Package: ALEPlot (via r-universe)

August 20, 2024

Type Package

Title Accumulated Local Effects (ALE) Plots and Partial Dependence (PD) Plots

Version 1.1

Date 2018-05-22

Author Dan Apley

Maintainer Dan Apley <apley@northwestern.edu>

Description Visualizes the main effects of individual predictor variables and their second-order interaction effects in black-box supervised learning models. The package creates either Accumulated Local Effects (ALE) plots and/or Partial Dependence (PD) plots, given a fitted supervised learning model.

Imports yaImpute

License GPL-2

NeedsCompilation no

Suggests R.rsp, nnet

VignetteBuilder R.rsp

RoxygenNote 6.0.1

Date/Publication 2018-05-24 16:14:07 UTC

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

RemoteUrl https://github.com/cran/ALEPlot

RemoteRef HEAD

RemoteSha d283def19a5d7f1840f18dfca82c073210df0d81

Contents

ALEPlot-package *Accumulated Local Effects (ALE) Plots and Partial Dependence (PD) Plots*

Description

Visualizes the main effects of individual predictor variables and their second-order interaction effects in black-box supervised learning models. The package creates either Accumulated Local Effects (ALE) plots and/or Partial Dependence (PD) plots, given a fitted supervised learning model.

Details

See the two individual functions `ALEPlot` and `PDPlot` that are included in this package.

Author(s)

Dan Apley

Maintainer: Dan Apley <apley@northwestern.edu>

References

Apley, D. W. (2016), "Visualizing the Effects of Predictor Variables in Black Box Supervised Learning Models," submitted for publication.

ALEPlot *Accumulated Local Effects (ALE) Plots*

Description

Computes and plots accumulated local effects (ALE) plots for a fitted supervised learning model. The effects can be either a main effect for an individual predictor (`length(J) = 1`) or a second-order interaction effect for a pair of predictors (`length(J) = 2`).

Usage

```
ALEPlot(X, X.model, pred.fun, J, K = 40, NA.plot = TRUE)
```

Arguments

`X`	The data frame of predictor variables to which the supervised learning model was fit. The names of the predictor variables must be the same as when the model was fit. The response variable should not be included in `X`.
`X.model`	The fitted supervised learning model object (e.g., a tree, random forest, neural network, etc.), typically an object to which a built-in `predict` command associated with that object can be applied.

pred.fun — A user-supplied function that will be used to predict the response for X.model for some specified inputs. pred.fun has two arguments. The first argument is named X.model and must be the same object as the X.model argument to the ALEPlot function. The second argument is named newdata and is a data frame of predictor values at which the object X.model is to be predicted. The output of pred.fun must be a numeric vector of predictions having length equal to the number of rows of newdata. For most X.model objects, pred.fun can simply call the predict function that was written as part of that modeling object package, assuming the package contains a predict function. An example of where a more customized pred.fun would be used is a multi (> 2) class classification problem for which the built-in predict function returns a vector of predicted probabilities, one for each response class. In this case it may make sense to have pred.fun return the predicted probabilities (or its log-odds, etc.) for one particular class of interest.

J — A numeric scalar or two-length vector of indices of the predictors for which the ALE plot will be calculated. J is either a single index (for a main effects plot) or a pair of indices (for a second-order interaction plot). For a single index, the corresponding predictor must be either numeric or a factor. For a pair of indices, the corresponding predictors must be either both numeric or the first a factor and the second numeric.

K — A numeric scalar that specifies the number of intervals into which the predictor range is divided when calculating the ALE plot effects. If length(J) = 2, the same K will be used for both predictors, resulting in an array of K^2 cells over the two-dimensional predictor space. Note that the algorithm may adjust (reduce) K internally if the predictors are discrete and have many repeated values. K is only used if the predictor is numeric. For factor predictors, the equivalent of K is the number of used levels of the factor, which is automatically determined internally.

NA.plot — A logical value that is only used if length(J) = 2. If NA.plot = TRUE (the default), the ALE second-order effects are also plotted for empty cells. Empty cells are defined as cells in the (X[,J[1]], X[,J[2]]) space into which no training observations fall. If NA.plot = FALSE, the accumulated local second-order effects are only plotted for non-empty cells, and black rectangles are plotted over any empty cells to indicate their locations. Either way, when accumulating the local second-order effects, values are need for the empty cells, and these values are taken to be the local second-order effects for the nearest-neighbor non-empty cell.

