

# Bayesian Additive Regression Trees

McCulloch, Sparapani

5/30/2023

Basic BART Ideas

Simple Examples

More on BART: Model and Prior

`wbart` Prior Options

BART MCMC

Added up Trees works great !!!!

Using Predict and Controlling What Is Saved

The Diabetes Example



## Basic BART Ideas

The original BART model is:

$$Y_i = f(x_i) + \epsilon_i, \quad \epsilon_i \sim N(0, \sigma^2), \quad iid.$$

*where*

*the function  $f$  is represented as the sum of many regression trees.*

*How can we find a complex nonlinear function  $f$  with little (human) work ???!!!*

BART was inspired by the Boosting literature, in particular the work of Jerry Friedman.

The connection to boosting is obvious in that the model is based on a sum of trees.

However, BART is a fundamentally different algorithm with some consequent pros and cons.

## BART is a Bayesian MCMC procedure.

We:

- ▶ put a prior on the model parameters  $(f, \sigma)$ .
- ▶ run a Markov chain with state  $(f, \sigma)$  such that the stationary distribution is the posterior

$$(f, \sigma) \mid D = \{x_i, y_i\}_{i=1}^n.$$

- ▶ Examine the draws as a representation of the full posterior.

The representation of  $f$  is complex with changing dimension so it is difficult to look directly of draws of  $f$ .

*However*, we can look at marginals of  $\sigma$  and  $f(x)$  at any given  $x$ .

Pick of set of  $\{x_j\}$ ,  $j = 1, 2, \dots, n_p$  and evaluate  $f$  at these  $x$  values.

So, if  $f_d$  is the  $d^{\text{th}}$  kept MCMC draw, then at every draw we have the fixed dimensional results

$$(f_d(x_1), f_d(x_2), \dots, f_d(x_{n_p}))$$



## Simple Examples

# A Simple One Dimensional Simulated Example

Let's simulate some simple data to try BART on.

```
##simulate training data
sigma = .1
f = function(x) {x^3}

set.seed(17)
n = 200
x = sort(2*runif(n)-1)

y = f(x) + sigma*rnorm(n)

#xtest: values we want to estimate f(x) at
# this is also our prediction for y.
xtest = seq(-1,1,by=.2)
```

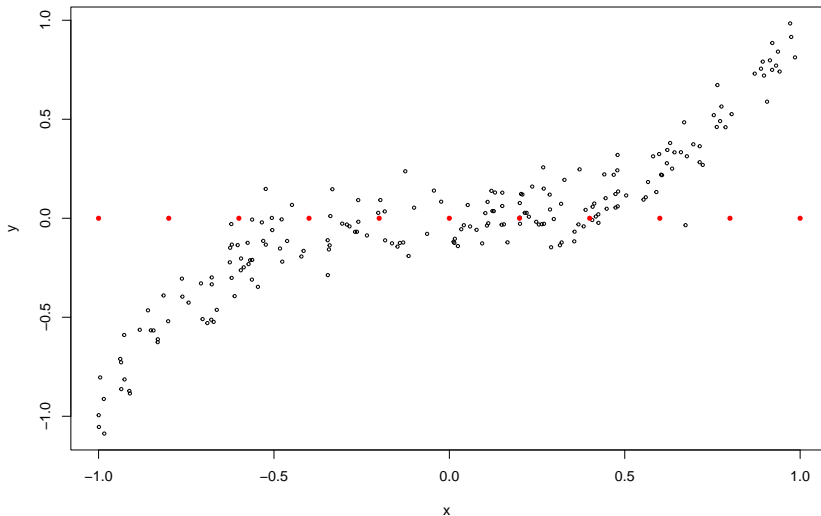
We have just one  $x$  so we can see the results with simple plots.

Let's have a look at our data:

```
##plot simulated data
```

```
plot(x,y,cex=.5)
```

```
points(xtest,rep(0,length(xtest)),col="red",pch=16,cex=.8)
```



Now we run BART on the simulated data using the function `BART::wbart`.

```
##run wbart
library(BART)
set.seed(14) #it is MCMC, set the seed!!
rb = wbart(x,y,xtest,nskip=500,ndpost=2000)
```

- ▶ `nskip`: the number of draws (MCMC iterations) that will be treated as burn-in, default is 500.
- ▶ `ndpost`: results will be kept after burn-in. Default is 1,000.

As usual, `rb` is a list containing results from the BART run.

`wbart` is for *weighted BART*, you can supply weights for each observation, but we won't be using this.

So, `rb` is the list containing the results of the call to `wbart`.

What is in `rb` ?????!!!!

```
names(rb)
## [1] "sigma"          "yhat.train.mean" "yhat.train"      "yhat.test.mean"
## [5] "yhat.test"      "varcount"        "varprob"         "treedraws"
## [9] "proc.time"     "mu"              "varcount.mean"  "varprob.mean"
## [13] "rm.const"
dim(rb$yhat.test)
## [1] 2000 11
```

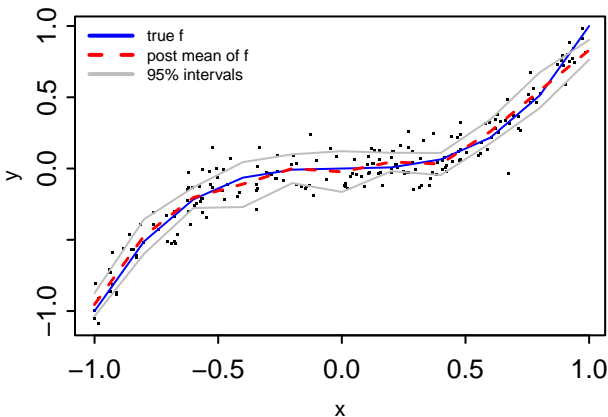
`$yhat.test`:

The  $(d, j)$  element of `yhat.test` is the  $d^{\text{th}}$  draw of  $f$  evaluated at the  $j^{\text{th}}$  value of `xtest`.

2,000 draws of  $f$ , each of which is evaluated at 11 `xtest` values.

Let's have a look at the fit and uncertainty.

```
plot(x,y,cex=.3,cex.axis=.8,cex.lab=.7, mgp=c(1.3,.3,0),tcl=-.2,pch=".")
lines(xtest,f(xtest),col="blue",lty=1)
lines(xtest,apply(rb$yhat.test,2,mean),col="red",lwd=1.5,lty=2) #post mean of  $f(x_j)$ 
qm = apply(rb$yhat.test,2,quantile,probs=c(.025,.975)) # post quantiles
lines(xtest,qm[1,],col="grey",lty=1,lwd=1.0)
lines(xtest,qm[2,],col="grey",lty=1,lwd=1.0)
legend("topleft",legend=c("true f","post mean of f","95% intervals"),
      col=c("blue","red","grey"), lwd=c(2,2,2), lty=c(1,2,1),bty="n",cex=.5,seg.len=3)
```



```

names(rb)
## [1] "sigma"          "yhat.train.mean" "yhat.train"      "yhat.test.mean"
## [5] "yhat.test"      "varcount"        "varprob"         "treedraws"
## [9] "proc.time"     "mu"              "varcount.mean"  "varprob.mean"
## [13] "rm.const"
dim(rb$yhat.train)
## [1] 2000 200
summary(rb$yhat.train.mean-apply(rb$yhat.train,2,mean))
##      Min.      1st Qu.      Median      Mean      3rd Qu.      Max.
## -1.166e-15 -8.327e-17 -3.469e-18  3.388e-17  5.551e-17  1.665e-15
summary(rb$yhat.test.mean-apply(rb$yhat.test,2,mean))
##      Min.      1st Qu.      Median      Mean      3rd Qu.      Max.
## -4.441e-16 -9.368e-17  1.041e-17  1.501e-16  1.353e-16  1.665e-15

```

`$yhat.train:`

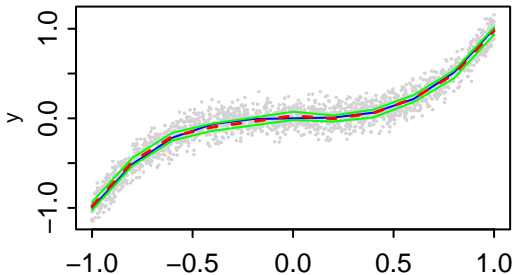
The  $(d, j)$  element of `yhat.train` is the  $d^{\text{th}}$  draw of  $f$  evaluated at the  $j^{\text{th}}$  value of  $x$ .

`$yhat.train(test).mean:`

Average the draws to get the estimate of the posterior mean of  $f(x)$  (remember,  $f$  is the random variable !!).

