

# Package ‘SPOTMisc’

May 2, 2022

**Type** Package

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

**Version** 1.18.14

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

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

**License** GPL (>= 2)

**Encoding** UTF-8

**LazyLoad** yes

**LazyData** true

**Date** 2022-04-30

**Depends** R (>= 4.0.0)

**RoxygenNote** 7.1.2

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

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

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

**VignetteBuilder** knitr

**NeedsCompilation** no

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

**Repository** CRAN

**Date/Publication** 2022-05-02 08:20:07 UTC

## R topics documented:

evalKerasGeneric	4
evalKerasMnist	5
evalKerasTransferLearning	6
evalParamCensus	7
funBBOBCall	9
funKerasGeneric	10
funKerasMnist	12
funKerasTransferLearning	13
genCatsDogsData	17
genericDataPrep	18
getDataCensus	19
getGenericTrainValTestData	20
getKerasConf	22
getMLConfig	23
getMlrResample	25
getMlrTask	26
getMnistData	27
getModelConf	28
getObjf	29
getPrefd	29
getSimpleKerasModel	30
ggparcoordPrepare	31
ggplotProgress	32
int2fact	33
kerasBuildCompile	33
kerasCompileResult	34
kerasEvalPrediction	36
kerasFit	37
kerasReturnDummy	38
makeLearnerFromHyperparameters	39
mapX2FLAGS	39
MSE	40
optimizer_adadelta	40
optimizer_adagrad	41
optimizer_adam	42
optimizer_adamax	43
optimizer_nadam	44
optimizer_rmsprop	46

optimizer_sgd . . . . .	47
plotParallel . . . . .	48
plotSensitivity . . . . .	49
plot_function_surface . . . . .	50
plot_parallel . . . . .	52
plot_sensitivity . . . . .	53
plot_surface . . . . .	54
predDICensus . . . . .	55
predMICensus . . . . .	56
prepareComparisonPlot . . . . .	57
prepareProgressPlot . . . . .	58
prepare_data_plot . . . . .	59
prepare_spot_result_plot . . . . .	60
printf . . . . .	61
printFLAGS . . . . .	61
resKerasMnist02 . . . . .	62
resKerasMnist02Default . . . . .	63
resKerasMnist03 . . . . .	63
resKerasMnist07 . . . . .	64
resKerasTransferLearning04 . . . . .	65
resKerasTransferLearning05 . . . . .	65
resKerasTransferLearning06 . . . . .	66
RMSE . . . . .	67
scorePredictions . . . . .	67
selectKerasOptimizer . . . . .	68
selectTarget . . . . .	69
sequentialBifurcation . . . . .	70
spotPlot . . . . .	71
SSE . . . . .	72
startCensusRun . . . . .	72
subgroups . . . . .	75
translate_levels . . . . .	76
trans_10pow . . . . .	77
trans_10pow_round . . . . .	77
trans_1minus10pow . . . . .	78
trans_2pow . . . . .	78
trans_2pow_round . . . . .	79
trans_id . . . . .	79
trans_mult2_round . . . . .	80

---

evalKerasGeneric	<i>evalKerasGeneric model building and compile</i>
------------------	--

---

### Description

Hyperparameter Tuning: Keras Generic Classification Function.

### Usage

```
evalKerasGeneric(x = NULL, kerasConf = getKerasConf(), 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 <a href="#">getSimpleKerasModel</a> ).
kerasConf	List of additional parameters passed to keras as described in <a href="#">getKerasConf</a> . Default: <code>kerasConf = getKerasConf()</code> .
specList	prepared data. See <a href="#">genericDataPrep</a> . See <a href="#">getGenericTrainValTestData</a> .

### Details

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

### Value

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

### See Also

[getKerasConf](#)

[funKerasGeneric](#)

[fit](#)

---

evalKerasMnist	<i>evalKerasMnist</i>
----------------	-----------------------

---

## Description

Hyperparameter Tuning: Keras MNIST Classification Test Function.

## Usage

```
evalKerasMnist(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 <a href="#">getKerasConf</a> . Default: <code>kerasConf = getKerasConf()</code> .
data	mnist data set. Default: <a href="#">getMnistData</a> .

## Details

Trains a simple deep NN on the MNIST dataset. Standard Code from <https://keras.rstudio.com/>  
Modified by T. Bartz-Beielstein ([tbb@bartzundbartz.de](mailto: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,
### June 2021. http://arxiv.org/abs/2105.14625.
PYTHON_RETICULATE = FALSE
if(PYTHON_RETICULATE){

  library("SPOTMisc")
  kerasConf <- getKerasConf()
  kerasConf$verbose <- 1
  lower <- c(1e-6, 1e-6, 16,0.6, 1e-9, 10, 6,0.4,0.99,1,1e-8)
```

```

upper <- c(0.5, 0.5, 512, 1.5, 1e-2, 50, 10,0.999,0.999,10,6e-8)
types <- c("numeric", "numeric", "integer", "numeric", "numeric",
           "integer", "integer", "numeric", "numeric", "integer",
           "numeric")

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

```

---

evalKerasTransferLearning

*evalKerasTransferLearning*

---

## Description

Hyperparameter Tuning: Keras TransferLearning Test Function.

## Usage

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

## 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 <a href="#">getKerasConf</a> . Default: kerasConf = getKerasConf().
data	data

## Details

Trains a transfer learning model. Standard Code from <https://keras.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**

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

  library("SPOTMisc")
  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",  
  k = 30,  
  directory = "data",  
  target = "age",  
  cachedir = "oml.cache",  
  task.type = "classif",  
  nobs = 10000,  
  nfactors = "high",
```

```

numericals = "high",
cardinality = "high",
prop = 2/3,
verbosity = 0
)

