Package ‘NeuralSens’

May 15, 2020

Version 0.2.1

Title Sensitivity Analysis of Neural Networks

Date 2020-05-15

Description Analysis functions to quantify inputs importance in neural network models. Functions are available for calculating and plotting the inputs importance and obtaining the activation function of each neuron layer and its derivatives. The importance of a given input is defined as the distribution of the derivatives of the output with respect to that input in each training data point.

Author José Portela González [aut],
Antonio Muñoz San Roque [aut],
Jaime Pizarroso Gonzalo [aut, ctb, cre]

Maintainer Jaime Pizarroso Gonzalo <jpizarroso@comillas.edu>

Imports ggplot2, gridExtra, NeuralNetTools, reshape2, caret, fastDummies, stringr, Hmisc, ggforce, scales, ggnewscale

Suggests h2o, neural, RSNNS, nnet, neuralnet

RoxygenNote 7.1.0

NeedsCompilation no

URL https://github.com/JaiPizGon/NeuralSens

BugReports https://github.com/JaiPizGon/NeuralSens/issues

License GPL (>= 2)

Encoding UTF-8

LazyData true

Repository CRAN

Date/Publication 2020-05-15 08:10:02 UTC

R topics documented:

`ActFunc` *Activation function of neuron*

Description

Evaluate activation function of a neuron

Usage

```
ActFunc(type = "sigmoid", ...)
```

Arguments

`type`	`character` name of the activation function
`...`	extra arguments needed to calculate the functions

Value

`numeric` output of the neuron

Examples

```
# Return the sigmoid activation function of a neuron
ActivationFunction <- ActFunc("sigmoid")
# Return the tanh activation function of a neuron
ActivationFunction <- ActFunc("tanh")
# Return the activation function of several layers of neurons
actfuncs <- c("linear","sigmoid","linear")
ActivationFunctions <- sapply(actfuncs, ActFunc)
```

CombineSens *Sensitivity analysis plot over time of the data*

Description

Plot of sensitivity of the neural network output respect to the inputs over the time variable from the data provided

Usage

```
CombineSens(object, comb_type = "mean")
```

Arguments

`object`	`SensMLP` object generated by `SensAnalysisMLP` with several outputs (classification MLP)
`comb_type`	Function to combine the matrixes of the `raw_sens` component of `object`. It can be "mean", "median" or "sqmean". It can also be a function to combine the rows of the matrixes

Value

`SensMLP` object with the sensitivities combined

Examples

```
## Not run:
# mod should be a neural network classification model
sens <- SensAnalysisMLP(mod)
combinesens <- CombineSens(sens, "sqmean")

## End(Not run)
```

ComputeHessMeasures *Plot sensitivities of a neural network model*

Description

Function to plot the sensitivities created by SensAnalysisMLP.

Usage

```
ComputeHessMeasures(sens)
```

Arguments

sens	SensAnalysisMLP object created by SensAnalysisMLP.

Value

SensAnalysisMLP object with the sensitivities calculated

Examples

```
## Load data -------------------------------------------------------------------
data("DAILY_DEMAND_TR")
fdata <- DAILY_DEMAND_TR

## Parameters of the NNET ------------------------------------------------------
hidden_neurons <- 5
iters <- 250
decay <- 0.1

################################################################################
#########################  REGRESSION NNET #####################################
################################################################################
## Regression dataframe --------------------------------------------------------
# Scale the data
fdata.Reg.tr <- fdata[,2:ncol(fdata)]
fdata.Reg.tr[,3] <- fdata.Reg.tr[,3]/10
fdata.Reg.tr[,1] <- fdata.Reg.tr[,1]/1000

# Normalize the data for some models
preProc <- caret::preProcess(fdata.Reg.tr, method = c("center","scale"))
nntrData <- predict(preProc, fdata.Reg.tr)

#' ## TRAIN nnet NNET --------------------------------------------------------
# Create a formula to train NNET
form <- paste(names(fdata.Reg.tr)[2:ncol(fdata.Reg.tr)], collapse = " + ")
form <- formula(paste(names(fdata.Reg.tr)[1], form, sep = " ~ "))

set.seed(150)
nnetmod <- nnet::nnet(form,
```

```
                           data = nntrData,
                           linear.output = TRUE,
                           size = hidden_neurons,
                           decay = decay,
                           maxit = iters)
# Try SensAnalysisMLP
sens <- NeuralSens::SensAnalysisMLP(nnetmod, trData = nntrData, plot = FALSE)
```

ComputeSensMeasures   *Plot sensitivities of a neural network model*

Description

Function to plot the sensitivities created by SensAnalysisMLP.

Usage

```
ComputeSensMeasures(sens)
```

Arguments

sens	SensAnalysisMLP object created by SensAnalysisMLP.

Value

SensAnalysisMLP object with the sensitivities calculated

Examples

```
## Load data -------------------------------------------------------------------
data("DAILY_DEMAND_TR")
fdata <- DAILY_DEMAND_TR

## Parameters of the NNET ------------------------------------------------------
hidden_neurons <- 5
iters <- 250
decay <- 0.1

################################################################################
#########################  REGRESSION NNET #####################################
################################################################################
## Regression dataframe --------------------------------------------------------
# Scale the data
fdata.Reg.tr <- fdata[,2:ncol(fdata)]
fdata.Reg.tr[,3] <- fdata.Reg.tr[,3]/10
fdata.Reg.tr[,1] <- fdata.Reg.tr[,1]/1000

# Normalize the data for some models
preProc <- caret::preProcess(fdata.Reg.tr, method = c("center","scale"))
nntrData <- predict(preProc, fdata.Reg.tr)
```

```
#' ## TRAIN nnet NNET --------------------------------------------------------
# Create a formula to train NNET
form <- paste(names(fdata.Reg.tr)[2:ncol(fdata.Reg.tr)], collapse = " + ")
form <- formula(paste(names(fdata.Reg.tr)[1], form, sep = " ~ "))

set.seed(150)
nnetmod <- nnet::nnet(form,
                           data = nntrData,
                           linear.output = TRUE,
                           size = hidden_neurons,
                           decay = decay,
                           maxit = iters)
# Try SensAnalysisMLP
sens <- NeuralSens::SensAnalysisMLP(nnetmod, trData = nntrData, plot = FALSE)
```

DAILY_DEMAND_TR *Data frame with 4 variables*

Description

Training dataset with values of temperature and working day to predict electrical demand

Format

A data frame with 1980 rows and 4 variables:

DATE date of the measure

DEM electrical demand

WD Working Day: index which express how much work is made that day

TEMP weather temperature

Author(s)

Jose Portela Gonzalez

DAILY_DEMAND_TV *Data frame with 3 variables*

Description

Validation dataset with values of temperature and working day to predict electrical demand

Format

A data frame with 7 rows and 3 variables:

DATE date of the measure

WD Working Day: index which express how much work is made that day

TEMP weather temperature

Author(s)

Jose Portela Gonzalez

`Der2ActFunc` *Second derivative of activation function of neuron*

Description

Evaluate derivative of activation function of a neuron

Usage

```
Der2ActFunc(type = "sigmoid", ...)
```

Arguments

`type`	`character` name of the activation function
`...`	extra arguments needed to calculate the functions

Value

`numeric` output of the neuron

Examples

```
# Return derivative of the sigmoid activation function of a neuron
ActivationFunction <- Der2ActFunc("sigmoid")
# Return derivative of the tanh activation function of a neuron
ActivationFunction <- Der2ActFunc("tanh")
# Return derivative of the activation function of several layers of neurons
actfuncs <- c("linear","sigmoid","linear")
ActivationFunctions <- sapply(actfuncs, Der2ActFunc)
```

DerActFunc — *Derivative of activation function of neuron*

Description

Evaluate derivative of activation function of a neuron

Usage

```
DerActFunc(type = "sigmoid", ...)
```

Arguments

type	character name of the activation function
...	extra arguments needed to calculate the functions

Value

numeric output of the neuron

Examples

```
# Return derivative of the sigmoid activation function of a neuron
ActivationFunction <- DerActFunc("sigmoid")
# Return derivative of the tanh activation function of a neuron
ActivationFunction <- DerActFunc("tanh")
# Return derivative of the activation function of several layers of neurons
actfuncs <- c("linear","sigmoid","linear")
ActivationFunctions <- sapply(actfuncs, DerActFunc)
```

diag3Darray — *Define function to create a 'diagonal' array or get the diagonal of an array*

Description

Define function to create a 'diagonal' array or get the diagonal of an array

Usage

```
diag3Darray(x = 1, dim = length(x), out = "vector")
```

Arguments

x	`number` or `vector` defining the value of the diagonal of `3D array`
dim	`integer` defining the length of the diagonal. Default is `length(x)`. If `length(x) != 1`, `dim` must be equal to `length(x)`.
out	`character` specifying which type of diagonal to return (`"vector"` or `"matrix"`). See `Details`

Details

The diagonal of a 3D array has been defined as those elements in positions c(int,int,int), i.e., the three digits are the same.

If the diagonal should be returned, `out` specifies if it should return a `"vector"` with the elements of position c(int,int,int), or `"matrix"` with the elements of position c(int,dim,int), i.e., `dim = 2` -> elements (1,1,1),(2,1,2),(3,1,3),(1,2,1),(2,2,2),(3,2,3),(3,1,3),(3,2,3),(3,3,3).

