bartMachine: Machine Learning with Bayesian Additive Regression Trees

Adam Kapelner
Queens College,
City University of New York

Justin Bleich
The Wharton School of the
University of Pennsylvania

Abstract

We present a new package in R implementing Bayesian additive regression trees (BART). The package introduces many new features for data analysis using BART such as variable selection, interaction detection, model diagnostic plots, incorporation of missing data and the ability to save trees for future prediction. It is significantly faster than the current R implementation, parallelized, and capable of handling both large sample sizes and high-dimensional data.

Keywords: Bayesian, machine learning, statistical learning, non-parametric, R, Java.

1. Introduction

Ensemble-of-trees methods have become popular choices for forecasting in both regression and classification problems. Algorithms such as random forests (Breiman 2001) and stochastic gradient boosting (Friedman 2002) are two well-established and widely employed procedures. Recent advances in ensemble methods include dynamic trees (Taddy, Gramacy, and Polson 2011) and Bayesian additive regression trees (BART; Chipman, George, and McCulloch 2010), which depart from predecessors in that they rely on an underlying Bayesian probability model rather than a pure algorithm. BART has demonstrated substantial promise in a wide variety of simulations and real world applications such as predicting avalanches on mountain roads (Blattenberger and Fowles 2014), predicting how transcription factors interact with DNA (Zhou and Liu 2008) and predicting movie box office revenues (Eliashberg 2010). This paper introduces **bartMachine** (Kapelner and Bleich 2016), a new R (R Core Team 2016) package available from the Comprehensive R Archive Network (CRAN) at `https://CRAN.R-project.org/package=bartMachine` that significantly expands the capabilities of using BART for data analysis.

Currently, there exists one other implementation of BART on CRAN: **BayesTree** (Chipman and McCulloch 2016), the package developed by the algorithm's original authors. One of the major drawbacks of this implementation is its lack of a `predict` function. Test data must be provided as an argument during the training phase of the model. Hence it is impossible to generate forecasts on future data without re-fitting the entire model. Since the run time is not trivial, forecasting becomes an arduous exercise. A significantly faster implementation of BART that contains master-slave parallelization was introduced in Pratola, Chipman, Higdon, McCulloch, and Rust (2013), but this is only available as standalone C++ source code and

Feature	bartMachine	BayesTree
Implementation language	Java	C++
External predict function	Yes	No
Model persistence across sessions	Yes	No
Parallelization	Yes	No
Native missing data mechanism	Yes	No
Built-in cross-validation	Yes	No
Variable importance	Statistical tests	Exploratory
Tree proposal types	3 types	4 types
Partial dependence plots	Yes	Yes
Convergence plots	Assess trees and σ^2	Assess σ^2
Model diagnostics	Yes	No
Incorporation into larger model	No	Through **dbarts**

Table 1: Comparison of features between **bartMachine** and **BayesTree**.

not integrated with R. Additionally, a recent package **dbarts** (Chipman, McCulloch, and Dorie 2014) allows updating of BART with new predictors and response values to incorporate BART into a larger Bayesian model. **dbarts** relies on **BayesTree** as its BART engine.

The goal of **bartMachine** is to provide a fast, easy-to-use, visualization-rich machine learning package for R users. Our implementation of BART is in Java and is integrated into R via **rJava** (Urbanek 2013). From a run time perspective, our algorithm is significantly faster and is parallelized, allowing computation on as many cores as desired. Not only is the model construction itself parallelized, but the additional features such as prediction, variable selection, and many others can be divided across cores as well.

Additionally, we include a variety of expanded and new features. We implement the ability to save trees in memory and provide convenience functions for prediction on test data as well as the ability to save models across R sessions. We also include plotting functions for both posterior credible and predictive intervals and plots for visually inspecting the convergence of BART's MCMC chain. We expand variable importance exploration to include permutation tests and interaction detection. We implement recently developed features for BART including a formal approach to variable selection and the ability to incorporate prior information for covariates (Bleich, Kapelner, Jensen, and George 2014). We also implement the strategy found in Kapelner and Bleich (2015) to incorporate missing data during training and handle missingness during prediction. Table 1 emphasizes the differences in features between **bartMachine** and **BayesTree**, the two existing R implementations of BART.

In Section 2, we provide an overview of BART with a special emphasis on the features that have been extended. In Section 3 we provide a general introduction to the package, highlighting the novel features. Section 4 provides step-by-step examples of the regression capabilities and Section 5 introduces additional step-by-step examples of features unique to classification problems. We conclude in Section 6. Appendix A discusses the details of our algorithm implementation and how it differs from **BayesTree**. Appendix B offers predictive performance comparisons.

2. Overview of BART

BART is a Bayesian approach to nonparametric function estimation using regression trees. Regression trees rely on recursive binary partitioning of predictor space into a set of hyperrectangles in order to approximate some unknown function f. The predictor space has dimension equal to the number of variables, which we denote p. Tree-based regression models have an ability to flexibly fit interactions and nonlinearities. Models composed of sums of regression trees have an even greater ability than single trees to capture interactions and non-linearities as well as additive effects in f.

BART can be considered a sum-of-trees ensemble, with a novel estimation approach relying on a fully Bayesian probability model. Specifically, the BART model can be expressed as:

$$\boldsymbol{Y} = f(\boldsymbol{X}) + \boldsymbol{\mathcal{E}} \approx \mathcal{T}_1^{\mathcal{M}}(\boldsymbol{X}) + \mathcal{T}_2^{\mathcal{M}}(\boldsymbol{X}) + \ldots + \mathcal{T}_m^{\mathcal{M}}(\boldsymbol{X}) + \boldsymbol{\mathcal{E}}, \qquad \boldsymbol{\mathcal{E}} \sim \mathcal{N}_n\left(\boldsymbol{0},\, \sigma^2 \boldsymbol{I}_n\right), \qquad (1)$$

where $\boldsymbol{Y}$ is the $n \times 1$ vector of responses, $\boldsymbol{X}$ is the $n \times p$ design matrix (the predictors column-joined) and $\boldsymbol{\mathcal{E}}$ is the $n \times 1$ vector of noise. Here we have m distinct regression trees, each composed of a tree structure, denoted by $\mathcal{T}$, and the parameters at the terminal nodes (also called leaves), denoted by $\mathcal{M}$. The two together, denoted as $\mathcal{T}^{\mathcal{M}}$ represents an entire tree with both its structure and set of leaf parameters.

The structure of a given tree $\mathcal{T}_t$ includes information on how any observation recurses down the tree. For each nonterminal (internal) node of the tree, there is a "splitting rule" taking the form $\boldsymbol{x}_j < c$, which consists of the "splitting variable" $\boldsymbol{x}_j$ and the "splitting value" c. An observation moves to the left child node if the condition set by the splitting rule is satisfied (and it moves to the right child node otherwise). The process continues until a terminal node is reached. Then, the observation receives the leaf value of the terminal node. We denote the set of the tree's leaf parameters as $\mathcal{M}_t = \{\mu_{t,1}, \mu_{t,2}, \ldots, \mu_{t,b_t}\}$ where b_t is the number of terminal nodes for a given tree. The observation's predicted value is then the sum of the m leaf values arrived at by recursing down all m trees.

BART can be distinguished from other ensemble-of-trees models due to its underlying probability model. As a Bayesian model, BART consists of a set of priors for the structure and the leaf parameters and a likelihood for data in the terminal nodes. The aim of the priors is to provide regularization, preventing any single regression tree from dominating the total fit.

We provide an overview of the BART priors and likelihood and then discuss how draws from the posterior distribution are made. A more complete exposition can be found in Chipman *et al.* (2010).

2.1. Priors and likelihood

The prior for the BART model has three components: (1) the tree structure itself, (2) the leaf parameters given the tree structure, and (3) the error variance σ^2 which is independent of the tree structure and leaf parameters

$$
\begin{aligned}
\mathbb{P}\left(\mathcal{T}_1^{\mathcal{M}}, \ldots, \mathcal{T}_m^{\mathcal{M}}, \sigma^2\right) &= \left[\prod_t \mathbb{P}\left(\mathcal{T}_t^{\mathcal{M}}\right)\right] \mathbb{P}\left(\sigma^2\right) = \left[\prod_t \mathbb{P}\left(\mathcal{M}_t \mid \mathcal{T}_t\right) \mathbb{P}\left(\mathcal{T}_t\right)\right] \mathbb{P}\left(\sigma^2\right) \\
&= \left[\prod_t \prod_\ell \mathbb{P}\left(\mu_{t,\ell} \mid \mathcal{T}_t\right) \mathbb{P}\left(\mathcal{T}_t\right)\right] \mathbb{P}\left(\sigma^2\right),
\end{aligned}
$$

where the last equality follows from an additional assumption of conditional independence of the leaf parameters given the tree's structure.

We first describe $\mathbb{P}\left(\mathcal{T}_t\right)$, the component of the prior which affects the locations of nodes within the tree. Node depth is defined as distance from the root. Thus, the root itself has depth 0, its first child node has depth 1, etc. Nodes at depth d are nonterminal with prior probability $\alpha(1+d)^{-\beta}$ where $\alpha \in (0,1)$ and $\beta \in [0,\infty]$. This component of the tree structure prior has the ability to enforce shallow tree structures, thereby limiting complexity of any single tree and resulting in more model regularization. Default values for these hyperparameters of $\alpha = 0.95$ and $\beta = 2$ are recommended by Chipman *et al.* (2010).

For nonterminal nodes, splitting rules occur in two parts. First, the predictor is randomly selected to serve as the splitting variable. In the original formulation, each available predictor is equally likely to be chosen from a discrete uniform distribution with probability that each variable is selected with probability $1/p$. This is relaxed in our implementation to allow for a generalized Bernoulli distribution where the user specifies $p_1, p_2, \ldots, p_p$ (such that $\sum_{j=1}^{p} p_j = 1$), where each denotes the probability of the jth variable being selected a priori. See Section 4.10 on the covariate prior feature for further details. Additionally, note that "structural zeroes," variables that do not have any valid split values, are assigned probability zero in the implementation (see Appendix A.1 for details). Once the splitting variable is chosen, the splitting value is chosen from the multiset (the non-unique set) of available values at the node via the discrete uniform distribution.

We now describe the prior component $\mathbb{P}\left(\mathcal{M}_t \mid \mathcal{T}_t\right)$ which controls the leaf parameters. Given a tree with a set of terminal nodes, each terminal node (or leaf) has a continuous parameter (the leaf parameter) representing the "best guess" of the response in this partition of predictor space. This parameter is the fitted value assigned to any observation that lands in that node. The prior on each of the leaf parameters is given as: $\mu_\ell \overset{\text{iid}}{\sim} \mathcal{N}\left(\mu_\mu/m,\ \sigma_\mu^2\right)$. The expectation, μ_μ, is picked to be the range center, $(y_{\min} + y_{\max})/2$. The range center can be affected by outliers. If this is a concern, the user can log-transform or windsorize the response before model construction.

The variance hyperparameter σ_μ^2 is empirically chosen so that the range center plus or minus $k = 2$ variances cover 95% of the provided response values in the training set (where $k = 2$ corresponding to 95% coverage is only by default and can be customized). Thus, since there are m trees, we then choose σ_μ such that $m\mu_\mu - k\sqrt{m}\sigma_\mu = y_{\min}$ and $m\mu_\mu + k\sqrt{m}\sigma_\mu = y_{\max}$. The aim of this prior is to provide model regularization by shrinking the leaf parameters towards the center of the distribution of the response. The larger the value of k, the smaller the value of σ_μ^2, resulting in more model regularization.

The final prior is on the error variance and is chosen to be $\sigma^2 \sim \text{InvGamma}\left(\nu/2,\ \nu\lambda/2\right)$. λ is determined from the data so that there is a $q = 90\%$ a priori chance (by default) that the BART model will improve upon the RMSE from an ordinary least squares regression. Therefore, the majority of the prior probability mass lies below the RMSE from least squares regression. Additionally, this prior limits the probability mass placed on small values of σ^2 to prevent overfitting. Thus, the higher the value of q, the larger the values of the sampled σ^2's, resulting in more model regularization.

Note that the adjustable hyperparameters are α, β, k, ν and q. Additionally, m, the number of trees, must be chosen. Default values generally provide good performance, but optimal tuning can be achieved automatically via cross-validation, a feature implemented and described in

Section 4.2.

Along with a set of priors, BART specifies the likelihood of responses in the terminal nodes. They are assumed a priori normal with the mean being the "best guess" in the leaf at the moment (i.e., in the current MCMC iteration) and variance being the best guess of the variance at the moment, $y_\ell \sim \mathcal{N}\left(\mu_\ell, \sigma^2\right)$.

2.2. Posterior distribution and prediction

A Metropolis-within-Gibbs sampler (Geman and Geman 1984; Hastings 1970) is employed to generate draws from the posterior distribution of $\mathbb{P}(\mathcal{T}_1^{\mathcal{M}}, \ldots, \mathcal{T}_m^{\mathcal{M}}, \sigma^2 \mid \boldsymbol{y})$. A key feature of this sampler for BART is to employ a form of "Bayesian backfitting" (Hastie and Tibshirani 2000) where the jth tree is fit iteratively, holding all other $m - 1$ trees constant by exposing only the residual response that remains unfitted:

$$\boldsymbol{R}_{-j} := \boldsymbol{y} - \sum_{t \neq j} \mathcal{T}_t^{\mathcal{M}}(\boldsymbol{X}). \tag{2}$$

The sampler,

$$
\begin{aligned}
1 &: \quad \mathcal{T}_1 \mid \boldsymbol{R}_{-1}, \sigma^2 \\
2 &: \quad \mathcal{M}_1 \mid \mathcal{T}_1, \boldsymbol{R}_{-1}, \sigma^2 \\
3 &: \quad \mathcal{T}_2 \mid \boldsymbol{R}_{-2}, \sigma^2 \\
4 &: \quad \mathcal{M}_2 \mid \mathcal{T}_2, \boldsymbol{R}_{-2}, \sigma^2 \\
&\quad \vdots \\
2m-1 &: \quad \mathcal{T}_m \mid \boldsymbol{R}_{-m}, \sigma^2 \\
2m &: \quad \mathcal{M}_m \mid \mathcal{T}_m, \boldsymbol{R}_{-m}, \sigma^2 \\
2m+1 &: \quad \sigma^2 \mid \mathcal{T}_1, \mathcal{M}_1, \ldots, \mathcal{T}_m, \mathcal{M}_m, \boldsymbol{\mathcal{E}},
\end{aligned}
\tag{3}
$$

proceeds by first proposing a change to the first tree's structure $\mathcal{T}$ which is accepted or rejected via a Metropolis-Hastings step. Note that sampling from the posterior of the tree structure does not depend on the leaf parameters, as they can be analytically integrated out of the computation (see Appendix A.1). Given the tree structure, samples from the posterior of the b leaf parameters $\mathcal{M}_1 := \{\mu_1, \ldots, \mu_b\}$ are then drawn. This procedure progresses iteratively for each tree, using the updated set of partial residuals $\boldsymbol{R}_{-j}$. Finally, conditional on the updated set of tree structures and leaf parameters, a draw from the posterior of σ^2 is made based on the full model residuals $\boldsymbol{\mathcal{E}} := \boldsymbol{y} - \sum_{t=1}^m \mathcal{T}_t^{\mathcal{M}}(\boldsymbol{X})$.

