Variable Selection with BART

Carlos Carvalho, Ed George, Richard Hahn and Rob McCulloch

5/31/2023

Variable Section Using BART

We present two approaches.

In our first approach we use the R package `BART` which is on CRAN.
This approach examines the BART draws to see which variables are being used in the decision trees.

In the second approach we use the R package `nonlinvarsel` which is available at http://www.rob-mcculloch.org.
This approach searches for variable subsets which can be used to approximate predictions based on the information in the complete set of variables.
See "A Utility Based Approach to Variable Selection" below.

Variable Selection Using the BART R Package

In this note we will illustrate the use of the R package `BART` to do variable selection.

The basic advantages of this approach are the effective stochastic search of the BART algorithm and the simple variable usage counts.
There is no need to elaborate the model.

The only tricky aspect is that to get a good idea of the variable usage, it is recommended that fewer trees be used in the sum of trees model.

The Data

To keep things simple we will use the time honored Boston Housing data.

Each observation corresponds to a region in the Boston area.
The response is the median house price in the region.
The explanatory features measure various things about the region.

```
library(MASS)
attach(Boston)
names(Boston)
##  [1] "crim"    "zn"      "indus"   "chas"    "nox"     "rm"      "age"
##  [8] "dis"     "rad"     "tax"     "ptratio" "black"   "lstat"   "medv"

y = Boston$medv
x = Boston[,1:13]
p = ncol(x)
```

Running BART

First we load the `BART` R package and set the seed.

We let `burn = 10000` burn-in draws and `nd = 10000` kept draws.
For the kept draws of the function, f_d, $d = 1, 2, \ldots, \text{nd}$, BART will evaluate $f_d(x)$ for train and test (if supplied) values of x.

We set the seed.

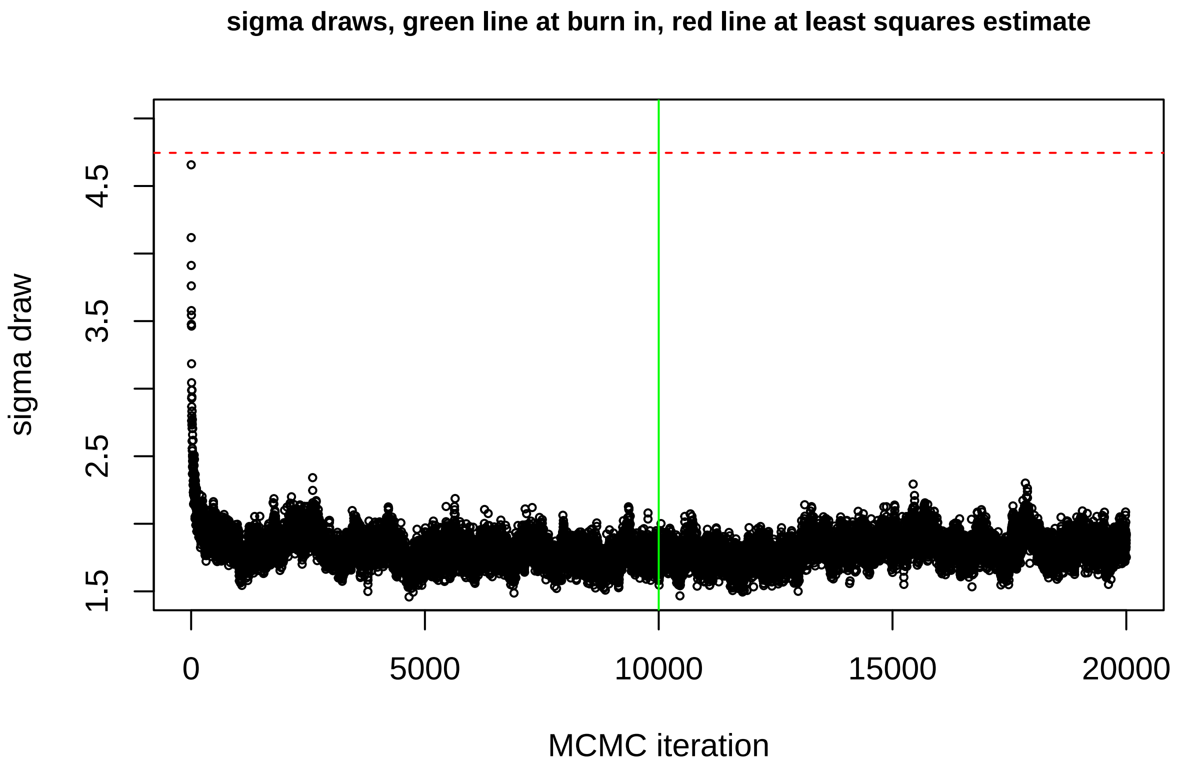
```
library(BART)
## Loading required package: nlme
## Loading required package: nnet
## Loading required package: survival

set.seed(99) # Wayne Gretzky
burn=10000; nd=10000
```

