2 norm exceeds this value.
- **clipvalue**: Gradients will be clipped when their absolute value exceeds this value.
- ... Unused, present only for backwards compatibility

Details

Default parameters follow those provided in the paper.
optimizer_rmsprop

Note

To enable compatibility with the ranges of the learning rates of the other optimizers, the learning rate `learning_rate` is internally mapped to $2 \times learning_rate$. That is, a learning rate of 0.001 will be mapped to 0.002 (which is the default.)

See Also


Other optimizers: optimizer_adadelta(), optimizer_adagrad(), optimizer_adamax(), optimizer_adam(), optimizer_rmsprop(), optimizer_sgd()

optimizer_rmsprop  
RMSProp optimizer

Description

RMSProp optimizer

Usage

```
optimizer_rmsprop(
  learning_rate = 0.001,
  rho = 0.9,
  epsilon = NULL,
  decay = 0,
  clipnorm = NULL,
  clipvalue = NULL,
  ...
)
```

Arguments

- `learning_rate`  float >= 0. Learning rate.
- `rho`  float >= 0. Decay factor.
- `epsilon`  float >= 0. Fuzz factor. If ’NULL’, defaults to ‘k_epsilon()’.
- `decay`  float >= 0. Learning rate decay over each update.
- `clipnorm`  Gradients will be clipped when their L2 norm exceeds this value.
- `clipvalue`  Gradients will be clipped when their absolute value exceeds this value.
- `...`  Unused, present only for backwards compatibility

Note

This optimizer is usually a good choice for recurrent neural networks.
Optimizer SGD

See Also

Other optimizers: optimizer_adadelta(), optimizer_adagrad(), optimizer_adamax(), optimizer_adam(), optimizer_nadam(), optimizer_sgd()

---

**optimizer_sgd**

*Stochastic gradient descent (SGD) optimizer*

**Description**

Stochastic gradient descent optimizer with support for momentum, learning rate decay, and Nesterov momentum.

**Usage**

```r
optimizer_sgd(
  learning_rate = 0.01,
  momentum = 0,
  decay = 0,
  nesterov = FALSE,
  clipnorm = NULL,
  clipvalue = NULL,
  ...
)
```

**Arguments**

- **learning_rate**: float >= 0. Learning rate.
- **momentum**: float >= 0. Parameter that accelerates SGD in the relevant direction and dampens oscillations.
- **decay**: float >= 0. Learning rate decay over each update.
- **nesterov**: boolean. Whether to apply Nesterov momentum.
- **clipnorm**: Gradients will be clipped when their L2 norm exceeds this value.
- **clipvalue**: Gradients will be clipped when their absolute value exceeds this value.
- **...**: Unused, present only for backwards compatibility

**Details**

Based on: [keras/R/optimizers.R](https://github.com/rstudio/keras/blob/main/R/optimizers.R). The following code is commented: backcompat_fix_rename_lr_to_learning_rate(...)

**Value**

Optimizer for use with compile.keras.engine.training.Model.
Note

To enable compatibility with the ranges of the learning rates of the other optimizers, the learning rate `learning_rate` is internally mapped to $10 \times learning_rate$. That is, a learning rate of 0.001 will be mapped to 0.01 (which is the default.)

See Also

Other optimizers: `optimizer_adadelta()`, `optimizer_adagrad()`, `optimizer_adamax()`, `optimizer_adam()`, `optimizer_nadam()`, `optimizer_rmsprop()`

---

**plotParallel**

*Parallel coordinate plot of a data set*

**Description**

Parallel plot based on `ggparcoord`.

**Usage**

```r
plotParallel(
  result,
  xlab = NULL,
  ylab = NULL,
  yrange = NULL,
  splineFactor = 1,
  colorOption = "A",
  scale = "uniminmax",
  boxplot = FALSE,
  alphaLines = 0.1,
  showPoints = TRUE,
  title = "",
  probs = seq(0.25, 0.75, 0.25),
  ...
)
```

**Arguments**

- `result` the result list returned by `spot`, importantly including the data `x`, `y`.
- `xlab` character, the value of the independent variable
- `ylab` character, the value of the dependent variable predicted by the corresponding model.
- `yrange` a two-element vector that specifies the range of `y` values to consider (data outside of that range will be excluded).
plotSensitivity

splineFactor logical or numeric operator indicating whether spline interpolation should be used. Numeric values will multiplied by the number of columns, TRUE will default to cubic interpolation, AsIs to set the knot count directly and 0, FALSE, or non-numeric values will not use spline interpolation. See ggparcoord. Default: "A".