Within a given terminal node, since both the prior and likelihood are normally distributed, the posterior of each of the leaf parameters in $\mathcal{M}$ is conjugate normal with its mean being a weighted combination of the likelihood and prior parameters (lines 2, 4, ..., 2m in Equation set 3). Due to the normal-inverse-gamma conjugacy, the posterior of σ^2 is inverse gamma as well (line $2m + 1$ in Equation set 3). The complete expressions for these posteriors can be found in Gelman, Carlin, Stern, and Rubin (2004).

Lines 1, 3, ..., $2m - 1$ in Equation set 3 rely on Metropolis-Hastings draws from the posterior of the tree distributions. These involve introducing small perturbations to the tree structure: growing a terminal node by adding two child nodes, pruning two child nodes (rendering their parent node terminal), or changing a split rule. We denote the three possible tree alterations

as: GROW, PRUNE, and CHANGE.[1] The mathematics associated with the Metropolis-Hastings step are tedious. Appendix A contains the explicit calculations. Once again, over many MCMC iterations, trees evolve to capture the fit left currently unexplained.

Pratola *et al.* (2013) argue that a CHANGE step is unnecessary for sufficient mixing of the Gibbs sampler. While we too observed this to be true for estimates of the posterior means, we found that omitting CHANGE can negatively impact the variable inclusion proportions (the feature introduced in Section 4.5). As a result, we implement a modified CHANGE step where we only propose new splits for nodes that are singly internal. These are nodes where both children nodes are terminal nodes (details are given in Appendix A.3). After a singly internal node is selected we (1) select a new split attribute from the set of available predictors and (2) select a new split value from the multiset of available values (these two uniform splitting rules were explained in detail previously). We emphasize that the CHANGE step does not alter the tree structure.

All $2m + 1$ steps represent a *single* Gibbs iteration. We have observed that generally no more than 1,000 iterations are needed as "burn-in" (the default package setting is 250). See Section 4.3 for the use of convergence diagnostics. An additional 1,000 iterations are usually sufficient to serve as draws from the posterior for $f(\boldsymbol{x})$. A single predicted value $\hat{f}(\boldsymbol{x})$ can be obtained by taking the average of the posterior values and a quantile estimate can be obtained by computing the appropriate quantile of the posterior values. Additional features of the posterior distribution will be discussed in Section 4.

2.3. BART for classification

BART can easily be modified to handle classification problems for categorical response variables. In Chipman *et al.* (2010), only binary outcomes were explored but recent work has extended BART to the multiclass problem (Kindo, Wang, and Pe 2013). Our implementation handles binary classification and we plan to implement multiclass outcomes in a future release.

For the binary classification problem (coded with outcomes "0" and "1"), we assume a probit model,

$$\mathbb{P}\left(Y = 1 \mid \boldsymbol{X}\right) = \Phi\left(\mathcal{T}_1^{\mathcal{M}}(\boldsymbol{X}) + \mathcal{T}_2^{\mathcal{M}}(\boldsymbol{X}) + \ldots + \mathcal{T}_m^{\mathcal{M}}(\boldsymbol{X})\right),$$

where Φ denotes the cumulative density function of the standard normal distribution. In this formulation, the sum-of-trees model serves as an estimate of the conditional probit at $\boldsymbol{x}$ which can be easily transformed into a conditional probability estimate of $Y = 1$.

In the classification setting, the prior on σ^2 is not needed as the model assumes $\sigma^2 = 1$. The prior on the tree structure remains the same as in the regression setting and a few minor modifications are required for the prior on the leaf parameters.

Sampling from the posterior distribution is again obtained via Gibbs sampling in combination with a Metropolis-Hastings step outlined in Section 2.2. Following the data augmentation approach of Albert and Chib (1993), an additional vector of latent variables $\boldsymbol{Z}$ is introduced into the Gibbs sampler. Then, a new step is created in the Gibbs sampler where draws of

---

[1] In the original formulation, Chipman *et al.* (2010) include an additional alteration called SWAP. Due to the complexity of the bookkeeping associated with this alteration, we do not implement it.

$Z \mid y$ are obtained by conditioning on the sum-of-trees model:

$$Z_i \mid y_i = 1 \quad \sim \quad \max\left\{ N\left(\sum_t \mathcal{T}_t^{\mathcal{M}}\left(\boldsymbol{X}\right), 1\right), 0\right\} \quad \text{and}$$

$$Z_i \mid y_i = 0 \quad \sim \quad \min\left\{ N\left(\sum_t \mathcal{T}_t^{\mathcal{M}}\left(\boldsymbol{X}\right), 1\right), 0\right\}.$$

Next, $\boldsymbol{Z}$ is used as the response vector instead of $\boldsymbol{y}$ in all steps of Equation 3.

Upon obtaining a sufficient number of samples from the posterior, inferences can be made using the posterior distribution of conditional probabilities and classification can be undertaken by applying a threshold to the averages (or another summary) of these posterior probabilities. The relevant classification features of **bartMachine** are discussed in Section 5.

3. The bartMachine package

The package **bartMachine** provides a novel implementation of Bayesian additive regression trees in R. The algorithm is substantially faster than the current R package **BayesTree** and our implementation is parallelized at the MCMC iteration level during prediction. Additionally, the interface with **rJava** allows for the entire posterior distribution of tree ensembles to persist throughout the R session, allowing for prediction and other calls to the trees without having to re-run the Gibbs sampler (a limitation in the current **BayesTree** implementation). The **bartMachine** object can be serialized, thereby persisting across R sessions as well (a feature discussed in Section 4.12). Since our implementation is different from **BayesTree**, we provide a predictive accuracy bakeoff on different datasets in Appendix B which illustrates that the two exhibit similar performance.

3.1. Speed improvements and parallelization

We make a number of significant speed improvements over the original implementation.

First, **bartMachine** is fully parallelized (with the number of cores customizable) during model creation, prediction, and many of the other features. During model creation, we chose to parallelize by creating one independent Gibbs chain per core. Thus, if we employ the default 250 burn-in samples and 1,000 post-burn-in samples and four cores, each core would sample 500 samples: 250 for a burn-in and 250 post-burn-in samples. The final model will aggregate the four post burn-in chains for the four cores yielding the desired 1,000 total post-burn-in samples. This has the drawback of effectively running the burn-in serially (which suffers from Amdahl's Law), but has the added benefit of reducing auto-correlation of the sum-of-trees samples in the posterior samples since the chains are independent which may provide greater predictive performance. Parallelization at the level of likelihood calculations is left for a future release as we were unable to address the costs of thread overhead. Parallelization for prediction and other features scale linearly in the number of cores without Amdahl's diminishing returns.

Additionally, we take advantage of a number of additional computational shortcuts:

1. Computing the unfitted responses for each tree (Equation 2) can be accomplished by keeping a running vector and making entry-wise updates as the Gibbs sampler (Equation 3)

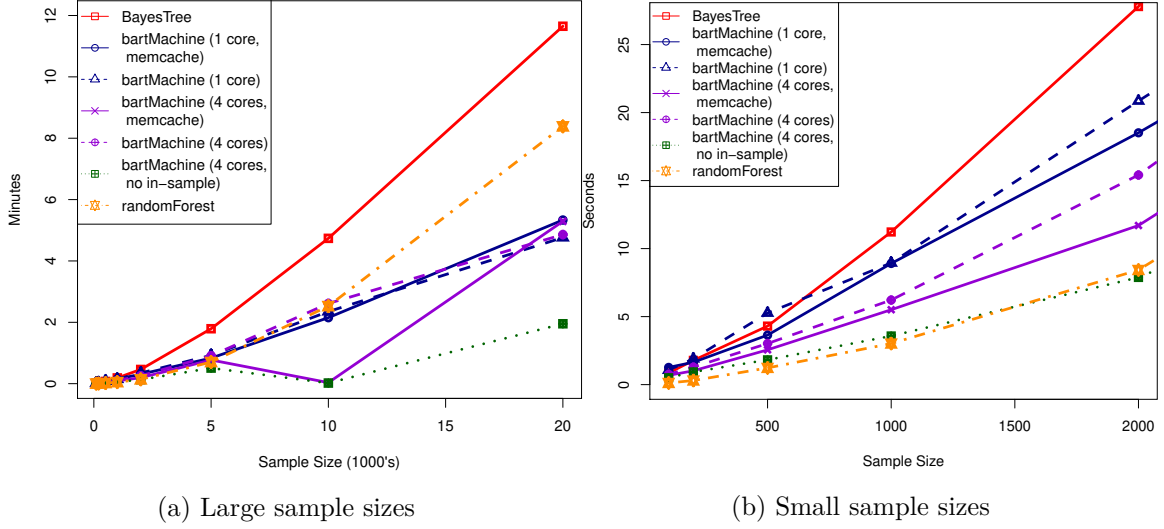(a) Large sample sizes   (b) Small sample sizes

Figure 1: Model creation times as a function of sample size for a number of settings of **bartMachine**, **BayesTree** and **randomForest**. Simulations were run on a quad-core 3.4GHz Intel i5 desktop with 24GB of RAM running the Windows 7 64bit operating system.

progresses from step 1 to $2m$. Additionally, during the σ^2 sampling (step $2m + 1$), the residuals do not have to be computed by dropping the data down all the trees.

2. Each node caches its acceptable variables for split rules and the acceptable unique split values so they do not need to be calculated at each tree sampling step. Recall from the discussion concerning uniform splitting rules in Section 2.1 that acceptable predictors and values change based on the data available at an arbitrary location in the tree structure. This speed enhancement, which we call *memcache* comes at the expense of memory and may cause issues for large data sets. We include a toggle in our implementation defaulted to "on."

3. Careful calculations in Appendix A eliminate many unnecessary computations. For instance, the likelihood ratios are only functions of the squared sum of responses and no longer require computing the sum of the responses squared.

Figure 1 displays model creation speeds for different values of n on a linear regression model with $p = 20$, normally distributed covariates, $\beta_1, \ldots, \beta_{20} \overset{\text{iid}}{\sim} \text{U}(-1, 1)$, and standard normal noise. Note that we do not vary p as it was already shown in Chipman *et al.* (2010) that BART's computation time is largely unaffected by the dimensionality of the problem. We include results for BART using **BayesTree**, **bartMachine** with one and four cores, the *memcache* option on and off, as well as four cores, *memcache* off and computation of in-sample statistics off (all with $m = 50$ trees). In-sample statistics by default are computed consisting of in-sample predictions ($\hat{y}$), residuals ($e := y - \hat{y}$), $L1$ error which is defined as $\sum_{i=1}^{n_{\text{train}}} |e_i|$, $L2$ error which is defined as $\sum_{i=1}^{n_{\text{train}}} e_i^2$, pseudo-$R^2$ which is defined as $1 - L2/(\sum_{i=1}^{n_{\text{train}}} (y_i - \bar{y})^2)$ and root mean squared error which is defined as $\sqrt{L2/n_{\text{train}}}$. We also include random forests model creation times via the package **randomForest** (Liaw and Wiener 2002) with its default settings.

1:	If x_{ij} is missing, send it $\longleftarrow$; if it is present and $x_{ij} \leq x_{ij}^*$, send it $\longleftarrow$, otherwise $\longrightarrow$.
2:	If x_{ij} is missing, send it $\longrightarrow$; if it is present and $x_{ij} \leq x_{ij}^*$, send it $\longleftarrow$, otherwise $\longrightarrow$.
3:	If x_{ij} is missing, send it $\longleftarrow$; if it is present, send it $\longrightarrow$.

Table 2: The MIA choices for all attributes $j \in \{1, \ldots, p\}$ and all split points x_{ij}^* where $i \in \{1, \ldots, n\}$ during a GROW or CHANGE step in **bartMachine**.

We first note that Figure 1a demonstrates that the **bartMachine** model creation run time is approximately linear in n (without in-sample statistics computed). There is about a 30% speed-up when using four cores instead of one. The *memcache* enhancement should only be turned off when regressing "large" sample sizes. (We cannot give general advice here; the user should experiment on their specific dataset and specific hardware to find when the RAM requirement of this feature exceeds capacity). Noteworthy is the 50% reduction in time of constructing the model when not computing in-sample statistics. In-sample statistics are computed by default because the user generally wishes to see them. Also, for the purposes of this comparison, **BayesTree** models compute the in-sample statistics by necessity since the trees are not saved. The **randomForest** implementation becomes slower just after $n = 1,000$ due to its reliance on greedy exhaustive search at each node.

Figure 1b displays results for smaller sample sizes ($n \leq 2,000$) that are often encountered in practice. We observe the *memcache* enhancement provides about a 10% speed improvement. Thus, if memory is an issue, it can be turned off with little performance degradation.

3.2. Missing data in bartMachine

bartMachine implements a native method for incorporating missing data into both model creation and future prediction with test data. The details are given in Kapelner and Bleich (2015) but we provide a brief summary here.

There are a number of ways to incorporate missingness into tree-based methods (see Ding and Simonoff 2010 for a review). The method implemented here is known as "Missing Incorporated in Attributes" (MIA; Twala, Jones, and Hand 2008, Section 2) which natively incorporates missingness by augmenting the nodes' splitting rules to (a) also handle sorting the missing data to the left or right and (b) use missingness *itself* as a variable to be considered in a splitting rule. Table 2 summarizes these new splitting rules as they are implemented within the package.

Implementing MIA into the BART procedure is straightforward. These new splitting rules are sampled uniformly during the GROW or CHANGE steps. For example, a splitting rule might be "$\boldsymbol{x}_j < c$ or $\boldsymbol{x}_j$ is missing." To account for splitting on missingness itself, we create dummy vectors of length n for each of the p attributes, denoted $\boldsymbol{M}_1, \ldots, \boldsymbol{M}_p$, which assume the value 1 when the entry is missing and 0 when the entry is present. The original training matrix is then augmented with these dummies, giving the opportunity to select missingness *itself* when choosing a new splitting rule during the grow or change steps. Note that this can increase the number of predictors by up to a factor of 2. We illustrate building a **bartMachine** model with missing data in Section 4.8. As described in Chipman *et al.* (2010, Section 6), BART's run time increases negligibly in the number of covariates and this has been our experience using the augmented training matrix.

3.3. Variable selection

Our package also implements the variable selection procedures developed in Bleich *et al.* (2014), which are best applied to data problems where the number of covariates influencing the response is small relative to the total number of covariates. We give a brief summary of the procedures here.

In order to select variables, we make use of the "variable inclusion proportions:" the proportion of times each predictor is chosen as a splitting rule divided by the total number of splitting rules appearing in the model (see Section 4.5 for more details). The variable selection procedure can be outlined as follows:

1. Compute the model's variable inclusion proportions.

2. Permute the response vector, thereby breaking the relationship between the covariates and the response. Rebuild the model and compute the "null" variable inclusion proportions. Repeat this a number of times to create a null permutation distribution.