Now we run BART using the default prior and model settings.

```
# run default BART
bf = wbart(x,y,nskip=burn,ndpost=nd,printevery=5000) #print progress every 5000th MCMC iteration
```

The `sigma` component of `bf` has all $10^4 + 10^4$ draws of `sigma`, that is, we include burn-in.
This way we can check that we chose a reasonable value for the number of burn-in draws.

```
# linear model fit
lmf = lm(medv~.,Boston)
plot(bf$sigma,ylim=c(1.5,5),xlab="MCMC iteration",ylab="sigma draw",cex=.5)
abline(h=summary(lmf)$sigma,col="red",lty=2) #least squares estimates
abline(v = burn,col="green")
title(main="sigma draws, green line at burn in, red line at least squares estimate",cex.main=.8)
```

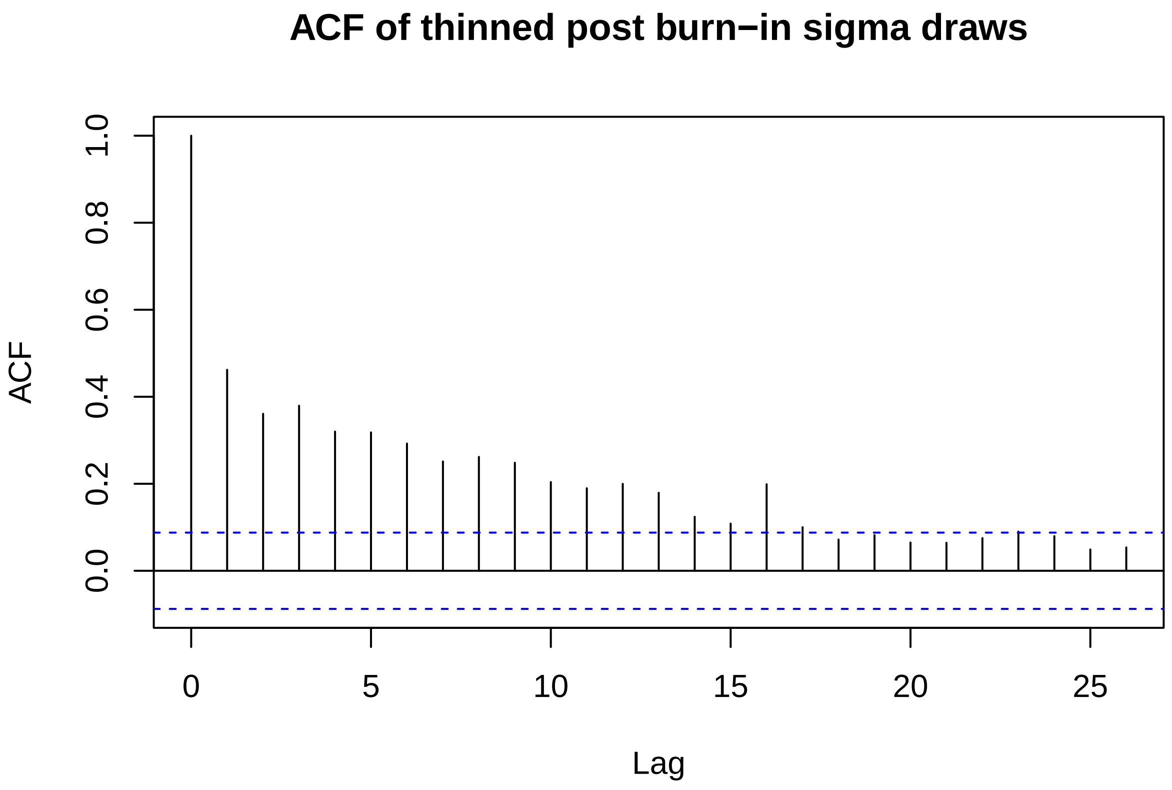


Our burn-in choice looks reasonable in that the `sigma` draws have “settled down” by the 1,000 the draw.
The draws exhibit substantial dependence, but seem like a reasonable exploration of plausible `sigma` values given the data.
The post burn-in `sigma` values suggest BART has found substantially more fit on the training data than the

simple linear model.

For example, if we thin to every 20th post burn-in draw, we get a non-neglible but reasonable ACF.

```
thin = 20
ii = burn + thin*(1:(nd/thin))
acf(bf$sigma[ii],main="ACF of thinned post burn-in sigma draws")
```



