

# Continuous outcomes with BART: Part 2

Robert McCulloch  
Arizona State University

Rodney Sparapani  
Medical College of Wisconsin

---

## Abstract

This short article illustrates examples of analyzing continuous outcomes with the **BART** R package.

*Keywords:* Bayesian Additive Regression Trees.

---

## 1. BART

In this section, we demonstrate the analysis of continuous outcomes with BART via the **BART** R package. For continuous outcomes, Bayesian Additive Regression Trees (BART) (Chipman, George, and McCulloch 2010) fit the basic model:

$$y_i = f(x_i) + \epsilon_i, \quad \epsilon_i \sim N(0, w_i^2 \sigma^2)$$

We use Markov Chain Monte Carlo (MCMC) to get draws from the posterior distribution of the parameter  $(f, \sigma)$ . In this section, we describe the functionality of **BART::wbart** which is the basic function in the **BART** R package. But first, we delve into the details of the BART prior itself.

### 1.1. Boston Housing Data

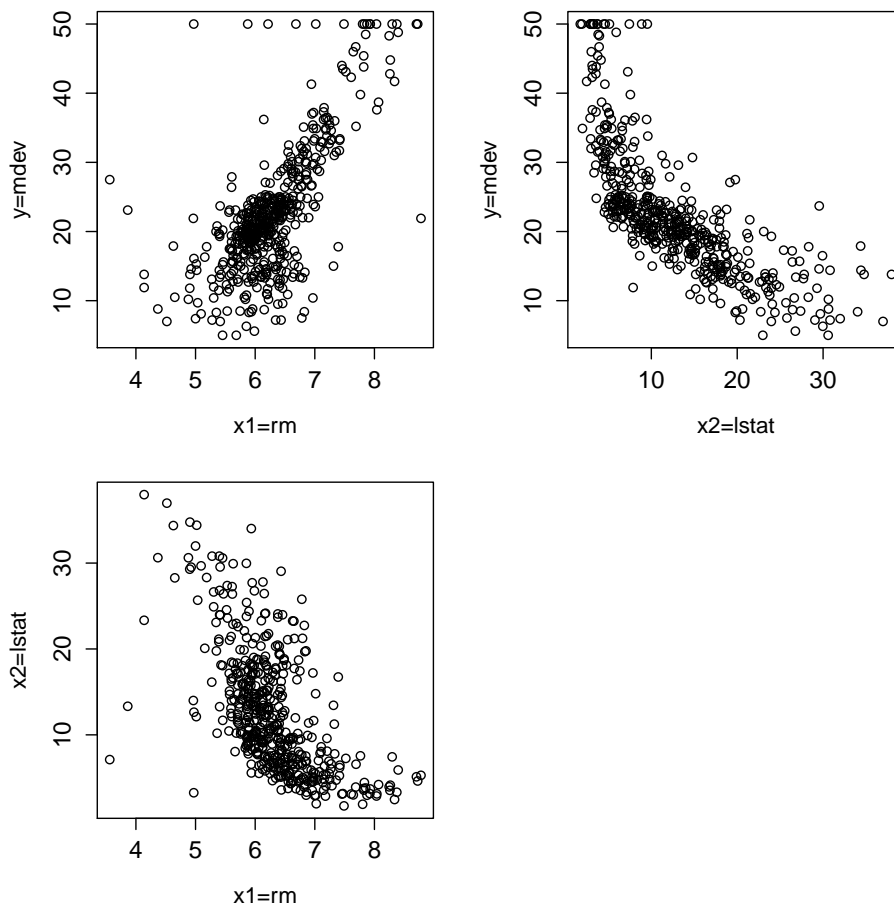
Let's examine the classic example of the Boston housing data. We'll predict the median house value, `y=medv`, from `x1 = rm` (number of rooms) and `x2=lstat` (lower status).

```
library(MASS)
x = Boston[,c(6,13)] #rm=number of rooms and lstat= percent lower status
y = Boston$medv # median value
head(cbind(x,y))

##      rm lstat    y
## 1 6.575  4.98 24.0
## 2 6.421  9.14 21.6
## 3 7.185  4.03 34.7
## 4 6.998  2.94 33.4
## 5 7.147  5.33 36.2
## 6 6.430  5.21 28.7
```

