Fitting the fit, variable selection using surrogate models and decision analysis, a brief introduction and tutorial

Carlos Carvalho
McCombs School of Business, University of Texas at Austin
and
Richard P. Hahn
School of Mathematical and Statistical Sciences, Arizona State University
and
Robert McCulloch
School of Mathematical and Statistical Sciences, Arizona State University

April 21, 2020

Abstract

This paper describes a practical procedure for Bayesian variable selection in non-linear regression and classification models. A first stage model is fit in which all variables are included. Typically, this first stage will not include the prior belief that only a small subset of variables is needed, but it may. Given this first stage fit, we look for functions of variable subsets which approximate the predictions from the first stage fit well. A computationally efficient surrogate model is used to search for approximating functions which depend on low numbers of predictors. Rather than assuming there is some true sparcity, we seek sparse approximations to the non-sparse truth. In the case that our first stage fit involves a Bayesian assessement of the uncertainty, we use this to gauge the uncertainty of our approximation error. If we learn that, with high probability, we can obtain a good approximation to the non-sparse truth using a subset of the variables, we deem that subset to be of interest. We demonstrate the procedure in empirical examples involving prediction and classification and simulated examples.

Keywords: BART, nonlinear, Machine Learning

1 Introduction

Model selection, and more particularly *variable selection* has been, and continues to be, a major focus of statistical research and practice. In the case of the linear model, there is a huge literature on variable selection. Researchers often want to know which variables are most important.

In the case of nonlinear modeling, methods such as random forests and deep neural nets have been proven to be remarkably effective. While it is common to hear complaints that these methods are "black box" and "uninterpretable", these methods typically provide measures of *variable importance*.

Carvalho et al. (2020) (CHM) develop an alternative approach to variable selection for nonlinear models which is generally applicable. Rather than building on the dubious notion that a variable has a meaningful importance on its own, CHM look for subsets of variables such that a function of the subset can approximate a non-sparse fit well *as a practical matter*. The CHM development for nonlinear models extends the ideas in Hahn & Carvalho (2015) designed for linear models. This paper provides an introduction to CHM and a tutorial on the usage of the corresponding `R` package `nonlinarsel`.

A simple way to get a feeling for the CHM approach is to think of the variable selection problem as one of function approximation. We outline our approach for the problem where the response Y is numeric, but the ideas extend nicely to the general case. Our basic statistical model is that we seek to know $E(Y \mid x)$ where Y is a variable we wish to predict given the information in a vector of predictor variables x. The vector x may contain many variables and variable selection involves choosing a subset of variables S such that x_S (the subset of predictor variables indicated by S) enables us to predict "almost" as well as the full x.

It is assumed that a first stage inference has successfully uncovered a trusted estimate $\hat{f}$ such that $E(Y \mid x) \approx \hat{f}(x)$. The approach then seeks subsets S and surrogate functions γ_S such that $\gamma_S(x) \approx \hat{f}(x)$ for x of interest. Crucially, γ_S only depends on x through the subset x_S. Can we find a function which only uses a subset of the variables and, as a practical matter, is an effective surrogate in the sense that it approximates the correct function well?

We find that this approach leads to a method which reliably finds useful subsets. In the case where Bayesian inference about f is available, we report the posterior distribution of the approximation error to gauge the uncertainty.

Our approach consists of the following steps:

1. Obtain an inference for f.
2. Seek approximations to the inferred f which only use a subset of the variables: *fit the fit !!*
3. Quantify uncertainty using the posterior distribution of the approximation error.

For our step (1) inference we typically use Bayesian Additive Regression Trees (BART (Chipman et al. (2010))). BART has the advantage that in typical statistical investigations, it finds a good fit without a lot of tuning. This is what is needed for steps (1) and (2) above. Our method can also be used with other approaches to nonlinear function estimation such as neural nets. We find that BART is easy to use and allows for assessment of the uncertainty through the posterior as represented by Markov Chain Monte Carlo (MCMC) draws which is done in step (3). For large data sets we often use XBART (Hahn & He (2020)).

For step (2) we use large trees fit to the fit from step (1). Given a subset of variables S, "fit the fit" means we work with the data $(x_S, \hat{f}(x))$ rather than the data (x, y). We fit a large tree to this data and the large tree is our $\gamma_S(x)$. This gives us a fast way to search through variable subsets without having to refit BART or a some other complex modeling strategy.

Note that when fitting large trees to data (x, y), we need to worry about over fitting. We do not have this concern when fitting the fit $\hat{f}$. We are using the large tree as a simple, general, approach to function approximation, as opposed function inference. We illustrate our use of big trees in Section 4.

In practice our formulation of the problem leads to an important conceptual simplification. We simply seek effective surrogates γ_S, depending on a subset S of the x variables, such that $\gamma_S(x) \approx \hat{f}(x) \approx E(Y \mid x)$. We develop simple search mechanisms to find good γ_S.

But, we do not need any guarantee that we have found the optimal γ_S. If we find a good one we are in business. If we find a better one later that is fine too. There are no tricky inferential issues related to estimation of a true function γ_S or the true subset S. Given any found γ_S, we can simply assess our uncertainty about it's effectiveness as a surrogate based on the inference in step (1). This is step (3).

Note that an essential feature of our approach, which distinguishes it from others, is that the investigator must choose the set x values at which $f(x)$ is needed. Because our method focuses on the practical value of variable selection, the user must specify what the model is going to be used for and this means we need to decide on a set x for which we need $E(Y \mid x) = f(x)$.

Additionally, since steps (2) and (3) are driven by the goal of approximation, the choice of metrics that are used to measure how well an approximation is working is a fundamental part of the procedure. Often we use standard statistical metrics, but we emphasize the problem dependent metrics motivated by the practical problem being addressed can easily be incorporated into the steps (2) and (3).

CHM provide a more formal development of our approach based on Bayesian decision theory and discuss how to it relates to other approaches. In addition, an extended example is discussed.

In Section 2, we present the cars data example which looks at predicting the prices of used cars. We use the cars data as our running example in Sections 3 to 5. In Section 3 we illustrate using BART to do our step (1) inference. In Section 4 we show how we can use big trees to quickly approximate nonlinear functions. In Section 5.1 we illustrate step (2) and find subsets of the variables that approximate our our inference in step (1) well. In Section 5.2 we illustrate step (3), showing how BART draws from the posterior can be used to assess our uncertainty. In Section 5.3 we explore an alternative approach for finding surrogate models. In Section 5.4 we compare our inferential conclusions with the more traditional out-of-sample analysis. In Section 6 we look at some simulated data. In Section 7 we look at data on penalties in hockey games where the goal is to predict which team will get the next penalty in a National Hockey League game (Abrevaya & McCulloch (2014)). Note that the hockey data is a classification problem whereas the previous problems were all regression problems in which we seek to predict a numeric response. Finally, in Section 8 we conclude.

2 The Cars Data

In this section we introduce the used cars data set. The goal of the study was to predict the sales price of a used car given characteristics of the car.

Let's read in the data and have a quick look at it.

```
cdat = read.csv("http://www.rob-mcculloch.org/data/susedcars.csv")
dim(cdat)
## [1] 1000    7
names(cdat)
## [1] "price"        "trim"         "isOneOwner"   "mileage"      "year"
## [6] "color"        "displacement"
```

Each of the 1000 observations corresponds to a car. The response y is the variable `price` which is the price the used car was sold for in dollars. To give our example an generic feel, we will call the response `y`. Let's also change the unit to thousands of dollars.

```
y = cdat[,"price"]/1000 #unit is now thousands of dollars
summary(y)
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.995  12.995  29.800  30.583  43.992  79.995
```

Note that these cars are used Mercedes and hence still quite expensive.

The rest of the variables are our explanatory x variables. We note that two interesting variables in `x` are `year` and `mileage` which is the model year of the car and the mileage (number of miles). So, the bigger `year` is, the newer the car is. The other variables are categorical.

```
summary(cdat[,"year"])
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1994    2004    2007    2007    2010    2013
summary(cdat[,"mileage"])/1000 #thousands of miles
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.997  40.133  67.919  73.652 100.138 255.419
cor(cdat[,c("price","year","mileage")])
##              price       year    mileage
## price    1.0000000  0.8805373 -0.8152458
## year     0.8805373  1.0000000 -0.7447292
## mileage -0.8152458 -0.7447292  1.0000000
```

An overly simple interpretation of the correlations above suggests that new cars with low mileage sell for more, a plausible result.

Several of the variables in `x` are categorical. To run BART, using the `BART` package, we will need to express these factors as dummy variables. To do this, we load the `nonlinvarsel` library and use the function `nonlinvarsel::bartModMat`. We will use the generic name `x` for the resulting set of variables.

```
library(nonlinvarsel)
cdat$mileage = cdat$mileage/1000
x = bartModMat(cdat[,-1])
```

The matrix `x` now includes all of the numerical explanatory variable as well as the categorical variables expanded into dummy indicator variables.

```
dim(x)
## [1] 1000   15
```

Our original 6 variables resulted in 15 variables after the dummies are created. Note that for use in BART, a categorical variables with `k` levels is expressed using all `k` dummies (one for each level of the variable) unlike in linear regression where we use `k-1` dummies.

As discussed in Section 1, we need to choose a set of x vectors and which to assess our approximations to $E(Y \mid x) = \hat{f}(x)$. Ideally, the user thinks carefully about what the set of x for which predictions will be needed. As a practical matter convenience choices are useful. We will use `xp` in our `R` code to denote the matrix whose rows consist of the x vectors at which we may need to predict. The most obvious convenience choice is `xp = x` where `x` is the observed training data. A second obvious choice is to randomly partition the data into a train/test split. This way we are careful to assess our uncertainty on x cases not used in estimation. For nonparametric methods such as BART, this can make a difference. Note that our motivation for a train/test split is very different from the usual motivation based on estimation of out of sample loss.

Let's use a train/test split with 75% of the data in the training sample.

```
set.seed(99)
n = length(y)
ii = sample(1:n,floor(.75*n))
xp = x[-ii,] ; yp = y[-ii]
x = x[ii,]; y = y[ii]
print(dim(x)) ; print(length(y))
## [1] 750  15
## [1] 750
print(dim(xp)) ; print(length(yp))
## [1] 250  15
## [1] 250
```

So, `(x,y)` is the training data and `(xp,yp)` is the test data.

3 Cars Data, BART Inference for f

In Section 1 we outlined how our approach to variable selection consists of three steps. In step (1) we obtain an inference for $E(Y \,|\, x) = f(x)$.

In this section we illustrate the use of Bayesian Additive Regression Trees (BART, Chipman et al. (2010)) to get inference for f with the cars data as our example (Section 2).

The BART model and Markov Chain Monte Carlo algorithm provide an inference for

$$Y = f(x) + \epsilon, \quad \epsilon \sim N(0, \sigma^2),$$

where we assume that the errors are iid $N(0, \sigma^2)$.