Value

`array` with all elements zero except the diagonal, with dimensions c(dim,dim,dim)

Examples

```
x <- diag3Darray(c(1,4,6), dim = 3)
x
# , , 1
#
# [,1] [,2] [,3]
# [1,]    1    0    0
# [2,]    0    0    0
# [3,]    0    0    0
#
# , , 2
#
# [,1] [,2] [,3]
# [1,]    0    0    0
# [2,]    0    4    0
# [3,]    0    0    0
#
# , , 3
#
# [,1] [,2] [,3]
# [1,]    0    0    0
# [2,]    0    0    0
# [3,]    0    0    6
diag3Darray(x)
# 1, 4, 6
```

`diag3Darray<-` *Define function to change the diagonal of array*

Description

Define function to change the diagonal of array

Usage

```
diag3Darray(x) <- value
```

Arguments

x	3D array whose diagonal must be c hanged
value	vector defining the new values of diagonal.

Details

The diagonal of a 3D array has been defined as those elements in positions c(int,int,int), i.e., the three digits are the same.

Value

array with all elements zero except the diagonal, with dimensions c(dim,dim,dim)

Examples

```
x <- array(1, dim = c(3,3,3))
diag3Darray(x) <- c(2,2,2)
x
#  , , 1
#
#  [,1] [,2] [,3]
#  [1,]    2    1    1
#  [2,]    1    1    1
#  [3,]    1    1    1
#
#  , , 2
#
#  [,1] [,2] [,3]
#  [1,]    1    1    1
#  [2,]    1    2    1
#  [3,]    1    1    1
#
#  , , 3
#
#  [,1] [,2] [,3]
#  [1,]    1    1    1
#  [2,]    1    1    1
#  [3,]    1    1    2
```

HessianMLP *Sensitivity of MLP models*

Description

Function for evaluating the sensitivities of the inputs variables in a mlp model

Usage

```
HessianMLP(
  MLP.fit,
  .returnSens = TRUE,
  plot = TRUE,
  .rawSens = FALSE,
  sens_origin_layer = 1,
  sens_end_layer = "last",
  sens_origin_input = TRUE,
  sens_end_input = FALSE,
  ...
)

## Default S3 method:
HessianMLP(
  MLP.fit,
  .returnSens = TRUE,
  plot = TRUE,
  .rawSens = FALSE,
  sens_origin_layer = 1,
  sens_end_layer = "last",
  sens_origin_input = TRUE,
  sens_end_input = FALSE,
  trData,
  actfunc = NULL,
  deractfunc = NULL,
  der2actfunc = NULL,
  preProc = NULL,
  terms = NULL,
  output_name = NULL,
  ...
)

## S3 method for class 'train'
HessianMLP(
  MLP.fit,
  .returnSens = TRUE,
  plot = TRUE,
  .rawSens = FALSE,
```

```
  sens_origin_layer = 1,
  sens_end_layer = "last",
  sens_origin_input = TRUE,
  sens_end_input = FALSE,
  ...
)

## S3 method for class 'H2OMultinomialModel'
HessianMLP(
  MLP.fit,
  .returnSens = TRUE,
  plot = TRUE,
  .rawSens = FALSE,
  sens_origin_layer = 1,
  sens_end_layer = "last",
  sens_origin_input = TRUE,
  sens_end_input = FALSE,
  ...
)

## S3 method for class 'H2ORegressionModel'
HessianMLP(
  MLP.fit,
  .returnSens = TRUE,
  plot = TRUE,
  .rawSens = FALSE,
  sens_origin_layer = 1,
  sens_end_layer = "last",
  sens_origin_input = TRUE,
  sens_end_input = FALSE,
  ...
)

## S3 method for class 'list'
HessianMLP(
  MLP.fit,
  .returnSens = TRUE,
  plot = TRUE,
  .rawSens = FALSE,
  sens_origin_layer = 1,
  sens_end_layer = "last",
  sens_origin_input = TRUE,
  sens_end_input = FALSE,
  trData,
  actfunc,
  ...
)
```

```
## S3 method for class 'mlp'
HessianMLP(
  MLP.fit,
  .returnSens = TRUE,
  plot = TRUE,
  .rawSens = FALSE,
  sens_origin_layer = 1,
  sens_end_layer = "last",
  sens_origin_input = TRUE,
  sens_end_input = FALSE,
  trData,
  preProc = NULL,
  terms = NULL,
  ...
)

## S3 method for class 'nn'
HessianMLP(
  MLP.fit,
  .returnSens = TRUE,
  plot = TRUE,
  .rawSens = FALSE,
  sens_origin_layer = 1,
  sens_end_layer = "last",
  sens_origin_input = TRUE,
  sens_end_input = FALSE,
  preProc = NULL,
  terms = NULL,
  ...
)

## S3 method for class 'nnet'
HessianMLP(
  MLP.fit,
  .returnSens = TRUE,
  plot = TRUE,
  .rawSens = FALSE,
  sens_origin_layer = 1,
  sens_end_layer = "last",
  sens_origin_input = TRUE,
  sens_end_input = FALSE,
  trData,
  preProc = NULL,
  terms = NULL,
  ...
)

## S3 method for class 'nnetar'
```

```
HessianMLP(
  MLP.fit,
  .returnSens = TRUE,
  plot = TRUE,
  .rawSens = FALSE,
  sens_origin_layer = 1,
  sens_end_layer = "last",
  sens_origin_input = TRUE,
  sens_end_input = FALSE,
  ...
)

## S3 method for class 'numeric'
HessianMLP(
  MLP.fit,
  .returnSens = TRUE,
  plot = TRUE,
  .rawSens = FALSE,
  sens_origin_layer = 1,
  sens_end_layer = "last",
  sens_origin_input = TRUE,
  sens_end_input = FALSE,
  trData,
  actfunc = NULL,
  preProc = NULL,
  terms = NULL,
  ...
)
```

Arguments

`MLP.fit`	fitted neural network model
`.returnSens`	DEPRECATED
`plot`	`logical` whether or not to plot the analysis. By default is `TRUE`.
`.rawSens`	DEPRECATED
`sens_origin_layer`	`numeric` specifies the layer of neurons with respect to which the derivative must be calculated. The input layer is specified by 1 (default).
`sens_end_layer`	`numeric` specifies the layer of neurons of which the derivative is calculated. It may also be 'last' to specify the output layer (default).
`sens_origin_input`	`logical` specifies if the derivative must be calculated with respect to the inputs (`TRUE`) or output (`FALSE`) of the `sens_origin_layer` layer of the model. By default is `TRUE`.
`sens_end_input`	`logical` specifies if the derivative calculated is of the output (`FALSE`) or from the input (`TRUE`) of the `sens_end_layer` layer of the model. By default is `FALSE`.
`...`	additional arguments passed to or from other methods

trData	data.frame containing the data to evaluate the sensitivity of the model
actfunc	character vector indicating the activation function of each neurons layer.
deractfunc	character vector indicating the derivative of the activation function of each neurons layer.
der2actfunc	character vector indicating the second derivative of the activation function of each neurons layer.
preProc	preProcess structure applied to the training data. See also preProcess
terms	function applied to the training data to create factors. See also train
output_name	character name of the output variable in order to avoid changing the name of the output variable in trData to '.outcome'

Details

In case of using an input of class factor and a package which need to enter the input data as matrix, the dummies must be created before training the neural network.

After that, the training data must be given to the function using the trData argument.

Value

SensMLP object with the sensitivity metrics and sensitivities of the MLP model passed to the function.

Plots

- Plot 1: colorful plot with the classification of the classes in a 2D map
- Plot 2: b/w plot with probability of the chosen class in a 2D map
- Plot 3: plot with the stats::predictions of the data provided

References

https://www.researchgate.net/publication/220577792_Use_of_some_sensitivity_criteria_for_choosing_networks_with_good_generalization_ability

