Package 'monmlp'

March 23, 2017

Type Package

Title Monotone Multi-Layer Perceptron Neural Network

Version 1.1.4

Author Alex J. Cannon

Maintainer Alex J. Cannon <alex.cannon@canada.ca>

Description Train and make predictions from a multi-layer perceptron neural network with optional partial monotonicity constraints.

License GPL-2

LazyLoad yes

Depends optimx

Repository CRAN

NeedsCompilation no

Date/Publication 2017-03-23 21:05:06 UTC

R topics documented:

monmlp-package *Monotone Multi-Layer Perceptron Neural Network*

Description

The monmlp package implements the monotone multi-layer perceptron neural network (MONMLP) regression model following Zhang and Zhang (1999). The main feature is the monotone constraint, which guarantees monotonically increasing behaviour of model outputs with respect to specified covariates. The package also features model architectures with one or two hidden layers, analytical calculation of the gradient via backpropagation, and optional use of early stopping in conjunction with bootstrap aggregation to control overfitting. The model reduces to a standard multi-layer perceptron neural network if the monotone constraint is not invoked.

MONMLP models are fit using the `monmlp.fit` function. Predictions from a fitted model are made using the `monmlp.predict` function. The `gam.style` function can be used to investigate fitted predictor/predictand relationships.

Details

Package:	monmlp
Type:	Package
License:	GPL-2
LazyLoad:	yes

Author(s)

Alex J. Cannon

Maintainer: Alex J. Cannon <acannon@eos.ubc.ca>

References

Lang, B., 2005. Monotonic multi-layer perceptron networks as universal approximators. In: W. Duch et al. (eds.): ICANN 2005, Lecture Notes in Computer Science, 3697:31-37. doi:10.1007/11550907

Minin, A., Velikova, M., Lang, B., and Daniels, H., 2010. Comparison of universal approximators incorporating partial monotonicity by structure. Neural Networks, 23:471-475. doi:10.1016/j.neunet.2009.09.002

Zhang, H. and Zhang, Z., 1999. Feedforward networks with monotone constraints. In: International Joint Conference on Neural Networks, vol. 3, p. 1820-1823. doi:10.1109/IJCNN.1999.832655

gam.style *GAM-style effects plots for interpreting MONMLP models*

Description

GAM-style effects plots provide a graphical means of interpreting fitted MONMLP predictor/predictor relationships. From Plate et al. (2000): The effect of the ith input variable at a particular input point Delta.i.x is the change in f resulting from changing X1 to x1 from b1 (the baseline value [...]) while keeping the other inputs constant. The effects are plotted as short line segments, centered at (x.i, Delta.i.x), where the slope of the segment is given by the partial derivative. Variables that strongly influence the function value have a large total vertical range of effects. Functions without interactions appear as possibly broken straight lines (linear functions) or curves (nonlinear functions). Interactions show up as vertical spread at a particular horizontal location, that is, a vertical scattering of segments. Interactions are present when the effect of a variable depends on the values of other variables.

Usage

```
gam.style(x, weights, column, baseline = mean(x[,column]),
         epsilon = 1e-5, seg.len = 0.02, seg.cols = "black",
         plot = TRUE, return.results = FALSE, ...)
```

Arguments

x	matrix with number of rows equal to the number of samples and number of columns equal to the number of predictor variables.
weights	list returned by monmlp.fit.
column	column of x for which effects plots should be returned.
baseline	value of x[,column] to be used as the baseline for calculation of predictor effects; defaults to mean(x[,column]).
epsilon	step-size used in the finite difference calculation of the partial derivatives.
seg.len	length of effects line segments expressed as a fraction of the range of x[,column].
seg.cols	colors of effects line segments.
plot	if TRUE (the default) then an effects plots for each predictand variable is produced.
return.results	if TRUE then values of effects and partial derivatives for each predictand variable are returned.
...	further arguments to be passed to plot.

Value

A list with elements:

effects	a matrix of predictor effects.
partials	a matrix of predictor partial derivatives.

