BAYESIAN ADDITIVE REGRESSION TREES FOR PROBABILISTIC PROGRAMMING

Miriana Quiroga
IMASL-CONICET

Pablo G Garay
IMASL-CONICET

Juan Martin Loyola
IMASL-CONICET

Juan M. Alonso
IMASL-CONICET

Osvaldo A Martin
IMASL-CONICET*
omarti@unsl.edu.ar

June 9, 2022

ABSTRACT

Bayesian additive regression trees (BART) is a non-parametric method to approximate functions. It is a black-box method based on the sum of many trees where priors are used to regularize inference, mainly by restricting trees' learning capacity so that no individual tree is able to explain the data, but rather the sum of trees. We discuss BART in the context of probabilistic programming languages (PPLs), specifically we introduce a BART implementation extending PyMC, a Python library for probabilistic programming. We present a few examples of models that can be built using this probabilistic programming-oriented version of BART, discuss recommendations for sample diagnostics and selection of model hyperparameters, and finally we close with limitations of the current approach and future extensions.

Keywords Bayesian inference · non-parametrics · PyMC · Python · binary trees · ensemble method

1 Introduction

Bayesian Additive Regression Trees, introduced by Chipman et al. [2010], has demonstrated to be a competitive method compared with alternatives such as Gaussian processes, random forests or neural networks. The main reason is that BART needs minimal user input and tuning while maintaining a good performance [Chipman et al., 2010, Rockova and Saha, 2018, Hill et al., 2020, Sparapani et al., 2021]. Additionally, and similar to other probabilistic methods like Gaussian processes, BART provides uncertainty quantification via probable intervals.

BART has been applied to solve numerous applications in recent years including estimation of causal effects [Leonti et al., 2010, Hill, 2011, Hu et al., 2022, Steele and Schwartz, 2022, Chen et al., 2022], species distribution modelling [Carlson, 2020], estimating indoor radon concentrations [Kropat et al., 2015], modeling of asteroid diameters [de Souza et al., 2021], genomics [Li et al., 2022], etc.

Variable selection has also been a target of BART, achieving reasonable results [Bleich et al., 2014, Linero, 2018]. The main approach is based on counting how many times a covariable is incorporated in the trees relative to the other covariables in the same model. In order to improve the variable selection of the model, Linero [2018] introduced a sparsity-inducing Dirichlet hyperprior on the splitting proportions of the regression tree prior.

In the literature, it is common to find that specific BART models are associated with specific samplers. That is, general methods to sample from BART models posteriors are not common. The main reason behind this is that the sampler for BART models usually rely on conjugate priors. Nevertheless, recently, two proposals have been introduced with a focus on generalizing BART; Tan and Roy [2019] presented the general BART framework unifying BART extensions that

*Universidad Nacional de San Luis. San Luis, Argentina.

were previously presented as separated models and Linero [2022] introduced a reversible jump Markov chain Monte Carlo algorithm to bypass the need for conjugacy.

In this work, we present an implementation of BART that attempts to remove the conflation of inference and modeling. Abstracting away the implementation of inference from modeling has been key to the success of applied Bayesian modeling in recent years, and the main reason for the popularity of PPLs like PyMC [Salvatier et al., 2016, Wiecki et al., 2022]. We must clarify that our current implementation is still restricted to some models. For example, currently we do not support independent variables with more than 1 dimension, i.e., multinomial models are not allowed. But, in principle, our implementation could be extended to support those models.

For the remainder of this article, we will focus primarily on practical aspects of BART. We start by giving a brief overview of the BART model in Section 2, then we continue with Section 3 describing how to invoke BART within PyMC, and which hyperparameters are available to the users. In Section 4 we demonstrate basic usage through examples. And in Section 5 we discuss how to choose the number of trees for BART, arguably the most impactful hyperparameter of BART. Finally, we conclude with Section 6 discussing the limitations and the future of BART. While our target audience are practitioners interested in adding BART to their Bayesian toolkit, we also provide details of the PGBART sampler in Appendix A, which we hope will help others interested in contributing to the base code.

BART is implemented as a module of PyMC-experimental. This is a sister package of the probabilistic programming language PyMC, intended to serve as a place for very new or experimental methods and features. PyMC is available from the Python Package Index at `https://pypi.org/project/pymc/`. Alternatively, it can be installed using Conda. PyMC-experimental is available from `https://github.com/pymc-devs/pymc-experimental.git`. The package documentation, including installation instructions and many examples of how to use PyMC and BART to conduct different statistical analysis, can be found at `https://docs.pymc.io`.

The version of PyMC used for this article is 4.0. All analyses are supported by extensive documentation in the form of interactive Jupyter notebooks [Kluyver et al., 2016] available in the paper repository on GitHub `https://github.com/aloctavodia/BART`, enabling readers to re-run, modify, and otherwise experiment with the models described here on their own machines. This repository also includes instructions on how to set up an environment with all the dependencies used when writing this manuscript.