Examples

```
## Load data -------------------------------------------------------------------
data("DAILY_DEMAND_TR")
fdata <- DAILY_DEMAND_TR
## Parameters of the NNET ------------------------------------------------------
hidden_neurons <- 5
iters <- 100
decay <- 0.1

################################################################################
#########################  REGRESSION NNET #####################################
################################################################################
## Regression dataframe --------------------------------------------------------
# Scale the data
```

```
fdata.Reg.tr <- fdata[,2:ncol(fdata)]
fdata.Reg.tr[,3] <- fdata.Reg.tr[,3]/10
fdata.Reg.tr[,1] <- fdata.Reg.tr[,1]/1000

# Normalize the data for some models
preProc <- caret::preProcess(fdata.Reg.tr, method = c("center","scale"))
nntrData <- predict(preProc, fdata.Reg.tr)

#' ## TRAIN nnet NNET --------------------------------------------------------
# Create a formula to train NNET
form <- paste(names(fdata.Reg.tr)[2:ncol(fdata.Reg.tr)], collapse = " + ")
form <- formula(paste(names(fdata.Reg.tr)[1], form, sep = " ~ "))

set.seed(150)
nnetmod <- nnet::nnet(form,
                      data = nntrData,
                      linear.output = TRUE,
                      size = hidden_neurons,
                      decay = decay,
                      maxit = iters)
# Try HessianMLP
NeuralSens::HessianMLP(nnetmod, trData = nntrData, plot = FALSE)

# Try HessianMLP to calculate sensitivities with respect to output of hidden neurones
NeuralSens::HessianMLP(nnetmod, trData = nntrData,
                             sens_origin_layer = 2,
                             sens_end_layer = "last",
                             sens_origin_input = FALSE,
                             sens_end_input = FALSE)
## Train caret NNET ------------------------------------------------------------
# Create trainControl
ctrl_tune <- caret::trainControl(method = "boot",
                                 savePredictions = FALSE,
                                 summaryFunction = caret::defaultSummary)
set.seed(150) #For replication
caretmod <- caret::train(form = DEM~.,
                              data = fdata.Reg.tr,
                              method = "nnet",
                              linout = TRUE,
                              tuneGrid = data.frame(size = 3,
                                                    decay = decay),
                              maxit = iters,
                              preProcess = c("center","scale"),
                              trControl = ctrl_tune,
                              metric = "RMSE")

# Try HessianMLP
NeuralSens::HessianMLP(caretmod)

## Train h2o NNET --------------------------------------------------------------
# Create a cluster with 4 available cores
h2o::h2o.init(ip = "localhost",
              nthreads = 4,
```

```
              max_mem_size = "2g")

# Reset the cluster
h2o::h2o.removeAll()
fdata_h2o <- h2o::as.h2o(x = fdata.Reg.tr, destination_frame = "fdata_h2o")

set.seed(150)
h2omod <-h2o:: h2o.deeplearning(x = names(fdata.Reg.tr)[2:ncol(fdata.Reg.tr)],
                                     y = names(fdata.Reg.tr)[1],
                                     distribution = "AUTO",
                                     training_frame = fdata_h2o,
                                     standardize = TRUE,
                                     activation = "Tanh",
                                     hidden = c(hidden_neurons),
                                     stopping_rounds = 0,
                                     epochs = iters,
                                     seed = 150,
                                     model_id = "nnet_h2o",
                                     adaptive_rate = FALSE,
                                     rate_decay = decay,
                                     export_weights_and_biases = TRUE)

# Try HessianMLP
NeuralSens::HessianMLP(h2omod)

# Turn off the cluster
h2o::h2o.shutdown(prompt = FALSE)
rm(fdata_h2o)

## Train neural NNET ------------------------------------------------------------
set.seed(150)
neuralmod <- neural::mlptrain(as.matrix(nntrData[,2:ncol(nntrData)]),
                                   hidden_neurons,
                                   as.matrix(nntrData[1]),
                                   it=iters,
                                   visual=FALSE)

# Try HessianMLP
trData <- nntrData
NeuralSens::HessianMLP(neuralmod, trData = trData, output_name = "DEM")

## Train RSNNS NNET -------------------------------------------------------------
# Normalize data using RSNNS algorithms
trData <- as.data.frame(RSNNS::normalizeData(fdata.Reg.tr))
names(trData) <- names(fdata.Reg.tr)
set.seed(150)
RSNNSmod <-RSNNS::mlp(x = trData[,2:ncol(trData)],
                           y = trData[,1],
                           size = hidden_neurons,
                           linOut = TRUE,
                           learnFuncParams=c(decay),
                           maxit=iters)
```

```
# Try HessianMLP
NeuralSens::HessianMLP(RSNNSmod, trData = trData, output_name = "DEM")

## TRAIN neuralnet NNET --------------------------------------------------------
# Create a formula to train NNET
form <- paste(names(fdata.Reg.tr)[2:ncol(fdata.Reg.tr)], collapse = " + ")
form <- formula(paste(names(fdata.Reg.tr)[1], form, sep = " ~ "))

set.seed(150)
nnmod <- neuralnet::neuralnet(form,
                              nntrData,
                              linear.output = TRUE,
                              rep = 1,
                              hidden = hidden_neurons,
                              lifesign = "minimal",
                              threshold = 7,
                              stepmax = iters,
                              learningrate = decay,
                              act.fct = "tanh")

# Try HessianMLP
NeuralSens::HessianMLP(nnmod)


## USE DEFAULT METHOD ----------------------------------------------------------
NeuralSens::HessianMLP(caretmod$finalModel$wts,
                        trData = fdata.Reg.tv,
                        mlpstr = caretmod$finalModel$n,
                        coefnames = caretmod$coefnames,
                        actfun = c("linear","sigmoid","linear"),
                        output_name = "DEM")

################################################################################
#########################  CLASSIFICATION NNET #################################
################################################################################
## Regression dataframe --------------------------------------------------------
# Scale the data
fdata.Reg.cl <- fdata[,2:ncol(fdata)]
fdata.Reg.cl[,2:3] <- fdata.Reg.cl[,2:3]/10
fdata.Reg.cl[,1] <- fdata.Reg.cl[,1]/1000

# Normalize the data for some models
preProc <- caret::preProcess(fdata.Reg.cl, method = c("center","scale"))
nntrData <- predict(preProc, fdata.Reg.cl)

# Factorize the output
fdata.Reg.cl$DEM <- factor(round(fdata.Reg.cl$DEM, digits = 1))

# Normalize the data for some models
preProc <- caret::preProcess(fdata.Reg.cl, method = c("center","scale"))
nntrData <- predict(preProc, fdata.Reg.cl)

## Train caret NNET ------------------------------------------------------------
```

```
# Create trainControl
ctrl_tune <- caret::trainControl(method = "boot",
                                 savePredictions = FALSE,
                                 summaryFunction = caret::defaultSummary)
set.seed(150) #For replication
caretmod <- caret::train(form = DEM~.,
                                data = fdata.Reg.cl,
                                method = "nnet",
                                linout = FALSE,
                                tuneGrid = data.frame(size = hidden_neurons,
                                                      decay = decay),
                                maxit = iters,
                                preProcess = c("center","scale"),
                                trControl = ctrl_tune,
                                metric = "Accuracy")

# Try HessianMLP
NeuralSens::HessianMLP(caretmod)

## Train h2o NNET --------------------------------------------------------------
# Create local cluster with 4 available cores
h2o::h2o.init(ip = "localhost",
              nthreads = 4,
              max_mem_size = "2g")

# Reset the cluster
h2o::h2o.removeAll()
fdata_h2o <- h2o::as.h2o(x = fdata.Reg.cl, destination_frame = "fdata_h2o")

set.seed(150)
h2omod <- h2o::h2o.deeplearning(x = names(fdata.Reg.cl)[2:ncol(fdata.Reg.cl)],
                                       y = names(fdata.Reg.cl)[1],
                                       distribution = "AUTO",
                                       training_frame = fdata_h2o,
                                       standardize = TRUE,
                                       activation = "Tanh",
                                       hidden = c(hidden_neurons),
                                       stopping_rounds = 0,
                                       epochs = iters,
                                       seed = 150,
                                       model_id = "nnet_h2o",
                                       adaptive_rate = FALSE,
                                       rate_decay = decay,
                                       export_weights_and_biases = TRUE)

# Try HessianMLP
NeuralSens::HessianMLP(h2omod)

# Apaga el cluster
h2o::h2o.shutdown(prompt = FALSE)
rm(fdata_h2o)

## Train neural NNET -----------------------------------------------------------
```

```
# set.seed(150)
# neuralmod <-mlptrain(as.matrix(nntrData[,2:ncol(nntrData)]),
#                           hidden_neurons,
#                           as.matrix(nntrData[1]),
#                           it=iters,
#                           visual=FALSE)
#
# # Try HessianMLP
# NeuralSens::HessianMLP(neuralmod, trData = trData)

# ## Train RSNNS NNET ------------------------------------------------------------
# # Normalize data using RSNNS algorithms
# trData <- as.data.frame(RSNNS::normalizeData(fdata.Reg.cl))
# names(trData) <- names(fdata.Reg.tr)
# set.seed(150)
# RSNNSmod <- RSNNS::mlp(x = trData[,2:ncol(trData)],
#                      y = trData[,1],
#                      size = hidden_neurons,
#                      linOut = FALSE,
#                      learnFuncParams=c(decay),
#                      maxit=iters)
#
# # Try HessianMLP
# NeuralSens::HessianMLP(RSNNSmod, trData = trData, output_name = "DEM")

## TRAIN neuralnet NNET --------------------------------------------------------
# Create a formula to train NNET
# form <- paste(names(fdata.Reg.tr)[2:ncol(fdata.Reg.tr)], collapse = " + ")
# form <- formula(paste(names(fdata.Reg.tr)[1], form, sep = " ~ "))
#
# set.seed(150)
# nnmod <- neuralnet(form,
#                    nntrData,
#                    linear.output = FALSE,
#                    rep = 1,
#                    hidden = hidden_neurons,
#                    lifesign = "minimal",
#                    threshold = 4,
#                    stepmax = iters,
#                    learningrate = decay,
#                    act.fct = "tanh")
#
# # Try HessianMLP
# NeuralSens::HessianMLP(nnmod)

## TRAIN nnet NNET --------------------------------------------------------
# Create a formula to train NNET
form <- paste(names(fdata.Reg.tr)[2:ncol(fdata.Reg.tr)], collapse = " + ")
form <- formula(paste(names(fdata.Reg.tr)[1], form, sep = " ~ "))

set.seed(150)
nnetmod <- nnet::nnet(form,
                      data = nntrData,
```

```
                      linear.output = TRUE,
                      size = hidden_neurons,
                      decay = decay,
                      maxit = iters)
# Try HessianMLP
NeuralSens::HessianMLP(nnetmod, trData = nntrData)
```

HessMLP *Constructor of the HessMLP Class*

Description

Create an object of HessMLP class

Usage

```
HessMLP(
  sens = list(),
  raw_sens = list(),
  mlp_struct = numeric(),
  trData = data.frame(),
  coefnames = character(),
  output_name = character()
)
```

Arguments

`sens`	`list` of sensitivity measures, one `list` per output neuron
`raw_sens`	`list` of sensitivities, one `array` per output neuron
`mlp_struct`	`numeric` vector describing the structur of the MLP model
`trData`	`data.frame` with the data used to calculate the sensitivities
`coefnames`	`character` vector with the name of the predictor(s)
`output_name`	`character` vector with the name of the output(s)

Value

`HessMLP` object

HessToSensMLP *Convert a HessMLP to a SensMLP object*

Description

Auxiliary function to turn a HessMLP object to a SensMLP object in order to use the plot-related functions associated with SensMLP

Usage

```
HessToSensMLP(x)
```

Arguments

x	HessMLP object

Value

SensMLP object

is.HessMLP *Check if object is of class* HessMLP

Description

Check if object is of class HessMLP

Usage

```
is.HessMLP(object)
```

Arguments

object	HessMLP object

Value

TRUE if object is a HessMLP object

is.SensMLP *Check if object is of class* SensMLP

Description

Check if object is of class SensMLP

Usage

```
is.SensMLP(object)
```

Arguments

object	SensMLP object

Value

TRUE if object is a SensMLP object

NeuralSens *NeuralSens: Sensitivity Analysis of Neural Networks*

Description

Visualization and analysis tools to aid in the interpretation of neural network models.

plot.HessMLP *Plot method for the HessMLP Class*

Description

Plot the sensitivities and sensitivity metrics of a HessMLP object.

