Package ‘btml’

May 19, 2025

Type Package

Title Bayesian Treed Machine Learning for Personalized Prediction and Precision Diagnostics

Version 0.1.0

Date 2025-05-12

Description Generalization of the Bayesian classification and regression tree (CART) model that partitions subjects into terminal nodes and tailors machine learning model to each terminal node.

License GPL (>= 2)

Depends R (>= 4.5.0), glmnet, randomForest, e1071, pROC, stats, graphics

NeedsCompilation no

Author Yunro Chung [aut, cre] (ORCID: <`https://orcid.org/0000-0001-9125-9277`>), Yaliang Zhang [aut]

Maintainer Yunro Chung <`yunro.chung@asu.edu`>

Repository CRAN

Date/Publication 2025-05-19 08:50:02 UTC

Contents

`btml` *Bayeisan Treed Machine Learning*

Description

The treed model generalizes the Bayesian classification and regression tree (BCART) model by partitioning subjects into terminal nodes and tailoring machine learning model to each terminal node.

Usage

```
btml(y,x,z,ynew,xnew,znew,MLlist,sparse,nwarm,niter,minsample,base,power)
```

Arguments

`y` Response vector. If a factor codied as 0 or 1, classification is assumed. Otherwise, regression is assumed.

`x` Data.frame or matrix of predictors that is used to estimate a tree structure.

`z` Data.frame or matrix of predictors that is used in terminal node specific ML models. See the description below about the difference between x and z.

`ynew` Response vector for the test set corresponding to y (default ynew=NULL).

`xnew` Data.frame or matrix for the test set corresponding to x (default xnew=NULL).

`znew` Data.frame or matrix for the test set corresponding to z (default znew=NULL).

`MLlist` Candidate ML models that can be assigned to each terminal node (default MLlist=c("lasso","rf","svm")). Any other ML models can be included. See the details below.

`sparse` Whether to perform variable and machine learning model selections based on a sparse Dirichlet prior rather than simply uniform (default sparse=TRUE).

`nwarm` Number of warm-up (default nwarm=1000).

`niter` Number of iteration (defaut niter=1000).

`minsample` The number of minimum sample size per each node, i.e., length(y)>min_sample if y is continuous and min(length(y==1),length(y==0))>min_sample (default min_sample=20).

`base` Base parameter for tree prior (default base=0.95).

`power` Power parameter for tree prior (default power=0.8).

Details

This treed model uses stochastic search to find the optimal decision-tree based rule that partitions subjects into distinct terminal nodes and assigns the most effective ML model to each terminal node. For high-dimensional variables, increase nwarm=10000 and niter=10000, or more; and increase minsample.

Ideally, there are two sets of predictors, x and z, e.g., demographic variables and biomarkers, where x is used to split trees, and z is assigned to each terminal node. However, if this is not possible, it allows us to use the same x and z in the btml function, e.g., btml(y=y, x=x, z=x, ...).

Regarding the node numbers, an internal node s has left and right child nodes 2*s and 2*s+1, respectively, where node 1 is a root node; nodes 2 and 3 are left and right child nodes of node 1; nodes 4 and 5 are left and right nodes of node 2; and so on.

Currently, lasso(), randomForest(), and svm(...,kernel="radial") functions from R packages cv.glmnet, randomForest, and e1071 are supported, but any ML models can be flexibly added to terminal nodes, e.g., see the example #3 below.

Value

An object of class btml, which is a list with the following components:

`terminal` Node numbers in terminal nodes.

`internal` Node numbers in internal nodes.

`splitVariable` Variable (i.e., x[,u] if splitVariable[k]=u) used to split the internal node k.

`cutoff` cutoff[k] is the cutoff value to split the internal node k.

`selML` ML model assigned to the terminal node t.

`fitML` fitML[[t]] is the fitted ML model at the terminal node t ∈ terminal.

`y.hat` Estimated y (or estimated probability) on the training set if y is continuous (or binary).

`node.hat` Estimated node on the training set.

`mse` Training MSE.

`bs` Training Brier Score.

`roc` Training ROC curve.

`auc` Training AUC.