What should happen when we bump up  $n$  (the sample size)?

```
n = 2000 #was 200 before !!
set.seed(17)
x = sort(2*runif(n)-1)
y = f(x) + sigma*rnorm(n)
rb2 = wbart(x,y,xtest) #run at defaults !!
plot(x,y,cex=.1,cex.axis=.8,cex.lab=.7, mgp=c(1.3,.3,0),tcl=-.2,pch=1,col="lightgrey")
lines(xtest,f(xtest),col="blue",lty=1)
lines(xtest,rb2$yhat.test.mean,col="red",lwd=1.5,lty=2)
qm = apply(rb2$yhat.test,2,quantile,probs=c(.025,.975))
lines(xtest,qm[1,],col="green",lty=1,lwd=1.0)
lines(xtest,qm[2,],col="green",lty=1,lwd=1.0)
```



We nail  $f$  and the uncertainty goes down.

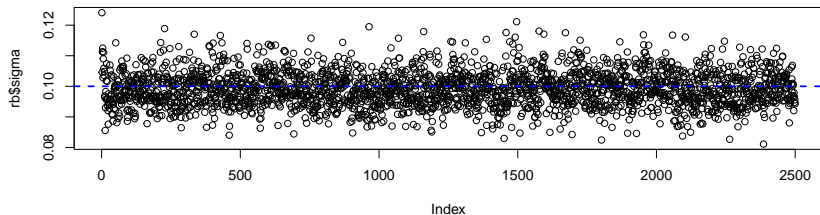


Our only simple parameter is  $\sigma$ . Let's look at the draws.  
The BART list component is `sigma`.

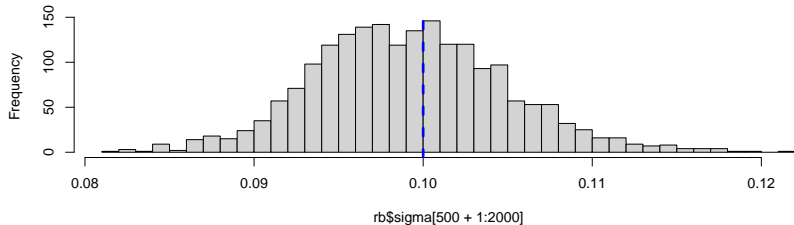
```
length(rb$sigma)
## [1] 2500
```

For  $\sigma$  we keep draws from all MCMC iterations, that is, we include the 500 burn-in draws and the 2,000 post burn-in draws.

```
par(mfrow=c(2,1))
#plot all draws to see burn-in, burns in fast!!
plot(rb$sigma); abline(h=sigma, lty=2, col='blue', lwd=2)
# just plot draws after (hopefully) burn-in
hist(rb$sigma[500+1:2000], nclass=50, main="histogram of sigma draws, blue at true")
abline(v=sigma, lty=2, col="blue", lwd=3)
```



histogram of sigma draws, blue at true



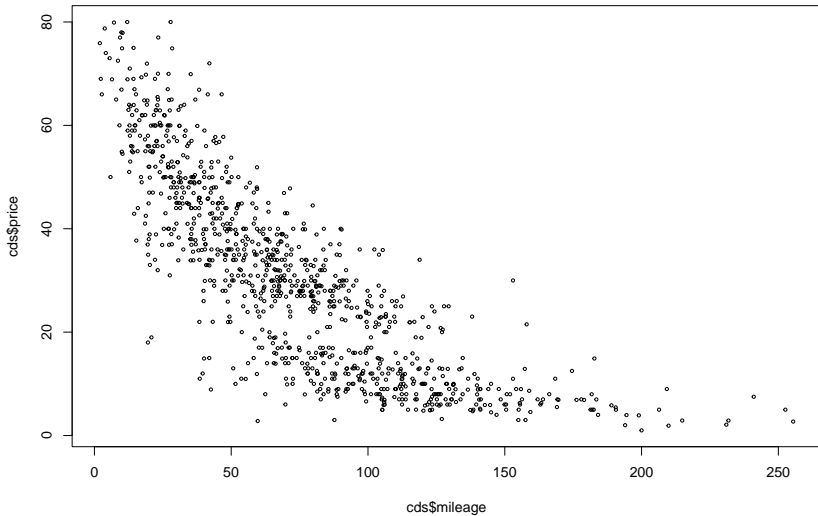
# Cars with Mileage and Price

Each observation corresponds to a used car which has been sold.

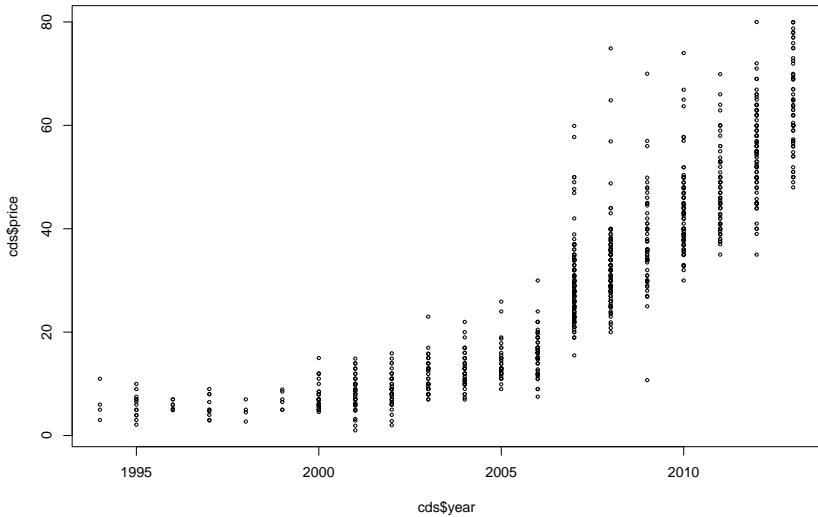
How does the price relate to mileage and year ?

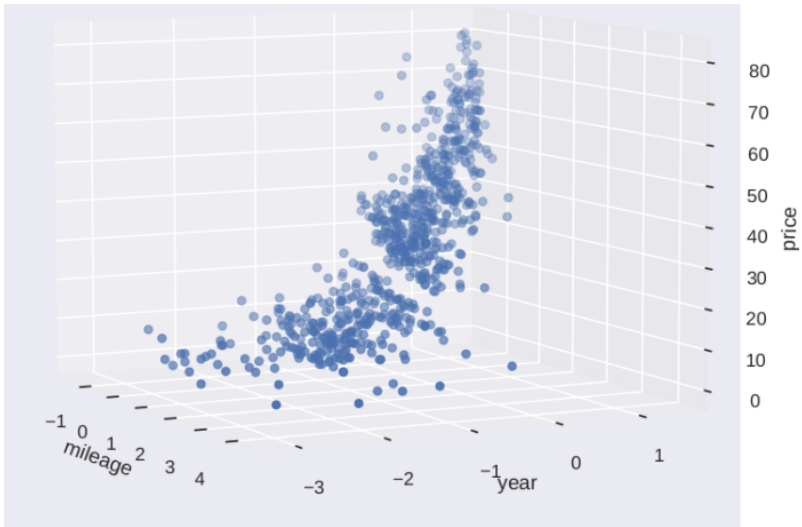
```
cd = read.csv("http://www.rob-mcculloch.org/data/susedcars.csv")
cds = data.frame(price = cd$price/1000, mileage = cd$mileage/1000, year = cd$year)
head(cds)
##   price mileage year
## 1  43.995  36.858 2008
## 2  44.995  46.883 2012
## 3  25.999 108.759 2007
## 4  33.880  35.187 2007
## 5  34.895  48.153 2007
## 6   5.995 121.748 2002
```

```
plot(cds$mileage,cds$price,pty='s',cex=.5)
```

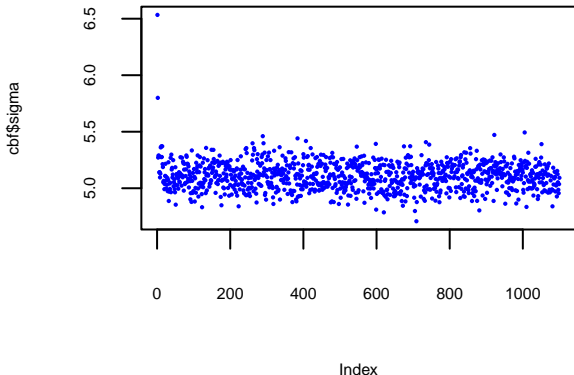


```
plot(cds$year, cds$price,pty='s',cex=.5)
```





```
cbf = wbart(cds[,2:3],cds[[1]]) #nskip=100, ndpost=1000  
plot(cbf$sigma,cex=.3,col='blue',pch=16,cex.lab=.5,cex.axis=.5)
```

