Package ‘NeuralSens’

Version 1.0.1
Title Sensitivity Analysis of Neural Networks
Date 2022-06-13
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 <doi:10.18637/jss.v102.i07>.
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,
magrittr
Suggests h2o, RSNNS, nnet, neuralnet, plotly, e1071
RoxygenNote 7.2.0
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 2022-06-21 11:30:08 UTC

R topics documented:

ActFunc ................................................................. 3
AlphaSensAnalysis .............................................. 3
<table>
<thead>
<tr>
<th>R topics documented:</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlphaSensCurve</td>
</tr>
<tr>
<td>CombineSens</td>
</tr>
<tr>
<td>ComputeHessMeasures</td>
</tr>
<tr>
<td>ComputeSensMeasures</td>
</tr>
<tr>
<td>DAILY_DEMAND_TR</td>
</tr>
<tr>
<td>DAILY_DEMAND_TV</td>
</tr>
<tr>
<td>Der2ActFunc</td>
</tr>
<tr>
<td>Der3ActFunc</td>
</tr>
<tr>
<td>DerActFunc</td>
</tr>
<tr>
<td>diag3Darray</td>
</tr>
<tr>
<td>diag3Darray&lt;-</td>
</tr>
<tr>
<td>diag4Darray</td>
</tr>
<tr>
<td>diag4Darray&lt;-</td>
</tr>
<tr>
<td>HessDotPlot</td>
</tr>
<tr>
<td>HessFeaturePlot</td>
</tr>
<tr>
<td>HessianMLP</td>
</tr>
<tr>
<td>HessMLP</td>
</tr>
<tr>
<td>HessToSensMLP</td>
</tr>
<tr>
<td>is.HessMLP</td>
</tr>
<tr>
<td>is.SensMLP</td>
</tr>
<tr>
<td>NeuralSens</td>
</tr>
<tr>
<td>plot.HessMLP</td>
</tr>
<tr>
<td>plot.SensMLP</td>
</tr>
<tr>
<td>PlotSensMLP</td>
</tr>
<tr>
<td>print.HessMLP</td>
</tr>
<tr>
<td>print.SensMLP</td>
</tr>
<tr>
<td>print.summary.HessMLP</td>
</tr>
<tr>
<td>print.summary.SensMLP</td>
</tr>
<tr>
<td>SensAnalysisMLP</td>
</tr>
<tr>
<td>SensDotPlot</td>
</tr>
<tr>
<td>SensFeaturePlot</td>
</tr>
<tr>
<td>SensitivityPlots</td>
</tr>
<tr>
<td>SensMatPlot</td>
</tr>
<tr>
<td>SensMLP</td>
</tr>
<tr>
<td>SensTimePlot</td>
</tr>
<tr>
<td>simdata</td>
</tr>
<tr>
<td>summary.HessMLP</td>
</tr>
<tr>
<td>summary.SensMLP</td>
</tr>
<tr>
<td>syntheticdata</td>
</tr>
</tbody>
</table>

**Index**
**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

**References**


**Examples**

```r
# 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)
```

---

**AlphaSensAnalysis**

**Sensitivity alpha-curve associated to MLP function**

**Description**

Obtain sensitivity alpha-curves associated to MLP function obtained from the sensitivities returned by SensAnalysisMLP.
Usage

\[
\text{AlphaSensAnalysis}\left(\text{sens,} \quad \text{tol} = \text{NULL,} \quad \text{max\_alpha} = 100, \quad \text{interpolate\_alpha} = \text{FALSE,} \quad \text{curve\_equal\_length} = \text{FALSE,} \quad \text{curve\_equal\_origin} = \text{FALSE,} \quad \text{curve\_divided\_max} = \text{FALSE}\right)
\]

Arguments

- **sens**: sensitivity object returned by `SensAnalysisMLP`
- **tol**: difference between M\_alpha and maximum sensitivity of the sensitivity of each input variable
- **max\_alpha**: maximum alpha value to analyze
- **interpolate\_alpha**: interpolate alpha mean if difference of maximum sensitivity and last alpha evaluated is less than tol
- **curve\_equal\_length**: make all the curves of the same length
- **curve\_equal\_origin**: make all the curves begin at (1,0)
- **curve\_divided\_max**: create second plot of curves divided by maximum alpha