3. Three selection rules can be used depending on the desired stringency of selection:

   (a) Local Threshold: Include a predictor $\boldsymbol{x}_k$ if its variable inclusion proportion exceeds the $1 - \alpha$ quantile of its own null distribution.

   (b) Global Max Threshold: Include a predictor $\boldsymbol{x}_k$ if its variable inclusion proportion exceeds the $1 - \alpha$ quantile of the distribution of the maximum of the null variable inclusion proportions from each permutation of the response.

   (c) Global SE Threshold: Select $\boldsymbol{x}_k$ if its variable inclusion proportion exceeds a threshold based from its own null distribution mean and SD with a global multiplier shared by all predictors.

The Local procedure is the least stringent in terms of selection and the Global Max procedure the most. The Global SE procedure is a compromise, but behaves more similarly to the Global Max. Bleich *et al.* (2014) demonstrate that the best procedure depends on the underlying sparsity of the problem, which is often unknown. Therefore, the authors include an additional procedure that chooses the best of these thresholds via cross-validation and this method is also implemented in **bartMachine**. As highlighted in Bleich *et al.* (2014), this method performs favorably compared to variable selection using random forests' "importance scores", which rely on the reduction in out-of-bag forecasting accuracy that occurs from shuffling the values for a particular predictor and dropping the out-of-bag observations down each tree. Examples of these procedures for variable selection are provided in Section 4.9.

4. bartMachine package features for regression

We illustrate the package features by using both real and simulated data, focusing first on regression problems.

4.1. Computing parameters

We first set some computing parameters. In this exploration, we allow up to 5GB of RAM for the Java heap[2] and we set the number of computing cores available for use to 4.

```
R> options(java.parameters = c("-Xmx20g", "--add-modules=jdk.incubator.vector", "-XX:+UseZ
R> library("bartMachine")
R> set_bart_machine_num_cores(4)


bartMachine now using 4 cores.
```

The following Sections 4.2–4.9 use a dataset obtained from the UCI Machine Learning Repository (Bache and Lichman 2013). The $n = 201$ observations are automobiles and the goal is to predict each automobile's price from 25 features (15 continuous and 10 nominal), first explored by Kibler, Aha, and Albert (1989).[3] This dataset also contains missing data. We omit missing data for now (41 observations that will later be retained in Section 4.8) and we create a variable for the design matrix X and the response y. The following code loads the data.

```
R> data("automobile", package = "bartMachine")
R> automobile <- na.omit(automobile)
R> y <- automobile$log_price
R> X <- automobile; X$log_price <- NULL
```

4.2. Model building

We are now ready to construct a **bartMachine** model. The default hyperparameters generally follow the recommendations of Chipman *et al.* (2010) and provide a ready-to-use algorithm for many data problems. Our hyperparameter settings are $m = 50$,[4] $\alpha = 0.95$, $\beta = 2$, $k = 2$, $q = 0.9$, $\nu = 3$, and probabilities of the GROW / PRUNE / CHANGE steps is 28% / 28% /44%. We retain the default number of burn-in Gibbs samples (250) as well as the default number of post-burn-in samples (1,000). We default the covariates to be equally important *a priori*. Other parameters and their defaults can be found in the package's online manual. Below is a default **bartMachine** model. Here, X denotes automobile attributes and y denotes the log price of the automobile.

```
R> bart_machine <- bartMachine(X, y)


Building bartMachine for regression ... evaluating in sample data...done
```

---

[2]Note that the maximum amount of memory can be set only *once* at the beginning of the R session (a limitation of **rJava** since only one Java Virtual Machine can be initiated per R session), but the number of cores can be respecified at any time.

[3]We first preprocess the data. We first drop one of the nominal predictors (car company) due to too many categories (22). We then coerce two of the of the nominal predictors to be continuous. Further, the response variable, price, was logged to reduce right skew in its distribution.

[4]In contrast to Chipman *et al.* (2010), we recommend this default as a good starting point rather than $m = 200$ due to our experience experimenting with the "RMSE by number of trees" feature found later in this section. Performance is often similar and computational time and memory requirements are dramatically reduced.

If one wishes to see more information about the individual iterations of the Gibbs sampler of Equation 3, the flag `verbose` can be set to `TRUE`. One can see more debug information from the `Java` program by setting the flag `debug_log` to `TRUE` and the program will print to `unnamed.log` in the current working directory.

Below we inspect the model object to query its in-sample performance and to be reminded of the input data and model hyperparameters.

```
R> bart_machine


bartMachine v1.2.1 for regression

training data n = 160 and p = 41
built in 0.5 secs on 4 cores, 50 trees, 250 burn-in and 1000 post. samples

sigsq est for y beforehand: 0.014
avg sigsq estimate after burn-in: 0.00751


in-sample statistics:
 L1 = 7.86
 L2 = 0.62
 rmse = 0.06
 Pseudo-Rsq = 0.9798
p-val for shapiro-wilk test of normality of residuals: 0.06781
p-val for zero-mean noise: 0.95335
```

Since the response was continuous, **bartMachine** for regression was employed automatically (line 1). Line 3 prints the dimensions of the design matrix. Line 4 records the creation time along with other model parameters. Line 6 records the MSE for the OLS model and Line 7 displays the **bartMachine** model's estimate of σ_e^2. We are then given in-sample statistics on error in lines 10–13. The pseudo-R^2 is calculated via $1 - SSE/SST$. Also provided are outputs from tests of the error distribution being normal (line 14) and mean centered (line 15). Note that the "p-val for shapiro-wilk test of normality of residuals" is marginally less than 5%. Thus we conclude that the noise of Equation 1 is not normally distributed. Just as when interpreting the results from a linear model, non-normality implies we should be circumspect concerning **bartMachine** output that relies on this distributional assumption such as the credible and prediction intervals of Section 4.4.

We can also obtain out-of-sample statistics to assess level of overfitting by using *k*-fold cross-validation. Using 10 randomized folds we find:
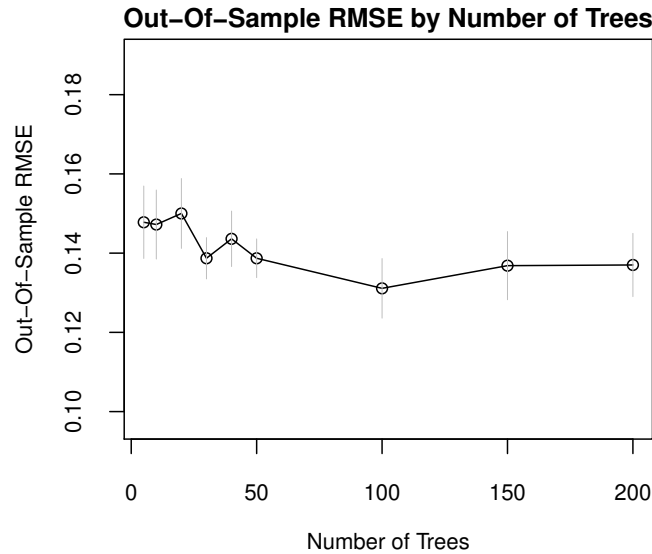
```
R> k_fold_cv(X, y, k_folds = 10)


..........
$L1_err
[1] 15.96878


$L2_err
```

Figure 2: Out-of-sample predictive performance by number of trees.

```
[1] 2.682349

$rmse
[1] 0.1294785

$PseudoRsq
[1] 0.9133438
```

The code provides the out-of-sample statistics for the model built above. This function also returns the $\hat{y}$ predictions as well as the vector of the fold indices (which are omitted in the output shown above).

The Pseudo-R^2 being lower out-of-sample (above) versus in-sample suggests that **bartMachine** is slightly overfitting (note also that the training sample during cross-validation is 10% smaller).

It may also be of interest how the number of trees m affects performance. One can examine how out-of-sample predictions vary by the number of trees via

```
R> rmse_by_num_trees(bart_machine, num_replicates = 20)
```

and the output is shown in Figure 2. This illustration suggests that predictive performance levels off around $m = 50$. We observe similar illustrations across a wide variety of datasets and hyperparameter choices which is the reason we have set $m = 50$ as the default value in the package.

Note that there is nominal improvement at $m = 200$. There may also be improvement if other hyperparameters are varied. We can attempt to build a better **bartMachine** model using the procedure `bartMachineCV` by grid-searching over a set of hyperparameter combinations, including m (for more details, see BART-cv in Chipman *et al.* 2010). The grid of interest can be customized by the user and defaults to a small grid.

```
R> bart_machine_cv <- bartMachineCV(X, y)
```

```
...
bartMachine CV win: k: 2 nu, q: 3, 0.9 m: 200
```

This function returns the "winning" model, which is the one with lowest out-of-sample RMSE over a 5-fold (by default) cross-validation. Here, the cross-validated **bartMachine** model has slightly better in-sample performance (L1 = 8.18, L2 = 0.68 and Pseudo-R^2 = 0.978) in comparison to the default model (output in this section, page 12) as well as slightly better out-of-sample performance (L1 = 21.05, L2 = 4.40 and Pseudo-R^2 = 0.858) in comparison to the vanilla BART performance above when assessed via:

```
R> k_fold_cv(X, y, k_folds = 10, k = 2, nu = 3, q = 0.9, num_trees = 200)
```

Predictions are handled with the `predict` function. Below are fits for the first seven observations.

```
R> predict(bart_machine_cv, X[1:7, ])
```

```
[1]  9.494941  9.780791  9.795532 10.058445  9.670211  9.702682  9.911394
```

We also include a convenience method `bart_predict_for_test_data` that will predict and return out-of-sample error metrics when the test outcomes are known.

4.3. Assumption checking

The package includes features that assess the plausibility of the BART model assumptions. Checking the mean-centeredness of the noise is addressed in the summary output of Section 4.2, page 12 and is simply a one-sample *t*-test of the average residual value against a null hypothesis of true mean zero. We assess both normality and heteroskedasticity via:

```
R> check_bart_error_assumptions(bart_machine_cv)
```

This will display a plot similar to Figure 3 which contains a QQ-plot (to assess normality) as well as a residual-by-predicted plot (to assess homoskedasticity). There is little evidence of the errors violating normality and homoskedasticity.

In addition to the model assumptions, BART requires convergence of its Gibbs sampler which can be investigated via:

```
R> plot_convergence_diagnostics(bart_machine_cv)
```

Figure 4 displays the plot which features four types of convergence diagnostics (each one is detailed in the figure caption). It appears via visual inspection that the **bartMachine** model has been sufficiently burned-in as each of the plots seems to exhibit a stationary process.
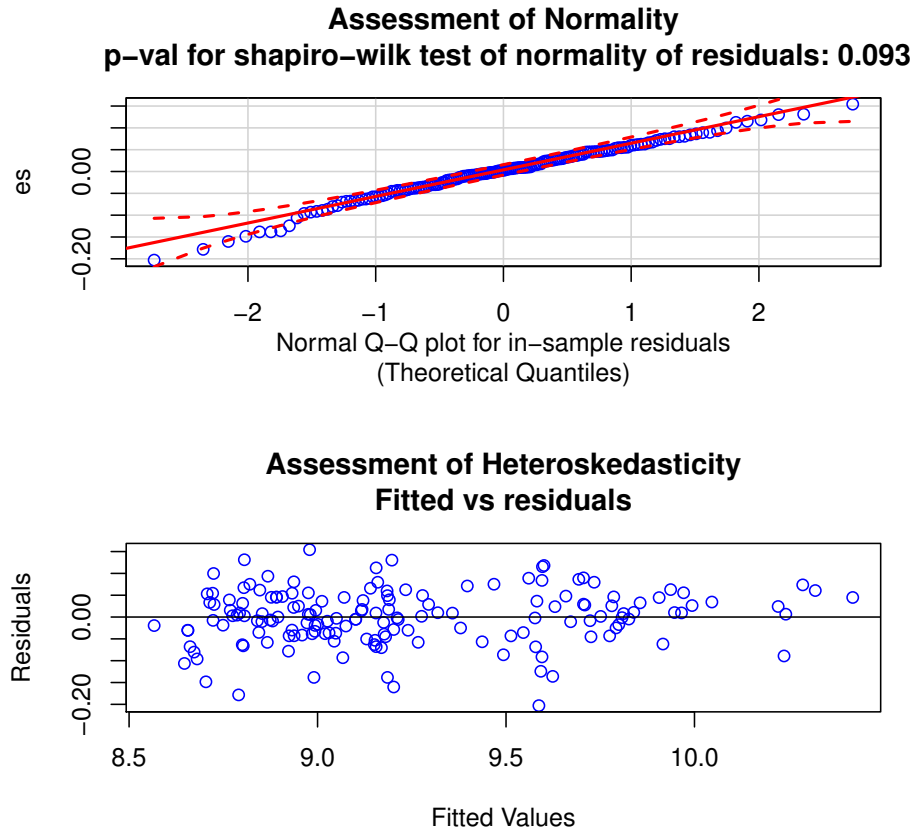
Assessment of Normality
p–val for shapiro–wilk test of normality of residuals: 0.093

Normal Q–Q plot for in–sample residuals
(Theoretical Quantiles)

Assessment of Heteroskedasticity
Fitted vs residuals

Fitted Values

Figure 3: Test of normality of errors using QQ-plot and the Shapiro-Wilk test (top), residual plot to assess heteroskedasticity (bottom).

4.4. Credible intervals and prediction intervals

An advantage of BART is that if we believe the priors and model assumptions, the Bayesian probability model and corresponding burned-in MCMC iterations provide the approximate posterior distribution of $f(\boldsymbol{x})$. Thus, one can compute uncertainty estimates via quantiles of the posterior samples. These provide Bayesian "credible intervals" which are intervals for the conditional expectation function, $\mathbb{E}[\boldsymbol{y} \mid \boldsymbol{X}]$.

Another useful uncertainty interval can be computed for individual predictions by combining uncertainty from the conditional expectation function with the systematic, homoskedastic normal noise produced by $\mathcal{E}$. This is accomplished by generating 1,000 samples (by default) from the posterior predictive distribution and then reporting the appropriate quantiles.