Usage

```
## S3 method for class 'HessMLP'
plot(x, plotType = c("sensitivities", "time", "matrix", "interactions"), ...)
```

Arguments

- x — HessMLP object created by HessianMLP
- plotType — character specifying which type of plot should be created. It can be:
  - "sensitivities" (default): use HessianMLP function
  - "time": use SensTimePlot function
  - "features": use SensFeaturePlot function
- ... — additional parameters passed to plot function of the NeuralSens package

Value

list of graphic objects created by ggplot

Examples

```
#' ## Load data -------------------------------------------------------------------
data("DAILY_DEMAND_TR")
fdata <- DAILY_DEMAND_TR

## Parameters of the NNET ------------------------------------------------------
hidden_neurons <- 5
iters <- 250
decay <- 0.1

################################################################################
#########################  REGRESSION NNET #####################################
################################################################################
## Regression dataframe --------------------------------------------------------
# Scale the data
fdata.Reg.tr <- fdata[,2:ncol(fdata)]
fdata.Reg.tr[,3] <- fdata.Reg.tr[,3]/10
fdata.Reg.tr[,1] <- fdata.Reg.tr[,1]/1000

# Normalize the data for some models
preProc <- caret::preProcess(fdata.Reg.tr, method = c("center","scale"))
nntrData <- predict(preProc, fdata.Reg.tr)

#' ## TRAIN nnet NNET --------------------------------------------------------
# Create a formula to train NNET
form <- paste(names(fdata.Reg.tr)[2:ncol(fdata.Reg.tr)], collapse = " + ")
form <- formula(paste(names(fdata.Reg.tr)[1], form, sep = " ~ "))

set.seed(150)
nnetmod <- nnet::nnet(form,
                           data = nntrData,
                           linear.output = TRUE,
                           size = hidden_neurons,
                           decay = decay,
                           maxit = iters)
# Try HessianMLP
sens <- NeuralSens::HessianMLP(nnetmod, trData = nntrData, plot = FALSE)
## Not run:
plot(sens)
plot(sens,"time")

## End(Not run)
```

---

plot.SensMLP *Plot method for the SensMLP Class*

Description

Plot the sensitivities and sensitivity metrics of a SensMLP object.

Usage

```
## S3 method for class 'SensMLP'
plot(x, plotType = c("sensitivities", "time", "features"), ...)
```

Arguments

x	SensMLP object created by SensAnalysisMLP
plotType	character specifying which type of plot should be created. It can be: • "sensitivities" (default): use SensAnalysisMLP function • "time": use SensTimePlot function • "features": use SensFeaturePlot function
...	additional parameters passed to plot function of the NeuralSens package

Value

list of graphic objects created by ggplot

Examples

```
#' ## Load data -------------------------------------------------------------------
data("DAILY_DEMAND_TR")
fdata <- DAILY_DEMAND_TR

## Parameters of the NNET ------------------------------------------------------
hidden_neurons <- 5
iters <- 250
decay <- 0.1

################################################################################
#########################  REGRESSION NNET #####################################
################################################################################
## Regression dataframe --------------------------------------------------------
# Scale the data
fdata.Reg.tr <- fdata[,2:ncol(fdata)]
fdata.Reg.tr[,3] <- fdata.Reg.tr[,3]/10
fdata.Reg.tr[,1] <- fdata.Reg.tr[,1]/1000

# Normalize the data for some models
preProc <- caret::preProcess(fdata.Reg.tr, method = c("center","scale"))
nntrData <- predict(preProc, fdata.Reg.tr)

#' ## TRAIN nnet NNET --------------------------------------------------------
# Create a formula to train NNET
form <- paste(names(fdata.Reg.tr)[2:ncol(fdata.Reg.tr)], collapse = " + ")
form <- formula(paste(names(fdata.Reg.tr)[1], form, sep = " ~ "))
```

```
set.seed(150)
nnetmod <- nnet::nnet(form,
                           data = nntrData,
                           linear.output = TRUE,
                           size = hidden_neurons,
                           decay = decay,
                           maxit = iters)
# Try SensAnalysisMLP
sens <- NeuralSens::SensAnalysisMLP(nnetmod, trData = nntrData, plot = FALSE)
## Not run:
plot(sens)
plot(sens,"time")
plot(sens,"features")

## End(Not run)
```

`PlotSensMLP` *Neural network structure sensitivity plot*

Description

Plot a neural interpretation diagram colored by sensitivities of the model

Usage

```
PlotSensMLP(
  MLP.fit,
  metric = "mean",
  sens_neg_col = "red",
  sens_pos_col = "blue",
  ...
)
```

Arguments

`MLP.fit`	fitted neural network model
`metric`	metric to plot in the NID. It can be "mean" (default), "median or "sqmean". It can be any metric to combine the raw sensitivities
`sens_neg_col`	`character` string indicating color of negative sensitivity measure, default 'red'. The same is passed to argument `neg_col` of plotnet
`sens_pos_col`	`character` string indicating color of positive sensitivity measure, default 'blue'. The same is passed to argument `pos_col` of plotnet
`...`	additional arguments passed to plotnet and/or SensAnalysisMLP

Value

A graphics object

Examples

```
## Load data -------------------------------------------------------------------
data("DAILY_DEMAND_TR")
fdata <- DAILY_DEMAND_TR
## Parameters of the NNET ------------------------------------------------------
hidden_neurons <- 5
iters <- 100
decay <- 0.1

################################################################################
#########################  REGRESSION NNET #####################################
################################################################################
## Regression dataframe --------------------------------------------------------
# Scale the data
fdata.Reg.tr <- fdata[,2:ncol(fdata)]
fdata.Reg.tr[,3] <- fdata.Reg.tr[,3]/10
fdata.Reg.tr[,1] <- fdata.Reg.tr[,1]/1000

# Normalize the data for some models
preProc <- caret::preProcess(fdata.Reg.tr, method = c("center","scale"))
nntrData <- predict(preProc, fdata.Reg.tr)

#' ## TRAIN nnet NNET --------------------------------------------------------
# Create a formula to train NNET
form <- paste(names(fdata.Reg.tr)[2:ncol(fdata.Reg.tr)], collapse = " + ")
form <- formula(paste(names(fdata.Reg.tr)[1], form, sep = " ~ "))

set.seed(150)
nnetmod <- nnet::nnet(form,
                      data = nntrData,
                      linear.output = TRUE,
                      size = hidden_neurons,
                      decay = decay,
                      maxit = iters)
# Try SensAnalysisMLP
NeuralSens::PlotSensMLP(nnetmod, trData = nntrData)
```

`print.HessMLP` *Print method for the HessMLP Class*

Description

Print the sensitivities of a `HessMLP` object.

Usage

```
## S3 method for class 'HessMLP'
print(x, n = 5, ...)
```

Arguments

x	HessMLP object created by HessianMLP
n	integer specifying number of sensitivities to print per each output
...	additional parameters

Examples

```
## Load data -------------------------------------------------------------------
data("DAILY_DEMAND_TR")
fdata <- DAILY_DEMAND_TR

## Parameters of the NNET ------------------------------------------------------
hidden_neurons <- 5
iters <- 250
decay <- 0.1

################################################################################
#########################  REGRESSION NNET #####################################
################################################################################
## Regression dataframe --------------------------------------------------------
# Scale the data
fdata.Reg.tr <- fdata[,2:ncol(fdata)]
fdata.Reg.tr[,3] <- fdata.Reg.tr[,3]/10
fdata.Reg.tr[,1] <- fdata.Reg.tr[,1]/1000

# Normalize the data for some models
preProc <- caret::preProcess(fdata.Reg.tr, method = c("center","scale"))
nntrData <- predict(preProc, fdata.Reg.tr)

#' ## TRAIN nnet NNET --------------------------------------------------------
# Create a formula to train NNET
form <- paste(names(fdata.Reg.tr)[2:ncol(fdata.Reg.tr)], collapse = " + ")
form <- formula(paste(names(fdata.Reg.tr)[1], form, sep = " ~ "))

set.seed(150)
nnetmod <- nnet::nnet(form,
                           data = nntrData,
                           linear.output = TRUE,
                           size = hidden_neurons,
                           decay = decay,
                           maxit = iters)
# Try HessianMLP
sens <- NeuralSens::HessianMLP(nnetmod, trData = nntrData, plot = FALSE)
sens
```

---

print.SensMLP *Print method for the SensMLP Class*

Description

Print the sensitivities of a SensMLP object.

