Package ‘nnetpredint’

December 21, 2015

Type Package

Title Prediction Intervals of Multi-Layer Neural Networks

Version 1.2

Date 2015-12-21

Suggests MASS, nnet, neuralnet

Imports stats, RSNNS

Author Xichen Ding <rockingdingo@gmail.com>

Maintainer Xichen Ding <rockingdingo@gmail.com>

Description Computing prediction intervals of neural network models (e.g.backpropagation) at certain confidence level. It can take the output from models trained by other packages like 'nnet', 'neuralnet', 'RSNNS', etc.

License GPL (>= 2)

Repository CRAN

Repository/R-Forge/Project nnetpredint

Repository/R-Forge/Revision 7

Repository/R-Forge/DateTimeStamp 2015-12-21 12:31:51

Date/Publication 2015-12-21 21:35:34

NeedsCompilation no

R topics documented:

`activate` *Neuron Activation Function*

Description

activation function for each neuron node, we provide the popular activation functions including 'sigmoid', 'tanh', etc.

Usage

```
activate(x, funName)
```

Arguments

`x`	input value to the activation function
`funName`	This package provides the most popular activation functions which can be used by setting the funName parameter to the following strings: 'sigmoid', 'tanh'. 'sigmoid' sigmoid function 1/(1 + exp(-x)). 'tanh' tanh is the hyperbolic tangent function equal to (exp(x) - exp(-x)) / (exp(x) + exp(-x)).

Examples

```
x <- c(-5:5)
y1 <- activate(x, funName = 'sigmoid')
y2 <- activate(x, funName = 'tanh')
```

`jacobian` *Jacobian Matrix of Gradient Function for Training Datasets*

Description

Calculate the Jacobian matrix of gradient function for the training dataset. It takes input from neural network models and the gradient at each weight parameters. The matrix has dimension of R [nObs * nPara], nObs denotes the number of training observations and nPara denotes the number of weights parameters.

Usage

```
jacobian(object, ...)

## S3 method for class 'nnet'
jacobian(object, xTrain, funName = 'sigmoid',...)
## S3 method for class 'nn'
jacobian(object, xTrain, funName = 'sigmoid',...)
## S3 method for class 'rsnns'
jacobian(object, xTrain, funName = 'sigmoid',...)
```

Arguments

`object`	object of class: nnet as returned by 'nnet' package, nn as returned by 'neuralnet' package, rsnns as returned by 'RSNNS' package.
`xTrain`	matrix or data frame of input values for the training dataset.
`funName`	activation function name of neuron, e.g. 'sigmoid', 'tanh', etc. In default, it is set to 'sigmoid'.
`...`	additional arguments passed to the method.

Details

Jacobian matrix with gradient function, in which J[ij] element denotes the gradient function at the jth weight parameters for the ith training observation. The dimension is equal to nObs * nPara.

Value

matrix which denotes the Jacobian matrix for training datasets.

Author(s)

Xichen Ding <rockingdingo@gmail.com>

See Also

`nnetPredInt`

Examples

```
library(nnet)
xTrain <- rbind(cbind(runif(150,min = 0, max = 0.5),runif(150,min = 0, max = 0.5)) ,
cbind(runif(150,min = 0.5, max = 1),runif(150,min = 0.5, max = 1))
)
nObs <- dim(xTrain)[1]
yTrain <- 0.5 + 0.4 * sin(2* pi * xTrain %*% c(0.4,0.6)) +rnorm(nObs,mean = 0, sd = 0.05)
# Training nnet models
net <- nnet(yTrain ~ xTrain,size = 3, rang = 0.1,decay = 5e-4, maxit = 500)

# Calculating Jacobian Matrix of the training samples
library(nnetpredint)
jacobMat = jacobian(net,xTrain)
dim(jacobMat)
```

`nnetPredInt` *Prediction Intervals of Neural Networks*

Description

Get the prediction intervals of new dataset at certain confidence level based on the training datasets and the gradient at weight parameters of the neural network model.

