Package: dnn (via r-universe)

October 11, 2024

Type Package

Title Deep Neural Network Tools for Probability and Statistic Models

Version 0.0.6

Date 2024-03-12

Author Bingshu E. Chen [aut, cre], Patrick Norman [aut, ctb], Wenyu Jiang [ctb], Wanlu Li [ctb]

Maintainer Bingshu E. Chen <bingshu.chen@queensu.ca>

Depends R (>= 3.5.0), ggplot2, survival, Rcpp

Imports methods

LinkingTo Rcpp, RcppArmadillo

Description Contains tools to build deep neural network with flexible users define loss function and probability models. Several applications included in this package are, 1) The (deepAFT) model, a deep neural network model for accelerated failure time (AFT) model for survival data. 2) The (deepGLM) model, a deep neural network model for generalized linear model (glm) for continuous, categorical and Poisson data.

License GPL (>= 2)

LazyLoad yes

NeedsCompilation yes

Repository CRAN

Date/Publication 2024-03-14 20:50:05 UTC

Contents

dnn-package 2
activation 3
bwdNN 5
deepAFT 6
deepGLM 9
deepSurv 11

dnnControl 13
dnnFit 14
dNNmodel 16
fwdNN 17
hyperTuning 18
ibs 20
msePICW 21
optimizerSGD 22
plot 23
predict 24
print 25
residuals 26
rsurv 27
survfit 28

Index **30**

`dnn-package` *An R package for the deep neural networks probability and statistics models*

Description

This package provides tools for deep neural network which allow user define loss function for complex outcome data with probability and statistics models such as generalized linear models, accelerated failure time (AFT) models, and Cox proportional hazards models.

It contains the essential building blocks such as feed forward network and back propagation. This gives users the flexibility to write their own loss function (i.e. cost function) and train the neural network.

Details

{dnn} is a R package for deep learning neural network with probability models that use the negative of the log-likelihood as the loss function. It provides functions for feed forward network from covariates to the output layer and back propagation to find the derivatives of the weight parameters. Different optimization methods such as stochastic gradient descent (SGD), Momentum and ADAM can be used to train the network.

Currently, { dnn } can be install by

the package source file 'dnn.tar.gz', use

install.packages("dnn.tar.gz", repos = NULL, type = "source")

users can use the following steps to install the most recent version of 'dnn' package:

1. First, you need to install the 'devtools' package. You can skip this step if you have 'devtools' installed in your R. Invoke R and then type

install.packages("devtools")

2. Load the devtools package.

library(devtools)

3. Install "dnn" package from github with R command

install_github("statapps/dnn")

A stable version of View the "dnn" package is also available from the Comprehensive R Archive Network (https://CRAN.R-project.org/package=dnn) and can be installed using R command

install.packages("dnn")

Author(s)

Bingshu E. Chen

Maintainer: Bingshu E. Chen <bingshu.chen@queensu.ca>

See Also

dNNmodel, bwdNN, fwdNN, deepAFT, deepGLM, deepSurv, coxph, glm survival

Examples

```
# Create the models with 3 layers
  model = dNNmodel(units=c(8, 6, 1), activation = c('elu', 'relu', 'sigmoid'),
          input_shape = c(3))
  print(model)
#
# Feed forward network with dummy data x
  x = matrix(runif(15), nrow = 5, ncol = 3)
  cache = fwdNN(x, model)
#
# Back propagation with dummy dy = dL/dyhat and minin batch for SGD
  dy = as.matrix(runif(5, -0.1, 0.1), nrow = 5)
  dW = bwdNN(dy, cache, model)
#
# Gradient descent with SGD
  lr_rate = 0.0001
  sgd = function(w, dw) {w-lr_rate*dw}
  model$params = mapply(sgd, w = model$params, dw = dW)
```

---

activation *Activation function*

Description

Different type of activation functions and the corresponding derivatives

Usage

```
sigmoid(x)
elu(x)
relu(x)
lrelu(x)
idu(x)
dsigmoid(y)
delu(y)
drelu(y)
dlrelu(y)
dtanh(y)   #activation function tanh(x) is already available in R
```

Arguments

x	input of the activation function
y	input of the derivative of the activation function

Details

Each function returns either the activation function (e.g. sigmoid, relu) or its derivative (e.g. dsigmoid, drelu).

Value

An activation function is applied to x and returns a matrix the same size as x. The detail formula for each activation function is:

sigmoid	return 1/(1+exp(-x))
elu	return x for x>0 and exp(x)-1 for x<0
relu	return x for x>0 and 0 for x<0
lrelu	return x for x>0 and 0.1*x for x<0
tanh	return tanh(x)
idu	return (x)

Author(s)

Bingshu E. Chen

See Also

bwdNN, fwdNN, dNNmodel, optimizerSGD, optimizerNAG

Examples

```
# Specify a dnn nodel with user define activation function in layer 2.
softmax  = function(x) {log(1+exp(x))}    # y = log(1+exp(x))
dsoftmax = function(y) {sigmoid(y)}       # x = exp(y)/(1+exp(y))
model = dNNmodel(units=c(8, 6, 1), activation= c('relu', 'softmax', 'sigmoid'),
```

```
          input_shape = c(3))
print(model)
```

---

bwdNN *Back propagation for dnn Models*

Description

{bwdNN} is an R function for back propagation in DNN network.

Usage

```
#
# To apply back propagation in with a feed forward model
#
# use
#
   bwdNN(dy, cache, model)
#
# to calculate derivative of dL/dW
```

Arguments

dy	the derivative of the cost function with respect to the output layer of the fwdNN function.
cache	the cached output of fwdNN.
model	a model return from dNNmodel function.