Below is an example of how both types of intervals are computed in the package (for the 100th observation of the training data):

```
R> round(calc_credible_intervals(bart_machine_cv, X[100, ],
+    ci_conf = 0.95), 2)

     ci_lower_bd ci_upper_bd
[1,]        8.74        8.98
```
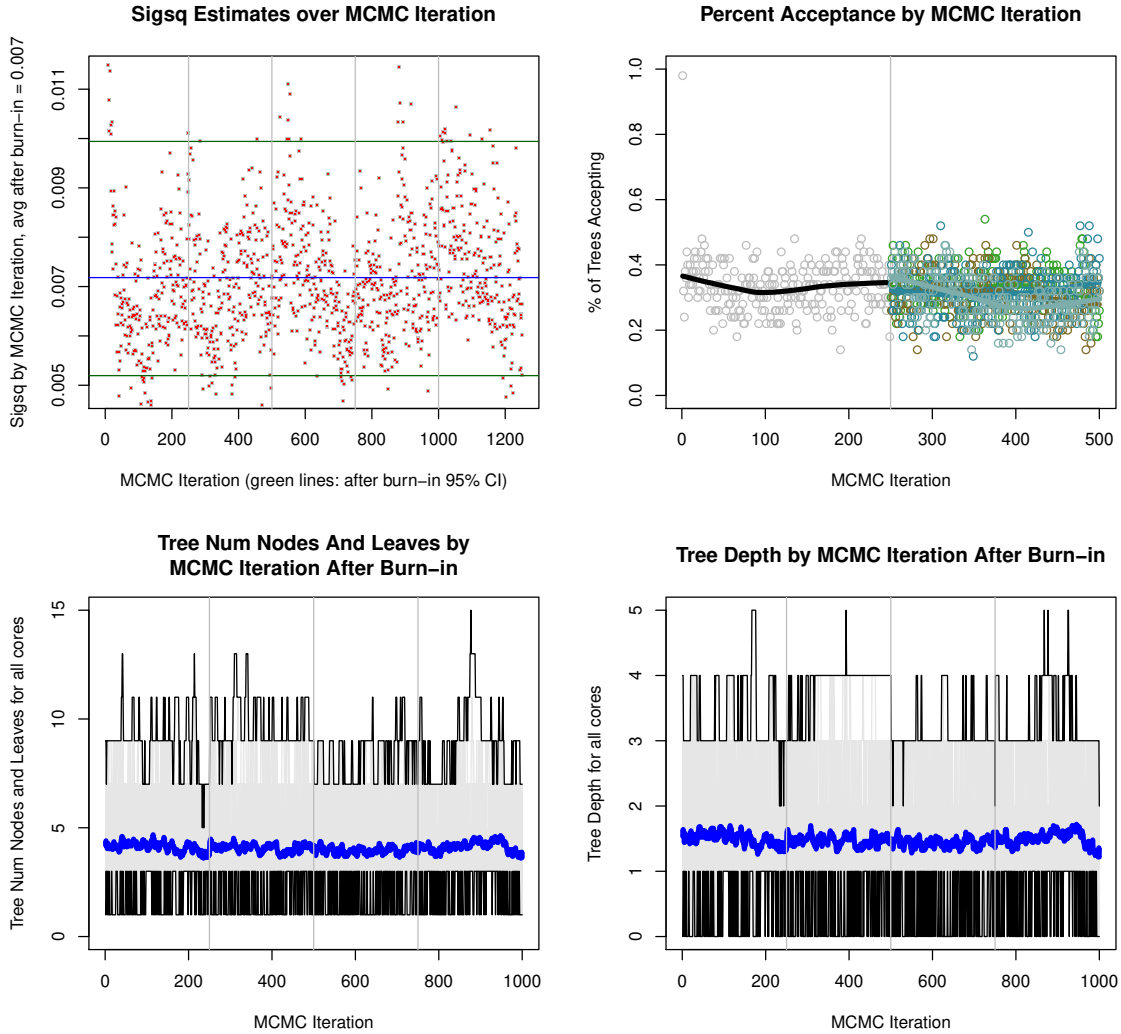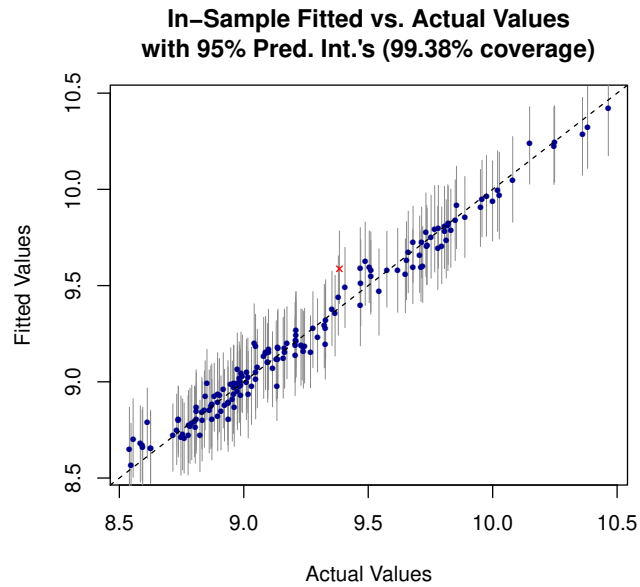
Figure 4: Convergence diagnostics for the cross-validated **bartMachine** model. Top left: σ^2 by MCMC iteration. Samples to the left of the first vertical gray line are burn-in from the first computing core's MCMC chain. The four subsequent plots separated by gray lines are the post-burn-in iterations from each of the four computing cores employed during model construction. Top right: percent acceptance of Metropolis-Hastings proposals across the m trees where each point plots one iteration. Points before the gray vertical line illustrate burn-in iterations and points after illustrate post-burn-in iterations. Each computing core is colored differently. Bottom left: average number of leaves across the m trees by iteration (post burn-in only where computing cores are separated by vertical gray lines). Bottom right: average tree depth across the m trees by iteration (post-burn-in only where computing cores are separated by vertical gray lines).

```
R> round(calc_prediction_intervals(bart_machine_cv, X[100, ],
+    pi_conf = 0.95), 2)

    pi_lower_bd pi_upper_bd
[1,]       8.65        9.07
```

In–Sample Fitted vs. Actual Values
with 95% Cred. Int.'s (90% coverage)
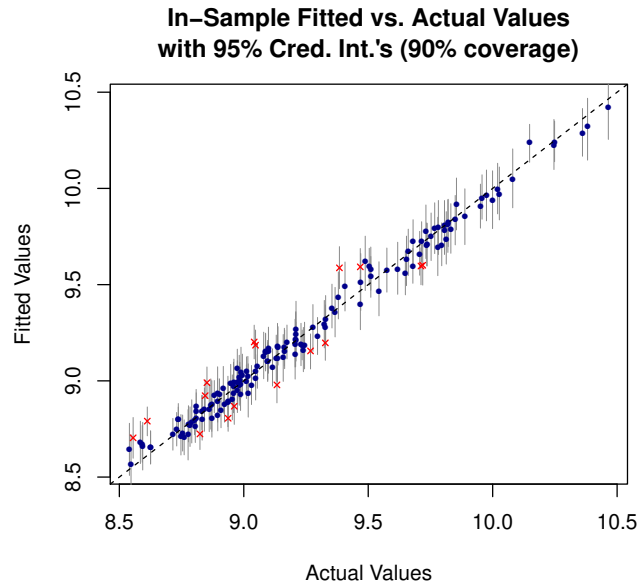
(a) Segments illustrate credible intervals

In–Sample Fitted vs. Actual Values
with 95% Pred. Int.'s (99.38% coverage)

(b) Segments illustrate prediction intervals

Figure 5: Fitted versus actual response values for the automobile dataset. Segments are 95% credible intervals (a) or 95% prediction intervals (b). Green dots indicate the true response is within the stated interval and red dots indicate otherwise. Note that the percent coverage in (a) is not expected to be 95% because the response includes a noise term.

Note that the prediction intervals are wider than the credible intervals because they reflect the uncertainty from the error term. We can then plot these intervals in-sample:

```
R> plot_y_vs_yhat(bart_machine_cv, credible_intervals = TRUE)
R> plot_y_vs_yhat(bart_machine_cv, prediction_intervals = TRUE)
```
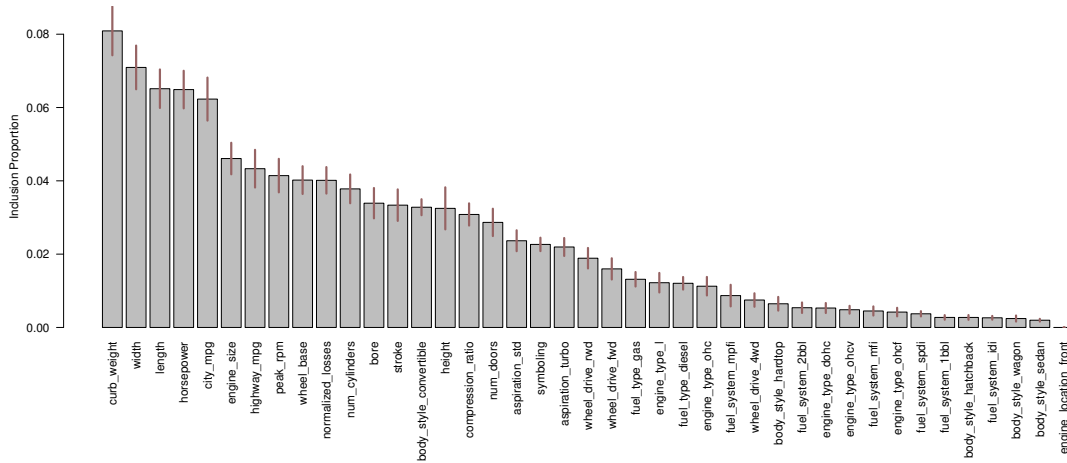
Figure 6: Average variable inclusion proportions in the cross-validated **bartMachine** model for the automobile data averaged over 100 model constructions to obtain stable estimates across many posterior modes in the sum-of-trees distribution (as recommended in Bleich *et al.* 2014). The segments atop the bars represent 95% confidence intervals. The eight predictors with inclusion proportions of zero are predictors with identically one value (after missing data was dropped).

Figure 5a shows how our prediction fared against the original response (in-sample) with 95% credible intervals. Figure 5b shows the same prediction versus the original response plot now with 95% prediction intervals.

4.5. Variable importance

After a **bartMachine** model is built, it is natural to ask the question: which variables are most important? This is assessed by examining the splitting rules in the m trees across the post-burn-in MCMC iterations which are known as "inclusion proportions" (Chipman *et al.* 2010). The inclusion proportion for any given predictor represents the proportion of times that variable is chosen as a splitting rule out of all splitting rules among the posterior draws of the sum-of-trees model. Figure 6 illustrates the inclusion proportions for all variables obtained via:

```
R> investigate_var_importance(bart_machine_cv, num_replicates_for_avg = 20)
```

Selection of variables which *significantly* affect the response is addressed briefly in Section 3.3, examples are provided in Section 4.9 but for full treatment of this feature, please see Bleich *et al.* (2014).

4.6. Variable effects

It is also natural to ask: does x_j affect the response, controlling for other variables in the model? This is roughly analogous to the t-test in ordinary least squares regression of no linear effect of x_j on y while controlling for all other variables, x_{-j}. The null hypothesis here is the same but the linearity constraint is relaxed. To test this, we employ a permutation approach

where we record the observed Pseudo-R^2 from the **bartMachine** model built with the original data. Then we permute the $\boldsymbol{x}_j$th column, thereby destroying any relationship between $\boldsymbol{x}_j$ and $\boldsymbol{y}$, construct a new duplicate **bartMachine** model from this permuted design matrix and record a "null" Pseudo-R^2. We then repeat this process to obtain a null distribution of Pseudo-R^2's. Since the alternative hypothesis is that $\boldsymbol{x}_j$ has an effect on $\boldsymbol{y}$ in terms of predictive power, our p value is the proportion of null Pseudo-R^2's greater than the observed Pseudo-R^2, making our procedure a natural one-sided test. Note, however, that this test is conditional on the BART model and its selected priors being true, similar to the assumptions of the linear model.

If we wish to test if a set of covariates $A \subset \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_p\}$ affect the response after controlling for other variables, we repeat the procedure outlined in the above paragraph by permuting the predictors in A in every null sample. We do not permute each column separately, but instead permute the rows as a unit in order to preserve collinearity structure. This is roughly analogous to the partial F-test in ordinary least squares regression.

If we wish to test if *any* of the covariates matter in predicting $\boldsymbol{y}$, we simply permute $\boldsymbol{y}$ during the null sampling. This procedure breaks the relationship between the response and the predictors but does not alter the existing associations between predictors. This is roughly analogous to the omnibus F-test in ordinary least squares regression.

At $\alpha = 0.05$, Figure 7a demonstrates an insignificant effect of the variable `width` of car on price. Even though `width` is putatively the "most important" variable as measured by proportions of splits in the posterior sum-of-trees model (Figure 6), note that this is largely an easy prediction problem with many collinear predictors. Figure 7b shows the results of a test of the putatively most important categorical variable, `body style` (which involves permuting the categories, then dummifying the levels to preserve the structure of the variable). We find a marginally significant effect ($p = 0.0495$). A test of the top ten most important variables is convincingly significant (Figure 7c). For the omnibus test, Figure 7d illustrates an extremely statistically significant result, as would be expected. The code to run these tests is shown below (output suppressed).

```
R> cov_importance_test(bart_machine_cv, covariates = "width")
R> cov_importance_test(bart_machine_cv, covariates = "body_style")
R> cov_importance_test(bart_machine_cv, covariates = c("width",
+    "curb_weight", "city_mpg", "length", "horsepower", "body_style",
+    "engine_size", "highway_mpg", "peak_rpm", "normalized_losses"))
R> cov_importance_test(bart_machine_cv)
```

4.7. Partial dependence

A data analyst may also be interested in understanding how $\boldsymbol{x}_j$ affects the response on average, after controlling for other predictors. This can be examined using Friedman (2001)'s partial dependence function (PDP),

$$f_j(\boldsymbol{x}_j) = \mathbb{E}_{\boldsymbol{x}_{-j}}\left[f(\boldsymbol{x}_j, \boldsymbol{x}_{-j})\right] := \int f(\boldsymbol{x}_j, \boldsymbol{x}_{-j})\mathrm{dP}\left(\boldsymbol{x}_{-j}\right). \tag{4}$$

The PDP of predictor $\boldsymbol{x}_j$ gives the average value of f when $\boldsymbol{x}_j$ is fixed and $\boldsymbol{x}_{-j}$ varies over its marginal distribution, $\mathrm{dP}\left(\boldsymbol{x}_{-j}\right)$. As neither the true model f nor the distribution of the