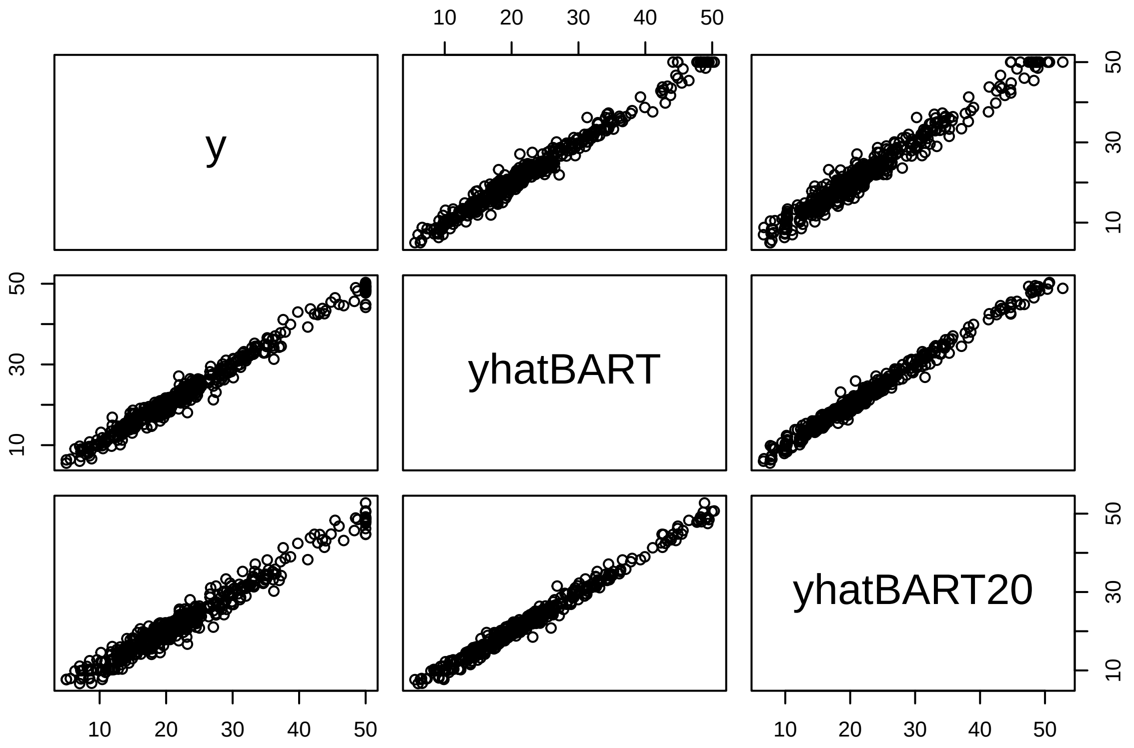
The default `BART` run uses 200 trees in the sum-of-trees model.

To get a good sense of the variable selection, we will see that it helps to use fewer trees.
Let's run BART, but just use 20 trees in the sum.

```
# run BART with 20 trees in the sum
set.seed(99)
bf20 = wbart(x,y,nskip=burn,ndpost=nd,ntree=20,printevery=5000)
```

Let's compare the in-sample fits from the two BART runs.

```
fitmat = cbind(y,bf$yhat.train.mean,bf20$yhat.train.mean)
colnames(fitmat) = c("y","yhatBART","yhatBART20")
pairs(fitmat)
```



```
print(cor(fitmat))
##                   y  yhatBART yhatBART20
## y          1.0000000 0.9884595  0.9800918
## yhatBART   0.9884595 1.0000000  0.9924667
## yhatBART20 0.9800918 0.9924667  1.0000000
```

We see that both BART and BART20 fit the data (in-sample) very well.

In general, the BART20 may not may not fit quite as well, but we will see in the next section that it sharpens our variable selection to use fewer trees.

Variable Selection

The component `varcount` of our BART fit the provides information about which variables are used in the trees.

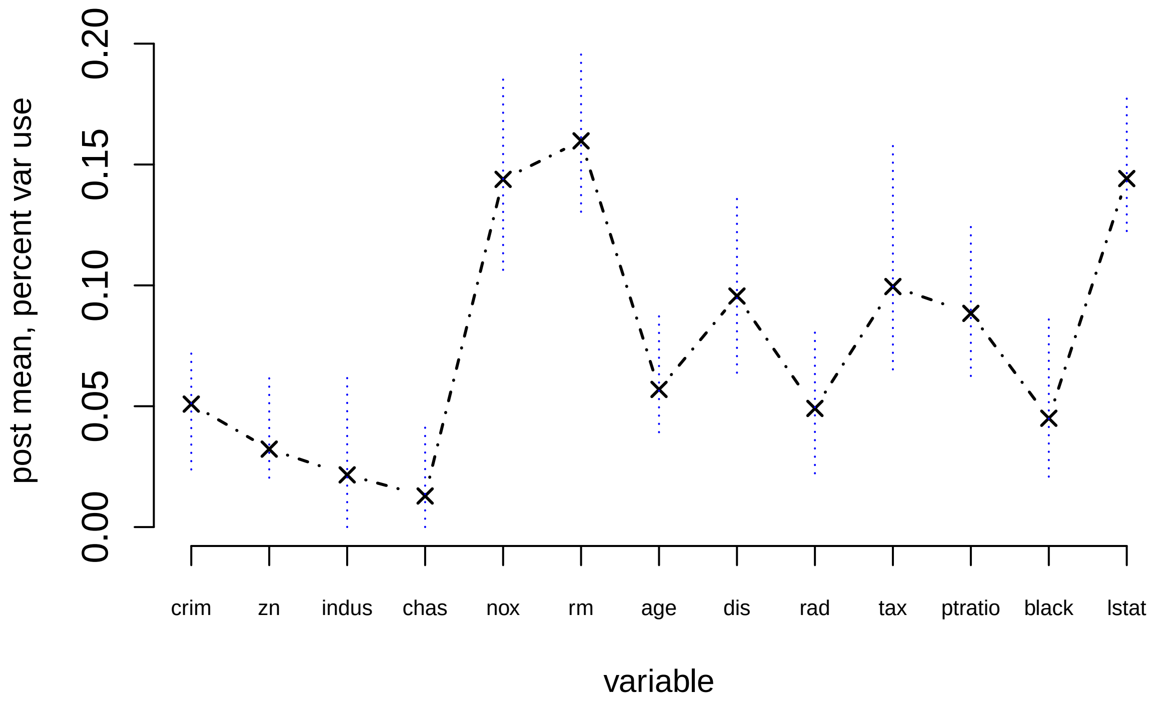
```
dim(bf20$varcount)
## [1] 10000    13
```

