SoftBart: Soft Bayesian Additive Regression Trees

Antonio R. Linero*

2022-10-28

Abstract

Bayesian additive regression tree (BART) models have seen increased attention in recent years as a general-purpose nonparametric modeling technique. BART combines the flexibility of modern machine learning techniques with the principled uncertainty quantification of Bayesian inference, and it has been shown to be uniquely appropriate for addressing the high-noise problems that occur commonly in many areas of science, including medicine and the social sciences. This paper introduces the `SoftBart` package for fitting the *Soft BART* algorithm of Linero and Yang (2018). In addition to improving upon the predictive performance of other BART packages, a major goal of this package has been to facilitate the inclusion of BART in larger models, making it ideal for researchers in Bayesian statistics. I show both how to use this package for standard prediction tasks and how to embed BART models in larger models; I illustrate by using `SoftBart` to implement a nonparametric probit regression model, a semiparametric varying coefficient model, and a partial linear model.

1 Introduction

Introduced by Chipman et al. (2010), Bayesian additive regression tree (or BART) models have attracted substantial interest from the Bayesian nonparametrics and machine learning communities. BART is used in nonparametric function estimation problems, where the function of interest $r(x)$ is modeled as a *decision tree ensemble* (containing T trees) of the form

$$r(x) = \sum_{t=1}^{T} \text{Tree}(x; \mathcal{T}_t, \mathcal{M}_t) \qquad \text{where} \qquad (\mathcal{T}_t, \mathcal{M}_t) \overset{\text{iid}}{\sim} \pi_{\mathcal{T}}(\mathcal{T}_t)\, \pi_{\mathcal{M}}(\mathcal{M}_t \mid \mathcal{T}_t),$$

and where $(\pi_{\mathcal{T}}, \pi_{\mathcal{M}})$ defines a prior for the the parameters of the decision trees. Each of the functions $\text{Tree}(x; \mathcal{T}_t, \mathcal{M}_t)$ defines a *regression tree* (see Figure 2). This model can be viewed as a Bayesian version of the famous *decision tree boosting* framework (Freund et al., 1999; Friedman, 2001), with the T decision trees in the ensemble representing "weak learners" that are then aggregated into a single "strong learner" for $r(x)$. The canonical problem where BART is applied is the *semiparametric Gaussian regression* problem

$$Y_i = r(X_i) + \epsilon_i \qquad \text{where} \qquad \epsilon_i \sim \text{Normal}(0, \sigma^2),$$

but it can also be used to model nonparametric functions $r(x)$ in essentially arbitrary problems; these problems include nonparametric probit and logit regression (Chipman et al., 2010; Murray, 2021), survival analysis (Sparapani et al., 2016; Linero et al., 2021; Basak et al., 2021; Henderson et al., 2020; Bonato et al., 2010), density regression (Li et al., 2022; Orlandi et al., 2021; Um et al., 2022), and estimation of inhomogeneous Poisson processes (Lamprinakou et al., 2020). Moreover, Linero (2022) shows how BART can be used with arbitrary probabilistic models using reversible jump Markov chain Monte Carlo, allowing it to be applied in the same settings that one can apply decision tree boosting.

In this way, BART can be viewed as a flexible drop-in replacement for other Bayesian nonparametric and/or tree-based function estimation approaches. In my view, the most important features of the BART framework are the following:

*University of Texas at Austin, antonio.linero@austin.utexas.edu

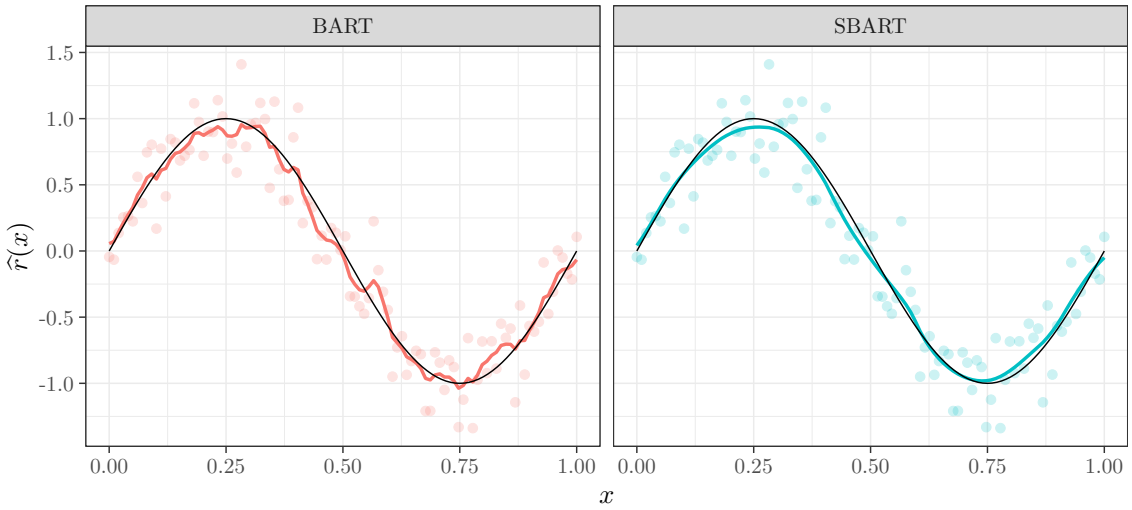

Figure 1: Comparison of the fit of the BART and `SoftBart` models to data generated from the relationship $Y_i = \sin(2\pi x) + \epsilon_i$ with $\epsilon_i \sim \text{Normal}(0, 0.1^2)$. The sine curve is overlayed in black.

1. Among Bayesian nonparametric approaches, BART is unique in that there are several high-quality, easy to use, software implementations, including the `BayesTree` (Chipman and McCulloch, 2016), `bartMachine` (Kapelner and Bleich, 2016), `dbarts` (Dorie, 2022), and `BART` (Sparapani et al., 2021) packages. These mainly focus on the problems of semiparametric Gaussian and nonparametric binary regression. Because of the existence of these packages (and the common practice in the BART community of proposing default priors) less expertise is required to use BART than other Bayesian nonparametric models such as Gaussian processes (Rasmussen and Williams, 2006) or Dirichlet process mixtures (Escobar and West, 1995).

2. BART has been shown to perform extremely well in the high-noise settings that are typical in the social sciences and medicine; for example Linero and Yang (2018) showed that BART routinely outperformed both random forests and decision tree boosting on average over many datasets. Because of this, BART has seen wide deployment in the causal inference literature (Hahn et al., 2020; Hill, 2011; Linero and Zhang, 2022), where it is used both for Bayesian estimation of heterogeneous causal effects and as a black-box machine learning algorithm by Frequentists.

3. Unlike most other black-box machine learning methods (e.g., neural networks, Gaussian processes, and random forests), the BART prior *shrinks towards low-order interactions* in the data — because the decision trees used by BART are usually shallow, realizations of $r(x)$ from the prior will tend to prioritize main effects, with a smaller number of second-order interactions, and even fewer high-order interactions. While complex high-order interactions are common in some fields (image recognition, natural language processing, and so forth), they are not thought to be very important in traditional areas of statistics. That BART emphasizes low-order interactions was part of the initial motivation of Chipman et al. (2010), as that BART is optimal in this regime was established theoretically in a series of recent papers (Ročková and van der Pas, 2020; Linero and Yang, 2018; Saha, 2021).

To balance these positives, BART has several shortcomings. One shortcoming of BART, and of decision tree ensembling approaches in general, is that the predictions produced by these models are generally non-smooth; for example, realizations from the BART prior are stepwise-continuous functions. The impact of this lack of smoothness can be seen in Figure 1, where BART performs suboptimally in estimating the univariate function $r(x) = \sin(2\pi x)$. In this case, the mean-squared error of the BART model is 140% larger than the mean-squared error of the `SoftBart` model we propose here.

To address the lack of smoothness of BART, Linero and Yang (2018) introduced the `SoftBart` model, with the authors demonstrating both theoretically (through studies of posterior concentration rates) and practically (through the analysis of benchmark datasets) that leveraging smoothness often results in substantially

improved prediction on real datasets. Since its introduction, the `SoftBart` model has been used by many researchers; in addition to extensions proposed by its progenitors, it has seen substantial interest in both the Bayesian nonparametrics and the Bayesian causal inference literature. A non-comprehensive list of applications include: Liu et al. (2021), who use `SoftBart` as the algorithm of choice for addressing non-response in surveys; Ran and Bai (2021), who constructed a MAP-reduce algorithm for fitting `SoftBart` to massive datasets; Bai et al. (2022), who found `SoftBart` to be a very high-quality competitor to their spike-and-slab group lasso GAM model; and Hahn et al. (2020), where the method was discussed both in the main manuscript and by multiple discussants.

1.1 Our Motivation

In this paper, we introduce the `SoftBart` package, and show how to apply it to a number of nonparametric estimation problems. Given the large number of high-quality packages for implementing BART, as well as the large number of competing nonparametric techniques (Bayesian or otherwise), it is natural to wonder at the value added by yet another package. There are two major goals of this package:

1. this package makes accessible the `SoftBart` methodology of Linero and Yang (2018) to a wider audience; the value of this is apparent from the analysis of benchmark datasets given by Linero and Yang (2018), as well as the fact that `SoftBart` has been consistently observed to outperform other variants of BART by other authors (Prado et al., 2021).
2. this package includes functionality that makes it easy to embed BART (or `SoftBart`) into larger `R` programs; this will aide researchers interested in BART by making it easy to include BART in custom models.

The second objective is a novel contribution of this package, as to the best of my knowledge `SoftBart` is the only BART package that allows users to embed BART within a larger model without having to modify the underlying `C++` code. To this point, work extending BART has mainly been carried out by, as most statisticians do not have the requisite programming knowledge (such as familiarity with `C++` or other compiled languages, knowledge of tree-based data structures, and experience implementing Markov chain Monte Carlo algorithms on discrete data structures) to modify existing BART code effectively. Part of the motivation for developing this functionality comes, in fact, from the realities of working on projects with graduate students; for many problems I have worked on, the modifications required were conceptually simple, but nevertheless required the student to learn a non-trivial amount of `C++` to implement. Making extending BART more convenient for graduate students was therefore an important aim for this package.

I show through several illustrations how `SoftBart` makes it easy for non-experts to implement extensions of BART; all that is needed is a conceptual understanding of Gibbs sampling and experience implementing MCMC algorithms in `R`. By embedding the BART updates inside of simple Gibbs sampling algorithms, I show how to implement the following models: a nonparametric probit regression BART model described by Chipman et al. (2010); the varying coefficient model of Deshpande et al. (2020), which I show also contains the Bayesian causal forests model of Hahn et al. (2020) as a special case; and the general BART model of Tan and Roy (2019). Beyond these models, I also note here that several existing works have used a preliminary version of this software to implement their proposed methodology, including the survival analysis model of Basak et al. (2021) and the skewed error model of Um et al. (2022).