Value

alpha-curves of the MLP function

Examples

\[
\begin{align*}
\text{mod} & \leftarrow \text{RSNNS::mlp(simdata[, c("X1", "X2", "X3")], simdata[, "Y"],}
\quad \text{maxit} = 1000, \text{ size} = 15, \text{ linOut} = \text{TRUE}) \\
\text{sens} & \leftarrow \text{SensAnalysisMLP(mod, trData = simdata,}
\quad \text{output\_name} = "Y", \text{ plot} = \text{FALSE}) \\
\text{AlphaSensAnalysis(sens)}
\end{align*}
\]
**AlphaSensCurve**

*Sensitivity alpha-curve associated to MLP function of an input variable*

**Description**

Obtain sensitivity alpha-curve associated to MLP function obtained from the sensitivities returned by SensAnalysisMLP of an input variable.

**Usage**

```
AlphaSensCurve(sens, tol = NULL, max_alpha = 100)
```

**Arguments**

- `sens`: raw sensitivities of the MLP output with respect to input variable.
- `tol`: difference between M_alpha and maximum sensitivity of the sensitivity of each input variable.
- `max_alpha`: maximum alpha value to analyze.

**Value**

alpha-curve of the MLP function

**Examples**

``` r
mod <- RSNNS::mlp(simdata[, c("X1", "X2", "X3")], simdata[, "Y"],
                   maxit = 1000, size = 15, linOut = TRUE)
sens <- SensAnalysisMLP(mod, trData = simdata,
                        output_name = "Y", plot = FALSE)
AlphaSensCurve(sens$raw_sens[[1]][,1])
```

---

**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")
```
ComputeHessMeasures

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

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

#' ## TRAIN nnet NNET --------------------------------------------------------
# Create a formula to train NNET
form <- paste(names(fdata)[1:ncol(fdata)-1], collapse = " + ")
form <- formula(paste(names(fdata)[5], form, sep = " ~ "))

set.seed(150)
mod <- nnet::nnet(form,
data = fdata,
linear.output = TRUE,
size = hidden_neurons,
decay = decay,
maxit = iters)
# mod should be a neural network classification model
sens <- SensAnalysisMLP(mod, trData = fdata, output_name = 'Species')
combinesens <- CombineSens(sens, "sqmean")
```

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**

```r
## Load data -------------------------------------------------------------------
data("DAILY_DEMAND_TR")
ndata <- DAILY_DEMAND_TR

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

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

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

# 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

References


Examples

```r
## 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[,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,
```
DAILY_DEMAND_TR  9

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

References


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
**Der2ActFunc**

**Author(s)**
Jose Portela Gonzalez

**References**


---

| Der2ActFunc | Second derivative of activation function of neuron |

**Description**

Evaluate second derivative of activation function of a neuron

**Usage**

```r
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**

```r
# 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)
```
**Der3ActFunc**

*Third derivative of activation function of neuron*

**Description**

Evaluate third derivative of activation function of a neuron

**Usage**

```r
Der3ActFunc(type = "sigmoid", ...)
```

**Arguments**

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

**Value**

numeric output of the neuron

**Examples**

```r
# Return derivative of the sigmoid activation function of a neuron
ActivationFunction <- Der3ActFunc("sigmoid")
# Return derivative of the tanh activation function of a neuron
ActivationFunction <- Der3ActFunc("tanh")
# Return derivative of the activation function of several layers of neurons
actfuncs <- c("linear","sigmoid","linear")
ActivationFunctions <- sapply(actfuncs, Der3ActFunc)
```

**DerActFunc**

*Derivative of activation function of neuron*

**Description**

Evaluate derivative of activation function of a neuron

**Usage**

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

**Arguments**

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

numeric output of the neuron

References


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).
diag3Darray<-  

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 changed
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**

```r
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
```

---

**diag4Darray**

*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**

```r
diag4Darray(x = 1, dim = length(x))
```

**Arguments**

- **x**: number or vector defining the value of the diagonal of 4D array
- **dim**: integer defining the length of the diagonal. Default is length(x). If length(x) != 1, dim must be equal to length(x).
Details
The diagonal of a 4D array has been defined as those elements in positions c(int,int,int,int), i.e., the four digits are the same.