colorOption A character string indicating the colormap option to use. Four options are available: "magma" (or "A"), "inferno" (or "B"), "plasma" (or "C"), "viridis" (or "D", the default option) and "cividis" (or "E"). See scale_colour_viridis_d

scale method used to scale the variables. Default: "uniminmax".

boxplot logical operator indicating whether or not boxplots should underlay the distribution of each variable

alphaLines value of alpha scaler for the lines of the parcoord plot or a column name of the data. Default: 0.1

showPoints logical operator indicating whether points should be plotted or not. Default: TRUE

title character string denoting the title of the plot. Default: ""

probs quantile probabilities. Default: seq(0, 1, 0.25)

... additional parameters to be passed to ggparcoord.

Value

plotly parallel coordinate plot (‘parcoords’) visualization (based on plot_ly)

See Also

plotFunction, plotData

Examples

```r
require("SPOT")
res <- spot(x=NULL, 
  funSphere, 
  lower=rep(-1,3), 
  upper=rep(1,3), 
  control=list(funEvals=25))
plotParallel(res, scale="std")
```

plotSensitivity Sensitivity ggplot of a model

Description

Generates a sensitivity plot.
Usage

plotSensitivity(
  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".

agg.sample number of samples for aggregation type (type="agg").

agg.fun function for aggregation (type="agg").

... additional parameters (currently unused).

Value

`ggplot2` visualization

See Also

`plotFunction, plotData`

---

**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 \( z = f(x_2, x_1) \) where \( Z \) is a two column matrix containing the sample locations \( x_1 \) and \( x_2 \).

lower boundary for \( x_1 \) and \( x_2 \) (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",
  ...
)
plot_sensitivity

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 

caracter, 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_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

```r
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).

contant  
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`)

Description

Predict deep learning models on Census data

Usage

```r
predDlCensus(
  x = NULL,
  target = "age",
  task.type = "classif",
  nobs = 10000,
  nfactors = "high",
  nnumericals = "high",
  cardinality = "high",
  cachedir = "oml.cache",
  k = 1,
  prop = 2/3,
  batch_size = 32,
  verbosity = 0
)
```
Arguments

- **x**: matrix with untransformed hyperparameters, e.g., result from `spot`. Hyperparameters will be transformed in `predDlCensus` with `transformX` and transformations defined in `getModelConf`.

- **target**: target

- **task.type**: class/reg

- **nobs**: number of observations, max: 229,285

- **nfactors**: (character) number of factor variables

- **nnumericals**: (character) number of numerical variables

- **cardinality**: (character) cardinality

- **cachedir**: cache directory

- **k**: number of repeats

- **prop**: vector. proportion between train/test and train/val. Default: 2/3. If one value is given, the same proportion will be used for both splits. Otherwise, the first entry is used for the test/training split and the second value for the training/validation split. If the second value is 1, the validation set is empty. Given `prop = (p1,p2)`, the data will be partitioned as shown in the following two steps:

  **Step 1**: `train1 = p1*data` and `test = (1-p1)*data`

  **Step 2**: `train2 = p2*train1 = p2*p1*data` and `val = (1-p2)*train1 = (1-p2)*p1*data`

  Note: If `p2=1`, no validation data will be generated.

- **batch_size**: batch_size. Default: 32.

- **verbosity**: verbosity. Default: 0

Value

- list of matrices with true and predicted values.

  - **trueY**: true values
  
  - **hatY**: predicted values

Examples

```r
### 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,

PYTHON_RETICULATE <- FALSE
if(PYTHON_RETICULATE){
cfg <- getModelConf(list(model="dl"))
x <- matrix(cfg$defaults, nrow=1)
res <- predDlCensus(x=x, k=2)
}
```
predict machine learning models on Census data

Description

Predict machine learning models on Census data

Usage

```r
predMlCensus(
  x = NULL,
  model = NULL,
  target = "age",
  task.type = "classif",
  nobs = 10000,
  nfactors = "high",
  nnumericals = "high",
  cardinality = "high",
  cachedir = "oml.cache",
  k = 1,
  prop = 2/3,
  verbosity = 0
)
```

Arguments

- `x` : matrix hyperparameter, e.g., result from `spot`. Load result data for ml model to get the hyperparameter vector x, e.g., `load("data/resdl11.RData")` and `x <- result$xbest` or use default.
- `model` : character ml model, e.g., "kknn" run: `result$xbest`. If NULL, default parameters will be used. Default: NULL.
- `target` : target
- `task.type` : class/reg
- `nobs` : number of observations, max: 229,285
- `nfactors` : (character) number of factor variables
- `nnumericals` : (character) number of numerical variables
- `cardinality` : (character) cardinality
- `cachedir` : cachedir
- `k` : number of repeats
- `prop` : split proportion. Default: c(3/5, 1).
- `verbosity` : verbosity. Default: 0

Value

- list of matrices with predictions and true values
**prepareComparisonPlot** prepare data frame for comparisons (boxplots, violin plots)

**Description**

converts result from a spot run into the long format for ggplot.

**Usage**