1.2 Description of Methodology

We begin by reviewing the original BART model of Chipman et al. (2010) before describing the `SoftBart` model. The BART framework models an unknown function $r(x)$ as a *sum of decision trees*

$$r(x) = \sum_{t=1}^{T} \text{Tree}(x; \mathcal{T}_t, \mathcal{M}_t) \tag{1}$$

where $\mathcal{T}_t$ denotes a *decision tree*, $\mathcal{M}_t$ denotes a collection of *leaf node parameters*, and $\text{Tree}(x; \mathcal{T}, \mathcal{M})$ is a *regression tree function* that returns the prediction associated to x for the pair $(\mathcal{T}, \mathcal{M})$. I illustrate this

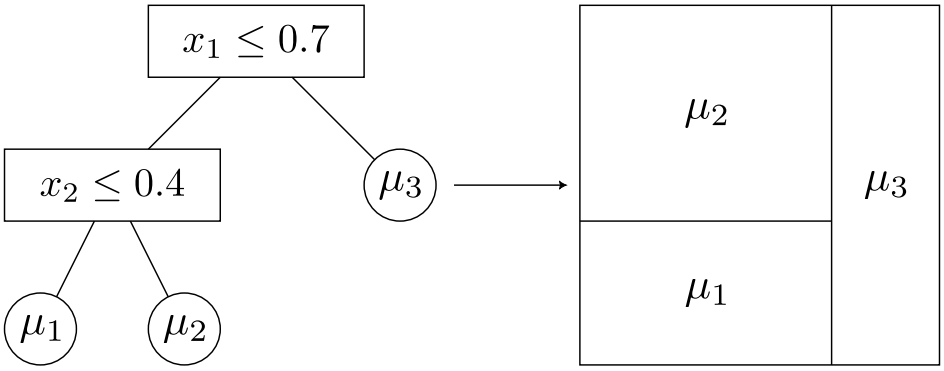

Figure 2: Schematic depiction of a decision tree; the left figure gives the decision tree $\mathcal{T}$ and its leaf node values $\mathcal{M} = (\mu_1, \mu_2, \mu_3)$, while the right figure gives the induced step function on $[0,1]^2$. A given value of x starts at the top of the tree and goes left if the rule at each node is true and goes right if the rule is false, until it reaches a terminal node.

in Figure 2, which gives a schematic of a decision tree $\mathcal{T}$ with leaf node parameters $\mathcal{M} = \{\mu_1, \mu_2, \mu_3\}$; for example, if $x = (0.3, 0.6)^\top$ then $\text{Tree}(x; \mathcal{T}, \mathcal{M}) = \mu_2$ because $[0.3 \leq 0.7]$ and $[0.6 > 0.4]$. Figure 2 also shows that each regression tree $\text{Tree}(x; \mathcal{T}, \mathcal{M})$ corresponds to a piecewise constant function.

As seen in Figure 2, each tree $\mathcal{T}$ consists of a collection of *leaf nodes* (the nodes with no "child" nodes) and a collection of *branch nodes* that each have an associated *splitting rule* of the form $[x_{j_b} \leq C_b]$. We write $\mathcal{L}(\mathcal{T})$ for the collection of leaf nodes ℓ and write $\mathcal{B}(\mathcal{T})$ for the collection of branch nodes b.

1.3 The Prior on Decision Trees

Unlike other decision tree ensembling strategies, such as random forests or boosting (Breiman, 2001; Freund et al., 1999), BART proceeds by placing a prior on the regression trees. Given the model hyperparameters $\vartheta = (s, \beta, \gamma, T, \sigma_\mu^2)$ (to be described later), BART takes the regression trees to be a-priori independent, i.e.,

$$\pi\big((\mathcal{T}_1, \mathcal{M}_1), \ldots, (\mathcal{T}_T, \mathcal{M}_T) \mid \vartheta\big) = \prod_{t=1}^{T} \pi_{\mathcal{T}}(\mathcal{T}_t \mid \vartheta)\, \pi_{\mathcal{M}}(\mathcal{M}_t \mid \mathcal{T}_t).$$

In `SoftBart`, the prior distribution for the trees $\pi_{\mathcal{T}}$ consists of two components:

1. a prior on the shape of the tree $\mathcal{T}$; and
2. a prior on the splitting rules $[x_{j_b} \leq C_b]$ for each branch node of the tree.

The prior on the shape of $\mathcal{T}$ is a *branching process* described by Chipman et al. (2010). We start with a tree consisting only of a root node of depth $d = 0$; we then make this root a branch node with two children with probability

$$\Pr(\text{is a branch}) = \frac{\gamma}{(1+d)^\beta} \tag{2}$$

otherwise the root becomes a leaf node. This process then iterates for $d = 1, 2, \ldots$ until all nodes at a given depth are leaf nodes.

`SoftBart` uses a prior for the splitting rules that first selects a predictor by sampling $j_b \sim \text{Categorical}(s)$ where $s = (s_1, \ldots, s_P)^\top$ is a probability vector. The prior then selects the cutpoint C_b by sampling $C_b \sim \text{Uniform}(A_{j_b}, B_{j_b})$ where $\prod_{j=1}^{P}[A_j, B_j]$ is the hyperrectangle of x values that are associated to branch b.

The hyperparameters $(s, \gamma, \beta, \sigma_\mu, T)$ are, by default, either fixed or given weakly-informative hyperpriors, but users can specify their own values/priors for these quantities if desired. We defer discussion of the prior on s to Section 3, as s plays an important role in variable selection. `SoftBart` follows the convention of Chipman

et al. (2010) by taking $\gamma = 0.95$ and $\beta = 2$ by default, and opts for using fewer trees ($T = 20$) by default than other BART packages. Finally, `SoftBart` uses the a half-Cauchy prior $\sigma_\mu \sim \text{Cauchy}_+(\widehat{\sigma}_\mu)$ for σ_μ, where $\widehat{\sigma}_\mu = 0.5/(k\sqrt{T})$ and $k = 2$. This is different than other BART packages in that we use a hyperprior for σ_μ, but the prior is centered at the default value of σ_μ recommended for semiparametric Gaussian regression by Chipman et al. (2010) (after scaling the outcome Y_i to lie in the interval $[-0.5, 0.5]$). For BART models other than the semiparametric Gaussian regression model, this choice of σ_μ might not be appropriate, and we use (for example) the default $\widehat{\sigma}_\mu = 3/\sqrt{T}$ for the probit regression model discussed in Section 4.1.

The prior used in `SoftBart` differs in several minor ways from the prior described by Chipman et al. (2010). First, we use continuous uniform cutpoints rather than taking the cutpoints to occur only at the observed values of the X_{ij}'s. Second, we do not place any restrictions on the number of X_i's required for a node to be made a branch; the high level motivation for such restrictions is to reduce the risk of overfitting, but I have found it to be redundant in practice given that the prior regularizes the predictions at the leaf nodes. Ultimately, I have found that these differences make very little difference in practice in terms of predictive performance, and my choice of prior is driven by other concerns (in particular, it allows for a simple conditionally conjugate prior for s).

1.4 Smoothing Decision Trees

The `SoftBart` model modifies the sum of decision trees $r(x)$ by replacing the regression trees $\text{Tree}(x; \mathcal{T}_t, \mathcal{M}_t)$ with *soft* regression trees (Irsoy et al., 2012). To generalize a regression tree to a soft regression tree, we begin by noting that we can rewrite

$$\text{Tree}(x; \mathcal{T}, \mathcal{M}) = \sum_{\ell \in \mathcal{L}(\mathcal{T})} \mu_\ell \, \phi_\ell(x; \mathcal{T})$$

where $\phi_\ell(x; \mathcal{T})$ is the indicator function of the event that x is associated to leaf ℓ of tree $\mathcal{T}$. Notice that we can rewrite $\phi_\ell(x; \mathcal{T})$ in terms of the branch rules as

$$\phi_\ell(x; \mathcal{T}) = \prod_{b \in A(\ell)} I(x_{j_b} \leq C_b)^{L_b} \, I(x_{j_b} > C_b)^{1 - L_b}, \tag{3}$$

where $A(\ell)$ is the set of leaf nodes that are *ancestral* to leaf node ℓ and $L_b = 1$ if the path from the root to ℓ goes left at b and $L_b = 0$ if the path goes right; for example, the leaf with μ_3 in Figure 2 has $A(\ell)$ consisting only of the root and has $L_{\text{root}} = 0$ since the path from the root goes right rather than left. The indicator functions in (3) are not ideal, as they encode a sharp jump from 0 to 1 as we increase x_b from $x_b \leq C_b$ to $x_b > C_b$. Linero and Yang (2018) generalize (3) by replacing the "hard" decision rules $I(x_j \leq C)$ with *soft decision rules* $\psi\left(\frac{x_j - C}{\tau}\right)$, where $\psi(x)$ is the cumulative distribution function of a symmetric random variable. The modified weights become

$$\phi_\ell(x; \mathcal{T}) = \prod_{b \in A(\ell)} \psi\left(\frac{x_{j_b} - C_b}{\tau}\right)^{L_b} \left\{1 - \psi\left(\frac{x_{j_b} - C_b}{\tau}\right)\right\}^{1 - L_b}. \tag{4}$$

Because the function $\psi(x)$ is continuous, the soft decision tree is continuous in x. The parameter τ controls how "sharp" the decisions are, with the limit $\tau \to 0$ corresponding to a standard decision tree. Figure 3 compares a hard decision tree to a soft decision tree in terms of the associated functions; we see that the soft decision tree smooths over the decision boundaries of the hard decision tree. `SoftBart` uses the logistic function $\psi(x) = (1 + \exp(-x))^{-1}$ and gives each tree $\mathcal{T}_t$ its own bandwidth parameter τ_t with $\tau_t \sim \text{Exponential}(\texttt{scale} = 0.1)$.

1.5 Scaling the Outcome and Covariates

In order to ensure that the default prior used by `SoftBart` is widely appropriate, most functions in the package either work with, or assume, that a default scaling has been used. The covariates X_{ij} are assumed to have been scaled to lie in the interval $[0, 1]$. The model fitting functions (`softbart`, `softbart_regression()`,

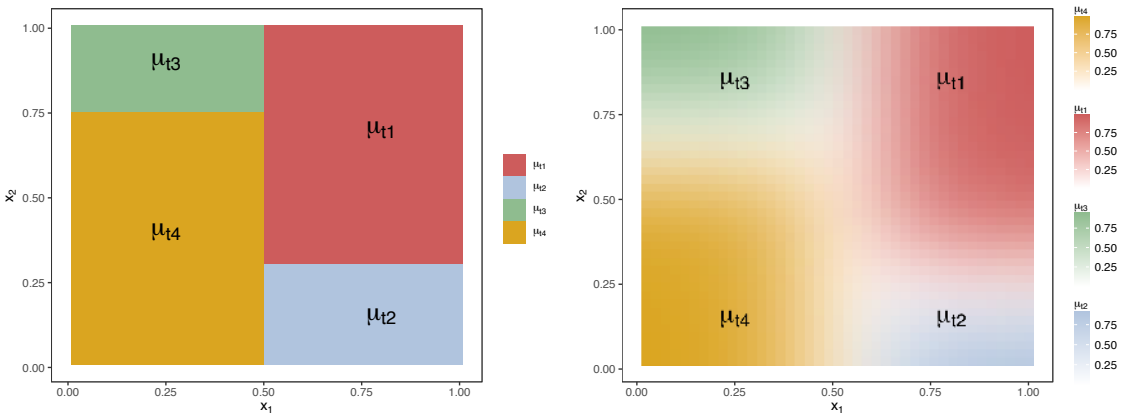

Figure 3: Left: a regression tree function corresponding to "hard" decision rules. Right: a "soft" version of the same regression tree. Points are colored according to the respective values of $\phi_\ell(x; \mathcal{T})$.