We will use the `R` package `BART`(see Sparapani et al. (2020) and McCulloch et al. (2019)) that implements the BART algorithm for the model above (as well as many others.)
We use the function `BART::wbart`:

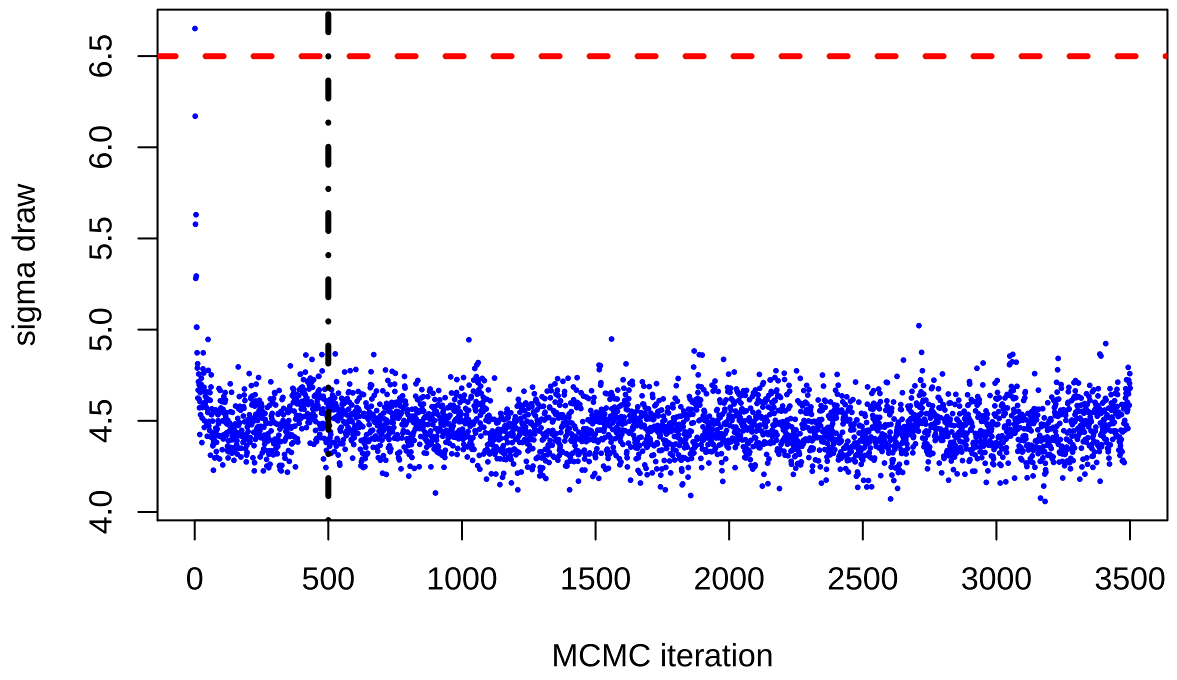
```
library(BART)
```

```
set.seed(14)
bf = wbart(x,y,xp,nskip=500,ndpost = 3000) #BART fit
```

As discussed in Section 2, `(x,y)` is our a training data and the third argument `xp` is the set of x for which we want inference for $f(x)$. `nskip=500` initial MCMC iterations will be used for burn-in and results for the subsequent `ndpost=3000` MCMC iterations will be used for inference.

We can get quick feeling for how the BART model and MCMC performed by plotting the draws of the parameter σ. The component `bf$sigma` contains all the σ draws including burn-in so that there are 500+3000=3,500 draws.

```
plot(bf$sigma,xlab="MCMC iteration",ylab="sigma draw",pch=16,cex=.4,col="blue")
lmf = lm(y~.,data.frame(x,y)) #linear model fit
abline(h=summary(lmf)$sigma,col="red",lty=2,lwd=3)
abline(v=500,col="black",lty=4,lwd=3)
```



We can see the initial burn-in as the σ draws drop, but it seems to have converged well before iteration 500. After burn-in, the σ draws vary about a mean of 4.47. The least squares estimate of σ is 6.499 (plotted with the red horizontal line) which is substantially larger, suggesting that BART has found some fit missed by the linear model.

Let's look at the correlations between `y` and the fitted values from the linear model (`lmf` above) and BART (`bf` above). The component `fb$yhat.test.mean` averages $f_d(x_i)$ over the MCMC draws f_d of f evaluated at each x_i in the test data (`xp` above). This gives us the MCMC estimate of the posterior mean of $f(x_i)$ which we use for our fitted values $\hat{f}(x_i)$.

```
yhatlm = predict(lmf,data.frame(xp))
## Warning in predict.lm(lmf, data.frame(xp)): prediction from a rank-deficient fit
## may be misleading
cor(cbind(y,lmf$fitted.values,bf$yhat.train.mean))
##           y
## y 1.0000000 0.9367010 0.9732643
##   0.9367010 1.0000000 0.9680446
##   0.9732643 0.9680446 1.0000000
```

We see that the the correlation between `y` and the BART in-sample fits is .97, while it is

.94 for the linear fits. (The warning about “fit may be misleading” is because we threw in all the dummies for each categorical x).

4 Big Tree Fit to the Fit

In this section we illustrate our use of big trees to approximate nonlinear functions. We use the R library rpart to fit the trees.

```
library(rpart)
```

Above we have used BART to estimate a function $\hat{f}$ such that $E(Y|x) \approx \hat{f}(x)$. Now we want to use a big tree to approximate this function. We choose the parameters of the argument control to rpart::rpart to make sure rpart fits a big tree.

```
np = length(bf$yhat.test.mean)
minbucket = floor(.5*log(np))
btA = rpart(fhat~.,data=data.frame(xp,fhat=bf$yhat.test.mean),
           control = rpart.control(minbucket=minbucket,
                                    minsplit = 2*minbucket,
                                    maxcompete=0,
                                    maxsurrogate=0,xval=0,cp=0))
```

To see how big the tree is we can look at the number of bottom nodes:

```
cat("the number of bottom nodes is: ",length(unique(btA$where)))
## the number of bottom nodes is:  107
```

We see that there are 107 bottom nodes.

Now let’s have a look at how well our big tree approximates the function $\hat{f}$.

```
yhatbtA = predict(btA,data.frame(xp)) # big tree approx, all x variables
cor(yhatbtA,bf$yhat.test.mean)
## [1] 0.9987708
```

The correlation 0.9987708 suggests that the big tree has done a pretty good job of approximating $\hat{f}$.

The basic idea of our method is to see if we can approximate $\hat{f}$ well using functions that only use a subset of the variables in x. To quickly see if this is the case, we fit a big tree to $\hat{f}$, but only use a subset of the variables. Let's use the variables (mileage, year, trim.other).

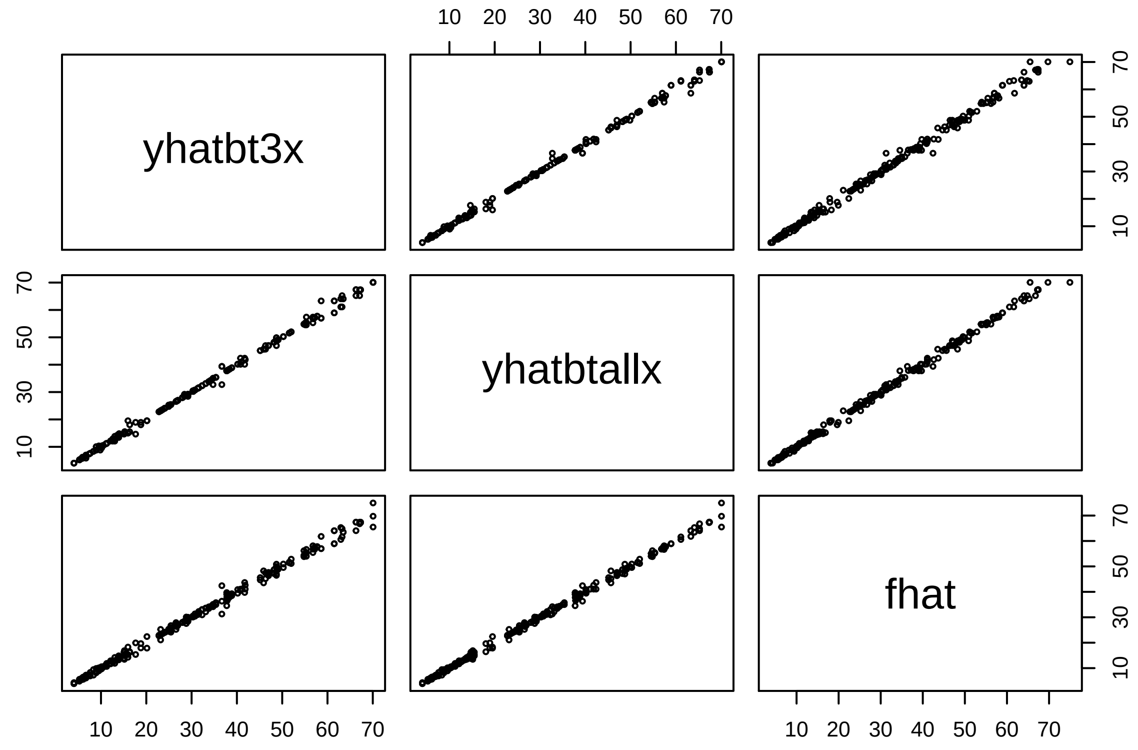
```
print(colnames(xp)[c(1,2,6)])
## [1] "mileage"    "year"       "trim.other"
btS = rpart(fhat~.,data=data.frame(xp[,c(1,2,6)],fhat=bf$yhat.test.mean),
           control = rpart.control(minbucket=minbucket,
                                   minsplit = 2*minbucket,
                                   maxcompete=0,
                                   maxsurrogate=0,xval=0,cp=0))
print(length(unique(btS$where)))
## [1] 108
```

We have fit a big tree with 108 bottom nodes using only the three variables.

If, the big tree fit with only three variables approximates $\hat{f}$ well, then we might just use those three to predict. Let's compare the approximation using just three variables, the approximation using all the variables, and $\hat{f}$. We'll look at the correlations and the three scatter plots using any two of the three.

```
yhatbtS = predict(btS,data.frame(xp)) #big tree approx, three variables
fmat = cbind(yhatbtS,yhatbtA,bf$yhat.test.mean)
colnames(fmat) = c("yhatbt3x","yhatbtallx","fhat")
cor(fmat)
##              yhatbt3x yhatbtallx      fhat
## yhatbt3x   1.0000000  0.9990113 0.9980005
## yhatbtallx 0.9990113  1.0000000 0.9987708
## fhat       0.9980005  0.9987708 1.0000000
```

```
pairs(fmat,cex=.5)
```

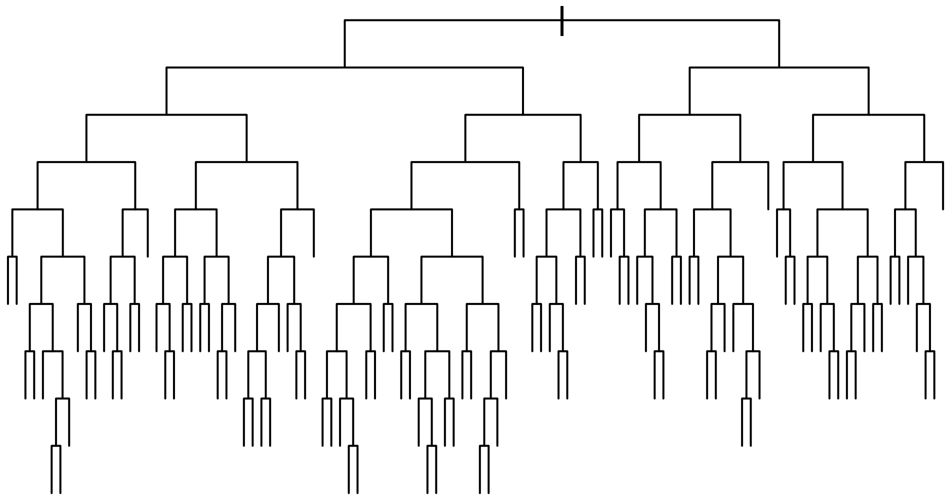


We can see that the big tree with all the variables does a pretty good job. More interestingly, the big tree using just three variables does not look bad! Our approach to variable selection uses simple search strategies which are guided by the big tree approximation.