Usage

```
## S3 method for class 'SensMLP'
print(x, n = 5, ...)
```

Arguments

x	SensMLP object created by SensAnalysisMLP
n	integer specifying number of sensitivities to print per each output
...	additional parameters

Examples

```
## Load data -------------------------------------------------------------------
data("DAILY_DEMAND_TR")
fdata <- DAILY_DEMAND_TR

## Parameters of the NNET ------------------------------------------------------
hidden_neurons <- 5
iters <- 250
decay <- 0.1

################################################################################
#########################  REGRESSION NNET #####################################
################################################################################
## Regression dataframe --------------------------------------------------------
# Scale the data
fdata.Reg.tr <- fdata[,2:ncol(fdata)]
fdata.Reg.tr[,3] <- fdata.Reg.tr[,3]/10
fdata.Reg.tr[,1] <- fdata.Reg.tr[,1]/1000

# Normalize the data for some models
preProc <- caret::preProcess(fdata.Reg.tr, method = c("center","scale"))
nntrData <- predict(preProc, fdata.Reg.tr)

#' ## TRAIN nnet NNET --------------------------------------------------------
# Create a formula to train NNET
form <- paste(names(fdata.Reg.tr)[2:ncol(fdata.Reg.tr)], collapse = " + ")
form <- formula(paste(names(fdata.Reg.tr)[1], form, sep = " ~ "))

set.seed(150)
nnetmod <- nnet::nnet(form,
                           data = nntrData,
                           linear.output = TRUE,
                           size = hidden_neurons,
                           decay = decay,
                           maxit = iters)
# Try SensAnalysisMLP
```

```
sens <- NeuralSens::SensAnalysisMLP(nnetmod, trData = nntrData, plot = FALSE)
sens
```

---

print.summary.HessMLP *Print method of the summary HessMLP Class*

Description

Print the sensitivity metrics of a HessMLP object. This metrics are the mean sensitivity, the standard deviation of sensitivities and the mean of sensitivities square

Usage

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

Arguments

x	summary.HessMLP object created by summary method of HessMLP object
...	additional parameters

Examples

```
## Load data -------------------------------------------------------------------
data("DAILY_DEMAND_TR")
fdata <- DAILY_DEMAND_TR

## Parameters of the NNET ------------------------------------------------------
hidden_neurons <- 5
iters <- 250
decay <- 0.1

################################################################################
#########################  REGRESSION NNET #####################################
################################################################################
## Regression dataframe --------------------------------------------------------
# Scale the data
fdata.Reg.tr <- fdata[,2:ncol(fdata)]
fdata.Reg.tr[,3] <- fdata.Reg.tr[,3]/10
fdata.Reg.tr[,1] <- fdata.Reg.tr[,1]/1000

# Normalize the data for some models
preProc <- caret::preProcess(fdata.Reg.tr, method = c("center","scale"))
nntrData <- predict(preProc, fdata.Reg.tr)

#' ## TRAIN nnet NNET --------------------------------------------------------
# Create a formula to train NNET
form <- paste(names(fdata.Reg.tr)[2:ncol(fdata.Reg.tr)], collapse = " + ")
form <- formula(paste(names(fdata.Reg.tr)[1], form, sep = " ~ "))
```

```
set.seed(150)
nnetmod <- nnet::nnet(form,
                           data = nntrData,
                           linear.output = TRUE,
                           size = hidden_neurons,
                           decay = decay,
                           maxit = iters)
# Try HessianMLP
sens <- NeuralSens::HessianMLP(nnetmod, trData = nntrData, plot = FALSE)
print(summary(sens))
```

`print.summary.SensMLP` *Print method of the summary SensMLP Class*

Description

Print the sensitivity metrics of a `SensMLP` object. This metrics are the mean sensitivity, the standard deviation of sensitivities and the mean of sensitivities square

Usage

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

Arguments

x	`summary.SensMLP` object created by summary method of `SensMLP` object
...	additional parameters

Examples

```
## Load data -------------------------------------------------------------------
data("DAILY_DEMAND_TR")
fdata <- DAILY_DEMAND_TR

## Parameters of the NNET ------------------------------------------------------
hidden_neurons <- 5
iters <- 250
decay <- 0.1

################################################################################
#########################  REGRESSION NNET #####################################
################################################################################
## Regression dataframe --------------------------------------------------------
# Scale the data
fdata.Reg.tr <- fdata[,2:ncol(fdata)]
fdata.Reg.tr[,3] <- fdata.Reg.tr[,3]/10
fdata.Reg.tr[,1] <- fdata.Reg.tr[,1]/1000
```

```
# Normalize the data for some models
preProc <- caret::preProcess(fdata.Reg.tr, method = c("center","scale"))
nntrData <- predict(preProc, fdata.Reg.tr)

#' ## TRAIN nnet NNET --------------------------------------------------------
# Create a formula to train NNET
form <- paste(names(fdata.Reg.tr)[2:ncol(fdata.Reg.tr)], collapse = " + ")
form <- formula(paste(names(fdata.Reg.tr)[1], form, sep = " ~ "))

set.seed(150)
nnetmod <- nnet::nnet(form,
                           data = nntrData,
                           linear.output = TRUE,
                           size = hidden_neurons,
                           decay = decay,
                           maxit = iters)
# Try SensAnalysisMLP
sens <- NeuralSens::SensAnalysisMLP(nnetmod, trData = nntrData, plot = FALSE)
print(summary(sens))
```

SensAnalysisMLP *Sensitivity of MLP models*

Description

Function for evaluating the sensitivities of the inputs variables in a mlp model

Usage

```
SensAnalysisMLP(
  MLP.fit,
  .returnSens = TRUE,
  plot = TRUE,
  .rawSens = FALSE,
  sens_origin_layer = 1,
  sens_end_layer = "last",
  sens_origin_input = TRUE,
  sens_end_input = FALSE,
  ...
)

## Default S3 method:
SensAnalysisMLP(
  MLP.fit,
  .returnSens = TRUE,
  plot = TRUE,
  .rawSens = FALSE,
  sens_origin_layer = 1,
```

```
  sens_end_layer = "last",
  sens_origin_input = TRUE,
  sens_end_input = FALSE,
  trData,
  actfunc = NULL,
  deractfunc = NULL,
  preProc = NULL,
  terms = NULL,
  output_name = NULL,
  ...
)

## S3 method for class 'train'
SensAnalysisMLP(
  MLP.fit,
  .returnSens = TRUE,
  plot = TRUE,
  .rawSens = FALSE,
  sens_origin_layer = 1,
  sens_end_layer = "last",
  sens_origin_input = TRUE,
  sens_end_input = FALSE,
  ...
)

## S3 method for class 'H2OMultinomialModel'
SensAnalysisMLP(
  MLP.fit,
  .returnSens = TRUE,
  plot = TRUE,
  .rawSens = FALSE,
  sens_origin_layer = 1,
  sens_end_layer = "last",
  sens_origin_input = TRUE,
  sens_end_input = FALSE,
  ...
)

## S3 method for class 'H2ORegressionModel'
SensAnalysisMLP(
  MLP.fit,
  .returnSens = TRUE,
  plot = TRUE,
  .rawSens = FALSE,
  sens_origin_layer = 1,
  sens_end_layer = "last",
  sens_origin_input = TRUE,
  sens_end_input = FALSE,
```

```
  ...
)

## S3 method for class 'list'
SensAnalysisMLP(
  MLP.fit,
  .returnSens = TRUE,
  plot = TRUE,
  .rawSens = FALSE,
  sens_origin_layer = 1,
  sens_end_layer = "last",
  sens_origin_input = TRUE,
  sens_end_input = FALSE,
  trData,
  actfunc,
  ...
)

## S3 method for class 'mlp'
SensAnalysisMLP(
  MLP.fit,
  .returnSens = TRUE,
  plot = TRUE,
  .rawSens = FALSE,
  sens_origin_layer = 1,
  sens_end_layer = "last",
  sens_origin_input = TRUE,
  sens_end_input = FALSE,
  trData,
  preProc = NULL,
  terms = NULL,
  ...
)

## S3 method for class 'nn'
SensAnalysisMLP(
  MLP.fit,
  .returnSens = TRUE,
  plot = TRUE,
  .rawSens = FALSE,
  sens_origin_layer = 1,
  sens_end_layer = "last",
  sens_origin_input = TRUE,
  sens_end_input = FALSE,
  preProc = NULL,
  terms = NULL,
  ...
)
```

```
## S3 method for class 'nnet'
SensAnalysisMLP(
  MLP.fit,
  .returnSens = TRUE,
  plot = TRUE,
  .rawSens = FALSE,
  sens_origin_layer = 1,
  sens_end_layer = "last",
  sens_origin_input = TRUE,
  sens_end_input = FALSE,
  trData,
  preProc = NULL,
  terms = NULL,
  ...
)

## S3 method for class 'nnetar'
SensAnalysisMLP(
  MLP.fit,
  .returnSens = TRUE,
  plot = TRUE,
  .rawSens = FALSE,
  sens_origin_layer = 1,
  sens_end_layer = "last",
  sens_origin_input = TRUE,
  sens_end_input = FALSE,
  ...
)

## S3 method for class 'numeric'
SensAnalysisMLP(
  MLP.fit,
  .returnSens = TRUE,
  plot = TRUE,
  .rawSens = FALSE,
  sens_origin_layer = 1,
  sens_end_layer = "last",
  sens_origin_input = TRUE,
  sens_end_input = FALSE,
  trData,
  actfunc = NULL,
  preProc = NULL,
  terms = NULL,
  ...
)
```

Arguments

`MLP.fit`	fitted neural network model
`.returnSens`	DEPRECATED
`plot`	`logical` whether or not to plot the analysis. By default is `TRUE`.
`.rawSens`	DEPRECATED
`sens_origin_layer`	`numeric` specifies the layer of neurons with respect to which the derivative must be calculated. The input layer is specified by 1 (default).
`sens_end_layer`	`numeric` specifies the layer of neurons of which the derivative is calculated. It may also be 'last' to specify the output layer (default).
`sens_origin_input`	`logical` specifies if the derivative must be calculated with respect to the inputs (`TRUE`) or output (`FALSE`) of the `sens_origin_layer` layer of the model. By default is `TRUE`.
`sens_end_input`	`logical` specifies if the derivative calculated is of the output (`FALSE`) or from the input (`TRUE`) of the `sens_end_layer` layer of the model. By default is `FALSE`.
`...`	additional arguments passed to or from other methods
`trData`	`data.frame` containing the data to evaluate the sensitivity of the model
`actfunc`	`character` vector indicating the activation function of each neurons layer.
`deractfunc`	`character` vector indicating the derivative of the activation function of each neurons layer.
`preProc`	preProcess structure applied to the training data. See also `preProcess`
`terms`	function applied to the training data to create factors. See also `train`
`output_name`	`character` name of the output variable in order to avoid changing the name of the output variable in `trData` to '.outcome'

Details

In case of using an input of class `factor` and a package which need to enter the input data as matrix, the dummies must be created before training the neural network.