Each row of varcount corresponds to one of our 10^4 kept post burn-in MCMC draws of the sum of tree model.

Each column corresponds to one of our 13 `x` variables.

We just have to summarize the values in `varcount`.

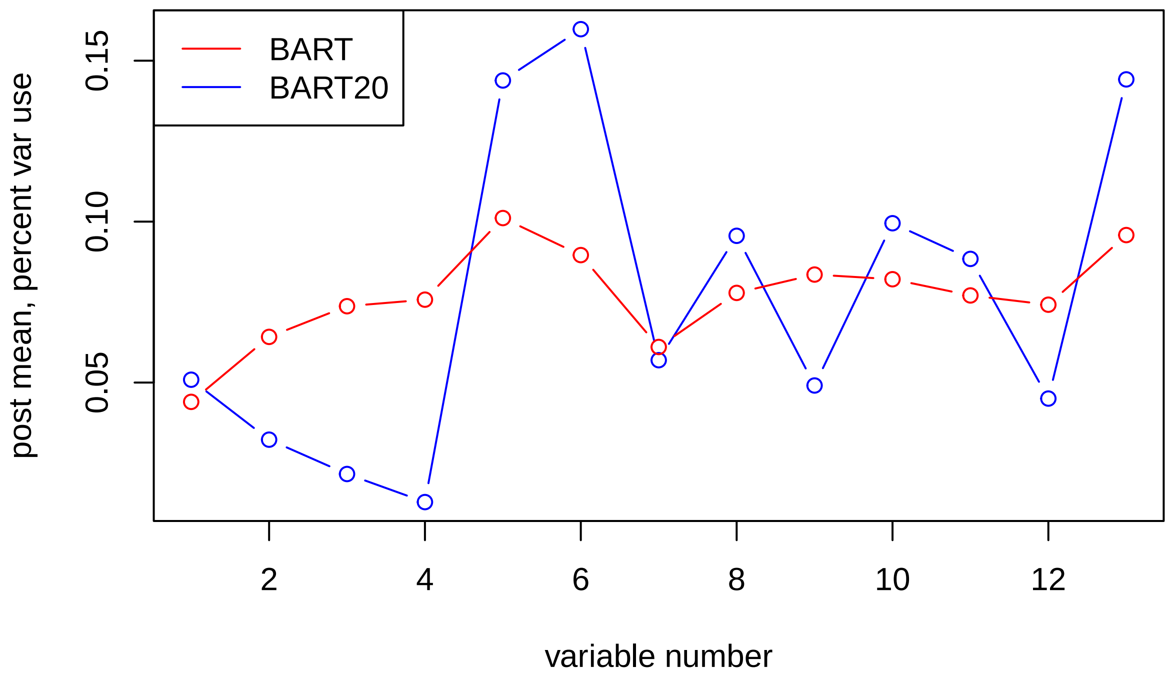
```
#compute row percentages
percount20 = bf20$varcount/apply(bf20$varcount,1,sum)
# mean of row percentages
mvp20 =apply(percount20,2,mean)
#quantiles of row percentags
qm = apply(percount20,2,quantile,probs=c(.05,.95))

print(mvp20)
##        crim          zn       indus        chas         nox          rm         age
## 0.05088151 0.03225757 0.02158979 0.01286549 0.14385611 0.15979102 0.05695193
##        dis        rad        tax    ptratio      black      lstat
## 0.09558875 0.04908700 0.09949849 0.08841762 0.04502477 0.14418993

rgy = range(qm)
plot(c(1,p),rgy,type="n",xlab="variable",ylab="post mean, percent var use",axes=FALSE)
axis(1,at=1:p,labels=names(mvp20),cex.lab=0.7,cex.axis=0.7)
axis(2,cex.lab=1.2,cex.axis=1.2)
lines(1:p,mvp20,col="black",lty=4,pch=4,type="b",lwd=1.5)
for(i in 1:p) {
   lines(c(i,i),qm[,i],col="blue",lty=3,lwd=1.0)
}
```



We see that `nox`, `rm`, and `lstat` get used more in the trees than the other variables.

