The BART3 R package

Rodney Sparapani Medical College of Wisconsin
Charles Spanbauer Medical College of Wisconsin
Robert McCulloch Arizona State University

Abstract

In this article, we introduce the **BART3** R package which is an acronym for Bayesian Additive Regression Trees. BART is a Bayesian nonparametric, machine learning, ensemble predictive modeling method for continuous, binary, categorical and time-to-event outcomes. Furthermore, BART is a tree-based, black-box method which fits the outcome to an arbitrary random function, f, of the covariates. The BART technique is relatively computationally efficient as compared to its competitors, but large sample sizes can be demanding. Therefore, the **BART3** package includes efficient state-of-the-art implementations for continuous, binary, categorical and time-to-event outcomes that can take advantage of modern off-the-shelf hardware and software multi-threading technology. The **BART3** package is written in C++ for both programmer and execution efficiency. The **BART3** package takes advantage of multi-threading via forking as provided by the **parallel** package and OpenMP when available and supported by the platform. The ensemble of binary trees produced by a BART fit can be stored and re-used later via the R `predict` function. In addition to being an R package, the installed BART routines can be called directly from C++. The **BART3** package provides the tools for your BART toolbox.

Keywords: binary trees, black-box, categorical, competing risks, continuous, ensemble predictive model, forking, multinomial, multi-threading, OpenMP, recurrent events, survival analysis.

N.B. This vignette is largely based on our previous work (Sparapani, Spanbauer, and McCulloch 2020) which corresponds to the **BART** R package on CRAN that is frozen at version 2.7. However, this vignette goes beyond that into the new developments found in the **BART3** R package that continues the legacy for version 3 and beyond on **github**. Now, **BART3** is also a C++ header-only class library that can be linked with other R packages via `Imports/LinkingTo` in the `DESCRIPTION` file.

1. Introduction

Bayesian Additive Regression Trees (BART) arose out of earlier research on Bayesian model fitting of an outcome to a single tree (Chipman, George, and McCulloch 1998). In this era from 1996 to 2001, the excellent predictive performance of ensemble models became apparent (Breiman 1996; Krogh and Solich 1997; Freund and Schapire 1997; Breiman 2001; Friedman 2001; Baldi and Brunak 2001). Instead of making a single prediction from a complex model, ensemble models make a single prediction which is the summary of the predictions from many simple models. Generally, ensemble models have desirable properties, e.g., they do not suffer from over-fitting (Kuhn and Johnson 2013). Like bagging (Breiman 1996), boosting (Freund and Schapire 1997; Friedman 2001) and random forests (Breiman 2001), BART relies on an ensemble of trees to predict the outcome; and, although, there are similarities, there are also differences between these approaches.

BART is a Bayesian nonparametric, sum of trees method for continuous, dichotomous, categorical and time-to-event outcomes. Furthermore, BART is a black-box, machine learning method which fits the outcome via an arbitrary random function, f, of the covariates. So-called black-box models generate functions of the covariates which are so complex that interpreting the internal details of the fitted model is generally abandoned in favor of assessment via evaluations of the fitted function, f, at chosen values of the covariates. As shown by Chipman, George, and McCulloch (2010), BART's out-of-sample predictive performance is generally equivalent to, or exceeds that, of alternatives like lasso with L1 regularization (Efron, Hastie, Johnstone, and Tibshirani 2004) or black-box models such as gradient boosting (Freund and Schapire 1997; Friedman 2001), neural nets with one hidden layer (Ripley 2007; Venables and Ripley 2013) and random forests (Breiman 2001). Over-fitting is the tendency to overly fit a model to an in-sample training data set at the expense of poor predictive performance for unseen out-of-sample data. Typically, BART does not over-fit to the training data due to the regularization tree-branching penalty of the BART prior, i.e., generally, each tree has few branches and plays a small part in the overall fit. So, the resulting fit from the ensemble of trees as a whole is generally a good fit that does not over-fit. Essentially, BART is a Bayesian nonlinear model with all the advantages of the Bayesian paradigm such as posterior inference including point and interval estimation. Conveniently, BART naturally scales to large numbers of covariates and facilitates variable selection; it does not require the covariates to be rescaled; neither does it require the covariate functional relationship, nor the interactions considered, to be pre-specified.

In this article, we give an overview of data analysis with BART and the **BART3** R package. In Section 2, we describe the R functions provided by the **BART3** package for analyzing continuous outcomes with BART. In Section 3, we demonstrate the typical usage of BART via the classic example of Boston housing values. In Section 4, we describe how BART can be used to analyze binary and categorical outcomes. In Section 5, we describe how BART can be used to analyze time-to-event outcomes with censoring including competing risks and recurrent events. In Appendix Section A, we describe how to get and install the **BART3** package. In Appendix Section B, we describe the basis of BART on binary trees along with the details of the BART prior. In Appendix Section C, we briefly describe the posterior computations required to use BART. In Appendix Section D, we describe how to perform the BART computations efficiently by resorting to parallel processing with multi-threading (N.B. by default, the Microsoft Windows operating system does not provide the multi-threading

interfaces employed by the R environment; in lieu of the provision, the **BART3** package is single-threaded on Windows, yet, otherwise, completely functional; see Appendix D for more details).

2. Continuous outcomes with BART

In this section, we document the analysis of continuous outcomes with the **BART3** R package. We provide two functions for continuous outcomes: 1) `wbart` named for weighted BART; and 2) `gbart` named for generic, or generalized, BART. Both functions have roughly the same functionality. `wbart` has a verbose interface while `gbart` is streamlined. Also, `wbart` is for continuous outcomes only whereas `gbart` also supports binary outcomes.

Typically, when calling the `wbart` and `gbart` functions, many of the arguments can be omitted since the default values are adequate for most purposes. However, there are certain common arguments which are either always needed or frequently provided. The `wbart` (`mc.wbart`) and `gbart` (`mc.gbart`) functions are for serial (parallel) computation; for more details on parallel computation see the Appendix Section D. The outcome `y.train` is a vector of numeric values. The covariates for training (validation, if any) are `x.train` (`x.test`) which can be matrices or data frames containing factors; in the display below, we assume matrices for simplicity. N.B. throughout we denote integer constants by upper case letters, e.g., in the following display: M for the number of posterior samples, B for the number of threads (generally, $B = 1$ for Windows), N for the number of observations in the training set, and Q for the number of observations in the test set.

```
R> set.seed(99)
R> post <- wbart(x.train, y.train, x.test, ndpost=M)
R> post <- mc.wbart(x.train, y.train, x.test, ndpost=M, mc.cores=B, seed=99)
R> post <- gbart(x.train, y.train, x.test, ndpost=M)
R> post <- mc.gbart(x.train, y.train, x.test, ndpost=M, mc.cores=B, seed=99)
```

Input matrices, `x.train` and, optionally, `x.test`: $\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_N \end{bmatrix}$ made up of $\boldsymbol{x}_i$ as row vectors

`post`, of type `wbart`, which is essentially a list

`post$yhat.train` and `post$yhat.test`: $\begin{bmatrix} \hat{y}_{11} & \cdots & \hat{y}_{N1} \\ \vdots & \ddots & \vdots \\ \hat{y}_{1M} & \cdots & \hat{y}_{NM} \end{bmatrix}$ $\hat{y}_{im} = \mu_0 + f_m(\boldsymbol{x}_i)$ mth posterior draw

The columns of `post$yhat.train` and `post$yhat.test` represent different covariate settings and the rows, the `M` draws from the posterior.

Often it is impractical to provide `x.test` in the call to `wbart`/`gbart` due to the large number of predictions considered, or all of the settings to be evaluated are not known at that time. To allow for this common problem, the **BART3** package returns the trees encoded in an ASCII string, `treedraws$trees`, and provides a `predict` function to generate any predictions

needed (more details on trees and, the string representation of trees, can be found in Appendix Section B). Note that if you need to perform the prediction in some later R instance, then you can save the `wbart` object returned and reload it when needed, e.g., save with `saveRDS(post, "post.rds")` and reload, `post <- readRDS("post.rds")` . The `x.test` input can be a matrix or a data frame; for simplicity, we assume a matrix below.
For serial computation

```
R> pred <- predict(post, x.test)
```

For parallel computation

```
R> pred <- predict(post, x.test, mc.cores=B)
```

$$\text{Input: } \texttt{x.test}: \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_Q \end{bmatrix} \text{ made up of } \boldsymbol{x}_h \text{ as row vectors}$$

$$\texttt{pred} \text{ is a matrix: } \begin{bmatrix} \hat{y}_{11} & \cdots & \hat{y}_{Q1} \\ \vdots & \ddots & \vdots \\ \hat{y}_{1M} & \cdots & \hat{y}_{QM} \end{bmatrix} \text{ where } \hat{y}_{hm} = \mu_0 + f_m(\boldsymbol{x}_h)$$

2.1. Posterior samples returned

The number of MCMC samples discarded for burn-in is specified by the `nskip` argument and the default is 100. The number of MCMC samples returned is specified by the `ndpost` argument and the default is 1000. Returning every l^{th} value, or thinning, can be specified by the `keepevery` argument which defaults to 1, i.e., no thinning. Some, but not all, returned values can be thinned. The following arguments are available with `wbart` and default to `ndpost`, but can be over-ridden as needed (with `gbart`, `ndpost` draws are always returned and can't be over-ridden).

- `nkeeptrain` : number of f draws to return corresponding to `x.train`
- `nkeeptest` : number of f draws to return corresponding to `x.test`
- `nkeeptestmeam` : number of f draws to use in computing `yhat.test.mean`
- `nkeeptreedraws` : number of tree ensemble draws to return for use with `predict`

Members of the object returned (which is essentially a list) include `varprob` and `varcount` which correspond to the variable selection probability and the observed counts in the ensemble of trees. When `sparse=TRUE`, `varprob` is the random variable selection probability, s_j; otherwise, it is the fixed constant $s_j = P^{-1}$. Besides the posterior samples, also the mean over the posterior is provided as `varprob.mean` and `varcount.mean`.

3. The Boston housing values example

Now, let's examine the classic Boston housing values example (Harrison Jr and Rubinfeld 1978). This data is from the 1970 US Census where each observation represents a Census tract in the Boston Standard Metropolitan Statistical Area. For each tract, there was a localized air pollution estimate, the concentration of nitrogen oxides, based on a meteorological model that was calibrated to monitoring data. Restricted to tracts with owner-occupied homes, there are 506 observations. We'll predict the median value of owner-occupied homes (in thousands of dollars truncated at 50), `y=mdev`, from two covariates: `rm` and `lstat`. `rm` is the number of rooms defined as the average number of rooms for owner-occupied homes. `lstat` is the percent of population that is lower status defined as the average of the proportion of adults without any high school education and the proportion of male workers classified as laborers. Below, we present several observations of the data and scatter plots in Figure 1.

```
R> library("MASS")
R> x = Boston[, c(6, 13)]
R> y = Boston$medv
R> head(cbind(x, y))
```

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

```
R> par(mfrow=c(2, 2))
R> plot(x[, 1], y, xlab="x1=rm", ylab="y=mdev")
R> plot(x[, 2], y, xlab="x2=lstat", ylab="y=mdev")
R> plot(x[, 1], x[, 2], xlab="x1=rm", ylab="x2=lstat")
R> par(mfrow=c(1, 1))
```

3.1. wbart for continuous outcomes

In this example, we fit the following BART model for continuous outcomes:

$$y_i = \mu_0 + f(x_i) + \epsilon_i \text{ where } \epsilon_i \sim N(0, \sigma^2)$$
$$(f, \sigma^2) \overset{\text{prior}}{\sim} \text{BART}$$

with i indexing subjects; $i = 1, \ldots, N$. We use Markov chain Monte Carlo (MCMC) to get draws from the posterior distribution of the parameter (f, σ^2).

```
R> library("BART")
R> set.seed(99)
R> nd = 200
R> burn = 50
R> post = wbart(x, y, nskip=burn, ndpost=nd)
```

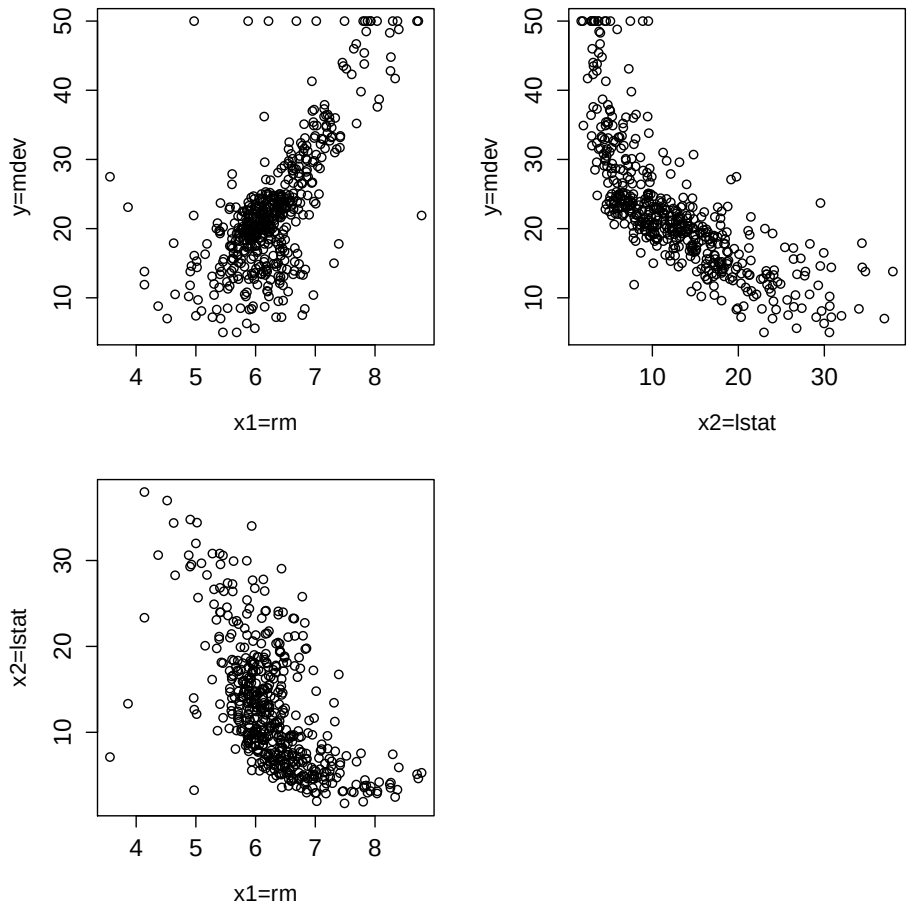


Figure 1: The Boston housing data was compiled from the 1970 US Census where each observation represents a Census tract in Boston with owner-occupied homes. For each tract, we have the median value of owner-occupied homes (in thousands of dollars truncated at 50), `y=mdev`, the average number of rooms, `x1=rm`, and the percent of the population that is lower status, `x2=lstat`. Here, we show scatter plots of the data.

```
*****Into main of wbart
*****Data:
data:n,p,np: 506, 2, 0
y1,yn: 1.467194, -10.632806
x1,x[n*p]: 6.575000, 7.880000
*****Number of Trees: 200
*****Number of Cut Points: 100 ... 100
*****burn and ndpost: 50, 200
*****Prior:beta,alpha,tau,nu,lambda: 2.000000,0.950000,0.795495,3.000000,5.979017
*****sigma: 5.540257
*****w (weights): 1.000000 ... 1.000000
*****Dirichlet:sparse,a,b,rho,augment: 0,0.5,1,2,0
*****nkeeptrain,nkeeptest,nkeeptestme,nkeeptreedraws: 200,200,200,200
*****printevery: 100
*****skiptr,skipte,skipteme,skiptreedraws: 1,1,1,1

MCMC
done 0 (out of 250)
done 100 (out of 250)
done 200 (out of 250)
time: 1s
check counts
trcnt,tecnt,temecnt,treedrawscnt: 200,0,0,200
```

3.2. Results returned from wbart

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

```
R> names(post)

 [1] "sigma"           "yhat.train.mean" "yhat.train"      "yhat.test.mean"
 [5] "yhat.test"       "varcount"        "varprob"         "treedraws"
 [9] "mu"              "varcount.mean"   "varprob.mean"    "rm.const"

R> length(post$sigma)

[1] 250

R> length(post$yhat.train.mean)

[1] 506

R> dim(post$yhat.train)

[1] 200 506
```

Remember, the training data has $n = 506$ observations, we had `burn`=50 burn-in discarded draws and `nd`=M=200 draws kept. Let's look at a couple of the key list components.
`$sigma`: both the 50 burn-in and 250 draws are kept for σ; burn-in are kept only for this parameter.
`$yhat.train`: the mth row and ith column is $f_m(x_i)$ (the m^{th} kept MCMC draw evaluated at the i^{th} training observation).
`$yhat.train.mean`: the posterior estimate of $f(x_i)$, i.e., $M^{-1}\sum_m f_m(x_i)$.

3.3. Assessing convergence with wbart

As with any high-dimensional MCMC, assessing convergence may be non-trivial. Posterior convergence diagnostics are recommended for BART especially with large data sets and/or a large number of covariates. Besides diagnostics, routine counter-measures such as longer chains, thinning and multiple chains may be warranted. For continuous outcomes, the simplest thing to look at are the draws of σ. See Section 4.5 for a primer on other convergence diagnostic options for binary and categorical outcomes that are also applicable for continuous outcomes.

Assessing convergence in this example, the parameter σ is the only identified parameter in the model and, of course, it is indicative of the size of the errors.

```
R> plot(post$sigma, type="l")
R> abline(v=burn, lwd=2, col="red")
```

In Figure 2, you can see that BART burned in very quickly. Just one initial draw looking a bit bigger than the rest. Apparently, subsequent variation is legitimate posterior variation. In a more difficult problem you may see the σ draws initially declining as the MCMC searches for a good fit.

3.4. wbart and linear regression compared

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

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

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

```
R> pairs(fitmat)
```

In Figure 3, we present scatter plots between `mdev`, the BART fit and the multiple linear regression. The BART fit is noticeably different from the linear fit.

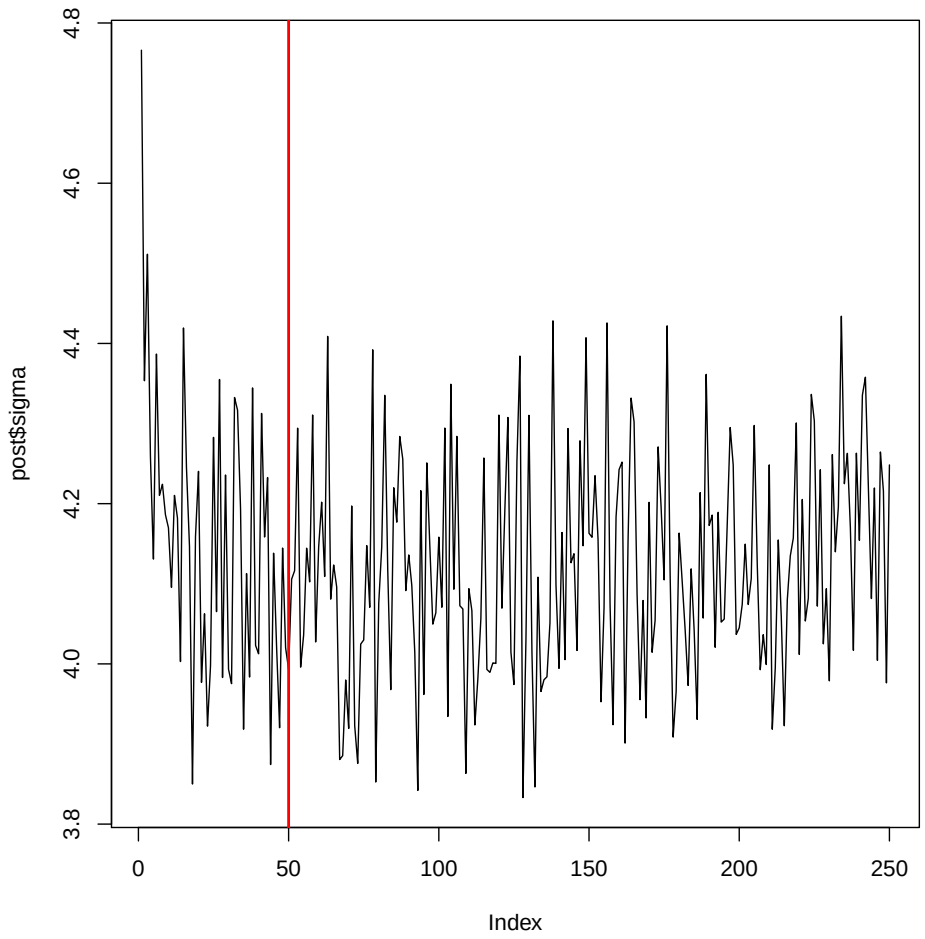


Figure 2: The Boston housing data was compiled from the 1970 US Census where each observation represents a Census tract in Boston with owner-occupied homes. For each tract, we have the median value of owner-occupied homes (in thousands of dollars truncated at 50), `y=mdev`, the average number of rooms, `x1=rm`, and the percent of the population that is lower status, `x2=lstat`. With BART, we predict `y=mdev` from `rm` and `lstat`. Here, we show a trace plot of the error variance, σ, that demonstrates convergence for BART rather quickly, i.e., by 50 iterations or earlier.

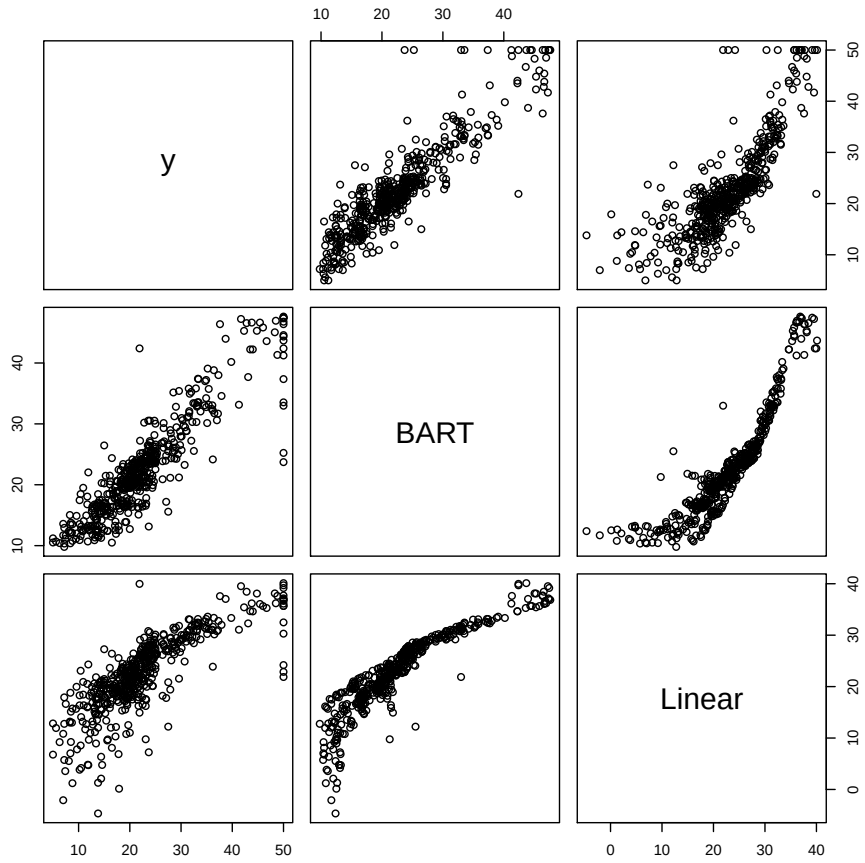


Figure 3: The Boston housing data was compiled from the 1970 US Census where each observation represents a Census tract in Boston with owner-occupied homes. For each tract, we have the median value of owner-occupied homes (in thousands of dollars truncated at 50), `y=mdev`, the average number of rooms, `x1=rm`, and the percent of the population that is lower status, `x2=lstat`. With BART, we predict `y=mdev` from `rm` and `lstat`. Here, we show scatter plots comparing `y=mdev`, the BART fit ("BART") and multiple linear regression ("Linear").

3.5. Prediction and uncertainty with wbart

In Figure 4, we order the observations by the fitted house value (`yhat.train.mean`) and then use boxplots to display the draws of $f(x)$ in each column of `yhat.train`.

```
R> i = order(post$yhat.train.mean)
R> boxplot(post$yhat.train[, i])
```

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

3.6. Using the predict function with wbart

We can get out of sample predictions in two ways. First, we can can just ask for them when we call `wbart` by supplying a matrix or data frame of test x values. Second, we can call a `predict` method. Now, let's split our data into train and test subsets.

```
R> n = length(y)
R> set.seed(14)
R> i = sample(1:n, floor(0.75*n))
R> x.train = x[i, ]; y.train=y[i]
R> x.test = x[-i, ]; y.test=y[-i]
R> cat("training sample size = ", length(y.train), "\n")
R> cat("testing sample size = ", length(y.test), "\n")

training sample size = 379
testing sample size = 127
```

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

```
R> set.seed(99)
R> post1 = wbart(x.train, y.train, x.test)
R> dim(post1$yhat.test)

[1] 1000  127

R> length(post1$yhat.test.mean)

[1] 127
```

The testing data is handled similarly to the training data.
`$yhat.test`: the mth row and hth column is $f_m(x_h)$ (the m^{th} kept MCMC draw evaluated at the h^{th} testing observation).
`$yhat.test.mean`: the posterior estimate of $f(x_h)$, i.e., $Q^{-1}\sum_m f_m(x_h)$.
Alternatively, we could run `wbart` saving all the MCMC results and then call `predict`.

