

# Package ‘NeuralSens’

March 7, 2023

**Version** 1.0.2

**Title** Sensitivity Analysis of Neural Networks

**Date** 2023-03-07

**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](https://doi.org/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](mailto:jpizarroso@comillas.edu)>

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

**Suggests** h2o, RSNNS, nnet, neuralnet, plotly, e1071

**RoxygenNote** 7.2.3

**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** 2023-03-07 12:00:06 UTC

## R topics documented:

ActFunc . . . . .	3
AlphaSensAnalysis . . . . .	3

AlphaSensCurve	5
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
HessDotPlot	20
HessFeaturePlot	22
HessianMLP	23
HessMLP	32
HessToSensMLP	33
is.HessMLP	33
is.SensMLP	34
NeuralSens	34
plot.HessMLP	34
plot.SensMLP	36
PlotSensMLP	37
print.HessMLP	39
print.SensMLP	40
print.summary.HessMLP	41
print.summary.SensMLP	42
SensAnalysisMLP	44
SensDotPlot	52
SensFeaturePlot	54
SensitivityPlots	55
SensMatPlot	57
SensMLP	59
SensTimePlot	60
simdata	61
summary.HessMLP	62
summary.SensMLP	63
syntheticdata	65

---

ActFunc	<i>Activation function of neuron</i>
---------	--------------------------------------

---

**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**

Pizarroso J, Portela J, Muñoz A (2022). NeuralSens: Sensitivity Analysis of Neural Networks. Journal of Statistical Software, 102(7), 1-36.

**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)
```

---

AlphaSensAnalysis	<i>Sensitivity alpha-curve associated to MLP function</i>
-------------------	---

---

**Description**

Obtain sensitivity alpha-curves associated to MLP function obtained from the sensitivities returned by [SensAnalysisMLP](#).

**Usage**

```
AlphaSensAnalysis(
  sens,
  tol = NULL,
  max_alpha = 100,
  interpolate_alpha = FALSE,
  curve_equal_length = FALSE,
  curve_equal_origin = FALSE,
  curve_divided_max = FALSE
)
```

**Arguments**

<code>sens</code>	sensitivity object returned by <a href="#">SensAnalysisMLP</a>
<code>tol</code>	difference between <code>M_alpha</code> and maximum sensitivity of the sensitivity of each input variable
<code>max_alpha</code>	maximum alpha value to analyze
<code>interpolate_alpha</code>	interpolate alpha mean if difference of maximum sensitivity and last alpha evaluated is less than <code>tol</code>
<code>curve_equal_length</code>	make all the curves of the same length
<code>curve_equal_origin</code>	make all the curves begin at (1,0)
<code>curve_divided_max</code>	create second plot of curves divided by maximum alpha

**Value**

alpha-curves of the MLP function

**Examples**

```
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)

AlphaSensAnalysis(sens)
```

---

AlphaSensCurve	<i>Sensitivity alpha-curve associated to MLP function of an input variable</i>
----------------	--

---

**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**

```
mod <- RSNN::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	<i>Sensitivity analysis plot over time of the data</i>
-------------	--

---

**Description**

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

**Usage**

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

**Arguments**

object	SensMLP object generated by <a href="#">SensAnalysisMLP</a> 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**

```
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")
```

---

ComputeHessMeasures *Plot sensitivities of a neural network model*

---

**Description**

Function to plot the sensitivities created by [SensAnalysisMLP](#).

**Usage**

```
ComputeHessMeasures(sens)
```

**Arguments**

sens                    SensAnalysisMLP object created by [SensAnalysisMLP](#).

**Value**

SensAnalysisMLP object with the sensitivities calculated

**Examples**

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

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

#####
##### REGRESSION NNET #####
#####

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

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

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

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

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

**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**

Pizarroso J, Portela J, Muñoz A (2022). NeuralSens: Sensitivity Analysis of Neural Networks. Journal of Statistical Software, 102(7), 1-36.