2 The BART model

A BART model can be represented as:

$$\mathbb{E}[Y] = \phi\left(\sum_{i=1}^{m} G_i(\boldsymbol{X}; \mathcal{T}_i, \mathcal{M}_i), \theta\right) \tag{1}$$

where $\boldsymbol{X}$ are the covariates and Y the response variable. The sum is done over m trees. Each G_i is a binary tree with structure, $\mathcal{T}_i$ i.e., the set of interior nodes (also known as splitting nodes) and the associated splitting rules, and the set of terminal nodes (also known as leaf nodes). $\mathcal{M}_i = \{\mu_{1,i}, \mu_{2,i}, \cdots, \mu_{b,i}\}$ represents the values at the terminal nodes. For an example of a single tree used for regression, see Figure 1. ϕ represents an arbitrary probability distribution, and θ other parameters from ϕ not modelled as a sum of trees, like the standard deviation for a Normal likelihood.

The model is completed by specifying priors for $\mathcal{T}$ and $\mathcal{M}$. For $\mathcal{T}$, independent priors are set for the depth of the trees, the splitting variables, and the splitting values. Details for such priors can be found in Appendix A. The overall effect of the BART priors is to prevent overfitting by making trees shallow, making leaf nodes values small on the scale of the data and regularizing statistical interactions. Additionally, the prior induces sparsity, effectively reducing the effect of spurious covariables.

We can think of BART as priors over step functions, i.e., priors over piecewise constant functions. In the limit of the number of trees $m \to \infty$, BART converges to a nowhere-differentiable Gaussian Process [Linero and Yang, 2018]. So, in principle such a prior can only approximate smooth functions, which are arguably the most common scenario. Additionally, we can only approximate such a prior. Nevertheless, BART can still be useful in practice, as judged by all its applications. Figure 2 shows an example of BART fitted to data generated from 3 simple functions, a line, a sine, and a step function. In all the examples, the sample size is 200. We can see the effect of `m` on the result; in Section 5 we provide some guidance on how to select `m`.

BART is part of the ensemble models family. These models are based on modeling a function, and making predictions, from a summary of simple models instead of from a single complex one. In the case of BART the simple models are the binary trees. Generally, ensemble models have desirable properties, like being less prone to over-fit [Zhou,

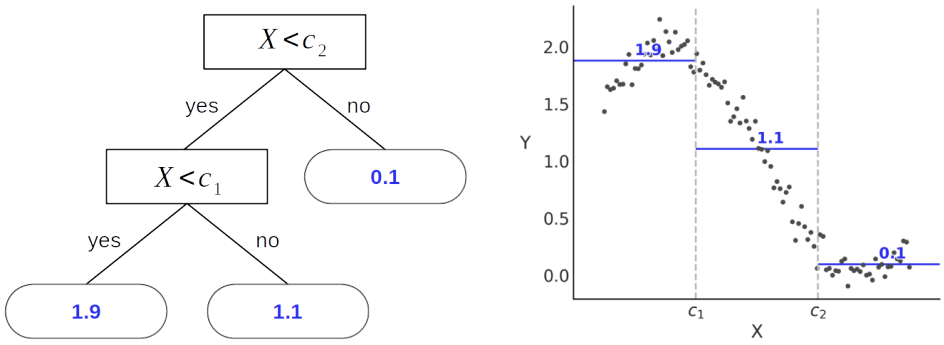

Figure 1: Example of a regression tree (left) and the induced partition of the data space (right). The interior nodes are represented with rectangles. Inside them, we can find the splitting variable X and the splitting values c_1 and c_2. The terminal nodes are represented using rounded rectangles, inside them, we find the leaf values in blue. In this example, we show a single tree fitting the data. For BART we use a sum of trees.