Details

Here 'dy' plays an import role in the back propagation { bwdNN } since the probability model's loss function takes the output layer of the { dnn } (denote as yhat) as one of its parameter. Then 'dy' equals to the partial derivative of the loss function (-Log Likelihood) with respect to yhat, that is, dy = dL/d(yhat). For example, if the 'dnn' predicts the probability (yhat = p) for the mixture of two populations f1 and f2, then the likelihood function is f = p*f1 + (1-p)*f2, and the loss function is L = -log(p*f1+(1-p)*f2). Hence, dy = dL/dp = -(f1-f2)/f.

'cache' is the cache of each input layer generated from the { fwdNN } function.

The function { bwdCheck } calculates the numerical derivatives of dL/dW, which can be used to check if the back propagation is correct or not, see example below.

Value

A list contains the derivatives of weight parameter W is returned.

Author(s)

Bingshu E. Chen (bingshu.chen@queensu.ca)

See Also

dNNmodel, fwdNN, plot.dNNmodel, print.dNNmodel, summary.dNNmodel,

Examples

```
### define a dnn model, calculate the feed forward network
   model = dNNmodel(units = c(8, 6, 1),
           activation = c("elu", "sigmoid", "sigmoid"), input_shape = 3)
   print(model)
   x = matrix(runif(15), nrow = 5, ncol = 3)
   cache = fwdNN(x, model)
   # dy = dL/dp, where L is the cost function such as the
   # log-likehood and p is the output layer parameter of the DNN
   dy = as.matrix(runif(5, -0.1, 0.1), nrow = 5)  # a dummy dy for bwdNN input
   y  = predict(model, x) + dy

   # back propagation
   dW = bwdNN(dy, cache, model)
   dw = bwdCheck(x, y, model)
   print(dW[[1]])
   print(dw[[1]])
```

---

deepAFT *Deep learning for the accelerated failure time (AFT) model*

Description

Fit a deep learning survival regression model. These are location-scale models for an arbitrary transform of the time variable; the most common cases use a log transformation, leading to accelerated failure time models.

Usage

```
deepAFT(x, ...)

## S3 method for class 'formula'
deepAFT(formula, model, data, control = list(...), method =
                 c("BuckleyJames", "ipcw", "transform"), ...)

## Default S3 method:
deepAFT(x, y, model, control, ...)

## S3 method for class 'ipcw'
deepAFT(x, y, model, control, ...)
# use:
#   deepAFT.ipcw(x, y, model, control)
# or
#   class(x) = "ipcw"
```

```
#   deepAFT(x, y, model, control)
#
## S3 method for class 'trans'
deepAFT(x, y, model, control, ...)
# use:
#   class(x) = "transform"
#   deepAFT(x, y, model, control)
```

Arguments

`formula`	a formula expression as for other regression models. The response is usually a survival object as returned by the 'Surv' function. See the documentation for 'Surv', 'lm' and 'formula' for details.
`model`	deep neural network model, see below for details.
`data`	a data.frame in which to interpret the variables named in the formula.
`x`	Covariates for the AFT model
`y`	Surv object for the AFT model
`method`	methods to handle censoring data in deep AFT model fit, 'BuckleyJames' for Buckley and James method, 'ipcw' for inverse probability censoring weights method. 'transform' for transformation based on book of Fan and Gijbels (1996, page 168)
`control`	a list of control values, in the format produced by 'dnnControl'. The default value 'dnnControl()'
`...`	optional arguments

Details

See "Deep learning with R" for details on how to build a deep learning model.

The following parameters in 'dnnControl' will be used to control the model fit process.

'epochs': number of deep learning epochs, default is 100.

'batch_size': batch size, default is 128. 'NaN' may be generated if batch size is too small and there is not event in a batch.

'verbose': verbose = 1 for print out verbose during the model fit, 0 for not print.

'epsilon': epsilon for convergence check, default is epsilon = 0.001.

'max.iter': number of maximum iteration, default is max.iter = 100.

'censor.groups': a vector for censoring groups. A KM curve for censoring will be fit for each group. If a matrix is provided, then a Cox model will be used to predict the censoring probability.

When the variance for covariance matrix X is too large, please use xbar = apply(x, 2, stndx) to standardize X.

Value

An object of class "deepAFT" is returned. The deepAFT object contains the following list components:

x	Covariates for the AFT model
y	Survival object for the AFT model, y = Surv(time, event)
model	A fitted artificial neural network (ANN) model
mean.ipt	mean survival or censoring time
predictor	predictor score mu = f(x)
risk	risk score = exp(predictor)
method	method for deepAFT fitting, either Buckley-James, IPCW or transformed model

Note

For right censored survival time only

Author(s)

Chen, B. E. and Norman P.

References

Buckley, J. and James, I. (1979). Linear regression with cencored data. Biometrika, 66, page 429-436.

Norman, P. Li, W., Jiang, W. and Chen, B. E. (2024). DeepAFT: A nonparametric accelerated failure time model with artificial neural network. Manuscript submitted to Statistics in Medicine.

Chollet, F. and Allaire J. J. (2017). Deep learning with R. Manning.

See Also

print.deepAFT, survreg, ibs.deepAFT

Examples

```
## Example for deep learning model for AFT survival data
  set.seed(101)
### define model layers
  model = dNNmodel(units = c(4, 3, 1), activation = c("elu", "sigmoid", "sigmoid"),
                   input_shape = 3)
  x = matrix(runif(15), nrow = 5, ncol = 3)
  time = exp(x[, 1])
  status = c(1, 0, 1, 1, 1)
  fit = deepAFT(Surv(time, status) ~ x, model)
```

deepGLM *Deep learning for the generalized linear model*

Description

Fit generalized linear models (Gaussian, Binomial and Poisson) using deep learning neural network (DNN). The glm formula is specified by giving a symbolic description of the predictor and a description of the error distribution.