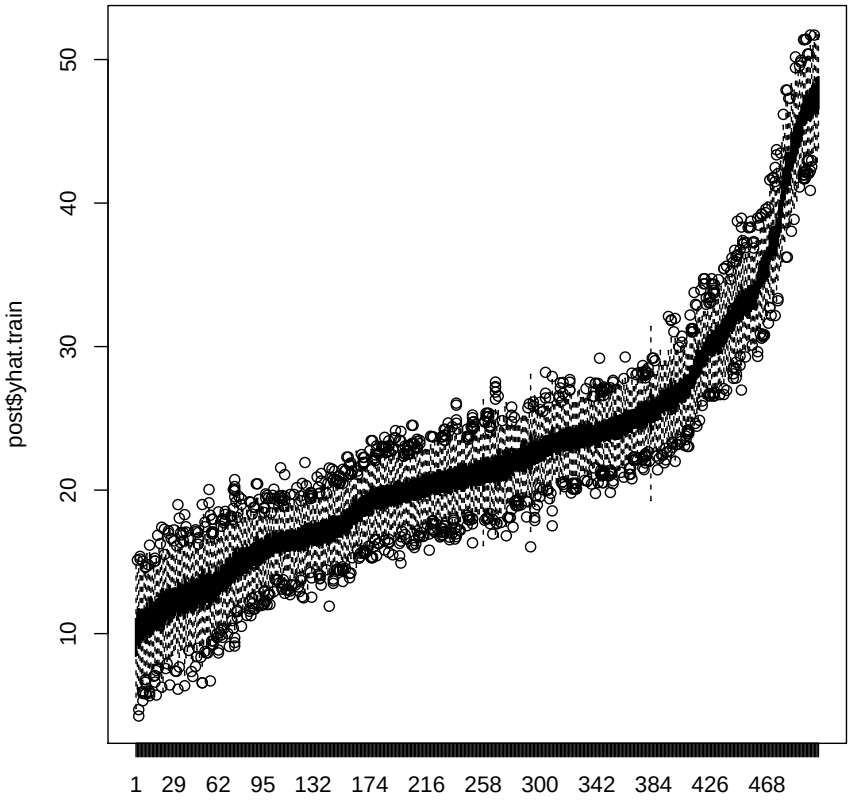


Figure 4: The Boston housing data was compiled from the 1970 US Census where each observation represents a Census tract in Boston with owner-occupied homes. For each tract, we have the median value of owner-occupied homes (in thousands of dollars truncated at 50), `mdev`, the average number of rooms, `rm`, and the percent of the population that is lower status, `lstat`. With BART, we predict `y=mdev` from `rm` and `lstat`. Here, we show boxplots of the posterior samples of predictions (on the y-axis) ordered by the average predicted home value per tract (on the x-axis).

```
R> set.seed(99)
R> post2 = wbart(x.train, y.train)
R> yhat = predict(post2, x.test)

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

R> dim(yhat)

[1] 1000 127

R> summary(as.double(yhat - post1$yhat.test))

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

So `yhat` and `post1$yhat.test` are practically identical.

3.7. wbart and thinning

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

```
R> set.seed(4)
R> post3 = wbart(x.train, y.train, nskip=1000, ndpost=10000,
+    nkeeptrain=0, nkeeptest=0, nkeeptestmean=0, nkeeptreedraws=200)
R> yhatthin = predict(post3, x.test)

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

R> dim(post3$yhat.train)

[1]   0 379

R> dim(yhatthin)
```

```
[1] 200 127
```

Now, there are no kept draws of $f(x)$ for training x, and we have 200 tree ensemble draws to use with `predict`. Of course, if we keep 200 out of 10000, then every 50th draw is kept.

The default values are to keep all the draws (e.g., `nkeeptrain=ndpost`). Now, let's have a look at the predictions.

```
R> fmat = cbind(y.test, post1$yhat.test.mean, apply(yhatthin, 2, mean))
R> colnames(fmat) = c("y", "yhat", "yhatThin")
R> pairs(fmat)
```

In Figure 5, we present scatter plots between `mdev`, "yhat" and "yhatThin". Recall, the predictions labeled "yhat" are from a BART run with `seed=99` and all default values. The predictions labeled "yhatThin" are thinned by 50 (after 1000 burnin discarded, 200 kept out of 10000 draws) with `seed=4`. It is very interesting how similar they are!

3.8. wbart and Friedman's partial dependence function

BART does not directly provide a summary of the effect of a single covariate, or a subset of covariates, on the outcome. This is also the case for black-box, or nonparametric regression, models in general that need to deal with this same issue. Developed for such complex models, Friedman's partial dependence function (Friedman 2001) can be employed with BART to summarize the marginal effect due to a subset of the covariates. Friedman's partial dependence function is a concept that is very flexible. So flexible that we are unable to provide abstract functional support in the **BART3** package; rather, we provide examples of the many practical uses in the `demo` directory.

We use S to denote the indices of the covariates in the subset and the collection itself, i.e., define the row vector for test setting h as $\boldsymbol{x}_{hS} = [x_{hj}]$ where $j \in S$. Similarly, we denote the complement of the subset as C with $S \cup C$ spanning all covariates. The complement row vector for training observation i is $\boldsymbol{x}_{iC} = [x_{ij}]$ where $j \in C$. The marginal dependence function is defined by fixing the subset at a test setting while aggregating over the training observations of the complement covariates: $f(\boldsymbol{x}_{hS}) = N^{-1} \sum_{i=1}^{N} f(\boldsymbol{x}_{hS}, \boldsymbol{x}_{iC})$. Other marginal functions can be obtained in a similar fashion. Estimates can be derived via functions of the posterior samples such as means, quantiles, etc., e.g., $\hat{f}(\boldsymbol{x}_{hS}) = M^{-1} N^{-1} \sum_{m=1}^{M} \sum_{i=1}^{N} f_m(\boldsymbol{x}_{hS}, \boldsymbol{x}_{iC})$ where m indexes posterior samples. However, care must be taken in the interpretation of the marginal effect as estimated by Friedman's partial dependence function. If there are strong relationships among the covariates, it may be unrealistic to assume that individual covariates can be manipulated independently.

For example, suppose that we want to summarize the median home value, `medv` (variable 14 of the `Boston` data frame), by the percent of the population with lower status, `lstat` (variable 13), while aggregating over the other twelve covariates in the Boston housing data. In Figure 6, we demonstrate the marginal estimate and its 95% credible interval.

```
R> x.train = as.matrix(Boston[i, -14])
R> set.seed(12)
R> post4 = wbart(x.train, y.train)
R> H = floor(0.75*length(y.train))
```

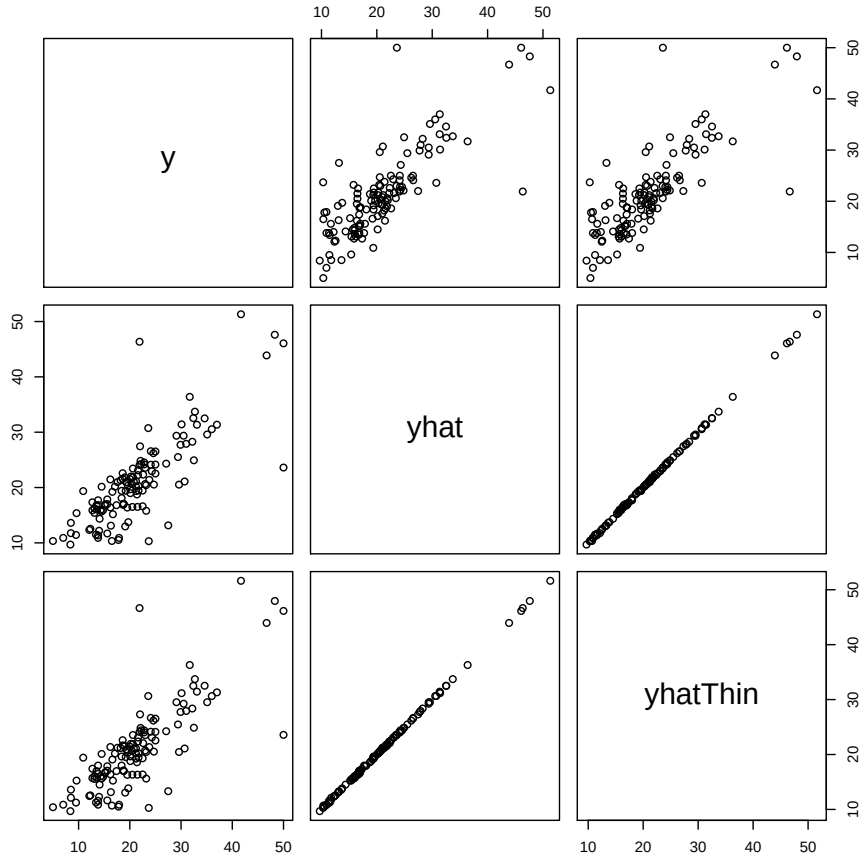


Figure 5: The Boston housing data was compiled from the 1970 US Census where each observation represents a Census tract in Boston with owner-occupied homes. For each tract, we have the median value of owner-occupied homes (in thousands of dollars truncated at 50), `mdev`, the average number of rooms, `rm`, and the percent of the population that is lower status, `lstat`. With BART, we predict `y=mdev` by `rm` and `lstat`. The predictions labeled "yhat" are from a BART run with `seed=99` and all default values. The predictions labeled "yhatThin" are thinned by 50 (after 1000 burnin discarded, 200 kept out of 10000 draws) with `seed=4`. It is very interesting how similar they are!

```
R> L = 41
R> x = seq(min(x.train[, 13]), max(x.train[, 13]), length.out=L)
R> x.test = cbind(x.train[, -13], x[1])
R> for(j in 2:L)
+    x.test = rbind(x.test, cbind(x.train[, -13], x[j]))
R> pred = predict(post4, x.test)
R> partial = matrix(nrow=1000, ncol=L)
R> for(j in 1:L) {
R>   h = (j - 1) * H + 1:H
R>   partial[, j] = apply(pred[, h], 1, mean)
R> }
R> plot(x, apply(partial, 2, mean), type='l',
+    xlab='percent lower status', ylab='median home value',
+    ylim=c(10, 50))
R> lines(x, apply(partial, 2, quantile, probs=0.025), lty=2)
R> lines(x, apply(partial, 2, quantile, probs=0.975), lty=2)
```

Besides the marginal effect, we can define the conditional effect of $x_1|x_2$ as $\frac{f(x_1+\delta,x_2)-f(x_1,x_2)}{\delta}$. However, BART is not fitting simple linear functions. For example, suppose the data follows a sufficiently complex function like so: $f(x_1, x_2) = b_1x_1 + b_2x_1^2 + b_3x_1x_2$. Then the conditional effect that BART is likely to fit is approximately $b_1 + 2b_2x_1 + b_2\delta + b_3x_2$. This function is not so easy to characterize (as the marginal effect) since it involves x_1, x_2 and δ. Nevertheless, these functions can be estimated by BART if these inputs are provided. But, these functions have the same limitations as Friedman's partial dependence function and, perhaps, even moreso. See the conditional effect example at the end of `demo("boston.R", package="BART")`.

4. Binary and categorical outcomes with BART

The **BART3** package supports binary outcomes via probit BART with Normal latents and logit BART with Logistic latents. Categorical outcomes are supported with Multinomial BART which defaults to probit for computational efficiency, but logit is available as an option. Convergence diagnostics are provided and variable selection as well.

4.1. Probit BART for binary outcomes

Probit BART for binary outcomes is provided by the **BART3** package as the `pbart` and `gbart` functions. In this case, the outcome, `y.train`, is an integer with values of 0 or 1. The model is as follows with i indexing subjects: $i = 1, \ldots, N$.

$$y_i|p_i \overset{\text{ind}}{\sim} \mathrm{B}(p_i) \text{ where } \mathrm{B}(.) \text{ is the Bernoulli distribution}$$
$$p_i = \Phi(\mu_0 + f(\boldsymbol{x}_i)) \text{ where } f \overset{\text{prior}}{\sim} \text{BART and } \Phi(.) \text{ is the standard Normal cdf}$$

This setup leads to the following likelihood: $[\boldsymbol{y}|f] = \prod_{i=1}^{N} p_i^{y_i}(1-p_i)^{1-y_i}$.

To extend BART to binary outcomes, we employ the technique of Albert and Chib (1993) that assumes there is an unobserved latent, z_i, where $y_i = \mathrm{I}(z_i > 0)$ and $i = 1, \ldots, n$ indexes subjects. Given y_i, we generate the truncated Normal latents, z_i; these auxiliary latents are

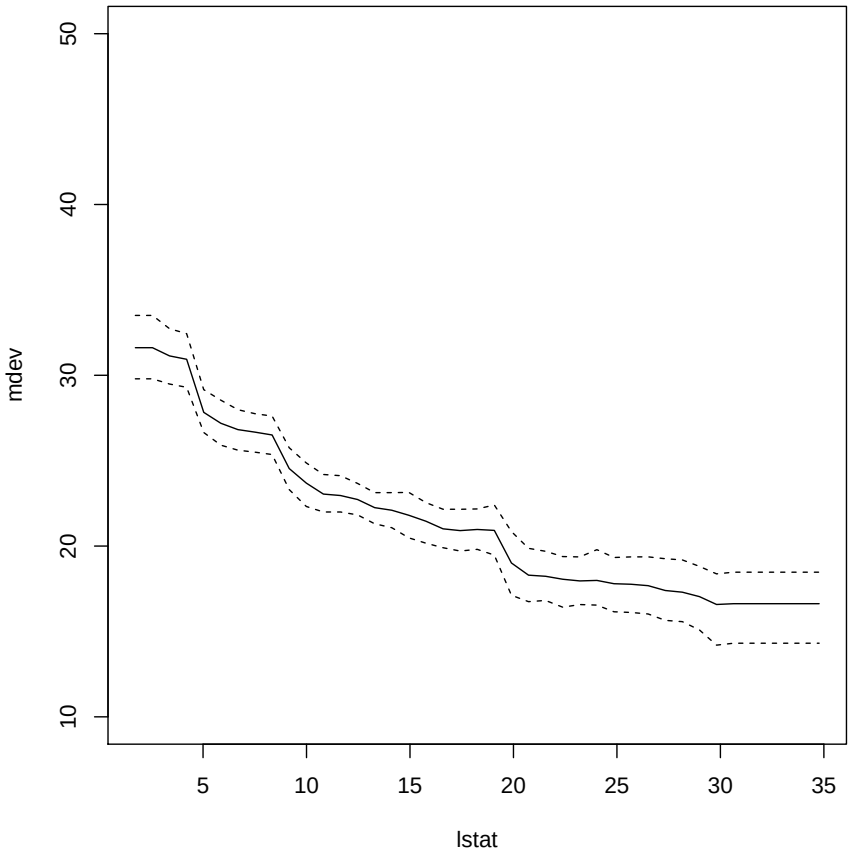


Figure 6: The Boston housing data was compiled from the 1970 US Census where each observation represents a Census tract in Boston with owner-occupied homes. For each tract, we have the median value of owner-occupied homes (in thousands of dollars truncated at 50), `mdev`, and the percent of the population that is lower status, `lstat`, along with eleven other covariates. We summarize the marginal effect of `lstat` on `mdev` while aggregating over the other covariates with Friedman's partial dependence function. The marginal estimate and its 95% credible interval are shown.

efficiently sampled (Robert 1995) and recast as the outcome for a continuous BART with unit variance as follows.

$$z_i|y_i, f \sim \mathrm{N}\left(\mu_0 + f(\boldsymbol{x}_i),\ 1\right) \begin{cases} \mathrm{I}\left(-\infty, 0\right) & \text{if } y_i = 0 \\ \mathrm{I}\left(0, \infty\right) & \text{if } y_i = 1 \end{cases}$$

Centering the latent z_i around the constant μ_0 is analogous to quasi-centering the probabilities, p_i, at $p_0 = \Phi(\mu_0)$, i.e., $\mathrm{E}\left[p_i\right]$ is approximately equal to p_0 which is all that is necessary for inference to be performed. The default value of μ_0 is $\Phi^{-1}(\bar{y})$ (which you can over-ride with the `binaryOffset` argument).

The `pbart` (`mc.pbart`) and `gbart` (`mc.gbart`) functions are for serial (parallel) computation. The outcome `y.train` is a vector containing zeros and ones. The covariates for training (validation, if any) are `x.train` (`x.test`) which can be matrices or data frames containing factors; in the display below, we assume matrices for simplicity. Notation: M for the number of posterior samples, B for the number of threads (generally, $B = 1$ for Windows), N for the number of observations in the training set, and Q for the number of observations in the test set.

```
R> set.seed(99)
R> post <- pbart(x.train, y.train, x.test, ndpost=M)
R> post <- mc.pbart(x.train, y.train, x.test, ndpost=M, mc.cores=B, seed=99)
R> post <- gbart(x.train, y.train, x.test, type='pbart', ndpost=M)
R> post <- mc.gbart(x.train, y.train, x.test, type='pbart', ndpost=M, ...
```

N.B. for `pbart`, the thinning argument, `keepevery` defaults to 1 while for `gbart` with `type='pbart'`, `keepevery` defaults to 10.

Input matrices: `x.train` and, optionally, `x.test`: $\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_N \end{bmatrix}$ or $\boldsymbol{x}_i$ as row vectors

`post`, of type `pbart`, which is essentially a list

`post$yhat.train` and `post$yhat.test`: $\begin{bmatrix} \hat{y}_{11} & \cdots & \hat{y}_{N1} \\ \vdots & \ddots & \vdots \\ \hat{y}_{1M} & \cdots & \hat{y}_{NM} \end{bmatrix}$ $\hat{y}_{im} = \mu_0 + f_m(\boldsymbol{x}_i)$

The columns of `post$yhat.train` and `post$yhat.test` represent different covariate settings and the rows, the `M` draws from the posterior. `post$yhat.train` and `post$yhat.test`, when requested, are returned, although, `post$prob.train` and `post$prob.test` are generally of more interest (and `post$prob.train.mean` and `post$prob.test.mean` which are the means of the posterior sample columns, not shown).

`post$prob.train` and `post$prob.test`: $\begin{bmatrix} \hat{p}_{11} & \cdots & \hat{p}_{N1} \\ \vdots & \ddots & \vdots \\ \hat{p}_{1M} & \cdots & \hat{p}_{NM} \end{bmatrix}$ where $\hat{p}_{im} = \Phi(\hat{y}_{im})$

Often it is impractical to provide `x.test` in the call to `pbart` due to the number of predictions considered or all the settings to evaluate are simply not known at that time. To allow for this common problem, the **BART3** package returns the trees encoded in an ASCII string, `treedraws$trees`, and provides a `predict` function to generate any predictions needed. Note that if you need to perform the prediction in some later R instance, then you can save the `pbart` object returned and reload it when needed, e.g., save with `saveRDS(post, "post.rds")` and reload, `post <- readRDS("post.rds")` . The `x.test` input can be a matrix or a data frame; for simplicity, we assume a matrix below.