Again, a big tree does not necessarily work well when fit to data (x, y) because you may over fit. *But*, a big tree fit to $(x, \hat{f}(x))$ may provide us with a quick and simple approximation to $\hat{f}$.

Here is a picture of a *big tree*!

```
plot(btS,uniform=TRUE)
```



5 Variable Selection for the Cars Data

In Section 1 we outlined how our approach to variable selection consists of three steps. For step (1), we use the BART fit to the cars data from Section 3.

In Section 5.1 we illustrate step (2) and in Section 5.2 we illustrate step (3), again using the cars data in each case.

5.1 Cars Data, fit the fit Search for Variable Subsets

In this section we use our variable selection method to find subsets of our 15 x variables that such that a function depending only on the subset approximates the BART fit well.

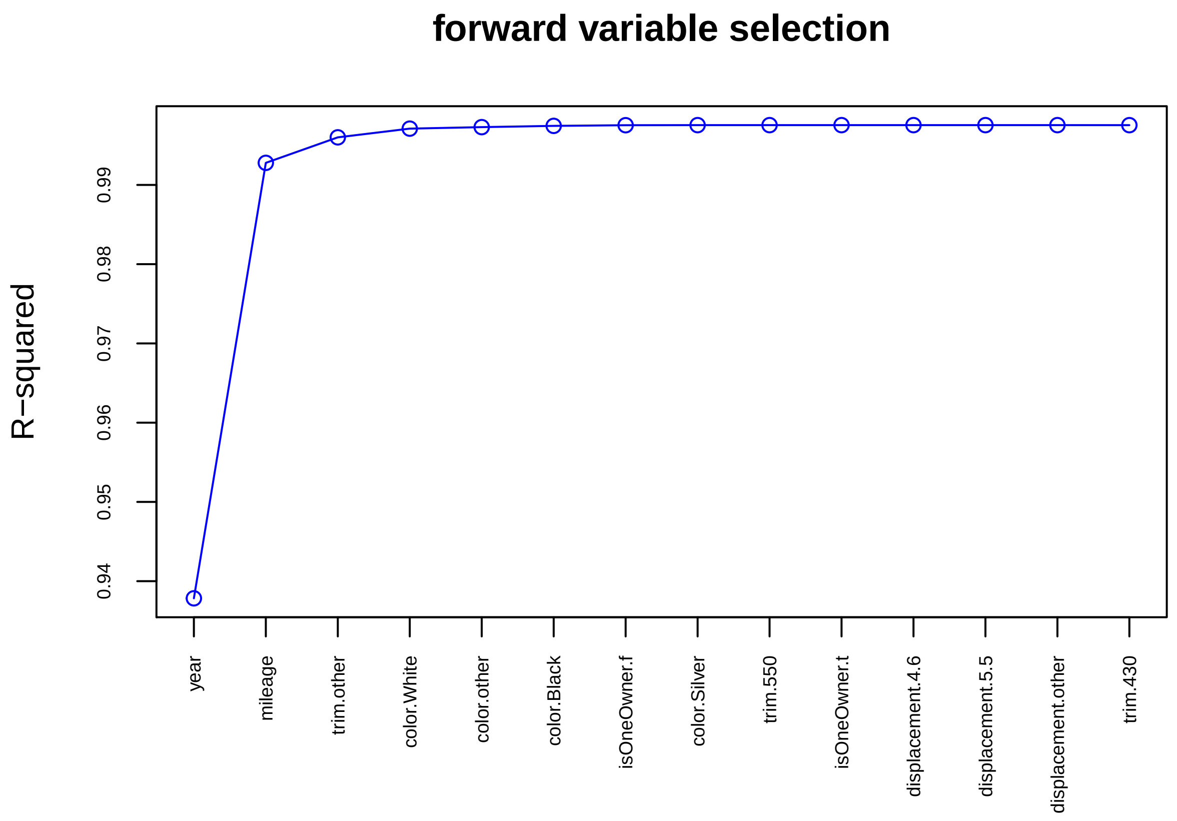
The `nonlinvarsel` package has methods the use (i) forward greedy search, (ii) backwards elimination, and (iii) all subsets evaluation. For each candidate variable subset, a large regression tree is fit using the variables in the subset to fit the fit from the BART inference. All subsets is not viable for a large number of variables.

Let's try forward greedy search using the function `nonlinvarsel::vsf`. Recall that `bf` is the data structure holding the BART fit from the training data `T = (x,y)` at the x values in `xp`. The vector `bf$yhat.test.mean` are the fitted values $\hat{f}(x) = E(f(x) \mid T)$ for x in the rows of `xp`.

```r
vsfr = vsf(xp,bf$yhat.test.mean)
```

The returned `vsfr` list has a plot method and and print method. Let's look at the plot.

```r
plot(vsfr,cex.axis=.6)
```



The variable names printed out along the x-axis indicate the order in which the variables come in using the forward greedy search. The y-axis is the square of the correlation (R^2) between a function of the variables included and the BART fit (`bf$yhat.test.mean`). So, for example, using only the first three variables (year, mileage, trim.other), we can construct of a function of these three variables that explains almost 99.6% of the variation in $\hat{f}(x)$ where the x are all those in `xp` and $\hat{f}$ is estimated using BART (posterior mean of $f(x)$).

Let's print `vsfr`. As the variables come in, we will see the R^2.

```r
print(vsfr)
##              year         mileage      trim.other       color.White
##         0.9378374       0.9927871       0.9960050         0.9971044
##          color.other          color.Black         isOneOwner.f         color.Silver
##            0.9972945            0.9974533            0.9975371            0.9975456
##            trim.550         isOneOwner.t   displacement.4.6   displacement.5.5
##            0.9975456            0.9975456            0.9975456            0.9975456
## displacement.other            trim.430             trim.500
##            0.9975456            0.9975431            1.0000000
```

We see that the R^2 using the first variable is .94 and .99 using the first two.

Let's review how the forward greedy search works. The first step of the search tries each variable in x one at a time. For each variable, we fit a big tree using just that one variable and $\hat{f}(x)$ as our response for x in `xp`. We pick the variable which gives us the best fit (as measured by R^2). This gives our first variable which is `year` in the output above. To get our next variable, we loop through the remaining variables and fit a big tree to see how well the resulting pair of variables can fit $\hat{f}(x)$. We then pick the pair that gives us the best fit. This gives us the pair `(year,mileage)` in the output above. Continuing in this manner, we introduce variables one at a time. See Section 4 where we explicitly illustrate fitting the fit $\hat{f}(x)$ with three variables. Note that the forward search uncovers one variable subset of each size. There is no notion of how important a variable is on its own.

The function `nonlinvarsel::vsf` returns a list:

```
names(vsfr)
## [1] "vL"     "R2"     "vrank"  "xnames" "fit"
```

Of particular interest is the component `vL` which gives the indices in the columns of `xp` corresponding to the variable subsets. The component `R2` gives the value R-squared of for each subset size.

```
vsfr$vL[[3]]
## [1] 2 1 6
colnames(xp)[vsfr$vL[[3]]]
## [1] "year"       "mileage"    "trim.other"
```

```
vsfr$R2[3]

## [1] 0.996005
```

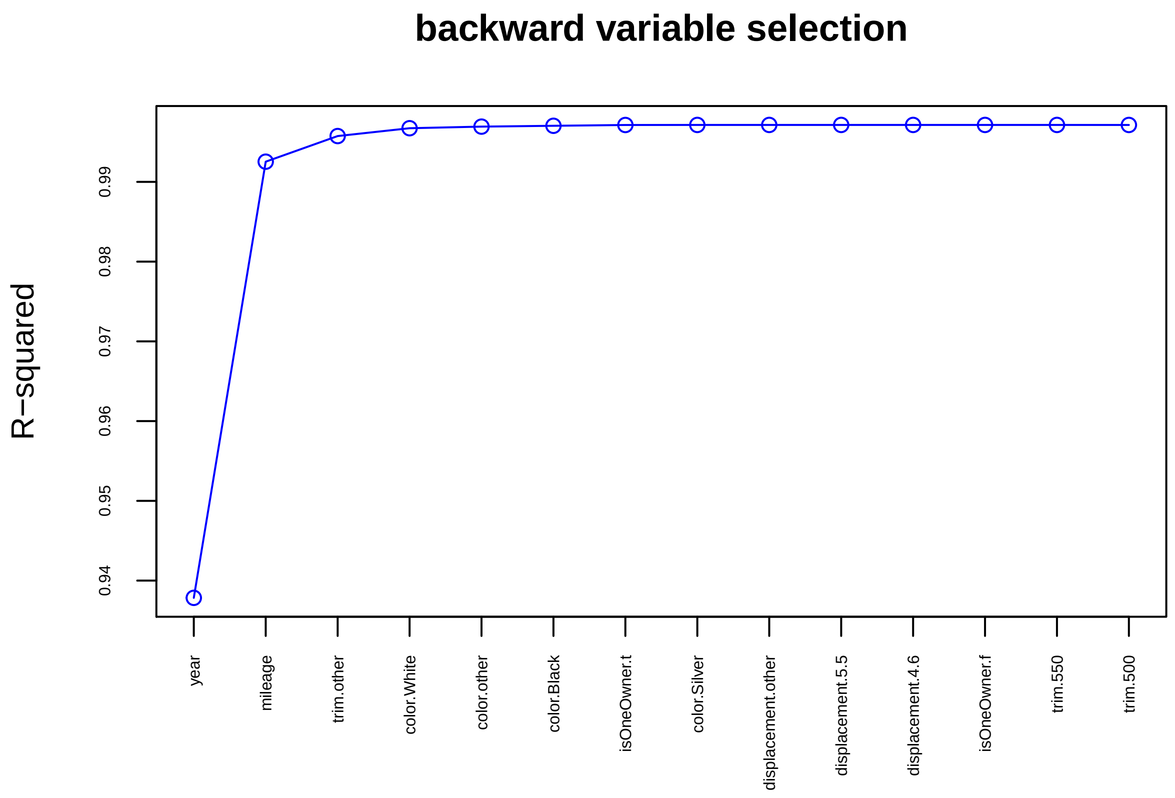
We see that the subset using three variables corresponds to the variables in columns 2,1, and 6 of `xp`. The R-squared value is 0.996 indicating that, using a big tree, we have found a function of these three variables which fits the unrestricted fit very well.

Now let's try the backward elimination search using `nolinvarsel::vsb`. Backward elimination start with all the variables in and then takes variables out one at a time.

```
vsbr = vsb(xp,bf$yhat.test.mean)
```

We can plot and print the results. Let's have a look at the plot.