```

### Arguments

runNr	run number (character)
model	ml/dl model (character)
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"
numericals	numericals
cardinality	cardinality
prop	proportion. Default: 2/3
verbosity	verbosity level (0 or 1)

### Examples

```

## Not run:
modellist <- list("dl", "cvglmnet", "kknn", "ranger", "rpart", "svm", "xgboost")
runNr <- list("Default", "00")
directory <- "data"
for (model in modellist){
  for (run in runNr){
    score <- evalParamCensus(model = model,
                             runNr = run,
                             prop=2/3,
                             k=2)
    fileName <- paste0(directory, "/", model, run, "Evaluation.RData")
    save(score, file = fileName)
  }
}

## End(Not run)

```



---

funBBOBCall	<i>funBBOBCall</i>
-------------	--------------------

---

## Description

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

## Usage

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

## Arguments

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

## Value

1-column matrix with resulting function values

## Examples

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

---

funKerasGeneric	<i>funKerasGeneric</i>
-----------------	------------------------

---

### Description

Hyperparameter Tuning: Generic Classification Objective Function.

### Usage

```
funKerasGeneric(x, kerasConf = getKerasConf(), 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 <a href="#">getKerasConf</a> . Default: <code>kerasConf = getKerasConf()</code> .
specList	prepared data. See <a href="#">genericDataPrep</a> .

### 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://keras.rstudio.com/> Modified by T. Bartz-Beielstein ([tbb@bartzundbartz.de](mailto: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>

### 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,
### June 2021. http://arxiv.org/abs/2105.14625.
PYTHON_RETICULATE = FALSE
if(PYTHON_RETICULATE){

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

## model configuration:
cfg <- getModelConf("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:
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

*funKerasMnist*

---

### Description

Hyperparameter Tuning: Keras MNIST Classification Test Function.

### Usage

```
funKerasMnist(x, kConf = getKerasConf(), data = getMnistData())
```

### Arguments

x	matrix of hyperparameter values to evaluate with the function. Rows for points and columns for dimension.
kConf	List of additional parameters passed to keras as described in <a href="#">getKerasConf</a> . Default: <code>kConf = getKerasConf()</code> .
data	mnist data set. Default: <a href="#">getMnistData</a> .

### 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://keras.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,
### June 2021. http://arxiv.org/abs/2105.14625.
PYTHON_RETICULATE = FALSE
if(PYTHON_RETICULATE){

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

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

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

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

```

---

funKerasTransferLearning

*funKerasTransferLearning*


---

**Description**

Hyperparameter Tuning: Keras TransfewrLearning Test Function.

**Usage**

```
funKerasTransferLearning(x, kConf = getKerasConf(), data = NULL)
```

**Arguments**

x	matrix of hyperparameter values to evaluate with the function. Rows for points and columns for dimension.
kConf	List of additional parameters passed to keras as described in <a href="#">getKerasConf</a> . Default: kConf = getKerasConf().
data	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://keras.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,  
### June 2021. http://arxiv.org/abs/2105.14625.
```

```
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,
                         kConf = kerasConf)
           )
save(res, file = paste0("resKerasTransferLearning", as.numeric(Sys.time()), ".RData"))

## Example: resKerasTransferLearning04
## Default objective function "validationLoss", 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")

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,
                      kConf = 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]

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,
                         kConf = 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,
                          noise = TRUE,
                          types = types,
                          optimizer=optimDE,
                          plots = FALSE,
                          progress = TRUE,
                          seedFun = 1,
                          seedSPOT = 1,
                          kConf = kerasConf)
)
save(res, file = paste0("resKerasTransferLearningTrainingLoss06",
as.numeric(Sys.time()),".RData"))
}

```

**Description**

Generate data for [funKerasTransferLearning](#)

**Usage**

```
genCatsDogsData(kConf = getKerasConf())
```

**Arguments**

kConf                   keras configuration. Default: kConf = [getKerasConf](#)

**Details**

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

**Value**

list with test, validation, and test data

---

genericDataPrep	<i>Create an input pipeline using tfdatasets</i>
-----------------	--

---

**Description**

Create an input pipeline using tfdatasets

**Usage**

```
genericDataPrep(data, batch_size = 32)
```

**Arguments**

data                    data. List, e.g., df\$trainCensus, df\$testGeneric, and df\$valCensus data)  
batch\_size            batch size. Default: 32

**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

```
target <- "age"
batch_size <- 32
prop <- 2/3
cachedir <- "oml.cache"
dfCensus <- getDataCensus(target = target,
  nobs = 1000, 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()
```

---

getDataCensus	<i>Get Census KDD data set (+variation)</i>
---------------	---

---

## 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

```
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**

```
## 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

*getGenericTrainValTestData*

---

**Description**

getGenericTrainValTestData

**Usage**

```
getGenericTrainValTestData(dfGeneric = NULL, prop = 0.5)
```

**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  $prop = (p_1, p_2)$ , the data will be partitioned as shown in the following two steps:

**Step 1:**  $train_1 = p_1 * data$  and  $test = (1 - p_1) * data$

**Step 2:**  $train_2 = p_2 * train_1 = p_2 * p_1 * data$  and  $val = (1 - p_2) * train_1 = (1 - p_2) * p_1 * data$

**Value**

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

**Note**

If  $p_2=1$ , no validation data will be generated.

**See Also**

[getKerasConf](#)  
[funKerasGeneric](#)  
[getDataCensus](#)

**Examples**

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

```

---

getKerasConf

*Get keras configuration parameter list*


---

## Description

Configuration list for keras's [fit](#) function.