```
R> pred <- predict(post, x.test, mc.cores=B)
```

$$\texttt{Input: x.test:} \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_Q \end{bmatrix} \text{ or } \boldsymbol{x}_h \text{ as row vectors}$$

`pred`, of type `pbart`, which is essentially a list

$$\texttt{pred\$yhat.test:} \begin{bmatrix} \hat{y}_{11} & \cdots & \hat{y}_{Q1} \\ \vdots & \ddots & \vdots \\ \hat{y}_{1M} & \cdots & \hat{y}_{QM} \end{bmatrix} \text{ where } \hat{y}_{hm} = \mu_0 + f_m(\boldsymbol{x}_h)$$

$$\texttt{pred\$prob.test:} \begin{bmatrix} \hat{p}_{11} & \cdots & \hat{p}_{Q1} \\ \vdots & \ddots & \vdots \\ \hat{p}_{1M} & \cdots & \hat{p}_{QM} \end{bmatrix} \text{ where } \hat{p}_{hm} = \Phi(\hat{y}_{hm})$$

$$\texttt{pred\$prob.test.mean: } [\hat{p}_1, \ldots, \hat{p}_Q] \text{ where } \hat{p}_h = M^{-1} \sum_{m=1}^{M} \hat{p}_{hm}$$

4.2. Probit BART and Friedman's partial dependence function

For an overview of Friedman's partial dependence function (including the notation adopted in this article and its meaning), please see Section 3.8 which discusses continuous outcomes. For probit BART, the f function is not directly of interest; rather, the probability of an event is more interpretable: $p(\boldsymbol{x}_{hS}) = N^{-1} \sum_{i=1}^{N} \Phi(\mu_0 + f(\boldsymbol{x}_{hS}, \boldsymbol{x}_{iC}))$.

Probit BART example: chronic pain and obesity

We want to explore the hypothesis that obesity is a risk factor for chronic lower-back pain (which includes buttock pain in this definition). A corollary to this hypothesis is that obesity is not considered to be a risk factor for chronic neck pain. A good source of data for this question is available in the National Health and Nutrition Examination Survey (NHANES) 2009-2010 Arthritis Questionnaire. 5106 subjects were surveyed. We will use probit BART to analyze the dichotomous outcomes of chronic lower-back pain and chronic neck pain. We restrict our attention to the following covariates: age, gender and anthropometric measurements including weight (kg), height (cm), body mass index (kg/m^2) and waist circumference (cm). Also, note

that survey sampling weights are available to extrapolate the rates from the survey to the US population as a whole. We will concentrate on body mass index (BMI) and gender, $\boldsymbol{x}_{hS}$, while utilizing Friedman's partial dependence function as defined above and incorporating the survey weights, i.e., $p_{hS}(\boldsymbol{x}_{hS}) = \sum_{i=1}^{N} w_i \Phi(\mu_0 + f(\boldsymbol{x}_{hS}, \boldsymbol{x}_{iC}))/\sum_{i'=1}^{N} w_{i'}$.

The **BART3** package provides two examples for the relationship between chronic pain and BMI: `demo("nhanes.pbart1", package="BART")` for the probabilities and `demo("nhanes.pbart2", package="BART")` for differences in these probabilities. In Figure 7, the left panel for lower-back pain and the right panel for neck pain, the unweighted relationship between chronic pain, BMI and gender are displayed: males (females) are represented by blue (red) solid lines with corresponding 95% credible intervals in dashed lines. Although there is a generous amount of uncertainty, it does not appear that the probability of chronic lower-back pain increases with BMI for either gender. Conversely, chronic neck pain does appear to be rising, yet again, the intervals are wide. In both cases, these findings are not anticipated given the original hypotheses. Based on survey weights (not shown), the results are basically the same. In Figure 8, the unweighted relationship for females between BMI and the difference in probability of chronic pain from a baseline BMI of 25 (which is the upper limit of normal) with corresponding 95% credible intervals in dashed lines: the left panel for lower-back pain (blue solid lines) and the right panel for neck pain (red solid lines). Again, we have roughly the same impression, i.e., there is no increase of lower-back chronic pain with BMI and it is possibly dropping while neck pain might be increasing, but the intervals are wide for both. The results are basically the same for males (not shown).

4.3. Logit BART for binary outcomes

Assuming a Normal distribution of the unobserved latent, z_i where $y_i = \mathrm{I}(z_i > 0)$, provides some challenges when estimating very small or very large probabilities, p_i, since the Normal distribution has relatively thin tails. This restriction can be relaxed by assuming the latents follow the Logistic distribution which has heavier tails. For Logistic latents, we employ a variant of the Holmes and Held (2006) technique by Gramacy and Polson (2012) to create what we call logit BART. However, it is important to recognize that logit BART is more computationally intensive than probit BART.

The outcome, `y.train`, is provided as an integer with values 0 or 1. Logit BART is provided by the `lbart` and `gbart` functions. Unlike probit BART where the auxiliary latents, z_i, have a unit variance $\sigma^2 = 1$; with Logisitic BART, we sample truncated Normal latents, z_i, with a variance σ_i^2 by the Robert (1995) technique. If $\sigma_i^2 = 4\psi_i^2$ where ψ_i is sampled from the Kolmogorov-Smirnov distribution, then z_i follow the Logistic distribution. Sampling from the Kolmogorov-Smirnov distribution is described by Devroye (1986). So, the conditionally Normal latents, $z_i|\sigma_i^2$, are the outcomes for a continuous BART with a given heteroskedastic variance, σ_i^2.

The z_i are centered around a known constant, μ_0, which is analagous to quasi-centering the probabilities, p_i, around $p_0 = F(\mu_0)$ where F is the standard Logistic distribution function. The default value of μ_0 is $F^{-1}(\bar{y})$ (which you can over-ride with the `binaryOffset` argument to `lbart` or the `offset` argument to `gbart`). Therefore, the probabilities are $p_i = F(\mu_0 + f(\boldsymbol{x}_i))$.

The input and output for `lbart` is essentially identical to `pbart`. Also, the `predict` function for objects of type `lbart` is analogous. The `gbart` function performs logit BART when passed the `type='lbart'` argument.

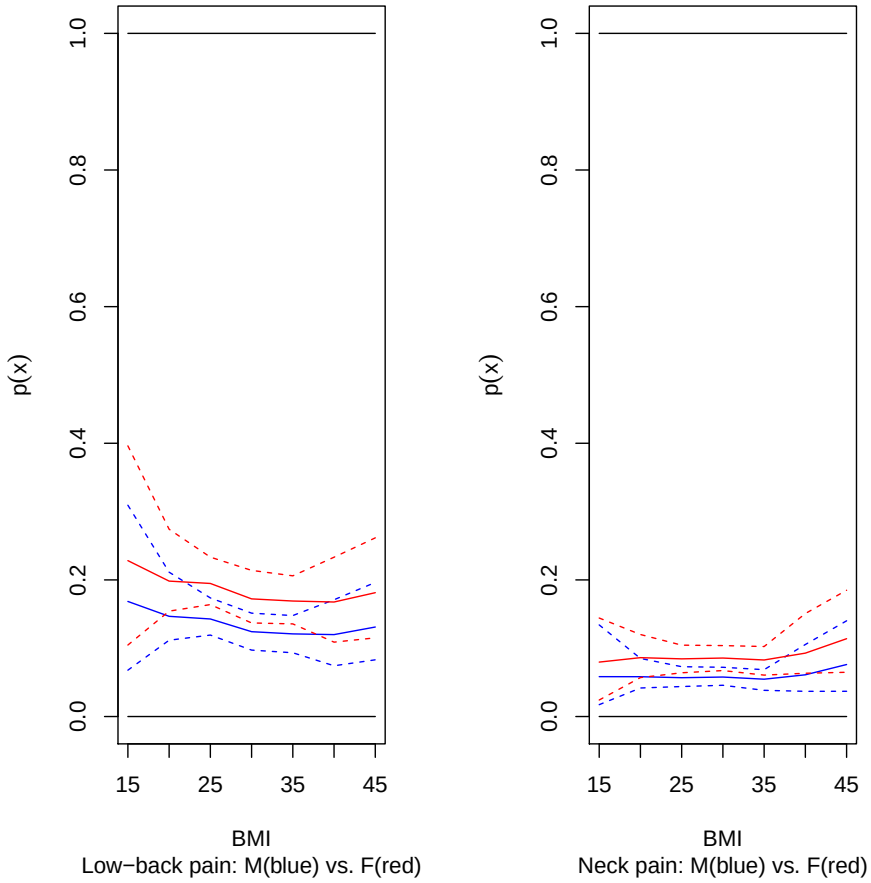


Figure 7: NHANES, BMI and the probability of chronic pain: the left panel for lower-back pain and the right panel for neck pain. The unweighted Friedman's partial dependence relationship between chronic pain, BMI and gender are displayed as ascertained from NHANES data: males (females) are represented by blue (red) lines with the corresponding 95% credible intervals (dashed lines). We want to explore the hypothesis that obesity is a risk factor for chronic lower-back pain (which includes buttock pain in this definition). A corollary to this hypothesis is that obesity is not considered to be a risk factor for chronic neck pain. Although there is a generous amount of uncertainty, it does not appear that the probability of chronic lower-back pain increases with BMI for either gender. Conversely, chronic neck pain does appear to be rising, yet again, the intervals are wide. In both cases, these findings are not anticipated.

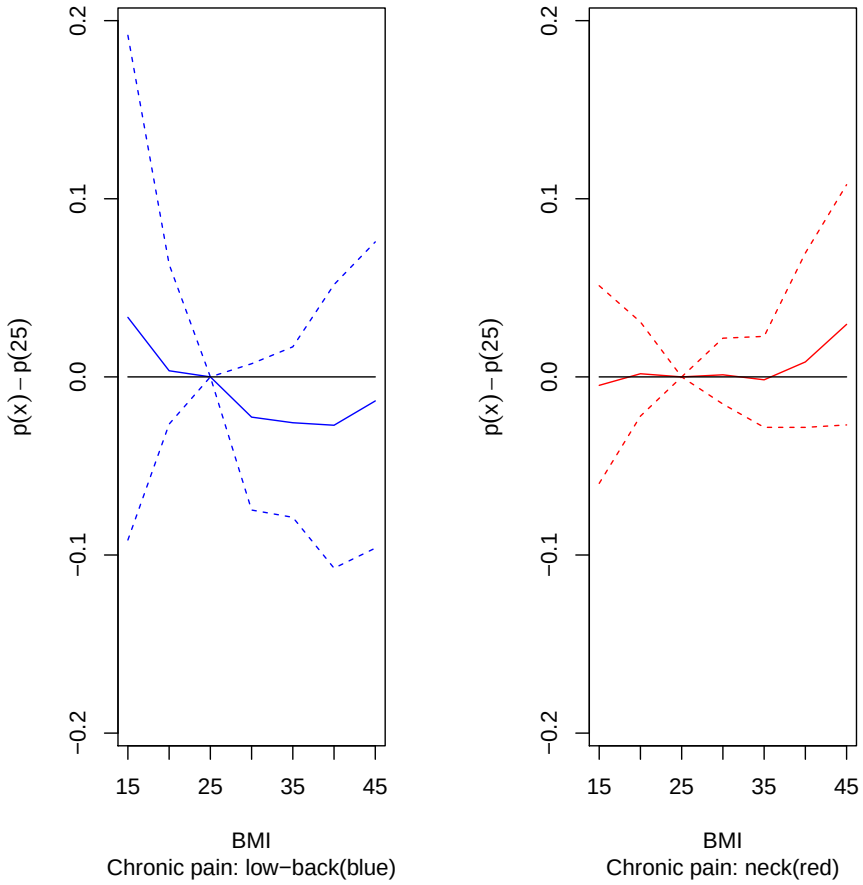


Figure 8: NHANES, BMI and the probability of chronic pain for females only: the left panel for lower-back pain and the right panel for neck pain. The unweighted Friedman's partial dependence relationship between chronic pain and BMI are displayed as ascertained from NHANES data for females only: lower-back (blue) and neck pain (red) are presented with the corresponding 95% credible intervals (dashed lines). The difference in probability of chronic pain from a baseline BMI of 25 (which is the upper limit of normal) is presented, i.e., $p(x) - p(25)$. We want to explore the hypothesis that obesity is a risk factor for chronic lower-back pain (which includes buttock pain in this definition). A corollary to this hypothesis is that obesity is not considered to be a risk factor for chronic neck pain. Although there is a generous amount of uncertainty, it does not appear that the probability of chronic lower-back pain increases with BMI. Conversely, chronic neck pain does appear to be rising, yet again, the intervals are wide. In both cases, these findings are not anticipated.

N.B. for `lbart`, the thinning argument, `keepevery` defaults to 1 while for `gbart` with `type='lbart'`, `keepevery` defaults to 10.

4.4. Multinomial BART for categorical outcomes

Several strategies for analyzing categorical outcomes have been proposed from the Bayesian perspective (Albert and Chib 1993; McCulloch and Rossi 1994; McCulloch, Polson, and Rossi 2000; Imai and Van Dyk 2005; Frühwirth-Schnatter and Frühwirth 2010; Scott 2011) including two BART implementations (Kindo, Wang, and Peña 2016; Murray 2017): our BART implementations differ from these; although, since we are working on the same problem, there are some similarities. Generally, the literature has taken a logit approach. Due to the relative computational efficiency, we prefer probit to logit (although, logit is available as an option). To extend BART to categorical outcomes, we have created two approaches to what we call Multinomial BART. The first approach works well when they are relatively few categories while the second is preferable otherwise.

Multinomial BART and conditional probability: mbart

In the first approach, we fit a novel sequence of binary BART models that bears some resemblance to continuation-ratio logits (Agresti 2003). Let's assume that we have K categories where each are represented by mutually exclusive binary indicators: $y_{i1}, \ldots, y_{iK}$ for subjects indexed by $i = 1, \ldots, N$. We denote the probability of these outcome indicators via conditional probabilities, p_{ij}, where $j = 1, \ldots, K$ as follows.

$$\begin{aligned}
p_{i1} &= \mathrm{P}\left[y_{i1} = 1\right] \\
p_{i2} &= \mathrm{P}\left[y_{i2} = 1 | y_{i1} = 0\right] \\
p_{i3} &= \mathrm{P}\left[y_{i3} = 1 | y_{i1} = y_{i2} = 0\right] \\
&\vdots \\
p_{i,K-1} &= \mathrm{P}\left[y_{i,K-1} = 1 | y_{i1} = \cdots = y_{i,K-2} = 0\right] \\
p_{iK} &= \mathrm{P}\left[y_{i,K-1} = 0 | y_{i1} = \cdots = y_{i,K-2} = 0\right]
\end{aligned}$$

Notice that $p_{iK} = 1 - p_{i,K-1}$ so we can specify the K conditional probabilities via $K - 1$ parameters. Furthermore, these conditional probabilities are, by construction, defined for subsets of subjects: let $S_1 = \{1, \ldots, N\}$ and $S_j = \{i : y_{i1} = \cdots = y_{i,j-1} = 0\}$ where $j = 2, \ldots, K - 1$. Now, the unconditional probability of these outcome indicators, π_{ij}, can be defined in terms of the conditional probablities and their complements, $q_{ij} = 1 - p_{ij}$, for all subjects.

$$\begin{aligned}
\pi_{i1} &= \mathrm{P}\left[y_{i1} = 1\right] = p_{i1} \\
\pi_{i2} &= \mathrm{P}\left[y_{i2} = 1\right] = p_{i2} q_{i1} \\
\pi_{i3} &= \mathrm{P}\left[y_{i3} = 1\right] = p_{i3} q_{i2} q_{i1} \\
&\vdots \\
\pi_{i,K-1} &= \mathrm{P}\left[y_{i,K-1} = 1\right] = p_{i,K-1} q_{i,K-2} \cdots q_{i1} \\
\pi_{iK} &= \mathrm{P}\left[y_{iK} = 1\right] = q_{i,K-1} q_{i,K-2} \cdots q_{i1}
\end{aligned}$$

N.B. the conditional probability construction of π_{ij} ensures that $\sum_{j=1}^{K} \pi_{ij} = 1$.

Our modelling of these conditional probabilities based on a vector of covariates $\boldsymbol{x}_i$ is what we call Multinomial BART:

$$\begin{aligned} y_{ij}|p_{ij} &\sim \mathrm{B}\left(p_{ij}\right) \text{ where } i \in S_j \text{ and } j = 1, \ldots, K-1 \\ p_{ij} &= \Phi(\mu_j + f_j(\boldsymbol{x}_i)) \\ f_j &\overset{\text{prior}}{\sim} \text{BART} \end{aligned}$$

with i indexing subjects, $i = 1, \ldots, N$; and the default value of $\mu_j = \Phi^{-1}\left[\frac{\sum_i y_{ij}}{\sum_i \mathrm{I}(i \in S_j)}\right]$. This formulation yields the Multinomial likelihood: $[\boldsymbol{y}|f_1, \ldots, f_{K-1}] = \prod_{i=1}^{N}\prod_{j=1}^{K}\pi_{ij}^{y_{ij}}$.

This approach is provided by the **BART3** package as the `mbart` function. The input for `mbart` is essentially identical to `gbart`, but the output is slightly different. For example, due to the way the model is estimated, the prediction for `x.train` is not available; therefore, to request it set the argument `x.test=x.train`. By default, probit BART is employed for computational efficiency, but logit BART can be specified with the argument `type='lbart'`. Notation: M for the number of posterior samples, B for the number of threads (generally, $B = 1$ for Windows), N for the number of observations in the training set, and Q for the number of observations in the test set.

