Package 'NeuralSens'

November 12, 2019

Version 0.1.0

Title Sensitivity Analysis of Neural Networks

Date 2019-10-12

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 [ctb, cre]

Maintainer Jaime Pizarroso Gonzalo <jpizarroso@comillas.edu>

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

Suggests h2o, neural, RSNNS, nnet, neuralnet

RoxygenNote 6.1.1

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 2019-11-11 23:50:17 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	list of data.frames with the sensitivity measures or array with the raw sensitivities calculated with SensAnalysisMLP
comb_type	if object is array, function to combine the third dimension of the array. It can be "mean" or "sqmean". It can also be a function to combine the rows of the array.

Value

sensitivities of the same type as object with the combine sensitivities of all outputs

Examples

```
## Not run:
# mod should be a neural network classification model
sens <- SensAnalysisMLP(mod)
combinesens <- CombineSens(sens)
rawsens <- SensAnalysisMLP(mod, .rawSens = TRUE)
meanCombinerawSens <- CombineSens(rawsens, "mean")
sqmeanCombinerawSens <- CombineSens(rawsens, "sqmean")

## End(Not run)
```

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:

fecha date of the measure

DEM electrical demand

WD day of the week

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:

fecha date of the measure

WD day of the week

TEMP weather temperature

Author(s)

Jose Portela Gonzalez

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)
```

NeuralSens *NeuralSens: Sensitivity Analysis of Neural Networks*

Description

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

SensAnalysisMLP *Sensitivity of NNET 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, ...)

## Default S3 method:
SensAnalysisMLP(MLP.fit, .returnSens = TRUE,
  plot = TRUE, .rawSens = FALSE, trData, actfunc = NULL,
  preProc = NULL, terms = NULL, output_name = NULL, ...)

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

```
## S3 method for class 'H2OMultinomialModel'
SensAnalysisMLP(MLP.fit,
  .returnSens = TRUE, plot = TRUE, .rawSens = FALSE, ...)

## S3 method for class 'H2ORegressionModel'
SensAnalysisMLP(MLP.fit, .returnSens = TRUE,
  plot = TRUE, .rawSens = FALSE, ...)

## S3 method for class 'list'
SensAnalysisMLP(MLP.fit, .returnSens = TRUE,
  plot = TRUE, .rawSens = FALSE, trData, actfunc, ...)

## S3 method for class 'mlp'
SensAnalysisMLP(MLP.fit, .returnSens = TRUE, plot = TRUE,
  .rawSens = FALSE, trData, preProc = NULL, terms = NULL, ...)

## S3 method for class 'nn'
SensAnalysisMLP(MLP.fit, .returnSens = TRUE, plot = TRUE,
  .rawSens = FALSE, preProc = NULL, terms = NULL, ...)

## S3 method for class 'nnet'
SensAnalysisMLP(MLP.fit, .returnSens = TRUE,
  plot = TRUE, .rawSens = FALSE, trData, preProc = NULL,
  terms = NULL, ...)

## S3 method for class 'nnetar'
SensAnalysisMLP(MLP.fit, .returnSens = TRUE,
  plot = TRUE, .rawSens = FALSE, ...)

## S3 method for class 'numeric'
SensAnalysisMLP(MLP.fit, .returnSens = TRUE,
  plot = TRUE, .rawSens = FALSE, trData, preProc = NULL,
  terms = NULL, ...)
```

Arguments

MLP.fit	fitted neural network model
.returnSens	logical value. If TRUE, sensitivity of the model is returned.
plot	logical whether or not to plot the analysis. By default is TRUE.
.rawSens	logical whether or not to return the sensitivity of each row of the data provided, or return the mean, sd and mean of the square of the sensitivities. 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.
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

dataframe with the sensitivities obtained for each variable if .returnSens = TRUE. If .returnSens = FALSE, the sensitivities without processing are returned in a 3D array. If there is more than one output, the sensitivities of each output are given in a list.

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



## 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(RegNNET$caret$finalModel$wts,
                            trData = fdata.Reg.tv,
                            mlpstr = RegNNET$caret$finalModel$n,
                            coefnames = RegNNET$caret$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, ...)
```

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
sensraw <- NeuralSens::SensAnalysisMLP(nnetmod, trData = nntrData, plot = FALSE, .rawSens = TRUE)
NeuralSens::SensFeaturePlot(sensraw, fdata = nntrData)
```

`SensitivityPlots` *Plot sensitivities of a neural network model*

Description

Function to plot the sensitivities created by `SensAnalysisMLP`.

Usage

```
SensitivityPlots(sens = NULL, der = NULL)
```

Arguments

`sens`	`data.frame` with the sensitivities calculated by `SensAnalysisMLP` using `.rawSens = FALSE`.
`der`	`matrix` with the sensitivities calculated by `SensAnalysisMLP` using `.rawSens = TRUE`.

Details

Due to the fact that `sens` is calculated from `dens`, if the latter is passed as argument the argument `sens` is overwritten to maintain coherence between the three plots even. If only `sens` is given, the last plot with the density plots of the inputs is not calculated.

Value

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 if param `dens` is not `NULL`

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)
sensraw <- NeuralSens::SensAnalysisMLP(nnetmod, trData = nntrData, plot = FALSE, .rawSens = TRUE)
NeuralSens::SensitivityPlots(der = sensraw[,,1])
```

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(2000,0.5,0.3), "X2" = rnorm(2000,-1,0.05), "X3" = rnorm(2000,0,0.8) )
simdata$Y <-(simdata$X1^2) -2.5*simdata$X2
simdata <-as.data.frame(scale(simdata))
```

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

`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