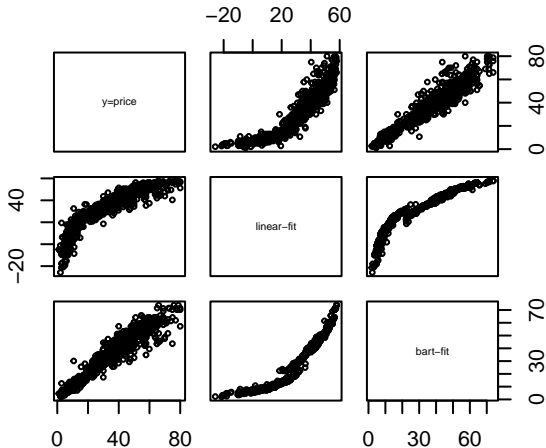


We see the burn in as the Markov chain initially searches for fit.

```

clf = lm(price~.,cds)
fmat = cbind(cds$price,clf$fitted.values,cbf$yhat.train.mean)
colnames(fmat)=c('y=price','linear-fit','bart-fit')
pairs(fmat,cex.labels=.5,cex=.5)

```



```

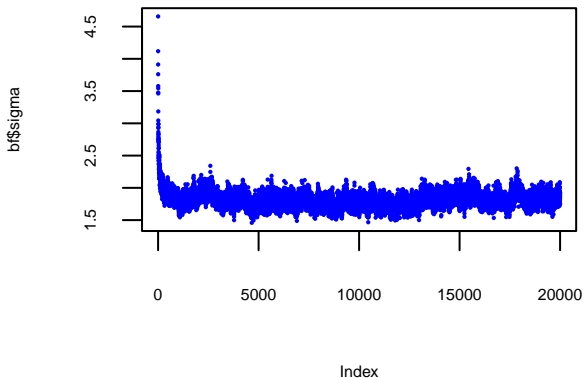
cor(fmat)
##           y=price linear-fit  bart-fit
## y=price   1.0000000  0.9123897  0.9634202
## linear-fit 0.9123897  1.0000000  0.9494321
## bart-fit  0.9634202  0.9494321  1.0000000

```



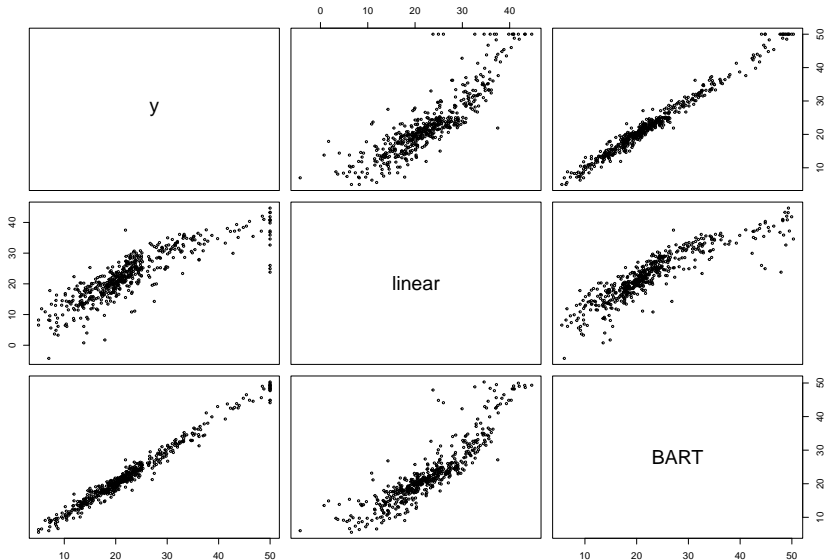
# Boston Housing

```
library(MASS)
attach(Boston)
y = Boston$medv
x = Boston[,1:13]
set.seed(99) # Wayne Gretzky
burn=10000; nd=10000
bf = wbart(x,y,nskip=burn,ndpost=nd,printevery=5000) #print progress every 5000th MCMC iter
plot(bf$sigma,cex=.3,col='blue',pch=16,cex.lab=.5,cex.axis=.5)
```



Longer burn-in.

```
lmf = lm(y~.,data.frame(x,y))  
fmat=cbind(y,lmf$fitted.values,bf$yhat.train.mean)  
colnames(fmat) = c('y','linear','BART')  
pairs(fmat,cex=.5,cex.lab=.5)
```



# The Friedman Simulated Example

Let's try a tougher simulated problem.

```
## f and sigma
f = function(x){
  10*sin(pi*x[,1]*x[,2]) + 20*(x[,3]-.5)^2+10*x[,4]+5*x[,5]
}
sigma = 1.0 #y = f(x) + sigma*z , z~N(0,1)

## simulate
n = 100 #number of observations
set.seed(99)
x=matrix(runif(n*10),n,10) #10 variables, only first 5 matter
Ey = f(x)
y=Ey+sigma*rnorm(n)

## run BART
burn=500;nd=5000
rbf = wbart(x,y,nskip=burn,ndpost=nd)

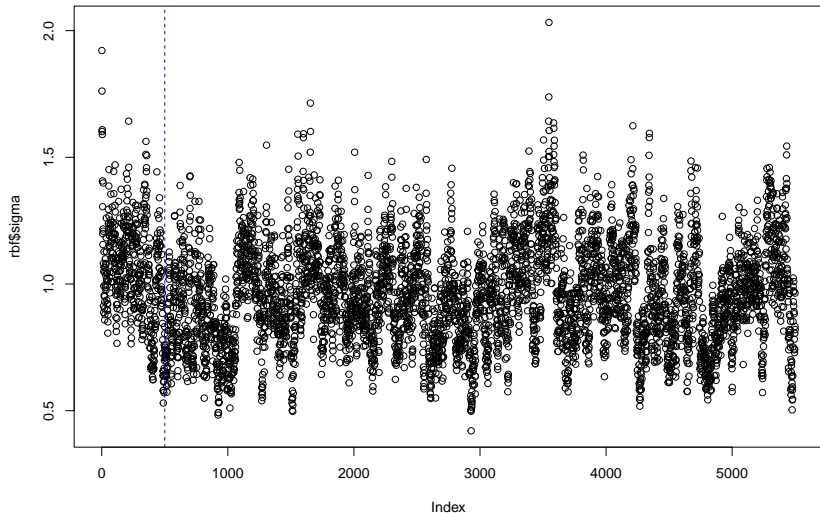
dim(rbf$yhat.train); dim(rbf$yhat.test); length(rbf$sigma)
## [1] 5000 100
## [1] 5000 0
## [1] 5500
$sigma:
```

MCMC draws of  $\sigma$  *including burn-in*.

So, the length is `nskip + ndpost`.

Let's look at the draws of  $\sigma$  including burn-in:

```
plot(rbf$sigma)
abline(v=burn,col="blue",lty=2)
```

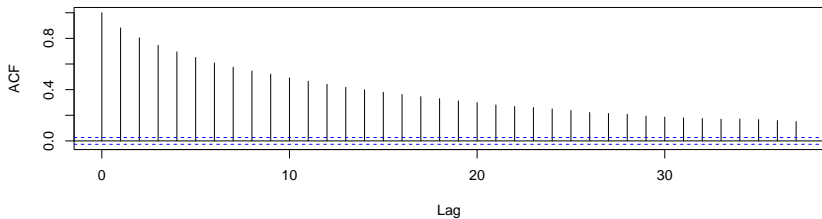


You can see the initial burn-in.

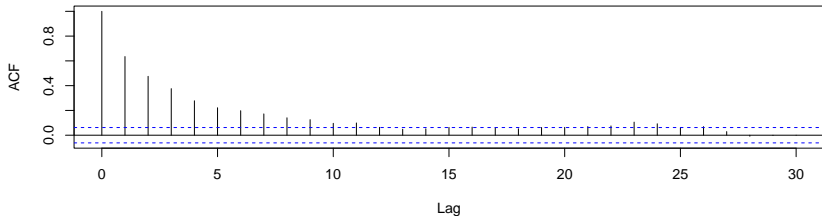
Burns in pretty fast, but after that there is substantial autocorrelation!

```
par(mfrow=c(2,1))
acf(rbf$sigma)
ii = burn + (1:1000)*(nd/1000)
acf(rbf$sigma[ii])
```

Series rbf\$sigma

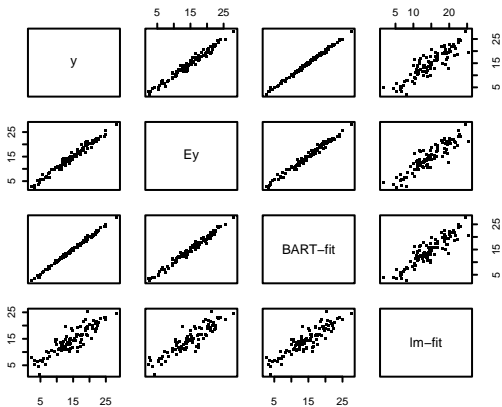


Series rbf\$sigma[ii]



## Did it work??!

```
lmf = lm(y~.,data.frame(x,y))  
fmat = cbind(y,Ey,rbf$yhat.train.mean,lmf$fitted)  
colnames(fmat) = c("y","Ey","BART-fit","lm-fit")  
pairs(fmat,cex=.3,cex.axis=.5,cex.lab=.7, mgp=c(1.3,.3,0),  
      tcl=-.2,pch=".",cex.labels=.6)
```



```
cor(fmat)
##           y           Ey  BART-fit  lm-fit
## y         1.000000  0.9847984  0.9973980  0.8841787
## Ey        0.9847984  1.0000000  0.9892571  0.9009389
## BART-fit  0.9973980  0.9892571  1.0000000  0.9008033
## lm-fit    0.8841787  0.9009389  0.9008033  1.0000000
```

- ▶ BART was able to fit the data very well.
- ▶ looks like it overfit a bit.
- ▶ burned in fast.
- ▶ draws may be autocorrelated, but not so much that you can't thin them out and get something nice.

