Package ‘elmNNRcpp’

May 4, 2021

Type Package

Title The Extreme Learning Machine Algorithm

Version 1.0.3

Date 2021-05-03

BugReports https://github.com/mlampros/elmNNRcpp/issues

URL https://github.com/mlampros/elmNNRcpp

Description
Training and predict functions for Single Hidden-layer Feedforward Neural Networks (SLFN) using the Extreme Learning Machine (ELM) algorithm. The ELM algorithm differs from the traditional gradient-based algorithms for very short training times (it doesn't need any iterative tuning, this makes learning time very fast) and there is no need to set any other parameters like learning rate, momentum, epochs, etc. This is a reimplementation of the 'elmNN' package using 'RcppArmadillo' after the 'elmNN' package was archived. For more information, see "Extreme learning machine: Theory and applications" by Guang-Bin Huang, Qin-Yu Zhu, Chee-Kheong Siew (2006), Elsevier B.V, <doi:10.1016/j.neucom.2005.12.126>.

License GPL (>= 2)

Encoding UTF-8

Depends R(>= 3.0.2), KernelKnn

Imports Rcpp (>= 0.12.17)

LinkingTo Rcpp, RcppArmadillo (>= 0.8)

Suggests testthat, covr, knitr, rmarkdown

VignetteBuilder knitr

RoxygenNote 7.1.1

NeedsCompilation yes

Author Lampros Mouselimis [aut, cre] (<https://orcid.org/0000-0002-8024-1546>), Alberto Gosso [aut]

Maintainer Lampros Mouselimis <mouselimislampros@gmail.com>

Repository CRAN

Date/Publication 2021-05-04 05:30:06 UTC
R topics documented:

elm_predict ........................................ 2
elm_train ........................................... 3
onehot_encode ........................................ 5

Index

elm_predict Extreme Learning Machine predict function

Description

Extreme Learning Machine predict function

Usage

elm_predict(elm_train_object, newdata, normalize = FALSE)

Arguments

elm_train_object

it should be the output of the elm_train function

newdata

an input matrix with number of columns equal to the x parameter of the elm_train function

normalize

a boolean specifying if the output predictions in case of classification should be normalized. If TRUE then the values of each row of the output-probability-matrix that are less than 0 and greater than 1 will be pushed to the [0,1] range

Examples

library(elmNNRcpp)

#---------
# Regression
#---------
data(Boston, package = 'KernelKnn')

Boston = as.matrix(Boston)
dimnames(Boston) = NULL

x = Boston[, -ncol(Boston)]
y = matrix(Boston[, ncol(Boston)], nrow = length(Boston[, ncol(Boston)]), ncol = 1)

out_regr = elm_train(x, y, nhid = 20, actfun = 'purelin', init_weights = 'uniform_negative')

pr_regr = elm_predict(out_regr, x)
elm_train

#------------------
# Classification
#------------------

data(ionosphere, package = 'KernelKnn')

x_class = ionosphere[, -c(2, ncol(ionosphere))]
x_class = as.matrix(x_class)
dimnames(x_class) = NULL

y_class = as.numeric(ionosphere[, ncol(ionosphere)])

y_class_onehot = onehot_encode(y_class - 1)  # class labels should begin from 0

out_class = elm_train(x_class, y_class_onehot, nhid = 20, actfun = 'relu')

pr_class = elm_predict(out_class, x_class, normalize = TRUE)

---

elm_train

**Extreme Learning Machine training function**

**Description**

Extreme Learning Machine training function

**Usage**

```r
elm_train(
  x,
  y,
  nhid,
  actfun,
  init_weights = "normal_gaussian",
  bias = FALSE,
  moorep_pseudoinv_tol = 0.01,
  leaky relu alpha = 0,
  seed = 1,
  verbose = FALSE
)
```

**Arguments**

- `x` a matrix. The columns of the input matrix should be of type numeric
- `y` a matrix. In case of regression the matrix should have `n` rows and 1 column. In case of classification it should consist of `n` rows and `n` columns, where `n > l` and equals to the number of the unique labels.
elm_train

**nhid**  
a numeric value specifying the hidden neurons. Must be >= 1

**actfun**  
a character string specifying the type of activation function. It should be one of the following: 'sig' (sigmoid), 'sin' (sine), 'radbas' (radial basis), 'hardlim' (hard-limit), 'hardlims' (symmetric hard-limit), 'satlins' (satlins), 'tansig' (tan-sigmoid), 'tribas' (triangular basis), 'relu' (rectifier linear unit) or 'purelin' (linear)

**init_weights**  
a character string specifying the distribution from which the input-weights and the bias should be initialized. It should be one of the following: 'normal_gaussian' (normal/Gaussian distribution with zero mean and unit variance), 'uniform_positive' (in the range [0,1]) or 'uniform_negative' (in the range [-1,1])

**bias**  
either TRUE or FALSE. If TRUE then bias weights will be added to the hidden layer

**moorep_pseudoinv_tol**  
a numeric value. See the references web-link for more details on Moore-Penrose pseudo-inverse and specifically on the pseudo inverse tolerance value

**leaky_relu_alpha**  
a numeric value between 0.0 and 1.0. If 0.0 then a simple relu (f(x) = 0.0 for x < 0, f(x) = x for x >= 0) activation function will be used, otherwise a leaky-relu (f(x) = alpha * x for x < 0, f(x) = x for x >= 0). It is applicable only if actfun equals to 'relu'

**seed**  
a numeric value specifying the random seed. Defaults to 1

**verbose**  
a boolean. If TRUE then information will be printed in the console

**Details**

The input matrix should be of type numeric. This means the user should convert any character, factor or boolean columns to numeric values before using the `elm_train` function

**References**

http://arma.sourceforge.net/docs.html  
https://en.wikipedia.org/wiki/Moore  
https://www.kaggle.com/robertbm/extreme-learning-machine-example  
http://rt.dgyblog.com/ml/ml-elm.html

**Examples**

```r
library(elmNNRcpp)

# Regression

data(Boston, package = 'KernelKnn')
```
Boston = as.matrix(Boston)
dimnames(Boston) = NULL

x = Boston[, -ncol(Boston)]
y = matrix(Boston[, ncol(Boston)], nrow = length(Boston[, ncol(Boston)]), ncol = 1)

out_regr = elm_train(x, y, nhid = 20, actfun = 'purelin', init_weights = 'uniform_negative')

#---------------
# Classification
#---------------

data(ionosphere, package = 'KernelKnn')

x_class = ionosphere[, -c(2, ncol(ionosphere))]
x_class = as.matrix(x_class)
dimnames(x_class) = NULL

y_class = as.numeric(ionosphere[, ncol(ionosphere)])
y_class_onehot = onehot_encode(y_class - 1)  # class labels should begin from 0

out_class = elm_train(x_class, y_class_onehot, nhid = 20, actfun = 'relu')

onehot_encode

One-hot-encoding of the labels in case of classification

Description
One-hot-encoding of the labels in case of classification

Usage
onehot_encode(y)

Arguments
y a numeric vector consisting of the response variable labels. The minimum value of the unique labels should begin from 0

Examples

library(elmNNRcpp)
y = sample(0:3, 100, replace = TRUE)
y_expand = onehot_encode(y)
Index

elm_predict, 2
elm_train, 3
onehot_encode, 5