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

Examples
```r
x <- diag4Darray(c(1,3,6,2), dim = 4)
x
# , , 1, 1
# # [,1] [,2] [,3] [,4]
# [1,] 1 0 0 0
# [2,] 0 0 0 0
# [3,] 0 0 0 0
# [4,] 0 0 0 0
# # , , 2, 1
# # [,1] [,2] [,3] [,4]
# [1,] 0 0 0 0
# [2,] 0 0 0 0
# [3,] 0 0 0 0
# [4,] 0 0 0 0
# # , , 3, 1
# # [,1] [,2] [,3] [,4]
# [1,] 0 0 0 0
# [2,] 0 0 0 0
# [3,] 0 0 0 0
# [4,] 0 0 0 0
# # , , 4, 1
# # [,1] [,2] [,3] [,4]
# [1,] 0 0 0 0
# [2,] 0 0 0 0
# [3,] 0 0 0 0
# [4,] 0 0 0 0
# # , , 1, 2
# # [,1] [,2] [,3] [,4]
# [1,] 0 0 0 0
# [2,] 0 0 0 0
# [3,] 0 0 0 0
# [4,] 0 0 0 0
# #
```
# , , 2, 2
# # [,1] [,2] [,3] [,4]
# [1,] 0 0 0 0
# [2,] 0 3 0 0
# [3,] 0 0 0 0
# [4,] 0 0 0 0
# # , , 3, 2
# # [,1] [,2] [,3] [,4]
# [1,] 0 0 0 0
# [2,] 0 0 0 0
# [3,] 0 0 0 0
# [4,] 0 0 0 0
# # , , 4, 2
# # [,1] [,2] [,3] [,4]
# [1,] 0 0 0 0
# [2,] 0 0 0 0
# [3,] 0 0 0 0
# [4,] 0 0 0 0
# # , , 1, 3
# # [,1] [,2] [,3] [,4]
# [1,] 0 0 0 0
# [2,] 0 0 0 0
# [3,] 0 0 0 0
# [4,] 0 0 0 0
# # , , 2, 3
# # [,1] [,2] [,3] [,4]
# [1,] 0 0 0 0
# [2,] 0 0 0 0
# [3,] 0 0 0 0
# [4,] 0 0 0 0
# # , , 3, 3
# # [,1] [,2] [,3] [,4]
# [1,] 0 0 0 0
# [2,] 0 0 0 0
# [3,] 0 0 6 0
# [4,] 0 0 0 0
# # , , 4, 3
# # [,1] [,2] [,3] [,4]
# [1,] 0 0 0 0
# [2,] 0 0 0 0
# [3,] 0 0 0 0
# [4,] 0 0 0 0
# # , , 1, 4
# # [,1] [,2] [,3] [,4]
# [1,] 0 0 0 0
# [2,] 0 0 0 0
# [3,] 0 0 0 0
# [4,] 0 0 0 0
# # , , 2, 4
# # [,1] [,2] [,3] [,4]
# [1,] 0 0 0 0
# [2,] 0 0 0 0
# [3,] 0 0 0 0
# [4,] 0 0 0 0
# # , , 3, 4
# # [,1] [,2] [,3] [,4]
# [1,] 0 0 0 0
# [2,] 0 0 0 0
# [3,] 0 0 0 0
# [4,] 0 0 0 0
# # , , 4, 4
# # [,1] [,2] [,3] [,4]
# [1,] 0 0 0 0
# [2,] 0 0 0 0
# [3,] 0 0 0 0
# [4,] 0 0 0 0
diag4Darray<-  

```r
# [3,] 0 0 0 0
# [4,] 0 0 0 0
#
# , , 1, 4
#
# [1,] 0 0 0 0
# [2,] 0 0 0 0
# [3,] 0 0 0 0
# [4,] 0 0 0 0
# , , 2, 4
#
# [1,] 0 0 0 0
# [2,] 0 0 0 0
# [3,] 0 0 0 0
# [4,] 0 0 0 0
# , , 3, 4
#
# [1,] 0 0 0 0
# [2,] 0 0 0 0
# [3,] 0 0 0 0
# [4,] 0 0 0 0
# , , 4, 4
#
# [1,] 0 0 0 0
# [2,] 0 0 0 0
# [3,] 0 0 0 0
# [4,] 0 0 0 2
diag4Darray(x)
# 1, 3, 6, 2
```