After that, the training data must be given to the function using the `trData` argument.

Value

`SensMLP` object with the sensitivity metrics and sensitivities of the MLP model passed to the function.

Plots

- Plot 1: colorful plot with the classification of the classes in a 2D map
- Plot 2: b/w plot with probability of the chosen class in a 2D map
- Plot 3: plot with the stats::predictions of the data provided

References

https://www.researchgate.net/publication/220577792_Use_of_some_sensitivity_criteria_for_choosing_networks_with_good_generalization_ability

Examples

```
## Load data -------------------------------------------------------------------
data("DAILY_DEMAND_TR")
fdata <- DAILY_DEMAND_TR
## Parameters of the NNET ------------------------------------------------------
hidden_neurons <- 5
iters <- 100
decay <- 0.1

################################################################################
#########################  REGRESSION NNET #####################################
################################################################################
## Regression dataframe --------------------------------------------------------
# Scale the data
fdata.Reg.tr <- fdata[,2:ncol(fdata)]
fdata.Reg.tr[,3] <- fdata.Reg.tr[,3]/10
fdata.Reg.tr[,1] <- fdata.Reg.tr[,1]/1000

# Normalize the data for some models
preProc <- caret::preProcess(fdata.Reg.tr, method = c("center","scale"))
nntrData <- predict(preProc, fdata.Reg.tr)

#' ## TRAIN nnet NNET --------------------------------------------------------
# Create a formula to train NNET
form <- paste(names(fdata.Reg.tr)[2:ncol(fdata.Reg.tr)], collapse = " + ")
form <- formula(paste(names(fdata.Reg.tr)[1], form, sep = " ~ "))

set.seed(150)
nnetmod <- nnet::nnet(form,
                      data = nntrData,
                      linear.output = TRUE,
                      size = hidden_neurons,
                      decay = decay,
                      maxit = iters)
# Try SensAnalysisMLP
NeuralSens::SensAnalysisMLP(nnetmod, trData = nntrData)

# Try SensAnalysisMLP to calculate sensitivities with respect to output of hidden neurones
NeuralSens::SensAnalysisMLP(nnetmod, trData = nntrData,
                             sens_origin_layer = 2,
                             sens_end_layer = "last",
                             sens_origin_input = FALSE,
                             sens_end_input = FALSE)
## Train caret NNET ------------------------------------------------------------
# Create trainControl
ctrl_tune <- caret::trainControl(method = "boot",
                                 savePredictions = FALSE,
```

```
                                 summaryFunction = caret::defaultSummary)
set.seed(150) #For replication
caretmod <- caret::train(form = DEM~.,
                              data = fdata.Reg.tr,
                              method = "nnet",
                              linout = TRUE,
                              tuneGrid = data.frame(size = 3,
                                                    decay = decay),
                              maxit = iters,
                              preProcess = c("center","scale"),
                              trControl = ctrl_tune,
                              metric = "RMSE")

# Try SensAnalysisMLP
NeuralSens::SensAnalysisMLP(caretmod)

## Train h2o NNET --------------------------------------------------------------
# Create a cluster with 4 available cores
h2o::h2o.init(ip = "localhost",
              nthreads = 4,
              max_mem_size = "2g")

# Reset the cluster
h2o::h2o.removeAll()
fdata_h2o <- h2o::as.h2o(x = fdata.Reg.tr, destination_frame = "fdata_h2o")

set.seed(150)
h2omod <-h2o:: h2o.deeplearning(x = names(fdata.Reg.tr)[2:ncol(fdata.Reg.tr)],
                                     y = names(fdata.Reg.tr)[1],
                                     distribution = "AUTO",
                                     training_frame = fdata_h2o,
                                     standardize = TRUE,
                                     activation = "Tanh",
                                     hidden = c(hidden_neurons),
                                     stopping_rounds = 0,
                                     epochs = iters,
                                     seed = 150,
                                     model_id = "nnet_h2o",
                                     adaptive_rate = FALSE,
                                     rate_decay = decay,
                                     export_weights_and_biases = TRUE)

# Try SensAnalysisMLP
NeuralSens::SensAnalysisMLP(h2omod)

# Turn off the cluster
h2o::h2o.shutdown(prompt = FALSE)
rm(fdata_h2o)

## Train neural NNET -----------------------------------------------------------
set.seed(150)
neuralmod <- neural::mlptrain(as.matrix(nntrData[,2:ncol(nntrData)]),
                                   hidden_neurons,
```

```
                                     as.matrix(nntrData[1]),
                                     it=iters,
                                     visual=FALSE)

# Try SensAnalysisMLP
trData <- nntrData
NeuralSens::SensAnalysisMLP(neuralmod, trData = trData, output_name = "DEM")

## Train RSNNS NNET ------------------------------------------------------------
# Normalize data using RSNNS algorithms
trData <- as.data.frame(RSNNS::normalizeData(fdata.Reg.tr))
names(trData) <- names(fdata.Reg.tr)
set.seed(150)
RSNNSmod <-RSNNS::mlp(x = trData[,2:ncol(trData)],
                           y = trData[,1],
                           size = hidden_neurons,
                           linOut = TRUE,
                           learnFuncParams=c(decay),
                           maxit=iters)

# Try SensAnalysisMLP
NeuralSens::SensAnalysisMLP(RSNNSmod, trData = trData, output_name = "DEM")

## TRAIN neuralnet NNET --------------------------------------------------------
# Create a formula to train NNET
form <- paste(names(fdata.Reg.tr)[2:ncol(fdata.Reg.tr)], collapse = " + ")
form <- formula(paste(names(fdata.Reg.tr)[1], form, sep = " ~ "))

set.seed(150)
nnmod <- neuralnet::neuralnet(form,
                                   nntrData,
                                   linear.output = TRUE,
                                   rep = 1,
                                   hidden = hidden_neurons,
                                   lifesign = "minimal",
                                   threshold = 7,
                                   stepmax = iters,
                                   learningrate = decay,
                                   act.fct = "tanh")

# Try SensAnalysisMLP
NeuralSens::SensAnalysisMLP(nnmod)


## USE DEFAULT METHOD ----------------------------------------------------------
NeuralSens::SensAnalysisMLP(caretmod$finalModel$wts,
                            trData = fdata.Reg.tv,
                            mlpstr = caretmod$finalModel$n,
                            coefnames = caretmod$coefnames,
                            actfun = c("linear","sigmoid","linear"),
                            output_name = "DEM")

################################################################################
```

```
#########################  CLASSIFICATION NNET #################################
################################################################################
## Regression dataframe --------------------------------------------------------
# Scale the data
fdata.Reg.cl <- fdata[,2:ncol(fdata)]
fdata.Reg.cl[,2:3] <- fdata.Reg.cl[,2:3]/10
fdata.Reg.cl[,1] <- fdata.Reg.cl[,1]/1000

# Normalize the data for some models
preProc <- caret::preProcess(fdata.Reg.cl, method = c("center","scale"))
nntrData <- predict(preProc, fdata.Reg.cl)

# Factorize the output
fdata.Reg.cl$DEM <- factor(round(fdata.Reg.cl$DEM, digits = 1))

# Normalize the data for some models
preProc <- caret::preProcess(fdata.Reg.cl, method = c("center","scale"))
nntrData <- predict(preProc, fdata.Reg.cl)

## Train caret NNET ------------------------------------------------------------
# Create trainControl
ctrl_tune <- caret::trainControl(method = "boot",
                                 savePredictions = FALSE,
                                 summaryFunction = caret::defaultSummary)
set.seed(150) #For replication
caretmod <- caret::train(form = DEM~.,
                                data = fdata.Reg.cl,
                                method = "nnet",
                                linout = FALSE,
                                tuneGrid = data.frame(size = hidden_neurons,
                                                      decay = decay),
                                maxit = iters,
                                preProcess = c("center","scale"),
                                trControl = ctrl_tune,
                                metric = "Accuracy")

# Try SensAnalysisMLP
NeuralSens::SensAnalysisMLP(caretmod)

## Train h2o NNET --------------------------------------------------------------
# Create local cluster with 4 available cores
h2o::h2o.init(ip = "localhost",
              nthreads = 4,
              max_mem_size = "2g")

# Reset the cluster
h2o::h2o.removeAll()
fdata_h2o <- h2o::as.h2o(x = fdata.Reg.cl, destination_frame = "fdata_h2o")

set.seed(150)
h2omod <- h2o::h2o.deeplearning(x = names(fdata.Reg.cl)[2:ncol(fdata.Reg.cl)],
                                       y = names(fdata.Reg.cl)[1],
                                       distribution = "AUTO",
```

```
                                        training_frame = fdata_h2o,
                                        standardize = TRUE,
                                        activation = "Tanh",
                                        hidden = c(hidden_neurons),
                                        stopping_rounds = 0,
                                        epochs = iters,
                                        seed = 150,
                                        model_id = "nnet_h2o",
                                        adaptive_rate = FALSE,
                                        rate_decay = decay,
                                        export_weights_and_biases = TRUE)

# Try SensAnalysisMLP
NeuralSens::SensAnalysisMLP(h2omod)

# Apaga el cluster
h2o::h2o.shutdown(prompt = FALSE)
rm(fdata_h2o)

## Train neural NNET -----------------------------------------------------------
# set.seed(150)
# neuralmod <-mlptrain(as.matrix(nntrData[,2:ncol(nntrData)]),
#                            hidden_neurons,
#                            as.matrix(nntrData[1]),
#                            it=iters,
#                            visual=FALSE)
#
# # Try SensAnalysisMLP
# NeuralSens::SensAnalysisMLP(neuralmod, trData = trData)

# ## Train RSNNS NNET ------------------------------------------------------------
# # Normalize data using RSNNS algorithms
# trData <- as.data.frame(RSNNS::normalizeData(fdata.Reg.cl))
# names(trData) <- names(fdata.Reg.tr)
# set.seed(150)
# RSNNSmod <- RSNNS::mlp(x = trData[,2:ncol(trData)],
#                       y = trData[,1],
#                       size = hidden_neurons,
#                       linOut = FALSE,
#                       learnFuncParams=c(decay),
#                       maxit=iters)
#
# # Try SensAnalysisMLP
# NeuralSens::SensAnalysisMLP(RSNNSmod, trData = trData, output_name = "DEM")

## TRAIN neuralnet NNET --------------------------------------------------------
# Create a formula to train NNET
# form <- paste(names(fdata.Reg.tr)[2:ncol(fdata.Reg.tr)], collapse = " + ")
# form <- formula(paste(names(fdata.Reg.tr)[1], form, sep = " ~ "))
#
# set.seed(150)
# nnmod <- neuralnet(form,
#                     nntrData,
```

```
#                     linear.output = FALSE,
#                     rep = 1,
#                     hidden = hidden_neurons,
#                     lifesign = "minimal",
#                     threshold = 4,
#                     stepmax = iters,
#                     learningrate = decay,
#                     act.fct = "tanh")
#
# # Try SensAnalysisMLP
# NeuralSens::SensAnalysisMLP(nnmod)

## TRAIN nnet NNET --------------------------------------------------------
# Create a formula to train NNET
form <- paste(names(fdata.Reg.tr)[2:ncol(fdata.Reg.tr)], collapse = " + ")
form <- formula(paste(names(fdata.Reg.tr)[1], form, sep = " ~ "))

set.seed(150)
nnetmod <- nnet::nnet(form,
                      data = nntrData,
                      linear.output = TRUE,
                      size = hidden_neurons,
                      decay = decay,
                      maxit = iters)
# Try SensAnalysisMLP
NeuralSens::SensAnalysisMLP(nnetmod, trData = nntrData)
```

