Tree-Based Methods

by

AINDRILA GARAI

MSC STATISTICS, IIT KANPUR

aindrilag22@iitk.ac.in

INTRODUCTION:

Tree-based methods for regression and classification involve stratifying or segmenting the predictor space into a number of simple regions. To predict for a given observation, we use the mean or the mode response value for the training observations in the region to which it belongs. Since the set of splitting rules used to segment the predictor space can be summarized in a tree, these types of approaches are known as decision tree methods. It is useful for interpretation but not for prediction accuracy.

Example : Suppose, it consists of a series of splitting rules. The top split assigns observations having Years<4.5 to the left branch. The predicted salary for these players is given by the mean response value for the players in the data set with Years<4.5. Players with Years>=4.5 are assigned to the right branch and then that group is further subdivided by Hits: who made fewer than 118 hits last year and who made at least 118 hits last year.

NOTES:

- The regions are known as terminal nodes or **leaves** of the tree.
- The points along the tree where the predictor space is splitted are referred to as **internal nodes**.
- The segments of the trees that connect the nodes is called **branches**.

Regression Trees:

1. Divide the predictor space into some distinct and non-overlapping high-dimensional rectangles that minimize the RSS. For the X_j and the cutpoint s the first pair of half-planes will be

   $$R_1(j,s) = \{X \mid X_j < s\} \text{ and } R_2(j,s) = \{X \mid X_j \geq s\},$$

   and we seek the value of j and s that minimize the equation

   $$\sum_{i:x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$$

   where $\hat{y}_{R_i}$ is the mean response for the training observations in $R_i(j,s)$. Next, repeat the process. It is known as recursive binary splitting.

2. Make prediction using mean of the response values for the training observations in that region.

Tree Pruning:

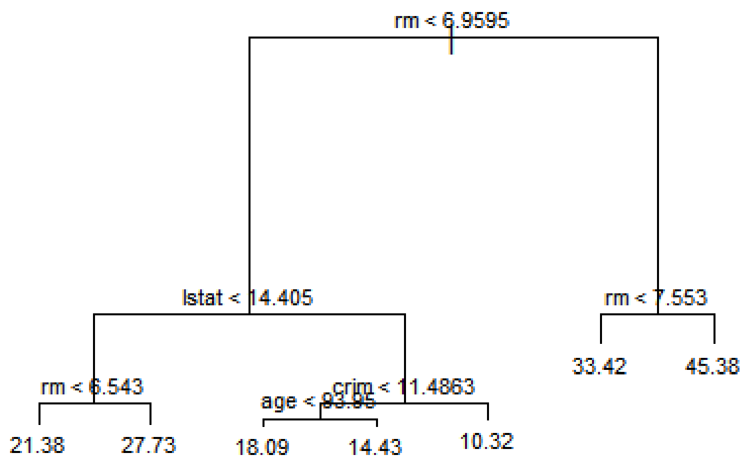
A smaller tree with fewer splits might lead to lower variance and better interpretation at the cost of a little bias. Our goal is to select a subtree that subtree leads to the lowest test error rate using cross-validation or the validation set approach. Cost complexity pruning (weakest link pruning) is used to do this. For each value of α there corresponds a subtree $T \subset T_0$ such that

$$\sum_{m=1}^{|T|} \sum_{i:x_i \in R_m} \left(y_i - \hat{y}_{R_m}\right)^2 + \alpha|T|$$

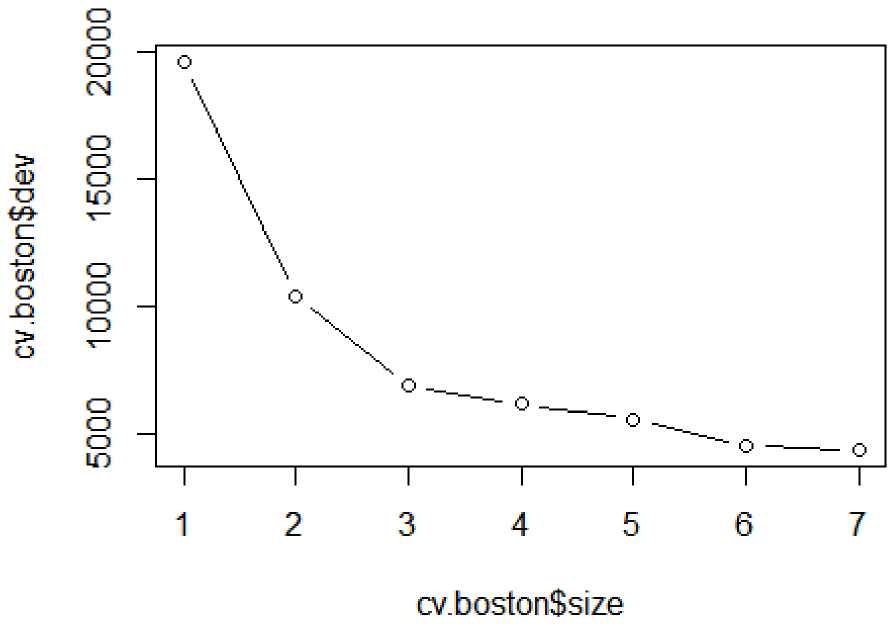
is small. Here |T| indicates the number of terminal nodes of the tree T,R_m is the rectangle corresponding to the mth terminal node, and $\hat{y}_{R_m}$ is the predicted response (mean) associated with R_m.