**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"))
nnetData <- predict(preProc, fdata.Reg.tr)

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

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



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

---

DAILY\_DEMAND\_TR      *Data frame with 4 variables*

---

### Description

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

### Format

A data frame with 1980 rows and 4 variables:

**DATE** date of the measure

**DEM** electrical demand

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

**TEMP** weather temperature

### Author(s)

Jose Portela Gonzalez

### References

Pizarroso J, Portela J, Muñoz A (2022). NeuralSens: Sensitivity Analysis of Neural Networks. Journal of Statistical Software, 102(7), 1-36.

---

DAILY\_DEMAND\_TV      *Data frame with 3 variables*

---

### Description

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

### Format

A data frame with 7 rows and 3 variables:

**DATE** date of the measure

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

**TEMP** weather temperature

**Author(s)**

Jose Portela Gonzalez

**References**

Pizarroso J, Portela J, Muñoz A (2022). NeuralSens: Sensitivity Analysis of Neural Networks. Journal of Statistical Software, 102(7), 1-36.

---

Der2ActFunc	<i>Second derivative of activation function of neuron</i>
-------------	---

---

**Description**

Evaluate second derivative of activation function of a neuron

**Usage**

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

**Arguments**

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

**Value**

numeric output of the neuron

**Examples**

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

---

Der3ActFunc	<i>Third derivative of activation function of neuron</i>
-------------	--

---

**Description**

Evaluate third derivative of activation function of a neuron

**Usage**

```
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**

```
# 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	<i>Derivative of activation function of neuron</i>
------------	--

---

**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

**References**

Pizarroso J, Portela J, Muñoz A (2022). NeuralSens: Sensitivity Analysis of Neural Networks. Journal of Statistical Software, 102(7), 1-36.

**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	<i>Define function to create a 'diagonal' array or get the diagonal of an array</i>
-------------	---

---

**Description**

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

**Usage**

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

**Arguments**

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

**Details**

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

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

**Value**

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

**Examples**

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

---

diag3Darray<-

*Define function to change the diagonal of array*


---

**Description**

Define function to change the diagonal of array

**Usage**

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

**Arguments**

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

**Details**

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

**Value**

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

**Examples**

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

---

diag4Darray	<i>Define function to create a 'diagonal' array or get the diagonal of an array</i>
-------------	---

---

**Description**

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

**Usage**

```
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**

```
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 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

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

**Arguments**

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

**Details**

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

**Value**

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

**Examples**

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

```

#
#      [,1] [,2] [,3] [,4]
# [1,] 1    1    1    1
# [2,] 1    1    1    1
# [3,] 1    1    1    1
# [4,] 1    1    1    1
#
# , , 2, 2
#
#      [,1] [,2] [,3] [,4]
# [1,] 1    1    1    1
# [2,] 1    2    1    1
# [3,] 1    1    1    1
# [4,] 1    1    1    1
#
# , , 3, 2
#
#      [,1] [,2] [,3] [,4]
# [1,] 1    1    1    1
# [2,] 1    1    1    1
# [3,] 1    1    1    1
# [4,] 1    1    1    1
#
# , , 4, 2
#
#      [,1] [,2] [,3] [,4]
# [1,] 1    1    1    1
# [2,] 1    1    1    1
# [3,] 1    1    1    1
# [4,] 1    1    1    1
#
# , , 1, 3
#
#      [,1] [,2] [,3] [,4]
# [1,] 1    1    1    1
# [2,] 1    1    1    1
# [3,] 1    1    1    1
# [4,] 1    1    1    1
#
# , , 2, 3
#
#      [,1] [,2] [,3] [,4]
# [1,] 1    1    1    1
# [2,] 1    1    1    1
# [3,] 1    1    1    1
# [4,] 1    1    1    1
#
# , , 3, 3
#
#      [,1] [,2] [,3] [,4]
# [1,] 1    1    1    1
# [2,] 1    1    1    1
# [3,] 1    1    2    1