Details

See the Apley (2016) reference paper listed below for details. For J = j (i.e., if the index for a single predictor x_j is specified), the function calculates and returns the ALE main effect of x_j, which is denoted by $f_{j,ALE}(x_j)$ in Apley (2016). It also plots $f_{j,ALE}(x_j)$. For J = c(j1,j2) (i.e., if the indices for a pair of predictors (x_{j1}, x_{j2}) are specified), the function calculates and returns the ALE second-order interaction effect of (x_{j1}, x_{j2}), which is denoted by $f_{j1,j2,ALE}(x_{j1}, x_{j2})$ in Apley (2016). It also plots $f_{j1,j2,ALE}(x_{j1}, x_{j2})$.

Value

K	The same as the input argument K, but possibly adjusted internally. For numeric predictors, K is the number of intervals into which the range of each predictor is divided. If the predictor is discrete with many repeated values, K can be reduced internally, as mentioned above. For length(J)=1, K is an integer. For length(J)=2, K = c(K1, K2), where K1 and K2 are the numbers of intervals for the X[,J(1)] and X[,J(2)] ranges, respectively. For factor predictors, K is the number of non-empty levels, which is calculated internally.
f.values	If length(J) = 1, a vector of ALE plot function values at the predictor values in x.values. If length(J) = 2, f.values is a matrix of ALE plot function values at the grid of values defined by the X[,J(1)] and X[,J(2)] values in x.values. The rows of f.values correspond to X[,J(1)], and the columns to X[,J(2)].
x.values	For numeric predictors, if length(J) = 1, a (K+1)-length vector specifying the ordered predictor values at which the ALE plot function is calculated. These are the break points for the K intervals into which the predictor range is divided, plus the lower boundary of the first interval and the upper boundary of the last interval. If length(J) = 2, a list of two such vectors, the first containing the X[,J(1)] values and the second containing the X[,J(2)] values at which the ALE plot function is calculated. x.values is the same for factor predictors, except it is a K-length character vector containing the ordered levels of the predictor (the ordering is determined internally, based on the similarity of the predictor in question to the other predictors), where K is the number of used levels. The elements of f.values are ordered accordingly.

Author(s)

Dan Apley

References

Apley, D. W. (2016), "Visualizing the Effects of Predictor Variables in Black Box Supervised Learning Models," submitted for publication.

See Also

See PDPlot for partial dependence plots.