---

**diag4Darray<-**  

*Define function to change the diagonal of array*

**Description**

Define function to change the diagonal of array

**Usage**

```r
diag4Darray(x) <- value
```
Arguments

- **x**: 3D array whose diagonal must be changed
- **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

```r
x <- array(1, dim = c(4,4,4,4))
diag4Darray(x) <- c(2,2,2,2)
x
# , , 1, 1
# [1,] 2 1 1 1
# [2,] 1 1 1 1
# [3,] 1 1 1 1
# [4,] 1 1 1 1
# , , 2, 1
# [1,] 1 1 1 1
# [2,] 1 1 1 1
# [3,] 1 1 1 1
# [4,] 1 1 1 1
# , , 3, 1
# [1,] 1 1 1 1
# [2,] 1 1 1 1
# [3,] 1 1 1 1
# [4,] 1 1 1 1
# , , 4, 1
# [1,] 1 1 1 1
# [2,] 1 1 1 1
# [3,] 1 1 1 1
# [4,] 1 1 1 1
# , , 1, 2
```
```r
# diag4Darray<- 
#
# [1,] 1 1 1 1
# [2,] 1 1 1 1
# [3,] 1 1 1 1
# [4,] 1 1 1 1
# , , 2, 2
# [1,] 1 1 1 1
# [2,] 1 2 1 1
# [3,] 1 1 1 1
# [4,] 1 1 1 1
# , , 3, 2
# [1,] 1 1 1 1
# [2,] 1 1 1 1
# [3,] 1 1 1 1
# [4,] 1 1 1 1
# , , 4, 2
# [1,] 1 1 1 1
# [2,] 1 1 1 1
# [3,] 1 1 1 1
# [4,] 1 1 1 1
# , , 1, 3
# [1,] 1 1 1 1
# [2,] 1 1 1 1
# [3,] 1 1 1 1
# [4,] 1 1 1 1
# , , 2, 3
# [1,] 1 1 1 1
# [2,] 1 1 1 1
# [3,] 1 1 1 1
# [4,] 1 1 1 1
# , , 3, 3
# [1,] 1 1 1 1
# [2,] 1 1 1 1
# [3,] 1 1 2 1
```
HessDotPlot

Second derivatives 3D scatter or surface plot against input values

Description

3D Plot of second derivatives of the neural network output respect to the inputs. This function uses plotly instead of ggplot2 to achieve better visualization.
Usage

HessDotPlot(
  object,
  fdata = NULL,
  input_vars = "all",
  input_vars2 = "all",
  output_vars = "all",
  surface = FALSE,
  grid = FALSE,
  color = NULL,
  ...)

Arguments

object fitted neural network model or array containing the raw second derivatives from the function HessianMLP
fdata data.frame containing the data to evaluate the second derivatives of the model.
input_vars character vector with the variables to create the scatter plot in x-axis. If "all", then scatter plots are created for all the input variables in fdata.
input_vars2 character vector with the variables to create the scatter plot in y-axis. If "all", then scatter plots are created for all the input variables in fdata.
output_vars character vector with the variables to create the scatter plot. If "all", then scatter plots are created for all the output variables in fdata.
surface logical if TRUE, a 3D surface is created instead of 3D scatter plot (only for combinations of different inputs)
grid logical. If TRUE, plots created are show together using arrangeGrob. It does not work on Windows platforms due to bugs in plotly library.
color character specifying the name of a numeric variable of fdata to color the 3D scatter plot.
... further arguments that should be passed to HessianMLP function