Let's compare the BART default run to the BART20 run.

```
percount = bf$varcount/apply(bf$varcount,1,sum)
mvp = apply(percount,2,mean)
plot(mvp20,xlab="variable number",ylab="post mean, percent var use",col="blue",type="b")
lines(mvp,type="b",col='red')
legend("topleft",legend=c("BART","BART20"),col=c("red","blue"),lty=c(1,1))
```



We see that the default BART run uses the variable with heavy shrinkage, so that while a variable is in a tree, the corresponing bottom node mean is shrunk heavily to 0 so that it is of no real importance.

With 20 trees, the means are shrunk less heavily so a variable tends to only come in when it actually does something!!

A Utility Based Approach to Variable Selection

In this section we illustrate a utility based approach to variable selection using BART inference.

The R package is available at http://www.rob-mcculloch.org/.

This approach uses any inference for a nonlinear function $\hat{y} = \hat{f}(x)$ and then seeks to find a subset of the variables in x such there there is a nonlinear function such that that $g_S(x_S) \approx \hat{f}(x)$ where x_s denotes the variables in the subset. The approximation should be reasonable close *as a practical matter*. Practical significance is emphasized over statistical significance, instead of the other way around.

If we are able to find such a subset of variables then clearly we can make predictions based on the subset where are as good as those using the full information in x and $\hat{f}$.

Many modern statistical/Machine Learning methods give us remarkably interesting $\hat{f}$ given training data and the method may be applied with any such $\hat{f}$. In addition, BART gives us draws $f_d, d = 1, 2, \ldots, D$ from the posterior of f. In this case we can assess our uncertainty by looking at the values $d(f_d, g_S), d = 1, 2, \ldots, D$ where d is a metric chosen to capture the practical difference in predictions.

Subset Search

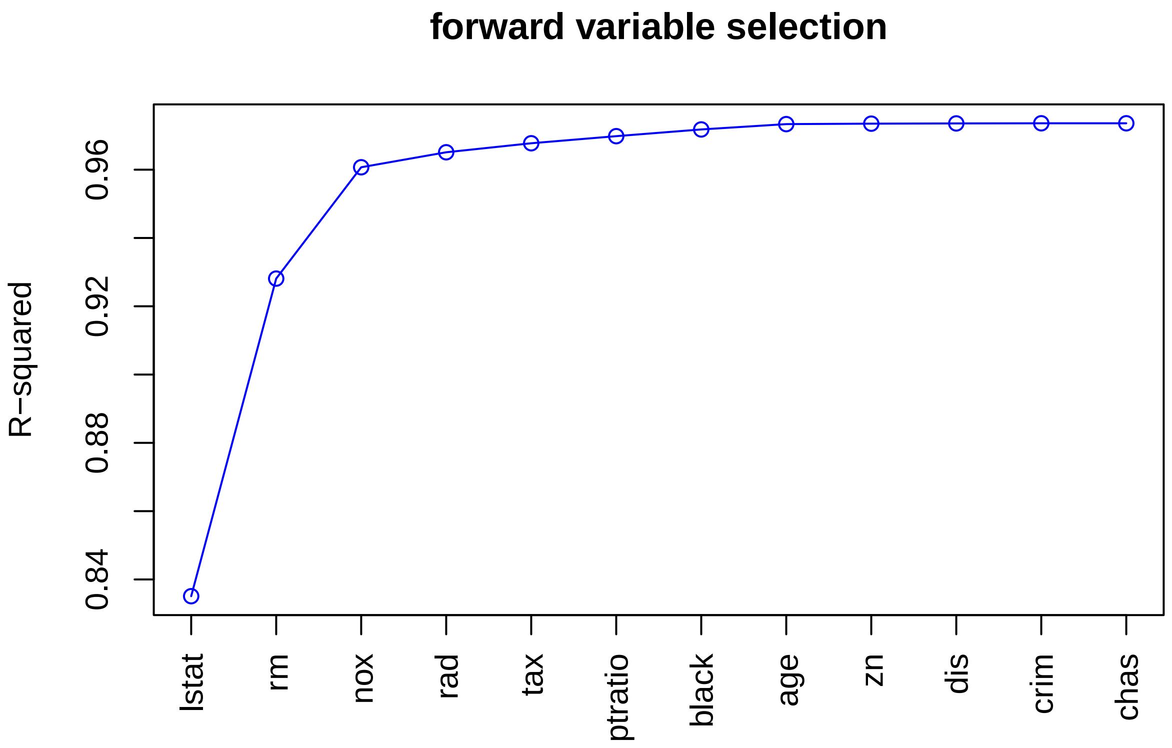
Let's load the R package.

```
library(nonlinvarsel)
## Loading required package: foreach
```

This approach to variable selection does not require us to tailor the prior or the number of trees.
We can use our default BART inference.