Usage

```
nnetPredInt(object, ...)

## Default S3 method:
nnetPredInt(object = NULL, xTrain, yTrain, yFit, node, wts, newData,
    alpha = 0.05 , lambda = 0.5, funName = 'sigmoid', ...)
## S3 method for class 'nnet'
nnetPredInt(object, xTrain, yTrain, newData, alpha = 0.05, lambda = 0.5,
    funName = 'sigmoid', ...)
## S3 method for class 'nn'
nnetPredInt(object, xTrain, yTrain, newData, alpha = 0.05, lambda = 0.5,
    funName = 'sigmoid', ...)
## S3 method for class 'rsnns'
nnetPredInt(object, xTrain, yTrain, newData, alpha = 0.05, lambda = 0.5,
    funName = 'sigmoid', ...)
```

Arguments

`object`	object of class: nnet as returned by 'nnet' package, nn as returned by 'neuralnet' package, rsnns as returned by 'RSNNS' package. Object set as NULL will use the default method which takes the weight parameters as the input from user.
`xTrain`	matrix or data frame of input values for the training dataset.
`yTrain`	vector of target values for the training dataset.
`newData`	matrix or data frame of the prediction dataset.
`yFit`	vector of the fitted values, as the output produced by the training model, e.g. nnet$fitted.values ('nnet') , nn$net.result[[1]] ('neuralnet') and rsnns$fitted.values ('RSNNS')
`node`	a vector of integers specifying the number of hidden nodes in each layer. Multi-layer network has the structure (s0, s1, ..., sm), in which s0 denotes the dimension for input layer and sm denotes the dimension of the output layer. sm is usually set as 1.
`wts`	a numeric vector of optimal weight parameters as the output of the neural network training model. The order of wts parameter is as follows: For any node i in layer k: c(bias ik, wi1k,wi2k,...wijk). nnet object, returned by 'nnet' package. We can directly set the wts as: wts = nnet$wts

	nn object, returned by 'neuralnet' package. We need to use transWeightListToVect function to transform the list of weights to a single vector first: wts = transWeightListToVect(wtsList, m). rsnns object, returned by 'RSNNS' package. We need to transform and combine the weight and bias parameters to a single vector: weightMatrix(object) and extractNetInfo(object)$unitDefinitions$unitBias.
`alpha`	confidence level. The confidence level is set to (1-alpha). In default, alpha = 0.05.
`lambda`	decay parameter of weights when the Jacobian matrix of training dataset is singular. In default, lamda is set to 0.5 .
`funName`	activation function name of neuron, e.g. 'sigmoid', 'tanh', etc. In default, it is set to 'sigmoid'.
`...`	additional arguments passed to the method.

Value

data frame of the prediction intervals, including prediction value, lower and upper bounds of the interval.

`yPredValue`	the column of prediction value in the data frame.
`lowerBound`	the column of prediction lower bounds in the data frame.
`upperBound`	the column of prediction upper bounds in the data frame.

Author(s)

Xichen Ding <rockingdingo@gmail.com>

References

De Veaux R. D., Schumi J., Schweinsberg J., Ungar L. H., 1998, "Prediction intervals for neural networks via nonlinear regression", Technometrics 40(4): 273-282.

Chryssolouris G., Lee M., Ramsey A., "Confidence interval prediction for neural networks models", IEEE Trans. Neural Networks, 7 (1), 1996, pp. 229-232.

'neuralnet' package by Stefan Fritsch, Frauke Guenther.

'nnet' package by Brian Ripley, William Venables.

'RSNNS' package by Christoph Bergmeir, Jose M. Benitez.