Examples

```
#########################################################################
## A transparent example in which the supervised learning model is a linear regression \code{lm},
## but we will pretend it is black-box
#########################################################################

## Generate some data and fit a \code{lm} supervised learning model
N=500
x1 <- runif(N, min=0, max=1)
x2 <- runif(N, min=0, max=1)
x3 <- runif(N, min=0, max=1)
```

```
y = x1 + 2*x2^2 + rnorm(N, 0, 0.1)
DAT = data.frame(y, x1, x2, x3)
lm.DAT = lm(y ~ .^2 + I(x1^2) + I(x2^2) + I(x3^2), DAT)

## Define the predictive function (easy in this case, since \code{lm} has a built-in
## predict function that suffices)
yhat <- function(X.model, newdata) as.numeric(predict(X.model, newdata))

## Calculate and plot the ALE main and second-order interaction effects of x1, x2, x3
par(mfrow = c(2,3))
ALE.1=ALEPlot(DAT[,2:4], lm.DAT, pred.fun=yhat, J=1, K=50, NA.plot = TRUE)
ALE.2=ALEPlot(DAT[,2:4], lm.DAT, pred.fun=yhat, J=2, K=50, NA.plot = TRUE)
ALE.3=ALEPlot(DAT[,2:4], lm.DAT, pred.fun=yhat, J=3, K=50, NA.plot = TRUE)
ALE.12=ALEPlot(DAT[,2:4], lm.DAT, pred.fun=yhat, J=c(1,2), K=20, NA.plot = TRUE)
ALE.13=ALEPlot(DAT[,2:4], lm.DAT, pred.fun=yhat, J=c(1,3), K=20, NA.plot = TRUE)
ALE.23=ALEPlot(DAT[,2:4], lm.DAT, pred.fun=yhat, J=c(2,3), K=20, NA.plot = TRUE)

## The following manually recreates the same plots produced by the above ALEPlot function calls
par(mfrow = c(2,3))
plot(ALE.1$x.values, ALE.1$f.values, type="l", xlab="x1", ylab="ALE main effect for x1")
plot(ALE.2$x.values, ALE.2$f.values, type="l", xlab="x2", ylab="ALE main effect for x2")
plot(ALE.3$x.values, ALE.3$f.values, type="l", xlab="x3", ylab="ALE main effect for x3")
image(ALE.12$x.values[[1]], ALE.12$x.values[[2]], ALE.12$f.values, xlab = "x1", ylab = "x2")
contour(ALE.12$x.values[[1]], ALE.12$x.values[[2]], ALE.12$f.values, add=TRUE, drawlabels=TRUE)
image(ALE.13$x.values[[1]], ALE.13$x.values[[2]], ALE.13$f.values, xlab = "x1", ylab = "x3")
contour(ALE.13$x.values[[1]], ALE.13$x.values[[2]], ALE.13$f.values, add=TRUE, drawlabels=TRUE)
image(ALE.23$x.values[[1]], ALE.23$x.values[[2]], ALE.23$f.values, xlab = "x2", ylab = "x3")
contour(ALE.23$x.values[[1]], ALE.23$x.values[[2]], ALE.23$f.values, add=TRUE, drawlabels=TRUE)



#########################################################################
## A larger example in which the supervised learning model is a neural network (\code{nnet})
#########################################################################

## Generate some data and fit a \code{nnet} supervised learning model

library(nnet)
N=5000
x1 <- runif(N, min=0, max=1)
x2 <- runif(N, min=0, max=1)
x3 <- runif(N, min=0, max=1)
y = x1 + 2*x2^2 +(x1-0.5)*(x3-0.5) + rnorm(N, 0, 0.1)
DAT = data.frame(y, x1, x2, x3)
nnet.DAT<-nnet(y~., data=DAT, linout=TRUE, skip=FALSE, size=10, decay=0.01,
 maxit=1000, trace=FALSE)

## Define the predictive function
yhat <- function(X.model, newdata) as.numeric(predict(X.model, newdata, type="raw"))

## Calculate and plot the ALE main and second-order interaction effects of x1, x2, x3
par(mfrow = c(2,3))
ALE.1=ALEPlot(DAT[,2:4], nnet.DAT, pred.fun=yhat, J=1, K=50, NA.plot = TRUE)
```

```
ALE.2=ALEPlot(DAT[,2:4], nnet.DAT, pred.fun=yhat, J=2, K=50, NA.plot = TRUE)
ALE.3=ALEPlot(DAT[,2:4], nnet.DAT, pred.fun=yhat, J=3, K=50, NA.plot = TRUE)
ALE.12=ALEPlot(DAT[,2:4], nnet.DAT, pred.fun=yhat, J=c(1,2), K=20, NA.plot = TRUE)
ALE.13=ALEPlot(DAT[,2:4], nnet.DAT, pred.fun=yhat, J=c(1,3), K=20, NA.plot = TRUE)
ALE.23=ALEPlot(DAT[,2:4], nnet.DAT, pred.fun=yhat, J=c(2,3), K=20, NA.plot = TRUE)

#########################################################################
## A binary classification example in which the supervised learning model is
## a neural network (\code{nnet}), and the log-odds of the predicted class probability
## is the function to be plotted
#########################################################################

## Generate some data and fit a \code{nnet} supervised learning model

library(nnet)
N=5000
x1 <- runif(N, min=0, max=1)
x2 <- runif(N, min=0, max=1)
x3 <- runif(N, min=0, max=1)
z = -3.21 + 2.81*x1 + 5.62*x2^2 + 2.81*(x1-0.5)*(x3-0.5) #true log-odds
p = exp(z)/(1+exp(z))
u = runif(N)
y = u < p
DAT = data.frame(y, x1, x2, x3)
nnet.DAT<-nnet(y~., data=DAT, linout=FALSE, skip=FALSE, size=10, decay=0.05,
 maxit=1000, trace=FALSE)

## Define the ALE function to be the log-odds of the predicted probability that y = TRUE
yhat <- function(X.model, newdata) {
    p.hat = as.numeric(predict(X.model, newdata, type="raw"))
    log(p.hat/(1-p.hat))
}

## Calculate and plot the ALE main and second-order interaction effects of x1, x2, x3
par(mfrow = c(2,3))
ALE.1=ALEPlot(DAT[,2:4], nnet.DAT, pred.fun=yhat, J=1, K=50, NA.plot = TRUE)
ALE.2=ALEPlot(DAT[,2:4], nnet.DAT, pred.fun=yhat, J=2, K=50, NA.plot = TRUE)
ALE.3=ALEPlot(DAT[,2:4], nnet.DAT, pred.fun=yhat, J=3, K=50, NA.plot = TRUE)
ALE.12=ALEPlot(DAT[,2:4], nnet.DAT, pred.fun=yhat, J=c(1,2), K=20, NA.plot = TRUE)
ALE.13=ALEPlot(DAT[,2:4], nnet.DAT, pred.fun=yhat, J=c(1,3), K=20, NA.plot = TRUE)
ALE.23=ALEPlot(DAT[,2:4], nnet.DAT, pred.fun=yhat, J=c(2,3), K=20, NA.plot = TRUE)
```