As α increases from zero the branches get pruned from the tree in a nested and predictable fashion, so obtaining the whole sequence of subtrees as a function of α is easy by cross validation.

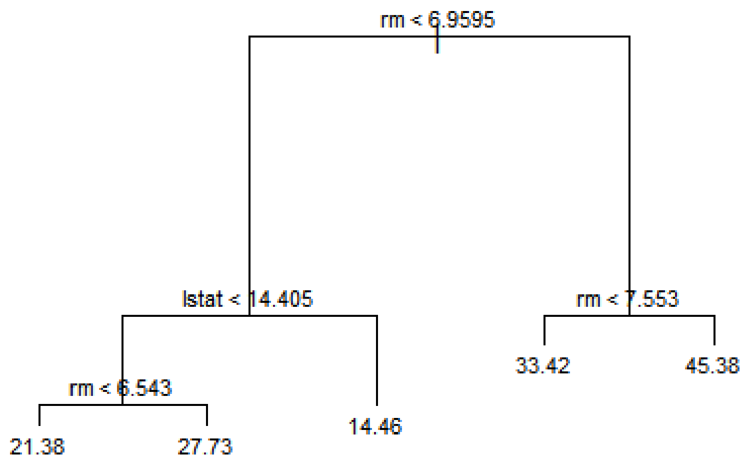
```
library(ISLR2)
library(tree)
set.seed (1)
train <- sample (1: nrow (Boston), nrow (Boston) / 2)
tree.boston <- tree (medv ~ ., Boston , subset = train)
summary (tree.boston)

##
## Regression tree:
## tree(formula = medv ~ ., data = Boston, subset = train)
## Variables actually used in tree construction:
## [1] "rm"    "lstat" "crim"  "age"
## Number of terminal nodes:  7
## Residual mean deviance:  10.38 = 2555 / 246
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -10.1800  -1.7770  -0.1775   0.0000   1.9230  16.5800

plot (tree.boston)
text (tree.boston , pretty = 0 , cex = 0.7)
```



```
cv.boston <- cv.tree (tree.boston)
plot (cv.boston$size , cv.boston$dev, type = "b")
```



```r
prune.boston <- prune.tree (tree.boston , best = 5)
plot (prune.boston)
text (prune.boston , pretty = 0 , cex = 0.7)
```



Classification Trees:

For a classification tree, we predict that each observation belongs to the most commonly occurring class of training observations in the region to which it belongs and class proportions among the training observations that fall into that region.

- Here we use classification error rate which is the fraction of the training observations in that region that do not belong to the most common class:

  $$E = 1 - \max_k(\hat{p}_{mk})$$

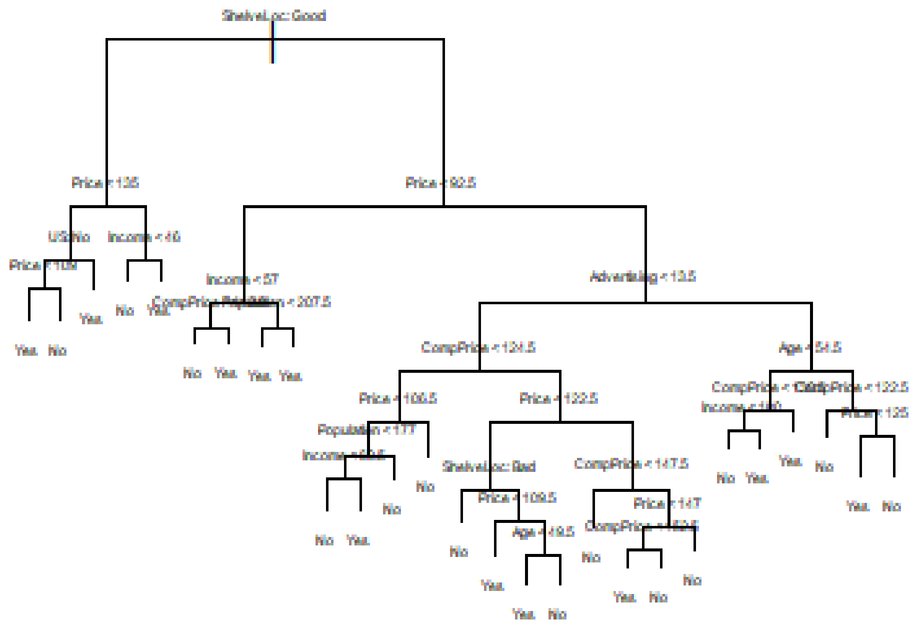
  Here $\hat{p}_{mk}$ represents the proportion of training observations in the mth region that are from the kth class. The other two measures are-

  $$G = \sum_{k=1}^{K} \hat{p}_{mk}\,(1 - \hat{p}_{mk}), D = -\sum_{k=1}^{K} \hat{p}_{mk}\log\hat{p}_{mk}$$

  the **Gini index** and **entropy**.

```r
library (tree)
library (ISLR2)
attach (Carseats)
High <- factor ( ifelse (Sales <= 8, "No", " Yes ") )
Carseats <- data.frame (Carseats , High)
```

```
tree.carseats <- tree (High ~ . - Sales , Carseats)
summary (tree.carseats)

##
## Classification tree:
## tree(formula = High ~ . - Sales, data = Carseats)
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Price"       "US"          "Income"      "CompPrice"
## [6] "Population"  "Advertising" "Age"
## Number of terminal nodes:  27
## Residual mean deviance:  0.4575 = 170.7 / 373
## Misclassification error rate: 0.09 = 36 / 400

plot (tree.carseats)
text (tree.carseats , pretty = 0 , cex = 0.3)
```



NOTES:

- **Gini index** is referred to as a measure of **node purity**— a small value indicates that a node contains predominantly observations from a single class.
- The **entropy** will take on a value near zero if the $\hat{p}_{mk}$ are all near zero or near one.
- The classification error rate is preferable if prediction accuracy of the final pruned tree is the goal otherwise we use either the Gini index or the entropy to evaluate the quality of a particular split due to have more sensitivity to node purity.
- The split is performed because it leads to increased node purity(Node purity is important to be pretty certain that a test observation belong to the particular region).

Trees Versus Linear Models: picture 339

From the above plots, - True decision boundary is linear: Use linear model not decision trees. - True decision boundary is non-linear: Use decision trees not linear model.

Advantages:

- Trees are very easy to explain to a layman.
- Decision trees more closely mirror human decision-making.
- Graphical representation.
- Need not to creat dummy variables to handle qualitative predictors.

Disadvantages:

- Trees do not have the same level of predictive accuracy.
- a small change in the data can cause a large change in the final estimated tree.

However, by aggregating many decision trees, using methods like bagging, random forests, and boosting, the predictive performance of trees can be substantially improved.

Ensemble Methods:

An ensemble method is an approach that combines many simple “building ensemble block” models in order to obtain a single and potentially very powerful model.

Bagging:

Bootstrap aggregation or bagging is a general-purpose procedure for reducing the bagging variance of a statistical learning method.

For quantitative outcome:

- A natural way to reduce the variance and increase the test set accuracy of a statistical learning method is to take many training sets from the population, build a separate prediction model using each training set and average the resulting predictions. Due to lack of multiple training sets we can bootstrap, by taking repeated samples from the (single) training data set. This is called bagging.

- To apply bagging to regression trees, we construct B regression trees using B bootstrapped training sets and average the resulting predictions. Each individual tree has high variance but low bias. Averaging these B trees reduces the variance.

For qualitative outcome:

For a given test observation, we can record the class predicted by each of the B trees and take a majority vote: the overall prediction is the most commonly occurring majority vote class among the B predictions.