`softbart_probit()`, etc) perform this standardization automatically by applying a *quantile transformation* to each numeric covariate, i.e., each covariate is passed through its empirical cdf, and categorical variables with C levels are binarized by introducing C "dummy" variables indicating the factor level. I generally recommend avoiding the legacy function `softbart()`, particularly when working with categorical variables, as this requires the user to manually preprocess the data.

Models with a continuous outcome Y_i preprocess the outcome as well. The `softbart()` function applies a linear transformation to put Y_i in the interval $[-0.5, 0.5]$ as recommended by Chipman et al. (2010), while later functions simply standardize Y_i to have mean 0 and variance 1. I have not found the choice of standardization for Y_i to have a large impact on the results.

Fitting custom models using the `MakeForest()` functionality described in Section 4 unfortunately requires users to scale the covariates and outcomes on their own, and it is also recommended that users think carefully about the hyperparameters they use, as there is no guarantee that what is appropriate for regression/classification problems is appropriate in general.

1.6 Fitting SoftBart Models

`SoftBart` models are fit using a Markov chain Monte Carlo algorithm referred to as *Bayesian backfitting* to approximately sample realizations of $r(x)$ from the posterior distribution. For more details, see Kapelner and Bleich (2016) or Linero and Yang (2018), and for a comprehensive treatment of Markov chain Monte Carlo see Robert and Casella (2004). Bayesian backfitting for the semiparametric Gaussian model proceeds by defining the residuals $R_{it} = Y_i - \sum_{k \neq t} \text{Tree}(X_i; \mathcal{T}_k, \mathcal{M}_k)$ and then noting that $R_{it} \sim \text{Normal}\{\text{Tree}(X_i; \mathcal{T}_t, \mathcal{M}_t), \sigma^2\}$. A valid Gibbs sampler could then proceed by iteratively sampling $(\mathcal{T}_t, \mathcal{M}_t)$ from the posterior distribution of the single-tree model with conditional

$$\pi(\mathcal{T}_t, \mathcal{M}_t \mid \text{Data}, \mathcal{T}_{-t}, \mathcal{M}_{-t}) \propto \prod_i \text{Normal}\{R_{it} \mid \text{Tree}(X_i; \mathcal{T}_t, \mathcal{M}_t), \sigma^2\}\, \pi_{\mathcal{T}}(\mathcal{T}_t)\, \pi_{\mathcal{M}}(\mathcal{M}_t \mid \mathcal{T}_t)$$

This can be sampled via Markov chain Monte Carlo in the following steps:

1. Integrate out $\mathcal{M}_t$ to obtain the marginal likelihood $L(\mathcal{T}_t) = \pi_{\mathcal{T}}(\mathcal{T}_t) \int \text{Normal}\{R_{it} \mid \text{Tree}(X_i; \mathcal{T}_t, \mathcal{M}_t), \sigma^2\}\, \pi_{\mathcal{M}}(\mathcal{M} \mid \mathcal{T}_t)\, d\mathcal{M}$.
2. Propose a modification $\mathcal{T}' \sim q(\cdot \mid \mathcal{T}_t)$ to $\mathcal{T}_t$ and accept this modification with probability $\frac{L(\mathcal{T}')\, q(\mathcal{T}_t \mid \mathcal{T}')}{L(\mathcal{T}_t)\, q(\mathcal{T}' \mid \mathcal{T}_t)} \wedge 1$ (otherwise, leave $\mathcal{T}_t$ unchanged).
3. Sample $\mathcal{M}_t$ from its full conditional given $\mathcal{T}_t$.

Linero and Yang (2018) shows how to carry out these individual steps in the case of `SoftBart`. As we show, this framework can be extended to many other models of interest, with these steps being automatically carried out in custom `SoftBart` models.

2 SoftBart in Action

The `SoftBart` package is available on CRAN and can be installed by running

```
install.packages("SoftBart")
```

Alternatively, for the most up-to-date version of the software, `SoftBart` can be installed from source using the `devtools` package:

```
devtools::install_github("www.github.com/theodds/SoftBart")
```

I show how to use the `softbart` and `softbart_regression` functions to fit a semiparametric Gaussian regression model. Both of these functions fit the same model; the difference is that `softbart` is designed to mirror the usage of the `bart` function in the original `BayesTree` package, while `softbart_regression` specifies models using formulas and also has an associated `predict` function for predicting on data that was not passed to the function.

2.1 The softbart Function

I first illustrate the `softbart` function, which has users pass a matrix of covariates `X`, a vector of outcomes `Y`, and a test set of covariates `X_test` that the user wants to predict $r(x)$ at. Below we generate data under a simulation setting of Friedman (1991), which takes

$$Y_i \sim \text{Normal}\{r_0(X_i), \sigma_0^2\} \quad \text{where} \quad r_0(x) = 10\sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5, \tag{5}$$

where $\sigma_0 = 1$ and $X_i \overset{\text{iid}}{\sim} \text{Uniform}([0,1]^P)$. Note that only $X_{i1}, \ldots, X_{i5}$ are relevant, and X_{ij} is a "nuisance predictor" when $j > 5$. We then use the function `softbart` to fit the semiparametric Gaussian regression model.

```
set.seed(1212)
sim_fried <- function(N,P,sigma) {
  X  <- matrix(runif(N * P), nrow = N, ncol = P)
  mu <- 10 * sin(pi * X[,1] * X[,2]) + 20 * (X[,3] - 0.5)^2 +
    10 * X[,4] + 5 * X[,5]
  Y <- mu + sigma * rnorm(N)
  return(data.frame(X = X, Y = Y, mu = mu))
}

training_data <- sim_fried(250, 250, 1)
test_data     <- sim_fried(250, 250, 1)

X_train <- model.matrix(Y ~ . - 1 - mu, data = training_data)
X_test  <- model.matrix(Y ~ . - 1 - mu, data = test_data)

fitted_softbart <- softbart(X = X_train, Y = training_data$Y,
                            X_test = X_test)
```

The `softbart` function returns an object of type `softbart`, and the associated `plot` function displays a traceplot for the parameter σ and a plot of the outcome Y_i against its prediction $\widehat{r}(X_i)$ where $\widehat{r}(x)$ is the posterior mean of $r(x)$.

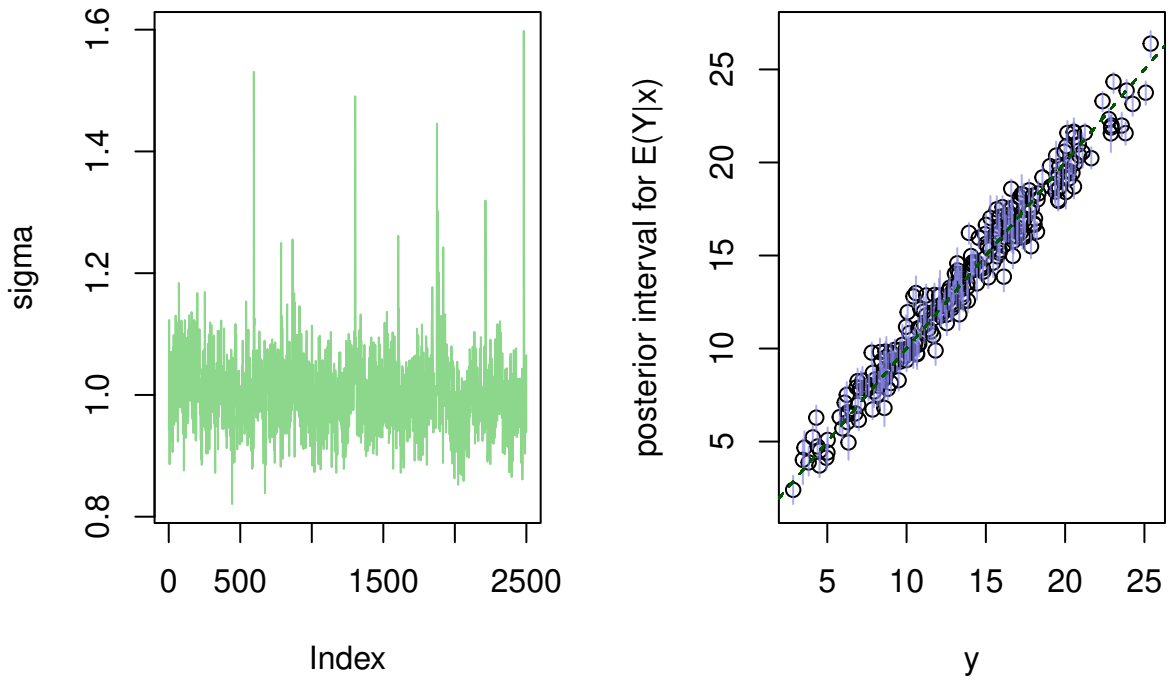

Figure 4: Left: traceplot of σ for the model fit using `softbart`. Right: plot of the predictions against their posterior predicted values, with vertical lines representing 95% confidence (not prediction) intervals for the mean outcome.

```
plot(fitted_softbart)
```

The output is given in Figure 4, and we see that the posterior distribution of σ is concentrated around its true value $\sigma_0 = 1$, the chain for σ mixes quite well, and the predictions produced by `softbart()` are quite close to their true values. We can also check the prediction error on the test set using the `rmse` function, which returns `rmse(x,y)` $= \sqrt{\frac{1}{N}\sum_i (\mathtt{x}_i - \mathtt{y}_i)^2}$. The Bayes estimates of $r_0(X_i)$ for the training and test sets are given by `y_hat_train_mean` and `y_hat_test_mean`, respectively. We compare the true values of $r_0(X_i)$ to their estimates as:

```
rmse(fitted_softbart$y_hat_train_mean, training_data$mu)
```

```
## [1] 0.4172506
```

```
rmse(fitted_softbart$y_hat_test_mean, test_data$mu)
```

```
## [1] 0.4404536
```

The raw samples of $r(X_i)$ for the training and test sets are given by `y_hat_train` and `y_hat_test`, which are matrices with rows corresponding to samples from the Markov chain. These can be used to, among other things, construct credible intervals for the predicted values from the model. For example, the following code constructs a credible interval for $r(X_1)$:

```
quantile(fitted_softbart$y_hat_train[,1], c(0.025, 0.975))
```

```
##     2.5%    97.5%
## 18.82388 20.24865
```

which contains the true value `training_data$mu[1]` $= 19.3$.

2.2 Model Options: Hypers and Opts

Users may also wish to customize the model hyperparameters or change the behavior of the Markov chain (for example, they may want to collect more samples, fix some hyperparameters to be constant, or thin the Markov chain). This can be done by setting the `hypers` and `opts` arguments of `softbart`. These arguments can be constructed using the functions `Hypers()` and `Opts()`, respectively. For example, the following code uses `Opts()` to construct an `opts` argument that (i) increases the number of samples, (ii) turns off the updating of the splitting proportion vector s, and (iii) turns off updating of σ_μ:

```
opts <- Opts(num_burn = 5000, num_save = 5000,
             update_s = FALSE, update_sigma_mu = FALSE)
```

The `Hypers()` function can similarly be used to change the hyperparameters. For example, we can modify the tree-growing prior by changing the values of γ and β in (2) and increase the number of trees T as follows:

```
hypers <- Hypers(X = X_train, Y = training_data$Y,
                 num_tree = 50, beta = 1, gamma = 0.9)
```

The objects `hypers` and `opts` can then be passed to the function `softbart()`.

```
set.seed(19320)
fitted_softbart_2 <- softbart(X_train, training_data$Y, X_test,
                              opts = opts, hypers = hypers)
```

We can then check the performance of the model:

```
rmse(test_data$mu, fitted_softbart_2$y_hat_test_mean)
```

```
## [1] 1.175639
```

We see that this new model performs worse than the old one; this is because the ground truth $r_0(x)$ is sparse, but we have turned off the variable selection prior for s.