Usage

```
deepGlm(formula, model, family = c("gaussian", "binomial",
        "poisson"), data, epochs = 200, lr_rate = 1e-04,
         batch_size = 64, alpha = 0.7, lambda = 1, verbose = 0,
         weights = NULL, ...)
```

Arguments

formula	a formula expression as for other regression models. The response is usually an object for glm response variable. See the documentation for 'glm', 'lm' and 'formula' for details.
model	a deep neural network model, created by function dNNmodel().
family	a description of the error distribution and link function to be used in the model. This can be either a character string of 'gaussian', 'binomial', or 'poisson', naming a family function, or result of a call to a family function (See 'family' for details of family functions).)
data	a data.frame in which to interpret the variables named in the formula.
epochs	number of deep learning epochs, default is 200.
batch_size	batch size, default is 64. 'NaN' may be generated if batch size is too small and there is not event in a batch.
lr_rate	learning rate for the gradient descent algorithm, default is lr_rate = 1e-04.
weights	an optional vector of 'prior weights' to be used in the fitting process. Should be NULL or a numeric vector.
alpha	momentum rate for the gradient descent method, alpha takes value in [0, 1), default is alpha = 0.70.
lambda	L2 regularization parameter for deep learning.
verbose	verbose = 1 for print out verbose during the model fit, 0 for not print.
...	optional arguments

Details

See dNNmodel for details on how to specify a deep learning model.

The following parameters in 'dnnControl' will be used to control the model fit process.

'epochs': number of deep learning epochs, default is 30.

'verbose': verbose = 1 for print out verbose during the model fit, 0 for not print.

When the variance for covariance matrix X is too large, please use xbar = scale(x) to standardize X.

Value

An object of class "deepGlm" is returned. The deepGlm object contains the following list components:

`x`	Covariates for glm model
`y`	Object for glm model
`model`	dnn model
`predictor`	predictor score mu = f(x)
`risk`	risk score = exp(predictor)

Note

For glm models with Gaussian, Binomial and Poisson only

Author(s)

Chen, B. E.

References

Chollet, F. and Allaire J. J. (2017). Deep learning with R. Manning.

See Also

deepAFT, dNNmodel, predict.deepGlm, print.deepSurv, glm

Examples

```
## Example for deep learning for glm models
  set.seed(101)
### define model layers
  model = dNNmodel(units = c(4, 3, 1), activation = c("elu", "sigmoid", "sigmoid"),
                   input_shape = 3)
  x = matrix(runif(15), nrow = 5, ncol = 3)
  y = exp(x[, 1] + rnorm(5))

  fit = deepGlm(y ~ x, model, family = "gaussian")
```

deepSurv *Deep learning for the Cox proportional hazards model*

Description

Fit a survival regression model under the Cox proportional hazards assumption using deep learning neural network (DNN).

Usage

```
deepSurv(formula, model, data, epochs = 200, lr_rate = 1e-04,
        batch_size = 64, alpha = 0.7, lambda = 1, verbose = 0,
        weights = NULL, ...)
```

Arguments

`formula`	a formula expression as for other regression models. The response is usually a survival object as returned by the 'Surv' function. See the documentation for 'Surv', 'lm' and 'formula' for details.
`model`	a deep neural network model, created by function dNNmodel().
`data`	a data.frame in which to interpret the variables named in the formula.
`epochs`	number of deep learning epochs, default is 200.
`batch_size`	batch size, default is 64. 'NaN' may be generated if batch size is too small and there is not event in a batch.
`lr_rate`	learning rate for the gradient descent algorithm, default is lr_rate = 1e-04.
`weights`	an optional vector of 'prior weights' to be used in the fitting process. Should be NULL or a numeric vector.
`alpha`	momentum rate for the gradient descent method, alpha takes value in [0, 1), default is alpha = 0.70.
`lambda`	L2 regularization parameter for deep learning.
`verbose`	verbose = 1 for print out verbose during the model fit, 0 for not print.
`...`	optional arguments

Details

See "Deep learning with R" for details on how to build a deep learning model.

The following parameters in 'dnnControl' will be used to control the model fit process.

'epochs': number of deep learning epochs, default is 30.

'verbose': verbose = 1 for print out verbose during the model fit, 0 for not print.

'epsilon': epsilon for convergence check, default is epsilon = 0.001.

'max.iter': number of maximum iteration, default is max.iter = 30.

When the variance for covariance matrix X is too large, please use xbar = scale(x) to standardize X.

Value

An object of class "deepSurv" is returned. The deepSurv object contains the following list components:

`x`	Covariates for Cox model
`y`	Surv object for Cox model
`model`	dnn model
`predictor`	predictor score mu = f(x)
`risk`	risk score = exp(predictor)

Note

For right censored survival time only

Author(s)

Chen, B. E. wrote the R code using the partial likelihood cost function proposed by Katzman et al (2018).

References

Katzman JL, Shaham U, Cloninger A, Bates J, Jiang T, Kluger Y. DeepSurv: Personalized treatment recommender system using a Cox proportional hazards deep neural network. BMC Medical Research Methodology 2018; 18: 24.

See Also

`deepAFT`, `deepGlm`, `print.deepSurv`, `survreg`

Examples

```
## Example for deep learning proportional hazards survival model
  set.seed(101)
### define model layers
  model = dNNmodel(units = c(4, 3, 1), activation = c("elu", "sigmoid", "sigmoid"),
                   input_shape = 3)
  x = matrix(runif(15), nrow = 5, ncol = 3)
  time = exp(x[, 1])
  status = c(1, 0, 1, 1, 1)
  fit = deepSurv(Surv(time, status) ~ x, model = model)
```

dnnControl *Auxiliary function for* dnnFit *dnnFit*

Description

dnnControl is an auxiliary function for dnnFit. Typically only used internally by the dnn package, may be used to construct a control argument for the deep learning neural network model to specify parameters such as a loss function.