Our course, *whether it and converged and how autocorrelated the draws are* can depend on what you look at.

For example, you might really care about  $f(x)$  at particular set of  $x$  and the draws  $f_d(x)$  might exhibit more autocorrelation than  $\sigma$ .

Nevertheless, looking at draws of  $\sigma$  gives you a quick feel for how BART is finding fit.



More on BART: Model and Prior

So far we have been using `wbart` with only an understanding of the abstract model

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

To use more of the `wbart` options we need to have deeper understanding of how the BART model works.

In particular, we need to understand something about the BART prior on the function  $f$ .

# Regression Trees

First, we review regression trees to set the notation for BART.

Note however that even in the simple regression tree case, our Bayesian approach is very different from the usual CART type approach.

The model will have *parameters* and corresponding *priors*.

## Model with a Single Tree

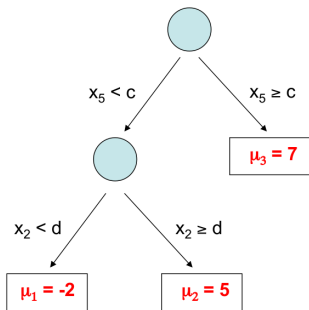
Let  $T$  denote the tree structure including the decision rules.

Let  $M = \{\mu_1, \mu_2, \dots, \mu_b\}$ , denote the set of bottom node  $\mu$ 's.

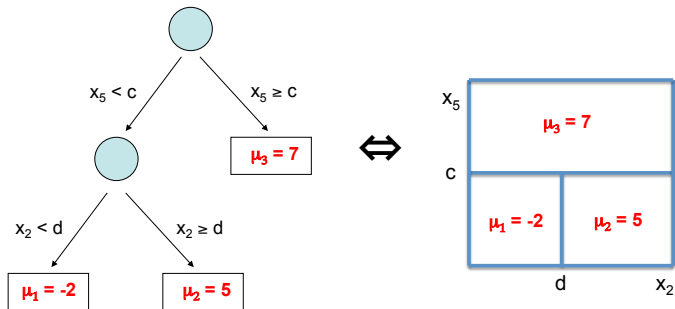
Let  $g(x; \theta)$ ,  $\theta = (T, M)$  be a regression tree function that assigns a  $\mu$  value to  $x$ .

A single tree model:

$$y = g(x; \theta) + \epsilon.$$



## A coordinate view of $g(x; \theta)$

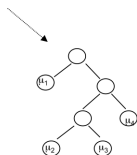


Easy to see that  $g(x; \theta)$  is just a step function.

# The BART Model

In BART, the function  $f$  is expressed as a sum of regression trees:

$$Y = g(x; T_1, M_1) + g(x; T_2, M_2) + \dots + g(x; T_m, M_m) + \sigma z, \quad z \sim N(0, 1)$$



$m = 200, 1000, \dots, \text{big}, \dots$

$f(x | \cdot)$  is the sum of all the corresponding  $\mu$ 's at each bottom node.

Such a model combines additive and interaction effects.

*All parameters but  $\sigma$  are unidentified !!!!*

... *the connection to Boosting is obvious*...

*But*, ....

Rather than simply adding in fit in an iterative scheme, we will explicitly specify a prior on the model which directly impacts the performance.

In `wbart` we have the parameter:

`ntree`: the parameter  $m$ , the number of trees in the sum.

## Complete the Model with a Regularization Prior

$$\begin{aligned}\pi(\theta) &= \pi((T_1, M_1), (T_2, M_2), \dots, (T_m, M_m), \sigma) \\ &= \pi(\sigma) \prod_{i=1}^m \pi(T_i, M_i) \\ &= \pi(\sigma) \prod_{i=1}^m \pi(T_i) \pi(M_i | T_i)\end{aligned}$$

$\pi$  wants:

- ▶ each  $T$  small.
- ▶ each  $\mu$  small.
- ▶ “nice”  $\sigma$  (e.g. smaller than the least squares estimate).

We refer to  $\pi$  as a regularization prior because it restrains the overall fit.

In addition, it keeps the contribution of each  $g(x; T_i, M_i)$  model component small.



$\pi(T)$ , prior on  $T$

Since the  $T_i$  are iid, we must choose a single  $\pi(T)$ .

We specify a process we can use to draw a tree from the prior.

The probability a current bottom node, at depth  $d$ , gives birth to a left and right child is

$$\frac{\alpha}{(1+d)^\beta}$$

In `wbart` we have the parameters:

**base**: the parameter  $\alpha$  with default value .95.

**power**: the parameter  $\beta$  with default value 2.0.

*This makes non-null but small trees likely.*

Prior distribution on the number of bottom nodes:

Number of Bottom Nodes	Probability
1	.05
2	.55
3	.28
4	.09
5	.03

*But note*, if the data demands it, you can build deeper trees than this prior would suggest.

This would be needed to model complex interactions.

$\pi(M | T)$ , prior on bottom node  $\mu$

Let  $b$  be the number of bottom nodes in the tree  $T$ .

Then  $M = \{\mu_1, \mu_2, \dots, \mu_b\}$ .

We let,

$$\pi(M | T) = \prod_{i=1}^b \pi(\mu_i)$$

That is

$$\mu_i \sim \pi(\mu), \quad iid$$

Now we only have to choose  $\pi(\mu)$ .

First of all we *center the data* so that we can assume we want to think  $f$ , and hence each  $\mu$  towards 0.

So, essentially, we fit the model:

$$Y_i = \mu + f(x_i) + \epsilon_i$$

but pick a value  $\hat{\mu}$  for  $\mu$ .

The code just subtracts  $\hat{\mu}$  from each  $y_i$  and then adds it back to the draws of  $f(x)$ .

In `wbart`:

`fmean` : the value  $\hat{\mu}$ , with default value  $\bar{y}$ .

`$mu`: the  $\hat{\mu}$  that was used (so default will just be  $\bar{y}$ ).

We let

$$\mu_i \sim N(0, \tau^2)$$

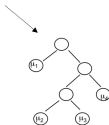
where the zero mean is justified by the centering.

*How do we choose the key parameter  $\tau$  ??*

The key observation is that:

$$f(x | T, M) = \mu_1 + \mu_2 + \dots + \mu_m.$$

$$Y = g(x; T_1, M_1) + g(x; T_2, M_2) + \dots + g(x; T_m, M_m) + \sigma z, \quad z \sim N(0, 1)$$



$$\mu_i \sim N(0, \tau^2)$$

$$f(x | T, M) = \mu_1 + \mu_2 + \cdots + \mu_m.$$

Which gives us,

$$f(x) | T \sim N(0, m\tau^2)$$

*which does not depend on T or x !!!!!*

$$f(x) | T \sim N(0, m\tau^2)$$

In `wbart` we use the parameter  $k$  to specify the  $\mu$  prior via

$$\tau = \frac{\max(y) - \min(y)}{2 k \sqrt{m}}$$

that way

$$k \sqrt{m} \tau = \frac{\max(y) - \min(y)}{2}$$

so that from the “middle” of  $f$  to the extreme is  $k$  standard deviations.

In `wbart`,  $k$  is called `k`:

`k`:

number of standard deviations of  $f(x)$  that equal half the range of  $y$ , the default value is 2.

You can also just directly set the standard deviation of  $f(x)$ .

$$f(x) \sim N(0, \sigma_f^2)$$

which is the same as just setting

$$\tau^2 = \frac{\sigma_f^2}{m}$$

In `wbart`:

`sigmaf` : prior standard deviation of  $f(x)$ .