```r
prepareComparisonPlot(
  runNrMl,
  runNrDl,
  directory,
  defaultModelList = list("dl", "cvglmnet", "knn","ranger","rpart","svm","xgboost"),
  tunedModelList = list("dl", "cvglmnet", "knn","ranger","rpart","svm","xgboost")
)
```

**Arguments**

- `runNrMl`: run number (character) of ml models
- `runNrDl`: run number (character) of dl models
- `directory`: location of the (non-default, e.g., tuned) parameter file
- `defaultModelList`: default model list. Default: `list("dl", "cvglmnet", "knn","ranger","rpart","svm","xgboost")`
- `tunedModelList`: tuned model list. Default: `list("dl", "cvglmnet", "knn","ranger","rpart","svm","xgboost")`

**Value**

data frame with results:

- `x`: integer representing step
- `y`: corresponding function value at step `x`
- `name`: ml/dl model name, e.g., ranger
- `size`: initial design size.
- `yInitMin`: min y value before SMBO is started, based on the initial design only.

**Examples**

```r
### These examples require an activated Python environment as described in
### Bartz-Beielstein, T., Rehbach, F., Sen, A., and Zaefterer, M.:  
### Surrogate Model Based Hyperparameter Tuning for Deep Learning with SPOT,  
```

`PYTHON_RETICULATE <- FALSE`
if(PYTHON_RETICULATE){
  runNrMl <- list("15")
  runNrDl <- list("28")
  directory <- "./book/data"
  prepareComparisonPlot(runNrMl,
                        runNrDl,
                        directory)
}

prepareProgressPlot

---

**prepareProgressPlot**  *prepare data frame for progress plot*

**Description**

converts result from a spot run into the long format.

**Usage**

```r
prepareProgressPlot(modelList, runNr, directory = NULL, maxY = NULL)
```

**Arguments**

- `modelList`: ml/dl model (character)
- `runNr`: run number (character)
- `directory`: location of the (non-default, e.g., tuned) parameter file. Note: load result only when directory is specified, otherwise use (only one!) result from the workspace.
- `maxY`: max number of y values. If NULL then all y values are used.

**Value**

data frame with results:

- `x`: integer representing step
- `y`: corresponding function value at step x.
- `name`: ml/dl model name, e.g., ranger
- `size`: initial design size.
- `yInitMin`: min y value before SMBO is started, based on the initial design only.

**Examples**

```r
modelList <- list("resDl")
runNr <- list("100")
result <- resDl100
directory <- NULL
prepareProgressPlot(modelList,
```
prepare_data_plot

Prepare data for plots

Description
Data preparation

Usage

```r
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 of 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.

Value
list with plotting data and information generated with prepare_data_plot

printf
formatted output

Description
Combine sprintf and writeLines to generate formatted output

Usage
printf(...) 

Arguments
... output to be printed

Examples
x <- 123
printf("x value: %d", x)
printFLAGS  \hspace{10cm} \textit{Print parameter values from FLAG list}  

\textbf{Description}  
Simple print method for FLAG list.

\textbf{Usage}  
\texttt{printFLAGS(FLAGS)}

\textbf{Arguments}  
\begin{itemize}  
\item \texttt{FLAGS} list of parameters, see \texttt{mapX2FLAGS}
\end{itemize}

\begin{longtable}{ll}
\hline
resDl100 & \textit{Results from the spot() run dl100} \\
\hline
\end{longtable}

\textbf{Description}  
Details and the corresponding R code to generate the data can be found in the package vignette \texttt{SPOTMiscVignette.Rmd}.

\textbf{Usage}  
\texttt{resDl100}

\textbf{Format}  
A list of 9 values  
\begin{itemize}  
\item \texttt{xbest} num:  
\item \texttt{ybest} num:  
\item \texttt{x} num  
\item \texttt{y} num  
\item \texttt{logInfo} logi  
\item \texttt{count} int  
\item \texttt{msg} chr  
\item \texttt{model fit} List of 13  
\item \texttt{ybestVec} num  
\end{itemize}
**RMSE**

*root mean squared errors*

**Description**

root mean squared errors

**Usage**

\[
\text{RMSE}(y, \hat{y})
\]

**Arguments**

- **y**: actual value
- **\(\hat{y}\)**: predicted value

**Value**

root mean squared errors

---

**scorePredictions**

*Score results from pred*

**Description**

errors for (actual, predicted) values. Based on package Metrics.