## Usage

```
getKerasConf()
```

## Details

Additional parameters passed to keras, e.g.,

**verbose:** Verbosity mode (0 = silent, 1 = progress bar, 2 = one line per epoch). Default: 0.

**nClasses:** Number of classes in (multi-class) classification. Specifies the number of units in the last layer (before softmax). Default: 1 (binary classification).

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

**loss:** character. Loss function for compile. Default: "loss\_binary\_crossentropy".

**metrics:** character. Metrics function for compile. Default: "binary\_accuracy".

**clearSession:** logical. Whether to call [k\\_clear\\_session](#) or not at the end of keras modelling. Default: FALSE.

**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". Or "model", i.e., the trained model is returned. Default: "validationLoss".

**callbacks:** List of callbacks to be called during training. Default: list().

**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.

`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.

`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.

`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.

`testData`: Test Data on which to evaluate the loss and any model metrics at the end of the optimization using `evaluate()`.

### Value

kerasConf list with configuration parameters.

### See Also

[evalKerasMnist](#)

[funKerasMnist](#)

[fit](#)

---

getMlConfig

*get ml config for keras on census*

---

### Description

get ml config for keras on census

### Usage

```
getMlConfig(
  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 <a href="#">getDataCensus</a>
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)

**Value**

cfg (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,
### June 2021. http://arxiv.org/abs/2105.14625.
PYTHON_RETICULATE = FALSE
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)  
}
```

---

getMlrResample      *Generate a fixed holdout instance for resampling*

---

## 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

<code>dataset</code>	census data set
<code>task.type</code>	character, either "classif" or "regr".
<code>data.seed</code>	seed

### Value

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

### See Also

[getDataCensus](#)

### Examples

```
## 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

*getMnistData*

---

## Description

getMnistData

## Usage

```
getMnistData(kerasConf = getKerasConf())
```

## Arguments

kerasConf      List of additional parameters passed to keras as described in [getKerasConf](#).  
Default: kerasConf = getKerasConf().

## Value

list with training and test data

## See Also

[getKerasConf](#)  
[funKerasMnist](#)

## Examples

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

library("SPOTMisc")
kerasConf <- getKerasConf()
mnist <- getMnistData()
}
```

---

getModelConf	<i>Get model configuration</i>
--------------	--------------------------------

---

### Description

Configure machine and deep learning models

### Usage

```
getModelConf(model, task.type = NULL, nFeatures = NULL)
```

### Arguments

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, <a href="#">rpart</a> "svm" support vector machines. "xgboost" gradient boosting, <a href="#">xgb.train</a> . "d1" deep learning
task.type	character, either "classif" or "regr".
nFeatures	number of features, e.g., <code>sum(task\$task.desc\$n.feats)</code>

### 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., [xgb.train](#)

relpars list of relative hyperparameters.

### Examples

```
# Get hyperparameter names and their defaults for fitting a
# (recursive partitioning and regression trees) model:
cfg <- getModelConf("rpart")
cfg$tunepars
cfg$defaults
```

---

getObjf	<i>Get objective function for mlr</i>
---------	---------------------------------------

---

### 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 `resample`: `resample(1rn, task, resample, measures = measures, show.info = FALSE)`, with measures defined as `mmce` for classification and `rmse` for regression, `timeboth`, `timetrain`, and `timepredict`.

### Usage

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

### Arguments

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

### 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	<i>Get predictions from mlr</i>
----------	---------------------------------

---

### 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)`,

---

<code>getSimpleKerasModel</code>	<code>getSimpleKerasModel</code>
----------------------------------	----------------------------------

---

**Description**

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

**Usage**

```
getSimpleKerasModel(specList, kerasConf = getKerasConf())
```

**Arguments**

<code>specList</code>	spec
<code>kerasConf</code>	keras configuration. Default: return value from <a href="#">getKerasConf</a> .

**Value**

model. Fitted keras model

**Examples**

```
target <- "age"
nobs <- 1000
batch_size <- 32
prop <- 2/3
dfCensus <- getDataCensus(target = target,
  nobs = nobs)
data <- getGenericTrainValTestData(dfGeneric = dfCensus,
  prop = prop)
specList <- genericDataPrep(data=data, batch_size = batch_size)
kerasConf <- getKerasConf()
simpleModel <- getSimpleKerasModel(specList = specList,
  kerasConf = kerasConf)
```

---

ggparcoordPrepare      *Build data frame for ggparcoord (parallel plot)*

---

### Description

Build data frame for ggparcoord (parallel plot)

### Usage

```
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 <a href="#">spot</a> 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

```
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) # 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,
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")

```

---

ggplotProgress

*simple progress plot*


---

## Description

simple progress plot

## Usage

```

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	<i>Helper function: transform integer to factor</i>
----------	---

---

**Description**

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

**Usage**

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

**Arguments**

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

**Value**

the same factor, now coded with the original levels

---

kerasBuildCompile	<i>evalKerasGeneric model building and compile</i>
-------------------	--

---

**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 <a href="#">getSimpleKerasModel</a> ).
kerasConf	List of additional parameters passed to keras as described in <a href="#">getKerasConf</a> . Default: <code>kerasConf = getKerasConf()</code> .
specList	prepared data. See <a href="#">genericDataPrep</a> . See <a href="#">getGenericTrainValTestData</a> .