Value

list of 3D geom_point plots for the inputs variables representing the sensitivity of each output respect to the inputs

Examples

## Load data -----------------------------------------------
data("DAILY_DEMAND_TR")
fdata <- DAILY_DEMAND_TR
## Parameters of the NNET ----------------------------------
hidden_neurons <- 5
iters <- 250
decay <- 0.1
# Regression dataframe

## TRAIN nnet NNET

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

preProc <- caret::preProcess(fdata.Reg.tr, method = c("center","scale"))
nntrData <- predict(preProc, fdata.Reg.tr)
```

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

HessFeaturePlot

## Description

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

## Usage

```
HessFeaturePlot(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
HessianMLP

Value

list of Feature sensitivity plot as described in https://www.r-bloggers.com/2019/03/a-gentle-introduction-to-shap-values-in-r/

Examples

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

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

# Scale the data
fdata.Reg.tr <- fdata[,2:ncol(fdata)]
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
hess <- NeuralSens::HessianMLP(nnetmod, trData = nntrData, plot = FALSE)
NeuralSens::HessFeaturePlot(hess)
```

<table>
<thead>
<tr>
<th>HessianMLP</th>
<th>Sensitivity of MLP models</th>
</tr>
</thead>
</table>

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(}
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,
  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,
HessianMLP

## 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 func-
tion.

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

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

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

# Create a formula to train NNET
form <- formula(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)
# 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 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")

## USE DEFAULT METHOD -------------------------------------------------------
NeuralSens::HessianMLP(caretmod$finalModel$wts,
trData = fdata.Reg.tr,
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[,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)

# 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 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**

```r
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 structure 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 \hspace{1cm} \textit{Check if object is of class SensMLP}

**Description**

Check if object is of class SensMLP

**Usage**

\texttt{is.SensMLP(object)}

**Arguments**

object \hspace{0.5cm} \text{SensMLP object}

**Value**

TRUE if object is a SensMLP object

**References**


\textbf{NeuralSens} \hspace{1cm} \textit{NeuralSens: Sensitivity Analysis of Neural Networks}

**Description**

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

\textbf{plot.HessMLP} \hspace{1cm} \textit{Plot method for the HessMLP Class}

**Description**

Plot the sensitivities and sensitivity metrics of a HessMLP object.

**Usage**

```r
## S3 method for class 'HessMLP'
plot(
  x,
  plotType = c("sensitivities", "time", "features", "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 `HessFeaturePlot` function
  - "matrix": use `SensMatPlot` function to show the values of second partial derivatives
  - "interactions": use `SensMatPlot` function to show the values of second partial derivatives and the first partial derivatives in the diagonal

... additional parameters passed to plot function of the NeuralSens package

**Value**

list of graphic objects created by `ggplot`

**Examples**

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

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

### REgression NNET ###

#' #' Regression data frame ---
# Scale the data
fdata.Reg.tr <- fdata[,2:ncol(fdata)]
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)
```
# Try HessianMLP
sens <- NeuralSens::HessianMLP(nnetmod, trData = nntrData, plot = FALSE)
plot(sens)
plot(sens,"time")

---

**plot.SensMLP**

*Plot method for the SensMLP Class*

**Description**

Plot the sensitivities and sensitivity metrics of a SensMLP object.

**Usage**

```r
## 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`

**References**


**Examples**

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

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

# Scale the data
fdata.Reg.tr <- fdata[,2:ncol(fdata)]
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
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)

plot(sens)
plot(sens,"time")
plot(sens,"features")
```

---

**PlotSensMLP**

**Neural network structure sensitivity plot**

**Description**

Plot a neural interpretation diagram colored by sensitivities of the model

**Usage**

```r
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

## Train nnet 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(form,
data = nntrData,
linear.output = TRUE,
size = hidden_neurons,
decay = decay,
maxit = iters)
```

print.HessMLP

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, round_digits = NULL, ...)