References

Cannon, A.J. and I.G. McKendry, 2002. A graphical sensitivity analysis for interpreting statistical climate models: Application to Indian monsoon rainfall prediction by artificial neural networks and multiple linear regression models. International Journal of Climatology, 22:1687-1708.

Plate, T., J. Bert, J. Grace, and P. Band, 2000. Visualizing the function computed by a feedforward neural network. Neural Computation, 12(6): 1337-1354.

See Also

monmlp.fit, monmlp.predict

Examples

```
set.seed(1)
x <- matrix(runif(350*6), ncol=6)
y <- as.matrix(5*sin(10*x[,1]*x[,2]) + 20*(x[,3]-0.5)^2 -
               10*x[,4] + 20*x[,5]*x[,6])

w <- monmlp.fit(x = x, y = y, hidden1 = 4, n.trials = 1,
                iter.max = 500)

for (i in seq(ncol(x))) gam.style(x, weights = w, column = i)
```

---

linear *Identity function*

Description

Computes a trivial identity function. Used as the hidden layer transfer function for linear MONMLP models.

Usage

```
linear(x)
```

Arguments

x	numeric vector.

See Also

linear.prime

linear.prime *Derivative of the linear function*

Description

Derivative of the linear function.

Usage

```
linear.prime(x)
```

Arguments

x	numeric vector.

See Also

linear

logistic *Logistic sigmoid function*

Description

Computes the logistic sigmoid function. Used as a hidden layer transfer function for nonlinear MONMLP models.

Usage

```
logistic(x)
```

Arguments

x	numeric vector.

See Also

logistic.prime

logistic.prime *Derivative of the logistic sigmoid function*

Description

Derivative of the logistic sigmoid function.

Usage

```
logistic.prime(x)
```

Arguments

x	numeric vector.

See Also

logistic

monmlp.fit *Fit a MONMLP model or an ensemble of MONMLP models*

Description

Fit an individual model or ensemble of MONMLP regression models using optimx optimization routines to minimize a least squares cost function. Optional stopped training and bootstrap aggregation (bagging) can be used to help avoid overfitting.

If invoked, the monotone argument enforces increasing behaviour between specified columns of x and model outputs. In this case, the exp function is applied to the relevant weights following initialization and during optimization; manual adjustment of init.weights may be needed.

Note: x and y are automatically standardized prior to fitting and predictions are automatically rescaled by monmlp.predict. This behaviour can be suppressed for y by the scale.y argument.

Usage

```
monmlp.fit(x, y, hidden1, hidden2 = 0, iter.max = 5000,
           n.trials = 1, n.ensemble = 1, bag = FALSE,
           cases.specified = NULL, iter.stopped = NULL,
           scale.y = TRUE, Th = tansig, To = linear,
           Th.prime = tansig.prime, To.prime = linear.prime,
           monotone = NULL, init.weights = NULL,
           max.exceptions = 10, silent = FALSE, method = "BFGS",
           control = list(trace = 0))
```

Arguments

x	covariate matrix with number of rows equal to the number of samples and number of columns equal to the number of covariates.
y	predictand matrix with number of rows equal to the number of samples and number of columns equal to the number of predictands.
hidden1	number of hidden nodes in the first hidden layer.
hidden2	number of hidden nodes in the second hidden layer.
iter.max	maximum number of iterations of the optimization algorithm.
n.trials	number of repeated trials used to avoid local minima.
n.ensemble	number of ensemble members to fit.
bag	logical variable indicating whether or not bootstrap aggregation (bagging) should be used.
cases.specified	if bag = TRUE, a list that specifies the bootstrapped cases to be used in each ensemble member.
iter.stopped	if bag = TRUE, specifies the number of stopped training iterations between calculation of the cost function on the out-of-bootstrap cases.
scale.y	logical determining if columns of the predictand matrix should be scaled to zero mean and unit variance prior to fitting. Set this to FALSE if using an output layer transfer function that limits the range of predictions.
Th	hidden layer transfer function.
To	output layer transfer function.
Th.prime	derivative of the hidden layer transfer function.
To.prime	derivative of the output layer transfer function.
monotone	column indices of covariates for which the monotonicity constraint should hold.
init.weights	either a vector giving the minimum and maximum allowable values of the random weights, an initial weight vector, or NULL to calculate based on fan-in.
max.exceptions	maximum number of exceptions of the optimization routine before fitting is terminated with an error.
silent	logical determining if diagnostic messages should be suppressed.
method	optimx optimization method.
control	list of optimx control parameters.