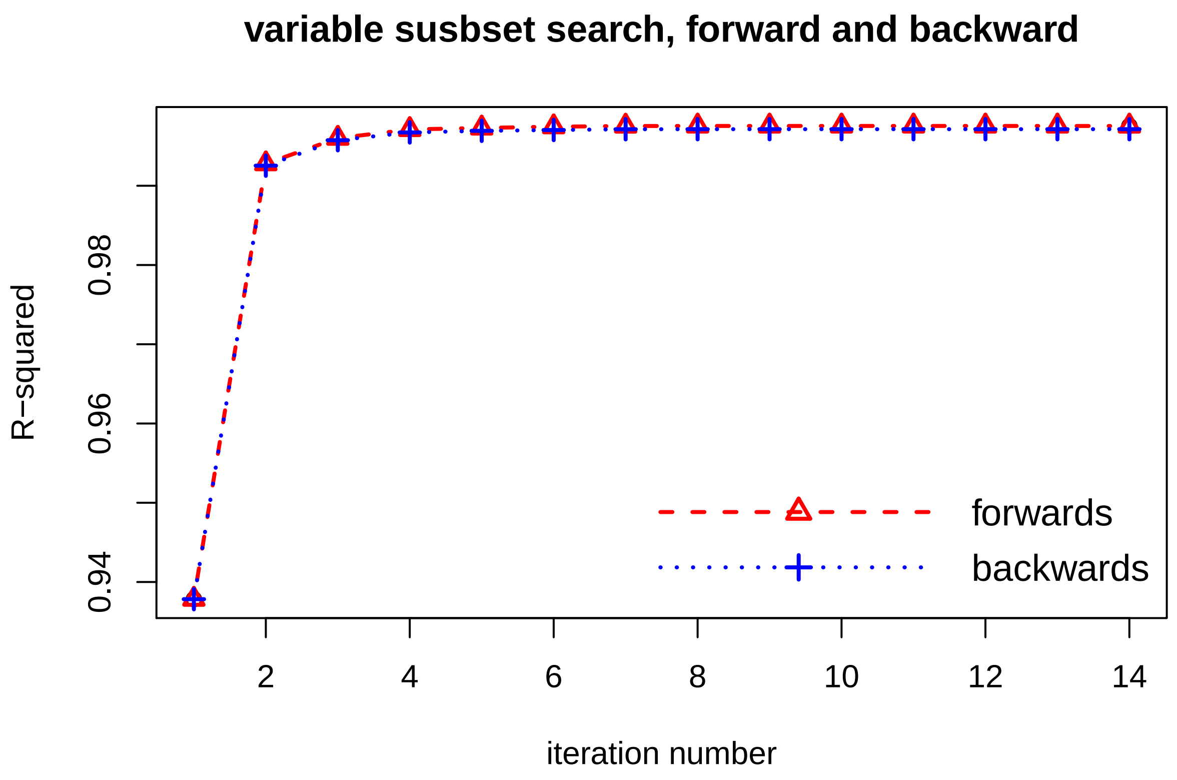
```
plot(vsbr, cex.axis=.5)
```



The results tell us that the first variable thrown out in the backwards elimination is `trim.500` and the last variable left is `year`. The obvious variables `year` and `mileage` work very well, but you may want to consider a variable representing a trim category. The backwards results are very similar to the forwards results.

We can plot the forwards and backwards search together to check their similarity:

```r
plotfb(vsfr,vsbr)
```



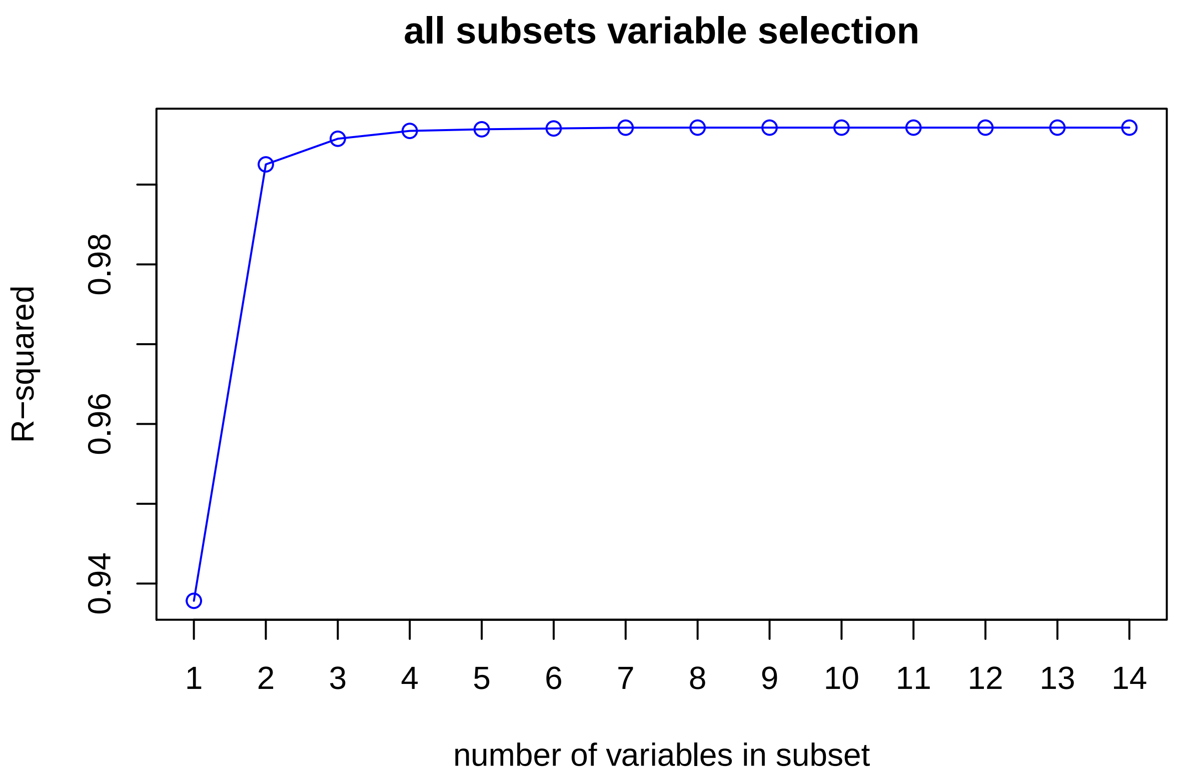
The results are identical.

Since we only have 15 variables, we can try all possible subsets. However, it takes substantially more time than the forwards and backwards searches and typically gives similar results.

```r
vsar = vsa(xp,bf$yhat.test.mean)
```

Let's plot the results. Note that since the subsets may not be nested (as in the forward search) we need the complete set of indices for each group size. Hence we just plot the subset size on the x-axis.

```r
plot(vsar)
```



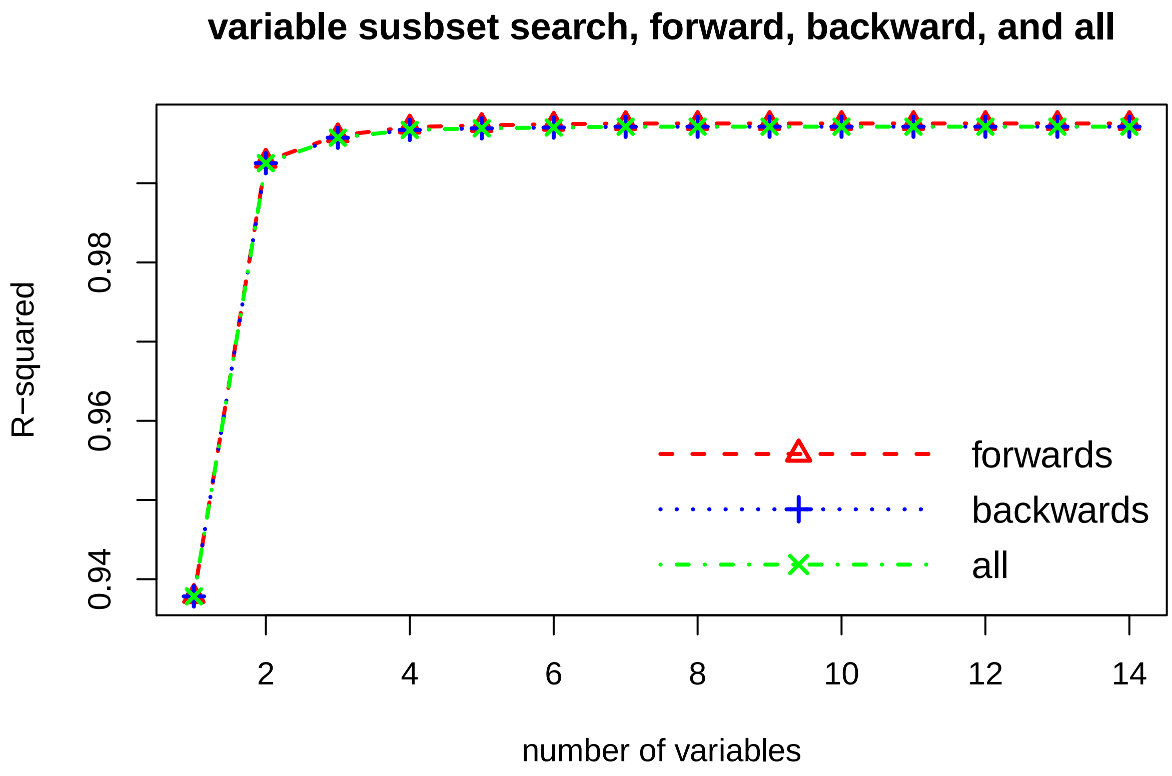
Let's print out some information about the subsets found using the components `vL` and `R2`. Let's look at the `R-squared` and indices for the subsets of sizes 1-4. Let's also print the names of the variables in the subset of size 4.

```r
print(vsar$R2[1:4])
## [1] 0.9378374 0.9925319 0.9957444 0.9967292
print(vsar$vL[1:4])
## [[1]]
## [1] 2
##
## [[2]]
## [1] 1 2
##
## [[3]]
## [1] 1 2 6
```

```
##
## [[4]]
## [1]  1  2  6 12
print(colnames(xp)[vsar$vL[[4]]])
## [1] "mileage"     "year"        "trim.other"  "color.White"
```

We can use the function `nonlinvarsel::plotfba` to plot all three searches.

```
plotfba(vsfr,vsbr,vsar)
```



We see that we obtain the same results we got from the forwards and backwards searches.

In general, it is possible for the searches to find different surrogates $\gamma(x_S)$ with the hope that $\gamma(x_S) \approx \hat{f}(x) \approx E(Y \mid x)$. From a statistical inference point of view it is important to remember that we are just seeking a good surrogate in this sense. If does not matter if we have the exact globally optimal surrogate and there are no tricky multiple-comparison type inference issues related to our search over different kinds of surrogates. All uncertainty is base on our step (1) inference. We illustrate this in the next section. If you find a good surrogate it is of real practical value in an uncomplicated way. And it is just fine if there is a better one out there, or if you find a better one a later date. This conceptual simplicity driven by what we feel is a more useful specification of the real practical issues is a major strength of our approach.

5.2 Posterior Uncertainty for the Approximation Error

An advantage of using BART for our function estimation is that we get draws from the posterior distribution of f rather than just the point estimate $\hat{f}$. Using these draws we can assess the uncertainty about the approximation error for each subset of x variables. This is step (3).

The component `bf$yhat.test` stores evaluations $f_d(x_j)$ for $d = 1, 2, \ldots, D$ post burn-in draws f_d of f and each x_j in the test data.

```
dim(bf$yhat.test)
## [1] 3000  250
```

We have 3000 post burn-in draws and 250 x vectors in the test data `xp`.