Out-of-Bag Error Estimation:

Each bagged tree makes use of around two-thirds of the observations. The remaining one-third of the observations not used to fit a given bagged tree are referred to as the out-of-bag (OOB) observations. We can predict the response for the ith observation using each of the trees in which that observation was OOB. This will yield around B/3 predictions for the ith observation. To obtain a single prediction for the ith observation, we can average these predicted responses (if regression is the goal) or can take a majority vote (if classification is the goal). OOB error is a valid estimate of the test error for the bagged model since the response for each observation is predicted using only the trees that were not fit using that observation.

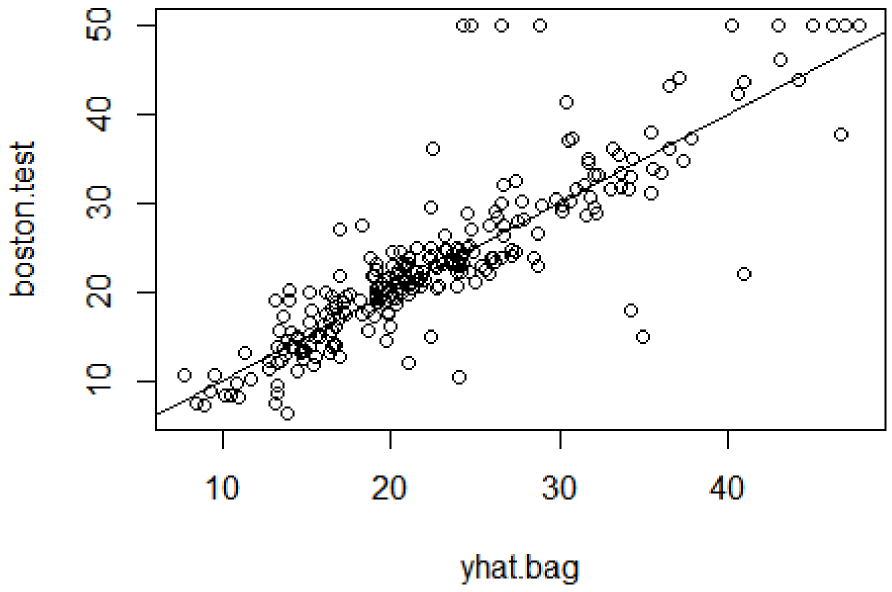
- Bagging improves prediction accuracy at the expense of interpretability. Variable importance is computed using the mean decrease in Gini index and expressed relative to the maximum.

Random Forests:

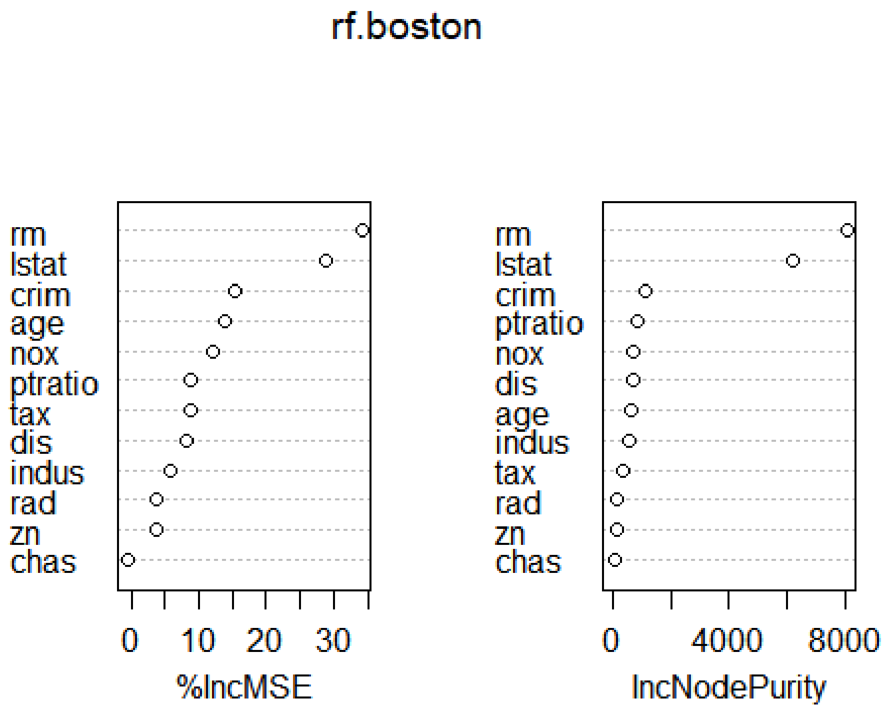
Using a small value of m in building a random forest will be helpful when we have a large number of correlated predictors. In building a random forest at each split in the tree, the algorithm is not even allowed to consider a majority of the available predictors. The predictions from the bagged trees will be highly correlated strong predictor in the data set, along with a number of other moderately strong predictors this process as decorrelating the trees, thereby making the average of the resulting trees less variable and hence more reliable. The main difference between bagging and random forests is the choice of predictor subset size m.

```
library (randomForest)

## randomForest 4.7-1.1

## Type rfNews() to see new features/changes/bug fixes.

set.seed (1)
train <- sample (1: nrow (Boston), nrow (Boston) / 2)
boston.test <- Boston[-train , "medv"]
tree.boston <- tree (medv ~ ., Boston , subset = train)
```

```
bag.boston <- randomForest (medv ~ ., data = Boston , subset = train , mtry =
12, importance = TRUE)
bag.boston

##
## Call:
##  randomForest(formula = medv ~ ., data = Boston, mtry = 12, importance = T
RUE,      subset = train)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 12
##
##           Mean of squared residuals: 11.25779
##                     % Var explained: 85.35

yhat.bag <- predict (bag.boston , newdata = Boston[-train , ])
plot (yhat.bag , boston.test)
abline (0, 1)
```



```
mean ((yhat.bag - boston.test)^2) # MSE

## [1] 23.40359

rf.boston <- randomForest (medv ~ ., data = Boston , subset = train , mtry =
6, importance = TRUE)
varImpPlot (rf.boston)
```