```
R> set.seed(99)
R> post <- mbart(x.train, y.train, x.test, ndpost=M)
R> post <- mc.mbart(x.train, y.train, x.test, ndpost=M, mc.cores=B, seed=99)
```

$$\text{Input: } \texttt{x.train} \text{ and } \texttt{x.test}: \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_Q \end{bmatrix} \text{ or } \boldsymbol{x}_i$$

$$\texttt{post}, \text{ of type } \texttt{mbart}$$

$$\texttt{post\$prob.test}: \begin{bmatrix} \hat{\pi}_{111} & \cdots & \hat{\pi}_{1K1} & \cdots & \hat{\pi}_{Q11} & \cdots & \hat{\pi}_{QK1} \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \hat{\pi}_{11M} & \cdots & \hat{\pi}_{1KM} & \cdots & \hat{\pi}_{Q1M} & \cdots & \hat{\pi}_{QKM} \end{bmatrix}$$

The columns of `post$prob.test` represent different covariate settings crossed with the K categories. The `predict` function for objects of type `mbart` is analogous.

Multinomial BART and the logit transformation: mbart2

The second approach is inspired by the logit transformation and is provided by the `mbart2` function which has a similar calling convention to `mbart` described above. Furthermore, as we shall see, the computationally friendly probit is even applicable in this instance. Here, y_i is categorical, i.e., $y_i \in \{1, \ldots, K\}$ (technically, the `mbart2` function does not require the categories to be $1, \ldots, K$; it only requires that there are K distinct categories). Now, we have the following framework motivated by the logit transformation.

$$\begin{aligned} \mathrm{P}\left[y_i = j\right] &= \frac{\exp(\mu_j + f_j(\boldsymbol{x}_i))}{\sum_{j'=1}^{K}\exp(\mu_{j'} + f_{j'}(\boldsymbol{x}_i))} = \pi_{ij} \\ &\text{where } f_j \overset{\text{prior}}{\sim} \text{BART}, \ j = 1, \ldots, K \end{aligned}$$

Suppose for the moment, the centering parameters, μ_j, are defined as in logit BART.

It would appear that this definition has an identifiability issue since $\pi_{ij} = \frac{\exp(\mu_j + f_j(\boldsymbol{x}_i))}{\sum_{j'=1}^{K} \exp(\mu_{j'} + f_{j'}(\boldsymbol{x}_i))} = \frac{\exp(\mu_j + f_j(\boldsymbol{x}_i) + c)}{\sum_{j'=1}^{K} \exp(\mu_{j'} + f_{j'}(\boldsymbol{x}_i) + c)}$. Identifiability could be restored by setting a single BART function to zero, i.e., $f_{j'}(\boldsymbol{x}_i) = 0$. However, this is really unnecessary since π_{ij} is identified regardless.

Computationally, this inference can be performed via a series of binary BARTs. This can be shown by following the work of (Holmes and Held 2006): define $\mathrm{P}[y_i = c] \propto \exp f_c(\boldsymbol{x}_i)$. Consider two cases: $\mathrm{P}[y_i = c]$ and $\mathrm{P}[y_i = j]$ where $j \neq c$. The first case gives us the following in terms of f_c.

$$\begin{aligned}
\mathrm{P}[y_i = c] &= \frac{\exp f_c(\boldsymbol{x}_i)}{\exp f_c(\boldsymbol{x}_i) + \sum_{k \neq c} \exp f_k(\boldsymbol{x}_i)} \\
&= \frac{\exp f_c(\boldsymbol{x}_i)}{\exp f_c(\boldsymbol{x}_i) + \exp S} \text{ where } S = \log \sum_{k \neq c} \exp f_k(\boldsymbol{x}_i) \\
&= \frac{\exp -S}{\exp -S} \frac{\exp f_c(\boldsymbol{x}_i)}{\exp f_c(\boldsymbol{x}_i) + \exp S} \\
&= \frac{\exp(f_c(\boldsymbol{x}_i) - S)}{\exp(f_c(\boldsymbol{x}_i) - S) + 1}
\end{aligned}$$

And the second case, where $j \neq c$, is as follows in terms of f_c.

$$\begin{aligned}
\mathrm{P}[y_i = j] &= \frac{\exp f_j(\boldsymbol{x}_i)}{\exp f_c(\boldsymbol{x}_i) + \sum_{k \neq c} \exp f_k(\boldsymbol{x}_i)} \\
&\propto \frac{1}{\exp f_c(\boldsymbol{x}_i) + \exp S} \\
&\propto \frac{1}{\exp -S} \frac{1}{\exp f_c(\boldsymbol{x}_i) + \exp S} \\
&= \frac{1}{\exp(f_c(\boldsymbol{x}_i) - S) + 1}
\end{aligned}$$

Thus, the conditional inference for f_c is equivalent to a binary indicator $\mathrm{I}(y = c)$. Therefore, `mbart2` computes a full series of all K BART functions for binary indicators.

The `mbart2` function defaults to `type='lbart'`, i.e., Logistic latents are used to compute the f_j's which fits nicely with the logit development of this approach. However, the Logistic latent fitting method can be computationally demanding. Therefore, Normal latents can be specified by `type='pbart'`. This latter setting would appear to contradict the development of this approach; but notice that π_{ij} is still a probability in this case and, in our experience, the results produced are often reasonable.

Multinomial BART example: alligator food preference

We demonstrate the usage of these functions by the American alligator food preference example (Delany, Linda, and Moore 1999; Agresti 2003). In 1985, American alligators were harvested by hunters from August 26 to September 30 in peninsular Florida from lakes Oklawaha (Putnam County), George (Putnam and Volusia counties), Hancock (Polk County)

and Trafford (Collier County). Lake, length and sex were recorded for each alligator. Stomachs from a sample of alligators 1.09:3.89m long were frozen prior to analysis. After thawing, stomach contents were removed and separated and food items were identified and tallied. Volumes were determined by water displacement. The stomach contents of 219 alligators were classified into five categories of primary food preference: bird, fish (the most common primary food choice), invertebrate (snails, insects, crayfish, etc.), reptile (turtles, alligators), bird, and other (amphibians, plants, household pets, stones, and other debris). The length of alligators was dichotomized into small, $\leq$2.3m, vs. large, $>$2.3m. We estimate the probability of each food preference category for the marginal effect of size by resorting to Friedman's partial dependence function (Friedman 2001). We have supplied Figure 9 which summarizes the BART results generated by the example `alligator.R`: you can find this demo with the command `demo("alligator", package="BART")`. The `mbart` function was used since the number of categories is small. The 95% credible intervals are wide, but it appears that large alligators are more likely to rely on a diet of fish while small alligators are more likely to rely on invertebrates. Although the true probabilities are obviously unknown, we compared `mbart` to an analysis by a single hidden-layer/feed-forward Neural Network via the **nnet** R package (Ripley 2007; Venables and Ripley 2013) and the results were essentially identical (see the `demo` for details).

4.5. Converegence diagnostics for binary and categorical outcomes

How do you perform convergence diagnostics for BART? For continuous outcomes, convegence can easily be determined from the trace plots of the the error standard deviation, σ. However, for probit and Multinomial BART with Normal latents, the error variance is fixed at 1 so this is not an option. Similarly, for logit BART, σ_i, are auxiliary latent variables not suitable for convergence diagnostics. Therefore, we adapt traditional MCMC diagnostic approaches to BART. We perform graphical checks via auto-correlation, trace plots and an approach due to Geweke (1992).

Geweke diagnostics are based on earlier work which characterizes MCMC as a time series (Hastings 1970). Once this transition is made, auto-regressive, moving-average (ARMA) process theory is employed (Silverman 1986). Generally, we define our Bayesian estimator as $\hat{\theta}_M = M^{-1}\sum_{m=1}^{M}\theta_m$. We represent the asymptotic variance of the estimator by $\sigma^2_{\hat{\theta}} = \lim_{M\to\infty} \mathrm{V}\left[\hat{\theta}_M\right]$. If we suppose that θ_m is an ARMA(p,q) process, then the spectral density of the estimator is defined as $\gamma(w) = (2\pi)^{-1}\sum_{m=-\infty}^{\infty}\mathrm{V}\left[\theta_0,\theta_m\right]\mathrm{e}^{imw}$ where $\mathrm{e}^{itw} = \cos(tw) + \mathrm{i}\sin(tw)$. This leads us to an estimator of the asymptotic variance which is $\hat{\sigma}^2_{\hat{\theta}} = \hat{\gamma}^2(0)$. We divide our chain into two segments, A and B, as follows: $m \in A = \{1,\ldots,M_A\}$ where $M_A = aM$; and $m \in B = \{M - M_B + 1,\ldots,M\}$ where $M_B = bM$. Note that $a + b < 1$. Geweke

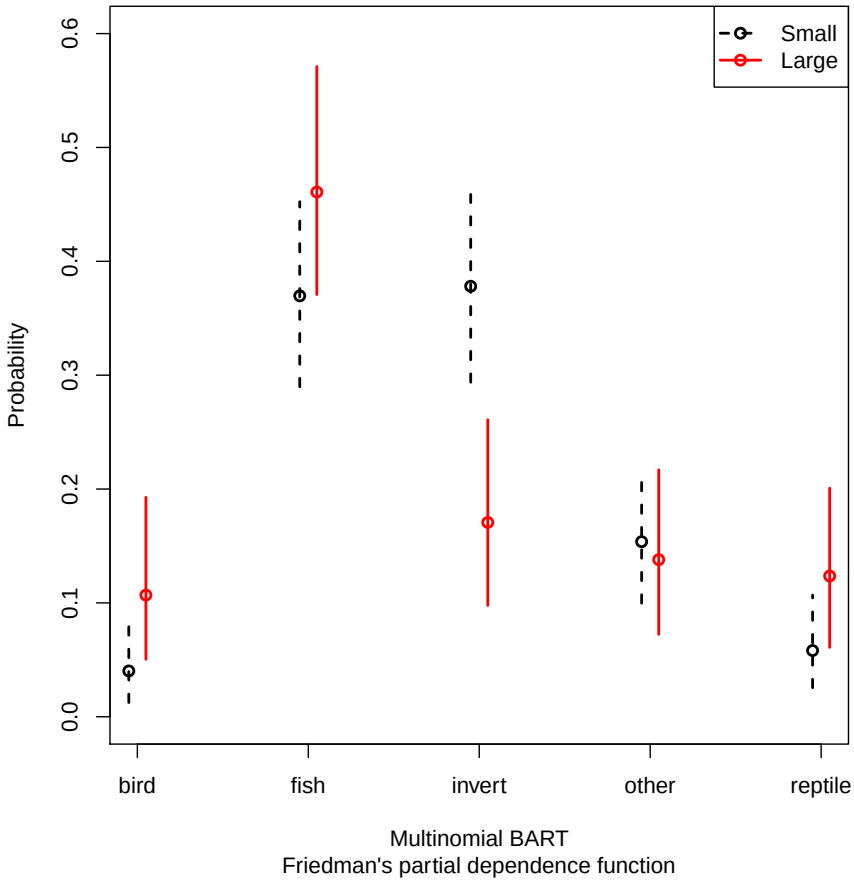


Figure 9: In 1985, American alligators were harvested by hunters in peninsular Florida from four lakes. Lake, length and sex were recorded for each alligator. The stomach contents of 219 alligators were classified into five categories based on the primary food preference: bird, fish, invertebrate, reptile and other. The length of alligators was dichotomized into small, $\leq$2.3m, vs. large, >2.3m. We estimate the probability of each food preference category for the marginal effect of size by resorting to Friedman's partial dependence function (Friedman 2001). The 95% credible intervals are wide, but it appears that large alligators are more likely to rely on a diet of fish while small alligators are more likely to rely on invertebrates.

suggests $a = 0.1,\ b = 0.5$ and recommends the following Normal test for convergence.

$$\hat{\theta}_A = M_A^{-1} \sum_{m \in A} \theta_m \qquad \hat{\theta}_B = M_B^{-1} \sum_{m \in B} \theta_m$$

$$\hat{\sigma}^2_{\hat{\theta}_A} = \hat{\gamma}^2_{m \in A}(0) \qquad \hat{\sigma}^2_{\hat{\theta}_B} = \hat{\gamma}^2_{m \in B}(0)$$

$$Z_{AB} = \frac{\sqrt{M}(\hat{\theta}_A - \hat{\theta}_B)}{\sqrt{a^{-1}\hat{\sigma}^2_{\hat{\theta}_A} + b^{-1}\hat{\sigma}^2_{\hat{\theta}_B}}} \sim \mathrm{N}\,(0,\ 1)$$

In our **BART3** package, we supply R functions adapted from the **coda** R package (Plummer, Best, Cowles, and Vines 2006) to perform Geweke diagnostics: `spectrum0ar` and `gewekediag`. But, how do we apply Geweke's diagnostic to BART? We can check convergence for any estimator of the form $\theta = h(f(\boldsymbol{x}))$, but often setting h to the identify function will suffice, i.e., $\theta = f(\boldsymbol{x})$. However, BART being a Bayesian nonparametric technique means that we have many potential estimators to check, i.e., essentially one estimator for every possible choice of $\boldsymbol{x}$.

We have supplied Figures 10, 11 and 12 generated by the example `geweke.pbart2.R`: `demo("geweke.pbart2", package="BART")`. The data are simulated by Friedman's five-dimensional test function (Friedman 1991) where 50 covariates are generated as $x_{ij} \sim \mathrm{U}\,(0,1)$ but only the first 5 covariates have an impact on the outcome at sample sizes $N = 200, 1000, 5000$.

$$\begin{aligned} f(\boldsymbol{x}_i) &= -1.5 + \sin(\pi x_{i1} x_{i2}) + 2(x_{i3} - 0.5)^2 + x_{i4} + 0.5 x_{i5} \\ z_i &\sim \mathrm{N}\,(f(\boldsymbol{x}_i),\ 1) \\ y_i &= \mathrm{I}\,(z_i > 0) \end{aligned}$$

The convergence for each of these data sets is graphically displayed in Figures 10, 11 and 12 where each figure is broken into four quadrants. In the upper left quadrant, we have plotted Friedman's partial dependence function for $f(x_{i4})$ vs. x_{i4} for 10 values of x_{i4}. This is a check that can't be performed for real data, but it is informative in this case. Notice that $f(x_{i4})$ vs. x_{i4} is directly proportional in each figure as expected. In the upper right quadrant, we plot the auto-correlations of $f(\boldsymbol{x}_i)$ for 10 randomly selected $\boldsymbol{x}_i$ where i indexes subjects. Notice that there is very little auto-correlation for $N = 200, 1000$, but a more notable amount for $N = 5000$. In the lower left quadrant, we display the corresponding trace plots for these same settings. The traces demonstrate that samples of $f(\boldsymbol{x}_i)$ appear to adequately traverse the sample space for $N = 200, 1000$, but less notably for $N = 5000$. In the lower right quadrant, we plot the Geweke Z_{AB} statistics for each subject i. Notice that for $N = 200$, the Z_{AB} exceed the 95% limits only a handful of times. Although, there are 10 times more comparisons, $N = 1000$ has seemingly more than 10 times as many values exceeding the 95% limits. And, for $N = 5000$, there are dramatically more values exceeding the 95% limits. Based on these figures, we conclude that the chains have converged for $N = 200$; for $N = 1000$, convergence is questionable; and, for $N = 5000$, convergence has not been attained. We would suggest that more thinning be employed for $N = 1000, 5000$ via the `keepevery` argument to `pbart`; perhaps, `keepevery=50` for $N = 1000$ and `keepevery=250` for $N = 5000$.

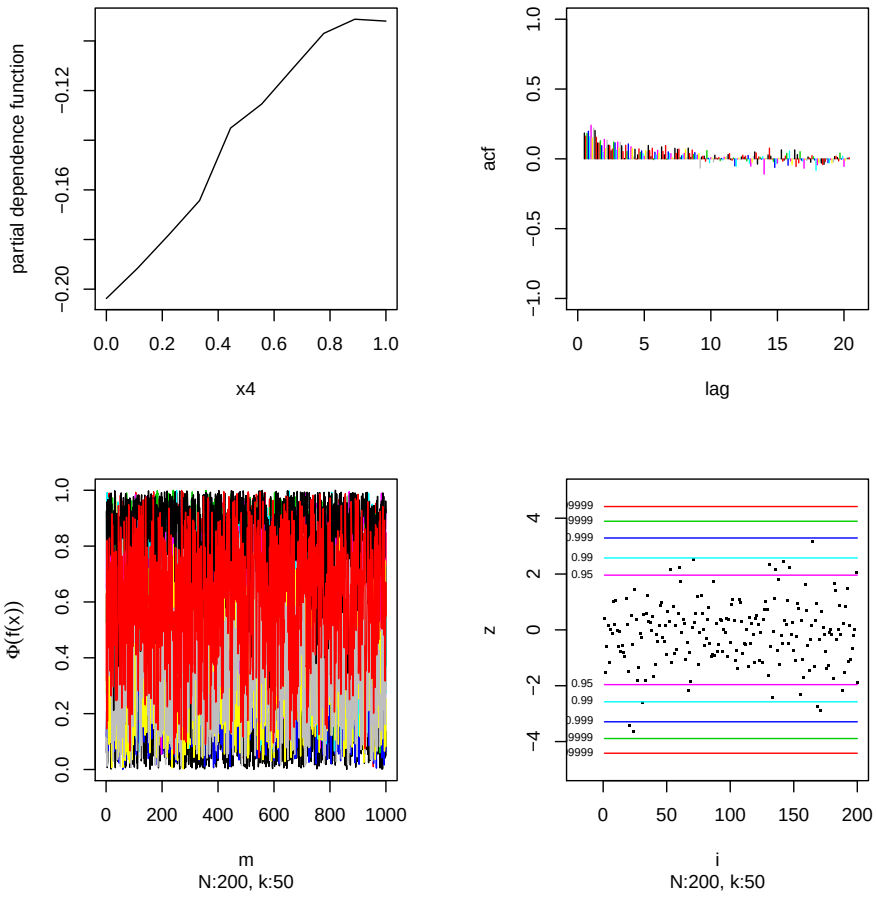


Figure 10: Geweke convergence diagnostics for probit BART: $N = 200$. In the upper left quadrant, we have plotted Friedman's partial dependence function for $f(x_{i4})$ vs. x_{i4} for 10 values of x_{i4}. This is a check that can't be performed for real data, but it is informative in this case. Notice that $f(x_{i4})$ vs. x_{i4} is mainly directly proportional expected. In the upper right quadrant, we plot the auto-correlations of $f(\boldsymbol{x}_i)$ for 10 randomly selected $\boldsymbol{x}_i$ where i indexes subjects. Notice that there is very little auto-correlation. In the lower left quadrant, we display the corresponding trace plots for these same settings. The traces demonstrate that samples of $f(\boldsymbol{x}_i)$ appear to adequately traverse the sample space. In the lower right quadrant, we plot the Geweke Z_{AB} statistics for each subject i. Notice that the Z_{AB} exceed the 95% limits only a handful of times. Based on this figure, we conclude that the chains have converged.

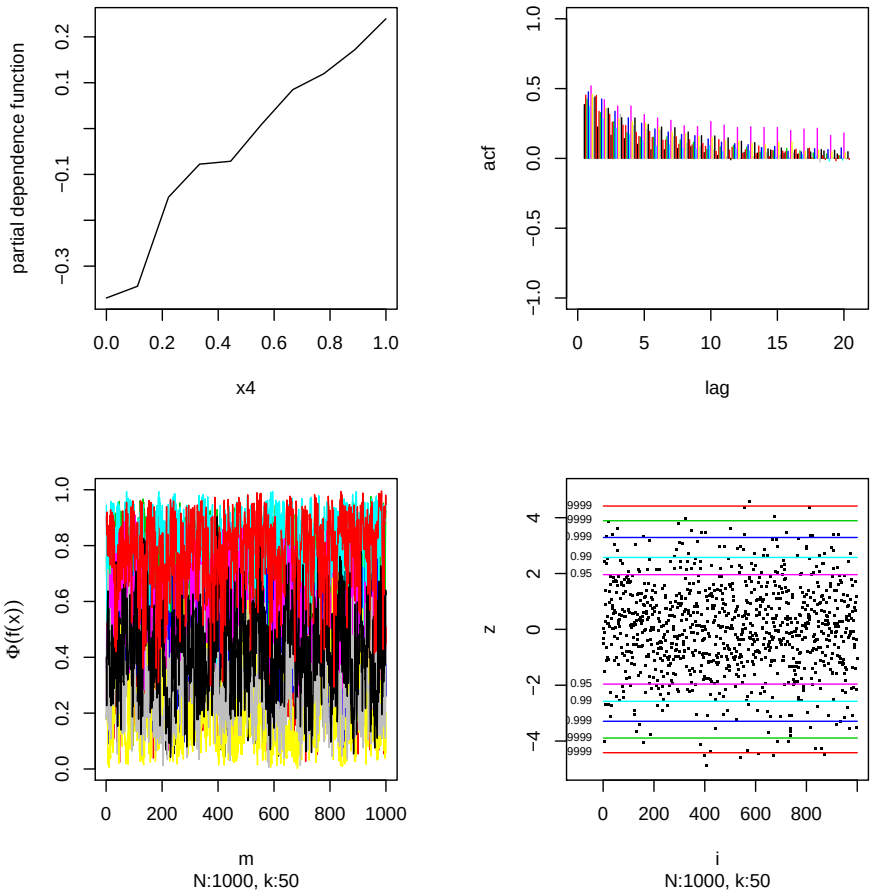


Figure 11: Geweke convergence diagnostics for probit BART: $N = 1000$. In the upper left quadrant, we have plotted Friedman's partial dependence function for $f(x_{i4})$ vs. x_{i4} for 10 values of x_{i4}. This is a check that can't be performed for real data, but it is informative in this case. Notice that $f(x_{i4})$ vs. x_{i4} is directly proportional as expected. In the upper right quadrant, we plot the auto-correlations of $f(\boldsymbol{x}_i)$ for 10 randomly selected $\boldsymbol{x}_i$ where i indexes subjects. Notice that there is very little auto-correlation. In the lower left quadrant, we display the corresponding trace plots for these same settings. The traces demonstrate that samples of $f(\boldsymbol{x}_i)$ appear to adequately traverse the sample space. In the lower right quadrant, we plot the Geweke Z_{AB} statistics for each subject i. Notice that there appear to be a considerable number exceeding the 95% limits. Based on this figure, we conclude that convergence is questionable. We would suggest that more thinning be employed via the `keepevery` argument to `pbart`; perhaps, `keepevery=50`.

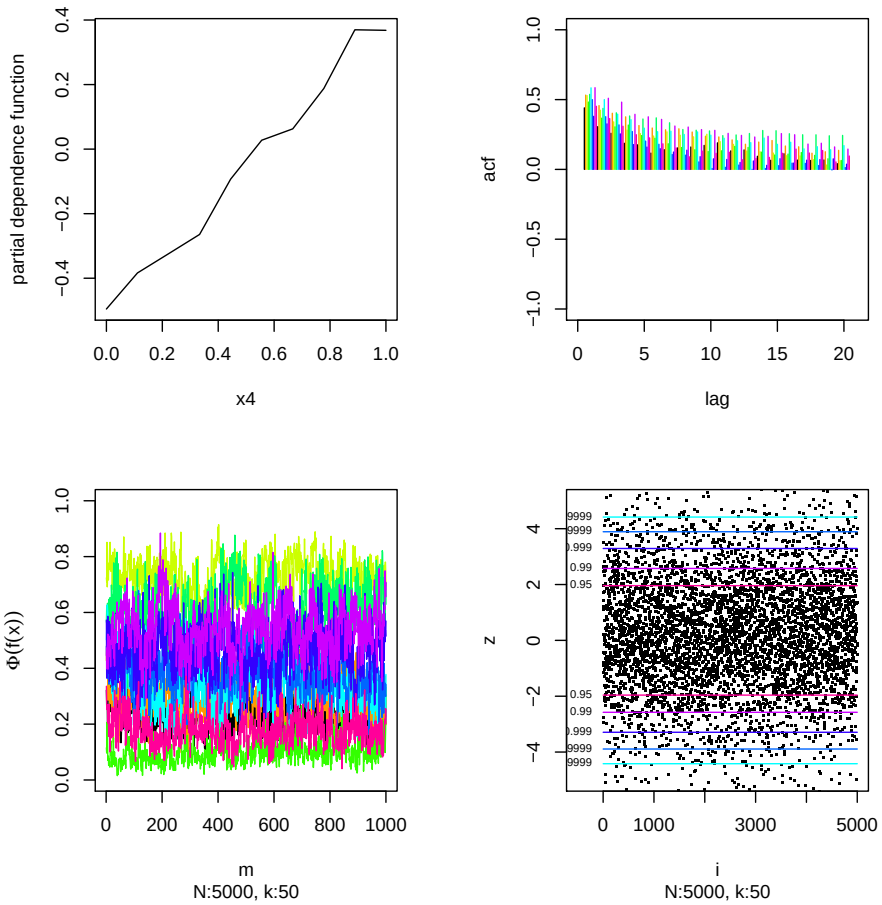


Figure 12: Geweke convergence diagnostics for probit BART: $N = 5000$. In the upper left quadrant, we have plotted Friedman's partial dependence function for $f(x_{i4})$ vs. x_{i4} for 10 values of x_{i4}. This is a check that can't be performed for real data, but it is informative in this case. Notice that $f(x_{i4})$ vs. x_{i4} is directly proportional as expected. In the upper right quadrant, we plot the auto-correlations of $f(\boldsymbol{x}_i)$ for 10 randomly selected $\boldsymbol{x}_i$ where i indexes subjects. Notice that there is some auto-correlation. In the lower left quadrant, we display the corresponding trace plots for these same settings. The traces demonstrate that samples of $f(\boldsymbol{x}_i)$ appear to traverse the sample space, but there are some slower oscillations. In the lower right quadrant, we plot the Geweke Z_{AB} statistics for each subject i. Notice that there appear to be far too many exceeding the 95% limits. Based on these figures, we conclude that convergence has not been attained. We would suggest that more thinning be employed via the `keepevery` argument to `pbart`; perhaps, `keepevery=250`.

4.6. BART and variable selection

Bayesian variable selection techniques applicable to BART have been studied by Chipman *et al.* (2010); Chipman, George, and McCulloch (2013); Bleich, Kapelner, George, and Jensen (2014); Hahn and Carvalho (2015); McCulloch, Carvalho, and Hahn (2015); Linero (2018). The **BART3** package supports the sparse prior of Linero (2018) by specifying `sparse=TRUE` (the default is `sparse=FALSE`). Let's represent the variable selection probabilities by s_j where $j = 1, \ldots, P$. Now, replace the uniform variable selection prior in BART with a Dirichlet prior. Also, place a Beta prior on the θ parameter.

$$[s_1, \ldots, s_P] \,|\theta \overset{\text{prior}}{\sim} \text{Dirichlet}\,(\theta/P, \ldots, \theta/P)$$
$$\frac{\theta}{\theta + \rho} \overset{\text{prior}}{\sim} \text{Beta}\,(a,\ b)$$

Typical settings are $b = 1$ and $\rho = P$ (the defaults) which you can over-ride with the `b` and `rho` arguments respectively. The value $a = 0.5$ (the default) is a sparse setting whereas an alternative setting $a = 1$ is not sparse; you can specify this parameter with argument `a`. If additional sparsity is desired, then you can set the argument `rho` to a value smaller than P: for more details, see Appendix B. Furthermore, Linero discusses two assumptions: Assumption 2.1 and Assumption 2.2 (see Linero (2018) for more details). Basically, Assumption 2.2 (2.1) is more (less) friendly to binary/ordinal covariates and is (is not) the default corresponding to `augment=FALSE` (`augment=TRUE`).

Let's return to the simulated probit BART example explored above which is in the **BART3** package: `demo("sparse.pbart", package="BART")`. For sample sizes of $N = 200, 1000, 5000$, there are $P = 100$ covariates, but only the first 5 are active. In Figure 13, the 5 (95) active (inactive) covariates are red (black) and circles (dots) are $>$ ($\leq$) P^{-1} which is chance association represented by a black line. For $N = 200$, all five active variables are identified, but notice that there are 20 false positives. For $N = 1000$, all five active covariates are identified, but notice that there are still 14 false positives. For $N = 5000$, all five active covariates are identified and notice that there is only one false positive.

We are often interested in the inter-relationship between covariates within our model. We can assess these relationships by inspecting the binary trees. For example, we can ascertain how often x_1 is chosen as a branch decision rule leading to a branch decision rule with x_2 further up the tree or vice versa. In this case, we call x_1 and x_2 a concordant pair and we denote by $x_1 \leftrightarrow x_2$ which is a symmetric relationship, i.e., $x_1 \leftrightarrow x_2$ implies $x_2 \leftrightarrow x_1$. If $\mathcal{B}_h$ is the number of branches in tree $\mathcal{T}_h$, then the concordant pair probability is: $\kappa_{ij} = \text{P}\left[x_i \leftrightarrow x_j \in \mathcal{T}_h | \mathcal{B}_h > 1\right]$ for $i = 1, \ldots, P-1$ and $j = i+1, \ldots, P$. See an example of calculating these probabilities in `demo("trees.pbart", package="BART")`.

5. Time-to-event outcomes with BART

The **BART3** package supports time-to-event outcomes including survival analysis, competing risks and recurrent events.

5.1. Survival analysis with BART

Survival analysis with BART is provided by the `surv.bart` function for serial computation

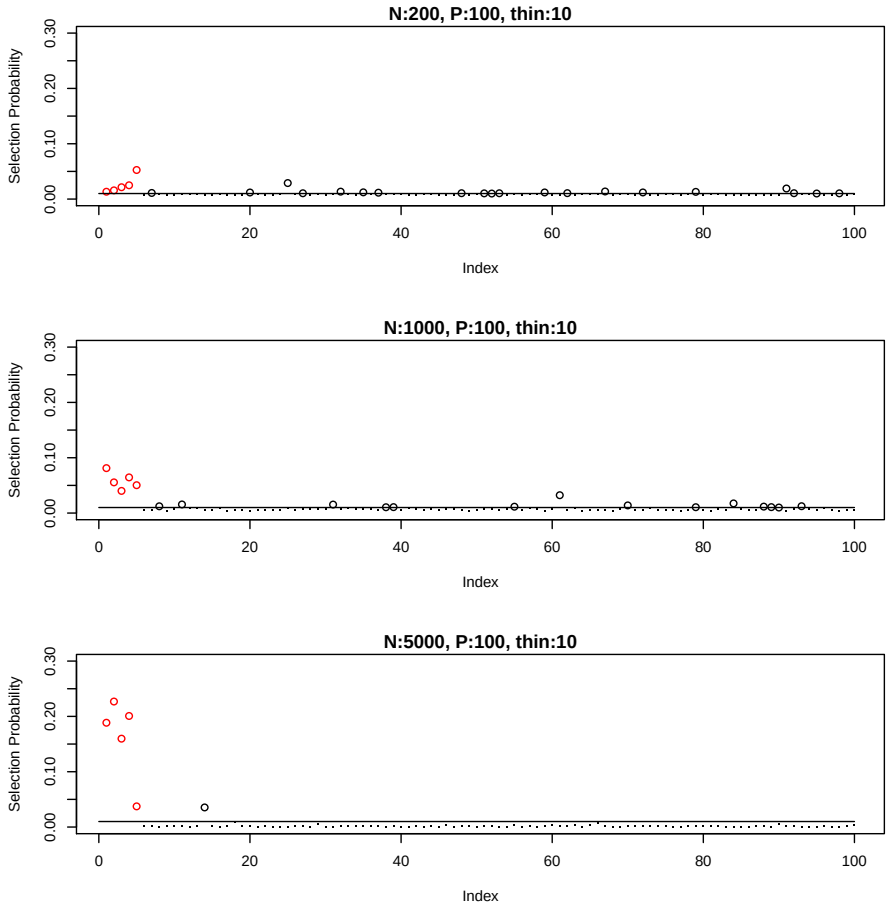


Figure 13: Probit BART and variable selection example. For sample sizes of $N = 200, 1000, 5000$, there are $P = 100$ covariates, but only the first 5 are active. The 5 (95) active (inactive) covariates are red (black) and circles (dots) are $>$ ($\leq$) P^{-1} which is chance association represented by a black line. For $N = 200$, all five active variables are identified, but notice that there are 20 false positives. For $N = 1000$, all five active covariates are identified, but notice that there are still 14 false positives. For $N = 5000$, all five active covariates are identified and notice that there is only one false positive.

and `mc.surv.bart` for parallel computation. Survival analysis has been studied by many, however, most take a proportional hazards approach (Cox 1972; Kalbfleisch and Prentice 1980; Klein and Moeschberger 2006). The complete details of our approach can be found in Sparapani, Logan, McCulloch, and Laud (2016) and a brief introduction follows. We take an approach that is tantamount to discrete-time survival analysis (Thompson Jr. 1977; Arjas and Haara 1987; Fahrmeir 2014). Relying on the capabilities of BART, we do not stipulate a linear relationship with the covariates nor proportional hazards.

The data is $(s_i, \delta_i, \boldsymbol{x}_i)$ where i indexes subjects, $i = 1, \ldots, N$; s_i is the time of an absorbing event, $\delta_i = 1$, or right censoring, $\delta_i = 0$; and $\boldsymbol{x}_i$ is a vector of covariates (which can be time-dependent, but, for simplicity, we assume that they are known at time zero). We construct a grid of the ordered distinct event times, $0 = t_{(0)} < \cdots < t_{(K)} < \infty$, and we consider the following time intervals: $(0, t_{(1)}], (t_{(1)}, t_{(2)}], \ldots (t_{(K-1)}, t_{(K)}]$.

Now, consider event indicators y_{ij} for each subject i at each distinct time $t_{(j)}$ up to and including the subject's last observation time $t_i = t_{(n_i)}$ with $n_i = \arg\max_j \left[t_{(j)} \leq t_i\right]$. This means $y_{ij} = 0$ if $j < n_i$ and $y_{in_i} = \delta_i$. Denote the probability of an event at time $t_{(j)}$, conditional on no previous event, by p_{ij}. Now, our model for y_{ij} is a nonparametric probit regression of y_{ij} on the time $t_{(j)}$ and the covariates $\boldsymbol{x}_i$.

So the model is

$$\begin{aligned} y_{ij} &= \delta_i \mathrm{I}\left(s_i = t_{(j)}\right), \ j = 1, \ldots, n_i \\ y_{ij}|p_{ij} &\sim \mathrm{B}\left(p_{ij}\right) \\ p_{ij} &= \Phi(\mu_{ij}), \ \mu_{ij} = \mu_0 + f(t_{(j)}, \boldsymbol{x}_i) \\ f &\overset{\text{prior}}{\sim} \text{BART} \end{aligned}$$

where i indexing subjects, $i = 1, \ldots, N$; and $\Phi(.)$ is the standard Normal cumulative distribution function. This formulation creates the likelihood of $[\boldsymbol{y}|f] = \prod_{i=1}^{N} \prod_{j=1}^{n_i} p_{ij}^{y_{ij}} (1 - p_{ij})^{1-y_{ij}}$.

If the event indicators, y_{ij}, have already been computed, then you can specify them with the `y.train` argument. However, it is likely that the indicators would need to be constructed, so for convenience, you can specify (s_i, δ_i) by the arguments `times` and `delta` respectively. In either case, the default value of μ_0 is $\Phi^{-1}(\bar{y})$ (which you can over-ride with the `offset` argument). For computational efficiency, probit (Albert and Chib 1993) is the default, but logit (Holmes and Held 2006; Gramacy and Polson 2012) can be specified as an option via `type="lbart"`.

Based on the posterior samples, we construct quantities of interest with BART for survival analysis. In discrete-time survival analysis, the instantaneous hazard from continuous-time survival is essentially replaced with the probability of an event in an interval, i.e., $p(t_{(j)}, \boldsymbol{x}) = \Phi(\mu_0 + f(t_{(j)}, \boldsymbol{x}))$. Now, the survival function is constructed as follows: $S(t_{(j)}|\boldsymbol{x}) = Pr(T > t_{(j)}|\boldsymbol{x}) = \prod_{l=1}^{j}(1 - p(t_{(l)}, \boldsymbol{x}))$.

Survival data pairs (s, δ) are converted to indicators by the helper function `surv.pre.bart` which is called automatically by `surv.bart` if `y.train` is not provided. `surv.pre.bart` returns a list which contains `y.train` for the indicators; `tx.train` for the covariates corresponding to `y.train` for training $f(t, \boldsymbol{x})$ (which includes time in the first column, and the rest of the covariates afterward, if any, i.e., rows of $[t, \boldsymbol{x}]$, hence the name `tx.train` to distinguish it from the original `x.train`); `tx.test` for the covariates to predict $f(t, \boldsymbol{x})$ rather than to train; `times` which is the grid of ordered distinct time points; and `K` which is the length of

`times`. Here is a very simple example of a data set with three observations and no covariates re-formatted for display (no covariates is an interesting special case but we will discuss the more common case with covariates further below).

```
R> times <- c(2.5, 1.5, 3.0)
R> delta <- c(  1,   1,   0)
R> surv.pre.bart(times=times, delta=delta)

$y.train  $tx.train  $tx.test  $times    $K
[1]              t         t  [1]       [1] 3
    0     [1,] 1.5   [1,] 1.5    1.5
    1     [2,] 2.5   [2,] 2.5    2.5
    1     [3,] 1.5   [3,] 3.0    3.0
    0     [4,] 1.5
    0     [5,] 2.5
    0     [6,] 3.0
```

Here is a diagram of the input and output for the `surv.pre.bart` function. `pre` is a list that is generated to contain the matrix `pre$tx.train` and the vector `pre$y.train`.

```
R> pre <- surv.pre.bart(times, delta, x.train, x.test=x.train)
```

$$\begin{matrix} \texttt{tx.train} \\ \left[\begin{array}{cc} t_{(1)} & \boldsymbol{x}_1 \\ \vdots & \vdots \\ t_{(n_1)} & \boldsymbol{x}_1 \\ \vdots & \vdots \\ t_{(1)} & \boldsymbol{x}_N \\ \vdots & \vdots \\ t_{(n_N)} & \boldsymbol{x}_N \end{array}\right] \end{matrix} \quad \begin{matrix} \texttt{y.train} \\ \left[\begin{array}{c} y_{11}=0 \\ \vdots \\ y_{1n_1}=\delta_1 \\ \vdots \\ y_{N1}=0 \\ \vdots \\ y_{Nn_N}=\delta_N \end{array}\right] \end{matrix}$$

For `pre$tx.test`, n_i is replaced by K which is very helpful so that each subject contributes an equal number of settings for programmatic convenience and noninformative estimation, i.e., if high-risk subjects with earlier events did not appear beyond their event, then estimates of survival for latter times would be biased upward. For other outcomes besides time-to-event, we provide two matrices of covariates, `x.train` and `x.test`, where `x.train` is for training and `x.test` is for validation. However, due to the variable n_i for time-to-event outcomes, we generally provide two arguments as follows: `x.train, x.test=x.train` where the former matrix will be expanded by `surv.pre.bart` to $\sum_{i=1}^{N} n_i$ rows for training $f(t, \boldsymbol{x})$ while the latter matrix will be expanded to $N \times K$ rows for $f(t, \boldsymbol{x})$ estimation only. If you still need to perform validation, then you can make a separate call to the `predict` function.

N.B. the argument `ndpost=M` is the length of the chain to be returned and the argument `keepevery` is used for thinning, i.e., return `M` observations where `keepevery` are culled in between each returned value. For BART with time-to-event outcomes which is based on `gbart`,

the default is `keepevery=10` since the grid of time points creates data set observations of order $N \times K$ which have a tendency towards higher auto-correlation, therefore, making thinning more necessary. To avoid unnecessarily enlarged data sets, it is often prudent to coarsen the time axis appropriately. Although this might seem drastic, times are often collected orders of magnitude more precisely than necessary for the problem under study. For example, cancer registries often collect survival times in days while time in months or quarters would suffice for many typical applications. You can coarsen automatically by supplying the optional `K` argument to coarsen the times to a grid of time quantiles: `1/K`, `2/K`, ..., `K/K` (not to be confused with the `k` argument which is a prior parameter for the distribution of the leaf terminal values).

Here is a diagram of the input and output for the `surv.bart` function for serial computation and `mc.surv.bart` for parallel computation.

Serial call

```
R> set.seed(99)
R> post=surv.bart(x.train, times=times, delta=delta, x.test=x.train,
+    ndpost=M)
```

Parallel call

```
R> post=mc.surv.bart(x.train, times=times, delta=delta, x.test=x.train,
+    ndpost=M, mc.cores=B, seed=99)
```

Input vector `times` with `K` distinct values and

`x.train`: $\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_N \end{bmatrix}$ with rows of $\boldsymbol{x}_i$

Output `post`, of type `survbart`, is essentially a list including the matrix

`post$surv.test`: $\hat{S}_m(t_{(j)}, \boldsymbol{x}_i)$

$$\begin{bmatrix} \hat{S}_1(t_{(1)},\boldsymbol{x}_1) & \ldots & \hat{S}_1(t_{(K)},\boldsymbol{x}_1) & \ldots & \hat{S}_1(t_{(1)},\boldsymbol{x}_N) & \ldots & \hat{S}_1(t_{(K)},\boldsymbol{x}_N) \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \hat{S}_M(t_{(1)},\boldsymbol{x}_1) & \ldots & \hat{S}_M(t_{(K)},\boldsymbol{x}_1) & \ldots & \hat{S}_M(t_{(1)},\boldsymbol{x}_N) & \ldots & \hat{S}_M(t_{(K)},\boldsymbol{x}_N) \end{bmatrix}$$

Here is a diagram of the input and output for the `predict.survbart` function.