Usage

```
dnnControl(loss = c("mse", "cox", "bin", "log", "mae"), epochs = 300,
    batch_size = 64, verbose = 0, lr_rate = 0.0001,
    alpha = 0.5, lambda = 1.0, epsilon = 0.01, max.iter = 100,
    censor.group = NULL, weights = NULL)
```

Arguments

loss	loss function for the neural network model, "mse" for mean square error (guassian glm model), "mae" for mean absolute error, "cox" for the Cox partial likelihood (proportional hazards model), "bin" for cross-entropy (binomial glm model), "log" for log-linear (poisson glm model).
epochs	number of deep learning epochs, default is 30.
batch_size	batch size, default is 64. 'NaN' may be generated if batch size is too small and there is not event in a batch.
lr_rate	learning rate, default is 0.0001.
weights	an optional vector of 'prior weights' to be used in the fitting process. Should be NULL or a numeric vector, default is NULL.
alpha	alpha decay rate for momentum gradient descent, default is 0.5.
lambda	regularization term for dnn weighting parameters, 0.5*lambda*W*W), default is 1.0.
verbose	verbose = 1 for print out verbose during the model fit, 0 for not print.
epsilon	epsilon for convergence check, default is epsilon = 0.01.
max.iter	number of maximum iteration, default is max.iter = 100. This is used in the deepAFT function
censor.group	a vector for censoring groups. A KM curve for censoring will be fit for each group. If a matrix is provided, then a Cox model will be used to predict the censoring probability. Used only in the deepAFT function.

Details

dnnControl is used in model fitting of "dnnFit". Additional loss functions will be added to the library in the future.

Value

This function checks the internal consistency and returns a list of values as input to control model fitting of "dnnFit".

Note

For right censored survival time only

Author(s)

Chen, B. E.

References

Norman, P. and Chen, B. E. (2023). DeepAFAT: A nonparametric accelerated failure time model with artificial neural network. Manuscript to be submitted.

See Also

deepAFT, deepGLM, deepSurv, dnnFit

Examples

```
## Example for dnnControl
##
# model = dNNmodel()

  control = dnnControl(loss='mse')

# can also be used in
# fit = dnnFit(y ~ x, model, control)
# print(fit)
```

---

dnnFit *Fitting a Deep Learning model with a given loss function*

Description

dnnFit is used to train a deep learning neural network model based on a specified loss function.

Usage

```
dnnFit(x, y, model, control)
```

Arguments

`x`	covariates for the neural network model
`y`	output (target) value for neural network model
`model`	the neural network model, see below for details
`control`	a list of control values, in the format produced by 'dnnControl'. The default value is dnnControl(loss='mse')

Details

The 'dnnFit' function takes the input data, the target values, the network architecture, and the loss function as arguments, and returns a trained model that minimizes the loss function. The function also supports various options for regularization and optimization of the model.

See `dNNmodel` for details on how to specify a deep learning model.

Parameters in `dnnControl` will be used to control the model fit process. The loss function can be specified as dnnControl(loss = "lossFunction"). Currently, the following loss functions are supported:

'mse': Mean square error loss = 0.5*sum(dy^2)

'cox': Cox partial likelihood loss = -sum(delta*(yhat - log(S0)))

'bin': Cross-entropy = -sum(y*log(p) + (1-y)*log(1-p))

'log': Log linear cost = -sum(y*log(lambda)-lambda)

'mae': Mean absolute error loss = sum(abs(dy))

Additional loss functions will be added to the library in the future.

{ dnnFit2 } is a C++ version of dnnFit, which runs about 20% faster, however, only loss = 'mse' and 'cox' are currently supported.

When the variance for covariance matrix X is too large, please use xbar = scale(x) to standardize X.

Value

An object of class "dnnFit" is returned. The dnnFit object contains the following list components:

`cost`	cost at the final epoch.
`dW`	the gradient at the final epoch dW = dL/dW.
`fitted.values`	predictor value mu = f(x).
`history`	a cost history at each epoch.
`lp`	predictor value mu = f(x).
`logLik`	-2*log Likelihood = cost.
`model`	a dNNmodel object.
`residuals`	raw residual dy = d log(L)/dmu
`dvi`	deviance dvi = dy*dy

Author(s)

Chen, B. E. and Norman P.

References

Buckley, J. and James, I. (1979). Linear regression with censored data. Biometrika, 66, page 429-436.

Norman, P. and Chen, B. E. (2019). DeepAFAT: A nonparametric accelerated failure time model with artificial neural network. Manuscript to be submitted.

Chollet, F. and Allaire J. J. (2017). Deep learning with R. Manning.

See Also

deepAFT, deepGlm, deepSurv, dnnControl

Examples

```
## Example for dnnFit with MSE loss function to do a non-linear regression
  set.seed(101)
### define model layers
  model = dNNmodel(units = c(4, 3, 1), activation = c("elu", "sigmoid", "sigmoid"),
                   input_shape = 3)
  x = matrix(runif(15), nrow = 5, ncol = 3)
  y = exp(x[, 1])
  control = dnnControl(loss='mse')
  fit = dnnFit(x, y, model, control)
```

---

dNNmodel *Specify a deep neural network model*

Description

{dNNmodel} is an R function to create a deep neural network model that is to be used in the feed forward network { fwdNN } and back propagation { bwdNN }.