**Usage**

\[
\text{scorePredictions}(val)
\]

**Arguments**

- **val**: list of matrices with true and predicted values, e.g., output from `predM1Census`

**Value**

matrix with scores
selectKerasActivation  Select keras activation function

**Description**

Select keras activation function

**Usage**

```
selectKerasActivation(activation)
```

**Arguments**

- `activation` integer specifying the activation function. Can be one of the following: 1=NULL, 2=RELU

**Value**

activation function use with `funKerasMnist`.

selectKerasOptimizer  Select keras optimizer

**Description**

Select one of the following optimizers: "SDG", "RMSPROP", "ADAGRAD", "ADADELTA", "ADAM", "ADAMAX", "NADAM".

**Usage**

```
selectKerasOptimizer(
  optimizer,
  learning_rate = 0.01,
  momentum = 0,
  decay = 0,
  nesterov = FALSE,
  clipnorm = NULL,
  clipvalue = NULL,
  rho = 0.9,
  epsilon = NULL,
  beta_1 = 0.9,
  beta_2 = 0.999,
  amsgrad = FALSE,
  ...
)
```
Arguments

- **optimizer**: integer specifying the algorithm. Can be one of the following: 1=SGD, 2=RMSProp, 3=ADAGRAD, 4=ADAMDELTA, 5=ADAM, 6=ADAMAX, or 7=NADAM.

  ### SGD:
  - **learning_rate**: float >= 0. Learning rate.
  - **momentum**: float >= 0. Parameter that accelerates SGD in the relevant direction and dampens oscillations.
  - **decay**: float >= 0. Learning rate decay over each update.
  - **nesterov**: boolean. Whether to apply Nesterov momentum.
  - **clipnorm**: Gradients will be clipped when their L2 norm exceeds this value.
  - **clipvalue**: Gradients will be clipped when their absolute value exceeds this value.

  ### RMS:
  - **rho**: float >= 0. Decay factor.
  - **epsilon**: float >= 0. Fuzz factor. If ‘NULL’, defaults to ‘k_epsilon()’.

  ### ADAM:
  - **beta_1**: The exponential decay rate for the 1st moment estimates. float, 0 < beta < 1. Generally close to 1.
  - **beta_2**: The exponential decay rate for the 2nd moment estimates. float, 0 < beta < 1. Generally close to 1.
  - **amsgrad**: Whether to apply the AMSGrad variant of this algorithm from the paper "On the Convergence of Adam and Beyond".
  - ... Unused, present only for backwards compatibility

Value

Optimizer for use with `compile.keras.engine.training.Model`.

---

**selectTarget**

*Select target variable in a data frame*

Description

Select target variable in a data frame

Usage

`selectTarget(df, target)`

Arguments

- df: data frame
- target: character specification of the target variable
**sequentialBifurcation**

**Value**

df with entry target

**Examples**

df <- data.frame(cbind(x=1:2, y=3:4))
df <- selectTarget(df=df, target="y")

---

**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 \( Y = \beta_0 + \sum_{i=1}^p \beta_i X_i + \sum_{1 \leq i < j \leq p} \gamma_{ij} X_i X_j \). 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 split right in the middle.
Value

A list with sensitivity information (effects) for subgroups.

References


Description

A wrapper that calls SPOT when optimizing a keras model with data

Usage

spotKeras(x = NULL, fun, lower, upper, control, kerasConf, kerasData, ...)

Arguments

- **x**: is an optional start point (or set of start points), specified as a matrix. One row for each point, and one column for each optimized parameter.
- **fun**: is the objective function. It should receive a matrix x and return a matrix y. In case the function uses external code and is noisy, an additional seed parameter may be used, see the control$seedFun argument below for details. Mostly, fun must have format y = f(x, ...). If a noisy function requires some specific seed handling, e.g., in some other non-R code, a seed can be passed to fun. For that purpose, the user must specify control$noise = TRUE and fun should be fun(x, seed, ...)
- **lower**: is a vector that defines the lower boundary of search space. This determines also the dimensionality of the problem.
- **upper**: is a vector that defines the upper boundary of search space.
- **control**: is a list with control settings for spot. See `spotControl`.
- **kerasConf**: List of additional parameters passed to keras as described in `getKerasConf`.
- **kerasData**: dataset to use
- **...**: additional parameters passed to fun.