`Hypers()` and `Opts()` can be used to modify many other settings, and this is often required when embedding `SoftBart` into other models. We revisit the usage of `Hypers()` and `Opts()` for this purpose in Section 4.

2.3 Other Options for Fitting Models

The `softbart()` function is designed to match the `BayesTree` package in terms of usage. Other functions in `SoftBart` instead use model specifications that are in line with how users specify (say) linear models using `lm()`. For example, the `softbart_regression()` function allows users to specify a model using a formula:

```
fitted_regression <- softbart_regression(Y ~ . - mu,
                                         data = training_data,
                                         test_data = test_data)
```

The `softbart_regression()` function returns an object of type `softbart_regression`, which also has several advantages over the output of `softbart()`. For example, `softbart_regression()` has an associated `predict()` generic that can be used to predict on data after the model is fit. Additionally, `softbart_regression()` and the other functions I discuss (`softbart_probit()`, `gsoftbart_regression()` and `vc_softbart_regression()`) are designed to work with data frames rather than matrices and allow for factors to be passed directly as predictors.

Both the `softbart_regression()` and `softbart_probit()` functions produce objects that can be used with the `predict()` generic, provided `Opts()` is called with `cache_trees = TRUE`, which is done by default. We predict as follows:

```
predicted_values <- predict(fitted_regression, test_data)
names(predicted_values)
```

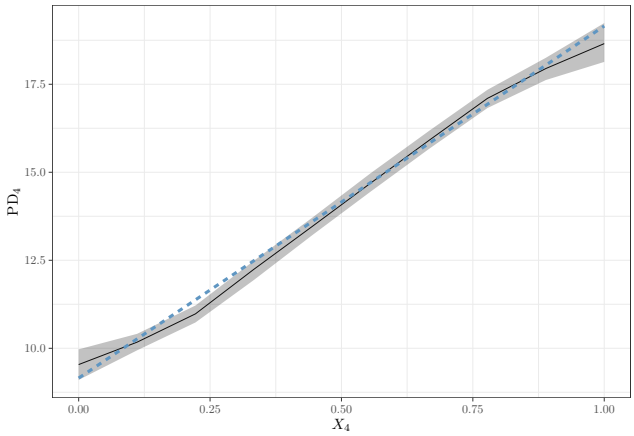

Figure 5: Point estimate (solid black line) and credible band (gray) of the partial dependence function for X_{i4}, with the true partial dependence function given by the dashed line.

```
## [1] "mu"      "mu_mean"
```

For `softbart_regression()`, `predict()` returns both samples of the predicted values (`mu`) on the test set and their posterior means (`mu_mean`).

2.4 Partial Dependence Plots

A common method for summarizing the information contained in black-box models is to construct a *partial dependence plot* (Friedman, 1991). A partial dependence plot for predictor j is constructed from a *partial dependence function*

$$\text{PD}_j(v) = \frac{1}{N}\sum_{i=1}^{N} r(X_{i1}, \ldots, X_{i(j-1)}, v, X_{i(j+1)}, \ldots, X_{iP}).$$

This reduces a multivariate function $r(x)$ to a univariate function that is much easier to visualize. Like other BART packages, `SoftBart` makes it easy to compute samples of $\text{PD}_j(v)$ at a specified collection of v's. This can be done using the `partial_dependence_regression()` function on `softbart_regression` objects. For example, using our model fit to (5) we can estimate PD_4 as

```
grid_x4 <- seq(from = 0, to = 1, length = 10)
pdf_x4 <- partial_dependence_regression(fitted_regression,
                                        training_data, "X.4", grid_x4)
```

```
library(tidyverse)
pdf_offset <- mean(training_data$mu - 10 * training_data$X.4)
ggplot(pdf_x4$pred_df, aes(x = X.4, y = mu)) +
  geom_line(stat = "summary", fun = mean) +
  geom_ribbon(stat = "summary", alpha = 0.3,
              fun.min = function(x) quantile(x, 0.025),
              fun.max = function(x) quantile(x, 0.975)) +
  xlab("$X_4$") + ylab("$\\mbox{PD}_4$") +
  stat_function(fun = function(x) pdf_offset + 10 * x,
                color = "#5F96C2", linetype = 2, size = 2) +
  theme_bw()
```

For convenience, this function returns both a "tidy" (Wickham et al., 2019) `data.frame` to be used with `ggplot()` (Wickham, 2016) as well as a matrix of samples of $\text{PD}_j(v)$ to be used with base `R` plotting functions. Based on the plot (see Figure 5), we see that the fitted model does a good of capturing the true partial

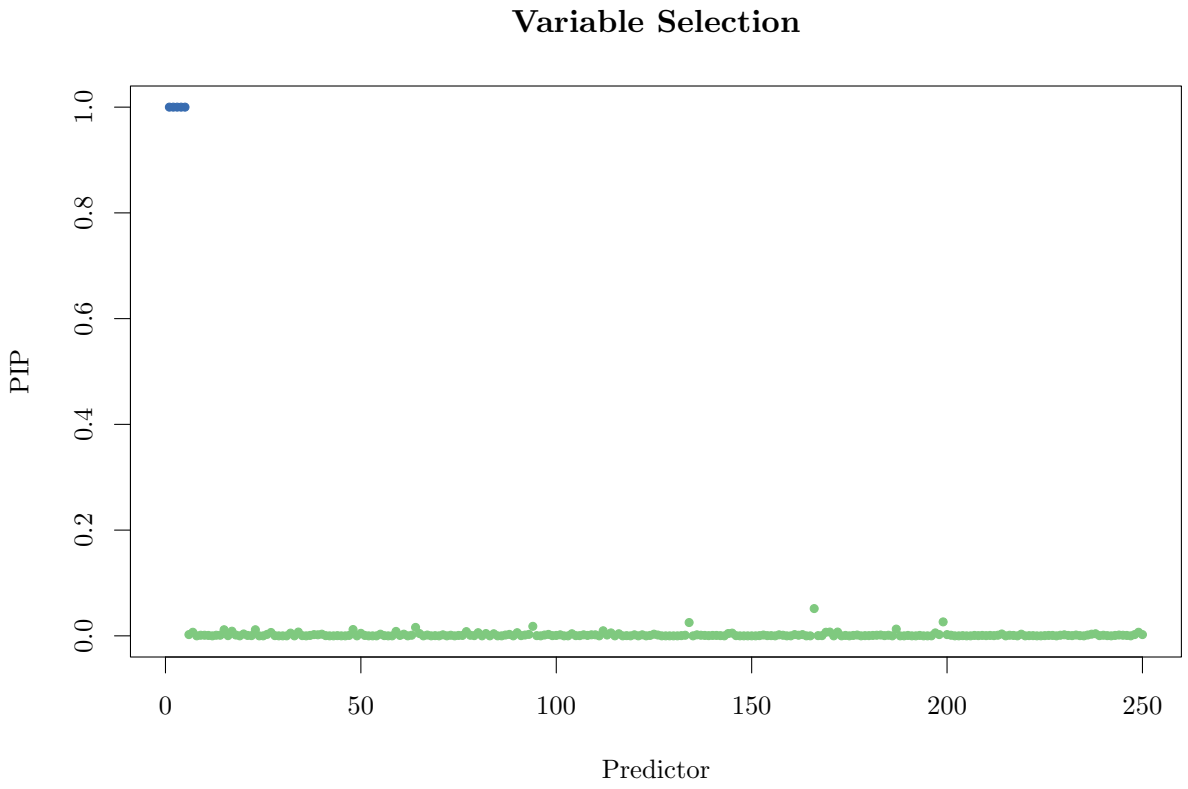

Figure 6: Plot of posterior inclusion probabilities produced by `softbart` using the variable selection prior.

dependence function, which (due to the fact that $r(x)$ is additive in x_4) is equal to $\text{PD}_4(v) = C + 10v$ for some constant C.

3 Variable Selection

By default, `SoftBart` uses the *sparsity inducing prior* introduced by Linero (2018) to perform variable selection. This prior induces sparsity on the vector of splitting proportions by taking $s \sim \text{Dirichlet}(\alpha/P, \ldots, \alpha/P)$, with the idea being that if a variable j is irrelevant then the model can remove j by taking s_j very small. Linero (2018) shows that, when both the number of predictors P and the number of branches B in the ensemble are large, then we have the prior approximation $Q - 1 \sim \text{Poisson}(\theta_B)$ where Q is the number of relevant predictors and $\theta_B = \alpha \sum_{i=0}^{B-1} (\alpha + i)^{-1}$. The value of α can be fixed to obtain a desired amount of sparsity a-priori, but by default `SoftBart` gives α a beta-prime hyperprior $\frac{\alpha}{\alpha+P} \sim \text{Beta}(0.5, 1)$.

Variables can be selected according to their *posterior inclusion probability*

$$\text{PIP}_j = \Pr(\text{predictor } j \text{ appears in the ensemble} \mid \text{Data}).$$

We can then extract the PIP's from the model fit using `posterior_probs()` and plot them (Figure 6):

```
variable_selection <- posterior_probs(fitted_softbart)
plot(variable_selection$post_probs,
     col = ifelse(1:250 < 6, "#386CB0", "#7FC97F"), pch = 20,
     xlab = "Predictor", ylab = "PIP", main = "Variable Selection")
```

The `posterior_probs()` function also returns the *median probability model*, defined by $\mathcal{S} = \{j : \text{PIP}_j \geq 0.5\}$. For the fit to the (5) data, we see that the median probability model coincides with the true data generating process:

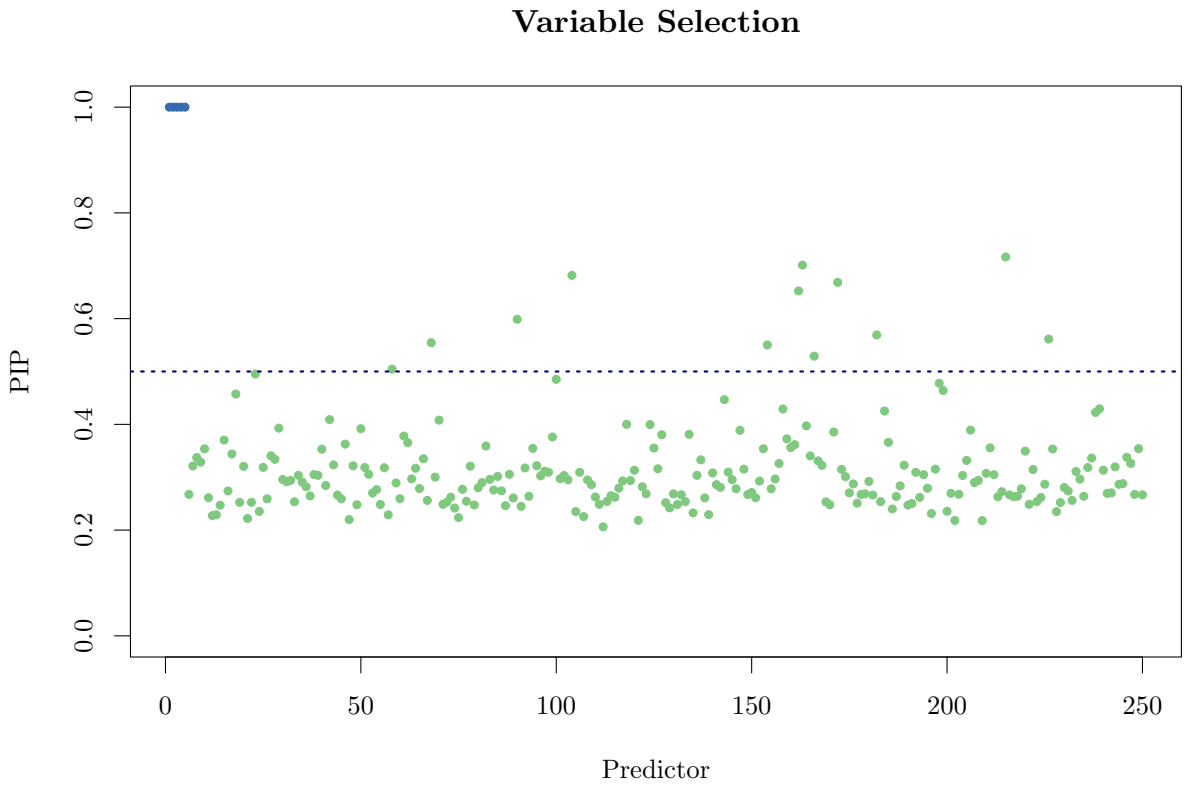

Figure 7: Plot of posterior inclusion probabilities produced by `softbart` without using the variable selection prior.

```
print(variable_selection$median_probability_model)
```

```
## [1] 1 2 3 4 5
```

The variable selection prior can be turned off by setting `update_s = FALSE` in `Opts()`. For example, we can compare the above fit to the fit `fitted_softbart_2` (Figure 7), which we recall does not place a prior on s:

```
variable_selection_2 <- posterior_probs(fitted_softbart_2)
plot(variable_selection_2$post_probs,
     col = ifelse(1:250 < 6, "#386CB0", "#7FC97F"), pch = 20,
     xlab = "Predictor", ylab = "PIP", main = "Variable Selection",
     ylim = c(0,1))
abline(h = 0.5, col = "darkblue", lwd = 2, lty = 3)
```

We see that there are many more variables selected by the median probability model when the variable selection prior is not used and, moreover, all of the predictors have a PIP higher than 0.2. For this reason, some works (Chipman et al., 2010; Bleich and Kapelner, 2014) recommend using only a small number of trees when the end goal is variable selection, as this forces the variables to "compete" for splitting rules in the ensemble. The use of the variable selection prior makes this restriction to small numbers of trees largely unnecessary.

Chipman et al. (2010) also suggest using the number of times a predictor is used in the ensemble as a measure of overall variable importance; this idea, or ideas like it, have also been used to measure the importance of variables for other types of tree-based machine learning algorithms, such as random forests (Breiman, 2001; Díaz-Uriarte and De Andres, 2006). The variable importances are given by the quantity `varimp` (see Figure 8):

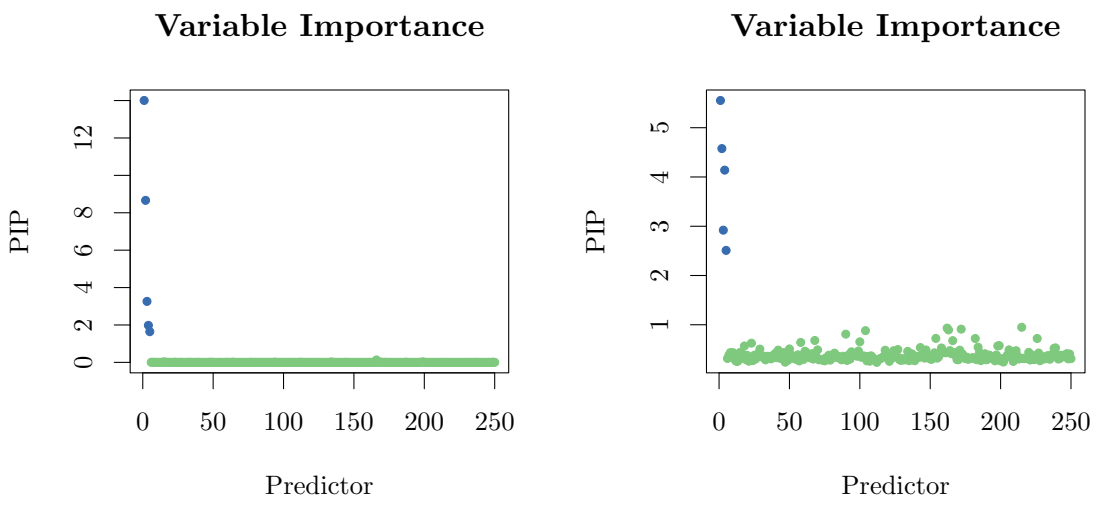

Figure 8: Left: variable importance when using the variable selection prior for s. Right: variable importances without using the variable selection prior for s.

```
par(mfrow = c(1,2))
plot(variable_selection$varimp,
     col = ifelse(1:250 < 6, "#386CB0", "#7FC97F"), pch = 20,
     xlab = "Predictor", ylab = "PIP", main = "Variable Importance")
plot(variable_selection_2$varimp,
     col = ifelse(1:250 < 6, "#386CB0", "#7FC97F"), pch = 20,
     xlab = "Predictor", ylab = "PIP", main = "Variable Importance")
```

We see that placing a prior on s leads to a much sharper transition between the relevant and irrelevant predictors in terms of variable importance. When the data generating mechanism is sparse this leads to better predictions, as the ensemble can dedicate more splitting rules to the relevant predictors; for example, we see in Figure 8 that placing a prior on s also allows the ensemble to use more splitting rules to capture the interaction between x_1 and x_2.

4 Embedding SoftBart Into Other Models

Beyond allowing for the use of soft decision trees, `SoftBart` has the unique advantage of making it easy for researchers to embed a `SoftBart` model into other models within `R` without having to modify the underlying `C++` code. This allows users to seamlessly extend `SoftBart` to any setting in which the Bayesian backfitting algorithm can be applied. Additionally, `SoftBart` allows users to construct *multiple* forests, which is required to implement (for example) the *Bayesian causal forest* (BCF) model of Hahn et al. (2020).

Users can construct a decision tree ensemble using the function `MakeForest()`, which returns a pointer to an `Rcpp` (Eddelbuettel and Balamuta, 2018) data structure called an `Rcpp_Forest`. `Rcpp_Forest`s can be interacted with using the `$` operator, and a full list of available functions is given in the documentation for `MakeForest()`. Some important functions include:

- `forest$do_gibbs(X, Y, X_test, i)` runs i iterations of the Bayesian backfitting algorithm with covariate matrix `X` and outcome `Y`. It returns predictions on the test set of covariates `X_test`. It also changes the data structure itself, with the state of `forest` now reflecting the forest after an additional i iterations have been run. The related function `do_gibbs_weights(X, Y, weights, X_test)` runs a heteroskedastic version of the Bayesian backfitting update, with the error variance for each observation proportional to `1/weights`.
- `forest$do_predict(X)` returns predictions for the covariate matrix `X` at the *current state* of `forest`.

- `forest$get_sigma()` and `forest$set_sigma()` allow us to retrieve and change the error variance σ^2 assumed in the regression model of `Y` on `X`. This can be useful if we have multiple forests being updated with only one error variance parameter.

We give some examples below. For convenience, more developed versions of the basic functions we write are included in `SoftBart` as the `softbart_probit()`, `vc_softbart_regression()`, and `goftbartbart_regression()` functions.

4.1 Probit Regression with Data Augmentation

It is straight-forward using an `Rcpp_Forest` object to implement data augmentation algorithms, such as the algorithm of Albert and Chib (1993). The nonparametric probit regression model takes $[Y_i \mid X_i = x] \sim \text{Bernoulli}[\Phi\{r(x)\}]$, and this model can be expressed in terms of latent variables as

$$Y_i = I(Z_i > 0) \qquad \text{where} \qquad Z_i \sim \text{Normal}\{r(X_i), 1\}.$$

Let $\text{Normal}(\mu, \sigma^2, A)$ denote the normal distribution truncated to the set A and let $A_0 = (-\infty, 0)$ and $A_1 = (0, \infty)$. The data augmentation algorithm of Albert and Chib (1993) alternates between (i) sampling the unobserved latent variables $Z_i \sim \text{Normal}\{r(X_i), 1, A_{Y_i}\}$ and (ii) updating $r(X_i)$ via Bayesian backfitting with the Z_i's as the outcomes. The first step can be accomplished in `R` using the `rtruncnorm()` function in the `truncnorm` package (Mersmann et al., 2018).

In implementing this algorithm, it is important to both set $\sigma = 1$ in our `Rcpp_Forest` and to ensure that σ is not updated, since the variance of Z_i is fixed at 1. The following basic function will fit the probit model using appropriate default values for the hyperparameters:

```
fit_probit <- function(X, Y, X_test, num_tree, num_iter) {

  ## Construct forest
  hypers <- Hypers(X, Y, k = 1/6,
                   num_tree = num_tree, sigma_hat = 1)
  opts <- Opts(update_sigma = FALSE)

  probit_forest <- MakeForest(hypers, opts)

  ## Store the output
  r_train <- matrix(nrow = num_iter, ncol = nrow(X))
  r_test  <- matrix(nrow = num_iter, ncol = X_test)

  ## Initialize chain
  r       <- probit_forest$do_predict(X)
  upper   <- ifelse(Y == 0, 0, Inf)
  lower   <- ifelse(Y == 0, -Inf, 0)
  Z     <- truncnorm::rtruncnorm(n = length(Y_probit_train),
                                 a = lower, b = upper, mean = r, sd = 1)

  ## Do MCMC
  for(i in 1:num_iter) {
    r <- probit_forest$do_gibbs(X, Z, X, 1)
    Z <- truncnorm::rtruncnorm(n = length(Y_probit_train),
                               a = lower, b = upper, mean = r, sd = 1)
    r_train[i,] <- r
    r_test[i,] <- probit_forest$do_predict(X_test)
  }

  ## Return results
```

```
  return(list(r_train = r_train, r_test = r_test))
}
```

The corresponding function `softbart_probit()` in `SoftBart` is more complicated, but ultimately uses the same implementation as the above code.