```

```

# [4,] 1 1 1 1
#
# , , 4, 3
#
# [1] [2] [3] [4]
# [1,] 1 1 1 1
# [2,] 1 1 1 1
# [3,] 1 1 1 1
# [4,] 1 1 1 1
#
# , , 1, 4
#
# [1] [2] [3] [4]
# [1,] 1 1 1 1
# [2,] 1 1 1 1
# [3,] 1 1 1 1
# [4,] 1 1 1 1
#
# , , 2, 4
#
# [1] [2] [3] [4]
# [1,] 1 1 1 1
# [2,] 1 1 1 1
# [3,] 1 1 1 1
# [4,] 1 1 1 1
#
# , , 3, 4
#
# [1] [2] [3] [4]
# [1,] 1 1 1 1
# [2,] 1 1 1 1
# [3,] 1 1 1 1
# [4,] 1 1 1 1
#
# , , 4, 4
#
# [1] [2] [3] [4]
# [1,] 1 1 1 1
# [2,] 1 1 1 1
# [3,] 1 1 1 1
# [4,] 1 1 1 2

```

---

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 use 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 <a href="#">HessianMLP</a>
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 <a href="#">arrangeGrob</a> . 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 <a href="#">HessianMLP</a> 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 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"))
nnetData <- 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 = nnetData,
                      linear.output = TRUE,
                      size = hidden_neurons,
                      decay = decay,
                      maxit = iters)

# Try HessDotPlot
NeuralSens::HessDotPlot(nnetmod, fdata = nnetData, surface = TRUE, color = "WD")
```

---

HessFeaturePlot

*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

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

## Arguments

<code>object</code>	fitted neural network model or array containing the raw sensitivities from the function <a href="#">SensAnalysisMLP</a>
<code>fdata</code>	<code>data.frame</code> containing the data to evaluate the sensitivity of the model. Not needed if the raw sensitivities has been passed as <code>object</code>
<code>...</code>	further arguments that should be passed to <a href="#">SensAnalysisMLP</a> function

**Value**

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

**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"))
nnetData <- 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 = nnetData,
                      linear.output = TRUE,
                      size = hidden_neurons,
                      decay = decay,
                      maxit = iters)

# Try SensAnalysisMLP
hess <- NeuralSens::HessianMLP(nnetmod, trData = nnetData, plot = FALSE)
NeuralSens::HessFeaturePlot(hess)
```

---

HessianMLP

*Sensitivity of MLP models*


---

**Description**

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

**Usage**

```
HessianMLP(  
  MLP.fit,  
  .returnSens = TRUE,  
  plot = TRUE,  
  .rawSens = FALSE,  
  sens_origin_layer = 1,  
  sens_end_layer = "last",  
  sens_origin_input = TRUE,  
  sens_end_input = FALSE,  
  ...  
)  
  
## Default S3 method:  
HessianMLP(  
  MLP.fit,  
  .returnSens = TRUE,  
  plot = TRUE,  
  .rawSens = FALSE,  
  sens_origin_layer = 1,  
  sens_end_layer = "last",  
  sens_origin_input = TRUE,  
  sens_end_input = FALSE,  
  trData,  
  actfunc = NULL,  
  deractfunc = NULL,  
  der2actfunc = NULL,  
  preProc = NULL,  
  terms = NULL,  
  output_name = NULL,  
  ...  
)  
  
## S3 method for class 'train'  
HessianMLP(  
  MLP.fit,  
  .returnSens = TRUE,  
  plot = TRUE,  
  .rawSens = FALSE,  
  sens_origin_layer = 1,  
  sens_end_layer = "last",  
  sens_origin_input = TRUE,  
  sens_end_input = FALSE,  
  ...  
)  
  
## S3 method for class 'H2OMultinomialModel'  
HessianMLP(  
  MLP.fit,  
  .returnSens = TRUE,  
  plot = TRUE,  
  .rawSens = FALSE,  
  sens_origin_layer = 1,  
  sens_end_layer = "last",  
  sens_origin_input = TRUE,  
  sens_end_input = FALSE,  
  ...  
)
```