Value

This function returns a result list.
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,

```r
PYTHON_RETICULATE <- FALSE
if(PYTHON_RETICULATE){

model <- "d1"
activeVars <- c("layers", "units", "epochs")
kerasConf <- getKerasConf()
kerasConf$active <- activeVars
cfg <- getModelConf("d1", active = activeVars)
lower <- cfg$lower
upper <- cfg$upper
types <- cfg$types
result <- spotKeras(x = NULL,
  fun = funKerasMnist,
  lower = lower,
  upper = upper,
  control = list(funEvals = 2,
    noise = TRUE,
    types = types,
    plots = FALSE,
    progress = TRUE,
    seedFun = 1,
    seedSPOT = 1,
    designControl = list(size = 1)),
  kerasConf = kerasConf,
  kerasData = getMnistData(kerasConf))
# The result does contain the active parameters only. To get the full vector, use
active2All(x=result$xbest, a=activeVars, model=model)
}
```

---

### spotPlot

**spot plot (generic function)**

**Description**


spotPlot provides a higher level of abstraction and the users can use every plotting function only by calling spotPlot.

**Usage**

```r
spotPlot(plotType, ...)
```
Arguments

plotType function type to be called. It should be given as either "data", "model", "fun", "parallel" or "sensitivity". Otherwise the function returns an error message.

... additional parameters passed to plotData or plotModel.plotFunction, plot_parallel or plot_sensitivity.

Author(s)

Alpar Guer <alpar.guer@smail.th-koeln.de>

See Also

plotData
plotModel
plotFunction

Examples

library("SPOT")
set.seed(1) # seed
k <- 30 # sample number
x <- matrix( cbind(runif(k)*10, runif(k)*10), k, 2) # create data
y <- funSphere(x) # generate random test data
fit <- buildLM(x,y) # create a model
result <- spot(x=NULL, funSphere, c(-5, -5), c(5, 5))

spotPlot(plotType="data", x, y, type="filled.contour")
spotPlot(plotType="model", object=fit, type="contour")
spotPlot(plotType="fun", f=function(x){rowSums(x^2)},
         lower=c(-10,0), upper=c(15,10), type="filled.contour")
spotPlot(plotType = "parallel", object=fit)
spotPlot(plotType = "sensitivity", object=result)


SSE

sum of squared errors

Description

sum of squared errors

Usage

SSE(y, yhat)
Arguments

- \( y \) 
  actual value
- \( \text{yhat} \) 
  predicted value

Value

- sum of squared errors

---

### startCensusRun

**Start hyperparameter optimization runs with spot based on US census data**

---

**Description**

Runs to compare standard machine learning and deep learning models

**Usage**

```r
startCensusRun(
  modelList = list("dl", "cvglmnet", "kknn", "ranger", "rpart", "svm", "xgboost"),
  runNr = "000",
  SPOTVersion = "2.10.12",
  SPOTMiscVersion = "1.19.2",
  timebudget = 3600,
  target = "age",
  cachedir = "oml.cache",
  task.type = "classif",
  nobs = 10000,
  nfactors = "high",
  nnumericals = "high",
  cardinality = "high",
  data.seed = 1,
  prop = 2/3,
  batch_size = 32,
  tuner.seed = 1,
  returnvalue = "validationLoss",
  initSizeFactor = 2,
  spotModel = buildKriging,
  spotOptim = optimDE,
  lower = NULL,
  upper = NULL,
  noise = TRUE,
  OCBA = TRUE,
  OCBABudget = 3,
  multiStart = 2,
  multFun = 200,
)```
startCensusRun

handleNAsMethod = handleNAsMean,
imputeCriteriaFuns = list(is.infinite, is.na, is.nan),
krigingTarget = "ei",
krigingUseLambda = TRUE,
krigingReinterpolate = FALSE,
defaultAsStartingPoint = TRUE,
plots = FALSE,
Rinit = 2,
replicates = 2,
resDummy = FALSE,
verbosity = 0
)

Arguments

modelList list of models. Default: list("dl", "cvglmnet", "kknn", "ranger", "rpart", 
                                  "svm", "xgboost")
runNr character, specifies the run number. Default: "000"
SPOTVersion smallest package version number
SPOTMiscVersion smallest package version number
timebudget time budget Default: 3600 (secs)
target target "age"
cachedir cache dir "oml.cache"
task.type task type "classif"
nobs number of observations 1e4
nfactors number of factorial variables "high"
nnumericals number of numerical variables "high"
cardinality cardinality "high"
data.seed 1
prop proportion 2 / 3
batch_size batch size (for dl) 32
tuner.seed seed for SPOT 1
returnValue "validationLoss"
initSizeFactor multiplier for the initial design size 2
spotModel buildKriging
spotOptim optimDE
lower NULL
upper NULL
noise TRUE
OCBA TRUE
OCBABudget 3
multiStart 2
multFun 200
handleNAsMethod
  handleNAsMean
imputeCriteriaFuns
  list(is.infinite, is.na, is.nan)
krigingTarget "ei"
krigingUseLambda TRUE
krigingReinterpolate FALSE
defaultAsStartingPoint TRUE
plots FALSE
Rinit 2
replicates 2
resDummy FALSE
verbosity 0

Examples