I now test this function using data from the probit regression model with $r(x) = \frac{3}{5}\{r_0(x) - 14\}$ where $r_0(x)$ is the same function used in our semiparametric regression illustration. I first generate the data:

```
set.seed(77887)
r_probit_train <- 3*(training_data$mu - 14) / 5
r_probit_test  <- 3*(test_data$mu - 14) / 5
p_train        <- pnorm(r_probit_train)
p_test        <- pnorm(r_probit_test)
Y_probit_train <- rbinom(length(p_train), size = 1, prob = p_train)
Y_probit_test <- rbinom(length(p_test), size = 1, prob = p_test)
```

I then fit the model:

```
set.seed(1903)
fitted_probit <- fit_probit(X = X_train, Y = Y_probit_train,
                            X_test = X_test,
                            num_tree = 20, num_iter = 5000)
```

The following code plots the estimated values of $r(X_i)$ from the `fit_probit()` output to against their true values, with the results in Figure 9:

```
plot(colMeans(fitted_probit$r_train), r_probit_train,
     xlab = "$\\widehat r(X_i)$", ylab = "$r(X_i)$",
     pch = 20, col = "#7FC97F")
abline(a = 0, b = 1, col = "#386CB0", lwd = 4, lty = 2)
```

The same functionality is available in the package with the `softbart_probit()` function, which can be fit as follows:

```
probit_data <- data.frame(X = X_train,
                          Y = factor(Y_probit_train, levels = c(0,1)))
probit_test <- data.frame(X = X_test,
                          Y = factor(Y_probit_test, levels = c(0,1)))

fitted_probit <- softbart_probit(Y ~ ., data = probit_data,
                                 test_data = probit_test, verbose = FALSE)
```

4.2 A Varying Coefficient BART Model and a Bayesian Causal Forest

The *varying coefficient BART* (VC-BART) model of Deshpande et al. (2020) assumes a linear relationship in a covariate of interest Z_i, with the regression coefficient possibly depending on the other covariates:

$$Y_i = \alpha(X_i) + Z_i\,\beta(X_i) + \epsilon_i. \tag{6}$$

Here, Z_i is the covariate of interest, X_i is a vector of other covariates, and $\epsilon_i \sim \text{Normal}(0, \sigma^2)$. This model can be fit via a two-stage Gibbs sampler by sampling from the distributions of $[\alpha, \sigma^2 \mid \beta, \text{Data}]$ and $[\beta \mid \alpha, \sigma^2, \text{Data}]$. We can derive the update for (α, σ^2) by forming the residuals $R_{\alpha i} = Y_i - Z_i\,\beta(X_i) = \alpha(X_i) + \epsilon_i$, which follow the usual BART model. An update for $\beta(\cdot)$ can be derived similarly by noting that

$$\frac{Y_i - \alpha(X_i)}{Z_i} \equiv R_{\beta i} \sim \text{Normal}\left\{\beta(X_i), \frac{\sigma^2}{Z_i^2}\right\}.$$

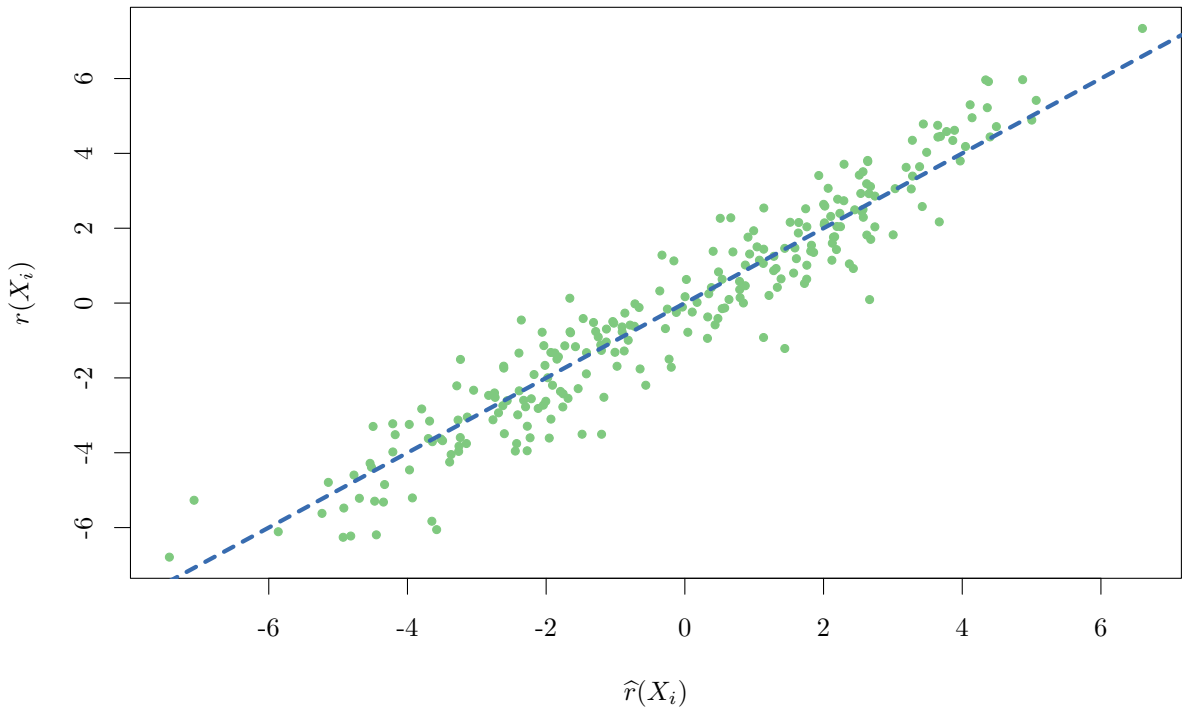

Figure 9: Results from fitting the probit regression model to the simulated data.

Hence $R_{\beta i}$ follows a *heteroskedastic BART* model, with weights $w_i = Z_i^2$. `Rcpp_Forest` objects allow users to specify a vector of weights using the `$do_gibbs_weighted()` method, and can therefore handle the update for β as well. A short function for fitting the VC-BART model is given in Appendix A; the main component of this code is the pair of updates for (α, σ^2) and β:

```
## Update alpha
R_alpha <- Y - Z * beta
alpha <- alpha_forest$do_gibbs(X, R_alpha, X, 1)
sigma <- alpha_forest$get_sigma()

## Update beta
R_beta <- (Y  - alpha) / Z
beta_forest$set_sigma(sigma)
beta <- beta_forest$do_gibbs_weight(X, R_beta, Z^2, X, 1)
```

Note that it is important that `beta_forest` and `alpha_forest` share the same value of σ internally, hence the call of `beta_forest$set_sigma(sigma)`. This design pattern is common in models where multiple forests are used.

To evaluate the model, we generate data that takes $\beta(x) = r(x)$ and $\alpha(x) \equiv 0$, where $r(x)$ is given in (5). Figure 10 shows that our VC-BART model effectively estimates $\beta(X_i)$, σ^2, and sets $\bar{\alpha} = \frac{1}{N}\sum_i \alpha(X_i) \approx 0$.

The varying coefficient model contains the *Bayesian causal forest* (BCF) model of Hahn et al. (2020) as a special case. This model takes the outcome to be the *observed outcome* $Y_i \equiv Y_i(A_i)$ of a pair of *potential outcomes* $\{Y_i(0), Y_i(1)\}$ under a binary treatment variable $a \in \{0, 1\}$. A BCF specifies

$$Y_i(a) = \mu(X_i) + a\,\tau(X_i) + \epsilon_i, \qquad \epsilon_i \sim \text{Normal}(0, \sigma^2).$$

When applying BCFs, one is typically interested in both the *population average causal effect* (PACE), given by $\tau = \mathbb{E}\{Y_i(1) - Y_i(0)\}$ or the *conditional average causal effect* (CACE) given by $\tau(x) = \mathbb{E}\{Y_i(1) - Y_i(0) \mid X_i = x\}$. A simple way to implement a BCF is to define $Z_i = 1/2 - A_i$ and fit the VC-BART model (6). Under the

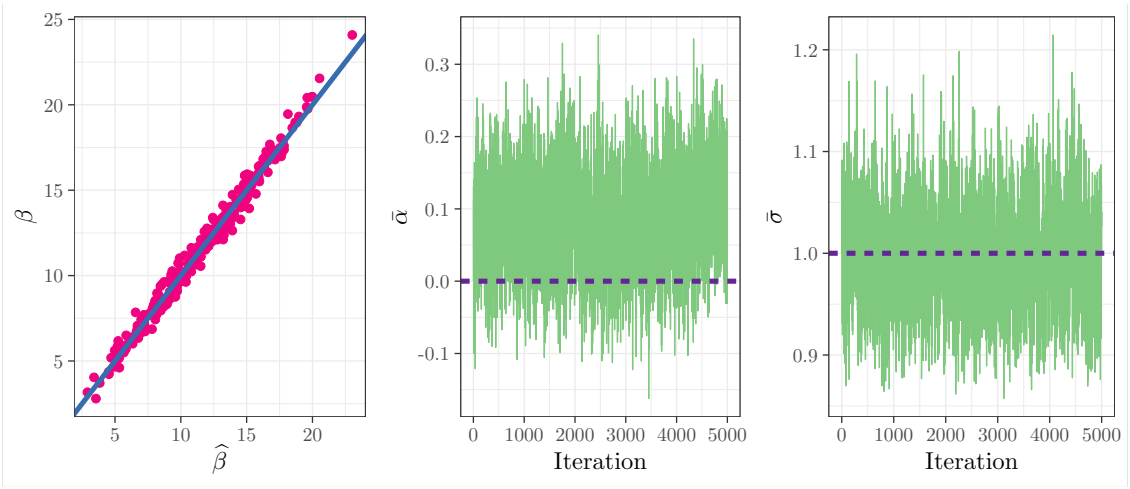

Figure 10: Fit of the varying coefficient model to the simulated data. Left: plot of $\widehat{\beta}(X_i)$ against $\beta(X_i)$ for each observation. Middle: traceplot of $\bar{\alpha}$. Right: traceplot of σ.

common *ignorability* assumption that the treatment Z_i is independent of the potential outcomes $\{Y_i(0), Y_i(1)\}$ conditional on the covariates X_i, it is then easy to show that $\beta(x) = \tau(x)$ for this choice of Z_i.

4.3 The General BART Model

Finally, I show how `SoftBart` can be used to implement the *general BART model* described by Tan and Roy (2019). This is effectively a partial linear model

$$Y_i = r(X_i) + Z_i^\top \beta + \epsilon_i,$$

which is straight-forward to also extend to probit outcomes. This model can also be used to encode a *mixed effects model* when β is a vector of random effects; for simplicity, I will take β to be a set of fixed effects, with a flat prior on β.

Code for fitting this model is given in Appendix B, with the relevant updates being given by the lines

```
## Update beta
R <- Y_train - r_train
beta <- update_beta(R, Z_train, sigma^2)

## Update forest and sigma
R <- Y_train - as.numeric(Z_train %*% beta)
r_train <- forest$do_gibbs(X_train, R, X_train, 1) %>% as.numeric()
sigma <- forest$get_sigma()
```

where the line `beta <- update_beta(R, Z_train, sigma^2)` samples β from its full conditional

$$\beta \sim \text{Normal}\{(\boldsymbol{Z}^\top \boldsymbol{Z})^{-1} \boldsymbol{Z}^\top \boldsymbol{R}, \sigma^2 (\boldsymbol{Z}^\top \boldsymbol{Z})^{-1}\}.$$

I illustrate the use of this code by fitting data generated from the semiparametric Gaussian regression model under (5), but now taking into account the fact that X_4 and X_5 have linear effects. Traceplots and posterior histograms for the parameters $(\beta_1, \beta_2, \sigma)$ (with ground truth values $(10, 5, 1)$) are given in Figure 11, where β_1 is the regression coefficient for X_4 and β_2 is the regression coefficient for X_5. We see that the partial linear model is capable of estimating both of the regression coefficients and the error variance accurately and that the chain mixes well.