We can now assess our uncertainty. We use `nonlinvarsel::sumpost`.

```
sp = sumpost(bf$yhat.test,vsfr$fit,bf$yhat.test.mean,distrmse)
## nd,n,p:  3000 250 14
```

The arguments are the draws of f (`bf$yhat.test`) representing our uncertainty about the true function, the fits from our candidate subsets (`vsfr$fit`), the fit using all the information in x (`bf$yhat.test.mean`), and a distance metric to measure the difference between a draw f_d and a candidate function. The distance metric used above is RMSE (root mean squared error).

$$D_{rmse}(f, \gamma_S) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(f(x_i) - \gamma_S(x_i))^2}.$$

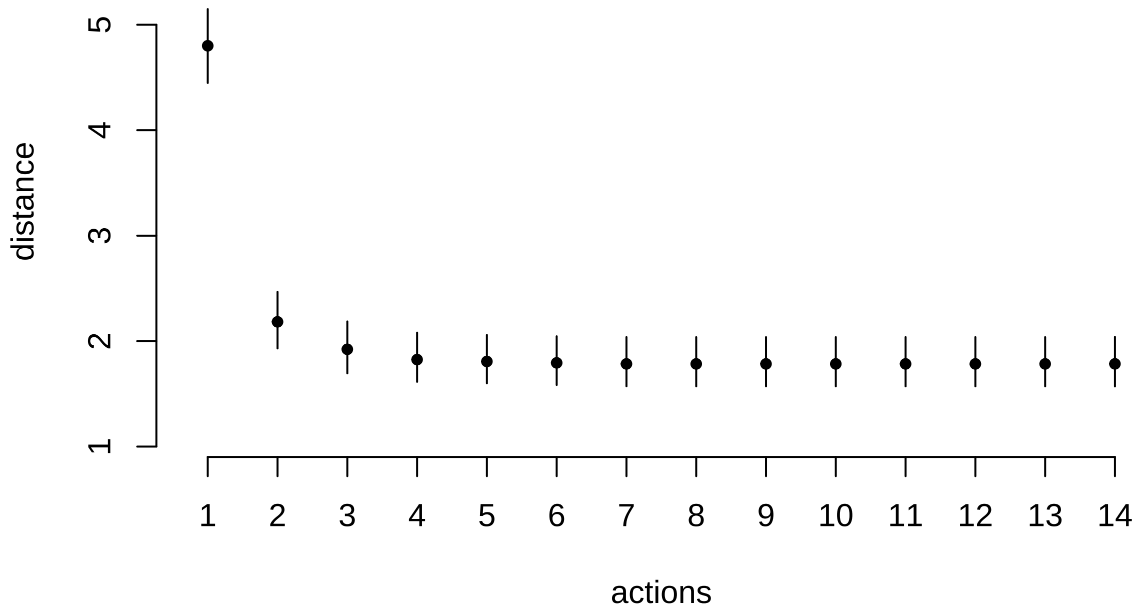
The sum is over $\{x_i\}$ in the n rows of `xp`. Note that any distance may be easily used. We may also be interested in $D_{rmse}(f, \gamma_S)$, the possible error of $\hat{f}$.

We can use a plot method for the value returned by 'nonlinvarsel::sumpost".
For each subset S, the plot method displays the values

$$D_{rmse}(f_d, \gamma_S), \quad d = 1, 2, \ldots, D.$$

This is our Monte Carlo estimate of the posterior distribution of the difference between f and our approximation γ_S.

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



The solid dot is the median and the vertical line indicates the 95% posterior interval. The component `distaction` of `sp` stores all the $D_{rmse}(f_d, \hat{f}_S)$ values.

```
quantile(sp$distaction[,3],probs=c(.025,.5,.975))
##     2.5%      50%    97.5%
## 1.656413 1.922544 2.266700
```

These three quantiles are what are plotted in the third vertical segment in the plot above. To assess our approximation error, we need to think about what kind of error matters as a *practical matter*. Recall from Section 2 that the median price of a car is 30 (thousand dollars). We can see that with high posterior probability our error is close to 2 and that that error is obtained with a subset of size four. For practical purposes, it is very likely that the loss due to approximation error using a subset of size three is negligible. A subset of size two, using only the simple variables `(year, mileage)` may be quite acceptable as a practical matter. This kind of simplifying information is often what practitioners seek.

Another useful way to look at the posterior uncertainty is by looking at the difference,

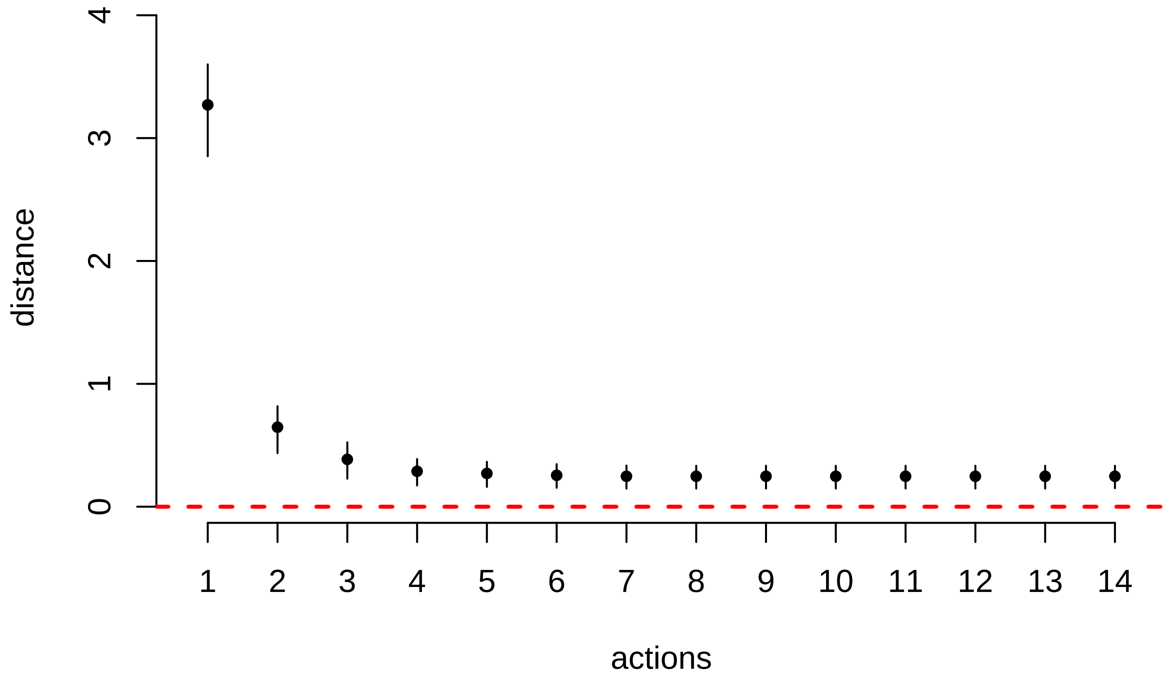
$$D(f, \gamma_S) - D(f, \hat{f}),$$

where $\hat{f}$ uses all of the variables as in our original BART inference.

With `diff = TRUE` (the default), our call to `nonlinvarsel::plot.sumpost` plots the posterior distributions of the differences which are are estimated by the Monte Carlo draws

$$D(f_d, \gamma_S) - D(f_d, \hat{f}), \;\; d = 1, 2, \ldots, D.$$

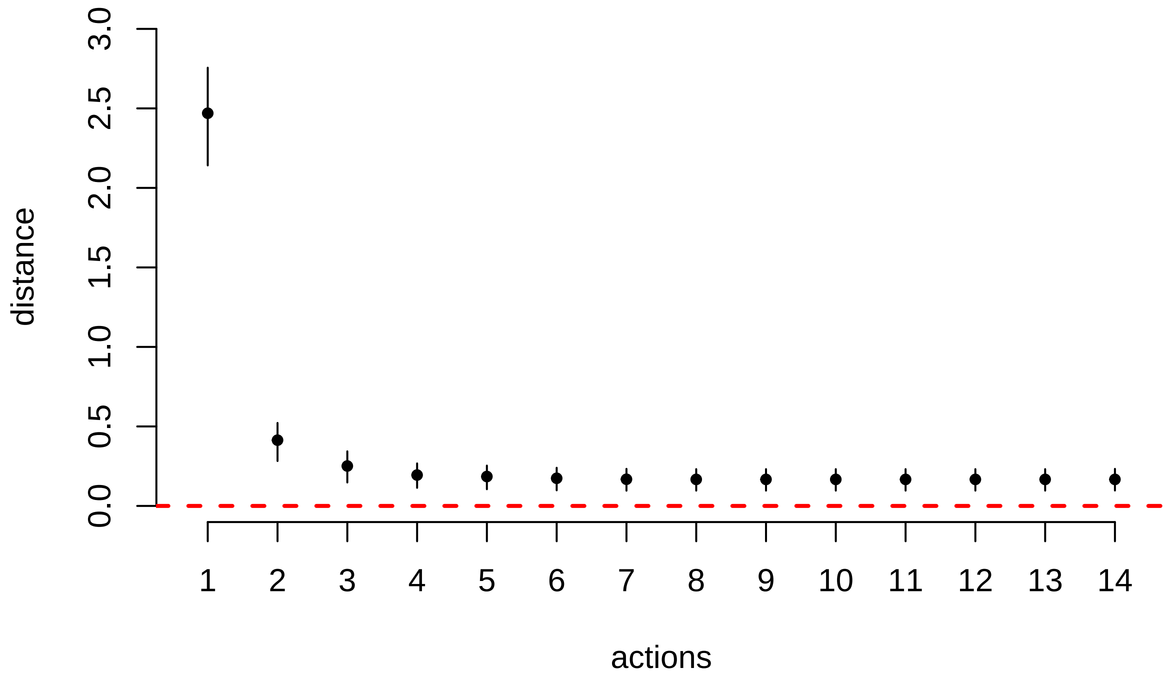
```
plot(sp, diff=TRUE) #diff = TRUE is the default
```



Clearly, with high posterior probability, as a practical matter, there is little difference between using the full subset and the discovered subset of size four. Again, subsets of size three or two might be acceptable.

Of course, we may want to use a metric other than root mean square error. We repeat the analysis using the average of the absolute value of the error.

```
spabs = sumpost(bf$yhat.test,vsfr$fit,bf$yhat.test.mean,distabs)
## nd,n,p:  3000 250 14
plot(spabs)
```



Again, with four variables there is a very small chance of an average error of practical importance.

Note that in our method, *practical significance* naturally emerges as the key concept.

5.3 Running BART on Subsets to Obtain Alternative Surrogates

In some examples we have found that the surrogate function found using the big tree approximation can be improved upon. We typically find that the subsets found using big tree guided searches are useful even in this case. Given, the subsets we can explore ways of finding alternative surrogate approximators without having to look at all possible subsets.

An obvious way to look for an alternative is to fit BART to the original data using a given variable subset. We can speed this up by doing the work in parallel using the `R` package `doParallel`. Of course, this depends on how many cores you have in the machine you are working with.

```
library(doParallel)
## Loading required package: iterators
## Loading required package: parallel
registerDoParallel(cores=4)
```

We now use `nonlinvarsel::bartSubs` to run `BART::wbart` using each of the 14 subsets found with the forward search.