Usage

```
dNNmodel(units, activation=NULL, input_shape = NULL, type = NULL,
          N = NULL, Rcpp=TRUE, optimizer = c("momentum", "nag", "adam"))
```

Arguments

units	number of nodes for each layer
activation	activation function
input_shape	the number of columns of input X, default is NULL.
N	the number of training sample, default is NULL.
type	default is "dense", currently only support dense layer.
Rcpp	use Rcpp (C++ for R) to speed up the fwdNN and bwdNN, default is "TRUE".
optimizer	optimizer used in SGD, default is "momentum".

Details

dNNmodel returns an object of class "dNNmodel".

The function "print" (i.e., "print.dNNmodel") can be used to print a summary of the dnn model,

The function "summary" (i.e., "summary.dNNmodel") can be used to print a summary of the dnn model,

Value

An object of class "dNNmodel" is a list containing at least the following components:

`units`	number of nodes for each layer
`activation`	activation function
`drvfun`	derivative of the activation function
`params`	the initial values of the parameters, to be updated in model training.
`input_shape`	the number of columns of input X, default is NULL.
`N`	the number of training sample, default is NULL.
`type`	default is "dense", currently only support dense layer.

Author(s)

Bingshu E. Chen (bingshu.chen@queensu.ca)

See Also

`plot.dNNmodel`, `print.dNNmodel`, `summary.dNNmodel`, `fwdNN`, `bwdNN`, `optimizerSGD`, `optimizerNAG`,

Examples

```
### To define a dnn model
 model = dNNmodel(units = c(8, 6, 1), activation = c("relu", "sigmoid", "sigmoid"),
        input_shape = c(3))
```

`fwdNN` *Feed forward and back propagation for dnn Models*

Description

{fwdNN} is an R function for feed forward network.

Usage

```
   fwdNN(X, model)
#
# to calculate a feed feedward model
#
```

Arguments

`X`	For "dNNmodel", X is a design matrix of dimension n * p.
`model`	a model return from dNNmodel function.

Details

'cache' is the cache of each input layer, will be used in the bwdNN function.

Value

The function fwdNN return a list containing at least the following components:

`cache`	a list contains the values of each output layer after activation function transformation and adding the intercept term (i.e. the bias term). The intercept does not add to the output layer in the cache.

Author(s)

Bingshu E. Chen (bingshu.chen@queensu.ca)

See Also

`bwdNN`, `plot.dNNmodel`, `print.dNNmodel`, `summary.dNNmodel`,

Examples

```
### define a dnn model, calculate the feed forward network
   model = dNNmodel(units = c(8, 6, 1), activation = c("elu", "sigmoid", "sigmoid"),
                   input_shape = 3)

### feed forward with a dummy x matrix
   x = matrix(runif(15), nrow = 5, ncol = 3)
   cache = fwdNN(x, model)
```

`hyperTuning` *A function for tuning of the hyper parameters*

Description

{ hyperTuning} is a tuning tool to find the optimal hyper parameter for the ANN model.

Usage

```
    hyperTuning(x, y, model, ER = c("cindex", "mse"),
           method = c('BuckleyJames', 'ipcw', 'transform', 'deepSurv'),
           lower = NULL, upper = NULL, node = FALSE,
           K = 5, R = 25)
### additional function used in hyperTuning is cross-validation prediction error
#
#  CVpredErr(x, y, model, control, method)
#
```

Arguments

`x`	Covariates for the deep neural network model
`y`	Surv object for the deep neural network model
`model`	A deep neural network model, created by function dNNmodel().
`ER`	Prediction error measurement to be used in the cross vaditation, can be either a concordance index (cindex) or a mean square error (mse), default is cindex
`method`	Methods to handle censoring data in deep AFT model fit, 'BuckleyJames' for the Buckley and James method, 'ipcw' for the inverse probability censoring weights method. 'transform' for the transformation method based on book of Fan and Gijbels (1996, page 168). 'deepSurv' for the deepSurv model(Katzman, 2017)
`node`	Tuning the number of nodes in each hidden layer, default is FALSE
`K`	Number of folders of the cross-validatin, default is K = 5.
`lower, upper`	Bounds on the hyper parameters for the deep learning method. If NULL, then the default value for lower = dnnControl(alpha = 0.5, lambda = 1.0, lr_rate = 0.0001), upper = dnnControl(alpha = 0.97, lambda = 10, lr_rate = 0.001).
`R`	Number of random sample draw from the hyper parameter space, default is R = 25.

Details

A random search method is used to optimal hyper parameter (Bergstra and Bengio, 2012). The function { CVpredErr} will be call to calculate the cross-validation prediction error for the given x and y with the specified method from the input argument.

Value

A list of "model" and "dnnControl" is returned. The list contains at least the following components,

`model`	The "model" contains the optimal number of nodes for each hidden layer in the model specified by `dNNmodel`
`control`	The "control" contains the optimal tuning parameters with list components the same as those created by `dnnControl`

Author(s)

Chen, B. E. (chenbe@queensu.ca)

References

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. The Journal of Machine Learning Research. 13, page 281-305.

See Also

deepAFT, deepGLM, deepSurv, dnnFit

Examples

```
### Tuning the hyper parameter for a deepAFT model:
#### cross-validation take a long time to run.

  set.seed(101)
### define model layers
  model = dNNmodel(units = c(4, 3, 1), activation = c("elu", "sigmoid", "sigmoid"),
                   input_shape = 3)
  x = matrix(runif(45), nrow = 15, ncol = 3)
  time = exp(x[, 1])
  status = rbinom(15, 1, 0.5)
  y = Surv(time, status)
  ctl = dnnControl(epochs = 30)
  hyperTuning(x, y, model, method = "BuckleyJames", K = 2, R = 2, lower = ctl)
```

---

ibs — *Calculate integrated Brier Score for deepAFT*

Description

The function ibs is used to calculate integrated Brier Score for deepAFT.