(a) `width`

(b) `body style`

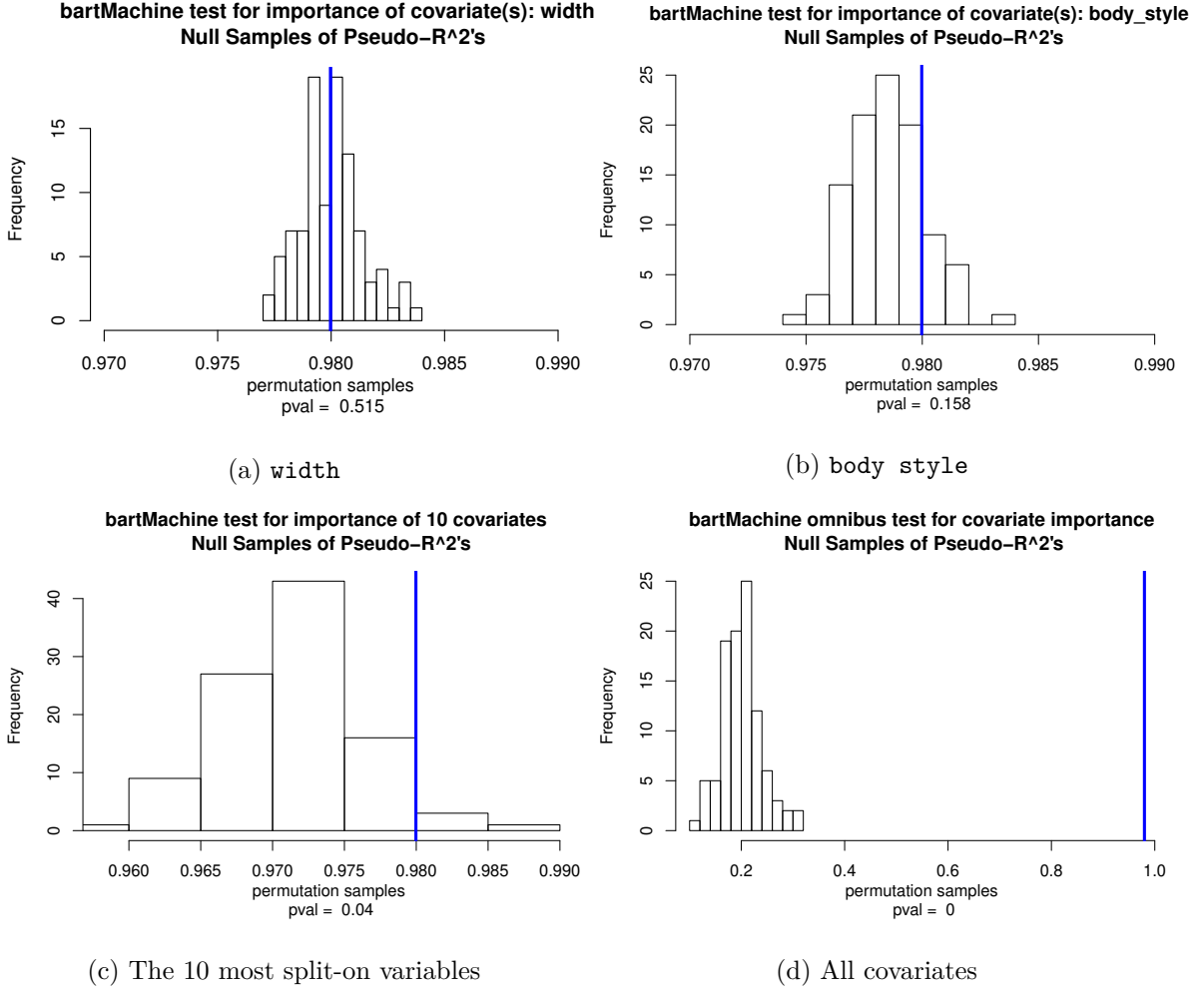(c) The 10 most split-on variables

(d) All covariates

Figure 7: Tests of covariate importance conditional on the cross-validated **bartMachine** model. All tests performed with 100 null samples.

predictors dP $(\boldsymbol{x}_{-j})$ are known, we estimate Equation 4 by computing

$$\hat{f}_j(\boldsymbol{x}_j) = \frac{1}{n} \sum_{i=1}^{n} \hat{f}(\boldsymbol{x}_j, \boldsymbol{x}_{-j,i}) \tag{5}$$

where n is the number of observations in the training data and $\hat{f}$ denotes predictions via the **bartMachine** model. Since BART provides an estimated posterior distribution, we can plot credible bands for the PDP function. Credible bands are computed as follows: in Equation 5, $\hat{f}$ can be replaced with a function that calculates the qth quantile of the post-burn-in MCMC iterations for $\hat{y}$. Figure 8a plots the PDP along with the 2.5%ile and the 97.5%ile for the variable `horsepower`. By varying over most of the range of `horsepower`, the price is predicted to increase by about $1000, holding all else constant. Figure 8b plots the PDP along with the 2.5%ile and the 97.5%ile for the variable `stroke`. This predictor seemed to be relatively unimportant according to Figure 6 and the PDP confirms this, with a very small, yet nonlinear average partial effect. The code for both plots is below.
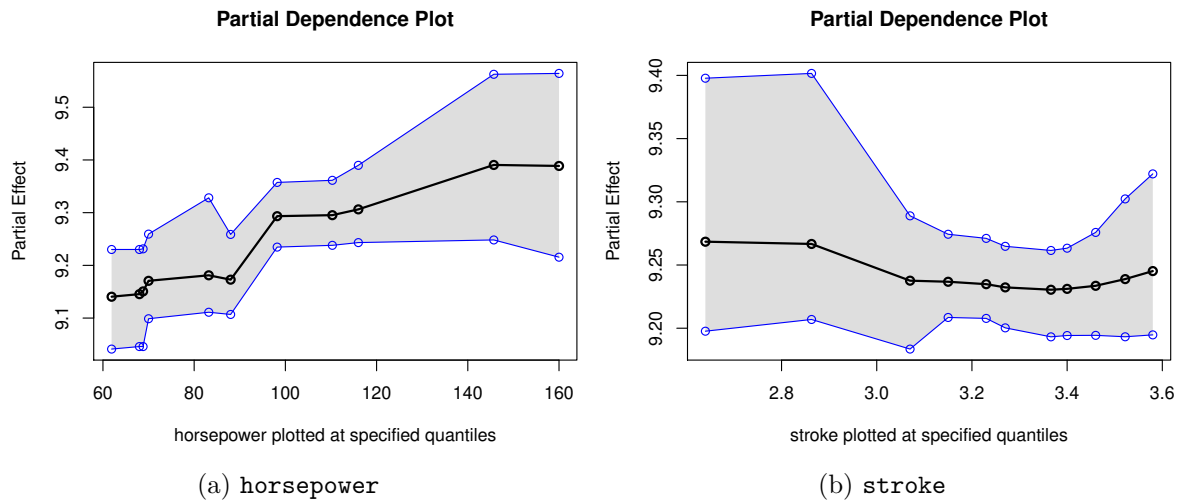
(a) `horsepower`   (b) `stroke`

Figure 8: PDPs plotted in black and 95% credible intervals plotted in blue for variables in the automobile dataset. Points plotted are at the 5%ile, 10%ile, 20%ile, ..., 90%ile and 95%ile of the values of the predictor. Lines plotted between the points approximate the PDP by linear interpolation.

```
R> pd_plot(bart_machine_cv, j = "horsepower")
R> pd_plot(bart_machine_cv, j = "stroke")
```

4.8. Incorporating missing data

The procedure for incorporating missing data was introduced in Section 3.2. We now build a **bartMachine** model using this procedure below:

```
R> y <- automobile$log_price
R> X <- automobile; X$log_price <- NULL
R> bart_machine <- bartMachine(X, y, use_missing_data = TRUE,
+    use_missing_data_dummies_as_covars = TRUE)
```

The model output below parallels the model output with the missing rows omitted (Section 4.2).

```
R> bart_machine

bartMachine v1.2.1 for regression

Missing data feature ON
training data n = 201 and p = 50
built in 1.4 secs on 1 core, 50 trees, 250 burn-in and 1000 post. samples

sigsq est for y beforehand: 0.016
avg sigsq estimate after burn-in: 0.00939
```

```
in-sample statistics:
 L1 = 11.49
 L2 = 1.04
 rmse = 0.07
 Pseudo-Rsq = 0.9794
p-val for shapiro-wilk test of normality of residuals: 0.69814
p-val for zero-mean noise: 0.96389
```

Note that the output reflects the use of the complete data set. There are 41 observations now included for which there are missing features. Also note that p has now increased from 41 to 50. The nine "new" predictors are:

```
[1] "engine_location_rear" "engine_type_rotor"    "fuel_system_4bbl"
[4] "fuel_system_spfi"     "M_normalized_losses"  "M_bore"
[7] "M_stroke"             "M_horsepower"         "M_peak_rpm"
```

The first two predictors are two new levels for the variable `engine_location` which appear in the 41 rows with missingness. The next two predictors are two new levels for the variable `fuel_system` which appear in the 41 rows with missingness as well. The last five new predictors are dummy variables which indicate missingness constructed from the predictors which exhibited missingness (due to the `use_missing_data_dummies_as_covars` parameter being set to true).

The procedure of Section 3.2 also natively incorporates missing data during prediction. Missingness will yield larger credible intervals. In the example below, we suppose that the `curb_weight` and `symboling` values were suddenly unavailable for the 20th automobile and we observe their credible intervals widening as a result.

```
R> x_star <- X[20, ]
R> calc_credible_intervals(bart_machine, x_star, ci_conf = 0.95)

     ci_lower_bd ci_upper_bd
[1,]    8.650093    8.824515


R> is.na(x_star[c("curb_weight", "symboling")]) <- TRUE
R> calc_credible_intervals(bart_machine, x_star, ci_conf = 0.95)

     ci_lower_bd ci_upper_bd
[1,]    8.622582    8.978313
```

4.9. Variable selection

In this section we demonstrate the variable selection procedures introduced in Section 3.3 and developed in detail in Bleich *et al.* (2014). The following code will select variables based on the three thresholds and also displays the plot in Figure 9.[5]

---

[5]By default, variable selection is performed individually on dummy variables for a factor. The variable selection procedures return the permutation distribution and an aggregation of the dummy variables' inclusion proportions can allow for variable selection to be performed on an entire factor.
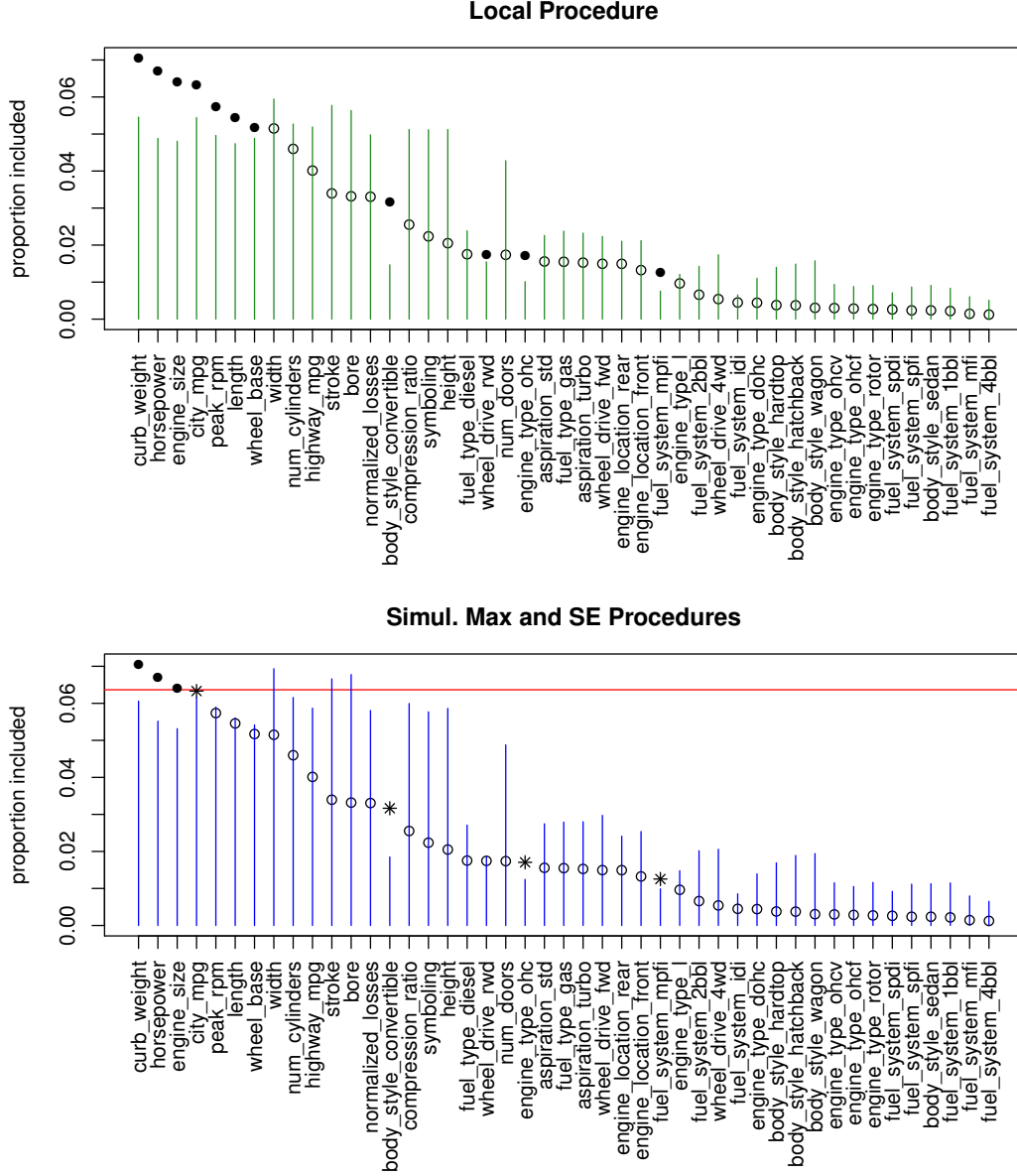
Local Procedure

Simul. Max and SE Procedures

Figure 9: Visualization of the three variable selection procedures outlined in Section 3.3 with $\alpha = 0.05$. The top plot illustrates the "Local" procedure. The green lines are the threshold levels determined from the permutation distributions that must be exceeded for a variable to be selected. The plotted points are the variable inclusion proportions for the observed data (averaged over five duplicate **bartMachine** models). If the observed value is higher than the green bar, the variable is included and is displayed as a solid dot; if not, it is not included and it is displayed as an open dot. The bottom plot illustrates both the "Global SE" and "Global Max" thresholds. The red line is the cutoff for "Global Max" and variables passing this threshold are displayed as solid dots. The blue lines represent the thresholds for the "Global SE" procedure. Variables that exceed this cutoff but not the "Global Max" threshold are displayed as asterisks. Open dots exceed neither threshold.

```
R> vs <- var_selection_by_permute(bart_machine,
+    bottom_margin = 10, num_permute_samples = 10)
R> vs$important_vars_local_names

  "curb_weight"  "city_mpg" "engine_size"    "horsepower"
  "length"       "width"    "num_cylinders"  "body_style_convertible"
  "wheel_base"   "peak_rpm" "highway_mpg"    "wheel_drive_fwd"

R> vs$important_vars_global_max_names

  "curb_weight" "city_mpg"     "engine_size" "horsepower"  "length"

R> vs$important_vars_global_se_names

  "curb_weight"   "city_mpg"       "engine_size" "horsepower" "length"
  "width"         "num_cylinders" "wheel_base"   "wheel_drive_fwd"
```

Usually, "Global Max" and "Global SE" perform similarly, as they are both more stringent in selection. However, in many situations it will not be clear to the data analyst which threshold is most appropriate. The "best" procedure can be chosen via cross-validation on out-of-sample RMSE as follows:

```
var_selection_by_permute_response_cv(bart_machine)

$best_method
[1] "important_vars_local_names"

$important_vars_cv
 [1] "body_style_convertible" "city_mpg"               "curb_weight"
 [4] "engine_size"            "engine_type_ohc"        "horsepower"
 [7] "length"                "num_cylinders"          "peak_rpm"
[10] "wheel_base"            "wheel_drive_fwd"        "wheel_drive_rwd"
[13] "width"
```

On this dataset, the "best" approach (as defined by out-of-sample prediction error) is the "Local" procedure.