PDPlot *Partial Dependence (PD) Plots*

Description

Computes and plots partial dependence (PD) plots for a fitted supervised learning model. The effects can be either a main effect for an individual predictor (`length(J) = 1`) or a second-order

interaction effect for a pair of predictors (`length(J) = 2`).

Usage

```
PDPlot(X, X.model, pred.fun, J, K)
```

Arguments

`X`	The data frame of predictor variables to which the supervised learning model was fit. The names of the predictor variables must be the same as when the model was fit. The response variable should not be included in `X`.
`X.model`	The fitted supervised learning model object (e.g., a tree, random forest, neural network, etc.), typically an object to which a built-in `predict` command associated with that object can be applied.
`pred.fun`	A user-supplied function that will be used to predict the response for `X.model` for some specified inputs. `pred.fun` has two arguments. The first argument is named `X.model` and must be the same object as the `X.model` argument to the `ALEPlot` function. The second argument is named `newdata` and is a data frame of predictor values at which the object `X.model` is to be predicted. The output of `pred.fun` must be a numeric vector of predictions having length equal to the number of rows of `newdata`. For most `X.model` objects, `pred.fun` can simply call the `predict` function that was written as part of that modeling object package, assuming the package contains a `predict` function. An example of where a more customized `pred.fun` would be used is a multi (> 2) class classification problem for which the built-in predict function returns a vector of predicted probabilities, one for each response class. In this case it may make sense to have `pred.fun` return the predicted probabilities (or its log-odds, etc.) for one particular class of interest.
`J`	A numeric scalar or two-length vector of indices of the predictors for which the PD plot will be calculated. `J` is either a single index (for a main effects plot) or a pair of indices (for a second-order interaction plot). For a single index, the corresponding predictor must be either numeric or a factor. For a pair of indices, the corresponding predictors must be either both numeric or the first a factor and the second numeric.
`K`	A numeric scalar that represents the number of discrete points at which the PD plot will be calculated. If `length(J) = 2`, the same `K` will be used for both predictors, resulting in a two-dimensional grid of `K^2` predictor values at with the PD plot will be calculated. `K` is only used if the predictor is numeric. For factor predictors, the equivalent of `K` is the number of used levels of the factor, which is automatically determined internally.

Details

This function calculates and plots the partial dependence (PD) plots first introduced in Friedman (2001). See the Apley (2016) reference paper listed below for details. For `J = j` (i.e., if the index for a single predictor x_j is specified), the function calculates and returns the PD main effect of x_j, which is denoted by $f_{j,PD}(x_j)$ in Apley (2016). It also plots $f_{j,PD}(x_j)$. For `J = c(j1,j2)` (i.e., if the indices for a pair of predictors (x_{j1}, x_{j2}) are specified), the function calculates and returns the

PD second-order interaction effect of (x_{j1}, x_{j2}), which is denoted by $f_{j1,j2,PD}(x_{j1}, x_{j2})$ in Apley (2016). It also plots $f_{j1,j2,PD}(x_{j1}, x_{j2})$.