```
R> pred <- predict(post, pre$tx.test, mc.cores=B)
```

$$\text{Input: } \texttt{x.test} \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_Q \end{bmatrix} \text{ with rows of } \boldsymbol{x}_i$$

$$\text{Output: } \texttt{pred} \text{ of type } \texttt{survbart} \text{ with } \texttt{pred\$surv.test}\text{: } \hat{S}_m(t_{(j)}, \boldsymbol{x}_i)$$

$$\begin{bmatrix} \hat{S}_1(t_{(1)},\boldsymbol{x}_1) & \dots & \hat{S}_1(t_{(K)},\boldsymbol{x}_1) & \dots & \hat{S}_1(t_{(1)},\boldsymbol{x}_Q) & \dots & \hat{S}_1(t_{(K)},\boldsymbol{x}_Q) \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \hat{S}_M(t_{(1)},\boldsymbol{x}_1) & \dots & \hat{S}_M(t_{(K)},\boldsymbol{x}_1) & \dots & \hat{S}_M(t_{(1)},\boldsymbol{x}_Q) & \dots & \hat{S}_M(t_{(K)},\boldsymbol{x}_Q) \end{bmatrix}$$

For an overview of Friedman's partial dependence function (including the notation adopted in this article and its meaning), please see Section 3.8 which discusses continuous outcomes. For survival analysis, we use Friedman's partial dependence function (Friedman 2001) with BART to summarize the marginal effect due to a subset of the covariates settings which, naturally, includes time, $(t_{(j)}, \boldsymbol{x}_{hS})$. For survival analysis, the f function is often not directly of interest; rather, the survival function is more readily interpretable: $S(t_{(j)}, \boldsymbol{x}_{hS}) = N^{-1} \sum_{i=1}^{N} S(t_{(j)}, \boldsymbol{x}_{hS}, \boldsymbol{x}_{iC})$.

Survival analysis with BART example: advanced lung cancer

Here we present an example that is available in the **BART3** package: `demo("lung.surv.bart", package="BART")`. The North Central Cancer Treatment Group surveyed 228 advanced lung cancer patients (Loprinzi, Laurie, Wieand, Krook, Novotny, Kugler, Bartel, Law, Bateman, and Klatt 1994). This data can be found in the `lung` data set. The study focused on prognostic variables. Patient responses were paired with a few clinical variables. We control for age, gender and Karnofsky performance score as rated by their physician. We compare the survival for males and females with Friedman's partial dependence function; see Figure 14. We also analyze this data set with logit BART and the results are quite similar (not shown): `demo("lung.surv.lbart", package="BART")`. Furthermore, we perform convergence diagnostics on the chain:
`demo("geweke.lung.surv.bart", package="BART")`.

5.2. Survival analysis and the concordance probability

The concordance probability (Gönen and Heller 2005) is a measure of the discriminatory ability of survival analysis analogous to the area under the receiver operating characteristic curve for binary outcomes. Suppose that we have two event times, t_1 and t_2, (let's say each based on a different subject profile), then the concordance probability is defined as $\kappa_{t_1,t_2} = \mathrm{P}[t_1 < t_2]$. A simple analytic example with the Exponential distribution is as follows.

$$t_i|\lambda_i \overset{\text{ind}}{\sim} \mathrm{Exp}(\lambda_i) \text{ where } i \in \{1,2\}$$

$$\mathrm{P}[t_1 < t_2|\lambda_1,\lambda_2] = \int_0^\infty \int_0^{t_2} \lambda_2 \mathrm{e}^{-\lambda_2 t_2} \lambda_1 \mathrm{e}^{-\lambda_1 t_1} \,\mathrm{d}t_1 \,\mathrm{d}t_2 = \frac{\lambda_1}{\lambda_1+\lambda_2}$$

$$1 - \mathrm{P}[t_1 > t_2|\lambda_1,\lambda_2] = 1 - \frac{\lambda_2}{\lambda_1+\lambda_2} = \frac{\lambda_1}{\lambda_1+\lambda_2}$$

$$= \mathrm{P}[t_1 < t_2|\lambda_1,\lambda_2]$$

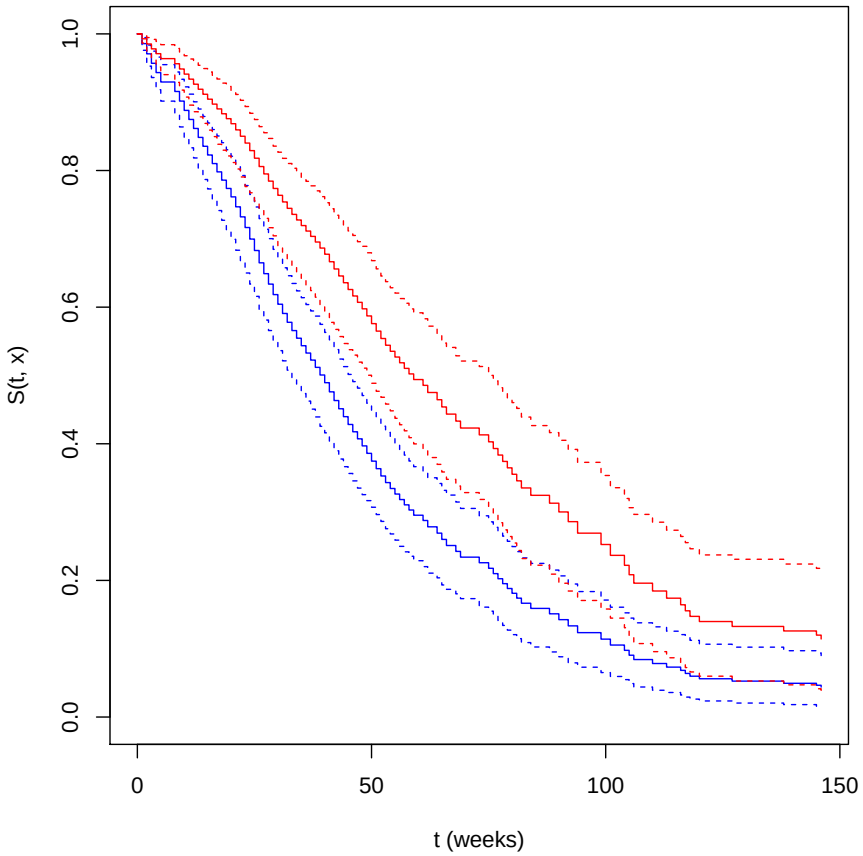


Figure 14: Advanced lung cancer example: Friedman's partial dependence function with 95% credible intervals: males (blue) vs. females (red). A cohort of advanced lung cancer patients was recruited from the North Central Cancer Treatment Group. For survival time, these patients were followed for nearly 3 years or until lost to follow-up.

Notice that the concordance is symmetric with respect to t_1 and t_2.

We can make a similar calculation based on our BART survival analysis model. Suppose that we have two event times, s_1 and s_2, which are conditionally independent, i.e., $s_1|(f, \boldsymbol{x}_1) \perp s_2|(f, \boldsymbol{x}_2)$. First, we calculate $\mathrm{P}\left[s_1 < s_2 | f, \boldsymbol{x}_1, \boldsymbol{x}_2\right]$ (from here on, we suppress f and $\boldsymbol{x}_i$ for notational convenience).

$$\begin{aligned}
\mathrm{P}\left[s_1 < s_2\right] =& \mathrm{P}\left[s_1 = t_{(1)}, s_2 > t_{(1)}\right] + \\
& \mathrm{P}\left[s_1 = t_{(2)}, s_2 > t_{(2)} | s_1 > t_{(1)}, s_2 > t_{(1)}\right] \mathrm{P}\left[s_1 > t_{(1)}, s_2 > t_{(1)}\right] + \ldots \\
=& \sum_{j=1}^{K} \mathrm{P}\left[s_1 = t_{(j)}, s_2 > t_{(j)} | s_1 > t_{(j-1)}, s_2 > t_{(j-1)}\right] \mathrm{P}\left[s_1 > t_{(j-1)}, s_2 > t_{(j-1)}\right] \\
=& \sum_{j=1}^{K} p_{1j} q_{2j} S_1(t_{(j-1)}) S_2(t_{(j-1)})
\end{aligned}$$

Now, we calculate the mirror image relationship.

$$\begin{aligned}
1 - \mathrm{P}\left[s_1 > s_2\right] =& 1 - \sum_{j=1}^{K} q_{1j} p_{2j} S_1(t_{(j-1)}) S_2(t_{(j-1)}) \\
=& 1 - \sum_{j=1}^{K} (1 - p_{1j})(1 - q_{2j}) S_1(t_{(j-1)}) S_2(t_{(j-1)}) \\
=& 1 - \sum_{j=1}^{K} (1 - p_{1j} - q_{2j} + p_{1j} q_{2j}) S_1(t_{(j-1)}) S_2(t_{(j-1)}) \\
=& 1 - \sum_{j=1}^{K} p_{1j} q_{2j} S_1(t_{(j-1)}) S_2(t_{(j-1)}) - \sum_{j=1}^{K} (q_{1j} - q_{2j}) S_1(t_{(j-1)}) S_2(t_{(j-1)})
\end{aligned}$$

However, note that these probabilities are not symmetric in this form. Yet, we can arrive at symmetry as follows.

$$\begin{aligned}
\kappa_{s_1,s_2} &= 0.5 \left(\mathrm{P}\left[s_1 < s_2\right] + 1 - \mathrm{P}\left[s_1 > s_2\right]\right) \\
&= 0.5 \left[1 - \sum_{j=1}^{K} (q_{1j} - q_{2j}) S_1(t_{(j-1)}) S_2(t_{(j-1)})\right]
\end{aligned}$$

See the concordance probability example at `demo("concord.surv.bart", package="BART")`.

5.3. Competing risks with BART

Competing risks survival analysis (Kalbfleisch and Prentice 1980; Fine and Gray 1999; Klein and Moeschberger 2006; Nicolaie, van Houwelingen, and Putter 2010; Ishwaran, Gerds, Kogalur, Moore, Gange, and Lau 2014; Sparapani, Logan, McCulloch, and Laud 2019) deal with events which are mutually exclusive, say, death from cardiovascular disease vs. death from other causes, i.e., a patient experiencing one of the events is incapable of experiencing another. We take two approaches to support competing risks with BART: both approaches are extensions of BART survival analysis. We flexibly model the cause-specific hazards and

eschew precarious restrictive assumptions like linearity of covariate effects, proportionality and/or parametric distributions of the outcomes.

Competing risks with crisk.bart

The first approach is supported by the function `crisk.bart` for serial computation and `mc.crisk.bart` for parallel computation. To accomodate competing risks, we adapt our notation slightly: (s_i, δ_i) where $\delta_i = 1$ for kind 1 events, $\delta_i = 2$ for kind 2 events, or $\delta_i = 0$ for censoring times. We create a single grid of time points for the ordered distinct times based on either kind of event or censoring: $0 = t_{(0)} < t_{(1)} < \cdots < t_{(K)} < \infty$. We model the probability for an event of kind 1, $p_1(t_{(j)}, \boldsymbol{x}_i)$, and an event of kind 2 conditioned on subject i being alive at time $t_{(j)}$, $p_2(t_{(j)}, \boldsymbol{x}_i)$. Now, we create event indicators by melding absorbing events survival analysis with mutually exclusive Multinomial categories where i indexes subjects: $i = 1, \ldots, N$.

$$
\begin{aligned}
y_{1ij} &= \mathrm{I}(\delta_i = 1)\,\mathrm{I}(j = n_i) \text{ where } j = 1, \ldots, n_i \\
y_{1ij} | p_{1ij} &\sim \mathrm{B}(p_{1ij}) \\
p_{1ij} &= \Phi(\mu_1 + f_1(t_{(j)}, \boldsymbol{x}_i)) \text{ where } f_1 \overset{\text{prior}}{\sim} \text{BART} \\
y_{2ij} &= \mathrm{I}(\delta_i = 2)\,\mathrm{I}(j = n_i) \text{ where } j = 1, \ldots, n_i - y_{1in_i} \\
y_{2ij} | p_{2ij} &\sim \mathrm{B}(p_{2ij}) \\
p_{2ij} &= \Phi(\mu_2 + f_2(t_{(j)}, \boldsymbol{x}_i)) \text{ where } f_2 \overset{\text{prior}}{\sim} \text{BART}
\end{aligned}
$$

The likelihood is: $[\boldsymbol{y}|f_1, f_2] = \prod_{i=1}^{N} \prod_{j=1}^{n_i} p_{1ij}^{y_{1ij}} (1 - p_{1ij})^{1 - y_{1ij}} \prod_{j'=1}^{n_i - y_{1in_i}} p_{2ij'}^{y_{2ij'}} (1 - p_{2ij'})^{1 - y_{2ij'}}$. Now, we can estimate the survival function and the cumulative incidence functions as follows.

$$
\begin{aligned}
S(t, \boldsymbol{x}_i) &= 1 - F(t, \boldsymbol{x}_i) = \prod_{j=1}^{k} (1 - p_{1ij})(1 - p_{2ij}) \text{ where } k = \arg\max_j \left[t_{(j)} \leq t \right] \\
F_1(t, \boldsymbol{x}_i) &= \int_0^t S(u-, \boldsymbol{x}_i) \lambda_1(u, \boldsymbol{x}_i) \mathrm{d}u = \sum_{j=1}^{k} S(t_{(j-1)}, \boldsymbol{x}_i) p_{1ij} \\
F_2(t, \boldsymbol{x}_i) &= \int_0^t S(u-, \boldsymbol{x}_i) \lambda_2(u, \boldsymbol{x}_i) \mathrm{d}u = \sum_{j=1}^{k} S(t_{(j-1)}, \boldsymbol{x}_i)(1 - p_{1ij}) p_{2ij}
\end{aligned}
$$

The returned object of type `criskbart` from `crisk.bart` or `mc.crisk.bart` provides the cumulative incidence functions and survival corresponding to `x.test` as follows: F_1 is `cif.test`, F_2 is `cif.test2` and S is `surv.test`.

Competing risks with crisk2.bart

The second approach is supported by the function `crisk2.bart` for serial computation and `mc.crisk2.bart` for parallel computation. We take a similar approach as Nicolaie *et al.* (2010). We model the probability for an event of either kind, $p_{ij} = p(t_{(j)}, \boldsymbol{x}_i)$ (this is standard survival analysis); and, given an event has occurred, the probability of a kind 1 event, $\pi_i = \pi(t_i, \boldsymbol{x}_i)$. Now, we create the corresponding event indicators y_{ij} and u_i where i indexes

subjects: $i = 1, \ldots, N$.

$$
\begin{aligned}
y_{ij} &= \mathrm{I}\left(\delta_i \neq 0\right) \mathrm{I}\left(j = n_i\right) \text{ where } j = 1, \ldots, n_i \\
y_{ij}|p_{ij} &\sim \mathrm{B}\left(p_{ij}\right) \\
p_{ij} &= \Phi(\mu_y + f_y(t_{(j)}, \boldsymbol{x}_i)) \text{ where } f_y \overset{\text{prior}}{\sim} \text{BART} \\
u_i &= \mathrm{I}\left(\delta_i = 1\right) \text{ where } i \in \{i' : \delta_{i'} \neq 0\} \\
u_i|\pi_i &\sim \mathrm{B}\left(\pi_i\right) \\
\pi_i &= \Phi(\mu_u + f_u(t_i, \boldsymbol{x}_i)) \text{ where } f_u \overset{\text{prior}}{\sim} \text{BART}
\end{aligned}
$$

The likelihood is: $[\boldsymbol{y}, \boldsymbol{u}|f_y, f_u] = \prod_{i=1}^{N} \prod_{j=1}^{n_i} p_{ij}^{y_{ij}} (1 - p_{ij})^{1-y_{ij}} \prod_{i':\delta_{i'} \neq 0} \pi_{i'}^{u_{i'}} (1 - \pi_{i'})^{1-u_{i'}}$. Now, we can estimate the survival function and the cumulative incidence functions similar to the first approach. The returned object of type `crisk2bart` from `crisk2.bart` or `mc.crisk2.bart` provides the cumulative incidence functions and survival corresponding to `x.test` as follows: F_1 is `cif.test`, F_2 is `cif.test2` and S is `surv.test`.

Competing risks with BART example: liver transplants

Here, we present the Mayo Clinic liver transplant waiting list data from 1990-1999 with $N = 815$ patients. During the study period, the liver transplant organ allocation policy was flawed. Blood type is an important matching factor to avoid organ rejection. Donor livers from subjects with blood type O can be used by patients with A, B, AB or O blood types; whereas a donor liver from the other types will only be transplanted to a matching recipient. Therefore, type O subjects on the waiting list were at a disadvantage since the pool of competitors was larger for type O donor livers. This data is of historical interest and provides a useful example of competing risks, but it has little relevance to liver transplants today. Current liver transplant policies have evolved and now depend on each individual patient's risk/need which are assessed and updated regularly while a patient is on the waiting list. Nevertheless, there still remains an acute shortage of donor livers today. The `transplant` data set is provided by the **BART3** R package as is this example: `demo("liver.crisk.bart", package="BART")`. We compare the nonparametric Aalen-Johansen competing risks estimator with BART for the transplant event of type O patients which are in general agreement; see Figure 15.

5.4. Recurrent events analysis with BART

The **BART3** package supports recurrent events (Sparapani, Rein, Tarima, Jackson, and Meurer 2018) with `recur.bart` for serial computation and `mc.recur.bart` for parallel computation. Survival analysis is generally concerned with absorbing events that a subject can only experience once like mortality. Recurrent events analysis is concerned with non-absorbing events that a subject can experience more than once like hospital admissions (Andersen and Gill 1982; Wei, Lin, and Weissfeld 1989; Kalbfleisch and Prentice 2002; Sparapani *et al.* 2018). Recurrent events analysis with BART provides much desired flexibility in modeling the dependence of recurrent events on covariates. Consider data in the form: $\delta_i, s_i, \boldsymbol{t}_i, \boldsymbol{u}_i, \boldsymbol{x}_i(t)$ where $i = 1, \ldots, N$ indexes subjects; s_i is the end of the observation period (death, $\delta_i = 1$, or censoring, $\delta_i = 0$); N_i is the number of events during the observation period; $\boldsymbol{t}_i = [t_{i1}, \ldots, t_{iN_i}]$ and t_{ik} is the event start time of the kth event (let $t_{i0} = 0$); $\boldsymbol{u}_i = [u_{i1}, \ldots, u_{iN_i}]$ and u_{ik} is the event end time of the kth event (let $u_{i0} = 0$); and $\boldsymbol{x}_i(t)$ is a vector of time-dependent

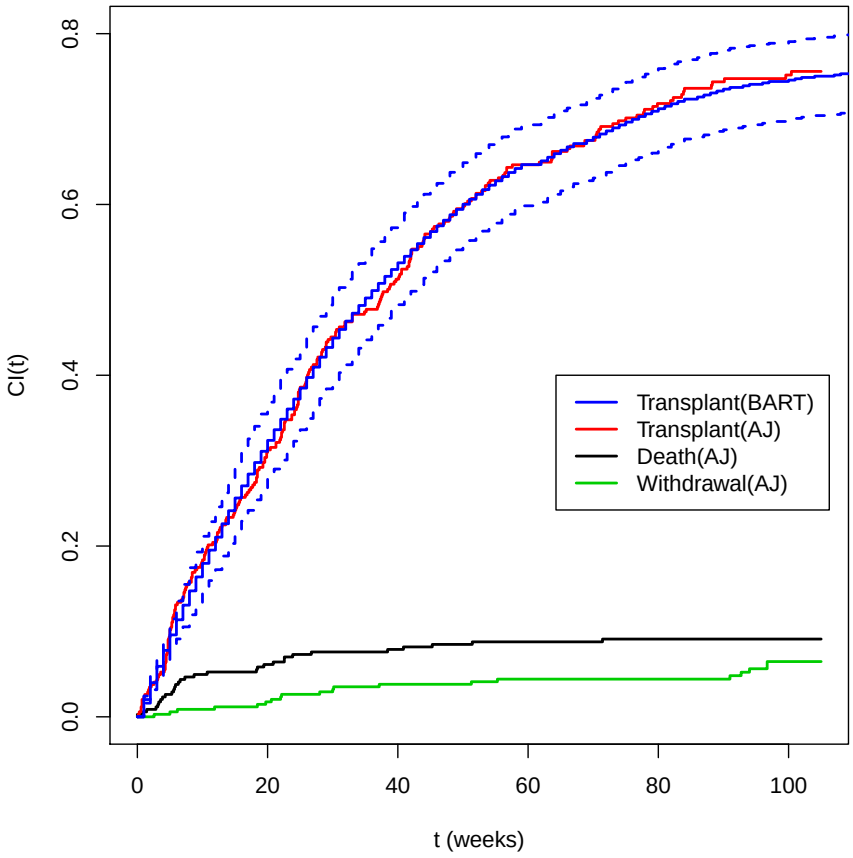


Figure 15: Liver transplant competing risks for type O patients estimated by BART and Aalen-Johansen. This data is from the Mayo Clinic liver transplant waiting list from 1990-1999. During the study period, the liver transplant organ allocation policy was flawed. Blood type is an important matching factor to avoid organ rejection. Donor livers from subjects with blood type O can be used by patients with all blood types; whereas a donor liver from the other types will only be transplanted to a matching recipient. Therefore, type O subjects on the waiting list were at a disadvantage since the pool of competitors was larger.

covariates. Both start and end times of events are necessary to define risk set eligibility for events of stochastic duration like readmissions since patients currently hospitalized cannot be readmitted. For instantaneous events (or roughly instantaneous events such as emergency department visits with time measured in days), the end times can be simply ignored.

We denote the K collectively distinct event start and end times for all subjects by $0 < t_{(1)} < \cdots < t_{(K)} < \infty$ thus taking $t_{(j)}$ to be the j^{th} order statistic among distinct observation times and, for convenience, $t_{(j')} = 0$ where $j' \leq 0$ (note that $t_{(j)}$ are constructed from all event start/end times for all subjects, but they may be a censoring time for any given subject). Now consider binary event indicators y_{ij} for each subject i at each distinct time $t_{(j)}$ up to the subject's last observation time $t_{(n_i)} \leq s_i$ with $n_i = \arg\max_j \left[t_{(j)} \leq s_i\right]$, i.e., $y_{i1}, \ldots, y_{in_i} \in \{0, 1\}$. We then denote by p_{ij} the probability of an event at time $t_{(j)}$ conditional on $\left(t_{(j)}, \tilde{\boldsymbol{x}}_i(t_{(j)})\right)$ where $\tilde{\boldsymbol{x}}_i(t_{(j)}) = \left(N_i(t_{(j-1)}), v_i(t_{(j)}), \boldsymbol{x}_i(t_{(j)})\right)$. Let $N_i(t-) \equiv \lim_{s \uparrow t} N_i(s)$ be the number of events for subject i just prior to time t and we also note that $N_i = N_i(s_i)$. Let $v_i(t) = t - u_{N_i(t-)}$ be the sojourn time for subject i, i.e., time since last event, if any. Notice that we can replace $N_i(t_{(j)}-)$ with $N_i(t_{(j-1)})$ since, by construction, the state of information available at time $t_{(j)}-$ is the same as that available at $t_{(j-1)}$. Assuming a constant intensity and constant covariates, $\tilde{\boldsymbol{x}}_i(t_{(j)})$, in the interval $(t_{(j-1)}, t_{(j)}]$, we define the cumulative intensity process as:

$$\Lambda(t_{(j)}, \tilde{\boldsymbol{x}}_i(t_{(j)})) = \int_0^{t_{(j)}} \mathrm{d}\Lambda(t, \tilde{\boldsymbol{x}}_i(t)) = \sum_{j'=1}^{j} \Pr N_i(t_{(j')}) - N_i(t_{(j'-1)}) = 1 \mid t_{(j')}, \tilde{\boldsymbol{x}}_i(t_{(j')}) = \sum_{j'=1}^{j} p_{ij'} \tag{1}$$

where these p_{ij} are currently unspecified and we provide their definition later in Equation 2. N.B. we follow the recurrent events literature's favored terminology by using the term "intensity" rather than "hazard", but they are generally interchangeable.

With absorbing events such as mortality there is no concern about the conditional independence of future events because there will never be any. Conversely, with recurrent events, there is a valid concern. Of course, conditional independence can be satisfied by conditioning on the entire event history, denoted by $N_i(s)$ where $0 \leq s < t$. However, conditioning on the entire event history is often impractical. Rather, we condition on both $N_i(t-)$ and $v_i(t)$ to satisfy any concern of conditional independence.

We now write the model for y_{ij} as a nonparametric probit regression of y_{ij} on $\left(t_{(j)}, \tilde{\boldsymbol{x}}_i(t_{(j)})\right)$ tantamount to parametric models of discrete-time intensity (Thompson Jr. 1977; Arjas and Haara 1987; Fahrmeir 2014). Specifically, with temporal data converted from $\delta_i, s_i, \boldsymbol{t}_i, \boldsymbol{u}_i, \boldsymbol{x}_i(t)$ to a sequence of longitudinal binary events as follows: $y_{ij} = \max_k \mathrm{I}\left(t_{ik} = t_{(j)}\right)$. However, note that the definition of j is currently unspecified. To understand the impetus of the range of j, let's look at an example.

Suppose that we have two subjects with the following values:

$$\begin{aligned} N_1 = 2, s_1 = 9, t_{11} = 3, u_{11} = 7, t_{12} = 8, u_{12} = 8 \Rightarrow y_{11} = 1, y_{12} = y_{13} = 0, y_{14} = 1, y_{15} = 0 \\ N_2 = 1, s_2 = 12, t_{21} = 4, u_{21} = 7 \Rightarrow y_{21} = 0, y_{22} = 1, y_{23} = y_{24} = y_{25} = y_{26} = 0 \end{aligned} \tag{2.3}$$

which creates the grid of times $(3, 4, 7, 8, 9, 12)$. For subject 1 (2), notice that $y_{12} = y_{13} = 0$ ($y_{23} = 0$) as it should be since no event occurred at times 4 or 7 (7). However, there were no events since their first event had not ended yet, i.e., these subjects are not chronologically

at risk for an event and, therefore, no corresponding random behavior contributed to the likelihood. The **BART3** package provides the `recur.pre.bart` function which you can use to construct these data sets. Here is a short demonstration of its capabilities adapted from `demo/data.recur.pre.bart.R` (re-formatted for display purposes).