4.10. Informed prior information on covariates

Bleich *et al.* (2014) propose a method for incorporating informed prior information about the predictors into the BART model. This can be achieved by modifying the prior on the splitting rules as well as the corresponding calculations in the Metropolis-Hastings step. In particular, covariates believed to influence the response can a priori be proposed more often as candidates for splitting rules. Useful prior information can aid in both variable selection and prediction tasks. We demonstrate the impact of a correctly informed prior in the context of the Friedman (1991) function (Equation 6).

$$\boldsymbol{y} = 10\sin\left(\pi\boldsymbol{x}_1\boldsymbol{x}_2\right) + 20(\boldsymbol{x}_3 - .5)^2 + 10\boldsymbol{x}_4 + 5\boldsymbol{x}_5 + \boldsymbol{\mathcal{E}}, \qquad \boldsymbol{\mathcal{E}} \sim \mathcal{N}_n\left(\boldsymbol{0}, \sigma^2\boldsymbol{I}\right). \qquad (6)$$

To illustrate, we construct a design matrix X where the first five columns are predictors which influence the response ($x_1, \ldots, x_5$ in Equation 6) and the next 95 columns are predictors that do not affect the response.

All that is required is a specification of relative weights for each predictor. These weights are normalized internally to become probabilities. As an example, we assign ten times the weight to the 5 true covariates of the model relative to the 95 useless covariates.

```
R> p <- 5
R> p0 <- 95
R> prior <- c(rep(10, times = p), rep(1, times = p0))
```

We now sample 500 observations from the Friedman function with $\sigma = 1$ for training and another 500 observations for testing.

```
R> gen_friedman_data = function(n, p, sigma){
+    if (p < 5){stop("p must be greater than or equal to 5")}
+    X <- matrix(runif(n * p), nrow = n, ncol = p)
+    y <- 10 * sin(pi * X[, 1] * X[, 2]) + 20 * (X[, 3] - .5)^2 +
+      10 * X[, 4] + 5 * X[, 5] + rnorm(n, 0, sigma)
+    data.frame(y, X)
+  }
R> ntrain <- 500
R> sigma <- 1
R> fr_data <- gen_friedman_data(ntrain, p + p0, sigma)
R> y <- fr_data$y
R> X <- fr_data[, 2:101]
R> ntest <- 500
R> fr_data <- gen_friedman_data(ntest, p + p0, sigma)
R> Xtest <- fr_data[, 2:101]
R> ytest <- fr_data$y
```

We construct a default **bartMachine** model as well as a **bartMachine** model with the informed prior and compare their performance using the RMSE metric on a test set of another 500 observations.

```
R> bart_machine <- bartMachine(X, y)
R> bart_machine_informed <- bartMachine(X, y, cov_prior_vec = prior)
R> bart_predict_for_test_data(bart_machine, Xtest, ytest)$rmse

[1] 1.661159

R> bart_predict_for_test_data(bart_machine_informed, Xtest, ytest)$rmse

[1] 1.232925
```

There is a substantial improvement in out-of-sample predictive performance when a properly informed prior is used.

Note that by default the prior vector down-weights the indicator variables that result from dummifying factors so that the total set of dummy variables has the same weight as a continuous covariate.

4.11. Interaction effect detection

In Section 4.5, we explored using variable inclusion proportions to understand the relative influences of different covariates. A similar procedure can be carried out for examining interaction effects within a BART model. This question was initially explored in Damien, Dellaportas, Polson, and Stephens (2013) where an interaction was considered to exist between two variables if they both appeared in at least one splitting rule in a given tree. We refine the definition of an interaction as follows.

We first begin with a $p \times p$ matrix of zeroes. Within a given tree, for each split rule variable j, we look at all split rule variables of child nodes, k, and we increment the j, k element of the matrix. Hence variables are considered to interact in a given tree *only if* they appear together in a contiguous downward path from the root node to a terminal node. Note that a variable may interact with itself (when fitting a linear effect, for instance). Since there is no order between the parent and child, we then add the j, k counts together with the k, j counts (if $j \neq k$). Summing across trees and MCMC iterations gives the total number of interactions for each pair of variables from which relative importance can be assessed.

We demonstrate interaction detection on the Friedman function using 10 covariates using the code below:

```
R> interaction_investigator(bart_machine, num_replicates_for_avg = 25,
+    num_var_plot = 10, bottom_margin = 5)
```

Figure 10 shows the ten most important interactions in the model. The illustration is averaged over 25 model constructions to obtain stable estimates across many posterior modes in the sum-of-trees distribution. Notice that the interaction between $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ dominates all other interaction terms, as **bartMachine** is correctly capturing the single true interaction effect: the $\sin(\pi \boldsymbol{x}_1 \boldsymbol{x}_2)$ term in Equation 6. Choosing which of these interactions *significantly* affect the response is not addressed in this paper. The methods suggested in Section 3.3 may be applicable here and we consider this to be fruitful future work.

4.12. bartMachine model persistence across R sessions

A convenient feature of **bartMachine** is its ability to "serialize" the model. Serialization allows the user to construct models and have them persist across R sessions. The cost is time during model creation and hard drive space. Thus, the serialize feature is defaulted to "off." We demonstrate the serialize feature using the code below:

```
R> bart_machine <- bartMachine(X, y, serialize = TRUE)
R> save.image("bart_demo.RData")
R> q("no")
```

Start a new R session:

```
R> options(java.parameters = "-Xmx2500m")
R> library(bartMachine)
```
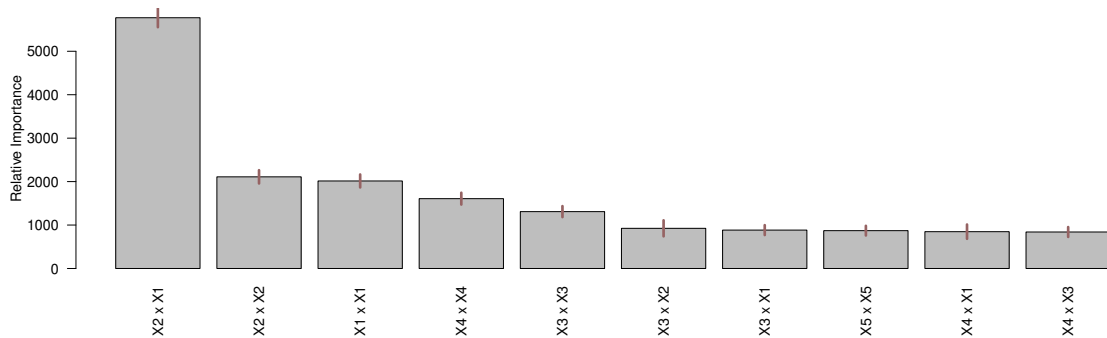
Figure 10: The top 10 average variable interaction counts (termed "relative importance") in the default **bartMachine** model for the Friedman function data averaged over 25 model constructions. The segments atop the bars represent 95% confidence intervals.

```
R> load("bart_demo.RData")
R> predict(bart_machine, X[1:4, ])
[1] 20.0954617 14.8860727 10.9483889 11.4350277
```

The training data is the same as in the previous section: $n = 100$ and $p = 10$. For the default **bartMachine** settings, $m = 50$, number of burn-in MCMC iterations is 250 and number of posterior samples is 1000. These settings yielded an almost instant serialization and a 2.1MB RData image file. For a more realistic dataset with $n = 5000$, $p = 1000$, $m = 100$ and 5000 posterior samples, the serialization and saving of the file took a few minutes and required 100MB.

Note that the package throws an error if the user attempts to make use of a **bartMachine** object in another session which was not serialized:

```
R> bart_machine <- bartMachine(X, y) #serialize is FALSE here
R> save.image("bart_demo.RData")
R> q("no")
```

Start a new R session:

```
R> options(java.parameters = "-Xmx2500m")
R> library(bartMachine)
R> load("bart_demo.RData")
R> predict(bart_machine, X[1:4, ])
```

```
Error in check_serialization(object) :
  This bartMachine object was loaded from an R image but was not serialized.
  Please build bartMachine using the option "serialize = TRUE" next time.
```

5. bartMachine package features for classification

In this section we highlight the features that differ from **bartMachine** for regression when the response is dichotomous and **bartMachine** for classification is employed. The illustrative dataset consists of 332 Pima Indians obtained from the UCI Machine Learning Repository. Of the 332 subjects, 109 were diagnosed with diabetes, the binary response variable. There are seven continuous predictors which are body metrics such as blood pressure, glucose concentration, etc. and there is no missing data.

Building a **bartMachine** model for classification has the same computing parameters except that q, ν cannot be specified since there is no longer a prior on σ^2 (see Section 2.3). We first build a cross-validated model below.

```
R> data("Pima.te", package = "MASS")
R> X <- data.frame(Pima.te[, -8])
R> y <- Pima.te[, 8]
R> bart_machine_cv <- bartMachineCV(X, y)

... bartMachine CV win: k: 3 m: 50

R> bart_machine_cv

bartMachine v1.2.1 for regression

training data n = 332 and p = 7
built in 0.7 secs on 4 cores, 50 trees, 250 burn-in and 1000 post. samples

confusion matrix:

          predicted No predicted Yes model errors
actual No      209.000        14.000        0.063
actual Yes      35.000        74.000        0.321
use errors       0.143         0.159        0.148
```

Classification models have an added hyperparameter, `prob_rule_class`, which is the rule for determining if the probability estimate is great enough to be classified into the positive category. We can see above that the **bartMachine** at times predicts "NO" for true "YES" outcomes and we suffer from a 37.6% error rate for this outcome. We can try to mitigate this error by lowering the threshold to increase the number of "YES" labels predicted:

```
R> bartMachine(X, y, prob_rule_class = 0.3)

bartMachine v1.2.1 for classification

training data n = 332 and p = 7
built in 0.7 secs on 4 cores, 50 trees, 250 burn-in and 1000 post. samples

confusion matrix:
```

```
            predicted No predicted Yes model errors
actual No        176.000        47.000        0.211
actual Yes        13.000        96.000        0.119
use errors         0.069         0.329        0.181
```

This lowers the model error to 10% for the "YES" class, but at the expense of increasing the error rate for the "NO" class. We encourage the user to cross-validate this rule based on an appropriate objective function for the problem at hand.

We can also check out-of-sample statistics:

```
R> oos_stats <- k_fold_cv(X, y, k_folds = 10)
R> oos_stats$confusion_matrix
```

```
            predicted No predicted Yes model errors
actual No        197.000        26.000        0.117
actual Yes        44.000        65.000        0.404
use errors         0.183         0.286        0.211
```

Note that it is possible to predict both class labels and probability estimates for given observations:

```
R> round(predict(bart_machine_cv, X[1 : 2, ], type = "prob"), 3)
```

```
[1] 0.599 0.100
```

```
R> predict(bart_machine_cv, X[1:2, ], type = "class")
```

```
[1] Yes No
Levels: No Yes
```

When using the covariate tests of Section 4.6, total misclassification error becomes the statistic of interest instead of Pseudo-R^2. The p value is calculated now as the proportion of null samples with *lower* misclassification error. Figure 11 illustrates the test showing that predictor `age` seems to matter in the prediction of `Diabetes`, controlling for other predictors.

The partial dependence plots of Section 4.7 are now scaled as probit of the probability estimate. Figure 12 is generated via

```
R> pd_plot(bart_machine_cv, j = "glu")
```

illustrates that as glucose increases, the probability of contracting `Diabetes` increases linearly on a probit scale.

Credible intervals are implemented for classification **bartMachine** and are displayed on the probit scale. Note that the prediction intervals of Section 4.4 do not exist for classification since $\mathcal{E}$ noise is not part of the probit model.

```
R> round(calc_credible_intervals(bart_machine_cv, X[1:2, ]), 3)
```

bartMachine test for importance of covariate(s): age
Null Samples of Misclassification Errors
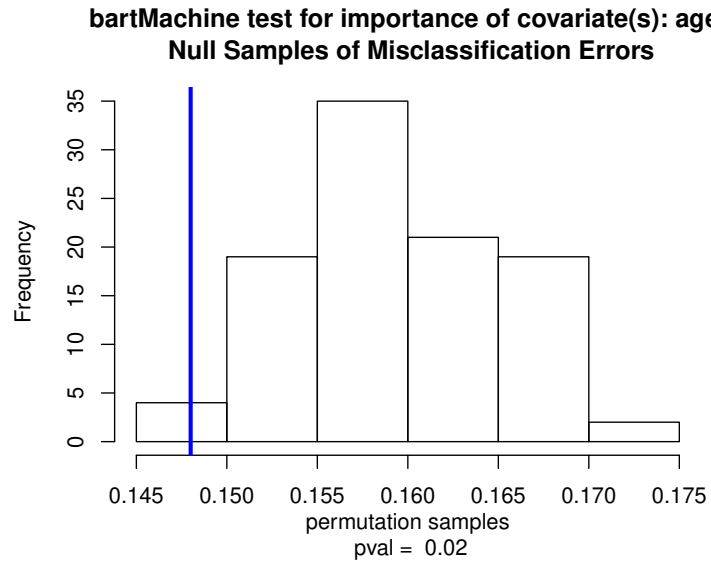
Figure 11: Test of covariate importance for predictor `age` on whether or not the subject will contract `Diabetes`.

Partial Dependence Plot
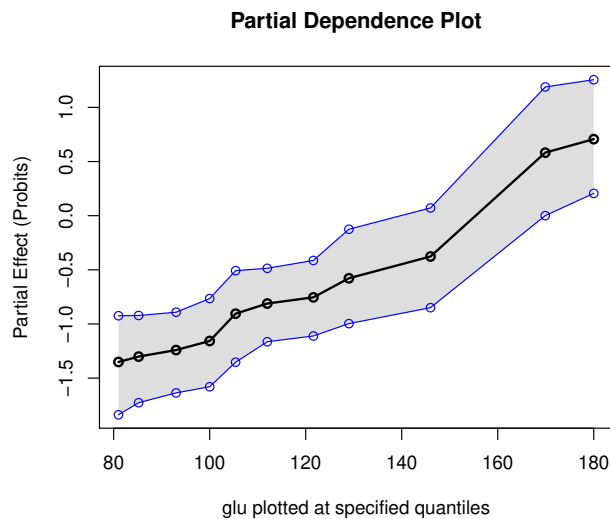
Figure 12: PDP for predictor `glu`. The blue lines are 95% credible intervals.

```
     ci_lower_bd ci_upper_bd
[1,]       0.273       0.878
[2,]       0.009       0.291
```

Other functions work similarly to regression except those that plot the responses and those that explicitly depend on RMSE as an error metric.

6. Discussion

This article introduced **bartMachine**, a new R package which implements Bayesian additive

regression trees. The goal of this package is to provide a fast, extensive and user-friendly implementation accessible to a wide range of data analysts, and increase the visibility of BART to a broader statistical audience. We hope we have provided organized, well-documented open-source code and we encourage the community to make innovations on this package.

