Package ‘elmNN’

February 19, 2015

Type Package

Title Implementation of ELM (Extreme Learning Machine) algorithm for SLFN (Single Hidden Layer Feedforward Neural Networks)

Version 1.0

Date 2012-07-17

Author Alberto Gosso

Maintainer Alberto Gosso <gosso.alberto@gmail.com>

Description Training and predict functions for SLFN (Single Hidden-layer Feedforward Neural Networks) using the ELM algorithm. 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.

Depends MASS

License GPL (>= 2)

Keyword models,regression,nonlinear

Repository CRAN

Date/Publication 2012-07-18 11:11:12

NeedsCompilation no

R topics documented:

elmNN-package ........................................... 2
elmtrain .................................................... 3
predict.elmNN ........................................... 5
print.elmNN ............................................. 7

Index 10
elmNN-package

Description

ELM algorithm is an alternative training method for SLFN (Single Hidden Layer Feedforward Networks) which does not need any iterative tuning nor setting parameters such as learning rate, momentum, etc., which are current issues of the traditional gradient-based learning algorithms (like backpropagation).

Training of a SLFN with ELM is a three-step learning model:
Given a training set P = \{(x_i, t_i)\}_{i=1}^{N}, \text{hidden node output function } G(a, b, x), \text{and the number of hidden nodes } L
1) Assign randomly hidden node parameters (a_i, b_i), i = 1,..., L. It means that the arc weights between the input layer and the hidden layer and the hidden layer bias are randomly generated.
2) Calculate the hidden layer output matrix H using one of the available activation functions.
3) Calculate the output weights B: B = ginv(H) \times T (matrix multiplication), where T is the target output of the training set.

ginv(H) is the Moore-Penrose generalized inverse of hidden layer output matrix H. This is calculated by the MASS package function ginv.

Once the SLFN has been trained, the output of a generic test set is simply Y = H \times B (matrix multiplication). Salient features:
- The learning speed of ELM is extremely fast.
- Unlike traditional gradient-based learning algorithms which only work for differentiable activation functions, ELM works for all bounded nonconstant piecewise continuous activation functions.
- Unlike traditional gradient-based learning algorithms facing several issues like local minima, improper learning rate and overfitting, etc, ELM tends to reach the solutions straightforward without such trivial issues.
- The ELM learning algorithm looks much simpler than other popular learning algorithms: neural networks and support vector machines.

Details

Package: elmNN
Type: Package
Version: 1.0
Date: 2012-07-17
License: GPL (>= 2)

To fit a neural network, the function to use is elmtrain (default version is elmtrain.formula). To predict values, the function to use is predict. Other functions are used internally by the training and predict functions.
Author(s)
Alberto Gosso
Maintainer: Alberto Gosso <gosso.alberto@gmail.com>

References
http://www.ntu.edu.sg/home/egbhuang/

See Also
elmtrain.formula to train a neural network, predict.elmnn to predict values from a trained neural network

Examples

```r
set.seed(1234)
Var1 <- runif(50, 0, 100)
sqrt.data <- data.frame(Var1, Sqrt=sqrt(Var1))
model <- elmtrain(Sqrt~Var1, data=sqrt.data, nhid=10, actfun="sig")
new <- data.frame(Sqrt=0, Var1 = runif(50,0,100))
p <- predict(model,newdata=new)

Var2 <- runif(50, 0, 10)
quad.data <- data.frame(Var2, Quad=(Var2)^2)
model <- elmtrain(Quad~Var2, data=quad.data, nhid=10, actfun="sig")
new <- data.frame(Quad=0, Var2 = runif(50,0,10))
p <- predict(model,newdata=new)
```

elmtrain Training of a SLFN (Single Hidden-layer Feedforward Neural Network)

Description
Training of a generic SLFN using ELM algorithm. First it generates input weights and hidden layer bias (both randomly chosen), then calculates the output from the hidden layer (given a particular activation function as a parameter) and at the end calculates output weights of the neural network. It returns an ELM model (an object of class elmnn) representing the trained neural network.
Usage

elmtrain(x, ...)  
## S3 method for class 'formula'
elmtrain(formula, data, nhid, actfun, ...)
## Default S3 method:
elmtrain(x, y, nhid, actfun, ...)