**Details**

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

**Value**

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

**See Also**

[getKerasConf](#)  
[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 <a href="#">getKerasConf</a>

**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](#)

[funKerasMnist](#)

**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
### Bartz-Beielstein, T., Rehbach, F., Sen, A., and Zaefferer, M.:
### Surrogate Model Based Hyperparameter Tuning for Deep Learning with SPOT,
### June 2021. http://arxiv.org/abs/2105.14625.
PYTHON_RETICULATE = FALSE
if(PYTHON_RETICULATE){
  library(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)
  simpleModel <- 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 fit*


---

## 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 <a href="#">getSimpleKerasModel</a> ).
specList	prepared data. See <a href="#">genericDataPrep</a> . See <a href="#">getGenericTrainValTestData</a> .
FLAGS	flags, see also <a href="#">mapX2FLAGS</a>
kerasConf	List of additional parameters passed to keras as described in <a href="#">getKerasConf</a> . Default: <code>kerasConf = getKerasConf()</code> .

## Details

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

**Value**

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

**See Also**

[getKerasConf](#)

[funKerasGeneric](#)

[fit](#)

---

kerasReturnDummy	<i>Return dummy values</i>
------------------	----------------------------

---

**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**

```
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**

Map x parameters to a list of named values

**Usage**

```
mapX2FLAGS(x)
```

**Arguments**

x	input values (11 dim). 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.: <a href="#">link{optimizer_sgd}</a> 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,  $0 < \text{beta} < 1$ . Generally close to 1.  
 x[9]: layers number of layers.  
 x[10]: epsilon float  $\geq 0$ . Fuzz factor. If NULL, defaults to k\_epsilon().  
 x[11]: name integer. Specifies optimizer.

**Value**

FLAGS named list (parameter names as specified above): dropout, dropoutfac, units, unitsfact, learning\_rate, epochs, beta\_1, beta\_2, layers, epsilon, name)

---

MSE	<i>mean squared errors</i>
-----	----------------------------

---

**Description**

mean squared errors

**Usage**

MSE(y, yhat)

**Arguments**

y	actual value
yhat	predicted value

**Value**

mean squared errors

---

optimizer_adadelta	<i>Adadelta optimizer.</i>
--------------------	----------------------------

---

**Description**

Adadelta optimizer as described in [ADADELTA: An Adaptive Learning Rate Method](<https://arxiv.org/abs/1212.5701>).



**Usage**

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

**Arguments**

<code>learning_rate</code>	float $\geq 0$ . Learning rate.
<code>rho</code>	float $\geq 0$ . Decay factor.
<code>epsilon</code>	float $\geq 0$ . Fuzz factor. If 'NULL', defaults to 'k_epsilon()'. 
<code>decay</code>	float $\geq 0$ . Learning rate decay over each update.
<code>clipnorm</code>	Gradients will be clipped when their L2 norm exceeds this value.
<code>clipvalue</code>	Gradients will be clipped when their absolute value exceeds this value.
<code>...</code>	Unused, present only for backwards compatability

**Note**

To enable compatibility with the ranges of the learning rates of the other optimizers, the learning rate `learning_rate` is internally mapped to  $1 - \text{learning\_rate}$ . 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.

**See Also**

Other optimizers: [optimizer\\_adagrad\(\)](#), [optimizer\\_adamax\(\)](#), [optimizer\\_adam\(\)](#), [optimizer\\_nadam\(\)](#), [optimizer\\_rmsprop\(\)](#), [optimizer\\_sgd\(\)](#)

---

<code>optimizer_adagrad</code>	<i>Adagrad optimizer.</i>
--------------------------------	---------------------------

---

**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 compatability

**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	<i>Adam optimizer</i>
----------------	-----------------------

---

**Description**

Adam optimizer as described in [Adam - A Method for Stochastic Optimization](<https://arxiv.org/abs/1412.6980v8>).

**Usage**

```
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 $\geq 0$ . 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 $\geq 0$ . Fuzz factor. If 'NULL', defaults to 'k_epsilon()'.
decay	float $\geq 0$ . 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 compatability

### References

- [Adam - A Method for Stochastic Optimization](<https://arxiv.org/abs/1412.6980v8>) - [On the Convergence of Adam and Beyond](<https://openreview.net/forum?id=ryQu7f-RZ>)

### Note

Default parameters follow those provided in the original paper.

### See Also

Other optimizers: [optimizer\\_adadelat\(\)](#), [optimizer\\_adagrad\(\)](#), [optimizer\\_adamax\(\)](#), [optimizer\\_nadam\(\)](#), [optimizer\\_rmsprop\(\)](#), [optimizer\\_sgd\(\)](#)

---

optimizer_adamax	<i>Adamax optimizer</i>
------------------	-------------------------

---

### 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 $\geq 0$ . 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 $\geq 0$ . Fuzz factor. If 'NULL', defaults to 'k_epsilon()'.
decay	float $\geq 0$ . 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 compatability

**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 rat of 0.001 will be mapped to 0.002 (which is the default.)

**See Also**

Other optimizers: [optimizer\\_adadelat\(\)](#), [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**

```
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 compatability

**Details**

Default parameters follow those provided in the paper. It is recommended to leave the parameters of this optimizer at their default values.

**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.)

**See Also**

[On the importance of initialization and momentum in deep learning](<https://www.cs.toronto.edu/~fritz/absps/momentum.pdf>)

Other optimizers: [optimizer\\_adadelat\(\)](#), [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 compatability

### Note

It is recommended to leave the parameters of this optimizer at their default values (except the learning rate, which can be freely tuned).

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

### See Also

Other optimizers: [optimizer\\_adadelata\(\)](#), [optimizer\\_adagrad\(\)](#), [optimizer\\_adamax\(\)](#), [optimizer\\_adam\(\)](#), [optimizer\\_nadam\(\)](#), [optimizer\\_sgd\(\)](#)

---

optimizer\_sgd                      *Keras optimizers.*

---

### Description

Based on: [keras/R/optimizers.R](https://github.com/rstudio/keras/blob/main/R/optimizers.R)

### Usage

```
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

The following code is commented: `backcompat_fix_rename_lr_to_learning_rate(...)`

Stochastic gradient descent optimizer

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

### 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  $10 * learning\_rate$ . That is, a learning rate of 0.001 will be mapped to 0.01 (which is the default.)