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

```
    sens_end_input = FALSE,
    trData,
    preProc = NULL,
    terms = NULL,
    ...
)

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

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

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

```

    ...
  )

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

```

### Arguments

MLP.fit	fitted neural network model
.returnSens	DEPRECATED
plot	logical whether or not to plot the analysis. By default is TRUE.
.rawSens	DEPRECATED
sens_origin_layer	numeric specifies the layer of neurons with respect to which the derivative must be calculated. The input layer is specified by 1 (default).
sens_end_layer	numeric specifies the layer of neurons of which the derivative is calculated. It may also be 'last' to specify the output layer (default).
sens_origin_input	logical specifies if the derivative must be calculated with respect to the inputs (TRUE) or output (FALSE) of the sens_origin_layer layer of the model. By default is TRUE.
sens_end_input	logical specifies if the derivative calculated is of the output (FALSE) or from the input (TRUE) of the sens_end_layer layer of the model. By default is FALSE.
...	additional arguments passed to or from other methods
trData	data.frame containing the data to evaluate the sensitivity of the model
actfunc	character vector indicating the activation function of each neurons layer.
deractfunc	character vector indicating the derivative of the activation function of each neurons layer.
der2actfunc	character vector indicating the second derivative of the activation function of each neurons layer.
preProc	preProcess structure applied to the training data. See also <a href="#">preProcess</a>

terms	function applied to the training data to create factors. See also <a href="#">train</a>
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

### Examples

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

#####
##### REGRESSION NNET #####
#####

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

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

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

set.seed(150)
```

```

nnetmod <- nnet::nnet(form,
                     data = nnetData,
                     linear.output = TRUE,
                     size = hidden_neurons,
                     decay = decay,
                     maxit = iters)

# Try HessianMLP
NeuralSens::HessianMLP(nnetmod, trData = nnetData, plot = FALSE)

# Try HessianMLP to calculate sensitivities with respect to output of hidden neurones
NeuralSens::HessianMLP(nnetmod, trData = nnetData,
                       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 RSNN NNET -----
# Normalize data using RSNN algorithms
trData <- as.data.frame(RSNN::normalizeData(fdata.Reg.tr))
names(trData) <- names(fdata.Reg.tr)
set.seed(150)
RSNNsmod <- RSNN::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[,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"))
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"))
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 = nnetData,
                      linear.output = TRUE,
                      size = hidden_neurons,
                      decay = decay,
                      maxit = iters)

# Try HessianMLP
NeuralSens::HessianMLP(nnetmod, trData = nnetData)

```

---

HessMLP

*Constructor of the HessMLP Class*


---

## Description

Create an object of HessMLP class

## Usage

```

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

```

## Arguments

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

## Value

HessMLP object



---

HessToSensMLP	<i>Convert a HessMLP to a SensMLP object</i>
---------------	--

---

**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	<i>Check if object is of class HessMLP</i>
------------	--

---

**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	<i>Check if object is of class SensMLP</i>
------------	--

---

**Description**

Check if object is of class SensMLP

**Usage**

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

**Arguments**

object            SensMLP object

**Value**

TRUE if object is a SensMLP object

**References**

Pizarroso J, Portela J, Muñoz A (2022). NeuralSens: Sensitivity Analysis of Neural Networks. Journal of Statistical Software, 102(7), 1-36.

---

NeuralSens	<i>NeuralSens: Sensitivity Analysis of Neural Networks</i>
------------	--

---

**Description**

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

---

plot.HessMLP	<i>Plot method for the HessMLP Class</i>
--------------	--

---

**Description**

Plot the sensitivities and sensitivity metrics of a HessMLP object.