Boosting:

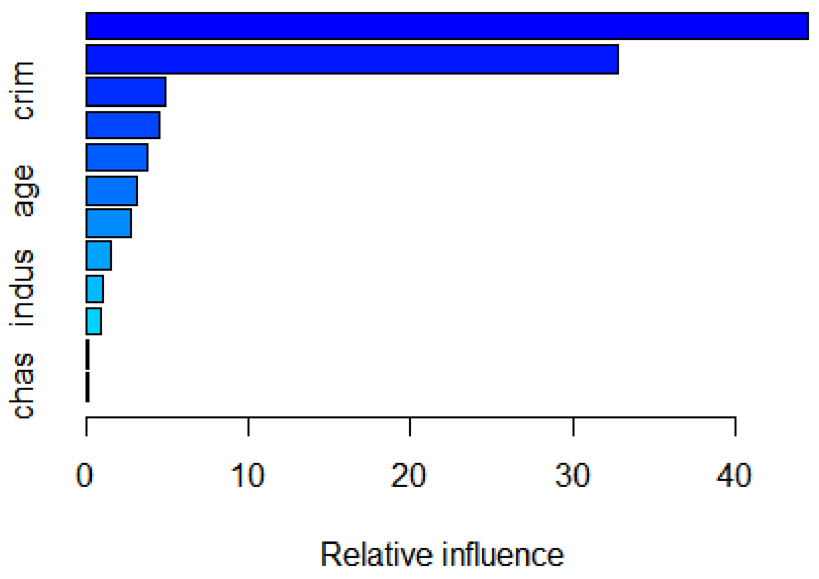
In Boosting each tree is grown using information from previously grown trees. Given the current model, we fit a decision tree to the residuals from the model. We then add this new decision tree into the fitted function to update the residuals. We slowly improve $\hat{f}$ in areas where it does not perform well.

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x)$$

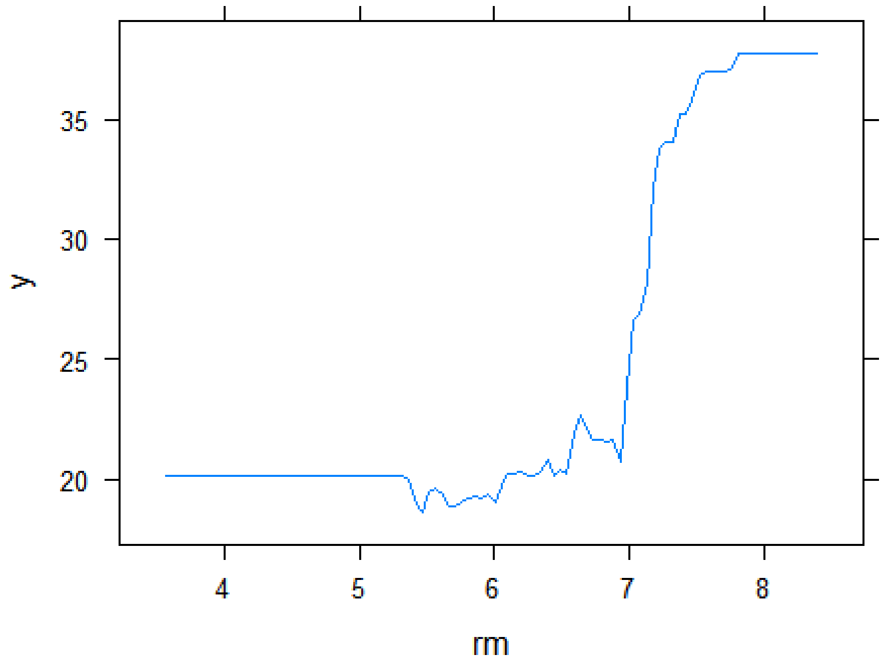
$\hat{f}^b$ is a fitted tree with d splits ($d + 1$ terminal nodes) to the training data (X, r).