Value

list containing fitted weight matrices with attributes including called values of x, y, Th, To, Th.prime, To.prime, monotone, bag, iter.max, and iter.stopped, along with values of covariate/predictand column means and standard deviations (x.center, x.scale, y.center, y.scale), out-of-bootstrap cases oob, predicted values y.pred, and, if stopped training is switched on, the iteration iter.best and value of the cost function cost.best that minimized the out-of-bootstrap validation error.

See Also

monmlp.predict, gam.style

Examples

```
set.seed(123)
x <- as.matrix(seq(-10, 10, length = 100))
y <- logistic(x) + rnorm(100, sd = 0.2)

dev.new()
plot(x, y)
lines(x, logistic(x), lwd = 10, col = "gray")

## MLP w/ 2 hidden nodes
w.mlp <- monmlp.fit(x = x, y = y, hidden1 = 2, iter.max = 500)
lines(x, attr(w.mlp, "y.pred"), col = "red", lwd = 3)

## MLP w/ 2 hidden nodes and stopped training
w.stp <- monmlp.fit(x = x, y = y, hidden1 = 2, bag = TRUE,
                    iter.max = 500, iter.stopped = 10)
lines(x, attr(w.stp, "y.pred"), col = "orange", lwd = 3)

## MONMLP w/ 2 hidden nodes
w.mon <- monmlp.fit(x = x, y = y, hidden1 = 2, monotone = 1,
                    iter.max = 500)
lines(x, attr(w.mon, "y.pred"), col = "blue", lwd = 3)
```

---

monmlp.predict *Make predictions from a fitted MONMLP model*

Description

Make predictions from a fitted MONMLP model or ensemble of MONMLP models.

Usage

```
monmlp.predict(x, weights)
```

Arguments

x	covariate matrix with number of rows equal to the number of samples and number of columns equal to the number of covariates.
weights	list containing MONMLP weight matrices and other parameters from monmlp.fit.

Value

a matrix with number of rows equal to the number of samples and number of columns equal to the number of predictand variables. If weights is from an ensemble of models, the matrix is the ensemble mean and the attribute ensemble contains a list with predictions for each ensemble member.

See Also

monmlp.fit

Examples

```
set.seed(123)
x <- as.matrix(seq(-10, 10, length = 100))
y <- logistic(x) + rnorm(100, sd = 0.2)

dev.new()
plot(x, y)
lines(x, logistic(x), lwd = 10, col = "gray")

## Ensemble of MONMLP models w/ 3 hidden nodes
w.mon <- monmlp.fit(x = x, y = y, hidden1 = 3, monotone = 1,
                    n.ensemble = 15, bag = TRUE, iter.max = 500,
                    control = list(trace = 0))
p.mon <- monmlp.predict(x = x, weights = w.mon)

## Plot predictions from ensemble members
matlines(x = x, y = do.call(cbind, attr(p.mon, "ensemble")),
         col = "cyan", lty = 2)

## Plot ensemble mean
lines(x, p.mon, col = "blue", lwd = 3)
```

---

tansig *Hyperbolic tangent sigmoid function*

Description

Computes the hyperbolic tangent sigmoid function. Used as a hidden layer transfer function for nonlinear MONMLP models.

Usage

```
tansig(x)
```

Arguments

x	numeric vector.

See Also

tansig.prime

tansig.prime *Derivative of the hyperbolic tangent function*

Description

Derivative of the hyperbolic tangent function.

Usage

```
tansig.prime(x)
```

Arguments

x	numeric vector.

See Also

tansig

Index