First we will the forward greedy search method for finding variable subsets.
We call the function `vsf` from the R package. We give the function the `x` values and the values of $\hat{f}$ evaluated at the rows of `x`.
Note that the `x` need not be the training values. The user should choose values which are relevant to actual predictions to be made in application.
In this case we are using `bf$yhat.train.mean` where `bf` is our previous BART run using the default prior and `yhat.train.mean` is $\hat{f}(x)$ for x in the training data and $\hat{f}$ the posterior mean from BART.

```
vsfr = vsf(x,bf$yhat.train.mean)

plot(vsfr)
```



On the horizontal axis we have the variables as the come into our forward greedy search.

On the vertical axis we have R^2, the squared correlation between $g_S(x_s)$ and $\hat{f}(x)$ for our chosen x values (in this case the training x).

Here are the R^2 values:

```
print(vsfr)
##     lstat        rm       nox       rad       tax   ptratio     black       age
## 0.8350978 0.9281046 0.9607106 0.9651039 0.9677454 0.9698201 0.9717911 0.9733491
##        zn       dis      crim      chas     indus
## 0.9734810 0.9735532 0.9735887 0.9735887 1.0000000
```

The strong suggestion is that we can predict just as well with the first three variables

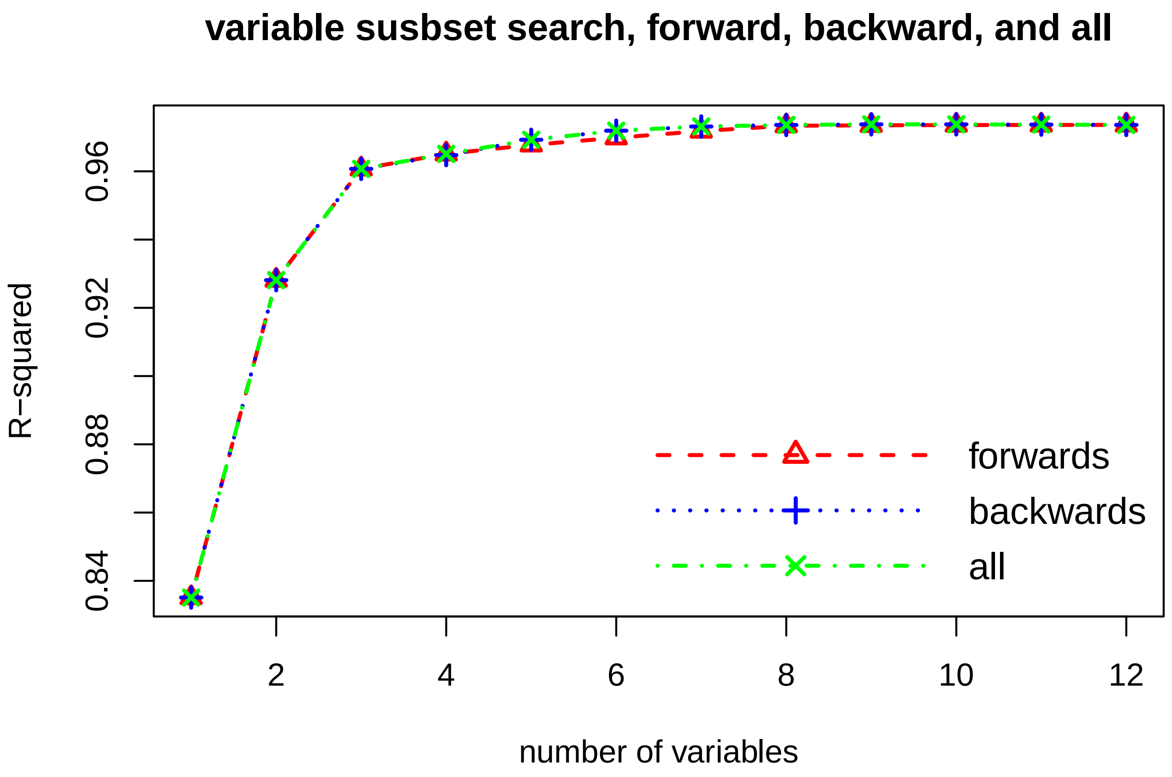
For small numbers of variables we can also try a backwards search in which variables are eliminated one at a time and search over all possible subsets.

```
## backward variable selection
vsbr = vsb(x,bf$yhat.train.mean)

## all variable selection
```

```
vsar = vsa(x,bf$yhat.train.mean)
```

We can then plot all three searches:

```
## plot all three searches
plotfba(vsfr,vsbr,vsar)
```



In this case all searches give similar results in terms of our ability to find small subsets that predict as well as our full set of x variables.

To get the actual $g_S(x_S)$ values for as set of x it may help to fit BART to each found subset.
Note this is at most p subsets (not 2^p) where p is the number of x variables.