**See Also**

Other optimizers: [optimizer\\_adadelat\(\)](#), [optimizer\\_adagrad\(\)](#), [optimizer\\_adamax\(\)](#), [optimizer\\_adam\(\)](#), [optimizer\\_nadam\(\)](#), [optimizer\\_rmsprop\(\)](#)

---

<code>plotParallel</code>	<i>Parallel coordinate plot of a data set</i>
---------------------------	---

---

**Description**

Parallel plot based on [ggparcoord](#).

**Usage**

```
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**

<code>result</code>	the result list returned by <a href="#">spot</a> , importantly including the data <code>x</code> , <code>y</code> .
<code>xlab</code>	character, the value of the independent variable
<code>ylab</code>	character, the value of the dependent variable predicted by the corresponding model.
<code>yrange</code>	a two-element vector that specifies the range of <code>y</code> values to consider (data outside of that range will be excluded).
<code>splineFactor</code>	logical or numeric operator indicating whether spline interpolation should be used. Numeric values will multiplied by the number of columns, <code>TRUE</code> will default to cubic interpolation, <code>AsIs</code> to set the knot count directly and <code>0</code> , <code>FALSE</code> , or non-numeric values will not use spline interpolation. See <a href="#">ggparcoord</a> . Default: <code>"A"</code> .
<code>colorOption</code>	A character string indicating the colormap option to use. Four options are available: <code>"magma"</code> (or <code>"A"</code> ), <code>"inferno"</code> (or <code>"B"</code> ), <code>"plasma"</code> (or <code>"C"</code> ), <code>"viridis"</code> (or <code>"D"</code> , the default option) and <code>"cividis"</code> (or <code>"E"</code> ). See <a href="#">scale_colour_viridis_d</a>



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 <a href="#">ggparcoord</a> .

**Value**

plotly parallel coordinate plot ('parcoords') visualization (based on [plot\\_ly](#))

**See Also**

[plotFunction](#), [plotData](#)

**Examples**

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

---

plotSensitivity	<i>Sensitivity ggplot of a model</i>
-----------------	--------------------------------------

---

**Description**

Sensitivity ggplot of a model

**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 <code>spot</code> , importantly including a <code>modelFit</code> , and the data <code>x, y</code> .
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	<i>Parallel coordinate plot of a data set</i>
---------------	---

---

**Description**

mlrTools

**Usage**

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

**Arguments**

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

**Value**

plotly parallel coordinate plot ('parcoords') visualization (based on [plot\\_ly](#))

**See Also**

[plotFunction](#), [plotData](#)

---

plot_sensitivity	<i>Sensitivity plot of a model</i>
------------------	------------------------------------

---

**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 <a href="#">spot</a> , importantly including a <code>modelFit</code> , and the data <code>x</code> , <code>y</code> .
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	<i>Surface plot of a model</i>
--------------	--------------------------------

---

**Description**

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

**Usage**

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

**Arguments**

object	the result list returned by <a href="#">spot</a> , importantly including a <code>modelFit</code> , and the data <code>x</code> , <code>y</code> .
which	a vector with two elements, each an integer giving the two independent variables of the plot (the integers are indices of the respective data set).
constant	a numeric vector that states for each variable a constant value that it will take on if it is not varied in the plot. This affects the parameters not selected by the <code>which</code> parameter. By default, this will be fixed to the best known solution, i.e., the one with minimal <code>y</code> -value, according to <code>which.min(object\$y)</code> . The length of this numeric vector should be the same as the number of columns in <code>object\$x</code>
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 <code>plotly</code> package and will work in RStudio, but not in the standard RGui.
...	additional parameters passed to the <code>contour</code> or <code>filled.contour</code> function.

**Value**

plotly visualization (based on [plot\\_ly](#))

---

predDICensus	<i>Predict deep learning models on Census data</i>
--------------	--

---

**Description**

Predict deep learning models on Census data

**Usage**

```
predDICensus(
  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 <a href="#">spot</a> . Hyperparameters will be transformed in predDICensus with <a href="#">transformX</a> and transformations defined in <a href="#">getModelConf</a> .
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

```
### These examples require an activated Python environment as described in
### Bartz-Beielstein, T., Rehbach, F., Sen, A., and Zaefferer, M.:
### Surrogate Model Based Hyperparameter Tuning for Deep Learning with SPOT,
### June 2021. http://arxiv.org/abs/2105.14625.
PYTHON_RETICULATE = FALSE
if(PYTHON_RETICULATE){
  cfg <- getModelConf("dl")
  x <- matrix(cfg$defaults, nrow=1)
  res <- predDlCensus(x=x, k=2)
}
```

---

predMlCensus

*Predict machine learning models on Census data*

---

### Description

Predict machine learning models on Census data

### Usage

```
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 <a href="#">spot</a> Load result data for ml model to get the hyperparameter vector x, e.g., <code>load("data/resd111.RData")</code> and <code>x &lt;- result\$xbest</code> or use default.
model	character ml model, e.g., "kkn" run: <code>result\$xbest</code> . 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: <code>c(3/5, 1)</code> .
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

```
prepareComparisonPlot(runNrMl, runNrDl, directory)
```

### 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

**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**