```
R> library("BART")
R> times <- matrix(c(3, 8, 9, 4, 12, 12), nrow=2, ncol=3, byrow=TRUE)
R> tstop <- matrix(c(7, 8, 0, 7,  0,  0), nrow=2, ncol=3, byrow=TRUE)
R> delta <- matrix(c(1, 1, 0, 1,  0,  0), nrow=2, ncol=3, byrow=TRUE)
R> recur.pre.bart(times=times, delta=delta, tstop=tstop)

$K   $times   $y.train  $tx.train     $tx.test
[1]  [1]      [1]             t v N         t v N
 6    3        1        [1,]  3 3 0   [1,]  3 3 0
      4        1        [2,]  8 5 1   [2,]  4 1 1
      7        0        [3,]  9 1 2   [3,]  7 4 1
      8        0        [4,]  3 3 0   [4,]  8 5 1
      9        1        [5,]  4 4 0   [5,]  9 1 2
     12        0        [6,]  8 4 1   [6,] 12 4 2
               0        [7,]  9 5 1   [7,]  3 3 0
               0        [8,] 12 8 1   [8,]  4 4 0
                                      [9,]  7 3 1
                                     [10,]  8 4 1
                                     [11,]  9 5 1
                                     [12,] 12 8 1
```

Notice that `$tx.test` is not limited to the same time points as `$tx.train`, i.e., we often want/need to estimate f at counter-factual values not observed in the data so each subject contributes an equal number of evaluations for estimation purposes.

It is now clear that the y_{ij} which contribute to the likelihood are those such that $j \in R_i$ which is the risk set for subject i. We formally define the risk set as

$$R_i = \left\{ j : \left[j \in \{1, \ldots, n_i\} \text{ and } \cap_{k=1}^{N_i} \{ t_{(j)} \notin (t_{ik}, u_{ik}) \} \right] \right\}$$

i.e., the risk set contains j if $t_{(j)}$ is during the observation period for subject i and $t_{(j)}$ is not contained within an already ongoing event for this subject.

Putting it all together, we arrive at the following recurrent events discrete-time model with i indexing subjects; $i = 1, \ldots, N$.

$$\begin{aligned} y_{ij}|p_{ij} &\sim \mathrm{B}\left(p_{ij}\right) \text{ where } j \in R_i \\ p_{ij} &= \Phi(\mu_{ij}),\ \mu_{ij} = \mu_0 + f(t_{(j)}, \tilde{\boldsymbol{x}}_i(t_{(j)})) \\ f &\overset{\text{prior}}{\sim} \text{BART} \end{aligned} \tag{2}$$

This produces the following likelihood: $[\boldsymbol{y}|f] = \prod_{i=1}^{N} \prod_{j \in R_i} p_{ij}^{y_{ij}} (1 - p_{ij})^{1 - y_{ij}}$. We center the BART function, f, by $\mu_0 = \Phi^{-1}(\bar{y})$ where $\bar{y} = \frac{\sum_i \sum_{j \in R_i} y_{ij}}{\sum_i \sum_{j=1}^{n_i} \mathrm{I}\left(j \in R_i\right)}$.

For computational efficiency, we carry out the probit regression via truncated Normal latent variables (Albert and Chib 1993) (this default can be over-ridden for logit with Logistic latents (Holmes and Held 2006; Gramacy and Polson 2012) by specifying `type="lbart"`).

With the data prepared as described in the above example, the BART model for binary data treats the probability of an event within an interval as a nonparametric function of time, t, and covariates, $\tilde{\boldsymbol{x}}(t)$. Conditioned on the data, BART provides samples from the posterior distribution of f. For any t and $\tilde{\boldsymbol{x}}(t)$, we obtain the posterior distribution of $p(t, \tilde{\boldsymbol{x}}(t)) = \Phi(\mu_0 + f(t, \tilde{\boldsymbol{x}}(t)))$.

For the purposes of recurrent events survival analysis, we are typically interested in estimating the cumulative intensity function as presented in Equation 1. With these estimates, one can accomplish inference from the posterior via means, quantiles or other functions of $p(t, \tilde{\boldsymbol{x}}_i(t))$ or $\Lambda(t, \tilde{\boldsymbol{x}}(t))$ as needed such as the relative intensity, i.e., $RI(t, \tilde{\boldsymbol{x}}_n(t), \tilde{\boldsymbol{x}}_d(t)) = \frac{p(t, \tilde{\boldsymbol{x}}_n(t))}{p(t, \tilde{\boldsymbol{x}}_d(t))}$ where $\tilde{\boldsymbol{x}}_n(t)$ and $\tilde{\boldsymbol{x}}_d(t)$ are two settings we wish to compare like two treatments.

Recurrent events with BART example: bladder tumors

An interesting example of recurrent events involves a clinical trial conducted by the Veterans Administration Cooperative Urological Research Group (Byar 1980). In this study, all patients had superficial bladder tumors when they entered the trial. These tumors were removed transurethrally and patients were randomly assigned to one of three treatments: placebo, thiotepa or pyridoxine (vitamin B6). Many patients had multiple recurrences of tumors during the study and new tumors were removed at each visit. For each patient, their recurrence time, if any, was measured from the beginning of treatment. There were 118 patients enrolled but only 116 were followed beyond time zero and contribute information. This data set is loaded by `data(bladder)` and the data frame of interest is `bladder1`. This data set is analyzed by `demo("bladder.recur.bart", package="BART")`. In Figure 16, notice that the relative intensity calculated by Friedman's partial dependence function finds thiotepa inferior to placebo from roughly 6 to 18 months and afterward they are about equal, but the 95% credible intervals are wide throughout. Similarly, the relative intensity calculated by Friedman's partial dependence function finds thiotepa inferior to vitamin B6 from roughly 3 to 24 months and afterward they are about equal, but the 95% credible intervals are wide throughout; see Figure 17. And, finally, vitamin B6 is superior to placebo throughout, but the 95% credible intervals are wide; see Figure 18.

6. Discussion

The **BART3** R package provides a user-friendly reference implementation of Bayesian Additive Regression Trees (BART). BART is a Bayesian nonparametric, tree-based ensemble, machine learning technique with best-of-breed properties. In the spirit of machine learning, BART *learns* the relationship between the covariates, $\boldsymbol{x}$, and the response variable arriving at $f(\boldsymbol{x})$ while not burdening the user to pre-specify the functional form of f nor the interaction terms among the covariates. By specifying an optional sparse Dirichlet prior, BART is capable of variable selection: a form of *learning* which is especially useful in high-dimensional settings. In the class of ensemble predictive models, BART's out-of-sample predictive performance is competitive with other leading members of this class. Due to its membership in the class of Bayesian nonparametric models, BART not only provides an estimate of $f(\boldsymbol{x})$, but naturally

generates the uncertainty as well.

There are user-friendly features that are inherent to BART itself which, of course, are available in this package as well. BART was designed to be very flexible via its prior arguments while providing the user robust, low information, default settings that will likely produce a good fit without resorting to computationally demanding cross-validation. BART itself is relatively computationally efficient, but larger data sets will naturally take more time to estimate. Therefore, the **BART3** package provides the user with simple and easy to use multi-threading to keep elapsed time to a minimum. Another important time-saver, the **BART3** package allows the user to save the trees from a BART model fit so that prediction via the R `predict` function can take place at a later time without having to re-fit the model. And these predictions can also take advantage of multi-threading.

The **BART3** package has been written in C++ for portability, maintainability and efficiency; this allows BART to be called either from R or from other computer source code written in many languages. The package supports missing data handling of the covariates and provides the user with access to BART implementations for several types of responses. The **BART3** package supports the following:

- continuous outcomes;
- binary outcomes via probit or logit transformation;
- categorical outcomes;
- time-to-event outcomes with right censoring including
  - absorbing events,
  - competing risks, and
  - recurrent events.

In this article, we have provided the user with an overview of much that is described in this section including (but not limited to): details of the BART prior and its arguments, sparse variable selection, prediction, multi-threading, support for the outcomes listed above and missing data handling. In addition, this article has provided primers on important BART topics such as posterior computation, Friedman's partial dependence function and convergence diagnostics. With a computational method such as BART, the user needs a reliable, well-documented software package with a diverse set of examples. With this article, and the **BART3** package itself, we believe that interested users now have the tools to successfully employ BART for their rigorous data analysis needs.

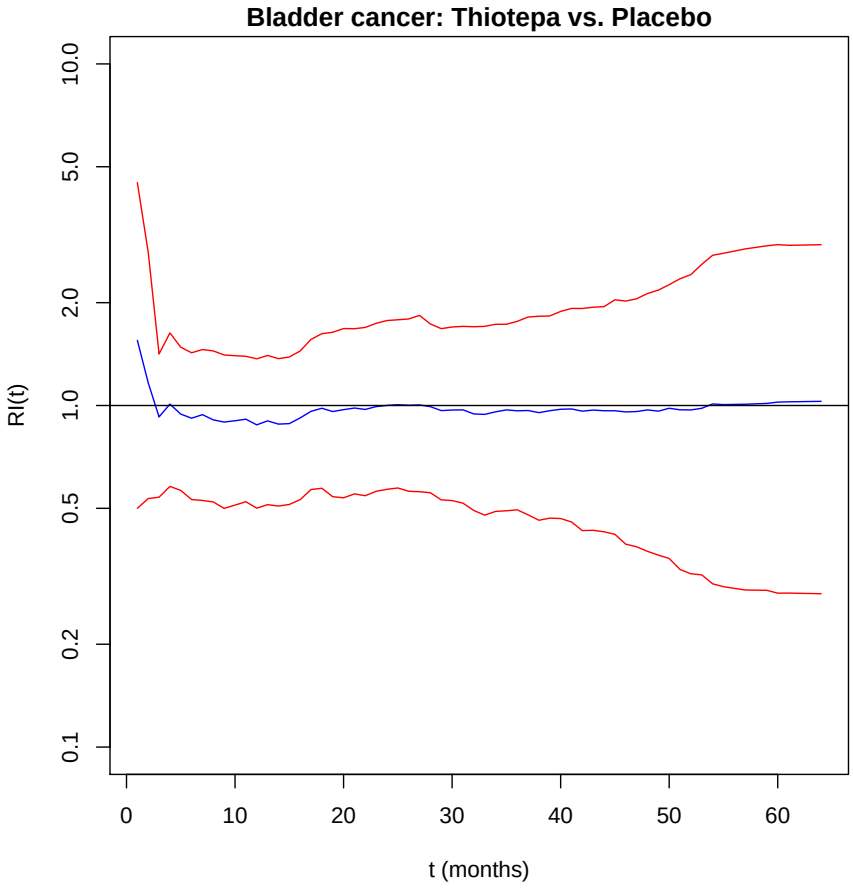


Figure 16: Relative Intensity: Thiotepa vs. Placebo. The relative intensity function is as follows: $RI(t, \tilde{\boldsymbol{x}}_T(t), \tilde{\boldsymbol{x}}_P(t)) = \frac{p(t,\tilde{\boldsymbol{x}}_T(t))}{p(t,\tilde{\boldsymbol{x}}_P(t))}$ where T is for Thiotepa and P is for Placebo. The blue lines are the relative intensity functions themselves and the red lines are their 95% credible intervals. The relative intensity is calculated by Friedman's partial dependence function, i.e., aggregated over all other covariates.

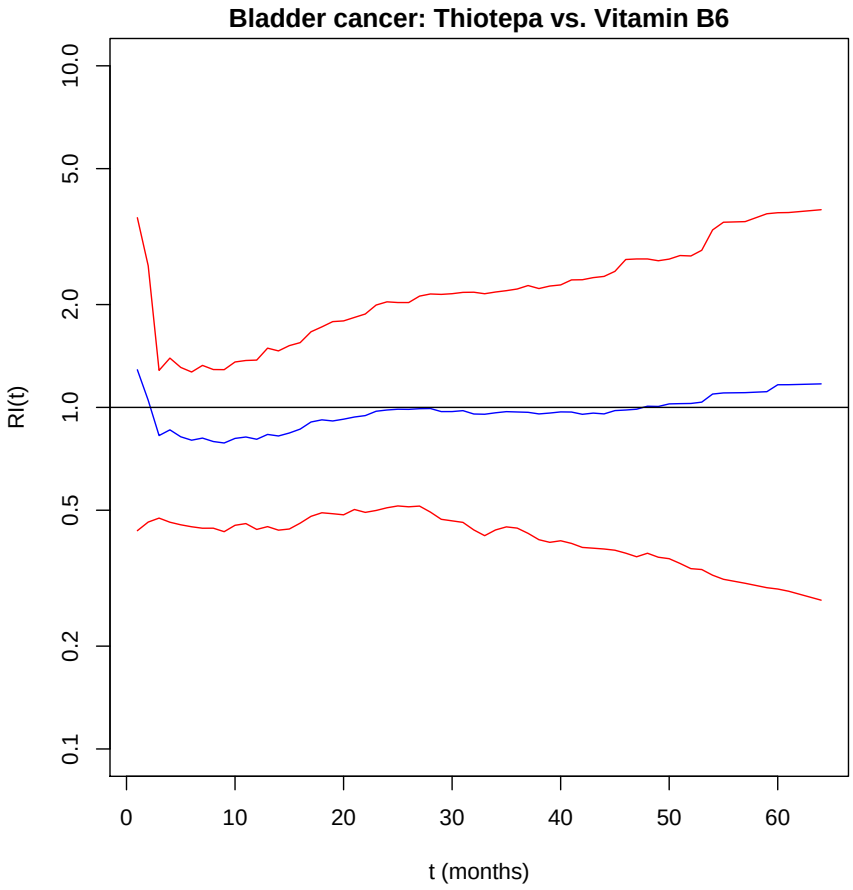


Figure 17: Relative Intensity: Thiotepa vs. Vitamin B6. The relative intensity function is as follows: $RI(t, \tilde{\boldsymbol{x}}_T(t), \tilde{\boldsymbol{x}}_B(t)) = \frac{p(t,\tilde{\boldsymbol{x}}_T(t))}{p(t,\tilde{\boldsymbol{x}}_B(t))}$ where T is for Thiotepa and B is for Vitamin B6. The blue lines are the relative intensity functions themselves and the red lines are their 95% credible intervals. The relative intensity is calculated by Friedman's partial dependence function, i.e., aggregated over all other covariates.

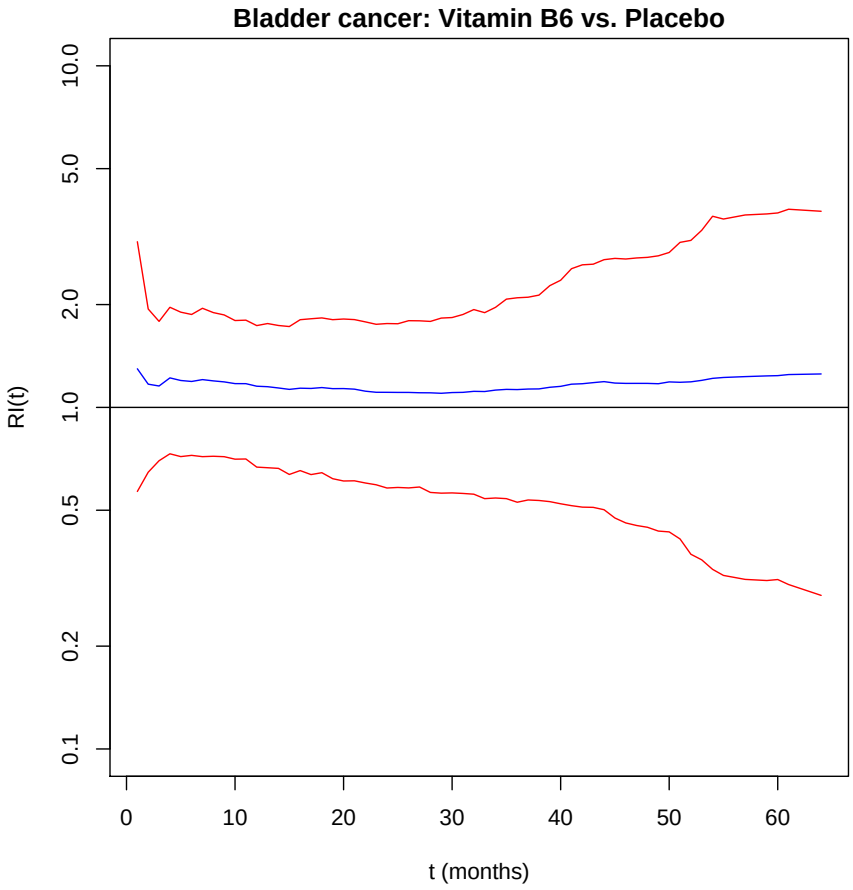


Figure 18: Relative Intensity: Vitamin B6 vs. Placebo. The relative intensity function is as follows: $RI(t, \tilde{\boldsymbol{x}}_B(t), \tilde{\boldsymbol{x}}_P(t)) = \frac{p(t,\tilde{\boldsymbol{x}}_B(t))}{p(t,\tilde{\boldsymbol{x}}_P(t))}$ where B is for Vitamin B6 and P is for Placebo. The blue lines are the relative intensity functions themselves and the red lines are their 95% credible intervals. The relative intensity is calculated by Friedman's partial dependence function, i.e., aggregated over all other covariates.

A. Getting and installing the BART R package

The **BART** package (McCulloch, Sparapani, Gramacy, Spanbauer, and Pratola 2019) is GNU General Public License (GPL) software available on the Comprehensive R Archive Network (CRAN). You can install it from CRAN as follows.

```
R> options(repos=c(CRAN="https://cran.r-project.org"))
R> install.packages("BART", dependencies=TRUE)
```

The examples in this article are included in the package. You can run the first example (described in Section 3) as follows.

```
R> options(figures=".")
R> if(.Platform$OS.type=="unix") {
R>   options(mc.cores=min(8, parallel::detectCores()))
R> } else {
R>   options(mc.cores=1)
R> }
R> demo("boston", package="BART"))
```

As we shall see, these examples produce R objects containing BART model fits. But, these fits are Bayesian nonparametric samples from the posterior and require statistical summarization before they are readily interpretable. Therefore, we often employ graphical summaries (such as the figures in this article) to visualize the BART model fit. Note that the `figures` option (in the code snippet above) specifies a directory where the Portable Document Format (PDF) graphics files will be produced; if it is not specified, then the graphics will be generated by R, however, no PDF files will be created. Furthermore, some of these BART model fits can take a few minutes so it is wise to utilize multi-threading when it is available (for a discussion of efficient computation with BART including multi-threading, see Appendix Section D). Returning to the snippet above, the option `mc.cores` specifies the number of cores to employ in multi-threading, e.g., there are diminishing returns so often 8 cores is sufficient. And, finally, to run all of the examples in this article (with the options as specified above), then do the following. `demo("replication", package="BART")`

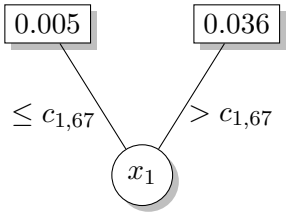
B. Binary trees and the BART prior

BART relies on an ensemble of H binary trees which are a type of a directed acyclic graph. We exploit the wooden tree metaphor to its fullest. Each of these trees grows from the ground up starting out as a root node. The root node is generally a branch decision rule, but it doesn't have to be; occasionally there are trees in the ensemble which are only a root terminal node consisting of a single leaf output value. If the root is a branch decision rule, then it spawns a left and a right node which each can be either a branch decision rule or a terminal leaf value and so on. In binary tree, $\mathcal{T}$, there are C nodes which are made of B branches and L leaves: $C = B + L$. There is an algebraic relationship between the number of branches and leaves which we express as $B = L - 1$.

The ensemble of trees is encoded in an ASCII string which is returned in the `treedraws$trees` list item. This string can be easily imported by R with the following:

```
R> write(post$treedraws$trees, "trees.txt")
R> tc <- textConnection(post$treedraws$tree)
R> trees <- read.table(file=tc, fill=TRUE, row.names=NULL, header=FALSE,
+    col.names=c("node", "var", "cut", "leaf"))
R> close(tc)
R> head(trees)
```