Arguments

- **formula**: a symbolic description of the model to be fitted.
- **data**: training data frame containing the variables specified in formula.
- **x**: training dataset.
- **y**: target output of the training dataset.
- **nhid**: number of hidden neurons. Must be >= 1.
- **actfun**: type of activation function. Furthermore a list of the implemented activation functions.
  - sig: sigmoid
  - sin: sine
  - radbas: radial basis
  - hardlim: hard-limit
  - hardlims: symmetric hard-limit
  - satlins: satlins
  - tansig: tan-sigmoid
  - tribas: triangular basis
  - poslin: positive linear
  - purelin: linear
- ... not used.

Details

**Note**: since part of ELM algorithm is random (setting of the input weights and hidden layer bias), output results of same activation function and same number of hidden neurons may change. To find the most accurate error rate for a fixed setting, it is convenient to make various test on the datasets (i.e. 20 times) using the same settings and calculate the mean error from these tests.

Value

returns the trained neural network, an object of class `elmNN`.

- **nhid**: number of hidden neurons selected
- **actfun**: activation function used
- **inweight**: matrix of input weights (randomly choosen)
- **biashid**: vector of hidden layer bias (randomly choosen)
- **outweight**: matrix of output weights (calculated by the algorithm)
Author(s)

Alberto Gosso

References

see elmNN-package documentation.

See Also

elmtrain.formula, elmtrain.default, predict.elmNN, elmNN-package

Examples

```r
set.seed(1234)
##'formula' version
Var1 <- runif(50, 0, 100)
sqrt.data <- data.frame(Var1, Sqrt=sqrt(Var1))
model <- elmtrain(Sqrt~Var1, data=sqrt.data, nhid=10, actfun="sig")
new <- data.frame(Sqrt=0, Var1 = runif(50,0,100))
p <- predict(model, newdata=new)

##'default' version
Var2 <- runif(50, 0, 10)
quad.data <- data.frame(Var2, Quad=(Var2)^2)
model <- elmtrain(x=quad.data$Var2, y=quad.data$Quad, nhid=10, actfun="sig")
new <- data.frame(Quad=0, Var2 = runif(50,0,10))
p <- predict(model, newdata=new$Var2)

## The function is currently defined as
function (x, ...)
UseMethod("elmtrain")
```

---

**predict.elmNN**

*Calculate the output of the ELM-trained neural network*

**Description**

Calculate the output predictions from a neural network trained using elmtrain. It is possible to calculate output predictions from a new data set or returns the output predictions from the previous training data set (fitted data).

**Usage**

```r
## S3 method for class 'elmNN'
predict(object, newdata = NULL, ...)
```
**predict.elmNN**

**Arguments**

- **object**
  - an object of class `elmNN` (a trained neural network)
- **newdata**
  - (optional) a new data set to calculate output from the model. If missing, `predict` returns the output prediction from the training set.

**Value**

returns a vector containing the output predictions.

**Author(s)**

Alberto Gosso

**References**

see `elmNN-package` documentation.

**See Also**

`elmtrain.default.elmtrain.formula.elmNN-package`

**Examples**

```r
set.seed(1234)
Var1 <- runif(50, 0, 100)
sqrt.data <- data.frame(Var1, Sqrt=sqrt(Var1))
model <- elmtrain.formula(Sqrt~Var1, data=sqrt.data, nhid=10, actfun="sig")
new <- data.frame(Sqrt=0,Var1 = runif(50,0,100))
p <- predict(model,newdata=new)
```

```r
## The function is currently defined as
## function (object, newdata = NULL, ...)
{
  if (is.null(newdata))
    predictions <- fitted(object)
  else {
    if (!is.null(object$formula)) {
      x <- model.matrix(object$formula, newdata)
    } else {
      x <- newdata
    }
    inpweight <- object$inpweight
    biashid <- object$biashid
    outweight <- object$outweight
    actfun <- object$actfun
    nhid <- object$nhid
    TV.P <- t(x)
    tmpHTest = inpweight %% TV.P
    biasMatrixTE <- matrix(rep(biashid, ncol(TV.P)), nrow = nhid,
```
print.elmNN

