Package 'NeuralSens'

September 13, 2019

Version 0.0.7

Title Sensitivity Analysis of Neural Networks

Date 2019-09-13

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.

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

Maintainer Jaime Pizarroso Gonzalo <jpizarroso@alu.comillas.edu>

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

Suggests h2o, neural, RSNNS, nnet, neuralnet

RoxygenNote 6.1.1

NeedsCompilation no

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

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

License GPL (>= 2)

Encoding UTF-8

LazyData true

Repository CRAN

Date/Publication 2019-09-13 20:00:03 UTC

R topics documented:

`ActFunc` *Activation function of neuron*

Description

Evaluate activation function of a neuron

Usage

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

Arguments

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

Value

`numeric` output of the neuron

Examples

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

`DAILY_DEMAND_TR` *Data frame with 4 variables*

Description

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

Format

A data frame with 1980 rows and 4 variables:

fecha date of the measure

DEM electrical demand

WD working day coefficient

TEMP weather temperature

Author(s)

Jose Portela Gonzalez

DAILY_DEMAND_TV *Data frame with 3 variables*

Description

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

Format

A data frame with 7 rows and 3 variables:

fecha date of the measure

WD working day coefficient

TEMP weather temperature

Author(s)

Jose Portela Gonzalez

DerActFunc *Derivate activation function of neuron*

Description

Evaluate derivate of activation function of a neuron

Usage

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

Arguments

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

Value

numeric output of the neuron

Examples

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

NeuralSens *NeuralSens: Sensibility analysis to NeuralNets*

Description

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

SensAnalysisMLP *Sensitivity of NNET models*

Description

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

Usage

```
SensAnalysisMLP(MLP.fit, .returnSens = TRUE, plot = TRUE,
  .rawSens = FALSE, ...)

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

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

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

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

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

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

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

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

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

Arguments

MLP.fit	fitted model from caret package using nnet method
.returnSens	logical value. If TRUE, sensibility of the model is returned.
plot	logical whether or not to plot the analysis. By default is TRUE.
.rawSens	logical whether or not to return the sensitivity of each row of the data provided, or return the mean, sd and mean of the square of the sensitivities. By default is FALSE.
...	additional arguments passed to or from other methods
trData	data frame containing the training data of the model
actfunc	character vector indicating the activation function of each neurons layer.
preProc	preProcess structure applied to the training data. See also preProcess
terms	function applied to the training data to create factors. See also train

Details

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

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

Value

dataframe with the sensitivities obtained for each variable if .returnSens `TRUE`. If there is more than one output, the sensitivities of each output are given in a list.

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

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

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

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

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

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

# Try SensAnalysisMLP
NeuralSens::SensAnalysisMLP(caretmod)

## Train h2o NNET --------------------------------------------------------------
# Creaci?n de un cluster local con todos los cores disponibles
h2o::h2o.init(ip = "localhost",
              # -1 indica que se empleen todos los cores disponibles.
              nthreads = 4,
              # M?xima memoria disponible para el cluster.
              max_mem_size = "2g")

# Se eliminan los datos del cluster por si ya hab?a sido iniciado.
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)
```

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

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

# Try SensAnalysisMLP
trData <- nntrData
names(trData)[1] <- ".outcome"
NeuralSens::SensAnalysisMLP(neuralmod, trData = trData)

## 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)
names(trData)[1] <- ".outcome"

# Try SensAnalysisMLP
NeuralSens::SensAnalysisMLP(RSNNSmod, trData = trData)

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

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

# Try SensAnalysisMLP
NeuralSens::SensAnalysisMLP(nnmod)
```

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

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

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

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

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

# Try SensAnalysisMLP
NeuralSens::SensAnalysisMLP(caretmod)

## Train h2o NNET -----------------------------------------------------------
# Creaci?n de un cluster local con todos los cores disponibles
h2o::h2o.init(ip = "localhost",
              # -1 indica que se empleen todos los cores disponibles.
              nthreads = 4,
              # M?xima memoria disponible para el cluster.
              max_mem_size = "2g")

# Se eliminan los datos del cluster por si ya hab?a sido iniciado.
h2o::h2o.removeAll()
fdata_h2o <- h2o::as.h2o(x = fdata.Reg.cl, destination_frame = "fdata_h2o")
```

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

# Try SensAnalysisMLP
NeuralSens::SensAnalysisMLP(h2omod)

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

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

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

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

```
# form <- formula(paste(names(fdata.Reg.tr)[1], form, sep = " ~ "))
#
# set.seed(150)
# nnmod <- neuralnet(form,
#                    nntrData,
#                    linear.output = FALSE,
#                    rep = 1,
#                    hidden = hidden_neurons,
#                    lifesign = "minimal",
#                    threshold = 4,
#                    stepmax = iters,
#                    learningrate = decay,
#                    act.fct = "tanh")
#
# # Try SensAnalysisMLP
# NeuralSens::SensAnalysisMLP(nnmod)

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

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

Index