```
  node var cut         leaf
1 1000 200   1           NA
2    3  NA  NA           NA
3    1   0  66 -0.001032108
4    2   0   0  0.004806880
5    3   0   0  0.035709372
6    3  NA  NA           NA
```



The string is encoded via the following binary tree notation. The first line is an exception which has the number of MCMC samples, M, in the field `node`; the number of trees, H, in the field `var`; and the number of variables, P, in the field `cut`. For the rest of the file, the field `node` is used for the number of nodes in the tree when all other fields are `NA`; or for a specific node when the other fields are present. The nodes are numbered in relation to the tree's tier level, $t(n) = \lfloor \log_2 n \rfloor$ or `t=floor(log2(node))`, as follows.

Tier							
t	2^t		$\ldots$		$2^{t+1}-1$		
$\vdots$							
2	4		5		6		7
1		2				3	
0				1			

The `var` field is the variable in the branch decision rule which is encoded $0, \ldots, P-1$ as a C/C++ array index (rather than an R index). Similarly, the `cut` field is the cutpoint of the variable in the branch decision rule which is encoded $0, \ldots, c_j - 1$ for variable j; note that the cutpoints are returned in the `treedraws$cutpoints` list item. The terminal leaf output value is contained in the field `leaf`. It is not immediately obvious which nodes are branches vs. leaves since, at first, it would appear that the `leaf` field is given for both branches and leaves. Leaves are always associated with `var=0` and `cut=0`; however, note that this is also a valid branch variable/cutpoint since these are C/C++ indices. The key to discriminating between branches and leaves is via the algebraic relationship between a branch, n, at tree tier $t(n)$ leading to its left, $l = 2n$, and right, $r = 2n+1$, nodes at tier $t(n)+1$, i.e., for each node, besides root, you can determine from which branch it arose and those nodes that are not a branch (since they have no leaves) are necessarily leaves.

Underlying this methodology is the BART prior. The BART prior specifies a flexible class of unknown functions, f, from which we can gather randomly generated fits to the given data via the posterior. N.B. we define f as returning a scalar value, but BART extensions which return multivariate values are conceivable. Let the function $g(\boldsymbol{x}; \mathcal{T}, \mathcal{M})$ assign a value based on the input $\boldsymbol{x}$. The binary decision tree $\mathcal{T}$ is represented by a set of ordered triples, (n, j, k), representing branch decision rules: $n \in \mathcal{B}$ for node n in the set of branches $\mathcal{B}$, j for covariate

x_j and k for the cutpoint c_{jk}. The branch decision rules are of the form $x_j \leq c_{jk}$ which means branch left and $x_j > c_{jk}$, branch right; or terminal leaves where it stops. $\mathcal{M}$ represents leaves and is a set of ordered pairs, (n, μ_n): $n \in \mathcal{L}$ where $\mathcal{L}$ is the set of leaves ($\mathcal{L}$ is the complement of $\mathcal{B}$) and μ_n for the outcome value.

The function, $f(\boldsymbol{x})$, is a sum of H trees:

$$f(\boldsymbol{x}) = \sum_{h=1}^{H} g(\boldsymbol{x}; \mathcal{T}_h, \mathcal{M}_h) \tag{3}$$

where H is "large", let's say, 50, 100 or 200.

For a continuous outcome, y_i, we have the following BART regression on the vector of covariates, $\boldsymbol{x}_i$:

$$y_i = \mu_0 + f(\boldsymbol{x}_i) + \epsilon_i \text{ where } \epsilon_i \overset{\text{iid}}{\sim} \mathrm{N}\left(0,\ w_i^2\sigma^2\right)$$

with i indexing subjects $i = 1, \ldots, N$. The unknown random function, f, and the error variance, σ^2, follow the BART prior expressed notationally as

$$(f, \sigma^2) \overset{\text{prior}}{\sim} \mathrm{BART}(H, \mu_0, \tau, k, \alpha, \gamma; \nu, \lambda, q)$$

where H is the number of trees, μ_0 is a known constant which centers y and the rest of the parameters will be explained later in this section (for brevity, we will often use the simpler shorthand $(f, \sigma^2) \overset{\text{prior}}{\sim} \mathrm{BART}$). The w_i are known standard deviation weight multiples which you can supply with the argument `w` that is only available for continuous outcomes, hence, the weighted BART name; the unit weight vector is the default. The centering parameter, μ_0, can be specified via the `fmean` argument where the default is taken to be $\bar{y}$.

BART is a Bayesian nonparametric prior. Using the Gelfand-Smith generic bracket notation for the specification of random variable distributions (Gelfand and Smith 1990), we represent the BART prior in terms of the collection of all trees, $\mathcal{T}$; collection of all leaves, $\mathcal{M}$; and the error variance, σ^2, as the following product: $[\mathcal{T}, \mathcal{M}, \sigma^2] = [\sigma^2]\,[\mathcal{T}, \mathcal{M}] = [\sigma^2]\,[\mathcal{T}]\,[\mathcal{M}|\mathcal{T}]$. Furthermore, the individual trees themselves are independent: $[\mathcal{T}, \mathcal{M}] = \prod_h [\mathcal{T}_h]\,[\mathcal{M}_h|\mathcal{T}_h]$. where $[\mathcal{T}_h]$ is the prior for the hth tree and $[\mathcal{M}_h|\mathcal{T}_h]$ is the collection of leaves for the hth tree. And, finally, the collection of leaves for the hth tree are independent: $[\mathcal{M}_h|\mathcal{T}_h] = \prod_n [\mu_{hn}|\mathcal{T}_h]$ where n indexes the leaf nodes.

The tree prior: $[\mathcal{T}_h]$. There are three prior components of $\mathcal{T}_h$ which govern whether the tree branches grow or are pruned. The first tree prior regularizes the probability of a branch at leaf node n in tree tier $t(n) = \lfloor \log_2 n \rfloor$ as

$$\mathrm{P}\left[B_n = 1\right] = \alpha(t(n) + 1)^{-\gamma} \tag{4}$$

where $B_n = 1$ represents a branch while $B_n = 0$ is a leaf, $0 < \alpha < 1$ and $\gamma \geq 0$. You can specify these prior parameters with arguments, but the following defaults are recommended: α is set by the parameter `base=0.95` and γ by `power=2`; for a detailed discussion of these parameter settings, see Chipman *et al.* (1998). Note that this prior penalizes branch growth, i.e., in prior probability, the default number of branches will likely be 1 or 2. Next, there is a prior dictating the choice of a splitting variable j conditional on a branch event B_n which defaults to uniform probability $s_j = P^{-1}$ where P is the number of covariates (however, you

can specify a Dirichlet prior which is more appropriate if the number of covariates is large (Linero 2018); see below). Given a branch event, B_n, and a variable chosen, x_j, the last tree prior selects a cut point, c_{jk}, within the range of observed values for x_j; this prior is uniform. We can also represent the probability of variable selection via the sparse Dirichlet prior as $[s_1, \ldots, s_P] \,|\theta \overset{\text{prior}}{\sim} \text{Dirichlet}\,(\theta/P, \ldots, \theta/P)$ which is specified by the argument `sparse=TRUE` while the default is `sparse=FALSE` for uniform $s_j = P^{-1}$. The prior parameter θ can be fixed or random: supplying a positive number will specify θ fixed at that value while the default `theta=0` is random and its value will be learned from the data. The random θ prior is induced via $\theta/(\theta+\rho) \overset{\text{prior}}{\sim} \text{Beta}\,(a,\ b)$ where the parameter ρ can be specified by the argument `rho` (which defaults to `NULL` representing the value P; provide a value to over-ride), the parameter b defaults to 1 (which can be over-ridden by the argument `b`) and the parameter a defaults to 0.5 (which can be over-ridden by the argument `a`). The distribution of `theta` controls the sparsity of the model: `a=0.5` induces a sparse posture while `a=1` is not sparse and similar to the uniform prior with probability $s_j = P^{-1}$. If additional sparsity is desired, then you can set the argument `rho` to a value smaller than P.

Here, we take the opportunity to provide some insight into how and why the sparse prior works as desired. The key to understanding the inducement of sparsity is the distribution of the arguments to the Dirichlet prior: θ/P. We are unaware of this result appearing elsewhere in the literature. But, it can be shown that $\theta/P \sim F(a, b, \rho/P)$ where $F(.)$ is the Beta Prime distribution scaled by ρ/P (Johnson, Kotz, and Balakrishnan 1995). The non-sparse setting is $(a, b, \rho/P) = (1, 1, 1)$. As you can see in the Figure 19, sparsity is increased by reducing ρ, reducing a or reducing both.

Unlike matrices, data frames can contain categorical factors. Therefore, factors can be supplied when `x.train` is a data frame. Factors with multiple levels are transformed into dummy variables with each level as their own binary indicator; factors with only two levels are a binary indicator with a single dummy variable.

The leaf prior: $[\mu_{hn}|\mathcal{T}_h]$. Given a tree, $\mathcal{T}_h$, there is a prior on its leaf values, $\mu_{hn}|\mathcal{T}_h$ and we denote the collection of all leaves in $\mathcal{T}_h$ by $\mathcal{M}_h = \{(n, \mu_{hn}) : n \in \mathcal{L}_h\}$. Suppose that $y_i \in [y_{\min}, y_{\max}]$ for all i and denote $\left[\mu_{1(i)}, \ldots, \mu_{H(i)}\right]$ as the leaf output values from each tree corresponding to the vector of covariates, $\boldsymbol{x}_i$. If $\mu_{h(i)}|\mathcal{T}_h \overset{\text{iid}}{\sim} \text{N}\left(0,\ \sigma_\mu^2\right)$, then the model estimate for subject i is $\mu_i = \text{E}\,[y_i|\boldsymbol{x}_i] = \mu_0 + \sum_h \mu_{h(i)}$ where $\mu_i \sim \text{N}\left(\mu_0,\ H\sigma_\mu^2\right)$. We choose a value for σ_μ which is the solution to the equations $y_{\min} = \mu_0 - k\sqrt{H}\sigma_\mu$ and $y_{\max} = \mu_0 + k\sqrt{H}\sigma_\mu$, i.e., $\sigma_\mu = \frac{y_{\max} - y_{\min}}{2k\sqrt{H}}$. Therefore, we arrive at $\mu_{hn} \overset{\text{prior}}{\sim} \text{N}\left(0,\ \left[\frac{\tau}{2k\sqrt{H}}\right]^2\right)$ where $\tau = y_{\max} - y_{\min}$. So, the prior for μ_{hn} is informed by the data, y, but only weakly via the extrema, $y_{\min}$ and $y_{\max}$. The parameter k calibrates this prior as follows.

$$\mu_i \sim \text{N}\left(\mu_0,\ \left[\frac{\tau}{2k}\right]^2\right)$$

$$\text{P}\,[y_{\min} \le \mu_i \le y_{\max}] = \Phi(k) - \Phi(-k)$$

$$\text{Since P}\,[\mu_i \le y_{\max}] = \text{P}\left[z \le 2k\frac{y_{\max} - \mu_0}{\tau}\right] \approx \text{P}\,[z \le k] = \Phi(k)$$

$$\text{Similarly P}\,[\mu_i \le y_{\min}] = \Phi(-k)$$

The default value, $k = 2$, corresponds to μ_i falling within the extrema with approximately 0.95 probability. Alternative choices of k can be supplied via the `k` argument. We have found

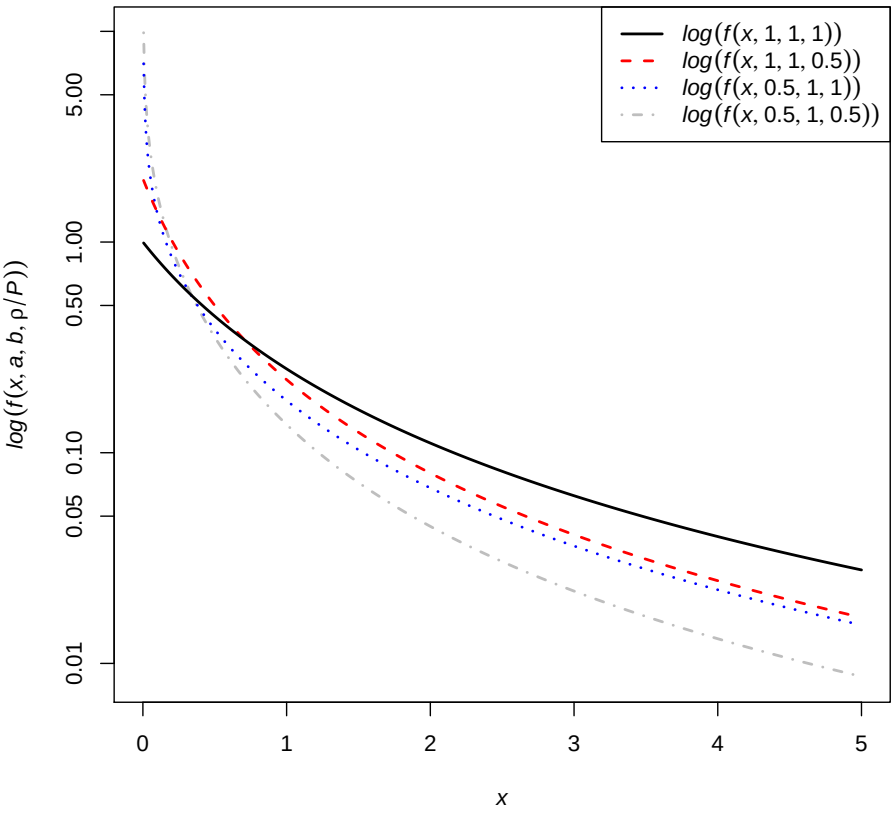


Figure 19: The distribution of θ/P and the sparse Dirichlet prior. The key to understanding the inducement of sparsity is the distribution of the arguments to the Dirichlet prior: $\theta/P \sim F(a, b, \rho/P)$ where $F(.)$ is the Beta Prime distribution scaled by ρ/P. Here we plot the natural logarithm of the scaled Beta Prime density, $f(.)$, at a non-sparse setting and three sparse settings. The non-sparse setting is $(a, b, \rho/P) = (1, 1, 1)$ (solid black line). As you can see in the figure, sparsity is increased by reducing ρ (long dashed red line), reducing a (short dashed blue line) or reducing both (mixed dashed gray line).

that values of $k \in [1, 3]$ generally yield good results. Note that k is a potential candidate parameter for choice via cross-validation.

The error variance prior: $\left[\sigma^2\right]$. The prior for σ^2 is the conjugate scaled inverse Chi-square distribution, i.e., $\nu\lambda\chi^{-2}(\nu)$. We recommend that the degrees of freedom, ν, be from 3 to 10 and the default is 3 which can be over-ridden by the argument `sigdf`. The λ parameter can be specified by the `lambda` argument which defaults to `NA`. If `lambda` is unspecified, then we determine a reasonable value for λ based on an estimate, $\widehat{\sigma}$, (which can be specified by the argument `sigest` and defaults to `NA`). If `sigest` is unspecified, the default value of `sigest` is determined via linear regression or the sample standard deviation: if $P < N$, then $y_i \sim \mathrm{N}\left(\boldsymbol{x}_i'\widehat{\boldsymbol{\beta}},\ \widehat{\sigma}^2\right)$; otherwise, $\widehat{\sigma} = s_y$. Now we solve for λ such that $\mathrm{P}\left[\sigma^2 \leq \widehat{\sigma}^2\right] = q$. This quantity, q, can be specified by the argument `sigquant` and the default is 0.9 whereas we also recommend considering 0.75 and 0.99. Note that the pair (ν, q) are potential candidate parameters for choice via cross-validation.

Other important arguments for the BART prior. We fix the number of trees at H which corresponds to the argument `ntree`. The default number of trees is 200 for continuous outcomes; as shown by Bleich *et al.* (2014), 50 is also a reasonable choice which is the default for all other outcomes: cross-validation could be considered. The number of cutpoints is provided by the argument `numcut` and the default is 100. The default number of cutpoints is achieved for continuous covariates. For continuous covariates, the cutpoints are uniformly distributed by default, or generated via uniform quantiles if the argument `usequants=TRUE` is provided. By default, discrete covariates which have fewer than 100 values will necessarily have fewer cutpoints. However, if you want a single discrete covariate to be represented by a group of binary dummy variables, one for each category, then pass the variable as a factor within a data frame.

C. Posterior computation for BART

In order to generate samples from the posterior for f, we sample the structure of all the trees $\mathcal{T}_h$, for $h = 1, \ldots, H$; the values of all leaves μ_{hn} for $n \in \mathcal{L}_h$ within tree h; and, when appropriate, the error variance σ^2. Additionally, with the sparsity prior, there are samples of the vector of splitting variable selection probabilities $[s_1, \ldots, s_P]$ and, when the sparsity parameter is random, samples of θ.

The leaf and variance parameters are sampled from the posterior using Gibbs sampling (Geman and Geman 1984; Gelfand and Smith 1990). Since the priors on these parameters are conjugate, the Gibbs conditionals are specified analytically. For the leaves, each μ_{hn} is drawn from a Normal conditional density. The error variance, σ^2, is drawn from a scaled inverse Chi-square conditional.

Drawing a tree from the posterior requires a Metropolis-within-Gibbs sampling scheme (Mueller 1991, 1993), i.e., a Metropolis-Hastings (MH) step (Metropolis, Rosenbluth, Rosenbluth, Teller, and Teller 1953; Hastings 1970) within Gibbs sampling. For single-tree models, four different proposal mechanisms are defined (Chipman *et al.* 1998) (N.B. other MCMC tree sampling strategies have been proposed: Denison, Mallick, and Smith (1998); Wu, Tjelmeland, and West (2007); Pratola (2016)). The complementary BIRTH/DEATH proposals are essential (the two other proposals are CHANGE and SWAP (Chipman *et al.* 1998)). For programming simplicity, the **BART** package only implements the BIRTH and DEATH proposals

each with equal probability. BIRTH selects a leaf and turns it into a branch, i.e., selects a new variable and cutpoint with two leaves "born" as its descendants. DEATH selects a branch leading to two terminal leaves and "kills" the branch by replacing it with a single leaf. To illustrate this discussion, we present the acceptance probability for a BIRTH proposal. Note that a DEATH proposal is the reversible inverse of a BIRTH proposal.

The algorithm assumes a fixed discrete set of possible split values for each x_j. Furthermore, the leaf values, μ_{hn}, are integrated over so that our search in tree space is over a large, but discrete, set of possibilities. At the mth MCMC step, let $\mathcal{T}^m$ denote the current state for the hth tree and $\mathcal{T}^*$ denotes the proposed hth tree (subscript h is suppressed for convenience). $\mathcal{T}^*$ are identical $\mathcal{T}^m$ except that one terminal leaf of $\mathcal{T}^m$ is replaced by a branch of $\mathcal{T}^*$ with two terminal leaves. The proposed tree is accepted with the following probability:

$$\pi_{\text{BIRTH}} = \min\left(1, \frac{\mathrm{P}\left[\mathcal{T}^*\right]}{\mathrm{P}\left[\mathcal{T}^m\right]} \frac{\mathrm{P}\left[\mathcal{T}^m|\mathcal{T}^*\right]}{\mathrm{P}\left[\mathcal{T}^*|\mathcal{T}^m\right]}\right)$$

where $\mathrm{P}\left[\mathcal{T}^m\right]$ and $\mathrm{P}\left[\mathcal{T}^*\right]$ are the posterior probabilities of T^m and T^* respectively. These are the targets of this sampling, each consisting of a likelihood contribution and prior contribution. Additionally, $\mathrm{P}\left[\mathcal{T}^m|\mathcal{T}^*\right]$ is the probability of proposing $\mathcal{T}^m$ given current state $\mathcal{T}^*$ (a DEATH) and $\mathrm{P}\left[\mathcal{T}^*|\mathcal{T}^m\right]$ is the probability of proposing $\mathcal{T}^*$ given current state $\mathcal{T}^m$ (a BIRTH).

First, we describe the likelihood contribution to the posterior. Let $\boldsymbol{y}_n$ denote the partition of $\boldsymbol{y}$ corresponding to the leaf node n given the tree $\mathcal{T}$. Because the leaf values are a priori conditionally independent, we have $\left[\boldsymbol{y}|\mathcal{T}\right] = \prod_n \left[\boldsymbol{y}_n|\mathcal{T}\right]$. So, for the ratio $\frac{\mathrm{P}[\mathcal{T}^*]}{\mathrm{P}[\mathcal{T}^m]}$ after cancellation of terms in the numerator and denominator, we have the likelihood contribution:

$$\frac{\mathrm{P}\left[\boldsymbol{y}_{\mathrm{L}}, \boldsymbol{y}_{\mathrm{R}}|\mathcal{T}^*\right]}{\mathrm{P}\left[\boldsymbol{y}_{\mathrm{LR}}|\mathcal{T}^m\right]} = \frac{\mathrm{P}\left[\boldsymbol{y}_{\mathrm{L}}|\mathcal{T}^*\right]\mathrm{P}\left[\boldsymbol{y}_{\mathrm{R}}|\mathcal{T}^*\right]}{\mathrm{P}\left[\boldsymbol{y}_{\mathrm{LR}}|\mathcal{T}^m\right]}$$

where $\boldsymbol{y}_{\mathrm{L}}$ is the partition corresponding to the newborn left leaf node; $\boldsymbol{y}_{\mathrm{R}}$, the partition for the newborn right leaf node; and $\boldsymbol{y}_{\mathrm{LR}} = \begin{bmatrix}\boldsymbol{y}_{\mathrm{L}} \\ \boldsymbol{y}_{\mathrm{R}}\end{bmatrix}$. N.B. the terms in the ratio are the predictive densities of a Normal mean with a known variance and a Normal prior for the mean.

Similarly, the terms that the prior contributes to the posterior ratio often cancel since there is only one "place" where the trees differ and the prior draws components independently at different "places" of the tree. Therefore, the prior contribution to $\frac{\mathrm{P}[\mathcal{T}^*]}{\mathrm{P}[\mathcal{T}^m]}$ is

$$\frac{\mathrm{P}\left[B_n = 1\right]\mathrm{P}\left[B_l = 0\right]\mathrm{P}\left[B_r = 0\right]s_j}{\mathrm{P}\left[B_n = 0\right]} = \frac{\alpha(t(n)+1)^{-\gamma}\left[1-\alpha(t(n)+2)^{-\gamma}\right]^2 s_j}{1-\alpha(t(n)+1)^{-\gamma}}$$

where $\mathrm{P}\left[B_n\right]$ is the branch regularity prior (see Equation 4), s_j is the splitting variable selection probability, n is the chosen leaf node in tree $\mathcal{T}^m$, $l = 2n$ is the newborn left leaf node in tree $\mathcal{T}^*$ and $r = 2n+1$ is the newborn right leaf node in tree $\mathcal{T}^*$.

Finally, the ratio $\frac{\mathrm{P}[\mathcal{T}^m|\mathcal{T}^*]}{\mathrm{P}[\mathcal{T}^*|\mathcal{T}^m]}$ is

$$\frac{\mathrm{P}\left[\text{DEATH}|\mathcal{T}^*\right]\mathrm{P}\left[n|\mathcal{T}^*\right]}{\mathrm{P}\left[\text{BIRTH}|\mathcal{T}^m\right]\mathrm{P}\left[n|\mathcal{T}^m\right]s_j}$$

where $\mathrm{P}\left[n|\mathcal{T}\right]$ is the probability of choosing node n given tree $\mathcal{T}$.

N.B. s_j appears in both the numerator and denominator of the acceptance probability π_{BIRTH}, therefore, canceling which is mathematically convenient.

Now, let's briefly discuss the posterior computation related to the Dirichlet sparse prior. If a Dirichlet prior is placed on the variable splitting probabilities, $\boldsymbol{s}$, then its posterior samples are drawn via Gibbs sampling with conjugate Dirichlet draws. The Dirichlet parameter is updated by adding the total variable branch count over the ensemble, m_j, to the prior setting, $\frac{\theta}{P}$, i.e., $\left[\frac{\theta}{P} + m_1, \ldots, \frac{\theta}{P} + m_P\right]$. In this way, the Dirichlet prior induces a "rich get richer" variable selection strategy. The sparsity parameter, θ, is drawn on a fine grid of values for the analytic posterior (Linero 2018). This draw only depends on $[s_1, \ldots, s_P]$.

D. Efficient computing with BART

If you had the task of creating an efficient implementation for a black-box model such as BART, which tools would you use? Surprisingly, linear algebra routines which are a traditional building block of scientific computing will be of little use for a tree-based method such as BART. So what is needed? Restricting ourselves to widely available off-the-shelf hardware and open-source software, we believe there are four key technologies necessary for a successful BART implementation.

- an object-oriented language to facilitate working with trees and matrices
- a parallel (or distributed) CPU computing framework for faster processing
- a high-quality parallel random number generator
- an interpreted shell for high-level data processing and analysis

In our implementation of BART, we pair the objected-oriented languages of R and C++ to satisfy these requirements. In this Section, we give a brief introduction to the concepts and technologies harnessed for efficient computing by our **BART** package.

D.1. A brief history of multi-threading

Writing multi-threaded programs is a fairly routine practice today with a high-level language like R and corresponding user-friendly interfaces such as the **parallel** R package (R Core Team 2018). Modern off-the-shelf laptops typically have 4 or 8 CPU cores placing reasonably priced multi-threaded hardware at your fingertips. Although, BART is often computationally undemanding, we find it very convenient, with the aid of multi-threading, to run in seconds that which would otherwise take minutes. To highlight the point that multi-threading is a mature technology, we now present a brief history of multi-threading. This is not meant to be exhaustive; rather, we only provide enough detail to explain the capability and popularity of multi-threading today.

Multi-threading emerged rather early in the digital computer age with pioneers laying the research groundwork in the 1960s. In 1961, Burroughs released the B5000 which was the first commercial hardware capable of multi-threading (Lynch 1965). The B5000 performed asymmetric multiprocessing which is commonly employed in modern hardware like numerical co-processors and/or graphical processors today. In 1962, Burroughs released the D825 which was the first commercial hardware capable of symmetric multiprocessing (SMP) with CPUs (Anderson, Hoffman, Shifman, and Williams 1962). In 1967, Gene Amdahl derived the theoretical limits for multi-threading which came to be known as Amdahl's law (Amdahl 1967).

If B is the number of CPUs and b is the fraction of work that can't be parallelized, then the gain due to multi-threading is $((1-b)/B+b)^{-1}$.

Now, fast-forward to the modern era of multi-threading. Hardware and software architectures in current use both directly, and indirectly, led to the wide availability of pervasive multi-threading today. In 2000, Advanced Micro Devices (AMD) released the AMD64 specification that created a new 64-bit x86 instruction set which was capable of co-existing with 16-bit and 32-bit x86 legacy instructions. This was an important advance since 64-bit math is capable of addressing vastly more memory than 16-bit or 32-bit (2^{64} vs. 2^{16} or 2^{32}) and multi-threading inherently requires more memory resources. In 2003, version 2.6 of the Linux kernel incorporated full SMP support; prior Linux kernels had either no support or very limited/crippled support. From 2005 to 2011, AMD released a series of Opteron chips with multiple cores for multi-threading: 2 cores in 2005, 4 cores in 2007, 6 cores in 2009, 12 cores in 2010 and 16 cores in 2011. From 2008 to 2010, Intel brought to market Xeon chips with their hyper-threading technology that allows each core to issue two instructions per clock cycle: 4 cores (8 threads) in 2008 and 8 cores (16 threads) in 2010. In today's era, most off-the-shelf hardware available features 1 to 4 CPUs each of which is capable of multi-threading. Therefore, in the span of only a few years, multi-threading rapidly trickled down from higher-end servers to mass-market products such as desktops and laptops. For example, the consumer laptop that **BART** is developed on, purchased in 2016, is capable of 8 threads (and hence many of the examples default to 8 threads).

D.2. Modern multi-threading software frameworks

Up to this point, we have introduced multi-threading with respect to parallelizing a task on a single system. Here we want to make a distinction between simple multi-threading on a single system and more complex multi-threading on multiple systems simultaneously which is often denoted by the term distributed computing. On a single system, various programming techniques can be used to create multi-threaded software. Basic multi-threading can be provided by the `fork` system call which is often termed forking. More advanced multi-threading is provided by software frameworks such as **OpenMP** and the Message Passing Interface (MPI). Please note that MPI can be employed for both simple multi-threading and for distributed computing, e.g., MPI software initially written for a single system could be extended to operate on multiple systems as computational needs expand. In the following, BART computations with multi-threading are explored where the term multi-threading is used for a single system and the term distributed computing is used for multiple systems.

In the late 1990s, MPI (Walker and Dongarra 1996) was introduced which is the dominant distributed computing framework in use today (Gabriel, Fagg, Bosilca, Angskun, Dongarra, Squyres, Sahay, Kambadur, Barrett, Lumsdaine, Castain, Daniel, Graham, and Woodall 2004). MPI support in R is built upon a fairly consistent interface provided by the **parallel** package (R Core Team 2018) which is extended by other CRAN packages such as **snow** (Tierney, Rossini, Li, and Sevcikova 2018) and **Rmpi** (Yu 2018). To support MPI, new BART software was created with a C++ object schema that is simple to program and maintain for distributed computing: we call this the MPI BART code-base (Pratola, Chipman, Gattiker, Higdon, McCulloch, and Rust 2014). The **BART** package source code is a descendent of MPI BART and its programmer-friendly objects, although, the multi-threading MPI support is now provided by R packages, e.g., **parallel**, **snow** and **Rmpi**.

The **BART** package supports multi-threading in two ways: 1) via **parallel** and related packages (which is how MPI is provided); and 2) via the OpenMP standard (Dagum and Menon 1998). OpenMP takes advantage of modern hardware by performing multi-threading on single machines which often have multiple CPUs each with multiple cores. Currently, the **BART** package only uses OpenMP for parallelizing `predict` function calculations. The challenge with OpenMP (besides the C/C++ programming required to support it) is that it is not available on all platforms. Operating system support can be detected by the GNU autotools (Calcote 2010) which define a C pre-processor macro called `_OPENMP` if it is available. There are numerous exceptions for operating systems so it is difficult to make universal statements. But, generally, Microsoft Windows lacks OpenMP detection since the GNU autotools do not natively exist on this platform. For Apple macOS, the standard Xcode toolkit does not provide OpenMP; however, the macOS compilers on CRAN do provide OpenMP (see `https://cran.r-project.org/bin/macosx/tools`). Most Linux and UNIX distributions provide OpenMP by default. We provide the function `mc.cores.openmp` which returns 1 if the `predict` function is capable of utilizing OpenMP; otherwise, returns 0.

The **parallel** package provides multi-threading via forking. Forking is available on Unix platforms, but not Windows (we use the term Unix to refer to UNIX, Linux and macOS since they are all in the UNIX family tree). The **BART** package uses forking for posterior sampling of the f function, and also for the `predict` function when OpenMP is not available. Except for `predict`, all functions that use forking start with `mc`. And, regardless of whether OpenMP or forking is employed, these functions accept the argument `mc.cores` which controls the number of threads to be used. The **parallel** package provides the function `detectCores` which returns the number of threads that your hardware can support and, therefore, the **BART** package can use.

D.3. BART implementations on CRAN

Currently, there are four BART implementations on the Comprehensive R Archive Network (CRAN); see the Appendix for a tabulated comparative summary of their features.

BayesTree was the first released in 2006 (Chipman and McCulloch 2016). Reported bugs will be fixed, but no future improvements are planned; so, we suggest choosing one of the newer packages such as **BART**. The basic interface and work-flow of **BayesTree** has strongly influenced the other packages which followed. However, the **BayesTree** source code is difficult to maintain and, therefore, improvements were limited leaving it with relatively fewer features than the other entries.

The second entrant is **bartMachine** which is written in Java and was first released in 2013 (Kapelner and Bleich 2018). It provides advanced features like multi-threading, variable selection (Bleich *et al.* 2014), a `predict` function, convergence diagnostics and missing data handling. However, the R to Java interface can be challenging to deal with. R is written in C and Fortran, consequentially, functions written in Java do not have a natural interface to R. This interface is provided by the **rJava** (Urbanek 2017) package which requires the Java Development Kit (JDK). Therefore, we highly recommend **bartMachine** for Java users.

The third entrant is **dbarts** which is written in C++ and was first released in 2014 (Dorie, Chipman, and McCulloch 2018). It is a clone of the **BayesTree** interface, but it does not share the source code; **dbarts** source has been re-written from scratch for efficiency and maintainability. **dbarts** is a drop-in replacement for **BayesTree**. Although, it lacks multi-

threading, the **dbarts** serial implementation is the fastest, therefore, it is preferable when multi-threading is unavailable such as on Windows.

The **BART** package which is written in C++ was first released in 2017 (McCulloch *et al.* 2019). It provides advanced features like multi-threading, variable selection (Linero 2018), a `predict` function and convergence diagnostics. The source code is a descendent of MPI BART. Although, R is mainly written in C and Fortran (at the time of this writing, 39.2% and 26.8% lines of source code respectively), C++ is a natural choice for creating R functions since they are both object-oriented languages. The C++ interface to R has been seamlessly provided by the **Rcpp** package (Eddelbuettel, François, Allaire, Ushey, Kou, Russel, Chambers, and Bates 2011) which efficiently passes object references from R to C++ (and vice versa) as well as providing direct access to the R random number generator. The source code can also be called from C++ alone without an R instance where the random number generation is provided by either the standalone Rmath library (R Core Team 2017) or the C++ `random` Standard Template Library. Furthermore, it is the only BART package to support categorical; and time-to-event outcomes (Sparapani *et al.* 2016, 2018, 2019). For one or more missing covariates, record-level hot-decking imputation (de Waal, Pannekoek, and Scholtus 2011) is employed that is biased towards the null, i.e., non-missing values from another record are randomly selected regardless of the outcome. This simple missing data imputation method is sufficient for data sets with relatively few missing values; for more advanced needs, we recommend the **sbart** package which utilizes the Sequential BART algorithm (Daniels and Singh 2018; Xu, Daniels, and Winterstein 2016) (N.B. **sbart** is also a descendent of MPI BART).

D.4. MCMC is embarrassingly parallel

In general, Bayesian Markov chain Monte Carlo (MCMC) posterior sampling is considered to be embarrassingly parallel (Rossini, Tierney, and Li 2007), i.e., since the chains only share the data and don't have to communicate with each other, parallel implementations are considered to be trivial. BART MCMC also falls into this class.

However, to clarify this point before proceeding, the embarrassingly parallel designation is in the context of simple multi-threading on single systems. An adaptation of distributed computing to large data sets exhaustively divides the data into mutually exclusive partitions, called shards, such that each system only processes a single shard. With sharded distributed computing, the embarrassingly parallel moniker does not apply. Recently, two advanced techniques have been developed for BART computations with sharding: Monte Carlo consensus (Pratola *et al.* 2014) and modified likelihood inflating sampling algorithm, or modified LISA, (Entezari, Craiu, and Rosenthal 2018). From here on, simple multi-threading is assumed.

Typical practice for Bayesian MCMC is to start in some initial state, perform a limited number of samples to generate a new random starting position and throw away the preceding samples which we call burn-in. The amount of burn-in in the **BART** package is controlled by the argument `nskip`: defaults to 100 with the exception of time-to-event outcomes which default to 250. The total length of the chain returned is controlled by the argument `ndpost` which defaults to 1000. The theoretical gain due to multi-threading can be calculated by what we call the MCMC Corollary to Amdahl's Law. Let b be the burn-in fraction and B be the number of threads, then the gain limit is $((1-b)/B+b)^{-1}$. (As an aside, note that we can derive Amdahl's Law as follows where the amount of work done is in the numerator and elapsed time is in the denominator: $\frac{1-b+b}{(1-b)/B+b} = \frac{1}{(1-b)/B+b}$). For example, see the diagram

Category	**BayesTree**	**bartMachine**	**dbarts**	**BART**
First release	2006	2013	2014	2017
Authors	Chipman & McCulloch	Kapelner & Bleich	Dorie, Chipman & McCulloch	McCulloch, Sparapani Gramacy, Spanbauer & Pratola
Source code	C++	Java	C++	C++
R package dependencies excluding Recommended	None	**rJava**, **car**, **randomForest**, **missForest**	None	**Rcpp**
Tree transition proposals	4	3	4	2
Multi-threaded	No	Yes	No	Yes
`predict` function	No	Yes	No	Yes
Variable selection	No	Yes	No	Yes
Continuous outcomes	Yes	Yes	Yes	Yes
Binary outcomes probit	Yes	Yes	Yes	Yes
Binary outcomes logit	No	No	No	Yes
Categorical outcomes	No	No	No	Yes
Time-to-event outcomes	No	No	No	Yes
Convergence diagnostics	No	Yes	No	Yes
Thinning	Yes	No	Yes	Yes
Missing data handling	No	Yes	No	Yes
Cross-validation	No	Yes	Yes	No
Partial dependence plots	Yes	Yes	Yes	No

Chipman and McCulloch (2016)
Kapelner and Bleich (2018)
Dorie *et al.* (2018)
McCulloch *et al.* (2019)

in Figure 20 where the burn-in fraction, $b = \frac{100}{1100} = 0.09$, and the number of CPUs, $B = 5$, results in an elapsed time of only $((1-b)/B + b) = 0.27$ or a $((1-b)/B + b)^{-1} = 3.67$ fold reduction which is the gain in efficiency. In Figure 21, we plot theoretical gains on the y-axis and the number of CPUs on the x-axis for two settings: $b \in \{0.025, 0.1\}$.

D.5. Multi-threading and random access memory

The IEEE standard 754-2008 (IEEE Computer Society 2008) specifies that every double-precision number consumes 8 bytes (64 bits). Therefore, it is quite simple to estimate the amount of random access memory (RAM) required to store a matrix. If A is $m \times n$, then the amount of RAM needed is $8 \times m \times n$ bytes. Large matrices held in RAM can present a challenge to system performance. If you consume all of the physical RAM, the system will "swap" segments out to virtual RAM which are disk files and this can degrade performance and possibly even crash the system. On Unix, you can monitor memory and swap usage with the `top` command-line utility. And, within R, you can determine the size of an object with the `object.size` function.

Mathematically, a matrix is represented as follows.

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

R is a column-major language, i.e., matrices are laid out in consecutive memory locations by traversing the columns: $[a_{11}, a_{21}, \ldots, a_{12}, a_{22}, \ldots]$. R is written in C and Fortran where Fortran is a column-major language as well. However, C and C++ are row-major languages, i.e., matrices are laid out in consecutive memory locations by traversing the rows: $[a_{11}, a_{12}, \ldots, a_{21}, a_{22}, \ldots]$. So, if you have written an R function in C/C++, then you need to be cognizant of the clash in paradigms (also note that R/Fortran array indexing goes from 1 to m while C/C++ indexing goes from 0 to $m-1$). As you might surmise, this is easily addressed with a transpose, i.e., instead of passing A from R to C/C++ pass $A^\top$.

R is very efficient in passing objects; rather, than passing an object (along with all of its memory consumption) on the stack, it passes objects merely by a pointer referencing the original memory location. However, R follows copy-on-write memory allocation, i.e., all objects present in the parent thread can be read by a child thread without a copy, but when an object is altered/written by the child, then a new copy is created in memory. Therefore, if we pass A from R to C/C++, and then transpose, we will create multiple copies of A consuming $8 \times m \times n \times B$ where B is the number of children. If A is a large matrix, then you may stress the system's limits. The simple solution is for the parent to create the transpose before passing A and avoiding the multiple copies, i.e., `A <- t(A)`. And this is the philosophy that the **BART** package follows for the multi-threaded BART functions; see the documentation for the `transposed` argument.

D.6. Multi-threading: interactive and batch processing

Interactive jobs must take precedence over batch jobs to prevent the user experience from suffering high latency. For example, have you ever experienced a system slowdown while you

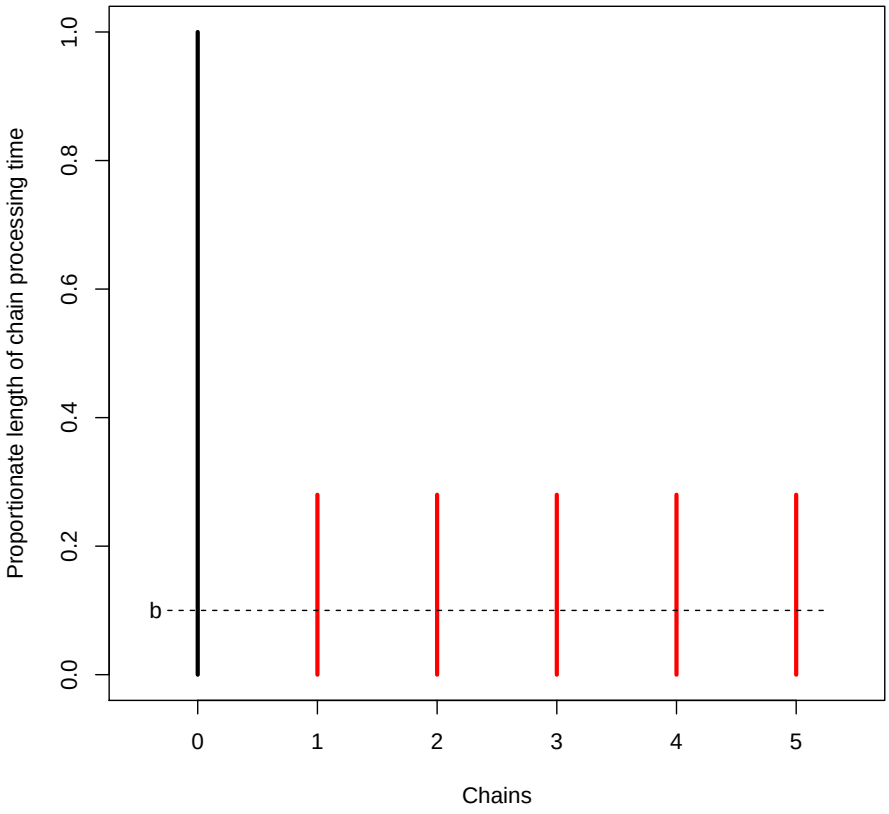


Figure 20: The theoretical gain due to multi-threading can be calculated by Amdahl's Law. Let b be the burn-in fraction and B be the number of threads, then the theoretical gain limit is $((1-b)/B+b)^{-1}$. In this diagram, the burn-in fraction, $b = \frac{100}{1100} = 0.09$, and the number of CPUs, $B = 5$, results in an elapsed time of only $((1-b)/B+b) = 0.27$ or a $((1-b)/B+b)^{-1} = 3.67$ fold reduction which is the gain in efficiency.

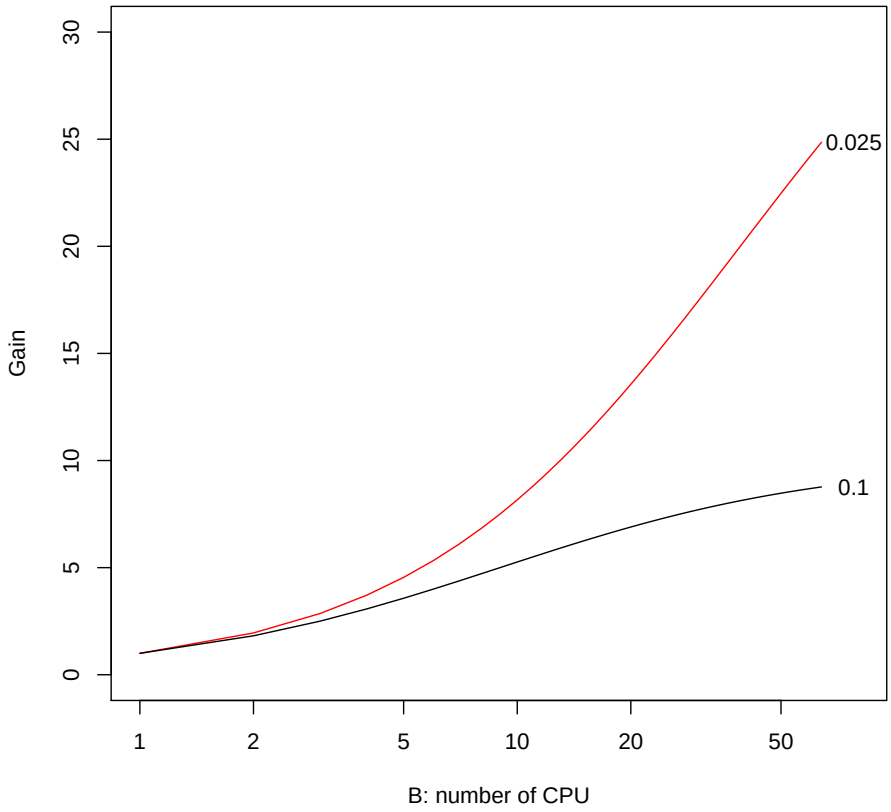


Figure 21: The theoretical gain due to multi-threading can be calculated by Amdahl's Law. Let b be the burn-in fraction and B be the number of threads, then the theoretical gain limit is $((1-b)/B+b)^{-1}$. In this figure, the theoretical gains are on the y-axis and the number of CPUs, the x-axis, for two settings: $b \in \{0.025, 0.1\}$.

are typing and the display of your keystrokes can not keep up; this should never happen and is the sign of something amiss. With large multi-threaded jobs, it is surprisingly easy to naively degrade system performance. But, this can easily be avoided by operating system support provided by R. In the **tools** package (R Core Team 2018), there is the `psnice` function. Paraphrased from the `?psnice` help page.

> Unix has a concept of process priority. Priority is assigned values from 0 to 39 with 20 being the normal priority and (counter-intuitively) larger numeric values denoting lower priority. Adding to the complexity, there is a "nice" value, the amount by which the priority exceeds 20. Processes with higher nice values will receive less CPU time than those with normal priority. Generally, processes with nice 19 are only run when the system would otherwise be idle.

Therefore, by default, the **BART** package children have their nice value set to 19.

D.7. Creating a BART executable

Occasionally, you may need to create a BART executable that you can run without an R instance. This is especially useful if you need to include BART in another C++ program. Or, when you need to debug the **BART** package C++ source code which is more difficult to do when you are calling the function from R. Several examples of these are provided with the **BART** package. With R, you can find the `Makefile` and the weighted BART example with `system.file("cxx-ex/Makefile", package="BART")` and `system.file("cxx-ex/wmain.cpp", package="BART")` respectively. Note that these examples require the installation of the standalone Rmath library (R Core Team 2017) which is contained in the R source code distribution. Rmath provides common R functions and random number generation, e.g., `pnorm` and `rnorm`. You will likely need to copy the `cxx-ex` directory to your workspace. Once done, you can build and run the weighted BART executable example from the command line shell as follows.