Value

`f.values` If `length(J) = 1`, a vector of PD plot function values at the predictor values in `x.values`. If `length(J) = 2`, `f.values` is a `K1xK` matrix of the PD plot function values at the grid of predictor values defined by the `X[,J[1]]` and `X[,J[2]]` values in `x.values`. For `X[,J[1]]` numeric, `K1 = K`. For `X[,J[2]]` a factor, `K1` is the number of used levels (empty levels are dropped). The rows of `f.values` correspond to `X[,J(1)]`, and the columns to `X[,J(2)]`.

`x.values` For numeric predictors, if `length(J) = 1`, a `K`-length vector specifying the ordered predictor values at which the PD plot function is calculated. If `length(J) = 2`, a list of two such vectors, the first containing the `X[,J(1)]` values and the second containing the `X[,J(2)]` values at which the PD plot function is calculated. `x.values` is the same for factor predictors, except it is a `K1`-length character vector of the levels of the predictor, where `K1` is determined internally as the number of unique levels of the predictor (empty levels are dropped).

Author(s)

Dan Apley

References

Friedman, J. H., (2001), "Greedy function approximation: A gradient boosting machine," Annals of Statistics, 29(5), pp. 1189-1232.

Apley, D. W. (2016), "Visualizing the Effects of Predictor Variables in Black Box Supervised Learning Models," submitted for publication.

See Also

See `ALEPlot` for partial dependence plots.