See Also

`transWeightListToVect` `jacobian`

Examples

```
# Example 1: Using the nn object trained by neuralnet package
set.seed(500)
library(MASS)
data <- Boston
maxs <- apply(data, 2, max)
mins <- apply(data, 2, min)
scaled <- as.data.frame(scale(data, center = mins, scale = maxs - mins)) # normalization
index <- sample(1:nrow(data),round(0.75*nrow(data)))
train_ <- scaled[index,]
test_ <- scaled[-index,]

library(neuralnet) # Training
n <- names(train_)
f <- as.formula(paste("medv ~", paste(n[!n %in% "medv"], collapse = " + ")))
nn <- neuralnet(f,data = train_,hidden = c(5,3),linear.output = FALSE)
plot(nn)

library(nnetpredint) # Getting prediction confidence interval
x <- train_[,-14]
y <- train_[,14]
newData <- test_[,-14]

# S3 generic method: Object of nn
yPredInt <- nnetPredInt(nn, x, y, newData)
print(yPredInt[1:20,])

# S3 default method for user defined weights input, without model object trained:
yFit <- c(nn$net.result[[1]])
nodeNum <- c(13,5,3,1)
m <- 3
wtsList <- nn$weights[[1]]
wts <- transWeightListToVect(wtsList,m)
yPredInt2 <- nnetPredInt(object = NULL, x, y, yFit, nodeNum, wts, newData, alpha = 0.05)
print(yPredInt2[1:20,])

# Compare to the predict values from the neuralnet Compute method
predValue <- compute(nn,newData)
print(matrix(predValue$net.result[1:20]))



# Example 2: Using the nnet object trained by nnet package
library(nnet)
xTrain <- rbind(cbind(runif(150,min = 0, max = 0.5),runif(150,min = 0, max = 0.5)) ,
cbind(runif(150,min = 0.5, max = 1),runif(150,min = 0.5, max = 1))
)
nObs <- dim(xTrain)[1]
yTrain <- 0.5 + 0.4 * sin(2* pi * xTrain %*% c(0.4,0.6)) +rnorm(nObs,mean = 0, sd = 0.05)
plot(xTrain %*% c(0.4,0.6),yTrain)

# Training nnet models
```

```
net <- nnet(yTrain ~ xTrain,size = 3, rang = 0.1,decay = 5e-4, maxit = 500)
yFit <- c(net$fitted.values)
nodeNum <- c(2,3,1)
wts <- net$wts

# New data for prediction intervals
library(nnetpredint)
newData <- cbind(seq(0,1,0.05),seq(0,1,0.05))
yTest <- 0.5 + 0.4 * sin(2* pi * newData %*% c(0.4,0.6))+rnorm(dim(newData)[1],
    mean = 0, sd = 0.05)

# S3 generic method: Object of nnet
yPredInt <- nnetPredInt(net, xTrain, yTrain, newData)
print(yPredInt[1:20,])

# S3 default method: xTrain,yTrain,yFit,...
yPredInt2 <- nnetPredInt(object = NULL, xTrain, yTrain, yFit, node = nodeNum, wts = wts,
    newData, alpha = 0.05, funName = 'sigmoid')

plot(newData %*% c(0.4,0.6),yTest,type = 'b')
lines(newData %*% c(0.4,0.6),yPredInt$yPredValue,type = 'b',col='blue')
lines(newData %*% c(0.4,0.6),yPredInt$lowerBound,type = 'b',col='red')   # lower bound
lines(newData %*% c(0.4,0.6),yPredInt$upperBound,type = 'b',col='red')   # upper bound



# Example 3: Using the rsnns object trained by RSNNS package
library(RSNNS)
data(iris)
#shuffle the vector
iris <- iris[sample(1:nrow(iris),length(1:nrow(iris))),1:ncol(iris)]
irisValues <- iris[,1:4]
irisTargets <- decodeClassLabels(iris[,5])[,'setosa']

iris <- splitForTrainingAndTest(irisValues, irisTargets, ratio=0.15)
iris <- normTrainingAndTestSet(iris)
model <- mlp(iris$inputsTrain, iris$targetsTrain, size=5, learnFuncParams=c(0.1),
maxit=50, inputsTest=iris$inputsTest, targetsTest=iris$targetsTest)
predictions <- predict(model,iris$inputsTest)

# Generating prediction intervals
library(nnetpredint)
# S3 Method for rsnns class prediction intervals
xTrain <- iris$inputsTrain
yTrain <- iris$targetsTrain
newData <- iris$inputsTest
yPredInt <- nnetPredInt(model, xTrain, yTrain, newData)
print(yPredInt[1:20,])
```

transWeightListToVect *Transform the List of Optimal Weights to a Numeric Vector*

Description

This function transforms the list of weight parameters,typically the output of neuralnet package (nn$weights) to a single numeric vector in specific order, which will be used as the input to the prediciton interval function. The order of weight parameters is the same as the output of nnet package (nnet$wts).

Usage

```
transWeightListToVect(wtsList,m = 2)
```

Arguments

`wtsList`	the list of weights found by neuralnet package (nn$weights). For nnet weights output(nnet$wts), there is no need for transformation and it can be used directly by the prediction interval method.
`m`	the number of layers of the neural networks, which is the number of hidden layer plus + 1. m is default to 2 for single hidden layer networks.

Value

the numeric vector of optimal weights found by neural networks, dimension 1 * nPara.

Author(s)

Xichen Ding <rockingdingo@gmail.com>

References

neuralnet package by Stefan Fritsch, Frauke Guenther

See Also

`nnetPredInt`

Index