`y.hat.new` Estimated y (or estimated probability) on the test set if y is continuous (or binary).

`node.hat.new` Estimated node on the test set.

`mse.new` Test MSE.

`bs.new` Test Brier Score.

`roc.new` Test ROC curve.

`auc.new` Test AUC.

Author(s)

Yaliang Zhang [aut], Yunro Chung [aut, cre]

References

Yaliang Zhang and Yunro Chung, Bayesian treed machine learning model (in preperation)

Examples

```
set.seed(123)
###
#1. continuous y
###
n=200*2 #n=200 & 200 for training & test sets

x=matrix(rnorm(n*10),n,10) #10 predictors
z=matrix(rnorm(n*10),n,10) #10 biomarkers

xcut=median(x[,1])
subgr=1*(x[,1]<xcut)+2*(x[,1]>=xcut) #2 subgroups
```

```
lp=rep(NA,n)
for(i in 1:n)
  lp[i]=1+3*z[i,subgr[i]]
y=lp+rnorm(n,0,1)

idx.nex=sample(1:n,n*1/2,replace=FALSE)
ynew=y[idx.nex]
xnew=x[idx.nex,]
znew=z[idx.nex,]

y=y[-idx.nex]
x=x[-idx.nex,]
z=z[-idx.nex,]

fit1=btml(y,x,z,ynew=ynew,xnew=xnew,znew=znew)
fit1$mse.new
plot(fit1$y.hat.new~ynew,ylab=”Predicted y”,xlab=”ynew”)

###
#2. binary y
###
x=matrix(rnorm(n*10),n,10) #10 predictors
z=matrix(rnorm(n*10),n,10) #10 biomarkers

xcut=median(x[,1])
subgr=1*(x[,1]<xcut)+2*(x[,1]>=xcut) #2 subgroups

lp=rep(NA,n)
for(i in 1:n)
  lp[i]=1+3*z[i,subgr[i]]
prob=1/(1+exp(-lp))
y=rbinom(n,1,prob)
y=as.factor(y)

idx.nex=sample(1:n,n*1/2,replace=FALSE)
ynew=y[idx.nex]
xnew=x[idx.nex,]
znew=z[idx.nex,]

y=y[-idx.nex]
x=x[-idx.nex,]
z=z[-idx.nex,]

fit2=btml(y,x,z,ynew=ynew,xnew=xnew,znew=znew)
fit2$auc.new
plot(fit2$roc.new)

###
#3. add new ML models
#   1) write two functions:
#      c_xx & c_xx_predict if y is continuous or
#      b_xx & b_xx.predict if y is binary
#   2) MLlist includes xx, not c.xx nor b.xx.
#   3) run btml using the updated MLlist.
#   The below is an example of adding ridge regression.
###
#3.1. ridge regression for continuous y.
c_ridge=function(y,x){
  x=data.matrix(x)
  fit=NULL
  suppressWarnings(try(fit<-glmnet::cv.glmnet(x,y,alpha=0),silent=TRUE))
  return(fit)
}
c_ridge_predict=function(fit,xnew){
  y.hat=rep(NA,nrow(xnew))
  if(!is.null(fit)){
    xnew=data.matrix(xnew)
    y.hat=as.numeric(predict(fit,newx=xnew,s="lambda.min",type="response"))
  }
  return(y.hat)
}

#3.2. ridge regression for binary y.
b_ridge=function(y,x){
  x=data.matrix(x)
  fit=NULL
 suppressWarnings(try(fit<-glmnet::cv.glmnet(x,y,alpha=1,family="binomial"),silent=TRUE))
  return(fit)
}
b_ridge_predict=function(fit,xnew){
  y.hat=rep(NA,nrow(xnew))
  if(!is.null(fit)){
    xnew=data.matrix(xnew)
    y.hat=as.numeric(predict(fit,newx=xnew,s="lambda.min",type="response"))
  }
  return(y.hat)
}

#3.3. update MLlist
MLlist=c("lasso","ridge")
fit3=btml(y,x,z,ynew=ynew,xnew=xnew,znew=znew,MLlist=MLlist)
fit3$auc.new
plot(fit3$roc.new)
```

Index