## 1.2. A Quick Look at the Data

```
par(mfrow=c(2,2))
par(mai=c(.8,.8,.2,.2))
plot(x[,1],y,xlab="x1=rm",ylab="y=mdev",cex.axis=1.3,cex.lab=1.2)
plot(x[,2],y,xlab="x2=lstat",ylab="y=mdev",cex.axis=1.3,cex.lab=1.2)
plot(x[,1],x[,2],xlab="x1=rm",ylab="x2=lstat",cex.axis=1.3,cex.lab=1.2)
```



## 1.3. Run wbart

```
library(BART) #BART package
set.seed(99) #MCMC, so set the seed
nd=200 # number of kept draws
burn=50 # number of burn in draws
bf = wbart(x,y,nskip=burn,ndpost=nd)

## *****Into main of wbart
## *****Data:
```

```
## data:n,p,np: 506, 2, 0
## y1,yn: 1.467194, -10.632806
## x1,x[n*p]: 6.575000, 7.880000
## *****Number of Trees: 200
## *****Number of Cut Points: 100 ... 100
## *****burn and ndpost: 50, 200
## *****Prior:beta,alpha,tau,nu,lambda: 2.000000,0.950000,0.795495,3.000000,5.979017
## *****sigma: 5.540257
## *****w (weights): 1.000000 ... 1.000000
## *****Dirichlet:sparse,theta,omega,a,b,rho,augment: 0,0,1,0.5,1,2,0
## *****nkeeptrain,nkeeptest,nkeepstme,nkeepreedraws: 200,200,200,200
## *****printevery: 100
## *****skiptr,skipte,skipteme,skiptreedraws: 1,1,1,1
##
## MCMC
## done 0 (out of 250)
## done 100 (out of 250)
## done 200 (out of 250)
## time: 1s
## check counts
## trcnt,tecnt,temecnt,treedrawscnt: 200,0,0,200
```

#### 1.4. Results returned with a list

We returned the results of running `wbart` in the object `bf` of type `wbart` which is essentially a list.

```
names(bf)

## [1] "sigma"          "yhat.train.mean" "yhat.train"
## [4] "yhat.test.mean" "yhat.test"       "varcount"
## [7] "varprob"        "treedraws"       "proc.time"
## [10] "mu"             "varcount.mean"   "varprob.mean"
## [13] "rm.const"

length(bf$sigma)

## [1] 250

length(bf$yhat.train.mean)

## [1] 506

dim(bf$yhat.train)

## [1] 200 506
```

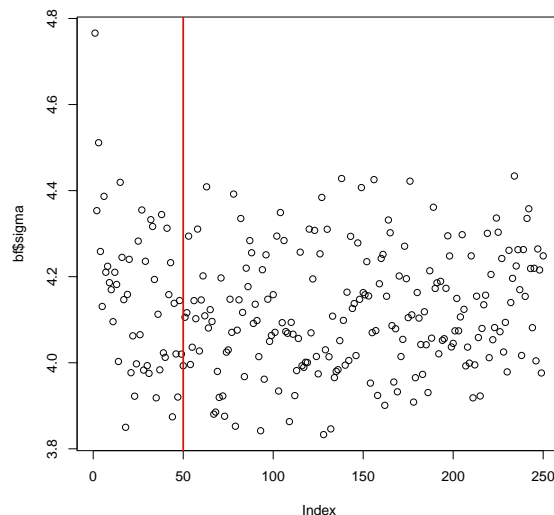
**Remember**, the training data has  $n = 506$  observations, we had `burn=50` burnin discarded draws and `nd=200` draws kept.

Let's look at a couple of the key list components. `$sigma`: burnin + kept (250) draws of  $\sigma$ . `yhat.train.mean`:  $j^{th}$  value is posterior mean of  $f(x_j)$ ,  $f$  evaluated at the  $j^{th}$  training observation. `yhat.train`:  $i,j$  value is the  $i^{th}$  kept MCMC draw of  $f(x_j)$ .