```
### These examples require an activated Python environment as described in
### Bartz-Beielstein, T., Rehbach, F., Sen, A., and Zaefferer, M.:
### Surrogate Model Based Hyperparameter Tuning for Deep Learning with SPOT,
### June 2021. http://arxiv.org/abs/2105.14625.
PYTHON_RETICULATE = FALSE
if(PYTHON_RETICULATE){
  runNrM1 <- list("15")
  runNrD1 <- list("28")
  directory <- "../book/data"
  prepareComparisonPlot(runNrM1,
                        runNrD1,
                        directory)
}
```

---

prepareProgressPlot    *prepare data frame for progress plot*

---

**Description**

converts result from a [spot](#) run into the long format.

**Usage**

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

**Arguments**

modelList            ml/dl model (character)

runNr                run number (character)

directory            location of the (non-default, e.g., tuned) parameter file

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**

```
modelList <- list("dl")
runNr <- list("28")
directory <- "../book/data"
prepareProgressPlot(modelList,
                    runNr,
                    directory)
```

---

prepare_data_plot	<i>Prepare data for plots</i>
-------------------	-------------------------------

---

**Description**

mlrTools

**Usage**

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

**Arguments**

model	a function that can be used to build a model based on the data, e.g. : buildRanger or buildKriging. Default is buildRanger, since it is fast and robust.
modelControl	a list of control settings for the respective model. Default is an empty list (use default model controls).

x	a matrix of x-values to be plotted (i.e., columns are the independent variables, rows are samples). Should have same number of rows as y and log.
namesx	character vector, printable names for the x data. Should have same length as x has columns. Default is x1, x2, ...
y	a one-column matrix of y-values to be plotted (dependent variable). Should have same number of rows as x and log.
namesy	character, giving a printable name for y. Default is "y".
log	matrix, a data set providing (optional) additional dependent variables (but these are not modeled). Should have same number of rows as y and x.
nameslog	character vector, printable names for the log data. Should have same length as log has columns. Default is NULL (no names).

**Value**

list with plotting data and information

---

```
prepare_spot_result_plot
```

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

---

**Description**

Preparation 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 <a href="#">spot</a> call.
model	a function that can be used to build a model based on the data, e.g. : <a href="#">buildRanger</a> or <a href="#">buildKriging</a> . Default is <a href="#">buildRanger</a> , 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 <a href="#">prepare_data_plot</a> : <a href="#">namesx</a> , <a href="#">namesy</a> , <a href="#">nameslog</a> character vectors providing names for x, y and logInfo data.

**Value**

list with plotting data and information generated with [prepare\\_data\\_plot](#)

---

printf	<i>formatted output</i>
--------	-------------------------

---

**Description**

Combine `sprintf` and `writeln` to generate formatted output

**Usage**

```
printf(...)
```

**Arguments**

...            output to be printed

**Examples**

```
x <- 123
printf("x value: %d", x)
```

---

printFLAGS	<i>Print parameter values from FLAG list</i>
------------	--

---

**Description**

Print values of the following hyperparameters:

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  $\geq 0$ . Fuzz factor. If NULL, defaults to `k_epsilon()`.

x[11]: name integer. Specifies optimizer.

**Usage**

```
printFLAGS(FLAGS)
```

**Arguments**

FLAGS            list of parameters, see [mapX2FLAGS](#)

---

resKerasMnist02        *Results from a very simple SPOT run, version 2, with replicates*

---

**Description**

Details and the corresponding R code to generate the data can be found in the package vignette `SPOTMiscVignette.Rmd`.

**Usage**

```
resKerasMnist02
```

**Format**

A list of 9 values

**xbest** num:

**ybest** num:

**x** num

**y** num

**logInfo** logi

**count** int

**msg** chr

**model fit** List of 13

**ybestVec** num

---

`resKerasMnist02Default`*Results from a very simple SPOT run, version 2, with replicates*

---

**Description**

Details and the corresponding R code to generate the data can be found in the package vignette `SPOTMiscVignette.Rmd`.

**Usage**`resKerasMnist02Default`**Format**

A list of 9 values

**xbest** num:

**ybest** num:

**x** num

**y** num

**logInfo** logi

**count** int

**msg** chr

**model fit** List of 13

**ybestVec** num

---

`resKerasMnist03`*Results from a very simple SPOT run, version 3, with replicates*

---

**Description**

Details and the corresponding R code to generate the data can be found in the package vignette `SPOTMiscVignette.Rmd`.

**Usage**`resKerasMnist03`

**Format**

A list of 9 values

**xbest** num:

**ybest** num:

**x** num

**y** num

**logInfo** logi

**count** int

**msg** chr

**model fit** List of 13

**ybestVec** num

---

resKerasMnist07

*Results from a very simple SPOT run, version 7, with replicates*

---

**Description**

Details and the corresponding R code to generate the data can be found in the package vignette SPOTMiscVignette.Rmd.

**Usage**

```
resKerasMnist07
```

**Format**

A list of 9 values

**xbest** num:

**ybest** num:

**x** num

**y** num

**logInfo** logi

**count** int

**msg** chr

**model fit** List of 13

**ybestVec** num



---

`resKerasTransferLearning04`*Results from a SPOT run, version 4, with replicates*

---

**Description**

SPOT results: `resKerasTransferLearning04`

Details and the corresponding R code to generate the data can be found in the package vignette `SPOTMiscVignette.Rmd`.

**Usage**

```
resKerasTransferLearning04
```

**Format**

A list of 9 values

**xbest** num:

**ybest** num:

**x** num

**y** num

**logInfo** logi

**count** int

**msg** chr

**model fit** List of 13

**ybestVec** num

---

`resKerasTransferLearning05`*Results from a SPOT run, version 5, with replicates: negValidationAccuracy*

---

**Description**

SPOT results: `resKerasTransferLearning05`

Details and the corresponding R code to generate the data can be found in the package vignette `SPOTMiscVignette.Rmd`.

**Usage**