ncol = ncol(TV.P), byrow = F)

tmpHTest = tmpHTest + biasMatrixTE
if (actfun == "sig")
  HTest = 1/(1 + exp(-1 * tmpHTest))
else {
  if (actfun == "sin")
    HTest = sin(tmpHTest)
  else {
    if (actfun == "radbas")
      HTest = exp(-1 * (tmpHTest^2))
    else {
      if (actfun == "hardlim")
        HTest = hardlim(tmpHTest)
      else {
        if (actfun == "hardlims")
          HTest = hardlims(tmpHTest)
        else {
          if (actfun == "satlins")
            HTest = satlins(tmpHTest)
          else {
            if (actfun == "tansig")
              HTest = 2/(1 + exp(-2 * tmpHTest)) - 1
            else {
              if (actfun == "tribas")
                HTest = tribas(tmpHTest)
              else {
                if (actfun == "poslin")
                  HTest = poslin(tmpHTest)
                else {
                  if (actfun == "purelin")
                    HTest = tmpHTest
                  else stop(paste("ERROR: ", actfun,
                    " is not a valid activation function.",
                    sep = ""))
                }
              }
            }
          }
        }
      }
    }
  }
}
}
}
}
}
TY = t(t(HTest) %*% outweight)
predictions <- t(TY)
predictions

print.elmNN
Print a summary of the attributes of a trained neural network
Description

Print the attributes of an \texttt{elmNN} object.

Usage

\texttt{## S3 method for class \textquotesingle{}elmNN\textquotesingle{}}
\texttt{print(x, ...)}

Arguments

\begin{itemize}
  \item \texttt{x} an object of class \texttt{elmNN}.
  \item \texttt{...} not used.
\end{itemize}

Value

Furthermore a list of the printed attributes.

\begin{itemize}
  \item \texttt{nhid} number of hidden neurons selected
  \item \texttt{actfun} activation function used
  \item \texttt{inpweight} head of the matrix of input weights (randomly calculated)
  \item \texttt{bihid} head of the vector of hidden layer bias (randomly calculated)
  \item \texttt{outweight} head of the matrix of output weights (calculated by the algorithm)
  \item \texttt{fitted(x)} head of the vector with the output prediction of the training set
\end{itemize}

Author(s)

Alberto Gosso

References

see \texttt{elmNN-package} documentation.

See Also

\texttt{elmtrain.default,elmtrain.formula,predict.elmNN,elmNN-package}

Examples

\begin{verbatim}
set.seed(1234)
Var1 <- runif(50, 0, 100)
sqrt.data <- data.frame(Var1, Sqrt=sqrt(Var1))
model <- elmtrain.formula(Sqrt~Var1, data=sqrt.data, nhid=10, actfun="sig")
print(model)

## The function is currently defined as
function (x, ...)
{
  cat("Call:\n")
  cat(paste(x$call, "\n"))
\end{verbatim}
cat("Number of hidden neurons:
")
cat(paste(x$hid, "\n"))
cat("Activation function:
")
cat(paste(x$actfun, "\n"))
cat("Input arc weights:
")
cat(paste(head(as.list(x$inpweight)), "...
"))
cat("Bias of hidden neurons:
")
cat(paste(head(x$biashid), "...
"))
cat("Output arc weights:
")
cat(paste(head(x$outweight), "...
"))
cat("Predictions on training set:
")
cat(paste(head(fitted(x)), "...
"))
Index

*Topic \textasciitilde kwd1
  print.elmNN, 8
*Topic \textasciitilde kwd2
  print.elmNN, 8
*Topic neural
  elmtrain, 3
  predict.elmNN, 5

elmNN (elmNN-package), 2
elmNN-package, 2
elmtrain, 3
elmtrain.default, 5, 6, 8
elmtrain.formula, 3, 5, 6, 8

ginv, 2

predict.elmNN, 3, 5, 5, 8
print.elmNN, 7