```r
### These examples require an activated Python environment as described in
### Model Based Hyperparameter Tuning for Deep Learning with SPOT,
PYTHON_RETICULATE <- FALSE
if(PYTHON_RETICULATE){
  library("dplyr")
  library("farff")
  library("GGally")
  library("keras")
  library("tensorflow")
  library("Metrics")
  library("mlr")
  library("reticulate")
  library("rpart")
  library("rpart.plot")
  library("SPOT")
  library("SPOTMisc")
  library("tfdatasets")
  library("rsample")
  startCensusRun(modelList=list("ranger", timebudget=60))
}
```
**startMnistRun**

Start hyperparameter optimization runs with spot based on MNIST data

**Description**

Runs to compare deep learning models. Note: Number of epochs is limited:

```r
model <- "dl"; cfg <- getModelConf(model = model); cfg$upper[6] <- 5
```

**Usage**

```r
startMnistRun(
  runNr = "000",
  SPOTVersion = "2.11.4",
  SPOTMiscVersion = "1.19.6",
  encoding = "tensor",
  network = "cnn",
  timebudget = 60,
  data.seed = 1,
  prop = 2/3,
  batch_size = 32,
  tuner.seed = 1,
  returnValue = "validationLoss",
  initSizeFactor = 1,
  spotModel = buildKriging,
  spotOptim = optimDE,
  lower = NULL,
  upper = NULL,
  noise = TRUE,
  OCBA = FALSE,
  OCBABudget = 0,
  multiStart = 2,
  multFun = 200,
  handleNAsMethod = handleNAsMean,
  imputeCriteriaFuns = list(is.infinite, is.na, is.nan),
  krigingTarget = "ei",
  krigingUseLambda = TRUE,
  krigingReinterpolate = TRUE,
  defaultAsStartingPoint = TRUE,
  plots = FALSE,
  Rinit = 1,
  replicates = 1,
  resDummy = FALSE,
  verbosity = 0
)
```
Arguments

runNr character, specifies the run number. Default: "000"
SPOTVersion smallest package version number
SPOTMiscVersion smallest package version number
encoding encoding: "oneHot" od "tensor". Default: "tensor"
network network: "dl" odr "cnn". Default: "cnn"
timebudget time budget Default: 3600 (secs)
data.seed 1
prop proportion 2 / 3
batch_size batch size (for dl) 32
tuner.seed seed for SPOT 1
returnValue "validationLoss"
initSizeFactor multiplier for the initial design size 2
spotModel buildKriging
spotOptim optimDE
lower NULL
upper NULL
noise TRUE
OCBA TRUE
OCBABudget 3
multiStart 2
multFun 200
handleNAsMethod
imputeCriteriaFuns
list(is.infinite, is.na, is.nan)
krigingTarget "ei"
krigingUseLambda TRUE
krigingReinterpolate FALSE
defaultAsStartingPoint FALSE
plots FALSE
Rinit 2
replicates 2
resDummy FALSE
verbosity 0
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,

```r
PYTHON_RETICULATE <- FALSE
if(PYTHON_RETICULATE){
  library("dplyr")
  library("farff")
  library("GGally")
  library("keras")
  library("tensorflow")
  library("Metrics")
  library("mlr")
  library("reticulate")
  library("rpart")
  library("rpart.plot")
  library("SPOT")
  library("SPOTMisc")
  library("tfdatasets")
  library("rsample")
  startMnistRun(timebudget=60, initSizeFactor = 1, verbosity = 1)
  startMnistRun(timebudget=60, encoding="tensor", network="cnn")
}
```

Description

Runs to compare standard machine learning and deep learning models

Usage

