Package ‘btml’

March 15, 2026

Type Package

Title Bayesian Treed Machine Learning for Personalized Prediction

Version 0.4.0

Date 2026-03-15

Description Generalization of the Bayesian classification and regression tree model that partitions subjects into terminal nodes and tailors predictive 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 2026-03-15 13:10:02 UTC

Contents

`btml` *Bayeisan Treed Machine Learning*

Description

Generalization of the Bayesian classification and regression tree model that partitions subjects into terminal nodes and tailors predictive 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 y is a factor codied as 0 or 1, classification is assumed. Otherwise, regression is assumed.
`x`	Data.frame or matrix that estimates a decision-tree structure.
`z`	Data.frame or matrix that predicts y in terminal nodes, i.e. terminal-node-specific ML models.
`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 predictive models 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 ML model selections based on a sparse Dirichlet prior rather than simply uniform (default sparse=TRUE).
`nwarm`	Number of warm-up (default nwarm=25000).
`niter`	Number of iteration (defaut niter=25000).
`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 if y is binary. (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

The btml function uses a stochastic search to identify a decision tree rule that partitions subjects into distinct terminal nodes and assigns the most effective predictive model to each terminal node.

Ideally, two sets of predictors are used: x and z (e.g., clinical variables and biomarkers), where x is used to construct the tree structure, and z is used for terminal-node-specific predictive models. If this separation is not possible, the same predictor x can be used for both tasks, for example:

btml(y=y, x=x, z=x, y=ynew, x=xnew, z=xnew)

Regarding node numbering, each internal node s has left and right child nodes 2s and 2s+1, respectively. The root node is indexed as node 1; 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.

Terminal-node-specific models include lasso(), randomForest(), and svm(...,kernel="radial") from the R packages cv.glmnet, randomForest, and e1071, respectively. Additional models can be flexibly incorporated; see Example 3 below for an illustration.

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 $\in$ 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(9)
###
#1. continuous y
###
n=200*2 #n=200 & 200 for training & test sets

x=matrix(rnorm(n*4),n,4)
z=matrix(rnorm(n*4),n,4)

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

```
lp=rep(NA,n)
for(i in 1:n){
  if(x[i,1]<0){
    lp[i]=1+3*z[i,1]
  }else{
    lp[i]=1+3*z[i,2]
  }
}
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,nwarm=1000,niter=1000)
fit1$mse.new
plot(fit1$y.hat.new~ynew,ylab="Predicted y",xlab="ynew")
abline(a=0,b=1,lwd=2,col="darkgray")

###
#2. binary y
###
x=matrix(rnorm(n*4),n,4)
z=matrix(rnorm(n*4),n,4)

lp=rep(NA,n)
for(i in 1:n){
  if(x[i,1]<0){
    lp[i]=1+3*z[i,1]
  }else{
    lp[i]=1+3*z[i,2]
  }
}
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,nwarm=1000,niter=1000)
```

```
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,nwarm=1000,niter=1000)
fit3$auc.new
plot(fit3$roc.new)
```

Index