**Usage**

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

**Arguments**

x	HessMLP object created by <a href="#">HessianMLP</a>
plotType	character specifying which type of plot should be created. It can be: <ul style="list-style-type: none"> <li>• "sensitivities" (default): use <a href="#">HessianMLP</a> function</li> <li>• "time": use <a href="#">SensTimePlot</a> function</li> <li>• "features": use <a href="#">HessFeaturePlot</a> function</li> <li>• "matrix": use <a href="#">SensMatPlot</a> function to show the values of second partial derivatives</li> <li>• "interactions": use <a href="#">SensMatPlot</a> function to show the values of second partial derivatives and the first partial derivatives in the diagonal</li> </ul>
...	additional parameters passed to plot function of the NeuralSens package

**Value**

list of graphic objects created by [ggplot](#)

**Examples**

```

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

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

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

# Normalize the data for some models
preProc <- caret::preProcess(fdata.Reg.tr, method = c("center","scale"))
nnetData <- 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 = nnetData,
                      linear.output = TRUE,
                      size = hidden_neurons,

```

```

                                decay = decay,
                                maxit = iters)
# Try HessianMLP
sens <- NeuralSens::HessianMLP(nnetmod, trData = nnetData, 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

```

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

```

## Arguments

x	SensMLP object created by <a href="#">SensAnalysisMLP</a>
plotType	character specifying which type of plot should be created. It can be: <ul style="list-style-type: none"> <li>"sensitivities" (default): use <a href="#">SensAnalysisMLP</a> function</li> <li>"time": use <a href="#">SensTimePlot</a> function</li> <li>"features": use <a href="#">SensFeaturePlot</a> function</li> </ul>
...	additional parameters passed to plot function of the NeuralSens package

## Value

list of graphic objects created by [ggplot](#)

## References

Pizarroso J, Portela J, Muñoz A (2022). NeuralSens: Sensitivity Analysis of Neural Networks. Journal of Statistical Software, 102(7), 1-36.

## 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"))
nnetData <- 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 = nnetData,
                      linear.output = TRUE,
                      size = hidden_neurons,
                      decay = decay,
                      maxit = iters)

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

```

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

```



```
NeuralSens::PlotSensMLP(nnetmod, trData = nnetData)
```

---

```
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 <a href="#">HessianMLP</a>
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 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"))
nnetData <- predict(preProc, fdata.Reg.tr)

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

```

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

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

```

---

```
print.SensMLP
```

*Print method for the SensMLP Class*

---

## Description

Print the sensitivities of a SensMLP object.

## Usage

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

## Arguments

<code>x</code>	SensMLP object created by <a href="#">SensAnalysisMLP</a>
<code>n</code>	integer specifying number of sensitivities to print per each output
<code>round_digits</code>	integer number of decimal places, default NULL
<code>...</code>	additional parameters

## References

Pizarroso J, Portela J, Muñoz A (2022). NeuralSens: Sensitivity Analysis of Neural Networks. Journal of Statistical Software, 102(7), 1-36.

## 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"))
ntrData <- 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 = ntrData,
                      linear.output = TRUE,
                      size = hidden_neurons,
                      decay = decay,
                      maxit = iters)

# Try SensAnalysisMLP
sens <- NeuralSens::SensAnalysisMLP(nnetmod, trData = ntrData, plot = FALSE)
sens
```

---

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

---

## Description

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

## Usage

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

## Arguments

x	summary.HessMLP object created by summary method of HessMLP object
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 NNET #####
#####
## Regression dataframe -----
# Scale the data
fdata.Reg.tr <- fdata[,2:ncol(fdata)]
fdata.Reg.tr[,3] <- fdata.Reg.tr[,3]/10
fdata.Reg.tr[,1] <- fdata.Reg.tr[,1]/1000

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

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

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

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

```

---

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

---

**Description**

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

**Usage**

```

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

```

**Arguments**

x summary.SensMLP object created by summary method of SensMLP object  
 round\_digits integer number of decimal places, default NULL  
 ... additional parameters

**References**

Pizarroso J, Portela J, Muñoz A (2022). NeuralSens: Sensitivity Analysis of Neural Networks. Journal of Statistical Software, 102(7), 1-36.