```
resKerasTransferLearning05
```

**Format**

A list of 9 values

**xbest** num:

**ybest** num:

**x** num

**y** num

**logInfo** logi

**count** int

**msg** chr

**model fit** List of 13

**ybestVec** num

---

resKerasTransferLearning06

*Results from a SPOT run, version 06, default*

---

**Description**

SPOT results: resKerasTransferLearning06: Evaluation of the default parameters

Details and the corresponding R code to generate the data can be found in the package vignette SPOTMiscVignette.Rmd.

**Usage**

resKerasTransferLearning06

**Format**

A list of 9 values

**xbest** num:

**ybest** num:

**x** num

**y** num

**logInfo** logi

**count** int

**msg** chr

**model fit** List of 13

**ybestVec** num

---

RMSE	<i>root mean squared errors</i>
------	---------------------------------

---

**Description**

root mean squared errors

**Usage**

```
RMSE(y, yhat)
```

**Arguments**

y	actual value
yhat	predicted value

**Value**

root mean squared errors

---

scorePredictions	<i>Score results from pred</i>
------------------	--------------------------------

---

**Description**

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

**Usage**

```
scorePredictions(val)
```

**Arguments**

val	list of matrices with true and predicted values, e.g., output from <a href="#">predM1Census</a>
-----	---

**Value**

matrix with scores

---

selectKerasOptimizer *Select keras optimizer*

---

## Description

Select keras optimizer

## Usage

```
selectKerasOptimizer(
    name,
    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

name	integer specifying the algorithm. Can be one of the following: 1=SDG, 2=RMSPROP, 3=ADAGRAD, 4=ADADELTA, 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, $0 < \text{beta} < 1$ . 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 compatability

**Value**

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

---

selectTarget	<i>Select target variable in a data frame</i>
--------------	---

---

**Description**

Select target variable in a data frame

**Usage**

```
selectTarget(df, target)
```

**Arguments**

df	data frame
target	character specification of the target variable

**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

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

### 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 splitted right in the middle.

### Value

sa list with sensitivity information (effects) for subgroups.

### References

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

---

spotPlot	<i>spotPlot</i>
----------	-----------------

---

## Description

A wrapper function for available plotting options in SPOT and SPOTMisc. Plotting functions from SPOT -> plotdata, plotModel, plotFunction. Plotting functions from SPOTMisc -> plot\_parallel, plot\_sensitivity.

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

## Usage

```
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	<i>sum of squared errors</i>
-----	------------------------------

---

### Description

sum of squared errors

### Usage

```
SSE(y, yhat)
```

### Arguments

y	actual value
yhat	predicted value

### Value

sum of squared errors

---

startCensusRun	<i>Start hyperparameter optimization runs with spot based on US census data</i>
----------------	---

---

### Description

Runs to compare standard machine learning and deep learning models

### Usage

```
startCensusRun(
  modelList = list("dl", "cvglmnet", "kknn", "ranger", "rpart", "svm", "xgboost"),
  runNr = "000",
  SPOTVersion = "2.9.50",
  SPOTMiscVersion = "1.18.12",
  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,
    handleNAsMethod = handleNAsMean,
    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
  )

```

### 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
                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,
### June 2021. http://arxiv.org/abs/2105.14625.
PYTHON_RETICULATE = FALSE
if(PYTHON_RETICULATE){
  library("dplyr")
  library("farff")
}

```

```
library("GGally")
library("keras")
library("tensorflow")
library("Metrics")
library("mlr")
library("OpenML")
library("reticulate")
library("rpart")
library("rpart.plot")
library("SPOT")
library("SPOTMisc")
library("tfdatasets")
library("rsample")
startCensusRun(modelList=list("ranger", timebudget=60))
}
```

---

subgroups

*Return effects for each subgroup*

---

### Description

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

### Usage

```
subgroups(x)
```

### Arguments

x                    data

### Value

data frame with group names and effects

### Examples

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

# Model-based optimization
```

```
res <- spot(,funSphere,
           lower, upper,
           control=list(funEvals=30,
                       optimizer = optimNLOPTR))

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

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

# Extract group information (variable effects) from sensitivity analysis
ps <- subgroups(sens)
colors <- brewer.pal(12, "Set3")
barplot(ps$effect, names.arg=ps$group, col= colors)
```

---

translate\_levels      *Helper function: translate levels*

---

## Description

Translate existing levels of a factor into new levels.

## Usage

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

## Arguments

x                    a factor vector to be translated

translations        a named list that specifies the translation: `list(newlevel=c(oldlevel1,oldlevel2,etc))`.

## Value

translated factor

---

trans_10pow	<i>10 power x transformation</i>
-------------	----------------------------------

---

**Description**

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

**Usage**

```
trans_10pow(x)
```

**Arguments**

x	input
---	-------

**Value**

$10^x$

---

trans_10pow_round	<i>10 power x transformation with round</i>
-------------------	---

---

**Description**

Parameter values can be translated, e.g., to base 10 as implemented in [trans\\_10pow](#). trans\_10pow\_round implements the transformation  $x \rightarrow \text{round}(10^x)$ .

**Usage**