Arguments

x 
HessMLP object created by HessianMLP

n 
integer specifying number of sensitivities to print per each output

round_digits 
integer number of decimal places, default NULL

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 dataframe --------------------------------
# Scale the data
fdata.Reg.tr <- fdata[,2:ncol(fdata)]
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 = " ~ "))
```r
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**

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

**Arguments**

- `x` SensMLP object created by `SensAnalysisMLP`
- `n` integer specifying number of sensitivities to print per each output
- `round_digits` integer number of decimal places, default NULL
- `...` additional parameters

**References**


**Examples**

```r
## Load data -----------------------------------------------
data("DAILY_DEMAND_TR")
fd <- DAILY_DEMAND_TR

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

```r
```

```r
```

# Load data -----------------------------------------------
data("DAILY_DEMAND_TR")
fd <- DAILY_DEMAND_TR

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

### Regression dataframe

#### Scale the data
```r
fdata.Reg.tr <- fdata[,2:ncol(fdata)]
fdata.Reg.tr[,1] <- fdata.Reg.tr[,1]/1000
```

#### Normalize the data for some models
```r
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
```r
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
```r
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**

```r
## S3 method for class 'summary.HessMLP'
print(x, round_digits = NULL, ...)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x</code></td>
<td>summary.HessMLP object created by summary method of HessMLP object</td>
</tr>
<tr>
<td><code>round_digits</code></td>
<td>integer number of decimal places, default NULL</td>
</tr>
<tr>
<td><code>...</code></td>
<td>additional parameters</td>
</tr>
</tbody>
</table>
Examples

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

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

# 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 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

```r
## S3 method for class 'summary.SensMLP'
print(x, round_digits = NULL, ...)
```
print.summary.SensMLP

Arguments
x summary.SensMLP object created by summary method of SensMLP object
round_digits integer number of decimal places, default NULL
... additional parameters

References

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

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

# Regression data frame -----------------------------------------------------
# Scale the data
fdata.Reg.tr <- fdata[,2:ncol(fdata)]
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"))
nntData <- predict(preProc, fdata.Reg.tr)

# 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 = nntData,
linear.output = TRUE,
size = hidden_neurons,
decay = decay,
maxit = iters)

# Try SensAnalysisMLP
sens <- NeuralSens::SensAnalysisMLP(nnetmod, trData = nntData, 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(
### SensAnalysisMLP

MLP.fit,

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

```r
## 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.
- **trData**: data.frame containing the data to evaluate the sensitivity of the model
- **actfunc**:
- **preProc**:
- **terms**:
- **...**: additional arguments passed to or from other methods
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

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

# Scale the data
fdata.Reg.tr <- fdata[,2:ncol(fdata)]
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)

# 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)

# 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)

# Create a cluster with 4 available cores
h2o::h2o.init(ip = "localhost",
nthreads = 4)

# Reset the cluster
h2o::h2o.removeAll()

fdata_h2o <- h2o::as.h2o(x = fdata.Reg.tr, destination_frame = "fdata_h2o")
```r
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 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")

## USE DEFAULT METHOD ------------------------------------------------------
NeuralSens::SensAnalysisMLP(caretmod$finalModel$wts,
    trData = fdata.Reg.tr,
    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[,1] <- fdata.Reg.cl[,1]/1000

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

# 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,
```
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)
```

---

**SensDotPlot**

*Sensitivity scatter plot against input values*

**Description**

Plot of sensitivities of the neural network output with respect to the inputs.