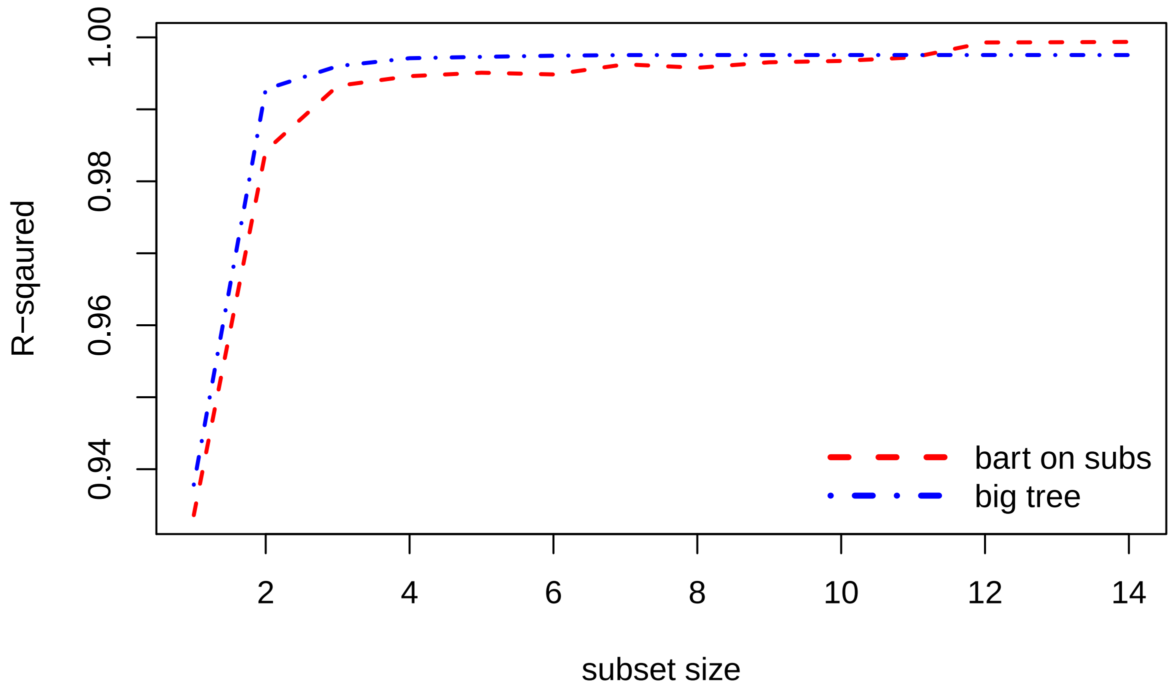
```
bsubs = bartSubs(x,y,xp,vsfr$vL,nskip=500,ndpost=3000)
```

```
dim(bsubs)
## [1] 250  14
```

Each of the 14 columns of `bsubs` is the fitted values on the test data `xp` from running BART on the training data using the subset of variables in `vsfr$vL`.

Let's plot the correlation between each of the columns of `bsubs` and the fitted values obtained using all of x.

```
ns = ncol(bsubs)
R2subs  = (cor(cbind(bsubs,bf$yhat.test.mean))[1:ns,ns+1])^2
plot(c(1,ns),range(c(R2subs,vsfr$R2)),type="n",xlab="subset size",ylab="R-sqaured")
lines(1:ns,R2subs,col="red",lty=2,lwd=2)
lines(1:ns,vsfr$R2,col="blue",lty=4,lwd=2)
legend("bottomright",legend=c("bart on subs","big tree"),
       col=c("red","blue"),lty=c(2,4),lwd=c(3,3),bty="n",seg.len=4)
```



In this case we see that it appears that the big tree fit has done an excellent job.

In the cars example, the message from the data was quite clear and all three searches gave the same result. In addition the big tree surrogates seem to be quite successful. In some example these simple results may not hold. It is possible to get different answers from forwards and backwards search and sometimes we find that we can improve on our big tree fits. However, we almost always find that the big tree guided search delivers subsets which are useful. Again, note that our inference is note dependent on finding the optimal solution.

5.4 Out of Sample Performance

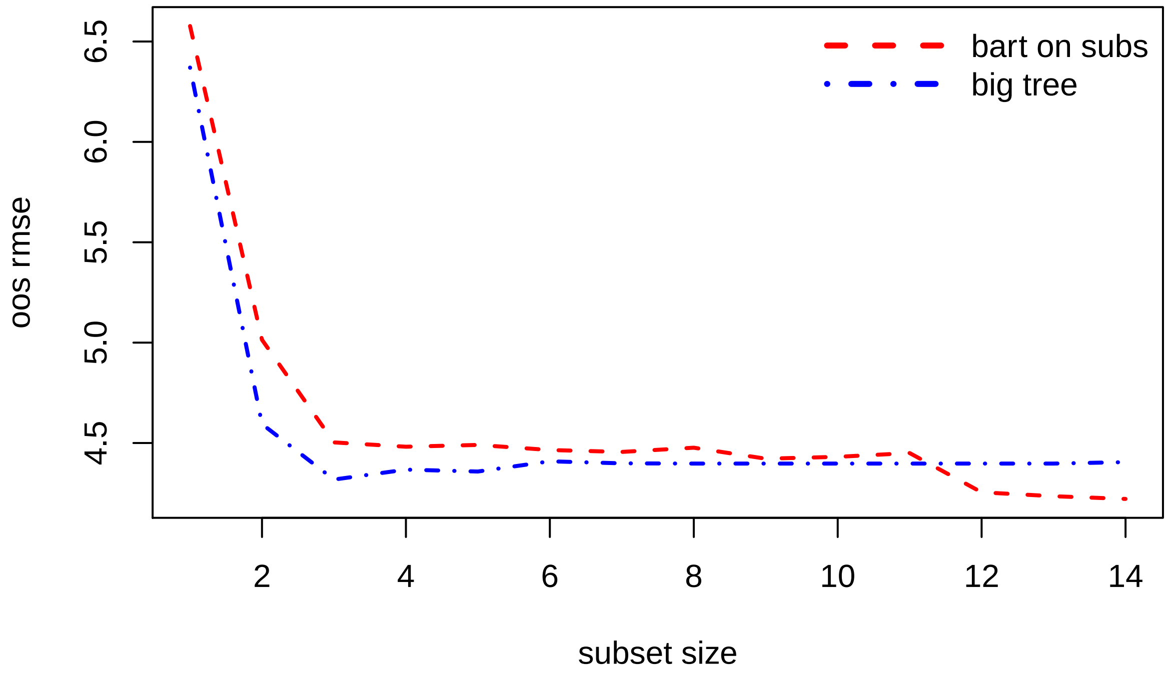
In general, as discussed in section 2 there is no need to obtain `xp` from a train/test split of the original data. We commonly use the simple `xp=x`, that is, `xp` is all the data. Given that we did use a train/test split, let's look at the out of sample performance.

First compute the out of sample root means squared error (RMSE) for both the big tree surrogates and the BART fit on subsets surrogates.

```
rmsef = function(y,yhat) { return(sqrt(mean((y-yhat)^2)))}
ns = ncol(bsubs)
ebt = rep(0,ns) # big tree oos rmse
ebs = rep(0,ns) # bart subs oos rmse
for(i in 1:ns) {
   ebt[i] = rmsef(yp,vsfr$fit[,i])
   ebs[i] = rmsef(yp,bsubs[,i])
}
```

Now we plot the (RMSE) versus subset size.

```
plot(c(1,ns),range(c(ebt,ebs)),type="n",xlab="subset size",ylab="oos rmse")
lines(1:ns,ebt,col="blue",lty=4,lwd=2)
lines(1:ns,ebs,col="red",lty=2,lwd=2)
legend("topright",legend=c("bart on subs","big tree"),
       col=c("red","blue"),lty=c(2,4),lwd=c(3,3),bty="n",seg.len=4)
```



The out of sample performance indicates that our subsets of size three or four (or even two) are quite adequate as a practical matter. This is consistent with our posterior analysis is Section 5.2. But let's review how tricky this kind of train/test analysis really is. We just did one train test split and just have 250 test `y` values. We could do 5 of 10 fold cross-validation but then we just have 5 or 10 draws of our training data. These are real practical concerns. Witness the time honored "1se" rule which is clearly at compromise at best. While there cannot be a guarantee that the BART draws have fully expressed our uncertainty, the simplicity or our approach seem appealing.

6 Simulated Data

In this section we simulate data using the commonly used simulation setup originally proposed by Friedman (1991). There are 10 variables in x, but only the first 5 are in the true function. The function is nonlinear in the first three x variables. Each x is iid uniform on $(0, 1)$.

We simulate the data:

```
# the Friedman function
f = function(x){
  10*sin(pi*x[,1]*x[,2]) + 20*(x[,3]-.5)^2+10*x[,4]+5*x[,5]
}


# training data
set.seed(99)
sigma = 1.0  #y = f(x) + sigma*z , z~N(0,1)
n = 100      #number of observations
p = 10
x = matrix(runif(n*p),n,p)
colnames(x)=paste0("x",1:p)
Ey = f(x)
y = Ey+sigma*rnorm(n)
```

So,

$$Y = 10\sin(\pi\, x_1 x_2) + 20(x_3 - .5)^2 + 10\, x_4 + 5\, x_5 + \epsilon, \quad \epsilon \sim N(0,1)$$

We next simulate the x values that we want to be able to predict at and store them in `xp` as in the car price example. In this example the simulated training data only has 100 observations. We will simulate a richer set of x values so that when we compare predictions we fully consider the kinds of predictions we might want to be able to make in the future. Here is a place where our approach differs fundamentally from many commonly used methods. Since we consider the practical implications of our model, it matters what we intend to do with it.

```r
set.seed(99)
np = 5000       #number of observations for prediction
p = 10
xp = matrix(runif(np*p),np,p)
colnames(xp)=paste0("x",1:p)
```

We now run BART to obtain our step (1) inference.

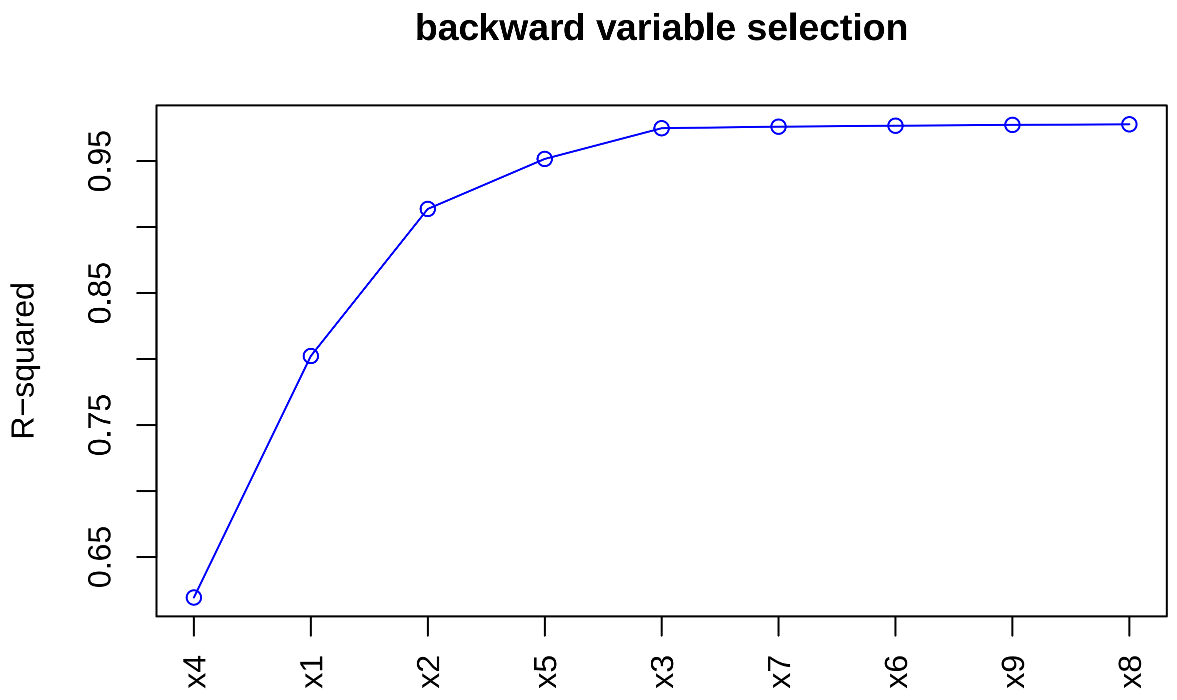
```r
burn = 1000; nd = 6000
#  first BART run
set.seed(99)
bff = wbart(x,y,xp,nskip=burn,ndpost = nd)
```