- Boosting can overfit if B(The number of trees) is too large.
- The shrinkage parameter λ, a small positive number controls the rate at which boosting learns.
- d is the interaction depth, and controls interaction depth the interaction order of the boosted model, since d splits can involve at most d variables.

```
library (gbm)

## Loaded gbm 2.1.8.1

set.seed (1)
boost.boston <- gbm (medv ~ ., data = Boston[train , ], distribution = "gauss
ian", n.trees = 5000, interaction.depth = 4)
summary (boost.boston)
```



```
##               var     rel.inf
## rm             rm 44.48249588
## lstat       lstat 32.70281223
## crim         crim  4.85109954
## dis           dis  4.48693083
## nox           nox  3.75222394
## age           age  3.19769210
## ptratio ptratio  2.81354826
## tax           tax  1.54417603
## indus       indus  1.03384666
## rad           rad  0.87625748
## zn             zn  0.16220479
## chas         chas  0.09671228

plot (boost.boston , i = "rm")
```



```
boost.boston <- gbm (medv ~ ., data = Boston[train , ], distribution = "gauss
ian", n.trees = 5000,
interaction.depth = 4, shrinkage = 0.2, verbose = F)
yhat.boost <- predict (boost.boston , newdata = Boston[-train , ], n.trees =
5000)
mean ((yhat.boost - boston.test)^2)

## [1] 16.54778
```

Bayesian Additive Regression Trees:

Bayesian additive regression trees (BART), another ensemble method that uses decision trees as its building blocks. In BART each tree is constructed in a random manner as in bagging and random forests and each tree tries to capture signal not yet accounted for by the current model as in boosting. The BART method can be viewed as a Bayesian approach to fitting an ensemble of trees: each time we randomly perturb a tree in order to fit the residuals, we are in fact drawing a new tree from a posterior distribution.

```
library (BART)

## Loading required package: nlme

## Loading required package: nnet

## Loading required package: survival

x <- Boston[, 1:12]
y <- Boston[, "medv"]
```

```r
xtrain <- x[train , ]
ytrain <- y[train]
xtest <- x[-train , ]
ytest <- y[-train]
set.seed (1)
bartfit <- gbart (xtrain , ytrain , x.test = xtest)
```

```
## *****Calling gbart: type=1
## *****Data:
## data:n,p,np: 253, 12, 253
## y1,yn: 0.213439, -5.486561
## x1,x[n*p]: 0.109590, 20.080000
## xp1,xp[np*p]: 0.027310, 7.880000
## *****Number of Trees: 200
## *****Number of Cut Points: 100 ... 100
## *****burn,nd,thin: 100,1000,1
## *****Prior:beta,alpha,tau,nu,lambda,offset: 2,0.95,0.795495,3,3.71636,21.7
866
## *****sigma: 4.367914
## *****w (weights): 1.000000 ... 1.000000
## *****Dirichlet:sparse,theta,omega,a,b,rho,augment: 0,0,1,0.5,1,12,0
## *****printevery: 100
##
## MCMC
## done 0 (out of 1100)
## done 100 (out of 1100)
## done 200 (out of 1100)
## done 300 (out of 1100)
## done 400 (out of 1100)
## done 500 (out of 1100)
## done 600 (out of 1100)
## done 700 (out of 1100)
## done 800 (out of 1100)
## done 900 (out of 1100)
## done 1000 (out of 1100)
## time: 3s
## trcnt,tecnt: 1000,1000
```

```r
yhat.bart <- bartfit$yhat.test.mean
mean ((ytest - yhat.bart)^2)
```

```
## [1] 15.94718
```