Replication

The stable version of **bartMachine** is on CRAN and the development version is located at https://github.com/kapelner/bartMachine. The package code is under the General Public License, version 3 (GPL-3). Results, tables, and figures found in this paper can be replicated via the scripts located in the `bart_package_paper` folder within the `git` repository.

Acknowledgments

References

Albert JH, Chib S (1993). "Bayesian Analysis of Binary and Polychotomous Response Data." *Journal of the American Statistical Association*, **88**(422), 669–679. doi:10.1080/01621459.1993.10476321.

Bache K, Lichman M (2013). "UCI Machine Learning Repository." URL http://archive.ics.uci.edu/ml.

Blattenberger G, Fowles R (2014). "Avalanche Forecasting: Using Bayesian Additive Regression Trees (BART)." In *Demand for Communications Services – Insights and Perspectives*, pp. 211–227. Springer-Verlag.

Bleich J, Kapelner A, Jensen ST, George EI (2014). "Variable Selection for BART: An Application to Gene Regulation." *The Annals of Applied Statistics*, **8**(3), 1750–1781. doi:10.1214/14-aoas755.

Breiman L (2001). "Statistical Modeling: The Two Cultures." *Statistical Science*, **16**(3), 199–231. doi:10.1214/ss/1009213726.

Chipman H, McCulloch R (2016). *BayesTree: Bayesian Additive Regression Trees*. R package version 0.3-1.3, URL https://CRAN.R-project.org/package=BayesTree.

Chipman H, McCulloch R, Dorie V (2014). *dbarts: Discrete Bayesian Additive Regression Trees Sampler*. R package version 0.8-5, URL https://CRAN.R-project.org/package=dbarts.

Chipman HA, George EI, McCulloch RE (2010). "BART: Bayesian Additive Regressive Trees." *The Annals of Applied Statistics*, **4**(1), 266–298. `doi:10.1214/09-aoas285`.

Damien P, Dellaportas P, Polson N, Stephens D (2013). *Bayesian Theory and Applications.* 1st edition. Oxford University Press.

Ding Y, Simonoff JS (2010). "An Investigation of Missing Data Methods for Classification Trees Applied to Binary Response Data." *Journal of Machine Learning Research*, **11**, 131–170.

Eliashberg J (2010). *Green-Lighting Movie Scripts: Revenue Forecasting and Risk Management.* Ph.D. thesis, University of Pennsylvania.

Friedman J (2001). "Greedy Function Approximation: A Gradient Boosting Machine." *The Annals of Statistics*, **29**(5), 1189–1232. `doi:10.1214/aos/1013203451`.

Friedman JH (1991). "Multivariate Adaptive Regression Splines." *The Annals of Statistics*, **19**, 1–67. `doi:10.1214/aos/1176347963`.

Friedman JH (2002). "Stochastic Gradient Boosting." *Computational Statistics & Data Analysis*, **38**(4), 367–378. `doi:10.1016/s0167-9473(01)00065-2`.

Gelman A, Carlin JB, Stern HS, Rubin DB (2004). *Bayesian Data Analysis.* 2nd edition. Chapman & Hall / CRC.

Geman S, Geman D (1984). "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images." *IEEE Transaction on Pattern Analysis and Machine Intelligence*, **6**, 721–741. `doi:10.1109/tpami.1984.4767596`.

Hastie T, Tibshirani R (2000). "Bayesian Backfitting." *Statistical Science*, **15**(3), 196–213. `doi:10.1214/ss/1009212815`.

Hastings WK (1970). "Monte Carlo Sampling Methods Using Markov Chains and Their Applications." *Biometrika*, **57**(1), 97–109. `doi:10.2307/2334940`.

Kapelner A, Bleich J (2015). "Prediction with Missing Data via Bayesian Additive Regression Trees." *Canadian Journal of Statistics*, **43**(2), 224–239. `doi:10.1002/cjs.11248`.

Kapelner A, Bleich J (2016). "bartMachine: Machine Learning with Bayesian Additive Regression Trees." *Journal of Statistical Software*, **70**(4), 1–40. `doi:10.18637/jss.v070.i04`.

Kibler DF, Aha DW, Albert MK (1989). "Instance Based Prediction of Real Valued Attributes." *Computational Intelligence*, **5**, 51. `doi:10.1111/j.1467-8640.1989.tb00315.x`.

Kindo B, Wang H, Pe E (2013). "MBACT – Multiclass Bayesian Additive Classification Trees." arXiv:1309.7821 [stat.ML], URL `https://arxiv.org/abs/1309.7821v1`.

Liaw A, Wiener M (2002). "Classification and Regression by **randomForest**." *R News*, **2**(3), 18–22. URL `https://CRAN.R-project.org/doc/Rnews/`.

Pratola MT, Chipman H, Higdon D, McCulloch R, Rust W (2013). "Parallel Bayesian Additive Regression Trees." *Technical report*, University of Chicago.

R Core Team (2016). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Taddy MA, Gramacy RB, Polson NG (2011). "Dynamic Trees for Learning and Design." *Journal of the American Statistical Association*, **106**(493), 109–123. doi:10.1198/jasa.2011.ap09769.

Twala B, Jones M, Hand DJ (2008). "Good Methods for Coping with Missing Data in Decision Trees." *Pattern Recognition Letters*, **29**(7), 950–956. doi:10.1016/j.patrec.2008.01.010.

Urbanek S (2013). **rJava***: Low-Level R to Java Interface.* R package version 0.9-6, URL https://CRAN.R-project.org/package=rJava.

Zhou Q, Liu J (2008). "Extracting Sequence Features to Predict Protein-DNA Interactions: A Comparative Study." *Nucleic Acids Research*, **36**(12), 4137–4148. doi:10.1093/nar/gkn361.

A. Sampling to modify tree structure

This section provides details on the implementation of Equation 3 (steps 1, 3, ..., $2m - 1$), the Metropolis-Hastings step for sampling new trees. Recall from Section 2.2 that trees can be altered via growing new child nodes from an existing terminal node, pruning two terminal nodes such that their parent becomes terminal, or changing the splitting rule in a node.

Below is the Metropolis ratio (Gelman *et al.* 2004, p. 291) where the parameter sampled is the tree and the data are the responses unexplained by other trees denoted by $\boldsymbol{R}$. We denote the new, proposal tree with an asterisk and the original tree without the asterisk.

$$r = \frac{\mathbb{P}\left(\mathcal{T}_* \to \mathcal{T}\right)}{\mathbb{P}\left(\mathcal{T} \to \mathcal{T}_*\right)} \frac{\mathbb{P}\left(\mathcal{T}_* \mid \boldsymbol{R}, \sigma^2\right)}{\mathbb{P}\left(\mathcal{T} \mid \boldsymbol{R}, \sigma^2\right)}. \tag{7}$$

We accept a draw from the posterior distribution of trees if a draw from a standard uniform distribution is less than the value of r. Immediately we note that it is difficult (if not impossible) to calculate the posterior probabilities for the trees themselves. Instead, we employ Bayes' rule,

$$\mathbb{P}\left(\mathcal{T} \mid \boldsymbol{R}, \sigma^2\right) = \frac{\mathbb{P}\left(\boldsymbol{R} \mid \mathcal{T}, \sigma^2\right) \mathbb{P}\left(\mathcal{T} \mid \sigma^2\right)}{\mathbb{P}\left(\boldsymbol{R} \mid \sigma^2\right)},$$

and plug the result into Equation 7 to obtain:

$$r \quad = \quad \underbrace{\frac{\mathbb{P}\left(\mathcal{T}_* \to \mathcal{T}\right)}{\mathbb{P}\left(\mathcal{T} \to \mathcal{T}_*\right)}}_{\text{transition ratio}} \quad \times \quad \underbrace{\frac{\mathbb{P}\left(\boldsymbol{R} \mid \mathcal{T}_*, \sigma^2\right)}{\mathbb{P}\left(\boldsymbol{R} \mid \mathcal{T}, \sigma^2\right)}}_{\text{likelihood ratio}} \quad \times \quad \underbrace{\frac{\mathbb{P}\left(\mathcal{T}_*\right)}{\mathbb{P}\left(\mathcal{T}\right)}}_{\text{tree structure ratio}} \quad .$$

Note that the probability of the tree structure is independent of σ^2.

The goal of this section is to explicitly calculate r for all possible tree proposals — GROW, PRUNE and CHANGE. For each proposal, the calculations are organized into separate sections detailing each of the three ratios — transition, likelihood and tree structure. Note that our actual implementation uses the following expressions in log form for numerical accuracy.

A.1. Grow proposal

Transition ratio

Transitioning from the original tree to a new tree involves growing two child nodes from a current terminal node:

$$\begin{aligned} \mathbb{P}\left(\mathcal{T} \to \mathcal{T}_*\right) &= \mathbb{P}\left(\text{GROW}\right) \mathbb{P}\left(\text{selecting } \eta \text{ to grow from}\right) \times \\ &\quad \mathbb{P}\left(\text{selecting the } j\text{th attribute to split on}\right) \times \\ &\quad \mathbb{P}\left(\text{selecting the } i\text{th value to split on}\right) \\ &= \mathbb{P}\left(\text{GROW}\right) \frac{1}{b} \frac{1}{p_{\text{adj}}(\eta)} \frac{1}{n_{j \cdot \text{adj}}(\eta)}. \end{aligned} \tag{8}$$

We chose one of the current b terminal nodes which we denote the ηth node, and then we pick an attribute and split point. $p_{\text{adj}}(\eta)$ denotes the number of predictors left available to

split on. This can be less than p if certain predictors do not have two or more unique values once the data reaches the ηth node. For example, this regularly occurs if a dummy variable was split on in some node higher up in the lineage. $n_{j\cdot\text{adj}}(\eta)$ denotes the number of *unique* values left in the pth attribute after adjusting for parents' splits.

Transitioning from the new tree back to the original tree involves pruning that node:

$$\mathbb{P}\left(\mathcal{T}_* \to \mathcal{T}\right) = \mathbb{P}\left(\text{PRUNE}\right)\mathbb{P}\left(\text{selecting } \eta \text{ to prune from}\right) = \mathbb{P}\left(\text{PRUNE}\right)\frac{1}{w_2^*},$$

where w_2^* denotes the number of second generation internal nodes (nodes with two terminal child nodes) in the new tree. Thus, the full transition ratio is:

$$\frac{\mathbb{P}\left(\mathcal{T}_* \to \mathcal{T}\right)}{\mathbb{P}\left(\mathcal{T} \to \mathcal{T}_*\right)} = \frac{\mathbb{P}\left(\text{PRUNE}\right)}{\mathbb{P}\left(\text{GROW}\right)}\frac{b\; p_{\text{adj}}(\eta)\; n_{j\cdot\text{adj}}(\eta)}{w_2^*}.$$

Note that when there are no variables with more than two or more unique values, the probability of GROW is set to zero and the step will be automatically rejected.

Likelihood ratio

To calculate the likelihood, the tree structure determines which responses fall into which of the b terminal nodes. Thus,

$$\mathbb{P}\left(R_1, \ldots, R_n \mid \mathcal{T}, \sigma^2\right) = \prod_{\ell=1}^{b} \mathbb{P}\left(R_{\ell_1}, \ldots, R_{\ell_{n_\ell}} \mid \sigma^2\right),$$

where each term on the right hand side is the probability of responses in one of the b terminal nodes, which are independent by assumption. The R_ℓ's denote the data in the ℓth terminal node and where n_ℓ denotes how many observations are in each terminal node and $n = \sum_{\ell=1}^{b} n_\ell$. We now find an analytic expression for the node likelihood term. Remember, if the mean in each terminal node, which we denote μ_ℓ, was known, then we would have $R_{\ell_1}, \ldots, R_{\ell_{n_\ell}} \mid \mu_\ell \sigma^2 \overset{\text{iid}}{\sim} \mathcal{N}\left(\mu_\ell,\, \sigma^2\right)$. BART requires μ_ℓ to be integrated out, allowing the Gibbs sampler in Equation 3 to avoid dealing with jumping between continuous spaces of varying dimensions (Chipman *et al.* 2010, p. 275). Recall that one of the BART model assumptions is a prior on the average value of $\mu \sim \mathcal{N}\left(0,\, \sigma_\mu^2\right)$ and thus,

$$\mathbb{P}\left(R_{\ell_1}, \ldots, R_{\ell_{n_\ell}} \mid \sigma^2\right) = \int_{\mathbb{R}} \mathbb{P}\left(R_{\ell_1}, \ldots, R_{\ell_{n_\ell}} \mid \mu_\ell, \sigma^2\right) \mathbb{P}\left(\mu_\ell; \sigma_\mu^2\right) d\mu_\ell,$$

which can be shown via completion of the square or convolution to be

$$\mathbb{P}\left(R_{\ell_1}, \ldots, R_{\ell_{n_\ell}} \mid \sigma^2\right) = \frac{1}{(2\pi\sigma^2)^{n_\ell/2}}\sqrt{\frac{\sigma^2}{\sigma^2 + n_\ell \sigma_\mu^2}} \times \tag{9}$$
$$\exp\left(-\frac{1}{2\sigma^2}\left(\sum_{i=1}^{n_\ell}\left(R_{\ell_i} - \bar{R}_\ell\right)^2 - \frac{\bar{R}_\ell^2 n_\ell^2}{n_\ell + \frac{\sigma^2}{\sigma_\mu^2}} + n_\ell \bar{R}_\ell^2\right)\right),$$

where $\bar{R}_\ell$ denotes the mean response in the node and R_{ℓ_i} denotes the observations $i = 1 \ldots n_\ell$ in the node.

Since the likelihoods are solely determined by the terminal nodes, the proposal tree differs from the original tree by only the selected node to be grown, denoted by ℓ, which becomes two children after the GROW step denoted by ℓ_L and ℓ_R. Hence, the likelihood ratio becomes:

$$\frac{\mathbb{P}\left(\boldsymbol{R} \mid \mathcal{T}_*, \sigma^2\right)}{\mathbb{P}\left(\boldsymbol{R} \mid \mathcal{T}, \sigma^2\right)} = \frac{\mathbb{P}\left(R_{\ell_{L,1}}, \ldots, R_{\ell_{L,n_{\ell,L}}} \mid \sigma^2\right) \mathbb{P}\left(R_{\ell_{R,1}}, \ldots, R_{R,\ell_{n_{\ell,R}}} \mid \sigma^2\right)}{\mathbb{P}\left(R_{\ell_1}, \ldots, R_{\ell_{n_\ell}} \mid \sigma^2\right)}. \tag{10}$$

Plugging Equation 9 into Equation 10 three times yields the ratio for the GROW step:

$$\sqrt{\frac{\sigma^2\left(\sigma^2 + n_\ell \sigma_\mu^2\right)}{\left(\sigma^2 + n_{\ell_L}\sigma_\mu^2\right)\left(\sigma^2 + n_{\ell_R}\sigma_\mu^2\right)}} \; \exp\left(\frac{\sigma_\mu^2}{2\sigma^2}\left(\frac{\left(\sum_{i=1}^{n_{\ell_L}} R_{\ell_L,i}\right)^2}{\sigma^2 + n_{\ell_L}\sigma_\mu^2} + \frac{\left(\sum_{i=1}^{n_{\ell_R}} R_{\ell_R,i}\right)^2}{\sigma^2 + n_{\ell_R}\sigma_\mu^2} - \frac{\left(\sum_{i=1}^{n_\ell} R_{\ell,i}\right)^2}{\sigma^2 + n_\ell \sigma_\mu^2}\right)\right),$$

where n_{ℓ_L} and n_{ℓ_R} denote the number of data points in the newly grown left and right child nodes.