Let’s try the backward search.

```r
vsbrf = vsb(xp, bff$yhat.test.mean)
```

Now we plot and print the results.

```r
plot(vsbrf)
```



```r
print(vsbrf)
##        x4        x1        x2        x5        x3        x7        x6        x9
## 0.6192673 0.8023019 0.9137632 0.9516807 0.9749607 0.9761311 0.9768612 0.9774760
##        x8       x10
## 0.9779176 1.0000000
```

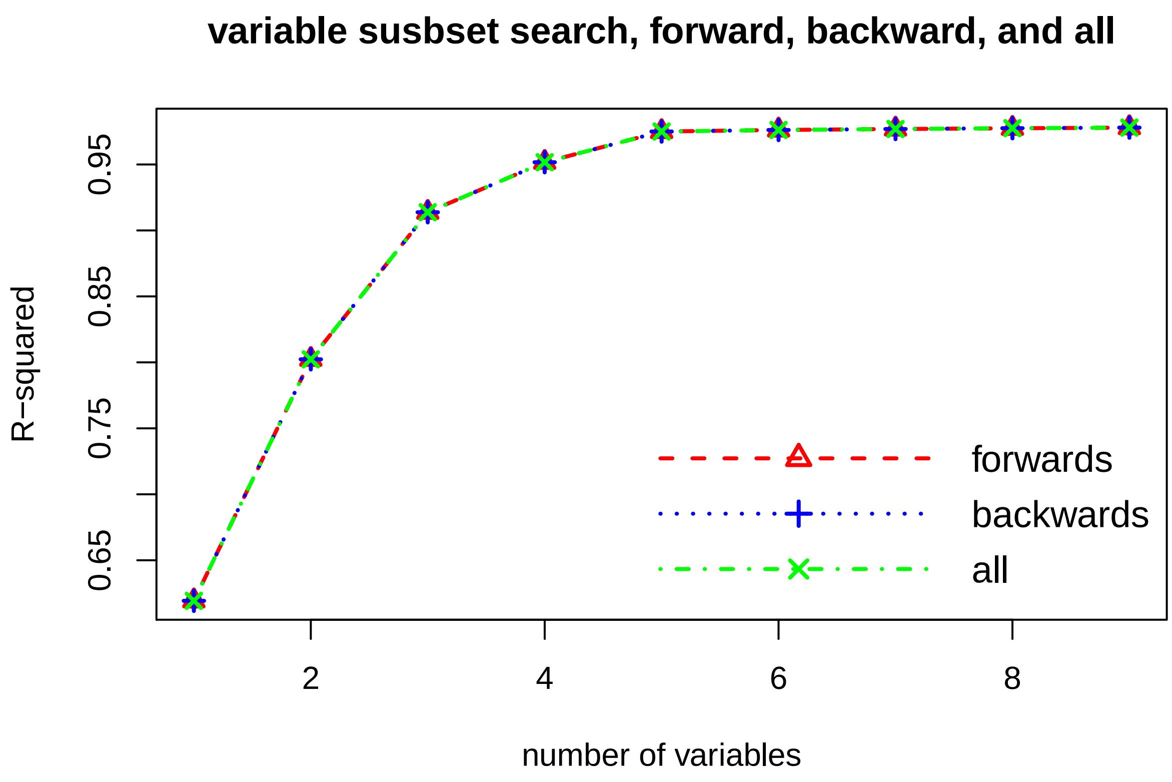
The backward search indicates that we can predict just as well using only the first five variables as we can using all of them. Each of the five variables $x_i, i = 1, 2, 3, 4, 5$ seems to help appreciably. Note that in the car price example, the units of y meant something to us but here they do not. Consequently, we cannot make statements like *as a practical matter*, x_3 (the first of the first 5 to go out) is not that important given the first four.

Let's try forward search and all possible subsets. Since we only have 10 variables, just trying all possible subsets is feasible.

```r
vsfrf = vsf(xp, bff$yhat.test.mean)
vsarf = vsa(xp,bff$yhat.test.mean)
```

We can compare the three searches by plotting them all together using `nonlinvarsel::plotfba`.

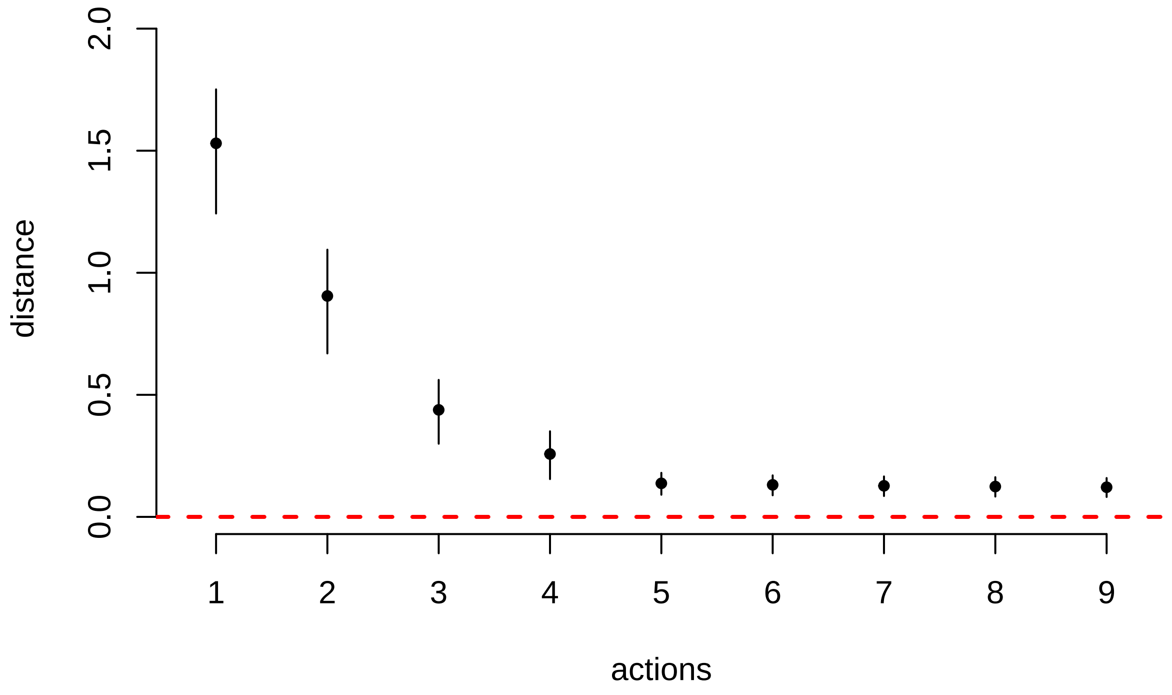
```
plotfba(vsfrf, vsbrf, vsarf)
```



We see that the three searches give the same results.

Now let's look at the uncertainty in our approximation error.

```
sp = sumpost(bff$yhat.test,vsfrf$fit,bff$yhat.test.mean,distrmse)
## nd,n,p:  6000 5000 9
plot(sp)
```

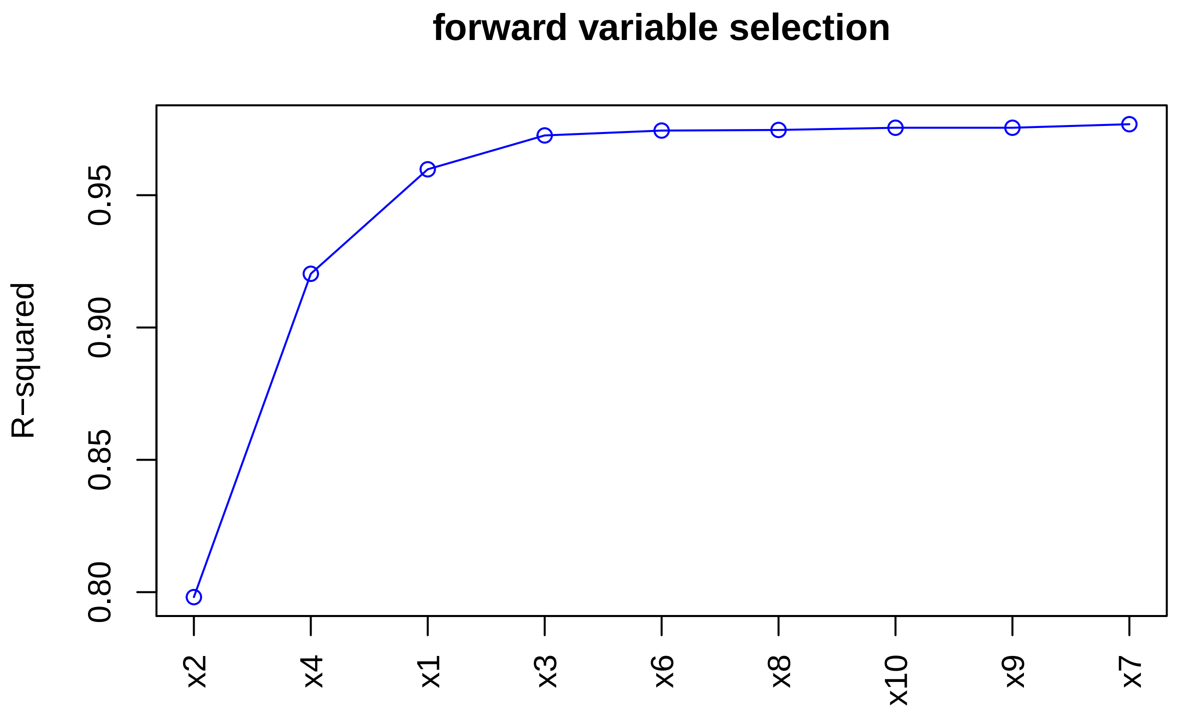


We have strong evidence that only the first 5 variables are needed.

As an exercise, let's see what would happen if we just used the 100 sample x values for `xp`. Our idea was that this may not be enough because 100 vectors in R^{10} can't capture the set of possible predictions we may want to make.

We don't have to rerun BART, we can just use the `train` components instead of the test components.

```
vsfrfS = vsf(x,bff$yhat.train.mean) # forward using the train x, S for Small xp set.
plot(vsfrfS)
```



Now we do not how a strong indication that you need to use x_5 as it comes in last. We do not describe this answer as “wrong”. Rather, given your posterior beliefs about f, if you just have to predict at those 100 x values, you don’t need to know x_5.