$\pi(\sigma)$ , prior on  $\sigma$ .

$$\sigma^2 \sim \frac{\nu \lambda}{\chi_\nu^2}$$

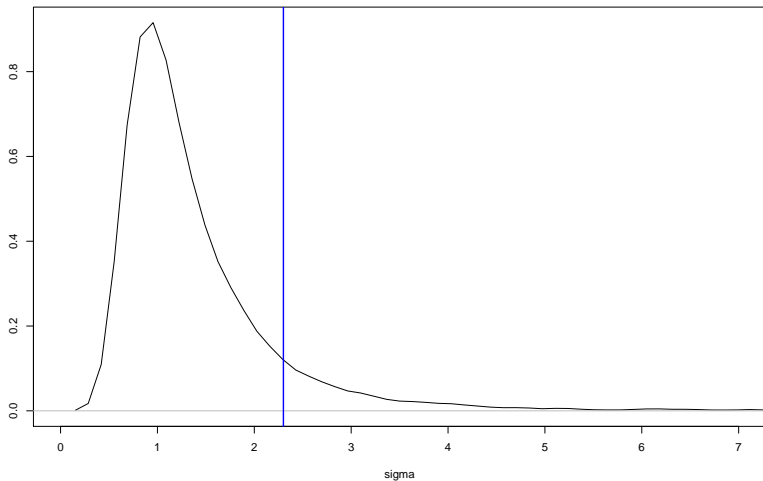
The `wbart` strategy is to choose  $\nu$  and a rough estimate of  $\sigma$ ,  $\hat{\sigma}$ .

If  $p < n$ ,  $\hat{\sigma}$  is the usual least squares estimate, else  $sd(y)$ .

We then choose  $\lambda$  so that  $\hat{\sigma}$  is at a chosen quantile  $q$  of the prior.

Solid blue line at  $\hat{\sigma}$ .

$$p(\sigma < \hat{\sigma}) = q.$$



In `wbart`:

`sigdf`:  $\nu$ , default is 3.

`sigest`:  $\hat{\sigma}$ , default is least squares estimate if  $p < n$  and `sd(y)` otherwise.

`sigquant`:  $q$ , default is .9.

You can also just set  $\lambda$  directly:

`lambda`:  $\lambda$ .

wbart Prior Options

## In summary:

wbart has the following BART prior related parameters:

T: power, base.

M: k, sigmaf.

$\sigma$ : sigest, sigdf, sigquant, lambda

In addition, you must choose `ntree` and `numcut`.

`numcut` is the number of cutpoints  $c$  used for the decision rules  $x_i < c$ .

Let's just look at a few examples to get some intuition for the effect of these prior choices on the BART inference.

Back to our 1-D example so we can see what happens.

```
sigma = .1
f = function(x) {x^3}
set.seed(17)
n = 100
x = sort(2*runif(n)-1)
y = f(x) + sigma*rnorm(n)
lmf = lm(y~x,data.frame(x,y))
sigest = summary(lmf)$sigma
cat("least squares estimate of sigma is: ", sigest ,"\n")
## least squares estimate of sigma is: 0.1742962
```

```
#density of sigma for sigma^2 ~ nu*lam/chisq_nu
sden=function(sv,nu,lam) {
  dchisq(nu*lam/sv^2,df=nu)*(2*nu*lam)/sv^3
}
#lam val which puts sigest and sigquant quantile
lamval = function(sigest,sigquant,nu) {
  qchi = qchisq(1.0-sigquant,nu)
  lambda = (sigest*sigest*qchi)/nu #lambda parameter for sigma prior
  lambda
}
```

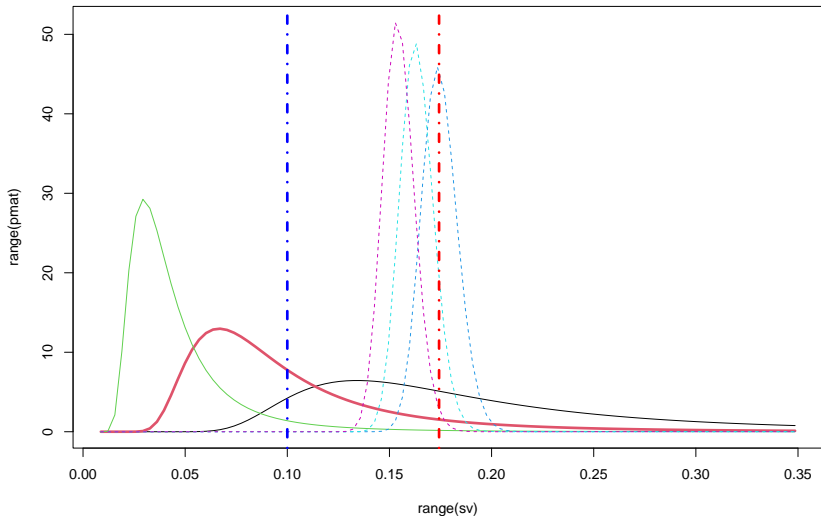
```

#vary quantile and nu
qv=c(.5,.9,.99); nuv = c(3,200)
pg = expand.grid(qv,nuv)
names(pg) = c("quantile","nu")
pg
##   quantile  nu
## 1      0.50   3
## 2      0.90   3
## 3      0.99   3
## 4      0.50 200
## 5      0.90 200
## 6      0.99 200

#evaluate prior density of sigma for our 6 priors
sv = seq(from=.05*sigest,to=2*sigest,length.out=100)
pmat = matrix(0.0,length(sv),nrow(pg))
for(i in 1:nrow(pg)) {
  pmat[,i] = sden(sv,pg[i,2],lamval(sigest,pg[i,1],pg[i,2]))
}

```

```
plot(range(sv),range(pmat),type="n")
lwtv = rep(1,nrow(pg)); lwtv[2]=3; ltyv = 1+floor(0:5/3)
for(i in 1:nrow(pg)) lines(sv,pmat[,i],col=i,lwd=lwtv[i],lty=ltyv[i])
abline(v=sigest,col="red",lty=4,lwd=3)
abline(v=sigma,col="blue",lty=4,lwd=3)
```



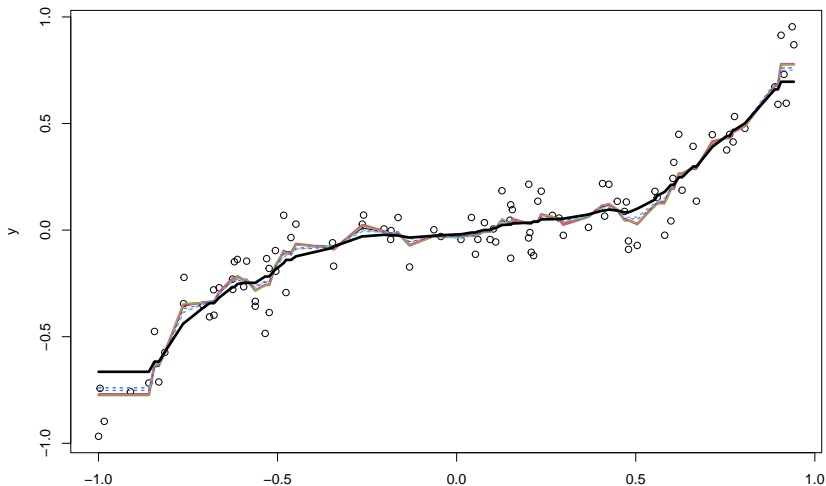


Run BART at our 6 prior settings and a seventh where we use `ntree=5000` and `k=5`, this will give us a smoother function.

```
fmat = matrix(0.0,n,nrow(pg))
for(i in 1:nrow(pg)) {
  bf = wbart(x,y,sigquant=pg[i,1],sigdf=pg[i,2])
  fmat[,i] = bf$yhat.train.mean
}
```

```
#more trees and more shrinkage => smoother f
bf2 = wbart(x,y,ntree=5000,k=5)
```

```
plot(x,y)
for(i in 1:nrow(pg)) lines(x,fmat[,i],col=i,lwd=lwv[i],lty=ltyv[i])
lines(x,bf2$yhat.train.mean,lty=1,lwd=3)
```



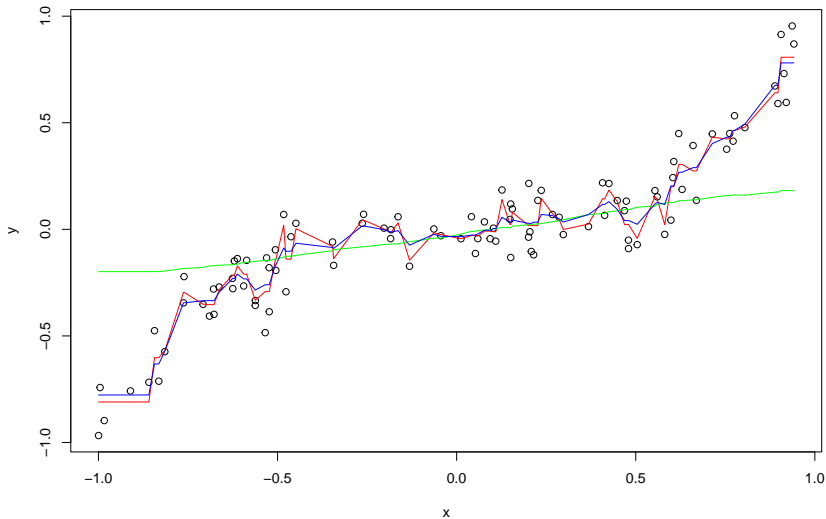
The fit with a lot of trees is indeed smoother, but the other fits are remarkably similar.