### 1.5. Assess Convergence

As with any high-dimensional MCMC, assessing convergence may be tricky. The simplest thing to look at are the draws of  $\sigma$ . The parameter  $\sigma$  is the only identified parameter in the model and it also gives us a sense of the size of the errors.

```
plot(bf$sigma)
abline(v=burn,lwd=2,col="red")
```



Looks like it burned in very quickly. Just one initial draw looking a bit bigger than the rest. Apparently, subsequent variation is legitimate posterior variation. In a more difficult problem you may see the  $\sigma$  draws initially declining as the MCMC searches for a good fit.

### 1.6. Look at in-sample Fit and Compare to a Linear Fit

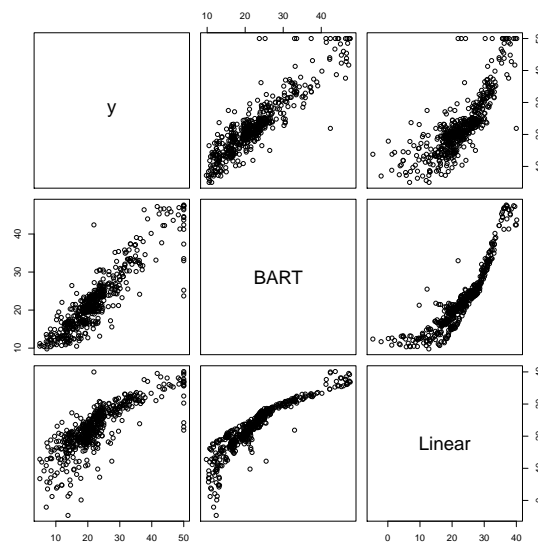
Let's look at the in-sample BART fit (`yhat.train.mean`) and compare it to `y=medv` fits from a multiple linear regression.

```
lmf = lm(y~.,data.frame(x,y))
fitmat = cbind(y,bf$yhat.train.mean,lmf$fitted.values)
colnames(fitmat)=c("y","BART","Linear")
cor(fitmat)
```

```
##          y          BART          Linear
```

```
## y      1.0000000 0.9051200 0.7991005
## BART   0.9051200 1.0000000 0.8978003
## Linear 0.7991005 0.8978003 1.0000000
```

```
pairs(fitmat)
```

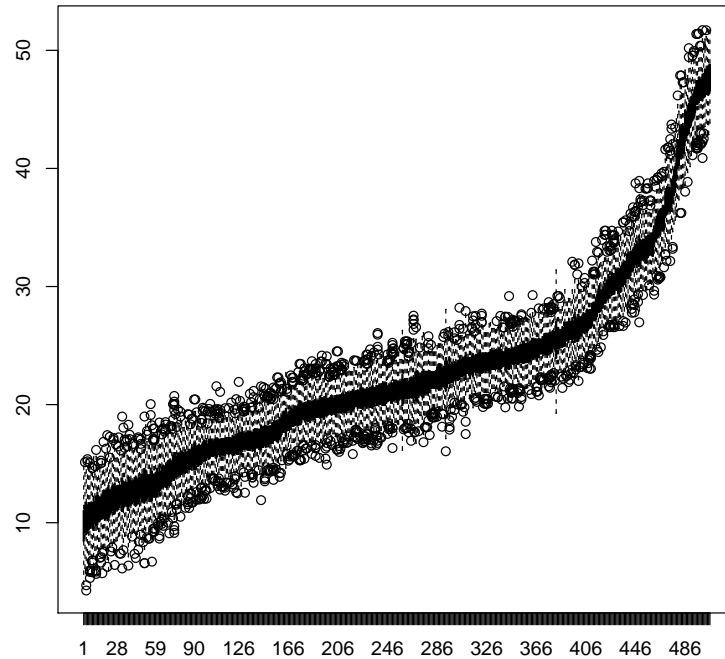


The BART fit is noticeably different from the linear fit.