```r
startXGBCensusRun(
  modelList = list("xgboost"),
  runNr = "000",
  SPOTVersion = "2.11.14",
  SPOTMiscVersion = "1.19.28",
  timebudget = 3600,
  target = "age",
  cachedir = "oml.cache",
  task.type = "classif",
  nobss = 10000,
  nFactors = "high",
  ...)  
```

```r
startXGBCensusRun
```

Start hyperparameter optimization runs with SPOT based on US census data

```r
startXGBCensusRun
```
nnnumericals = "high",
cardinality = "high",
data.seed = 1,
prop = 2/3,
batch_size = 32,
tuner.seed = 1,
returnValue = "validationLoss",
initSizeFactor = 2,
spotModel = buildKriging,
spotOptim = optimDE,
lower = NULL,
upper = NULL,
noise = TRUE,
OCBA = TRUE,
OCBABudget = 3,
multiStart = 2,
multFun = 200,
handleNASMethod = handleNASMean,
imputeCriteriaFuns = list(is.infinite, is.na, is.nan),
krigingTarget = "ei",
krigingUseLambda = TRUE,
krigingReinterpolate = FALSE,
defaultAsStartingPoint = TRUE,
plots = FALSE,
Rinit = 2,
replicates = 2,
resDummy = FALSE,
verbosity = 0
)

Arguments

modelList list of models. Default: list("xgboost")
runNr character, specifies the run number. Default: "000"
SPOTVersion smallest package version number
SPOTMiscVersion smallest package version number
timebudget time budget Default: 3600 (secs)
target target "age"
cachedir cache dir "oml.cache"
task.type task type "classif"
nobs number of observations 1e4
nfactors number of factorial variables "high"
nnumericals number of numerical variables "high"
cardinality cardinality "high"
Examples

```r
### These examples require an activated Python environment as described in
###
### PYTHON_RETICULATE <- FALSE
### if(PYTHON_RETICULATE){
###   library("dplyr")
###   library("farff")
```
### 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

```r
subgroups(x)
```

#### Arguments

- **x**  
  data

#### Value

data frame with group names and effects

#### Examples

```r
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,
```
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**  
\[ \text{trans}_\text{10pow}(x) \]

**Arguments**  
\[ x \quad \text{input} \]

**Value**  
\[ 10^x \]

**Examples**

```r
f2 <- function(x) {2^x}
fn <- c("identity", "exp", "f2")
xNat <- diag(3)
SPOT::transformX(xNat, fn)
```

---

**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 \( x \rightarrow \text{round}(2^x) \).

**Usage**  
\[ \text{trans}_\text{10pow}_\text{round}(x) \]

**Arguments**  
\[ x \quad \text{input} \]

**Value**  
\[ \text{round}(10^x) \]
trans_1minus10pow

Examples

f2 <- function(x){2^x}
fn <- c("identity", "exp", "f2")
xNat <- diag(3)
SPOT::transformX(xNat, fn)

---

trans_1minus10pow  10 power x transformation

Description

Parameter values x are transformed to 1-10^x. This is helpful for parameters that are likely to be set very close to (but below) a value of 1, such as discount factors in reinforcement learning.

Usage

trans_1minus10pow(x)

Arguments

x  input

Value

1-10^x

Examples

f2 <- function(x){2^x}
fn <- c("identity", "exp", "f2")
xNat <- diag(3)
SPOT::transformX(xNat, fn)

---

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)
trans_2pow_round

Arguments
x

Value
2^x

Examples
f2 <- function(x){2^x}
fn <- c("identity", "exp", "f2")
xNat <- diag(3)
SPOT::transformX(xNat, fn)

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

Value
round(2^x)

Examples
f2 <- function(x){2^x}
fn <- c("identity", "exp", "f2")
xNat <- diag(3)
SPOT::transformX(xNat, fn)
### 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
```

**Examples**

```
f2 <- function(x){2^x}
fn <- c("identity", "exp", "f2")
xNat <- diag(3)
SPOT::transformX(xNat, fn)
```

### trans_mult2_round

**Mult 2 transformation**

**Description**

Parameter values can be translated, implements the multiplication (transformation), i.e., x is mapped to round(2x).

**Usage**

```
trans_mult2_round(x)
```

**Arguments**

- **x**
  - *input*

**Value**

```
x
```
trans_odd_round

Examples

\[
f_2 \leftarrow \text{function}(x)(2^x) \\
fn \leftarrow c(\text{"identity"}, \text{"exp"}, \text{"f2"}) \\
xNat \leftarrow \text{diag}(3) \\
\text{SPOT::transformX}(xNat, fn)
\]

Description

Generate odd numbers, i.e., \( x \rightarrow 2x-1 \) for \( x > 0 \). Return values are rounded using round.