SensFeaturePlot — *Feature sensitivity plot*

Description

Show the distribution of the sensitivities of the output in `geom_sina()` plot which color depends on the input values

Usage

```
SensFeaturePlot(object, fdata = NULL, ...)
```

Arguments

`object`	fitted neural network model or `array` containing the raw sensitivities from the function `SensAnalysisMLP`
`fdata`	`data.frame` containing the data to evaluate the sensitivity of the model. Not needed if the raw sensitivities has been passed as `object`
`...`	further arguments that should be passed to `SensAnalysisMLP` function

Value

list of Feature sensitivity plot as described in https://www.r-bloggers.com/a-gentle-introduction-to-shap-values

Examples

```
## Load data -------------------------------------------------------------------
data("DAILY_DEMAND_TR")
fdata <- DAILY_DEMAND_TR

## Parameters of the NNET ------------------------------------------------------
hidden_neurons <- 5
iters <- 250
decay <- 0.1

################################################################################
#########################  REGRESSION NNET #####################################
################################################################################
## Regression dataframe --------------------------------------------------------
# Scale the data
fdata.Reg.tr <- fdata[,2:ncol(fdata)]
fdata.Reg.tr[,3] <- fdata.Reg.tr[,3]/10
fdata.Reg.tr[,1] <- fdata.Reg.tr[,1]/1000

# Normalize the data for some models
preProc <- caret::preProcess(fdata.Reg.tr, method = c("center","scale"))
nntrData <- predict(preProc, fdata.Reg.tr)

#' ## TRAIN nnet NNET --------------------------------------------------------
# Create a formula to train NNET
form <- paste(names(fdata.Reg.tr)[2:ncol(fdata.Reg.tr)], collapse = " + ")
form <- formula(paste(names(fdata.Reg.tr)[1], form, sep = " ~ "))

set.seed(150)
nnetmod <- nnet::nnet(form,
                           data = nntrData,
                           linear.output = TRUE,
                           size = hidden_neurons,
                           decay = decay,
                           maxit = iters)
# Try SensAnalysisMLP
sens <- NeuralSens::SensAnalysisMLP(nnetmod, trData = nntrData, plot = FALSE)
NeuralSens::SensFeaturePlot(sens)
```

SensitivityPlots *Plot sensitivities of a neural network model*

Description

Function to plot the sensitivities created by SensAnalysisMLP.

Usage

```
SensitivityPlots(
  sens = NULL,
  der = TRUE,
  zoom = TRUE,
  quit.legend = FALSE,
  output = 1
)
```

Arguments

sens	SensAnalysisMLP object created by SensAnalysisMLP or HessMLP object created by HessianMLP.
der	logical indicating if density plots should be created. By default is TRUE
zoom	logical indicating if the distributions should be zoomed when there is any of them which is too tiny to be appreciated in the third plot. facet_zoom function from ggforce package is required.
quit.legend	logical indicating if legend of the third plot should be removed. By default is FALSE
output	numeric or character specifying the output neuron or output name to be plotted. By default is the first output (output = 1).

Value

List with the following plot for each output:

- Plot 1: colorful plot with the classification of the classes in a 2D map
- Plot 2: b/w plot with probability of the chosen class in a 2D map
- Plot 3: plot with the stats::predictions of the data provided if param der is FALSE

Examples

```
## Load data -------------------------------------------------------------------
data("DAILY_DEMAND_TR")
fdata <- DAILY_DEMAND_TR

## Parameters of the NNET ------------------------------------------------------
hidden_neurons <- 5
iters <- 250
decay <- 0.1

################################################################################
#########################  REGRESSION NNET #####################################
################################################################################
## Regression dataframe --------------------------------------------------------
# Scale the data
fdata.Reg.tr <- fdata[,2:ncol(fdata)]
fdata.Reg.tr[,3] <- fdata.Reg.tr[,3]/10
fdata.Reg.tr[,1] <- fdata.Reg.tr[,1]/1000
```

```
# Normalize the data for some models
preProc <- caret::preProcess(fdata.Reg.tr, method = c("center","scale"))
nntrData <- predict(preProc, fdata.Reg.tr)

#' ## TRAIN nnet NNET --------------------------------------------------------
# Create a formula to train NNET
form <- paste(names(fdata.Reg.tr)[2:ncol(fdata.Reg.tr)], collapse = " + ")
form <- formula(paste(names(fdata.Reg.tr)[1], form, sep = " ~ "))

set.seed(150)
nnetmod <- nnet::nnet(form,
                           data = nntrData,
                           linear.output = TRUE,
                           size = hidden_neurons,
                           decay = decay,
                           maxit = iters)
# Try SensAnalysisMLP
sens <- NeuralSens::SensAnalysisMLP(nnetmod, trData = nntrData, plot = FALSE)
NeuralSens::SensitivityPlots(sens)
```

SensMatPlot *Plot sensitivities of a neural network model*

Description

Function to plot the sensitivities created by HessianMLP.

Usage

```
SensMatPlot(
  hess,
  sens = NULL,
  output = 1,
  metric = c("mean", "std", "meanSensSQ"),
  senstype = c("matrix", "interactions"),
  ...
)
```

Arguments

hess	HessMLP object created by HessianMLP.
sens	SensMLP object created by SensAnalysisMLP.
output	numeric or character specifying the output neuron or output name to be plotted. By default is the first output (output = 1).
metric	character specifying the metric to be plotted. It can be "mean", "std" or "meanSensSQ".

senstype	character specifying the type of plot to be plotted. It can be "matrix" or "interactions". If type = "matrix", only the second derivatives are plotted. If type = "interactions" the main diagonal are the first derivatives respect each input variable.
...	further argument passed similar to ggcorrplot arguments.

Details

Most of the code of this function is based on ggcorrplot() function from package ggcorrplot. However, due to the inhability of changing the limits of the color scale, it keeps giving a warning if that function is used and the color scale overwritten.

Value

a list of ggplots, one for each output neuron.