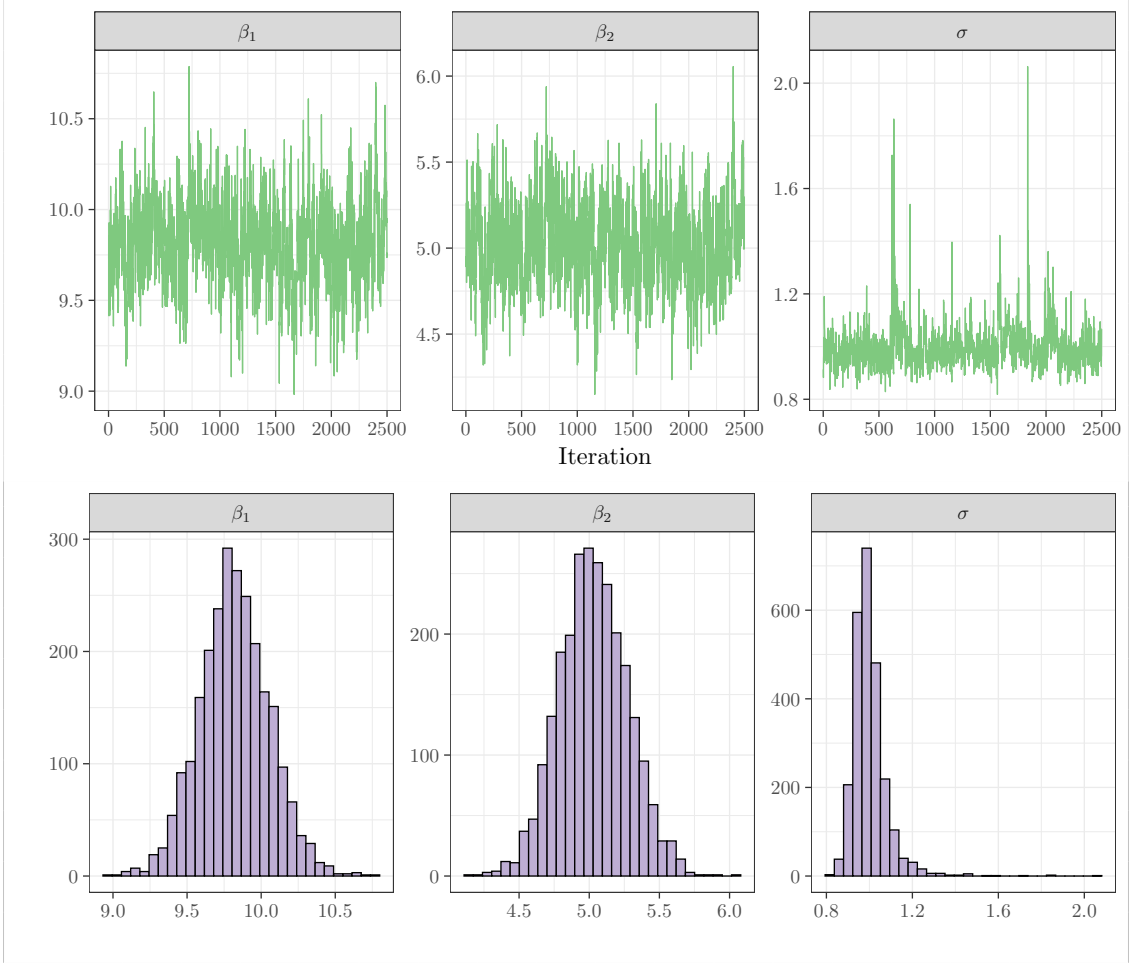

Figure 11: Traceplots (top) and posterior histograms (bottom) associated with the general BART model fit to the simulation setting of (5).

5 Illustration: Diamonds

I illustrate the use of `SoftBart` on the `diamonds` dataset available in the `IIS` package (Schneider, 2017):

```
library(IIS)
data("diamonds_carats_color_cost")
diamonds_pre <- diamonds_carats_color_cost
head(diamonds_pre)
```

```
##   carat color clarity certification_body price
## 1   0.3     D     VS2                GIA  1302
## 2   0.3     E     VS1                GIA  1510
## 3   0.3     G    VVS1                GIA  1510
## 4   0.3     G     VS1                GIA  1260
## 5  0.31     D     VS1                GIA  1641
## 6  0.31     E     VS1                GIA  1555
```

Linero and Yang (2018) showed that `SoftBart` performs better than competing methods (random forests, gradient boosted trees, BART, and the lasso) on this dataset. I fit a semiparametric regression model using `log(price)` as the outcome and with the remaining variables used as predictors.

```
set.seed(77777)
diamonds <- diamonds_pre %>%
  mutate(logprice = log(as.numeric(as.character(price))),
         carat = as.numeric(as.character(carat))) %>% select(-price)

opts <- Opts(num_burn = 5000, num_save = 2500, num_thin = 4)
fitted_diamonds <- softbart_regression(logprice ~ ., data = diamonds,
                                       test_data = diamonds,
                                       opts = opts)
```

This model has four predictors: `carat`, `clarity`, `color`, and `certification_body`. Examining the posterior inclusion probabilities, all of the variables are included in the median probability model with the exception of `certification_body`:

```
posterior_probs(fitted_diamonds)[["post_probs"]]
```

```
##              carat              color            clarity certification_body
##             1.0000             1.0000             1.0000             0.3704
```

Next, I use the `partial_dependence_regression()` function to visualize the partial dependence functions for the variables `carat` and `clarity`. As `carat` measures the weight of a diamond, we expect that PD_{carat} should be increasing in `carat`, as larger diamonds should, all other things being equal, be more expensive:

```
pd_clarity <- partial_dependence_regression(
  fit = fitted_diamonds,
  test_data = diamonds,
  var_str = "clarity",
  grid = unique(diamonds$clarity)
)

pd_carat <- partial_dependence_regression(
  fit = fitted_diamonds,
  test_data = diamonds,
  var_str = "carat",
  grid = unique(diamonds$carat)
)
```

The code below plots the posterior mean and 95% credible bands for the partial dependence functions (see Figure 12):

```
LCL <- function(x) quantile(x, 0.025)
UCL <- function(x) quantile(x, 0.975)

pdp_clarity <- ggplot(pd_clarity$pred_df, aes(x = clarity, y = mu)) +
  geom_point(stat = "summary", fun = mean) +
  geom_errorbar(stat = "summary", fun.min = LCL, fun.max = UCL) +
  xlab("Clarity") +
  ylab("$\\mbox{PD}_{\\mbox{Clarity}}$") +
  theme_bw()

pdp_carat <- ggplot(pd_carat$pred_df, aes(x = carat, y = mu)) +
  geom_ribbon(stat = "summary",
              fun.min = LCL,
              fun.max = UCL,
              alpha = 0.3
  ) +
```

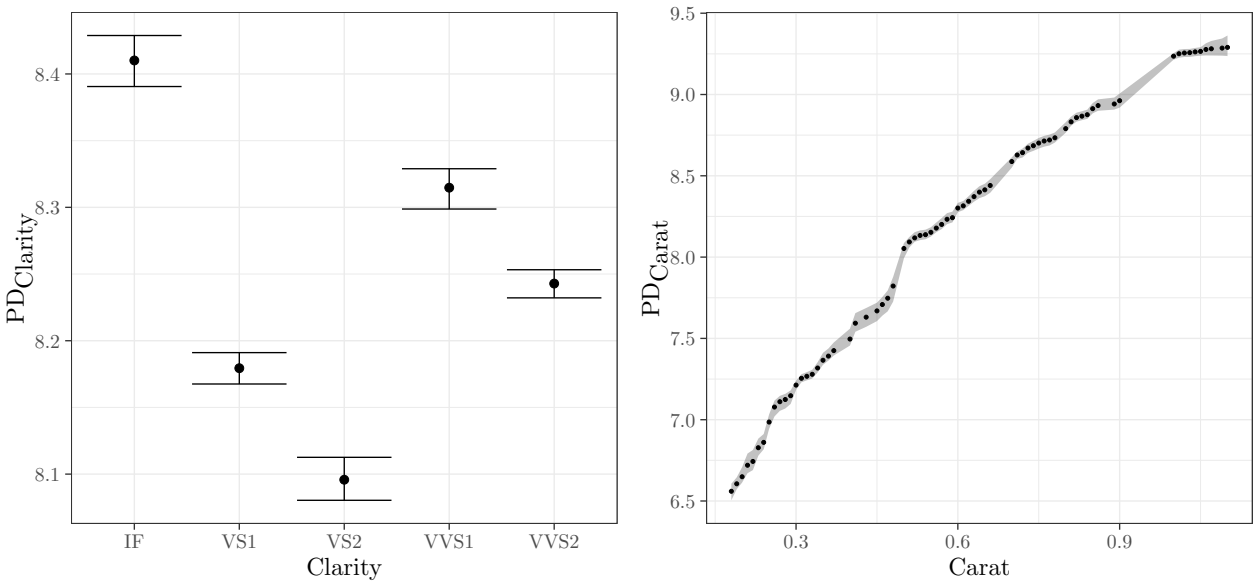

Figure 12: Estimates and credible intervals for the posterior dependence function of Clarity (left) and Carat(right).

```
  geom_point(stat = "summary",
             fun = mean,
             size = 0.5
  ) +
  xlab("Carat") +
  ylab("$\\mbox{PD}_{\\mbox{Carat}}$") +
  theme_bw()

gridExtra::grid.arrange(pdp_clarity, pdp_carat, nrow = 1)
```

We see that there is very little uncertainty in the relationship between `carat` and `log(price)`, although there are some values of `carat` where we do not have much data. There is also a clear effect of `clarity` on `log(price)`; in reality, the variables `color` and `clarity` are ordinal, with the model correctly ranking the possible values of `clarity` as IF (best), VVS1, VVS2, VS1, and VS2 (worst).

`SoftBart` is, to the best of my knowledge, the only package for fitting BART models that allows for partial dependence plots to be computed for categorical variables with three or more levels, and hence it would not be convenient to compute the partial dependence function for `clarity` using other packages.

6 Discussion

Moving forward, it is my plan to expand `SoftBart` to include recent developments in BART methodology. In ongoing work, the `MakeForest()` functionality has been used to extent BCFs to mediation analysis (Imai et al., 2010) and panel data (Bell and Jones, 2015) settings.

There are a couple of important settings where neither `SoftBart` nor most other existing BART packages are applicable. Several recent applications of BART have made critical use of *targeted smoothing* over a variable of interest, such as time (Starling et al., 2020; Li et al., 2022; Linero et al., 2021), and, while it would be feasible to do so, the functionality to do this has not been added to `SoftBart` at this point. Additionally, recent work of Murray (2021) has allowed for the extension of BART to loglinear models, gamma regression models (Linero et al., 2020), heteroskedastic regression models (Pratola et al., 2020), and Cox survival models

(Linero et al., 2021), but the Bayesian backfitting algorithms used for these extensions cannot be used with soft decision trees. Finally, the trees used are strictly univariate, and `SoftBart` does not allow for the use of multivariate decision trees as used by Linero et al. (2020).