Ok, let's mess it up!!

We'll put in  $\sigma$  too small prior, a  $\sigma$  too big prior, and a  $\sigma$  just right prior.

```
bf1 = wbart(x,y,sigest=.01,sigdf=500) #sigma small => overfit  
bf2 = wbart(x,y,sigest=3,sigdf=500) #sigma big => underfit  
bfg = wbart(x,y,sigest=sigma,sigdf=5000) #got sigma right!
```

```
plot(x,y)
lines(x,bfb1$yhat.train.mean,col="red")
lines(x,bfb2$yhat.train.mean,col="green")
lines(x,bfg$yhat.train.mean,col="blue")
```



That worked.

## BART MCMC

$$Y = g(x; T_1, M_1) + \dots + g(x; T_m, M_m) + \sigma z$$

plus

$$\pi((T_1, M_1), \dots, (T_m, M_m), \sigma)$$

First, it is a “simple” Gibbs sampler:

$$\begin{array}{l|l} (T_i, M_i) & (T_1, M_1, \dots, T_{i-1}, M_{i-1}, T_{i+1}, M_{i+1}, \dots, T_m, M_m, \sigma) \\ \sigma & (T_1, M_1, \dots, \dots, T_m, M_m) \end{array}$$

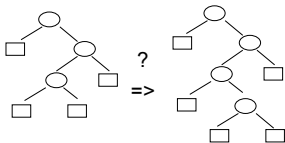
To draw  $(T_i, M_i) | \cdot$

we subtract the contributions of the other trees from both sides to get a simple one-tree model.

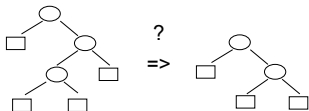
We integrate out  $M$  to draw  $T$  and then draw  $M | T$ .

To draw  $T$  we use a Metropolis-Hastings with Gibbs step.

We use various moves, but the key is a “birth-death” step.



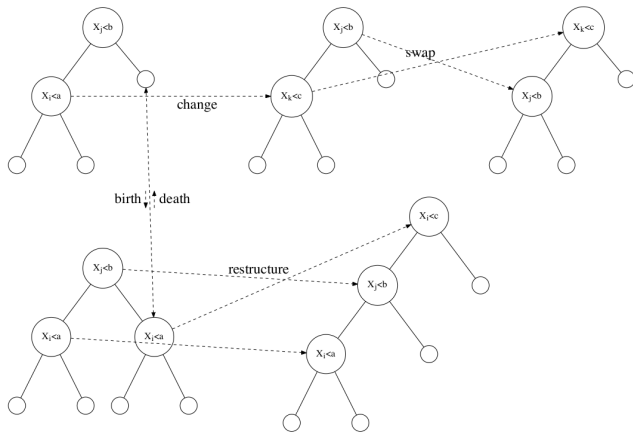
propose a more complex tree



propose a simpler tree

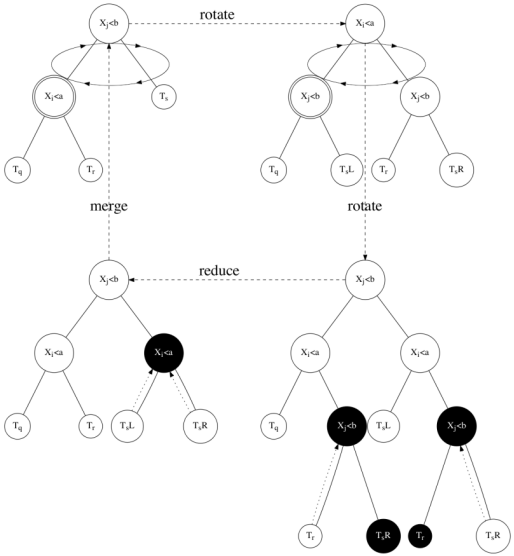
Getting the MCMC to work well with a single tree model is quite difficult.

Original Bayesian CART moves:



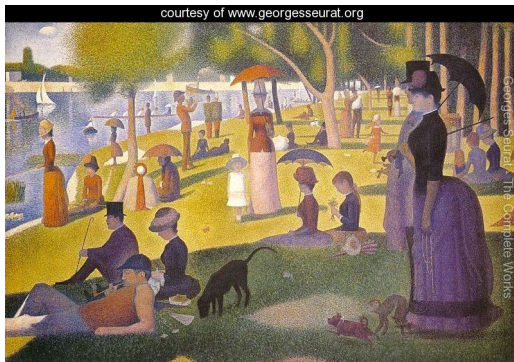


Matt Pratola has developed a set of sophisticated moves that help.



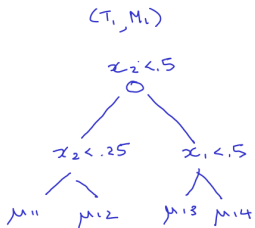
Added up Trees works great !!!!

*But, the BART MCMC works amazingly well !!!!*



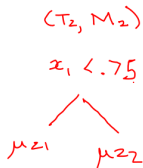
Adding in little bits of fit at a time make for a great model *and a great MCMC !!!!!*.

But, the BART MCMC works amazingly well !!!!



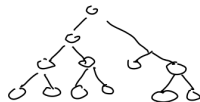
$\mu_{13}$	$\mu_{14}$
$\mu_{12}$	
$\mu_{11}$	

+



$\mu_{21}$	$\mu_{22}$
------------	------------

=



+

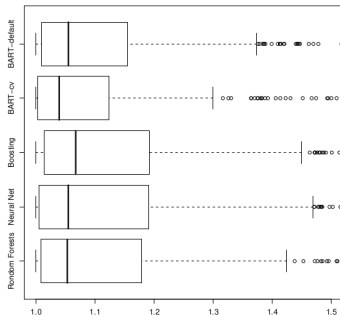
=

$\mu_{13} + \mu_{24}$	$\mu_{14} + \mu_{21}$	$\mu_{14} + \mu_{22}$
$\mu_{12} + \mu_{21}$	$\mu_{21}$	$\mu_{12} + \mu_{22}$
$\mu_{11} + \mu_{21}$	$\mu_{21}$	$\mu_{11} + \mu_{22}$

## Did out of sample predictive comparisons on 42 data sets. (*thanks to Wei-Yin Loh!!*)

- ▶  $p=3 - 65$ ,  $n = 100 - 7,000$ .
- ▶ for each data set 20 random splits into 5/6 train and 1/6 test
- ▶ use 5-fold cross-validation on train to pick hyperparameters (except BART-default!)
- ▶ gives  $20 \times 42 = 840$  **out-of-sample predictions**, for each prediction, divide rmse of different methods by the smallest

- + each boxplots represents 840 predictions for a method
- + 1.2 means you are 20% worse than the best
- + BART-cv best
- + BART-default (use default prior) does amazingly well!!



Using Predict and Controlling What Is Saved

Up to now we have given wbart  $(X, y, X_p)$ .

At every iteration of the MCMC,  $(f, \sigma)$  is updated.

Updating  $f$  means all the  $(T_j, M_j), j = 1, 2, \dots, m$  are updated.

In addition, we evaluate  $f$  at each  $x$  in  $X$  and each  $x$  in  $X_p$ .

We can simplify things and speed things up by thinning these evaluations for both  $x \in X$  and  $x \in X_p$ .

In addition, we can save draws of  $f$  and evaluate them at  $x$  after we have run the MCMC as you would with a “predict” function.

`wbart` has the parameters:

**`nkeeptrain`:**

number of draws of  $f$  to evaluate the training data at. The draws are evenly spaced so that if `ndpost`=1,000 and `nkeeptran`=100, then we would evaluate every 10th draw of  $f$  at the  $x \in X$ .