Tree structure ratio

In Section 2.1 we discussed the prior on the tree structure (where the splits occur) as well as the tree rules. For the entire tree,

$$\mathbb{P}\left(\mathcal{T}\right) = \prod_{\eta \in H_{\text{terminals}}} \left(1 - \mathbb{P}_{\text{SPLIT}}\left(\eta\right)\right) \prod_{\eta \in H_{\text{internals}}} \mathbb{P}_{\text{SPLIT}}\left(\eta\right) \prod_{\eta \in H_{\text{internals}}} \mathbb{P}_{\text{RULE}}\left(\eta\right),$$

where $H_{\text{terminals}}$ denotes the set of terminal nodes and $H_{\text{internals}}$ denotes the internal nodes.

Recall that the probability of splitting on a given node η is $\mathbb{P}_{\text{SPLIT}}\left(\eta\right) = \alpha/\left(1 + d_\eta\right)^\beta$. The probability is controlled by two hyperparameters, α and β, and d_η is the depth (number of parent generations) of node η. When assigning a rule, recall that BART picks from all available attributes and then from all available unique split points. Using the notation from the transition ratio section, $\mathbb{P}_{\text{RULE}}\left(\eta\right) = 1/p_{\text{adj}}(\eta) \times 1/n_{j\cdot\text{adj}}(\eta)$.

Once again, the original tree features a node η that was selected to be grown. The proposal tree differs with two child nodes denoted η_L and η_R. We can now form the ratio:

$$\begin{aligned}
\frac{\mathbb{P}\left(\mathcal{T}_*\right)}{\mathbb{P}\left(\mathcal{T}\right)} &= \frac{\left(1 - \mathbb{P}_{\text{SPLIT}}\left(\eta_L\right)\right)\left(1 - \mathbb{P}_{\text{SPLIT}}\left(\eta_R\right)\right)\mathbb{P}_{\text{SPLIT}}\left(\eta\right)\mathbb{P}_{\text{RULE}}\left(\eta\right)}{\left(1 - \mathbb{P}_{\text{SPLIT}}\left(\eta\right)\right)} \\[2mm]
&= \frac{\left(1 - \dfrac{\alpha}{(1+d_{\eta_L})^\beta}\right)\left(1 - \dfrac{\alpha}{(1+d_{\eta_R})^\beta}\right)\dfrac{\alpha}{(1+d_\eta)^\beta}\dfrac{1}{p_{\text{adj}}(\eta)}\dfrac{1}{n_{j\cdot\text{adj}}(\eta)}}{1 - \dfrac{\alpha}{(1+d_\eta)^\beta}} \\[2mm]
&= \alpha \frac{\left(1 - \dfrac{\alpha}{(2+d_\eta)^\beta}\right)^2}{\left((1+d_\eta)^\beta - \alpha\right)p_{\text{adj}}(\eta)n_{j\cdot\text{adj}}(\eta)}.
\end{aligned}$$

The last line follows from some algebra and using the fact that the depth of the grown nodes is the depth of the parent node incremented by one ($d_{\eta_L} = d_{\eta_R} = d_\eta + 1$).

A.2. Prune proposal

A prune proposal is the "opposite" of a grow proposal. Prune selects a singly internal node (a node whose children are both terminal) and removes both of its children. Thus, each ratio will be approximately the inverse of the ratios found in the previous section concerning the grow proposal. Note also that prune steps are not considered in trees that consist of a single root node.

Transition ratio

We begin with transitioning from the original tree to the proposal tree:

$$\mathbb{P}\left(\mathcal{T} \to \mathcal{T}_*\right) = \mathbb{P}\left(\text{PRUNE}\right) \mathbb{P}\left(\text{selecting } \eta \text{ to prune from}\right) = \mathbb{P}\left(\text{PRUNE}\right) \frac{1}{w_2},$$

where w_2 denotes the number of singly internal parent nodes which have two terminal children (thus no grandchildren). To transition in the opposite direction, we are obligated to grow from node η. This is similar to Equation 8 except the proposed tree has one less terminal node due to the pruning of the original tree, resulting in a $1/(b-1)$ term:

$$\mathbb{P}\left(\mathcal{T}_* \to \mathcal{T}\right) = \mathbb{P}\left(\text{GROW}\right) \frac{1}{b-1} \frac{1}{p_{\text{adj}}(\eta^*)} \frac{1}{n_{j^* \cdot \text{adj}}(\eta^*)}.$$

Thus, the transition ratio is:

$$\frac{\mathbb{P}\left(\mathcal{T}_* \to \mathcal{T}\right)}{\mathbb{P}\left(\mathcal{T} \to \mathcal{T}_*\right)} = \frac{\mathbb{P}\left(\text{GROW}\right)}{\mathbb{P}\left(\text{PRUNE}\right)} \frac{w_2}{(b-1)p_{\text{adj}}(\eta^*)n_{j^* \cdot \text{adj}}(\eta^*)}.$$

Likelihood ratio

This is simply the inverse of the likelihood ratio for the grow proposal:

$$\frac{\mathbb{P}\left(\boldsymbol{R} \mid \mathcal{T}_*, \sigma^2\right)}{\mathbb{P}\left(\boldsymbol{R} \mid \mathcal{T}, \sigma^2\right)} = \sqrt{\frac{\left(\sigma^2 + n_{\ell_L}\sigma_\mu^2\right)\left(\sigma^2 + n_{\ell_R}\sigma_\mu^2\right)}{\sigma^2\left(\sigma^2 + n_\ell\sigma_\mu^2\right)}} \times$$

$$\exp\left(\frac{\sigma_\mu^2}{2\sigma^2}\left(\frac{\left(\sum_{i=1}^{n_\ell} R_{\ell,i}\right)^2}{\sigma^2 + n_\ell\sigma_\mu^2} - \frac{\left(\sum_{i=1}^{n_{\ell_L}} R_{\ell L,i}\right)^2}{\sigma^2 + n_{\ell_L}\sigma_\mu^2} - \frac{\left(\sum_{i=1}^{n_{\ell_R}} R_{\ell R,i}\right)^2}{\sigma^2 + n_{\ell_R}\sigma_\mu^2}\right)\right).$$

Tree structure ratio

This is also simply the inverse of the tree structure ratio for the grow proposal:

$$\frac{\mathbb{P}\left(\mathcal{T}_*\right)}{\mathbb{P}\left(\mathcal{T}\right)} = \frac{\left((1+d_\eta)^\beta - \alpha\right)p_{\text{adj}}(\eta^*)n_{j^* \cdot \text{adj}}(\eta^*)}{\alpha\left(1 - \frac{\alpha}{(2+d_\eta)^\beta}\right)^2}.$$

A.3. Change proposal

A change proposal involves picking an internal node and changing its rule by picking both a new available predictor to split on and a new valid split value among values of the selected predictor. Although this could be implemented for use in any internal node in the tree, for simplicity we limit our implementation to singly internal nodes: those that have two terminal child nodes.

Transition ratio

The transition to a proposal tree is below:

$$
\begin{aligned}
\mathbb{P}\left(\mathcal{T}\to\mathcal{T}_*\right) \;=\; & \mathbb{P}\left(\mathrm{CHANGE}\right)\mathbb{P}\left(\text{selecting node }\eta\text{ to change}\right)\times \\
& \mathbb{P}\left(\text{selecting the new attribute to split on}\right)\times \\
& \mathbb{P}\left(\text{selecting the new value to split on}\right).
\end{aligned}
$$

When calculating the ratio, the first three terms are shared in both numerator and denominator. The probability of selecting the new value to split on will differ as different split features have different numbers of unique values available. Thus we are left with

$$
\frac{\mathbb{P}\left(\mathcal{T}_* \to \mathcal{T}\right)}{\mathbb{P}\left(\mathcal{T}\to\mathcal{T}_*\right)} = \frac{n_{j^*\cdot\mathrm{adj}}(\eta^*)}{n_{j\cdot\mathrm{adj}}(\eta)},
$$

where $n_{j^*\cdot\mathrm{adj}}(\eta^*)$ is the number of split values available under the proposal tree's splitting rule and $n_{j\cdot\mathrm{adj}}(\eta)$ is the number of split values available under the original tree's splitting rule.

Likelihood ratio

The proposal tree differs from the original tree only in the two child nodes of the selected change node. These two terminal nodes have the unexplained responses apportioned differently. Denote $R_1.$ as the residuals of the first child node and $R_2.$ as the unexplained responses in the second child node. Thus we begin with:

$$
\frac{\mathbb{P}\left(\boldsymbol{R}\mid\mathcal{T}_*,\sigma^2\right)}{\mathbb{P}\left(\boldsymbol{R}\mid\mathcal{T},\sigma^2\right)} \;=\; \frac{\mathbb{P}\left(R_{1^*,1},\ldots,R_{1^*,n_{1^*}}\mid\sigma^2\right)\mathbb{P}\left(R_{2^*,1},\ldots,R_{2^*,n_{2^*}}\mid\sigma^2\right)}{\mathbb{P}\left(R_{1,1},\ldots,R_{1,n_1}\mid\sigma^2\right)\mathbb{P}\left(R_{2,1},\ldots,R_{2,n_2}\mid\sigma^2\right)},
$$

where the responses denoted with an asterisk are the responses in the proposal tree's child nodes.

Substituting Equation 9 four times and using some algebra, the following expression is obtained for the ratio:

$$
\sqrt{\frac{\left(\frac{\sigma^2}{\sigma_\mu^2}+n_1\right)\left(\frac{\sigma^2}{\sigma_\mu^2}+n_2\right)}{\left(\frac{\sigma^2}{\sigma_\mu^2}+n_1^*\right)\left(\frac{\sigma^2}{\sigma_\mu^2}+n_2^*\right)}}\times
$$

$$
\exp\left(\frac{1}{2\sigma^2}\left(\frac{\left(\sum_{i=1}^{n_{1^*}}R_{1^*,i}\right)^2}{n_{1^*}+\frac{\sigma^2}{\sigma_\mu^2}}+\frac{\left(\sum_{i=1}^{n_{2^*}}R_{2^*,i}\right)^2}{n_{2^*}+\frac{\sigma^2}{\sigma_\mu^2}}-\frac{\left(\sum_{i=1}^{n_1}R_{1,i}\right)^2}{n_1+\frac{\sigma^2}{\sigma_\mu^2}}-\frac{\left(\sum_{i=1}^{n_2}R_{2,i}\right)^2}{n_2+\frac{\sigma^2}{\sigma_\mu^2}}\right)\right),
$$

which simplifies to

$$
\exp\left(\frac{1}{2\sigma^2}\left(\frac{\left(\sum_{i=1}^{n_{1^*}}R_{1^*,i}\right)^2-\left(\sum_{i=1}^{n_{1^*}}R_{1,i}\right)^2}{n_1+\frac{\sigma^2}{\sigma_\mu^2}}+\frac{\left(\sum_{i=1}^{n_{1^*}}R_{2^*,i}\right)^2-\left(\sum_{i=1}^{n_{1^*}}R_{2,i}\right)^2}{n_2+\frac{\sigma^2}{\sigma_\mu^2}}\right)\right),
$$

	bartMachine	**BayesTree**	**randomForest**
boston	3.003*	4.506	4.581
triazine	0.130	0.130	0.119
ozone	4.105	4.129	4.068
baseball	700.453*	713.488	730.359
wine.red	0.627*	0.653	0.641
ankara	1.303*	1.462	1.571
wine.white	0.715*	0.764	0.745
pole	11.731*	12.764	10.699
compactiv	3.261	2.820*	2.994

Table 3: RMSE values for three machine learning algorithms averaged across 20 replicates. Asterisks indicate a significant difference between **bartMachine** and **BayesTree** at a significance level of 5% with a Bonferroni correction. Comparisons with **randomForest**'s performance were not conducted.

if the number of responses in the children do not change in the proposal ($n_1 = n_1^*$ and $n_2 = n_2^*$).

Tree structure ratio

The proposal tree has the same structure as the original tree. Thus we only need to take into account the changed node's children:

$$\frac{\mathbb{P}\left(\mathcal{T}_*\right)}{\mathbb{P}\left(\mathcal{T}\right)} = \frac{\left(1 - \mathbb{P}_{\text{SPLIT}}\left(\eta_{1*}\right)\right)\left(1 - \mathbb{P}_{\text{SPLIT}}\left(\eta_{2*}\right)\right)\mathbb{P}_{\text{SPLIT}}\left(\eta_*\right)\mathbb{P}_{\text{RULE}}\left(\eta_*\right)}{\left(1 - \mathbb{P}_{\text{SPLIT}}\left(\eta_1\right)\left(1 - \mathbb{P}_{\text{SPLIT}}\left(\eta_2\right)\right)\right)\mathbb{P}_{\text{SPLIT}}\left(\eta\right)\mathbb{P}_{\text{RULE}}\left(\eta\right)}.$$

The probability of splits remains the same because the child nodes are at the same depths. Thus we only need to consider the ratio of the probability of the rules. Once again, the probability of selecting the new value to split on will differ as different split features have different numbers of unique values available. We are left with $\mathbb{P}(\mathcal{T}_*)/\mathbb{P}(\mathcal{T}) = n_{j\cdot\text{adj}}(\eta)/n_{j^*\cdot\text{adj}}(\eta^*)$.

Note that this is the inverse of the transition ratio. Hence, for the change step, only the likelihood ratio needs to be computed to determine the Metropolis-Hastings ratio r.

B. Bakeoff

We baked off nine regression data sets and assessed out-of-sample RMSE using 10-fold cross-validation. We average the results across 20 replications of cross-validation. The results are displayed in Table 3. We conclude that the implementation outlined in this paper performs approximately the same as the previous implementation with regards to predictive accuracy. Table 4 shows the average run time for each algorithm. Note that **bartMachine** is run using 4 cores.

Affiliation:

Adam Kapelner
Department of Mathematics

	bartMachine	BayesTree	randomForest
boston	7.8	28.5	5.1
triazine	5.7	10.7	2.6
ozone	4.7	17.6	2.1
baseball	5.6	18.6	3.3
wine.red	13.5	51.1	10.6
ankara	12.8	27.0	10.9
wine.white	13.5	56.0	11.0
pole	18.2	7.0	12.0
compactiv	16.3	18.4	19.2

Table 4: Average run times (in seconds) for each complete k-fold estimation for three machine learning algorithms on a Windows 7 desktop with four cores at 3.4GHz, 24GB RAM, running R 3.3.0 development build.

Queens College, City University of New York
65-30 Kissena Blvd Room 604
Flushing, NY, 11367
E-mail: kapelner@qc.cuny.edu
URL: https://kapelner.com/