Usage

```
ibs(object, ...)
### To calculate Brier score for the original fitted data
## Default S3 method:
ibs(object, ...)
### To calculate Brier score for new data with new outcomes
## S3 method for class 'deepAFT'
ibs(object, newdata=NULL, newy = NULL, ...)
```

Arguments

object	the results of a deepAFT fit.
newdata	optional argument, if no null, new data and new y will be used for calculation.
newy	optional argument, used together with new data.
...	other unused arguments.

Details

ibs is called to calculate integrate Brier score for the deepAFT model `deepAFT`.

Value

A list contains the integrate Brier score and the Brier score is returned:

`ibs`	Integerate Brier score
`bs`	Brier score

Author(s)

Bingshu E. Chen

See Also

`deepAFT`

`msePICW` *Mean Square Error (mse) for a survival Object*

Description

Compute Mean Square Error (mse) values for a survival object

Usage

```
## S3 method for class 'deepAFT'
mseIPCW(object, newdata, newy)
```

Arguments

`object`	the results of a model fit using a deepAFT or a survreg function.
`newdata`	optional new data at which to do predictions. If absent, predictions are for the dataframe used in the original fit.
`newy`	optional new outcome variable y.

Details

predict is called to predict object from a deepAFT `deepAFT` or a survreg model.

IPCW method is used to calcuate the mean square error for censored survival time.

Value

mseIPCW returns the mse for the predicted survival data.

Author(s)

Bingshu E. Chen

See Also

The default method for predict `predict`, `deepAFT`, `survfit.dSurv`

`optimizerSGD` *Functions to optimize the gradient descent of a cost function*

Description

Different type of optimizer functions such as SGD, Momentum, AdamG and NAG.

Usage

```
optimizerMomentum(V, dW, W, alpha = 0.63, lr = 1e-4, lambda = 1)
```

Arguments

`V`	Momentum V = alpha*V - lr*(dW + lambda*W); W = W + V. NAG V = alpha*(V - lr*(dW + lambda*W); W = W + V - lr*(dW + lambda*W)
`dW`	derivative of cost with respect to W, can be founde by dW = bwdNN2(dy, cache, model),
`W`	weights for DNN model, optimizerd by W = W + V
`alpha`	Momentum rate 0 < alpha < 1, default is alpah = 0.5.
`lr`	learning rate, default is lr = 0.001.
`lambda`	regulation rate for cost + 0.5*lambda*||W||, default is lambda = 1.0.

Details

For SGD with momentum, use

V = 0; obj = optimizerMomentum(V, dW, W); V = obj$V; W = obj$W

For SDG with MAG

V = 0; obj = optimizerNAG(V, dW, W); V = obj$V; W = obj$W

Value

return and updated W and other parameters such as V, V1 and V2 that will be used on SGD.

Author(s)

Bingshu E. Chen

See Also

`activation`, `bwdNN`, `fwdNN`, `dNNmodel`, `dnnFit` `plot` *Plot methods in dnn package*

Description

Plot function for plotting of R objects in the dnn package.

Several different type of plots can be produced for the deep learning mdels. Plot method is used to provide a summary of outputs from "deepAFT", "deepGLM", "deepSurv" and "dnn".

Use "methods(plot)" and the documentation for these for other plot methods.

Usage

```
## S3 method for class 'dNNmodel'
plot(x, ...)
## S3 method for class 'deepAFT'
plot(x, type = c("predicted", "residuals", "baselineKM"), ...)
```

Arguments

`x`	a class of "dNNmodel".
`type`	type of plot in deepAFT object, "predicted" to plot the linear predicted values, "residuals" to plot residuals, "baselineKM" to plot baseline Kaplan-Meier survival curve.
`...`	other options used in plot().

Details

plot.deepAFT is called to plot the fitted deep learning AFT model.

plot.dNNmodel is called to plot fitted dnn model

The default method, plot.default has its own help page. Use methods("plot") to get all the methods for the plot generic.

Value

No return value, called to plot a figure.

Author(s)

Bingshu E. Chen

See Also

The default method for plot `plot.default`. `glm` predict *Predicted Values for a deepAFT Object*

Description

Compute predicted values for a deepAFT object

Usage

```
## S3 method for class 'deepAFT'
## S3 method for class 'dSurv'
predict(object, newdata, newy=NULL, ...)
```

Arguments

`object`	the results of a model fit using the deepAFT function.
`newdata`	optional new data at which to do predictions. If absent, predictions are for the dataframe used in the original fit.
`newy`	optional new outcome variable y.
`...`	other options used in predict().

Details

predict.dSurv is called to predict object from the deepAFT or deepSurv model `deepAFT`.

The default method, predict has its own help page. Use methods("predict") to get all the methods for the predict generic.

Value

predict.dSurv returns a list of predicted values, prediction error and residuals.

`lp`	linear predictor of beta(w)*Z, where beta(w) is the fitted regression coefficient and Z is covariance matrix.
`risk`	risk score, exp(lp). When new y is provided, both lp and risk will be ordered by survival time of the new y.
`cumhaz`	cumulative hzard function.
`time`	time for cumulative hazard function. Time from new y will be used is provided

Author(s)

Bingshu E. Chen

See Also

The default method for predict `predict`, `deepAFT`, `survfit.dSurv`

print — *print a summary of fitted deep learning model object*

Description

print is used to provide a short summary of outputs from deepAFT, deepSurv, deepGLM, and dNNmodel.