**Examples**

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

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

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

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

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

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

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

---

SensAnalysisMLP      *Sensitivity of MLP models*

---

## Description

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

## Usage

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

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

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

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

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

## S3 method for class 'mlp'
```

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

```

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

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

```

### Arguments

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

actfunc	character vector indicating the activation function of each neurons layer.
deractfunc	character vector indicating the derivative of the activation function of each neurons layer.
preProc	preProcess structure applied to the training data. See also <a href="#">preProcess</a>
terms	function applied to the training data to create factors. See also <a href="#">train</a>
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

Pizarroso J, Portela J, Muñoz A (2022). NeuralSens: Sensitivity Analysis of Neural Networks. Journal of Statistical Software, 102(7), 1-36.

### Examples

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

#####
##### REGRESSION NNET #####
#####

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



```

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

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

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

# Try SensAnalysisMLP
NeuralSens::SensAnalysisMLP(nnetmod, trData = nntrData)

# Try SensAnalysisMLP to calculate sensitivities with respect to output of hidden neurones
NeuralSens::SensAnalysisMLP(nnetmod, trData = nntrData,
                             sens_origin_layer = 2,
                             sens_end_layer = "last",
                             sens_origin_input = FALSE,
                             sens_end_input = FALSE)

## Train caret NNET -----
# Create trainControl
ctrl_tune <- caret::trainControl(method = "boot",
                                 savePredictions = FALSE,
                                 summaryFunction = caret::defaultSummary)

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

# Try SensAnalysisMLP
NeuralSens::SensAnalysisMLP(caretmod)

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

# 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 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[,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"))
nntData <- 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"))
nntData <- 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,

```

```

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

# Try SensAnalysisMLP
NeuralSens::SensAnalysisMLP(nnetmod, trData = nnetData)

```

---

SensDotPlot

*Sensitivity scatter plot against input values*


---

## Description

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

## Usage

```

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

```

**Arguments**

object	fitted neural network model or array containing the raw sensitivities from the function <a href="#">SensAnalysisMLP</a>
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 determine 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 <a href="#">arrangeGrob</a>
...	further arguments that should be passed to <a href="#">SensAnalysisMLP</a> function

**Value**

list of 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 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"))
nnetData <- 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 = nnetData,
                      linear.output = TRUE,
                      size = hidden_neurons,
                      decay = decay,
                      maxit = iters)

# Try SensDotPlot
NeuralSens::SensDotPlot(nnetmod, fdata = nnetData)

```

---

SensFeaturePlot	<i>Feature sensitivity plot</i>
-----------------	---------------------------------

---

### 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 <a href="#">SensAnalysisMLP</a>
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 <a href="#">SensAnalysisMLP</a> function

### Value

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

### References

Pizarroso J, Portela J, Muñoz A (2022). NeuralSens: Sensitivity Analysis of Neural Networks. Journal of Statistical Software, 102(7), 1-36.

### Examples

```

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

## Parameters of the NNET -----

```

```

hidden_neurons <- 5
iters <- 250
decay <- 0.1

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

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

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

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

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

```

---

SensitivityPlots

*Plot sensitivities of a neural network model*


---

## Description

Function to plot the sensitivities created by [SensAnalysisMLP](#).

## Usage

```

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

```

**Arguments**

sens	SensAnalysisMLP object created by <a href="#">SensAnalysisMLP</a> or HessMLP object created by <a href="#">HessianMLP</a> .
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. <a href="#">facet_zoom</a> 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**

Pizarroso J, Portela J, Muñoz A (2022). NeuralSens: Sensitivity Analysis of Neural Networks. Journal of Statistical Software, 102(7), 1-36.

**Examples**

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

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

#####
##### REGRESSION NNET #####
#####

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

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



```

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

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

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

```

SensMatPlot

*Plot sensitivities of a neural network model***Description**

Function to plot the sensitivities created by [HessianMLP](#).