Examples

```
## Load data -------------------------------------------------------------------
data("DAILY_DEMAND_TR")
fdata <- DAILY_DEMAND_TR
## Parameters of the NNET ------------------------------------------------------
hidden_neurons <- 5
iters <- 100
decay <- 0.1

################################################################################
#########################  REGRESSION NNET #####################################
################################################################################
## Regression dataframe --------------------------------------------------------
# Scale the data
fdata.Reg.tr <- fdata[,2:ncol(fdata)]
fdata.Reg.tr[,3] <- fdata.Reg.tr[,3]/10
fdata.Reg.tr[,1] <- fdata.Reg.tr[,1]/1000

# Normalize the data for some models
preProc <- caret::preProcess(fdata.Reg.tr, method = c("center","scale"))
nntrData <- predict(preProc, fdata.Reg.tr)

#' ## TRAIN nnet NNET --------------------------------------------------------
# Create a formula to train NNET
form <- paste(names(fdata.Reg.tr)[2:ncol(fdata.Reg.tr)], collapse = " + ")
form <- formula(paste(names(fdata.Reg.tr)[1], form, sep = " ~ "))

set.seed(150)
nnetmod <- nnet::nnet(form,
                      data = nntrData,
                      linear.output = TRUE,
                      size = hidden_neurons,
                      decay = decay,
                      maxit = iters)
# Try HessianMLP
```

```
H <- NeuralSens::HessianMLP(nnetmod, trData = nntrData, plot = FALSE)
NeuralSens::SensMatPlot(H)
S <- NeuralSens::SensAnalysisMLP(nnetmod, trData = nntrData, plot = FALSE)
NeuralSens::SensMatPlot(H, S, senstype = "interactions")
```

SensMLP *Constructor of the SensMLP Class*

Description

Create an object of SensMLP class

Usage

```
SensMLP(
  sens = list(),
  raw_sens = list(),
  mlp_struct = numeric(),
  trData = data.frame(),
  coefnames = character(),
  output_name = character()
)
```

Arguments

sens	`list` of sensitivity measures, one `data.frame` per output neuron
raw_sens	`list` of sensitivities, one `matrix` per output neuron
mlp_struct	`numeric` vector describing the structur of the MLP model
trData	`data.frame` with the data used to calculate the sensitivities
coefnames	`character` vector with the name of the predictor(s)
output_name	`character` vector with the name of the output(s)

Value

SensMLP object

SensTimePlot *Sensitivity analysis plot over time of the data*

Description

Plot of sensitivity of the neural network output respect to the inputs over the time variable from the data provided

Usage

```
SensTimePlot(
  object,
  fdata = NULL,
  date.var = NULL,
  facet = FALSE,
  smooth = FALSE,
  nspline = NULL,
  ...
)
```

Arguments

object	fitted neural network model or `array` containing the raw sensitivities from the function `SensAnalysisMLP`
fdata	`data.frame` containing the data to evaluate the sensitivity of the model. Not needed if the raw sensitivities has been passed as `object`
date.var	`Posixct vector` with the date of each sample of `fdata` If `NULL`, the first variable with Posixct format of `fdata` is used as dates
facet	`logical` if `TRUE`, function `facet_grid` from `ggplot2` is used
smooth	`logical` if `TRUE`, `geom_smooth` plots are added to each variable plot
nspline	`integer` if `smooth` is TRUE, this determine the degree of the spline used to perform `geom_smooth`. If `nspline` is NULL, the square root of the length of the timeseries is used as degrees of the spline.
...	further arguments that should be passed to `SensAnalysisMLP` function

Value

list of `geom_line` plots for the inputs variables representing the sensitivity of each output respect to the inputs over time

Examples

```
## Load data -------------------------------------------------------------------
data("DAILY_DEMAND_TR")
fdata <- DAILY_DEMAND_TR
fdata[,3] <- ifelse(as.data.frame(fdata)[,3] %in% c("SUN","SAT"), 0, 1)
```

```
## Parameters of the NNET ------------------------------------------------------
hidden_neurons <- 5
iters <- 250
decay <- 0.1

################################################################################
#########################  REGRESSION NNET #####################################
################################################################################
## Regression dataframe --------------------------------------------------------
# Scale the data
fdata.Reg.tr <- fdata[,2:ncol(fdata)]
fdata.Reg.tr[,3] <- fdata.Reg.tr[,3]/10
fdata.Reg.tr[,1] <- fdata.Reg.tr[,1]/1000

# Normalize the data for some models
preProc <- caret::preProcess(fdata.Reg.tr, method = c("center","scale"))
nntrData <- predict(preProc, fdata.Reg.tr)

#' ## TRAIN nnet NNET --------------------------------------------------------
# Create a formula to train NNET
form <- paste(names(fdata.Reg.tr)[2:ncol(fdata.Reg.tr)], collapse = " + ")
form <- formula(paste(names(fdata.Reg.tr)[1], form, sep = " ~ "))

set.seed(150)
nnetmod <- nnet::nnet(form,
                      data = nntrData,
                      linear.output = TRUE,
                      size = hidden_neurons,
                      decay = decay,
                      maxit = iters)
# Try SensTimePlot
NeuralSens::SensTimePlot(nnetmod, fdata = nntrData, date.var = NULL)
```

---

simdata *Simulated data to test the package functionalities*

Description

`data.frame` with 2000 rows of 4 columns with 3 input variables `X1,X2,X3` and one output variable `Y`. The data is already scaled, and has been generated using the following code:

`set.seed(150)`

`simdata <-data.frame( "X1" = rnorm(2000,0,1), "X2" = rnorm(2000,0,1), "X3" = rnorm(2000,0,1) )`

`simdata$Y <-simdata$X1^2 + 0.5*simdata$X2 + 0.1*rnorm(2000,0,1)`

Format

A data frame with 2000 rows and 4 variables:

X1 Random input 1

X2 Random input 2

X3 Random input 3

Y Output

Author(s)

Jaime Pizarroso Gonzalo

`summary.HessMLP` *Summary Method for the HessMLP Class*

Description

Print the sensitivity metrics of a `HessMLP` object. This metrics are the mean sensitivity, the standard deviation of sensitivities and the mean of sensitivities square

Usage

```
## S3 method for class 'HessMLP'
summary(object, ...)
```

Arguments

`object`	`HessMLP` object created by `HessianMLP`
`...`	additional parameters

Value

summary object of the `HessMLP` object passed

Examples

```
## Load data -------------------------------------------------------------------
data("DAILY_DEMAND_TR")
fdata <- DAILY_DEMAND_TR

## Parameters of the NNET ------------------------------------------------------
hidden_neurons <- 5
iters <- 250
decay <- 0.1

################################################################################
#########################  REGRESSION NNET #####################################
################################################################################
## Regression dataframe --------------------------------------------------------
# Scale the data
fdata.Reg.tr <- fdata[,2:ncol(fdata)]
```

```
fdata.Reg.tr[,3] <- fdata.Reg.tr[,3]/10
fdata.Reg.tr[,1] <- fdata.Reg.tr[,1]/1000

# Normalize the data for some models
preProc <- caret::preProcess(fdata.Reg.tr, method = c("center","scale"))
nntrData <- predict(preProc, fdata.Reg.tr)

#' ## TRAIN nnet NNET --------------------------------------------------------
# Create a formula to train NNET
form <- paste(names(fdata.Reg.tr)[2:ncol(fdata.Reg.tr)], collapse = " + ")
form <- formula(paste(names(fdata.Reg.tr)[1], form, sep = " ~ "))

set.seed(150)
nnetmod <- nnet::nnet(form,
                           data = nntrData,
                           linear.output = TRUE,
                           size = hidden_neurons,
                           decay = decay,
                           maxit = iters)
# Try HessianMLP
sens <- NeuralSens::HessianMLP(nnetmod, trData = nntrData, plot = FALSE)
summary(sens)
```

`summary.SensMLP` *Summary Method for the SensMLP Class*

Description

Print the sensitivity metrics of a `SensMLP` object. This metrics are the mean sensitivity, the standard deviation of sensitivities and the mean of sensitivities square

Usage

```
## S3 method for class 'SensMLP'
summary(object, ...)
```

Arguments

`object`	`SensMLP` object created by `SensAnalysisMLP`
`...`	additional parameters

Value

summary object of the `SensMLP` object passed

Examples

```
## Load data -------------------------------------------------------------------
data("DAILY_DEMAND_TR")
fdata <- DAILY_DEMAND_TR

## Parameters of the NNET ------------------------------------------------------
hidden_neurons <- 5
iters <- 250
decay <- 0.1

################################################################################
#########################  REGRESSION NNET #####################################
################################################################################
## Regression dataframe --------------------------------------------------------
# Scale the data
fdata.Reg.tr <- fdata[,2:ncol(fdata)]
fdata.Reg.tr[,3] <- fdata.Reg.tr[,3]/10
fdata.Reg.tr[,1] <- fdata.Reg.tr[,1]/1000

# Normalize the data for some models
preProc <- caret::preProcess(fdata.Reg.tr, method = c("center","scale"))
nntrData <- predict(preProc, fdata.Reg.tr)

#' ## TRAIN nnet NNET --------------------------------------------------------
# Create a formula to train NNET
form <- paste(names(fdata.Reg.tr)[2:ncol(fdata.Reg.tr)], collapse = " + ")
form <- formula(paste(names(fdata.Reg.tr)[1], form, sep = " ~ "))

set.seed(150)
nnetmod <- nnet::nnet(form,
                           data = nntrData,
                           linear.output = TRUE,
                           size = hidden_neurons,
                           decay = decay,
                           maxit = iters)
# Try SensAnalysisMLP
sens <- NeuralSens::SensAnalysisMLP(nnetmod, trData = nntrData, plot = FALSE)
summary(sens)
```

`syntheticdata` *List of 4 dataframes to test the functions with different variables types*

Description

List of 4 dataframes to test the functions with different variables types (numeric and character output and inputs)

Format

list of 4 data.frames with 4 columns for 3 inputs and one output:

RegOutNumInp data.frame

- X1 Input 1 of the subset 1 (numeric)
- X2 Input 2 of the subset 1 (numeric)
- X3 Input 3 of the subset 1 (numeric)
- Y Output of the subset 1 (numeric)

ClsOutNumInp data.frame

- X1 Input 1 of the subset 2 (numeric)
- X2 Input 2 of the subset 2 (numeric)
- X3 Input 3 of the subset 2 (numeric)
- Y Output of the subset 2 (character)

ClsOutClsInp data.frame

- X1 Input 1 of the subset 3 (character)
- X2 Input 2 of the subset 3 (numeric)
- X3 Input 3 of the subset 3 (numeric)
- Y Output of the subset 3 (character)

ClsOutClsInp data.frame

- X1 Input 1 of the subset 4 (numeric)
- X2 Input 2 of the subset 4 (character)
- X3 Input 3 of the subset 4 (numeric)
- Y Output of the subset 4 (numeric)

Author(s)

Jose Portela Gonzalez

Index