Usage

```
## S3 method for class 'deepAFT'
print(x, ...)
## S3 method for class 'summary.deepAFT'
print(x, ...)
## S3 method for class 'deepAFT'
summary(object, ...)

## S3 method for class 'dNNmodel'
print(x, ...)
## S3 method for class 'dNNmodel'
summary(object, ...)
```

Arguments

`x`	a class returned from deepAFT, deepSurv, deepGLM model fit or a dNNmodel
`object`	a class of deepAFT object
`...`	other options used in print()

Details

print.deepAFT is called to print object or summary of object from the deep learning AFT models deepAFT. summary(fit) provides detail summary of ‘deepAFT’ model fit, including predictors, baseline survival function for T0=T/exp(mu), and martingale residuals for the fitted model.

print.dNNmodel is called to print object or summary of object from the dNNmodel.

The default method, print.default has its own help page. Use methods("print") to get all the methods for the print generic.

Value

An object of class "summary.deepAFT" is returned. The object contains the following list components:

`location`	location parameter exp(mu), to predice the mean value of survival time.
`sfit`	survfit object of the baselie survival function of T0=T/exp(mu).
`cindex`	Concordance index of the fitted deepAFT model.
`resid`	martingle residuals of the fitted deepAFT model.
`method`	the model used to fit the deepAFT model.

Author(s)

Bingshu E. Chen

See Also

The default method for print `print.default`. Other methods include `survreg`, `deepAFT`, `summary`

`residuals` *Calculate Residuals for a deepAFT Fit.*

Description

Calculates martingale, deviance or Cox-Snell residuals for a previously fitted (deepAFT) model.

Usage

```
## S3 method for class 'deepAFT'
## S3 method for class 'dSurv'
residuals(object, type = c("martingale", "deviance", "coxSnell"), ...)
```

Arguments

`object`	the results of a (deepAFT) fit.
`type`	character string indicating the type of residual desired. Possible values are "martingale", "deviance". Only enough of the string to determine a unique match is required.
`...`	other unused arguments.

Details

residuals.deepAFT is called to compute baseline survival function S_T0(t) from the deepAFT model `deepAFT`, where T0 = T/exp(mu), or log(T) = log(T) - mu.

The default method, residuals has its own help page. Use methods("residuals") to get all the methods for the residuals generic.

Value

For martingale and deviance residuals, the returned object is a vector with one element for each subject. The row order will match the input data for the original fit.

See `residuals` for more detail about other output values.

Note

For deviance residuals, the status variable may need to be reconstructed.

Author(s)

Bingshu E. Chen

See Also

The default method for residuals `residuals`, `predict.dSurv`, `survfit.dSurv`, and `deepAFT`.

---

rsurv *The Survival Distribution*

Description

Density, distribution function, quantile function and random variable generation for a survival distribution with a provided hazard function or cumulative hazard function

Usage

```
dsurv(x, h0 = NULL, H0 = function(x){x}, log=FALSE)
psurv(q, h0 = NULL, H0 = function(x){x}, low.tail=TRUE, log.p=FALSE)
qsurv(p, h0 = NULL, H0 = function(x){x}, low.tail=TRUE)
rsurv(n, h0 = NULL, H0 = function(x){x})
rcoxph(n, h0 = NULL, H0 = function(x){x}, lp = 0)
```

Arguments

`x, q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations.
`h0`	hazard function, default is h0 = NULL.
`H0`	cumulative hazard function, default is H0(x) = x.
`lp`	linear predictor for rcoxph, H(x) = H0(x)exp(lp).
`log, log.p`	logical; if TRUE, probabilities p are give as log(p).
`low.tail`	logical; if TRUE, probabilities are P[X < or = x] otherwise, S(x) = P[X>x].

Details

If { h0 } or { H0 } are not specified, they assume the default values of h0(x) = 1 and H0(x) = x, respectively.

The survival distribution function is given by,

S(x) = exp(-H0(x)),

where H0(x) is the cumulative hazard function. Only one of h0 or H0 can be specified, if h0 is given, then H0(x) = integrate(h0, 0, x, subdivisions = 500L)

To generate Cox PH survival time, use

u = exp(-H(t)*exp(lp))

then, -log(u)*exp(-lp) = H(t). Find t such that H(t) = -log(u)exp(-lp).

Value

{ dsurv } gives the density h(x)/S(x), { psurv } gives the distribution function, { qsurv } gives the quantile function, { rsurv } generates random survival time, and { rcoxph } generates random survival time with Cox proportional hazards model.

The length of the result is determined by n for rsurv and rcoxph.

Author(s)

Bingshu E. Chen

References

Johnson, N. L., Kotz, S. and Balakrishnan, N. (1995). Continuous Univariate Distributions, volume 1. Wiley, New York.

See Also

`Distributions` for other standard distributions, including `dweibull` for the Weibull distribution.

Examples

```
#### use qsurv to generate quantiles for weibull distribution
H1 = function(x) x^3
qsurv(seq(0.1, 0.9, 0.2), H0 = H1) ### shall be the same as
qweibull(seq(0.1, 0.9, 0.2), 3)
#### to get random survival time from the cumulative hazard function H1(t)
rsurv(15, H0 = H1)
```

---

survfit — *Compute a Survival Curve from a deepAFT or a deepSurv Model*

Description

Computes the predicted survival function of a previously fitted deepAFT or deepSurv model.

Usage

```
## S3 method for class 'deepAFT' or 'deepSurv'
## S3 method for class 'dSurv'
survfit(formula, se.fit=TRUE, conf.int=.95, ...)
```

Arguments

`formula`	a deepAFT or deepSurv fit object.
`se.fit`	a logical value indicating whether standard errors shall be computed. Default is TRUE
`conf.int`	the level for a two-sided confidence interval on the survival curve. Default is 0.95
`...`	other unused arguments.