```
sh% make wmain.out ## to build
sh% ./wmain.out ## to run
```

By default, these examples are based on the Rmath random number generator. However, you can specify the C++ Standard Template Library random number generator (contained in the STL `random` header file) by uncommenting the following line in the `Makefile` (by removing the pound, #, symbols):

```
## CPPFLAGS = -I. -I/usr/local/include -DMATHLIB_STANDALONE -DRNG_random
```

(which still requires Rmath for other purposes). These examples were developed on Linux and macOS, but they should be readily adaptable to UNIX and Windows as well.

References

Agresti A (2003). *Categorical Data Analysis*. John Wiley & Sons, Hoboken, NJ.

Albert J, Chib S (1993). "Bayesian Analysis of Binary and Polychotomous Response Data." *Journal of the American Statistical Association*, **88**, 669–79.

Amdahl G (1967). "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities." In *AFIPS Conference Proceedings*, volume 30, pp. 483–5.

Andersen PK, Gill RD (1982). "Cox's regression model for counting processes: a large sample study." *The annals of statistics*, pp. 1100–1120.

Anderson JP, Hoffman SA, Shifman J, Williams RJ (1962). "D825 - A Multiple-Computer System for Command and Control." In *AFIPS Conference Proceedings*, volume 24.

Arjas E, Haara P (1987). "A Logistic Regression Model for Hazard: Asymptotic Results." *Scandinavian Journal of Statistics*, **14**, 1–18.

Baldi P, Brunak S (2001). *Bioinformatics: The Machine Learning Approach*. MIT Press, Cambridge, MA.

Bleich J, Kapelner A, George EI, Jensen ST (2014). "Variable Selection for BART: An Application to Gene Regulation." *The Annals of Applied Statistics*, **8**(3), 1750–1781.

Breiman L (1996). "Bagging Predictors." *Machine Learning*, **24**(2), 123–140.

Breiman L (2001). "Random Forests." *Machine Learning*, **45**(1), 5–32.

Byar D (1980). "The Veterans Administration Study of Chemoprophylaxis for Recurrent Stage I Bladder Tumours: Comparisons of Placebo, Pyridoxine and Topical Thiotepa." In *Bladder Tumors and Other Topics in Urological Oncology*, pp. 363–370. Springer-Verlag, New York, NY.

Calcote J (2010). *Autotools: A Practitioner's Guide to GNU Autoconf, Automake, and Libtool*. No Starch Press.

Chipman H, McCulloch R (2016). ***BayesTree**: Bayesian Additive Regression Trees*. R package version 0.3-1.4, URL https://CRAN.R-project.org/package=BayesTree.

Chipman HA, George EI, McCulloch RE (1998). "Bayesian CART Model Search." *Journal of the American Statistical Association*, **93**(443), 935–948.

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

Chipman HA, George EI, McCulloch RE (2013). "Bayesian Regression Structure Discovery." *Bayesian Theory and Applications,(Eds, P. Damien, P. Dellaportas, N. Polson, D. Stephens), Oxford University Press, Oxford, UK*.

Cox DR (1972). "Regression Models and Life Tables (with Discussion)." *Journal of the Royal Statistical Society, Series B*, **34**, 187–220.

Dagum L, Menon R (1998). "OpenMP: An Industry Standard API for Shared-Memory Programming." *IEEE Computational Science and Engineering*, **5**(1), 46–55.

Daniels M, Singh A (2018). ***sbart**: Sequential BART for Imputation of Missing Covariates*. R package version 0.1.1, URL https://CRAN.R-project.org/package=sbart.

de Waal T, Pannekoek J, Scholtus S (2011). *Handbook of Statistical Data Editing and Imputation*. John Wiley & Sons, Hoboken, NJ.

Delany MF, Linda SB, Moore CT (1999). "Diet and Condition of American Alligators in 4 Florida Lakes." In *Proceedings of the Annual Conference of the Southeastern Association of Fish and Wildlife Agencies*, volume 53, pp. 375–389.

Denison DG, Mallick BK, Smith AF (1998). "A Bayesian CART Algorithm." *Biometrika*, **85**(2), 363–377.

Devroye L (1986). *Non-Uniform Random Variate Generation*. Springer-Verlag, New York, NY.

Dorie V, Chipman H, McCulloch R (2018). ***dbarts**: Discrete Bayesian Additive Regression Trees Sampler*. R package version 0.9-8, URL https://CRAN.R-project.org/package=dbarts.

Eddelbuettel D, François R, Allaire J, Ushey K, Kou Q, Russel N, Chambers J, Bates D (2011). "**Rcpp**: Seamless R and C++ Integration." *Journal of Statistical Software*, **40**(8), 1–18.

Efron B, Hastie T, Johnstone I, Tibshirani R (2004). "Least Angle Regression." *The Annals of Statistics*, **32**(2), 407–499.

Entezari R, Craiu RV, Rosenthal JS (2018). "Likelihood inflating sampling algorithm." *Canadian Journal of Statistics*, **46**(1), 147–175.

Fahrmeir L (2014). "Discrete Survival-Time Models." *Wiley StatsRef: Statistics Reference Online*. [https://doi.org/10.1002/9781118445112.stat06012].

Fine JP, Gray RJ (1999). "A Proportional Hazards Model for the Subdistribution of a Competing Risk." *Journal of the American Statistical Association*, **94**, 496–509.

Freund Y, Schapire RE (1997). "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting." *Journal of Computer and System Sciences*, **55**(1), 119–139.

Friedman JH (1991). "Multivariate Adaptive Regression Splines (with Discussion and a Rejoinder by the Author)." *The Annals of Statistics*, **19**, 1–67.

Friedman JH (2001). "Greedy Function Approximation: A Gradient Boosting Machine." *The Annals of Statistics*, **29**, 1189–1232.

Frühwirth-Schnatter S, Frühwirth R (2010). "Data Augmentation and MCMC for Binary and Multinomial Logit Models." In *Statistical Modelling and Regression Structures*, pp. 111–132. Springer-Verlag, New York, NY.

Gabriel E, Fagg GE, Bosilca G, Angskun T, Dongarra JJ, Squyres JM, Sahay V, Kambadur P, Barrett B, Lumsdaine A, Castain R, Daniel D, Graham R, Woodall T (2004). "Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation." In *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*, pp. 97–104. Springer-Verlag, New York, NY.

Gelfand AE, Smith AF (1990). "Sampling-Based Approaches to Calculating Marginal Densities." *Journal of the American Statistical Association*, **85**(410), 398–409.

Geman S, Geman D (1984). "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **6**, 721–741.

Geweke J (1992). *Bayesian Statistics*, chapter Evaluating the Accuracy of Sampling-Based Approaches to Calculating Posterior Moments. fourth edition. Clarendon Press, Oxford, UK.

Gönen M, Heller G (2005). "Concordance Probability and Discriminatory Power in Proportional Hazards Regression." *Biometrika*, **92**(4), 965–970.

Gramacy RB, Polson NG (2012). "Simulation-Based Regularized Logistic Regression." *Bayesian Analysis*, **7**(3), 567–590.

Hahn P, Carvalho C (2015). "Decoupling Shrinkage and Selection in Bayesian Linear Models: a Posterior Summary Perspective." *Journal of the American Statistical Association*, **110**, 435–48.

Harrison Jr D, Rubinfeld DL (1978). "Hedonic Housing Prices and the Demand for Clean Air." *Journal of Environmental Economics and Management*, **5**(1), 81–102.

Hastings W (1970). "Monte Carlo Sampling Methods Using Markov Chains and Their Applications." *Biometrika*, **57**, 97–109.

Holmes C, Held L (2006). "Bayesian Auxiliary Variable Models for Binary and Multinomial Regression." *Bayesian Analysis*, **1**, 145–68.

IEEE Computer Society (2008). IEEE Std 754-2008.

Imai K, Van Dyk DA (2005). "A Bayesian Analysis of the Multinomial Probit Model Using Marginal Data Augmentation." *Journal of Econometrics*, **124**(2), 311–334.

Ishwaran H, Gerds TA, Kogalur UB, Moore RD, Gange SJ, Lau BM (2014). "Random survival forests for competing risks." *Biostatistics*, **15**(4), 757–773.

Johnson NL, Kotz S, Balakrishnan N (1995). *Continuous univariate distributions*, volume 2. second edition. Houghton Mifflin, Boston.

Kalbfleisch J, Prentice R (1980). *The Statistical Analysis of Failure Time Data*. first edition. John Wiley & Sons, Hoboken, NJ.

Kalbfleisch J, Prentice R (2002). *The Statistical Analysis of Failure Time Data*. second edition. John Wiley & Sons, Hoboken, NJ.

Kapelner A, Bleich J (2018). ***bartMachine**: Bayesian Additive Regression Trees*. R package version 1.2.4.2, URL https://CRAN.R-project.org/package=bartMachine.

Kindo BP, Wang H, Peña EA (2016). "Multinomial Probit Bayesian Additive Regression Trees." *Stat*, **5**(1), 119–131.

Klein JP, Moeschberger ML (2006). *Survival analysis: techniques for censored and truncated data*. second edition. Springer-Verlag, New York, NY.

Krogh A, Solich P (1997). "Statistical Mechanics of Ensemble Learning." *Physical Review E*, **55**, 811–25.

Kuhn M, Johnson K (2013). *Applied Predictive Modeling*. Springer-Verlag, New York, NY.

Linero A (2018). "Bayesian Regression Trees for High Dimensional Prediction and Variable Selection." *Journal of the American Statistical Association*, **113**(522), 626–36.

Loprinzi CL, Laurie JA, Wieand HS, Krook JE, Novotny PJ, Kugler JW, Bartel J, Law M, Bateman M, Klatt NE (1994). "Prospective Evaluation of Prognostic Variables from Patient-Completed Questionnaires. North Central Cancer Treatment Group." *Journal of Clinical Oncology*, **12**(3), 601–607.

Lynch J (1965). "The Burroughs B8500." *Datamation*, pp. 49–50.

McCulloch R, Carvalho C, Hahn R (2015). "A General Approach to Variable Selection Using Bayesian Nonparametric Models." Joint Statistical Meetings, Seattle, 08/09/15-08/13/15.

McCulloch R, Rossi PE (1994). "An Exact Likelihood Analysis of the Multinomial Probit Model." *Journal of Econometrics*, **64**(1-2), 207–240.

McCulloch R, Sparapani R, Gramacy R, Spanbauer C, Pratola M (2019). ***BART**: Bayesian Additive Regression Trees*. R package version 2.6, URL https://CRAN.R-project.org/package=BART.

McCulloch RE, Polson NG, Rossi PE (2000). "A Bayesian Analysis of the Multinomial Probit Model with Fully Identified Parameters." *Journal of Econometrics*, **99**(1), 173–193.

Metropolis N, Rosenbluth AW, Rosenbluth MN, Teller AH, Teller E (1953). "Equation of State Calculations by Fast Computing Machines." *The Journal of Chemical Physics*, **21**(6), 1087–1092.

Mueller P (1991). "A Generic Approach to Posterior Integration and Gibbs Sampling." *Technical Report 91-09*, Purdue University, West Lafayette, Indiana. [http://www.stat.purdue.edu/research/technical_reports/pdfs/1991/tr91-09.pdf].

Mueller P (1993). "Alternatives to the Gibbs Sampling Scheme." *Technical report*, Institute of Statistics and Decision Sciences, Duke University, Durham, North Carolina. [http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.48.5613&rep=rep1&type=pdf].

Murray JS (2017). "Log-Linear Bayesian Additive Regression Trees for Categorical and Count Responses." *arXiv preprint arXiv:1701.01503*.

Nicolaie M, van Houwelingen HC, Putter H (2010). "Vertical Modeling: A Pattern Mixture Approach for Competing Risks Modeling." *Statistics in Medicine*, **29**(11), 1190–1205.

Plummer M, Best N, Cowles K, Vines K (2006). "CODA: Convergence Diagnosis and Output Analysis for MCMC." *R News*, **6**(1), 7–11. [https://journal.r-project.org/archive].

Pratola MT (2016). "Efficient Metropolis–Hastings Proposal Mechanisms for Bayesian Regression Tree Models." *Bayesian Analysis*, **11**(3), 885–911.

Pratola MT, Chipman HA, Gattiker JR, Higdon DM, McCulloch R, Rust WN (2014). "Parallel Bayesian Additive Regression Trees." *Journal of Computational and Graphical Statistics*, **23**(3), 830–52.

R Core Team (2017). ***Mathlib**: A C Library of Special Functions*. R Foundation for Statistical Computing, Vienna, Austria. URL https://CRAN.R-project.org/doc/manuals/r-release/R-admin.html#The-standalone-Rmath-library.

R Core Team (2018). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org.

Ripley BD (2007). *Pattern Recognition and Neural Networks*. Cambridge University press.

Robert CP (1995). "Simulation of Truncated Normal Variables." *Statistics and Computing*, **5**(2), 121–125.

Rossini AJ, Tierney L, Li N (2007). "Simple Parallel Statistical Computing in R." *Journal of Computational and Graphical Statistics*, **16**(2), 399–420.

Scott SL (2011). "Data Augmentation, Frequentist Estimation, and the Bayesian Analysis of Multinomial Logit Models." *Statistical Papers*, **52**(1), 87–109.

Silverman B (1986). *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, London.

Sparapani R, Logan BR, McCulloch RE, Laud PW (2019). "Nonparametric Competing Risks Analysis Using Bayesian Additive Regression Trees (BART)." *Statistical Methods in Medical Research*, **(in press)**. [https://doi.org/10.1177/0962280218822140].

Sparapani R, Rein L, Tarima S, Jackson T, Meurer J (2018). "Nonparametric Recurrent Events Analysis with BART and an Application to the Hospital Admissions of Patients with Diabetes." *Biostatistics*, **(in press)**. [https://doi.org/10.1093/biostatistics/kxy032].

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.

Sparapani RA, Logan BR, McCulloch RE, Laud PW (2016). "Nonparametric Survival Analysis Using Bayesian Additive Regression Trees (BART)." *Statistics in Medicine*, **35**(16), 2741–53.

Thompson Jr W (1977). "On the Treatment of Grouped Observations in Life Studies." *Biometrics*, **33**, 463–70.

Tierney L, Rossini A, Li N, Sevcikova H (2018). ***snow**: Simple Network of Workstations*. R package version 0.4-3, URL https://CRAN.R-project.org/package=snow.

Urbanek S (2017). ***rJava**: Low-Level R to Java Interface*. R package version 0.9-10, URL https://CRAN.R-project.org/package=rJava.

Venables WN, Ripley BD (2013). *Modern Applied Statistics with S-PLUS*. Springer-Verlag, New York, NY. **nnet** R package version 7.3-12, URL https://CRAN.R-project.org/package=nnet.

Walker DW, Dongarra JJ (1996). "MPI: A Standard Message Passing Interface." *Supercomputer*, **12**, 56–68.

Wei LJ, Lin DY, Weissfeld L (1989). "Regression analysis of multivariate incomplete failure time data by modeling marginal distributions." *Journal of the American Statistical Association*, **84**(408), 1065–1073.

Wu Y, Tjelmeland H, West M (2007). "Bayesian CART: Prior Specification and Posterior Simulation." *Journal of Computational and Graphical Statistics*, **16**(1), 44–66.

Xu D, Daniels MJ, Winterstein AG (2016). "Sequential BART for Imputation of Missing Covariates." *Biostatistics*, **17**(3), 589–602.

Yu H (2018). ***Rmpi**: Interface (Wrapper) to MPI (Message-Passing Interface)*. R package version 0.6-9, URL https://CRAN.R-project.org/package=Rmpi.

Affiliation:

Rodney Sparapani rsparapa@mcw.edu
Division of Biostatistics
Institute for Health and Equity
Medical College of Wisconsin, Milwaukee campus
8701 Watertown Plank Road
Milwaukee, WI 53226, USA