**Usage**

```

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

```

**Arguments**

<code>hess</code>	HessMLP object created by <a href="#">HessianMLP</a> .
<code>sens</code>	SensMLP object created by <a href="#">SensAnalysisMLP</a> .
<code>output</code>	numeric or character specifying the output neuron or output name to be plotted. By default is the first output ( <code>output = 1</code> ).
<code>metric</code>	character specifying the metric to be plotted. It can be "mean", "std" or "meanSensSQ".
<code>senstype</code>	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.
<code>...</code>	further argument passed similar to <code>ggcorrplot</code> 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

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

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

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

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

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

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

---

SensMLP

*Constructor of the SensMLP Class*

---

## Description

Create an object of SensMLP class

## Usage

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

## Arguments

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

## Value

SensMLP object

## References

Pizarroso J, Portela J, Muñoz A (2022). NeuralSens: Sensitivity Analysis of Neural Networks. *Journal of Statistical Software*, 102(7), 1-36.

---

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 <a href="#">SensAnalysisMLP</a>
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 <code>facet_grid</code> from <code>ggplot2</code> is used
smooth	logical if TRUE, <code>geom_smooth</code> plots are added to each variable plot
nspline	integer if smooth is TRUE, this determine the degree of the spline used to perform <code>geom_smooth</code> . 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 <a href="#">SensAnalysisMLP</a> function

### Value

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

### References

Pizarroso J, Portela J, Muñoz A (2022). NeuralSens: Sensitivity Analysis of Neural Networks. *Journal of Statistical Software*, 102(7), 1-36.

**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"))
nnetData <- 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 = nnetData,
                      linear.output = TRUE,
                      size = hidden_neurons,
                      decay = decay,
                      maxit = iters)

# Try SensTimePlot
NeuralSens::SensTimePlot(nnetmod, fdata = nnetData, date.var = NULL)

```

simdata

*Simulated data to test the package functionalities***Description**

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

```

set.seed(150)
simdata <- data.frame("X1" = rnorm(2000, 0, 1), "X2" = rnorm(2000, 0, 1), "X3" = rnorm(2000,
0, 1))
simdata$Y <- simdata$X1^2 + 0.5*simdata$X2 + 0.1*rnorm(2000, 0, 1)

```

**Format**

A data frame with 2000 rows and 4 variables:

**X1** Random input 1

**X2** Random input 2

**X3** Random input 3

**Y** Output

**Author(s)**

Jaime Pizarroso Gonzalo

**References**

Pizarroso J, Portela J, Muñoz A (2022). NeuralSens: Sensitivity Analysis of Neural Networks. Journal of Statistical Software, 102(7), 1-36.

---

summary.HessMLP

*Summary Method for the HessMLP Class*

---

**Description**

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

**Usage**

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

**Arguments**

object            HessMLP object created by [HessianMLP](#)  
...                additional parameters

**Value**

summary object of the HessMLP object passed

**Examples**

```

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

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

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

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

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

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

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

```

summary.SensMLP

*Summary Method for the SensMLP Class***Description**

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

**Usage**

```

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

```

**Arguments**

object            SensMLP object created by [SensAnalysisMLP](#)  
 ...                additional parameters

**Value**

summary object of the SensMLP object passed

**References**

Pizarroso J, Portela J, Muñoz A (2022). NeuralSens: Sensitivity Analysis of Neural Networks. Journal of Statistical Software, 102(7), 1-36.

**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"))
nnetData <- 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 = nnetData,
                      linear.output = TRUE,
                      size = hidden_neurons,
                      decay = decay,
                      maxit = iters)

# Try SensAnalysisMLP
sens <- NeuralSens::SensAnalysisMLP(nnetmod, trData = nnetData, plot = FALSE)
summary(sens)
```



---

syntheticdata

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

---

**Description**

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

**Format**

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

**RegOutNumInp** data.frame

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

**ClsOutNumInp** data.frame

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

**ClsOutClsInp** data.frame

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

**ClsOutClsInp** data.frame

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

**Author(s)**

Jose Portela Gonzalez

**References**

Pizarroso J, Portela J, Muñoz A (2022). NeuralSens: Sensitivity Analysis of Neural Networks. Journal of Statistical Software, 102(7), 1-36.

# 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
- plot.SensMLP, 36
- 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