Examples

```
##########################################################################
## A transparent example in which the supervised learning model is a linear regression \code{lm},
## but we will pretend it is black-box
##########################################################################

## Generate some data and fit a \code{lm} supervised learning model
N=500
x1 <- runif(N, min=0, max=1)
x2 <- runif(N, min=0, max=1)
x3 <- runif(N, min=0, max=1)
y = x1 + 2*x2^2 + rnorm(N, 0, 0.1)
DAT = data.frame(y, x1, x2, x3)
lm.DAT = lm(y ~ .^2 + I(x1^2) + I(x2^2) + I(x3^2), DAT)
```

```
## Define the predictive function (easy in this case, since \code{lm} has
## a built-in predict function that suffices)
yhat <- function(X.model, newdata) as.numeric(predict(X.model, newdata))

## Calculate and plot the PD main effects and second-order interaction effects of x1, x2, x3
par(mfrow = c(2,3))
PD.1=PDPlot(DAT[,2:4], lm.DAT, pred.fun=yhat, J=1, K=50)
PD.2=PDPlot(DAT[,2:4], lm.DAT, pred.fun=yhat, J=2, K=50)
PD.3=PDPlot(DAT[,2:4], lm.DAT, pred.fun=yhat, J=3, K=50)
PD.12=PDPlot(DAT[,2:4], lm.DAT, pred.fun=yhat, J=c(1,2), K=30)
PD.13=PDPlot(DAT[,2:4], lm.DAT, pred.fun=yhat, J=c(1,3), K=30)
PD.23=PDPlot(DAT[,2:4], lm.DAT, pred.fun=yhat, J=c(2,3), K=30)

## The following manually recreates the same plots produced by the above PDPlot function calls
par(mfrow = c(2,3))
plot(PD.1$x.values, PD.1$f.values, type="l", xlab="x1", ylab="PD main effect for x1")
plot(PD.2$x.values, PD.2$f.values, type="l", xlab="x2", ylab="PD main effect for x2")
plot(PD.3$x.values, PD.3$f.values, type="l", xlab="x3", ylab="PD main effect for x3")
image(PD.12$x.values[[1]], PD.12$x.values[[2]], PD.12$f.values, xlab = "x1", ylab = "x2")
contour(PD.12$x.values[[1]], PD.12$x.values[[2]], PD.12$f.values, add=TRUE, drawlabels=TRUE)
image(PD.13$x.values[[1]], PD.13$x.values[[2]], PD.13$f.values, xlab = "x1", ylab = "x3")
contour(PD.13$x.values[[1]], PD.13$x.values[[2]], PD.13$f.values, add=TRUE, drawlabels=TRUE)
image(PD.23$x.values[[1]], PD.23$x.values[[2]], PD.23$f.values, xlab = "x2", ylab = "x3")
contour(PD.23$x.values[[1]], PD.23$x.values[[2]], PD.23$f.values, add=TRUE, drawlabels=TRUE)


#########################################################################
## A larger example in which the supervised learning model is a neural network (\code{nnet})
#########################################################################

## Generate some data and fit a \code{nnet} supervised learning model

library(nnet)
N=5000
x1 <- runif(N, min=0, max=1)
x2 <- runif(N, min=0, max=1)
x3 <- runif(N, min=0, max=1)
y = x1 + 2*x2^2 +(x1-0.5)*(x3-0.5) + rnorm(N, 0, 0.1)
DAT = data.frame(y, x1, x2, x3)
nnet.DAT<-nnet(y~., data=DAT, linout=TRUE, skip=FALSE, size=10, decay=0.01,
 maxit=1000, trace=FALSE)

## Define the predictive function
yhat <- function(X.model, newdata) as.numeric(predict(X.model, newdata, type="raw"))

## Calculate and plot the PD main and second-order interaction effects of x1, x2, x3
par(mfrow = c(2,3))
PD.1=PDPlot(DAT[,2:4], nnet.DAT, pred.fun=yhat, J=1, K=50)
PD.2=PDPlot(DAT[,2:4], nnet.DAT, pred.fun=yhat, J=2, K=50)
PD.3=PDPlot(DAT[,2:4], nnet.DAT, pred.fun=yhat, J=3, K=50)
PD.12=PDPlot(DAT[,2:4], nnet.DAT, pred.fun=yhat, J=c(1,2), K=20)
PD.13=PDPlot(DAT[,2:4], nnet.DAT, pred.fun=yhat, J=c(1,3), K=20)
PD.23=PDPlot(DAT[,2:4], nnet.DAT, pred.fun=yhat, J=c(2,3), K=20)
```

```
#########################################################################
## A binary classification example in which the supervised learning model is
## a neural network (\code{nnet}), and the log-odds of the predicted class
## probability is the function to be plotted
#########################################################################

## Generate some data and fit a \code{nnet} supervised learning model

library(nnet)
N=5000
x1 <- runif(N, min=0, max=1)
x2 <- runif(N, min=0, max=1)
x3 <- runif(N, min=0, max=1)
z = -3.21 + 2.81*x1 + 5.62*x2^2 + 2.81*(x1-0.5)*(x3-0.5) #true log-odds
p = exp(z)/(1+exp(z))
u = runif(N)
y = u < p
DAT = data.frame(y, x1, x2, x3)
nnet.DAT<-nnet(y~., data=DAT, linout=FALSE, skip=FALSE, size=10, decay=0.05,
 maxit=1000, trace=FALSE)

## Define the ALE function to be the log-odds of the predicted probability that y = TRUE
yhat <- function(X.model, newdata) {
    p.hat = as.numeric(predict(X.model, newdata, type="raw"))
    log(p.hat/(1-p.hat))
}

## Calculate and plot the PD main and second-order interaction effects of x1, x2, x3
par(mfrow = c(2,3))
PD.1=PDPlot(DAT[,2:4], nnet.DAT, pred.fun=yhat, J=1, K=50)
PD.2=PDPlot(DAT[,2:4], nnet.DAT, pred.fun=yhat, J=2, K=50)
PD.3=PDPlot(DAT[,2:4], nnet.DAT, pred.fun=yhat, J=3, K=50)
PD.12=PDPlot(DAT[,2:4], nnet.DAT, pred.fun=yhat, J=c(1,2), K=20)
PD.13=PDPlot(DAT[,2:4], nnet.DAT, pred.fun=yhat, J=c(1,3), K=20)
PD.23=PDPlot(DAT[,2:4], nnet.DAT, pred.fun=yhat, J=c(2,3), K=20)
```

Index