```
trans_10pow_round(x)
```

**Arguments**

x	input
---	-------

**Value**

$\text{round}(10^x)$

---

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**

$1-10^x$

---

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  $x \rightarrow 2^x$ .

**Usage**

trans\_2pow( $x$ )

**Arguments**

$x$                       input

**Value**

$2^x$

---

trans_2pow_round	<i>2 power x transformation with round</i>
------------------	--

---

**Description**

Parameter values can be translated, e.g., to base 10 as implemented in [trans\\_10pow](#). trans\_2pow\_round implements the transformation  $x \rightarrow \text{round}(2^x)$ .

**Usage**

```
trans_2pow_round(x)
```

**Arguments**

x	input
---	-------

**Value**

```
round(2^x)
```

---

trans_id	<i>Identity transformation</i>
----------	--------------------------------

---

**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
```

---

<code>trans_mult2_round</code>	<i>Mult 2 transformation</i>
--------------------------------	------------------------------

---

**Description**

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

**Usage**

```
trans_mult2_round(x)
```

**Arguments**

<code>x</code>	input
----------------	-------

**Value**

`x`



# Index

## \* datasets

- resKerasMnist02, 62
- resKerasMnist02Default, 63
- resKerasMnist03, 63
- resKerasMnist07, 64
- resKerasTransferLearning04, 65
- resKerasTransferLearning05, 65
- resKerasTransferLearning06, 66

## \* optimizers

- optimizer\_adadelta, 40
- optimizer\_adagrad, 41
- optimizer\_adam, 42
- optimizer\_adamax, 43
- optimizer\_nadam, 44
- optimizer\_rmsprop, 46
- optimizer\_sgd, 47

compile.keras.engine.training.Model,  
47, 69

evalKerasGeneric, 4, 10  
evalKerasMnist, 5, 12, 14, 23, 35  
evalKerasTransferLearning, 6, 14  
evalParamCensus, 7

fit, 4, 5, 7, 10, 12, 14, 22, 23, 34, 38  
funBBOBCall, 9  
funKerasGeneric, 4, 10, 21, 34, 38  
funKerasMnist, 5, 7, 12, 23, 27, 35  
funKerasTransferLearning, 7, 13, 18

genCatsDogsData, 17  
genericDataPrep, 4, 10, 18, 34, 37  
getDataCensus, 19, 21, 24, 26  
getGenericTrainValTestData, 4, 20, 34, 37  
getKerasConf, 4–7, 10, 12, 14, 18, 21, 22, 27,  
30, 34, 37, 38  
getMlConfig, 23  
getMlrResample, 25  
getMlrTask, 25, 26

getMnistData, 5, 12, 27  
getModelConf, 28, 55  
getObjf, 29  
getPredf, 29  
getSimpleKerasModel, 4, 30, 34, 37  
ggparcoord, 31, 48, 49  
ggparcoordPrepare, 31  
ggplotProgress, 32

int2fact, 33

k\_clear\_session, 22  
kerasBuildCompile, 33  
kerasCompileResult, 34  
kerasEvalPrediction, 36  
kerasFit, 37  
kerasReturnDummy, 38

makeBBOBFunction, 9  
makeClassifTask, 26  
makeFixedHoldoutInstance, 25  
makeLearnerFromHyperparameters, 39  
makeRegrTask, 26  
mapX2FLAGS, 37, 39, 62  
MSE, 40

optimizer\_adadelta, 40, 42–46, 48  
optimizer\_adagrad, 41, 41, 43–46, 48  
optimizer\_adam, 41, 42, 42, 44–46, 48  
optimizer\_adamax, 41–43, 43, 45, 46, 48  
optimizer\_nadam, 41–44, 44, 46, 48  
optimizer\_rmsprop, 41–45, 46, 48  
optimizer\_sgd, 41–46, 47

plot\_function\_surface, 50  
plot\_ly, 49, 52–55  
plot\_parallel, 52  
plot\_sensitivity, 53  
plot\_surface, 54  
plotData, 49, 50, 53, 54, 71  
plotFunction, 49, 50, 53, 54, 71

plotModel, [71](#)  
plotParallel, [48](#)  
plotSensitivity, [49](#)  
predDlCensus, [55](#)  
predMlCensus, [56](#), [67](#)  
prepare\_data\_plot, [59](#), [60](#)  
prepare\_spot\_result\_plot, [60](#)  
prepareComparisonPlot, [57](#)  
prepareProgressPlot, [32](#), [58](#)  
printf, [61](#)  
printFLAGS, [61](#)  
  
resample, [29](#), [30](#)  
resKerasMnist02, [62](#)  
resKerasMnist02Default, [63](#)  
resKerasMnist03, [63](#)  
resKerasMnist07, [64](#)  
resKerasTransferLearning04, [65](#)  
resKerasTransferLearning05, [65](#)  
resKerasTransferLearning06, [66](#)  
RMSE, [67](#)  
rpart, [28](#)  
  
sb, [70](#), [75](#)  
scale\_colour\_viridis\_d, [48](#)  
scorePredictions, [67](#)  
selectKerasOptimizer, [68](#)  
selectTarget, [69](#)  
sensitivity, [70](#)  
sequentialBifurcation, [70](#)  
spot, [29–31](#), [48](#), [50](#), [52–55](#), [57](#), [58](#), [60](#)  
spotPlot, [71](#)  
sprintf, [61](#)  
SSE, [72](#)  
startCensusRun, [72](#)  
subgroups, [75](#)  
  
trans\_10pow, [77](#), [77](#), [78](#), [79](#)  
trans\_10pow\_round, [77](#)  
trans\_1minus10pow, [78](#)  
trans\_2pow, [78](#)  
trans\_2pow\_round, [79](#)  
trans\_id, [79](#)  
trans\_mult2\_round, [80](#)  
transformX, [55](#)  
translate\_levels, [76](#)  
  
writeLines, [61](#)  
  
xgb.train, [28](#)