## 1.7. A Quick Look at the Uncertainty

We order the observations by the fitted house value (`yhat.train.mean`) and then use boxplots to display the draws of  $f(x)$  in each column of `yhat.train`.

```
ii = order(bf$yhat.train.mean) #order observations by predicted value
boxplot(bf$yhat.train[,ii]) #boxplots of f(x) draws
```



Substantial predictive uncertainty, but you can still be fairly certain that some houses should cost more than other.

### 1.8. Using `predict.wbart`

We can get out of sample predictions in two ways. First, we can just ask for them when we call `wbart` by supplying a matrix or data frame of test  $x$  values. Second, we can call a `predict` method.

Let's split our data into train and test subsets.

```
n=length(y) #total sample size
set.seed(14) # Dave Keon, greatest Leaf of all time!
ii = sample(1:n,floor(.75*n)) # indices for train data, 75% of data
xtrain=x[ii,]; ytrain=y[ii] # training data
xtest=x[-ii,]; ytest=y[-ii] # test data
cat("train sample size is ",length(ytrain)," and test sample size is ",length(ytest),"\n")

## train sample size is  379  and test sample size is  127
```

And now we can run `wbart` using the training data to learn and predict at `xtest`. First, we'll just pass `xtest` to the `wbart` call.

```
dim(bfp1$yhat.test)

## [1] 1000 127

length(bfp1$yhat.test.mean)

## [1] 127
```

Now, `yhat.test`: the  $i, j$  value is the  $i^{th}$  kept MCMC draw of  $f(x_j)$  where  $x_j$  is the  $j^{th}$  row of `xtest`.

`yhat.test.mean`: the  $j^{th}$  value is the posterior mean of  $f(x_j)$ , i.e.,  $f$  evaluated at the  $j^{th}$  row of `xtest`.

Alternatively, we could run `wbart` saving all the MCMC results and then call `predict.wbart`.

```
set.seed(99)
bfp2 = wbart(xtrain,ytrain)

## *****Into main of wbart
## *****Data:
## data:n,p,np: 379, 2, 0
## y1,yn: -4.903958, 1.596042
## x1,x[n*p]: 6.431000, 9.520000
## *****Number of Trees: 200
## *****Number of Cut Points: 100 ... 100
## *****burn and ndpost: 100, 1000
## *****Prior:beta,alpha,tau,nu,lambda: 2.000000,0.950000,0.795495,3.000000,5.669300
## *****sigma: 5.394855
## *****w (weights): 1.000000 ... 1.000000
## *****Dirichlet:sparse,theta,omega,a,b,rho,augment: 0,0,1,0.5,1,2,0
## *****nkeeptrain,nkeeptest,nkeeptestme,nkeeptreedraws: 1000,1000,1000,1000
## *****printevery: 100
## *****skiptr,skipte,skipteme,skiptreedraws: 1,1,1,1
##
## 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
## check counts
## trcnt,tecnt,temecnt,treedrawscnt: 1000,0,0,1000

yhat = predict(bfp2,as.matrix(xtest)) #predict wants a matrix

## *****In main of C++ for bart prediction
## tc (threadcount): 1
## number of bart draws: 1000
## number of trees in bart sum: 200
## number of x columns: 2
## from x,np,p: 2, 127
## ***using serial code
```

So `yhat` and `bfp1$yhat.test` are the same.

```
dim(yhat)

## [1] 1000 127

summary(as.double(yhat-bfp1$yhat.test))

##      Min.      1st Qu.      Median      Mean      3rd Qu.      Max.
## -9.091e-09 -1.186e-09  2.484e-11  2.288e-12  1.188e-09  6.790e-09
```