Details

survfit.dSurv is called to compuate baseline survival function S_T0(t) from the deepAFT model `deepAFT`, where T0 = T/exp(mu), or log(T) = log(T) - mu.

For the deepSurv model `deepAFT`, survfit.dSurv evaluates the Nelson-Aalen estimate of the baseline survival function.

The default method, survfit has its own help page. Use methods("survfit") to get all the methods for the survfit generic.

Value

survfit.deepAFT returns a list of predicted baseline survival function, cumulative hazard function and residuals.

`surv`	Predicted baseline survival function for T0=T/exp(mu).
`cumhaz`	Baseline cumulative hazard function, -log(surv).
`hazard`	Baseline hazard function.
`varhaz`	Variance of the baseline hazard.
`residuals`	Martingale residuals of the (deepAFT) model.
`std.err`	Standard error for the cumulative hazard function, if se.fit = TRUE.

See `survfit` for more detail about other output values such as upper, lower, conf.type. Confidence interval is based on log-transformation of survival function.

Author(s)

Bingshu E. Chen

See Also

The default method for survfit `survfit`, `predict.dSurv`

Index

∗ **Back propagation**
activation, 3
bwdNN, 5
∗ **Cox PH random variable**
rsurv, 27
∗ **Deep Neural Networks**
dnn-package, 2
∗ **Deep Neural Network**
activation, 3
bwdNN, 5
dNNmodel, 16
fwdNN, 17
∗ **Feed forward**
activation, 3
fwdNN, 17
∗ **Hyper parameter**
hyperTuning, 18
∗ **IPCW**
msePICW, 21
∗ **Integrated Brier Score**
ibs, 20
∗ **MSE**
msePICW, 21
∗ **Survival distribution**
rsurv, 27
∗ **activation function**
activation, 3
∗ **dNNmodel**
deepAFT, 6
∗ **deepAFT**
deepAFT, 6
hyperTuning, 18
plot, 23
print, 25
∗ **deepGLM**
deepGLM, 9
plot, 23
print, 25
∗ **deepSurv**
deepSurv, 11
hyperTuning, 18
plot, 23
print, 25
∗ **dnnControl**
deepAFT, 6
∗ **dnnFit**
dnnControl, 13
dnnFit, 14
∗ **dnn**
deepGLM, 9
deepSurv, 11
∗ **optimizer AdamG**
optimizerSGD, 22
∗ **optimizer Momentum**
optimizerSGD, 22
∗ **optimizer NAG**
optimizerSGD, 22
∗ **optimizer SDG**
optimizerSGD, 22
∗ **plot**
plot, 23
∗ **predict**
predict, 24
∗ **print**
print, 25
∗ **residuals**
residuals, 26
∗ **summary**
plot, 23
print, 25
∗ **survfit**
survfit, 28

activation, 3, 22

bwdCheck *(*bwdNN*)*, 5
bwdNN, *3*, *4*, 5, *17*, *18*, *22*
bwdNN2 *(*bwdNN*)*, 5

coxph, *3*
CVpredErr (hyperTuning), 18

deepAFT, *3*, 6, *10*, *12*, *14*, *16*, *20–22*, *24–27*, *29*
deepGLM, *3*, 9, *14*, *20*, *25*
deepGlm, *12*, *16*
deepGlm (deepGLM), 9
deepSurv, *3*, 11, *14*, *16*, *20*, *25*
delu (activation), 3
didu (activation), 3
Distributions, *28*
dlrelu (activation), 3
dnn (dnn-package), 2
dnn-doc (dnn-package), 2
dnn-package, 2
dnnControl, 13, *15*, *16*, *19*
dnnFit, *13*, *14*, 14, *20*, *22*
dnnFit2 (dnnFit), 14
dNNmodel, *3*, *4*, *6*, *10*, *15*, 16, *19*, *22*, *25*
drelu (activation), 3
dsigmoid (activation), 3
dsurv (rsurv), 27
dtanh (activation), 3
dweibull, *28*

elu (activation), 3

fwdNN, *3*, *4*, *6*, *17*, 17, *22*
fwdNN2 (fwdNN), 17

glm, *3*, *10*, *23*

hyperTuning, 18

ibs, 20
ibs.deepAFT, *8*
idu (activation), 3

lrelu (activation), 3

mseIPCW (msePICW), 21
msePICW, 21

optimizerAdamG (optimizerSGD), 22
optimizerMomentum (optimizerSGD), 22
optimizerNAG, *4*, *17*
optimizerNAG (optimizerSGD), 22
optimizerSGD, *4*, *17*, 22

plot, 23
plot.default, *23*
plot.dNNmodel, *6*, *17*, *18*
predict, *22*, *24*, 24
predict.deepGlm, *10*
predict.deepGlm (deepGLM), 9
predict.dNNmodel (fwdNN), 17
predict.dSurv, *27*, *29*
print, 25
print.deepAFT, *8*
print.deepSurv, *10*, *12*
print.default, *26*
print.dNNmodel, *6*, *17*, *18*
psurv (rsurv), 27

qsurv (rsurv), 27

rcoxph (rsurv), 27
relu (activation), 3
residuals, *26*, 26, *27*
residuals.deepGlm (deepGLM), 9
rSurv (rsurv), 27
rsurv, 27

sigmoid (activation), 3
summary, *26*
summary.deepAFT (print), 25
summary.deepGlm (deepGLM), 9
summary.deepSurv (deepSurv), 11
summary.dNNmodel, *6*, *17*, *18*
summary.dNNmodel (print), 25
survfit, 28, *29*
survfit.dSurv, *22*, *24*, *27*
survival, *3*
survreg, *8*, *12*, *26*