**`nkeptest`:** same thing for the test data.

**`nkeptestmean`:** number of draws of  $f$  to evaluate the test data in order to compute `$yhat.test.mean`.

**`nkeepreedraws`:** number of draws of  $f$  (the set of  $(T_j, M_j)$ ) to return.

The tree draws are not terribly interpretable, but by saving them we can evaluate  $f$  draws at a new set of  $x$  vectors without rerunning the MCMC.



Let's try it.

```
##simulate data
set.seed(99)
n=5000
p=10
beta = 3*(1:p)/p
sigma=1.0
X = matrix(rnorm(n*p),ncol=p)
y = 10+X %*% matrix(beta,ncol=1) + sigma*rnorm(n)
y=as.double(y)

np=100000
Xp = matrix(rnorm(np*p),ncol=p)
```

```

set.seed(99)
bfk = wbart(X,y,Xp,
            nkeeptrain=200,nkeptest=100,nkeptestmean=500,nkeptreedraws=100)

dim(bfk$yhat.train)
## [1] 200 5000
dim(bfk$yhat.test)
## [1] 100 100000
names(bfk$treedraws)
## [1] "cutpoints" "trees"
str(bfk$treedraws$trees)
## chr "100 200 10\n3\n1 5 85 0.03572449378\n2 0 0 0.1218427186\n3 0 0 0.4938837128\n3\n1 0

```

The trees are stored in one long character string which is not too useful to look at but we can see that there are 100 draws each consisting of 200 trees.

For example, it might be fast to just keep 100 trees:

```
t1 = system.time({bfk1 = wbart(X,y,Xp)})
t2 = system.time({bfk2 = wbart(X,y,Xp,
                             nkeeptrain=0,nkeeptest=0,nkeeptestmean=0,nkeeptreedraws=100)})
t3 = system.time({pred2 = predict(bfk2,Xp,mc.cores=8)})
```

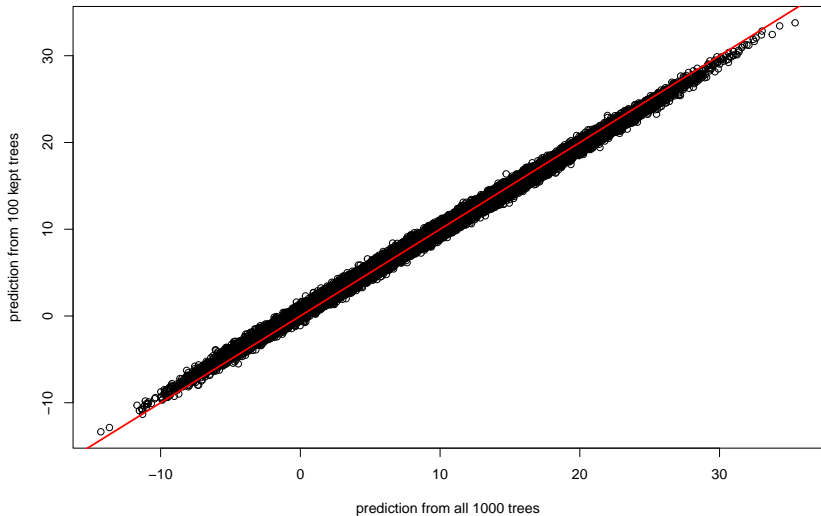
```
dim(pred2)
## [1] 100 100000
t1
## user system elapsed
## 115.793 0.400 116.630
t2
## user system elapsed
## 23.347 0.000 23.344
t3
## user system elapsed
## 72.003 0.033 9.077
```

pred2 has row dimension equal to the number of kept tree draws (100) and column dimension equal to the number of row in  $X_p$  (100,000).

33+18 seconds to get predictions on 100,000 observations using 100 stored trees. while it took 188 seconds to do it the old way.

Let's compare the predictions:

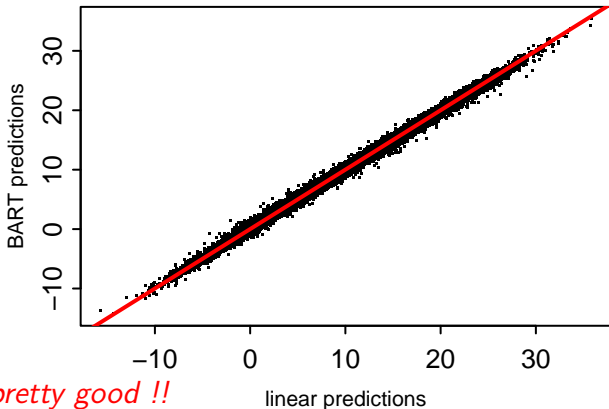
```
plot(bfk1$yhat.test.mean, apply(pred2, 2, mean),  
     xlab="prediction from all 1000 trees", ylab="prediction from 100 kept trees")  
abline(0, 1, col="red", lwd=2)
```



Could be perfectly good as a practical matter.

Let's compare the BART fit to a linear fit  
(we simulated from the linear model).

```
lmf = lm(y~.,data.frame(X,y))
predl = predict(lmf,data.frame(Xp))
plot(predl,bfk1$yhat.test.mean,xlab="linear predictions",ylab="BART predictions",
      cex=.3,cex.axis=.8,cex.lab=.7, mgp=c(1.3,.3,0),tcl=-.2,pch=".")
abline(0,1,col="red",lwd=2)
```



*Looks pretty good !!*

## The Diabetes Example

Let's look at an example with real data.

We'll try the diabetes data which is available at [Hastie, Tibshirani, Wainright website](#).

## **Diabetes data**

These data consists of observations on 442 patients, with the response of interest being a quantitative measure of disease progression one year after baseline. There are ten baseline variables—age, sex, body-mass index, average blood pressure, and six blood serum measurements—plus quadratic terms, giving a total of 64 features.

First used in LARS paper

Tab-delimited diabetes data (text file)

Note that these data are first standardized to have zero mean and unit L2 norm before they are used in the examples.

Let's read in the data.

```
db = read.csv("http://www.rob-mcculloch.org/data/diabetes.csv")
#first col is y, 2-11 are x, 12-65 are xi^2, x_i * x_j (except sex dummy)
xB = as.matrix(db[,2:11]) #x for BART
xL = as.matrix(db[,2:ncol(db)]) #x for lasso, includes transformations
yBL = db$y #y for BART and lasso

dim(db)
## [1] 442 65
names(db[,1:15])
## [1] "y" "age" "sex" "bmi" "map" "tc" "ldl" "hdl" "tch"
## [10] "ltg" "glu" "age.2" "bmi.2" "map.2" "tc.2"
```

So there are 442 observations. The first column is our response and the next 10 columns are the  $x$ 's.

The rest of the columns are the squares and products of the original 10  $x$ 's giving 64 "variables".

For BART we just want a matrix of the 10  $x$ 's.

We will compare BART with just the 10 original vars to glmnet with all 64 predictors.

Note that all the predictors have been standardized.

This is crucial for glmnet and does not matter for BART.



We'll just do some random train/test splits and to compare the out of sample performance of BART to that of the lasso.

```
rmsef = function(y,yhat) {return(sqrt(mean((y-yhat)^2)))}  
nd=30 #number of train/test splits  
  
n = length(yBL)  
ntrain=floor(.75*n) #75% in train each time  
  
rmseB = rep(0,nd) #BART rmse  
rmseL = rep(0,nd) #lasso rmse  
fmatL=matrix(0.0,n-ntrain,nd) #out-of-sample lasso predictions  
fmatB=matrix(0.0,n-ntrain,nd) #out-of-sample BART predictions
```

So, 30 times we will randomly pick 75% of the data to be train and predict on the remaining 25%.

We'll just do some random train/test splits and to compare the out of sample performance of BART to that of the lasso.

```
library(glmnet)
for(i in 1:nd) {
  set.seed(i)
  # train/test split
  ii=sample(1:n,ntrain)
  #y
  yTrain = yBL[ii]; yTest = yBL[-ii]
  #x for BART
  xBTrain = xB[ii,]; xBTest = xB[-ii,]
  #x for lasso
  xLTrain = xL[ii,]; xLTest = xL[-ii,]

  #lasso
  cvL = cv.glmnet(xLTrain,yTrain,family="gaussian",standardize=FALSE)
  bestlam = cvL$lambda.min
  fLTrain = glmnet(xLTrain,yTrain,family="gaussian",lambda=c(bestlam),
                  standardize=FALSE)

  #BART
  bfTrain = mc.wbart(xBTrain,yTrain,xBTest,mc.cores=8,keeptrainfits=FALSE)

  #get predictions on test
  yBhat = bfTrain$yhat.test.mean
  yLhat = predict(fLTrain,xLTest,s=bestlam,type="response")[,1]
  #store results
  rmseB[i] = rmsef(yTest,yBhat); rmseL[i] = rmsef(yTest,yLhat)
  fmatL[,i]=yLhat; fmatB[,i]=yBhat
}
```

Just using default BART!!