## 1.9. Thinning

In our simple example of the Boston housing data set `wbart` runs pretty fast. But with more data and longer runs you may want to speed things up by saving less and then using `predict`. Let's just keep a thinned subset of 200 tree ensembles.

```
set.seed(4) #Bobby Orr's jersey number is the seed
bfthin = wbart(xtrain,ytrain,nskip=1000,ndpost=10000,
               nkeeptrain=0,nkeeptest=0,nkeeptestmean=0,nkeptreedraws=200)

## *****Into main of wbart
## *****Data:
## data:n,p,np: 379, 2, 0
## y1,yn: -4.903958, 1.596042
## x1,x[n*p]: 6.431000, 9.520000
## *****Number of Trees: 200
## *****Number of Cut Points: 100 ... 100
## *****burn and ndpost: 1000, 10000
## *****Prior:beta,alpha,tau,nu,lambd: 2.000000,0.950000,0.795495,3.000000,5.669300
## *****sigma: 5.394855
```



```
## *****w (weights): 1.000000 ... 1.000000
## *****Dirichlet:sparse,theta,omega,a,b,rho,augment: 0,0,1,0.5,1,2,0
## *****nkeeptrain,nkeeptest,nkeeptestme,nkeepreedraws: 0,0,0,200
## *****printevery: 100
## *****skiptr,skipte,skipteme,skiptreedraws: 10001,10001,10001,50
##
## MCMC
## done 0 (out of 11000)
## done 100 (out of 11000)
## done 200 (out of 11000)
## done 300 (out of 11000)
## done 400 (out of 11000)
## done 500 (out of 11000)
## done 600 (out of 11000)
## done 700 (out of 11000)
## done 800 (out of 11000)
## done 900 (out of 11000)
## done 1000 (out of 11000)
## done 1100 (out of 11000)
## done 1200 (out of 11000)
## done 1300 (out of 11000)
## done 1400 (out of 11000)
## done 1500 (out of 11000)
## done 1600 (out of 11000)
## done 1700 (out of 11000)
## done 1800 (out of 11000)
## done 1900 (out of 11000)
## done 2000 (out of 11000)
## done 2100 (out of 11000)
## done 2200 (out of 11000)
## done 2300 (out of 11000)
## done 2400 (out of 11000)
## done 2500 (out of 11000)
## done 2600 (out of 11000)
## done 2700 (out of 11000)
## done 2800 (out of 11000)
## done 2900 (out of 11000)
## done 3000 (out of 11000)
## done 3100 (out of 11000)
## done 3200 (out of 11000)
## done 3300 (out of 11000)
## done 3400 (out of 11000)
## done 3500 (out of 11000)
## done 3600 (out of 11000)
## done 3700 (out of 11000)
## done 3800 (out of 11000)
```

```
## done 3900 (out of 11000)
## done 4000 (out of 11000)
## done 4100 (out of 11000)
## done 4200 (out of 11000)
## done 4300 (out of 11000)
## done 4400 (out of 11000)
## done 4500 (out of 11000)
## done 4600 (out of 11000)
## done 4700 (out of 11000)
## done 4800 (out of 11000)
## done 4900 (out of 11000)
## done 5000 (out of 11000)
## done 5100 (out of 11000)
## done 5200 (out of 11000)
## done 5300 (out of 11000)
## done 5400 (out of 11000)
## done 5500 (out of 11000)
## done 5600 (out of 11000)
## done 5700 (out of 11000)
## done 5800 (out of 11000)
## done 5900 (out of 11000)
## done 6000 (out of 11000)
## done 6100 (out of 11000)
## done 6200 (out of 11000)
## done 6300 (out of 11000)
## done 6400 (out of 11000)
## done 6500 (out of 11000)
## done 6600 (out of 11000)
## done 6700 (out of 11000)
## done 6800 (out of 11000)
## done 6900 (out of 11000)
## done 7000 (out of 11000)
## done 7100 (out of 11000)
## done 7200 (out of 11000)
## done 7300 (out of 11000)
## done 7400 (out of 11000)
## done 7500 (out of 11000)
## done 7600 (out of 11000)
## done 7700 (out of 11000)
## done 7800 (out of 11000)
## done 7900 (out of 11000)
## done 8000 (out of 11000)
## done 8100 (out of 11000)
## done 8200 (out of 11000)
## done 8300 (out of 11000)
## done 8400 (out of 11000)
```

```

## done 8500 (out of 11000)
## done 8600 (out of 11000)
## done 8700 (out of 11000)
## done 8800 (out of 11000)
## done 8900 (out of 11000)
## done 9000 (out of 11000)
## done 9100 (out of 11000)
## done 9200 (out of 11000)
## done 9300 (out of 11000)
## done 9400 (out of 11000)
## done 9500 (out of 11000)
## done 9600 (out of 11000)
## done 9700 (out of 11000)
## done 9800 (out of 11000)
## done 9900 (out of 11000)
## done 10000 (out of 11000)
## done 10100 (out of 11000)
## done 10200 (out of 11000)
## done 10300 (out of 11000)
## done 10400 (out of 11000)
## done 10500 (out of 11000)
## done 10600 (out of 11000)
## done 10700 (out of 11000)
## done 10800 (out of 11000)
## done 10900 (out of 11000)
## time: 30s
## check counts
## trcnt,tecnt,temecnt,treedrawscnt: 0,0,0,200

yhatthin = predict(bfthin,as.matrix(xtest)) #predict wants a matrix

## *****In main of C++ for bart prediction
## tc (threadcount): 1
## number of bart draws: 200
## number of trees in bart sum: 200
## number of x columns: 2
## from x,np,p: 2, 127
## ***using serial code

dim(bfthin$yhat.train)

## [1] 0 379

dim(yhatthin)

## [1] 200 127