A Code for the VC-BART Model

The following function for fitting the VC-BART model takes as input two forests `alpha_forest` and `beta_forest` constructed using the `MakeForest()` function, along with a design matrix `X`, an outcome `y` (both of which are assumed to have been appropriately scaled), and a covariate to be treated linearly `Z`. `SoftBart` includes broader functionality for the VC-BART model via the `vc_softbart_regression()` function.

```
fit_vc_bart <- function(alpha_forest, beta_forest, y, X, Z, num_iter) {

  ## Variables to save
  alpha_out <- matrix(NA, nrow = num_iter, ncol = nrow(X))
  beta_out  <- matrix(NA, nrow = num_iter, ncol = nrow(X))
  sigma_out <- numeric(num_save)

  ## Initializing alpha vector
  alpha <- alpha_forest$do_predict(X)

  for(i in 1:num_iter) {
    R <- (y - alpha) / Z
    beta <- beta_forest$do_gibbs_weighted(X, R, Z^2, X, 1)
    sigma <- beta_forest$get_sigma()
    alpha_forest$set_sigma(sigma)
    R <- (y - Z * beta)
    alpha <- alpha_forest$do_gibbs(X, R, X, 1)

    alpha_out[i,] <- alpha
    beta_out[i,]  <- beta
    sigma_out[i]  <- sigma
  }

  mu_out <- alpha_out + t(Z * t(beta_out))
  return(list(alpha = alpha_out, beta = beta_out,
              sigma = sigma_out, mu = mu_out))
}
```

B Code for the General BART Model

The following function does a conjugate update for a parameter β in a Bayesian linear regression model $R_i = Z_i^\top \beta + \epsilon_i$ under a flat prior for β.

```
update_beta <- function(R, Z, sigma_sq) {
  ZtR <- t(Z) %*% R
  ZtZi <- solve(t(Z) %*% Z)
  beta_hat <- ZtZi %*% ZtR
  Sigma <- sigma_sq * ZtZi
  beta <- MASS::mvrnorm(n = 1, mu = beta_hat, Sigma = Sigma) %>%
    as.numeric()
  return(beta)
}
```

Using the function to update β, along with the functionality from `SoftBart`, we can fit the general BART model. The following function for fitting the generalized BART model takes as input a forest `r_forest` constructed using the `MakeForest()` function, along with a design matrix `X`, an outcome `y` (both of which are assumed to be scaled) and a design matrix `Z` of covariates to be treated linearly. This functionality is available in the `gsoftbart_regression()` function in `SoftBart`.

```
fit_gbart <- function(r_forest, y, X, Z, num_iter) {

  ## Variables to save
  r_out <- matrix(NA, nrow = num_iter, ncol = nrow(X))
  beta_out <- matrix(NA, nrow = num_iter, ncol = ncol(Z))
  sigma_out <- numeric(num_save)
  eta_out <- matrix(NA, nrow = num_iter, ncol = nrow(X))

  ## Initializing
  r <- r_forest$do_predict(X)
  sigma <- r_forest$get_sigma()

  for(i in 1:num_iter) {
    R <- y - r
    beta <- update_beta(R, Z, sigma^2)
    eta <- as.numeric(X %*% beta)
    R <- y - eta
    r <- r_forest$do_gibbs(X, R, X, 1)
    sigma <- r_forest$get_sigma()

    r_out[i,] <- r
    beta_out[i,] <- beta_out
    sigma_out[i] <- sigma
    eta_out[i,] <- eta
  }

  return(list(r = r_out, beta = beta_out, sigma = sigma_out,
              eta = eta_out, mu = eta_out + r_out))

}
```

References

Albert, J. H. and Chib, S. (1993). Bayesian analysis of binary and polychotomous response data. *Journal of the American Statistical Association*, 88(422):669–679.

Bai, R., Moran, G. E., Antonelli, J. L., Chen, Y., and Boland, M. R. (2022). Spike-and-slab group lassos for grouped regression and sparse generalized additive models. *Journal of the American Statistical Association*, 117(537):184–197.

Basak, P., Linero, A. R., Sinha, D., and Lipsitz, S. (2021). Semiparametric analysis of clustered interval-censored survival data using soft Bayesian additive regression trees (SBART). *Biometrics*, 78(3):880–893.

Bell, A. and Jones, K. (2015). Explaining fixed effects: Random effects modeling of time-series cross-sectional and panel data. *Political Science Research and Methods*, 3(1):133–153.

Bleich, J. and Kapelner, A. (2014). Bayesian additive regression trees with parametric models of heteroskedasticity. *arXiv preprint arXiv:1402.5397*.

Bonato, V., Baladandayuthapani, V., Broom, B. M., Sulman, E. P., Aldape, K. D., and Do, K.-A. (2010). Bayesian ensemble methods for survival prediction in gene expression data. *Bioinformatics*, 27(3):359–367.

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

Chipman, H. and McCulloch, R. (2016). *BayesTree: Bayesian Additive Regression Trees*. R package version 0.3-1.4.

Chipman, H. A., George, E. I., and McCulloch, R. E. (2010). BART: Bayesian additive regression trees. *The Annals of Applied Statistics*, 4(1):266–298.

Deshpande, S. K., Bai, R., Balocchi, C., Starling, J. E., and Weiss, J. (2020). Vcbart: Bayesian trees for varying coefficients. *arXiv preprint arXiv:2003.06416*.

Díaz-Uriarte, R. and De Andres, S. A. (2006). Gene selection and classification of microarray data using random forest. *Bioinformatics*, 7(1):1.

Dorie, V. (2022). dbarts: Discrete Bayesian additive regression trees sampler. R package version 0.9-22.

Eddelbuettel, D. and Balamuta, J. J. (2018). Extending `R` with `C++`: A Brief Introduction to `Rcpp`. *The American Statistician*, 72(1):28–36.

Escobar, M. D. and West, M. (1995). Bayesian density estimation and inference using mixtures. *Journal of the American Statistical Association*, 90:577–588.

Freund, Y., Schapire, R., and Abe, N. (1999). A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 4(5):771–780.

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

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232.

Hahn, P. R., Murray, J. S., and Carvalho, C. M. (2020). Bayesian regression tree models for causal inference: Regularization, confounding, and heterogeneous effects (with discussion). *Bayesian Analysis*, 15(3):965–1056.

Henderson, N. C., Louis, T. A., Rosner, G. L., and Varadhan, R. (2020). Individualized treatment effects with censored data via fully nonparametric Bayesian accelerated failure time models. *Biostatistics*, 21(1):50–68.

Hill, J. L. (2011). Bayesian nonparametric modeling for causal inference. *Journal of Computational and Graphical Statistics*, 20(1):217–240.

Imai, K., Keele, L., and Tingley, D. (2010). A general approach to causal mediation analysis. *Psychological Methods*, 15(4):309.

Irsoy, O., Yildiz, O. T., and Alpaydin, E. (2012). Soft decision trees. In *Proceedings of the International Conference on Pattern Recognition*.

Kapelner, A. and Bleich, J. (2016). bartMachine: Machine learning with Bayesian additive regression trees. *Journal of Statistical Software*, 70(4):1–40.

Lamprinakou, S., McCoy, E., Barahona, M., Gandy, A., Flaxman, S., and Filippi, S. (2020). BART-based inference for Poisson processes. *arXiv preprint arXiv:2005.07927*.

Li, Y., Linero, A. R., and Murray, J. S. (2022). Adaptive conditional distribution estimation with Bayesian decision tree ensembles. *Journal of the American Statistical Association*. Advance online publication.

Linero, A. R. (2018). Bayesian regression trees for high-dimensional prediction and variable selection. *Journal of the American Statistical Association*, 113(522):626–636.

Linero, A. R. (2022). Generalized Bayesian additive regression trees models: Beyond conditional conjugacy. *arXiv preprint arXiv:2202.09924*.

Linero, A. R., Basak, P., Li, Y., and Sinha, D. (2021). Bayesian survival tree ensembles with submodel shrinkage. *Bayesian Analysis*. Advance online publication.

Linero, A. R., Sinha, D., and Lipsitz, S. R. (2020). Semiparametric mixed-scale models using shared Bayesian forests. *Biometrics*, 76(1):131–144.

Linero, A. R. and Yang, Y. (2018). Bayesian regression tree ensembles that adapt to smoothness and sparsity. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 80(5):1087–1110.

Linero, A. R. and Zhang, Q. (2022). Mediation analysis using Bayesian tree ensembles. *Psychological Methods*. Advance online publication.

Liu, Y., Gelman, A., and Chen, Q. (2021). Inference from non-random samples using Bayesian machine learning. *arXiv preprint arXiv:2104.05192*.

Mersmann, O., Trautmann, H., Steuer, D., and Bornkamp, B. (2018). *truncnorm: Truncated Normal Distribution*. R package version 1.0-8.

Murray, J. S. (2021). Log-linear Bayesian additive regression trees for multinomial logistic and count regression models. *Journal of the American Statistical Association*, 116(534):756–769.

Orlandi, V., Murray, J., Linero, A., and Volfovsky, A. (2021). Density regression with Bayesian additive regression trees. *arXiv preprint arXiv:2112.12259*.

Prado, E. B., Moral, R. A., and Parnell, A. C. (2021). Bayesian additive regression trees with model trees. *Statistics and Computing*, 31(3):1–13.

Pratola, M. T., Chipman, H. A., George, E. I., and McCulloch, R. E. (2020). Heteroscedastic BART via multiplicative regression trees. *Journal of Computational and Graphical Statistics*, 29(2):405–417.

Ran, H. and Bai, Y. (2021). Distributed soft Bayesian additive regression trees. *arXiv preprint arXiv:2108.11600*.

Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. MIT Press, Cambridge.

Robert, C. and Casella, G. (2004). *Monte Carlo Statistical Methods*. Springer Science & Business Media, New York.

Ročková, V. and van der Pas, S. (2020). Posterior concentration for Bayesian regression trees and forests. *The Annals of Statistics*, 48(4):2108 – 2131.

Saha, E. (2021). *Flexible Bayesian Methods for High Dimensional Data*. PhD thesis, The University of Chicago.

Schneider, G. (2017). *IIS: Datasets to Accompany Wolfe and Schneider - Intuitive Introductory Statistics*. R package version 1.0.

Sparapani, R., Spanbauer, C., and McCulloch, R. (2021). Nonparametric machine learning and efficient computation with Bayesian additive regression trees: The BART R package. *Journal of Statistical Software*, 97(1):1–66.

Sparapani, R. A., Logan, B. R., McCulloch, R. E., and Laud, P. W. (2016). Nonparametric survival analysis using Bayesian additive regression trees (BART). *Statistics in Medicine*, 35(16):2741–2753.

Starling, J. E., Murray, J. S., Carvalho, C. M., Bukowski, R. K., and Scott, J. G. (2020). BART with targeted smoothing: An analysis of patient-specific stillbirth risk. *Annals of Applied Statistics*, 14(1):28–50.

Tan, Y. V. and Roy, J. (2019). Bayesian additive regression trees and the General BART model. *Statistics in Medicine*, 38(25):5048–5069.

Um, S., Linero, A. R., Sinha, D., and Bandyupadhyay, D. (2022). Bayesian additive regression trees for multivariate skewed responses. *Statistics in Medicine*. To appear.

Wickham, H. (2016). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York.

Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., Takahashi, K., Vaughan, D., Wilke, C., Woo, K., and Yutani, H. (2019). Welcome to the `tidyverse`. *Journal of Open Source Software*, 4(43):1686.