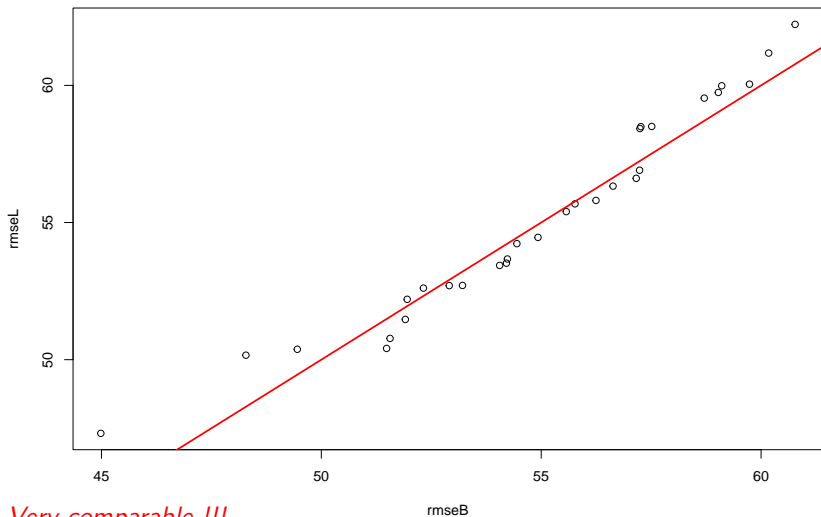
Using parallel version of `wbart`.

The basic arguments for `mc.wbart` are the same as in `wbart`.

Data is already standardized so don't have `standardize=FALSE` in `glmnet`.

Let's compare the out of sample RMSEs.

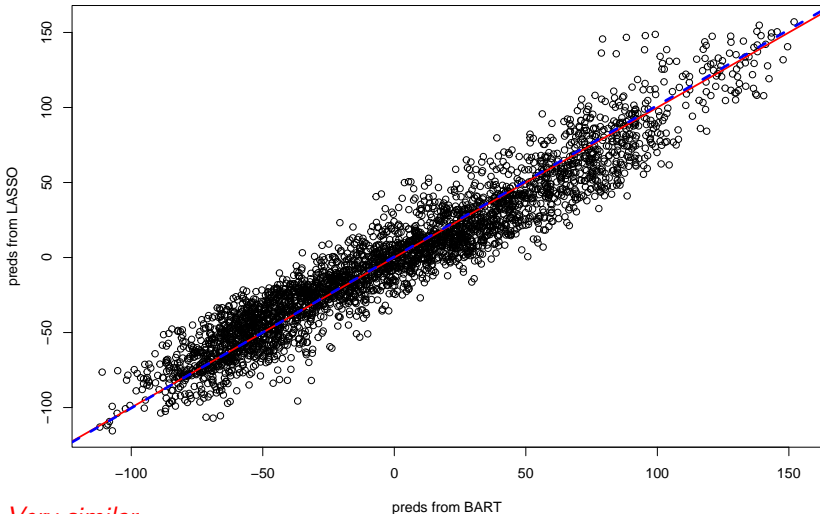
```
qqplot(rmseB,rmseL)  
abline(0,1,col="red",lwd=2)
```



*Very comparable !!!.*

Let's compare the out of sample predictions.

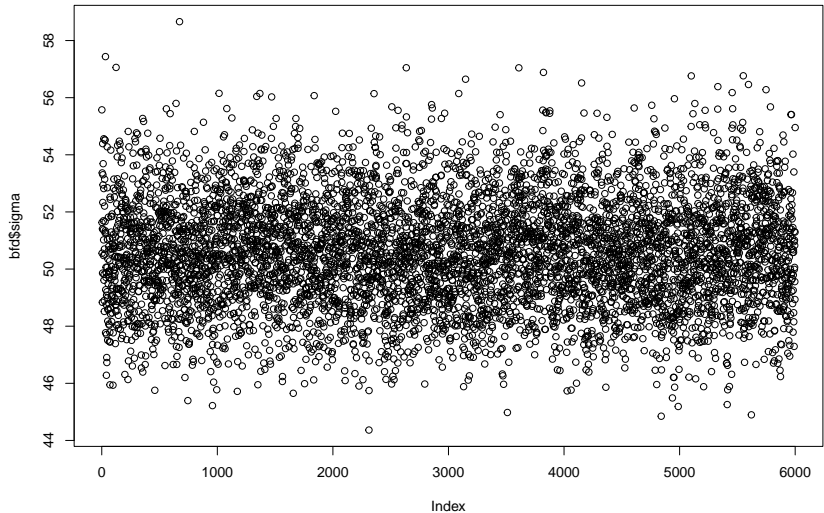
```
plot(as.double(fmatB),as.double(fmatL),xlab="preds from BART",ylab="preds from LASSO")  
abline(0,1,col="red",lwd=2)  
lmtemp = lm(B~L,data.frame(B=as.double(fmatB),L=as.double(fmatL)))  
abline(lmtemp$coef,col="blue",lty=2,lwd=3)
```



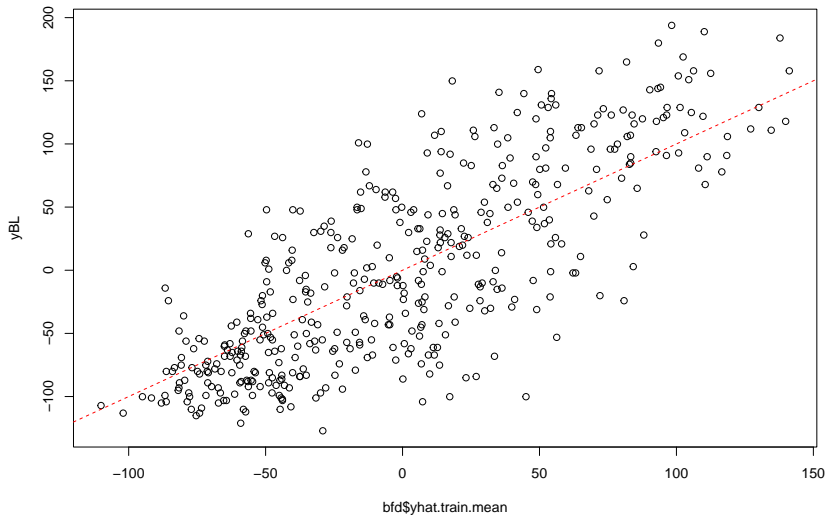
Now that we know the default prior *may* be giving reasonable results let's look at a longer run.

```
bfd = wbart(xB,yBL,nskip=1000,ndpost=5000) #keep it simple
```

```
plot(bfd$sigma)
```

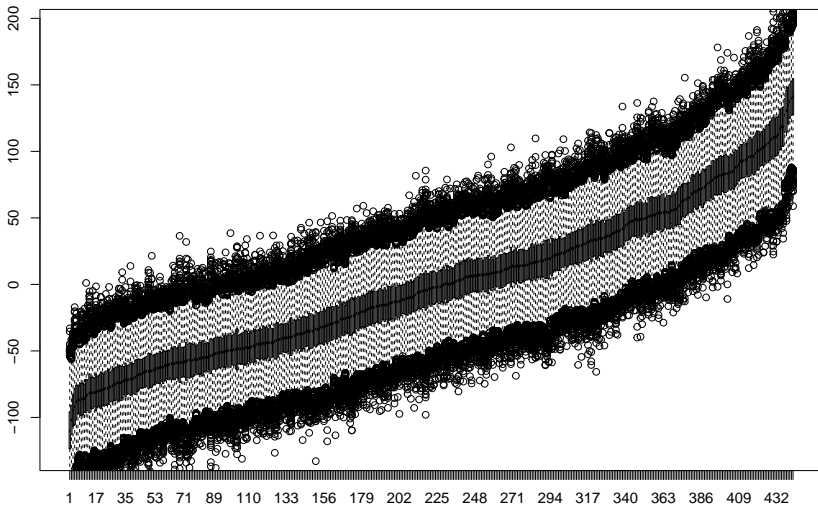


```
plot(bfd$yhat.train.mean,yBL)
abline(0,1,col="red",lty=2)
```



$\sigma$  is about 50 !!

```
ii = order(bfd$yhat.train.mean)
boxplot(bfd$yhat.train[,ii],ylim=range(yBL))
```



*A lot of uncertainty !!!!*



- ▶ Very automatically, BART gives us good out of sample results, did not have to choose transformations to include !!
- ▶ BART gave us the uncertainty!!!