**Usage**

```
SensDotPlot(
  object,
  fdata = NULL,
  input_vars = "all",
  output_vars = "all",
  smooth = FALSE,
  nspline = NULL,
  color = NULL,
  grid = FALSE,
  ...
)
```
SensDotPlot

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.
- input_vars: character vector with the variables to create the scatter plot. If "all", then scatter plots are created for all the input variables in fdata.
- output_vars: character vector with the variables to create the scatter plot. If "all", then scatter plots are created for all the output variables in fdata.
- smooth: logical if TRUE, geom_smooth plots are added to each variable plot
- nspline: integer if smooth is TRUE, this determines the degree of the spline used to perform geom_smooth. If nspline is NULL, the square root of the length of the data is used as degrees of the spline.
- color: character specifying the name of a numeric variable of fdata to color the scatter plot.
- grid: logical. If TRUE, plots created are show together using arrangeGrob
- ...: further arguments that should be passed to SensAnalysisMLP function

Value

list of geom_point plots for the input variables representing the sensitivity of each output respect to the inputs

Examples

```r
## Load data -------------------------------------------------------------------
data("DAILY_DEMAND_TR")
fdata <- DAILY_DEMAND_TR
## Parameters of the NNET ------------------------------------------------------
hidden_neurons <- 5
iters <- 250
decay <- 0.1
# Scale the data
fdata.Reg.tr <- fdata[,2:ncol(fdata)]
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)
```

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

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 SensDotPlot
NeuralSens::SensDotPlot(nnetmod, fdata = 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/2019/03/a-gentle-introduction-to-shap-values-in-r/

References

Examples
## Load data -----------------------------------------------
data("DAILY_DEMAND_TR")
fdata <- DAILY_DEMAND_TR
## Parameters of the NNET --------------------------------

## Try SensDotPlot --------------------------------------
NeuralSens::SensDotPlot(nnetmod, fdata = nntrData)
hidden_neurons <- 5
iters <- 250
decay <- 0.1

set.seed(150)
nnetmod <- nnet::nnet(formula(paste(names(fdata.Reg.tr)[1], form, sep = " ~ ")),
data = nntrData,
linear.output = TRUE,
size = hidden_neurons,
decay = decay,
maxit = iters)

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

References


Examples

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

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

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

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

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

References

SensTimePlot

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.
  ...

Value

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

References

Examples

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

# 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:

```r
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)
```
summary.HessMLP

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

References


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

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

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

# Try HessianMLP
sens <- NeuralSens::HessianMLP(nnetmod, trData = nntrData, plot = FALSE)
summary(sens)

### 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

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

object SensMLP object created by \code{SensAnalysisMLP}

... additional parameters

Value

summary object of the SensMLP object passed

References


Examples

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

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

# Scale the data
fdata.Reg.tr <- fdata[,2:ncol(fdata)]
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)

# Regression dataframe ---------------------------------
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 dataframes 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

**References**

Index

* data
  DAILY_DEMAND_TR, 9
  DAILY_DEMAND_TV, 9
  simdata, 61
  syntheticdata, 65

ActFunc, 3
AlphaSensAnalysis, 3
AlphaSensCurve, 5
arrangeGrob, 21, 53

CombineSens, 5
ComputeHessMeasures, 6
ComputeSensMeasures, 7

DAILY_DEMAND_TR, 9
DAILY_DEMAND_TV, 9
Der2ActFunc, 10
Der3ActFunc, 11
DerActFunc, 11
diag3Darray, 12
diag3Darray<-, 13
diag4Darray, 14
diag4Darray<-, 17

facet_zoom, 56
ggplot, 35, 36, 58

HessDotPlot, 20
HessFeaturePlot, 22, 35
HessianMLP, 21, 23, 35, 39, 56, 57, 62
HessMLP, 32
HessToSensMLP, 33

is.HessMLP, 33
is.SensMLP, 34

NeuralSens, 34

plot.HessMLP, 34
plotnet, 38
PlotSensMLP, 37
preProcess, 27, 48
print.HessMLP, 39
print.SensMLP, 40
print.summary.HessMLP, 41
print.summary.SensMLP, 42

SensAnalysisMLP, 3–8, 22, 36, 38, 40, 44, 53–57, 60, 64
SensDotPlot, 52
SensFeaturePlot, 36, 54
SensitivityPlots, 55
SensMatPlot, 35, 57
SensMLP, 59
SensTimePlot, 35, 36, 60
simdata, 61
summary.HessMLP, 62
summary.SensMLP, 63
syntheticdata, 65

train, 28, 48