Usage

\[
\text{trans_odd_round}(x)
\]

Arguments

\[
x \quad \text{input}
\]

Value

\[
x
\]

Examples

\[
f_2 \leftarrow \text{function}(x)(2^x) \\
fn \leftarrow c(\text{"trans_odd_round"}, \text{"exp"}, \text{"f2"}) \\
xNat \leftarrow \text{diag}(3) \\
\text{SPOT::transformX}(xNat, fn)
\]
Index

* datasets
  - dataCensusFull, 5
  - resDL100, 70
* optimizers
  - optimizer_adadelta, 49
  - optimizer_adagrad, 50
  - optimizer_adam, 51
  - optimizer_adamax, 52
  - optimizer_nadam, 53
  - optimizer_rmsprop, 54
  - optimizer_sgd, 55

active2All, 4
compile.keras.engine.training.Model, 55, 73
dataCensusFull, 5
evalKerasGeneric, 5, 13
evalKerasMnist, 6, 15, 17, 19, 30, 43
evalKerasMnist_0, 7
evalKerasTransferLearning, 8, 19
evalParamCensus, 10
fit, 6, 8, 9, 13, 15, 17, 19, 28, 30, 42, 46
funBBOBCall, 11
funKerasGeneric, 6, 12, 27, 42, 46
funKerasMnist, 6, 8, 9, 14, 30, 34, 43, 72
funKerasMnist_0, 16
funKerasTransferLearning, 9, 19, 23
genCatsDogsData, 23
genericDataPrep, 5, 12, 24, 42, 45
getDataCensus, 25, 27, 30, 33
gerDataCensusTrainValTestData, 5, 26, 42, 45
getIndices, 28
gerKerasConf, 5–9, 12–15, 17, 19, 23, 27, 28, 34, 37, 42, 43, 45, 46, 75
getMlConfig, 30
getMlrResample, 31
getMlrTask, 32, 33
getMnistData, 6, 7, 14, 17, 29, 34
getModelConf, 28, 29, 35, 38, 48, 64
getObjf, 36
getPredf, 37
gsimpleKerasModel, 5, 37, 42, 45
getVarNames, 38
getparcoord, 39, 56, 57
getparcoordPrepare, 39
getplotProgress, 40
int2fact, 41
k_clear_session, 29
kerasBuildCompile, 42
kerasCompileResult, 42
kerasEvalPrediction, 44
kerasFit, 45
kerasReturnDummy, 46
makeBBOBFunction, 11
makeClassifTask, 33
makeFixedHoldoutInstance, 32
makeLearnerFromHyperparameters, 47
makeRegrTask, 33
mapX2FLAGS, 45, 47, 70
MSE, 48
optimizer_adadelta, 49, 50, 52–56
optimizer_adagrad, 50, 52–56
optimizer_adam, 50, 51, 53–56
optimizer_adamax, 50, 52, 54–56
optimizer_nadam, 50, 52, 53, 55, 56
optimizer_rmsprop, 50, 52–54, 54, 56
optimizer_sgd, 50, 52–55, 55
plot_function_surface, 58
plot_ly, 57, 60–63
plot_parallel, 60
plot_sensitivity, 61
plot_surface, 62
plotData, 57, 58, 61, 62, 77
plotFunction, 57, 58, 61, 62, 77
plotModel, 77
plotParallel, 56
plotSensitivity, 57
predDlCensus, 63
predMlCensus, 65, 71
prepare_data_plot, 68, 69
prepare_spot_result_plot, 69
prepareComparisonPlot, 66
prepareProgressPlot, 41, 67
printf, 69
printFLAGS, 70
resample, 36, 37
resDl100, 70
RMSE, 71
rpart, 35
sb, 74, 86
scale_colour_viridis_d, 57
scorePredictions, 71
selectKerasActivation, 72
selectKerasOptimizer, 72
selectTarget, 73
sensitivity, 74
sequentialBifurcation, 74
spot, 37, 39, 56, 58, 61–67, 69
spotControl, 75
spotKeras, 75
spotPlot, 76
sprintf, 69
SSE, 77
startCensusRun, 78
startMnistRun, 81
startXGBCensusRun, 83
subgroups, 86
to_categorical, 34
trans_10pow, 88, 88, 89–91
trans_10pow_round, 88
trans_1minus10pow, 89
trans_2pow, 89
trans_2pow_round, 90
trans_id, 91
trans_mult2_round, 91
trans_odd_round, 92
transformX, 64
translate_levels, 87
writeLines, 69
xgb.train, 35, 36