We can do this computaion in parallel.

```
ncores=4
library(doParallel)
## Loading required package: iterators
## Loading required package: parallel
registerDoParallel(cores=ncores)
cat("ncores: ",ncores,"\n")
## ncores:  4
cat("number of workers is: ",getDoParWorkers(),"\n")
## number of workers is:  4
```

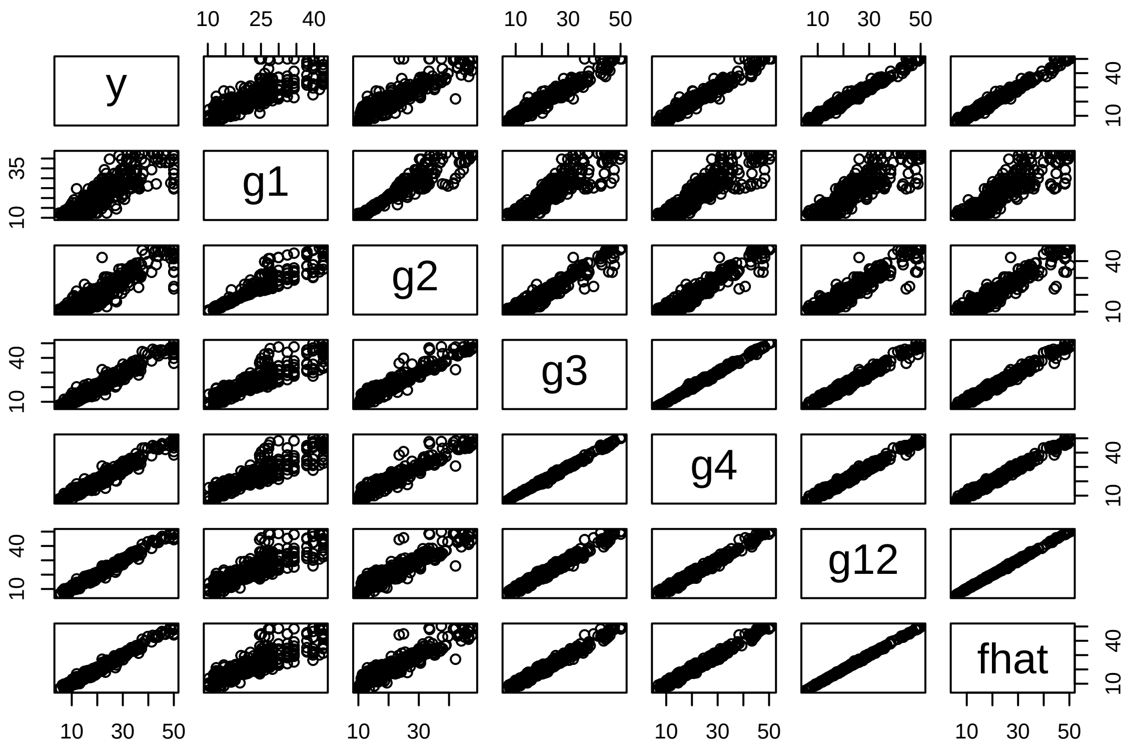
Now we can use the function `bartSubs` to fit the BART using each subset and get predictions (posterior means) for a chosen set of x.

Here we use the subsets from the all possible subsets search (`vsar$vL`) and use all the x in the training data.

```
bsubs = bartSubs(x,y,x,vsar$vL,nskip=burn,ndpost=nd)
```

To see how well are subsets are working, let's plot the predictions using subsets of size 1,2,3,4 and 12 along with the full BART fit and y.

```
print(dim(bsubs))
## [1] 506  12
fmat = cbind(y,bsubs[,c(1:4,12)],bf$yhat.train.mean)
colnames(fmat) = c("y",paste0('g',c(1:4,12)),'fhat')
pairs(fmat)
```



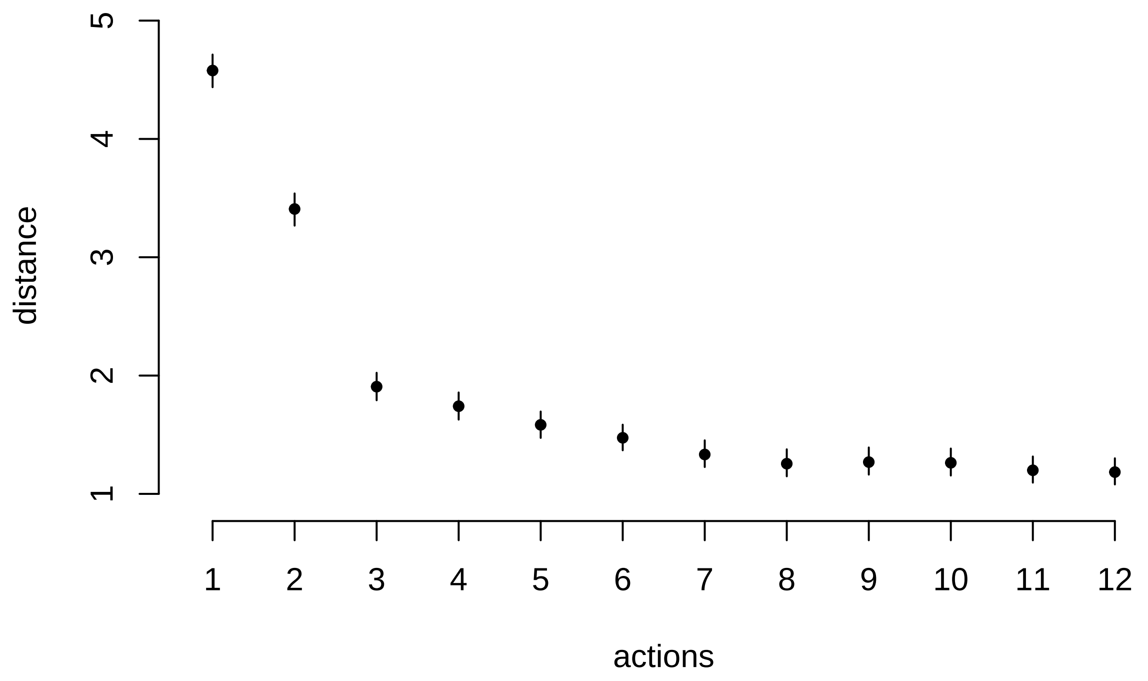
We see that with just three or 4 variables we fit as well as with them all.

