Package: elmNNRcpp (via r-universe)

November 16, 2026

Type Package

Title The Extreme Learning Machine Algorithm

Version 1.0.5

Date 2025-09-15

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

URL https://github.com/mlampros/elmNNRcpp,
https://mlampros.github.io/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 7.3.2

Repository https://mlampros.r-universe.dev

Date/Publication 2025-10-01 17:12:49 UTC

RemoteUrl https://github.com/mlampros/elmnnrcpp

RemoteRef HEAD

RemoteSha 8c339b78aefb0ea8f4954bf1bed4d655313ed8fa

Contents

elm 2
elm_predict 4
elm_train 5
onehot_encode 7
predict.elm 8

Index **9**

`elm` *Fit an extreme learning model*

Description

Formula interface for `elm_train`, transforms a data frame and formula into the necessary input for elm_train, automatically calls `onehot_encode` for classification.

Usage

```
elm(
  formula,
  data,
  nhid,
  actfun,
  init_weights = "normal_gaussian",
  bias = FALSE,
  moorep_pseudoinv_tol = 0.01,
  leaky_relu_alpha = 0,
  seed = 1,
  verbose = FALSE
)
```

Arguments

`formula`	formula used to specify the regression or classification.
`data`	data.frame with the data
`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 spcecifying 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

Value

elm object which can be used with predict, residuals and fitted.

Examples

```
elm(Species ~ ., data = iris, nhid = 20, actfun="sig")

mod_elm <- elm(Species ~ ., data = iris, nhid = 20, actfun="sig")

# predict classes
predict(mod_elm, newdata = iris[1:3,-5])

# predict probabilities
predict(mod_elm, newdata = iris[1:3,-5], type="prob")

# predict elm output
predict(mod_elm, newdata = iris[1:3,-5], type="raw")

data("Boston")
elm(medv ~ ., data = Boston, nhid = 40, actfun="relu")

data("ionosphere")
elm(class ~ ., data = ionosphere, nhid=20, actfun="relu")
```

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)


#---------------
# 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

```
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* > *1* and equals to the number of the unique labels.
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 spcecifying 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

```
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)
```

`predict.elm` *Predict with elm*

Description

Wrapper for `elm_predict`.

Usage

```
## S3 method for class 'elm'
predict(object, newdata, type = c("class", "prob", "raw"), ...)
```

Arguments

`object`	elm model fitted with `elm`.
`newdata`	data.frame with the new data
`type`	only used with classification, can be either "class", "prob", "raw", which are class (vector), probability (matrix) or the output of the elm function (matrix).
`...`	not used

Value

predicted values

Index

elm, 2, *8*
elm_predict, 4, *8*
elm_train, *2*, 5

onehot_encode, *2*, 7

predict.elm, 8