7 The Hockey Penalty Data

In this section we illustrate the method on hockey penalty data (see Abrevaya & McCulloch (2014)). Note that in this section we simply present the results without providing the `R` code.

Data was collected on each penalty in each National Hockey League (NHL) game from 1996 to 2001. The goal is to predict which team will get the next penalty for each penalty in a game after the first penalty. The total number of observations is 57,883. The response y is coded as indicating whether there is a “reverse call”, that is, the current penalty is not on the same team as the previous penalty. About 60% of the time the penalty is a reverse call.

The variables in x seek to capture things about the game at the time of the penalty and characteristics of the two teams that are playing the game. There are a total of $p = 26$

variables in x. See Abrevaya & McCulloch (2014) for a full description of the variables. Key variables record such things as whether the last two penalties were on the same team, the score in the game at the time of the penalty, and the time since the last penalty.

Abrevaya & McCulloch (2014) split the data into 47,883 training observations and 10,000 out of sample test observations. The models were again estimated using a version of BART appropriate for a binary outcome using the training data.

In this section our response y is binary. In order to keep the exposition simple in Section 1 we focused on a numeric y and the corresponding prediction mechanism $E(Y \mid x)$. In this section our focus is on $P(Y = 1 \mid x)$ where $Y = 1$ indicates that the penalty is a reverse call. Of course, for a Bernoulli Y, we do have $P(Y = 1 \mid x) = E(Y \mid x)$ but we think about it differently. In particular, we consider different kinds of loss functions when evaluating our uncertainty in step (3). While we emphasize that the best choice of loss is application driven, it is useful to have generic loss function. In this application we will use the Kullback-Leibler divergence. Recall that if $Y \in \{0, 1\}$ and P is the distribution such that $P(Y = 1) = p$ and Q is such that $P(Y = 1) = q$, then the Kullback-Leibler divergence between P and Q is

$$K(P, Q) = E_P(P(y)/Q(y)) = p\,\log(p/q) + (1 - p)\log((1 - p)/(1 - q)) \equiv k(p, q).$$

A BART fit gives us a function such that $\hat{f}(x) \approx P(Y = 1 \mid x)$ and again we can use large regression trees to search for surrogates such that $\gamma_S(x) \approx \hat{f}(x)$. For our step (3) loss we use

$$D(f, \gamma_S) = 2\,\frac{1}{n}\sum_{i=1}^{n} k(f(x_i), \gamma_S(x_i)).$$

This is twice the average Kullback-Leibler divergence between the Bernoulli $Y \sim \text{Bernoulli}(f(x))$ and the Bernoulli $Y \sim \text{Bernoulli}(\gamma_S(x))$. It is common (but not necessary) to double the KL distance to make it more comparable to the deviance used below.

We used the forward search to uncover good subsets. The step (3) posterior uncertainty of the approximation error is in Figure 1. We use the default option which reports the posterior of $D(f, \gamma_S) - D(f, \hat{f})$. We see a strong indication the only about 12 of the 26 variables are needed. The first six variables are `inrow2`, `home1`, `numpen`, `timebetpens`, `goaldiff1`, `pf1`, and `pf2`, where, for example, `inrow2` indicates whether the last two penalties were on the same time and `goaldiff1` indicates the lead of the last penalized team.

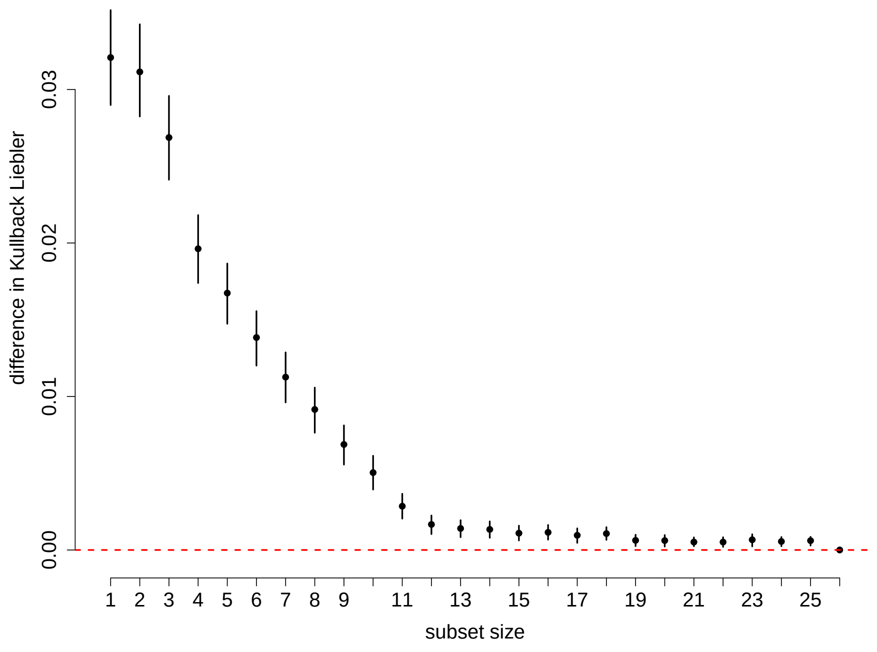


Figure 1: Posterior summary for the hockey penalty data.

Figure 2 presents the out-of-sample loss on the 10,000 test observations as measured by the deviance for BART models estimated using the training data and the variable subsets discovered by our method as in Section 5.3. The out-of-sample performance corresponds very closely to posterior inference for the approximation error.

At the same time we emphasize that our assessment of posterior uncertainty in 1 is a fundamentally different thing from the out-of-sample error estimation in 2. In application, the information provided by the posterior uncertainty may be a key component of the variable selection and is not available in train/test estimation of out of sample loss.

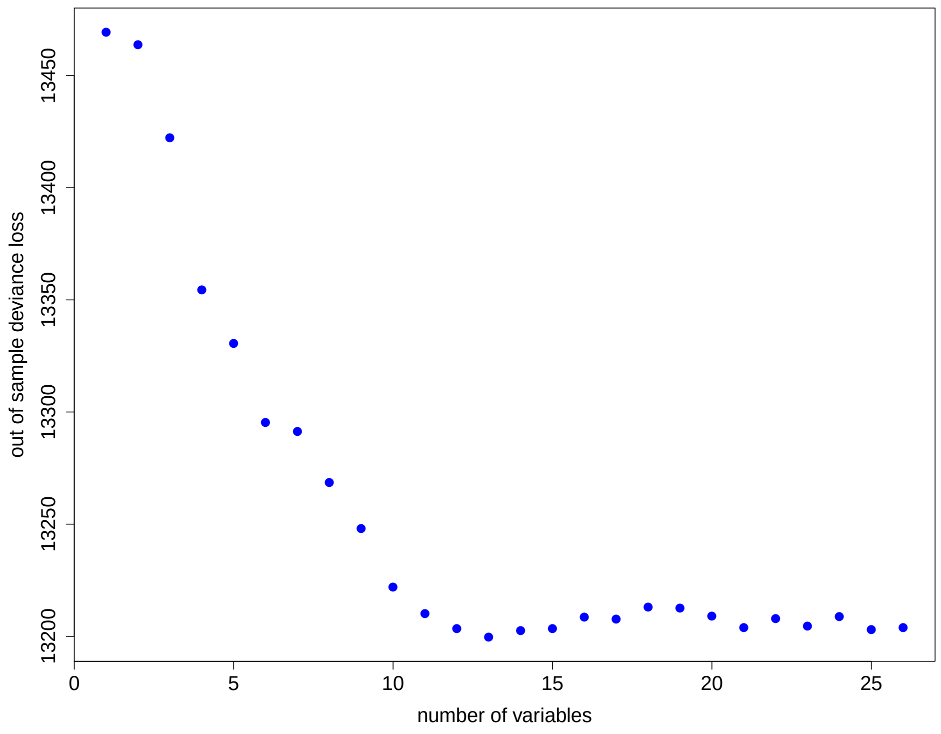


Figure 2: Out of sample performance for the hockey penalty data.

8 Conclusion

The is paper provide and intuitive introduction to the approach for variable selection for nonlinear models presented in CHM. CHM provide a Bayesian decision theoretic motivation for the approach. Here we informally emphasis that as a matter of *practical significance* we seek variable subsets that provide the same amount of information about the response as the full set of variables.

We also provide a tutorial introduction to the `R` package `nonlinvarsel` which implements the approach.

Our examples have used BART for our step (1) inference. We have also successfully used deep neural nets for step (1) and supplied the $\hat{f}$ from the neural net fit to step (2).

Our approach to Bayesian variable selection differs fundamentally from the approaches most commonly developed in the literature. The usual approach injects prior beliefs that many of the variables in x don't matter and then computes the posterior. Such beliefs often take the form of a "sparsity prior" which, one way or another, expresses beliefs that only a subset of the variables are related to the response in any way. The effect of most variables on Y is *exactly* zero. In the examples in this paper, our BART inference has no prior guidance to seek sparsity. We obtain simple models by looking for simple approximations to our unconstrained inference. We look for variables which, has a practical matter are not important, without having to believe they are completely unrelated to Y. Our variable selection is driven by the utility: as a practical matter it is useful to find simple models. The usual approach is driven more by the prior: we believe that the truth is a simple structure and engineer our prior to seek it.

Of course, we could use a step (1) inference that incorporated a sparsity prior. See Linero (2018) for a development of BART which includes a sparsity prior. But our experience suggests that keeping the prior part of the model simple greatly simplifies the process and consequently gives good results with relatively little input from the user. Our step (3) inference tells us what we want. We can readily see which variables are likely to be of little practical use where "likely" refers to the full Bayesian posterior from step (1). Our relatively simple search mechanisms typically find very useful subsets without having to worry about whether we have successfully inferred "the truth". The usual approach based on prior beliefs entails difficult issues surrounding the specification of the prior and the computation of the posterior which we entirely avoid.

References

Abrevaya, J. & McCulloch, R. (2014), 'Reversal of fortune: A statistical analysis of penalty calls in the national hockey league', *Journal of Quantitative Analysis in Sports* **10**(2), 207–224.

Carvalho, C., Hahn, P. & McCulloch, R. (2020), 'Fitting the fit, variable selection using surrogate models and decision analysis'.

Chipman, H., George, E. & McCulloch, R. (2010), 'Bart: Bayesian additive regression trees', *The Annals of Applied Statistics* **4**(1), 266–298.

Friedman, J. H. (1991), 'Multivariate adaptive regression splines (Disc: P67-141)', *The Annals of Statistics* **19**, 1–67.

Hahn, P. R. & Carvalho, C. M. (2015), 'Decoupling shrinkage and selection in bayesian linear models: a posterior summary perspective', *Journal of the American Statistical Association* **110**(509), 435–448.

Hahn, P. R. & He, J. (2020), 'Stochastic tree ensemblesfor regularized nonlinear regression'.

Linero, A. (2018), 'Bayesian regression trees for high dimensional prediction and variable selection', *Journal of the American Statistical Association* **113**(522), 626–36.

McCulloch, R., Sparapani, R., Gramacy, R., Spanbauer, C. & Pratola, M. (2019), *BART: Bayesian Additive Regression Trees*. R package version 2.7.
URL: *https://CRAN.R-project.org/package=BART*

Sparapani, R., Spanbauer, C. & McCulloch, R. (2020), 'Nonparametric machine learning and efficient computation with bayesian additive regression trees: the bart r package', *Journal of Statistical Software* (in press), 1–71.