```

Now, there are no kept draws of  $f(x)$  for training  $x$ , and we have 200 tree ensembles to use with `predict.wbart`.

The thinning arguments.

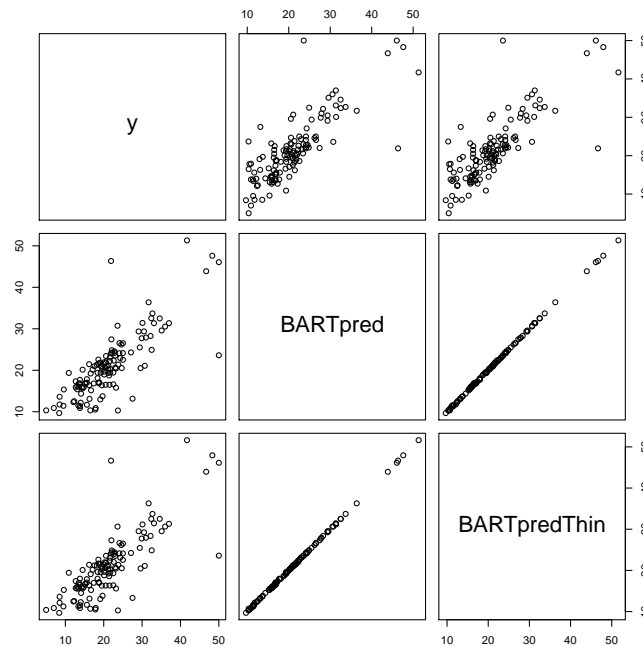
- `nkeeptrain` : number of  $f(x)$  draws to save for training  $x$ .
- `nkeeptest` : number of  $f(x)$  draws to save for test  $x$ .
- `nkeeptestmean` : number of draws to use in computing `yhat.test.mean`.
- `nkeptreedraws` : number of tree ensembles to keep.

The default values are to keep all the draws (e.g., `nkeeptrain=ndpost`).

Of course, if you keep 100 out of 100,000, you keep every 1,000th draw.

Now, let's have a look at the predictions.

```
fmat=cbind(ytest,bfp1$yhat.test.mean,apply(yhatthin,2,mean))
colnames(fmat) = c("y","BARTpred","BARTpredThin")
pairs(fmat)
```



**Recall**, the predictions labeled "BARTpred" are from a BART run with `seed=99` and all default values. The predictions labeled "BARTpredThin" are from 200 kept trees out of a long run with 1,000 burnins discarded and 10,000 draws kept with `seed=4`. **It is interesting how similar they are !!!!**

## References

Chipman HA, George EI, McCulloch RE (2010). “BART: Bayesian Additive Regression Trees.” *Annals of Applied Statistics*, 4, 266–98.

**Affiliation:**

Rodney Sparapani rsparapa@mcw.edu

Division of Biostatistics, Institute for Health and Equity

Medical College of Wisconsin, Milwaukee campus