Assessing Uncertainty

To assess uncertainty, we look at the posterior distribution of our approximation error. We use the approximations from the BART refits to all subsets search and root mean squared error (and the training data in this case) as a measure of approximation error.

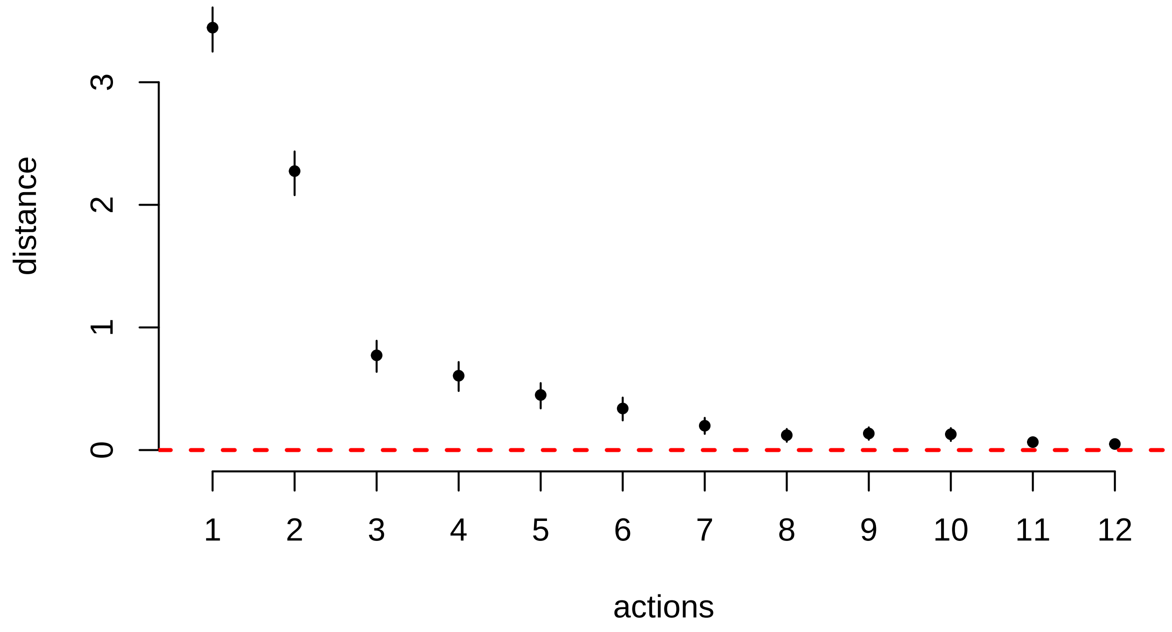
```
sp = sumpost(bf$yhat.train,bsubs,bf$yhat.train.mean,distrmse)
## nd,n,p:  10000 506 12
```

```
plot(sp,diff=FALSE)
```



It can also be helpful to look at the difference between the approximation error using a variable subset and the approximation error using the full subset.

```
plot(sp)
```



```
summary(y)
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    5.00   17.02   21.20   22.53   25.00   50.00
```

We see that, as a practical matter, with high posterior probability, we can get good predictions using just three or at most seven, of the variables.

Here are the variable subsets from the all subsets search using 3, 4 and 7 variables.

```
names(x)[vsar$vL[[3]]]
## [1] "nox"   "rm"    "lstat"
names(x)[vsar$vL[[4]]]
## [1] "nox"   "rm"    "rad"   "lstat"
names(x)[vsar$vL[[7]]]
## [1] "nox"     "rm"      "age"     "dis"     "ptratio" "black"   "lstat"
```

And forwards:

```
names(x)[vsfr$vL[[3]]]
## [1] "lstat" "rm"    "nox"
names(x)[vsfr$vL[[4]]]
## [1] "lstat" "rm"    "nox"   "rad"
names(x)[vsfr$vL[[7]]]
## [1] "lstat"   "rm"      "nox"     "rad"     "tax"     "ptratio" "black"
```