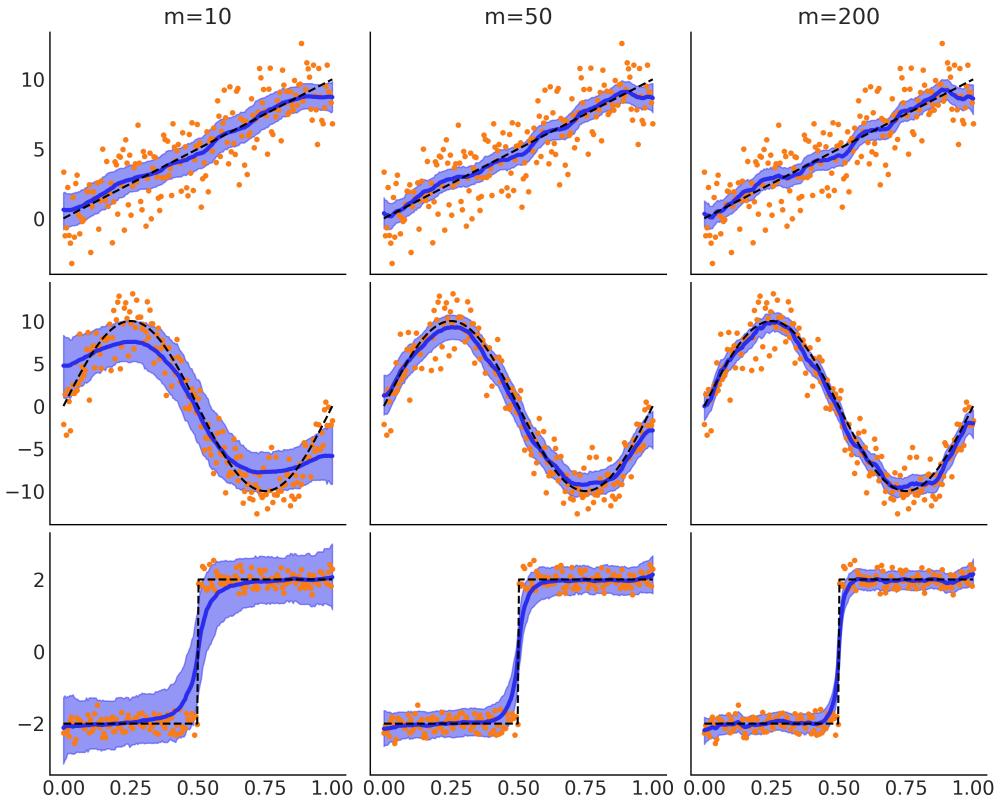

Figure 2: Examples of a BART model fitted to data (orange dots) from 3 different functions (black dashed lines) with `m`=10 (left column), `m`=50 (center column) or `m`=200 (right column).

2012, Kuhn and Johnson, 2013]. A drawback of ensemble methods, as well as other black-box methods, is that a meaningful interpretation of the details of the fitted model is difficult or even impossible; instead models are assessed via evaluations at given values of the covariates. Later, we present examples of partial dependence plots, which can be computed with tools we provide and can help interpret the results from BART models.

3 API

For those familiar with PyMC syntax, defining BART models is straightforward, see for example Code Block 1. A `pmx.BART` random variable behaves similarly to other PyMC variables, with a few caveats. Besides the name, it requires two mandatory arguments, `X`, a 2D NumPy array or pandas DataFrame representing the values of the covariates, and `Y`, a 1D NumPy array or pandas DataFrame representing the output variable. Currently, our implementation does not support a response variable of dimension larger than 1, but we are considering extending it.

Another difference is that a `pmx.BART` variable does not accept other PyMC random variables as arguments. The main reason is that, as in other implementations, the priors for the BART variables are not directly set by the users, but instead hyperparameters are used to adjust them indirectly[2]. The hyperparameters that can be changed by the user are:

- The number of trees `m`. This is a positive integer that defaults to 50. For some datasets, values as low as 20 could provide a good approximation; for others, values as high as 200 may be needed. The value of `m` can be defined using cross-validation. In our experiments, we found that Pareto Smoothed Importance Sampling Leave One Out Cross Validation PSIS-LOO-CV [Vehtari et al., 2017, 2021b], can be used to find reasonable values of `m` (See section 5).
- The value of `alpha` , controlling the node depth. It can take values in the interval $[0, 1)$. By default, this value is 0.25. It seems that there is little reason for users to change this default value, as the effect on the results is very small (see Figure 11).
- The prior over the splitting variables. This is uniform over the covariates `X`. Users can pass an array, if they have prior information about the relative importance of the variables. Each element should be in the $[0, 1]$ interval, and the elements should sum up to 1. Otherwise, they will be normalized.

PyMC is capable of automatically assigning different sampling algorithms to different parameters. Thus, PyMC will use the `pmx.PGBART` sampler, for BART, and use generic samplers for θ. If θ is a continuous parameter, then PyMC will automatically choose the `pm.NUTS` sampler [Hoffman and Gelman, 2014]. `pmx.PGBART` is a sampler we have specifically developed for BART variables, see Appendix A for details. These are the hyperparameters related to the sampler:

- `num_particles`. The number of particles used to sample a new tree. Defaults to 40. In cases where the $\hat{R}$ values [Vehtari et al., 2021a] are too high, increasing the number of particles can help.
- `batch`. Number of trees out of the `m` trees fitted per step. Defaults to "auto", which is 10% of `m` during and after tuning. Users can provide a tuple, with the first element being the batch size during tuning and the second the batch size after tuning. Increasing it can help to reduce the $\hat{R}$ but in our experience it is better to increase the number of particles.

Priors for θ, i.e., non BART related parameters, can be arbitrarily set by the user using the standard PyMC syntax.

PyMC uses ArviZ's InferenceData object [Kumar et al., 2019, Martin et al., 2022] to store posterior samples, prior/posterior predictive samples, stats generated during sampling, etc. InferenceData is a rich data structure based on Xarray [Hoyer and Hamman, 2017]. For PyMC models with a `pmx.BART` variable, the InferenceData object will also store the trees generated during sampling. These trees can be used, for example, by functions like those in `pmx.bart.utils` for example to obtain partial dependence plots as we show in Section 4. We also use ArviZ for convergence diagnostics including $\hat{R}$, effective sample size, trace plots and rank plots [Vehtari et al., 2021a, Martin et al., 2021].

[2]While this seems to work in practice, extensions could be considered, like using a prior over m, instead of a fixed number.

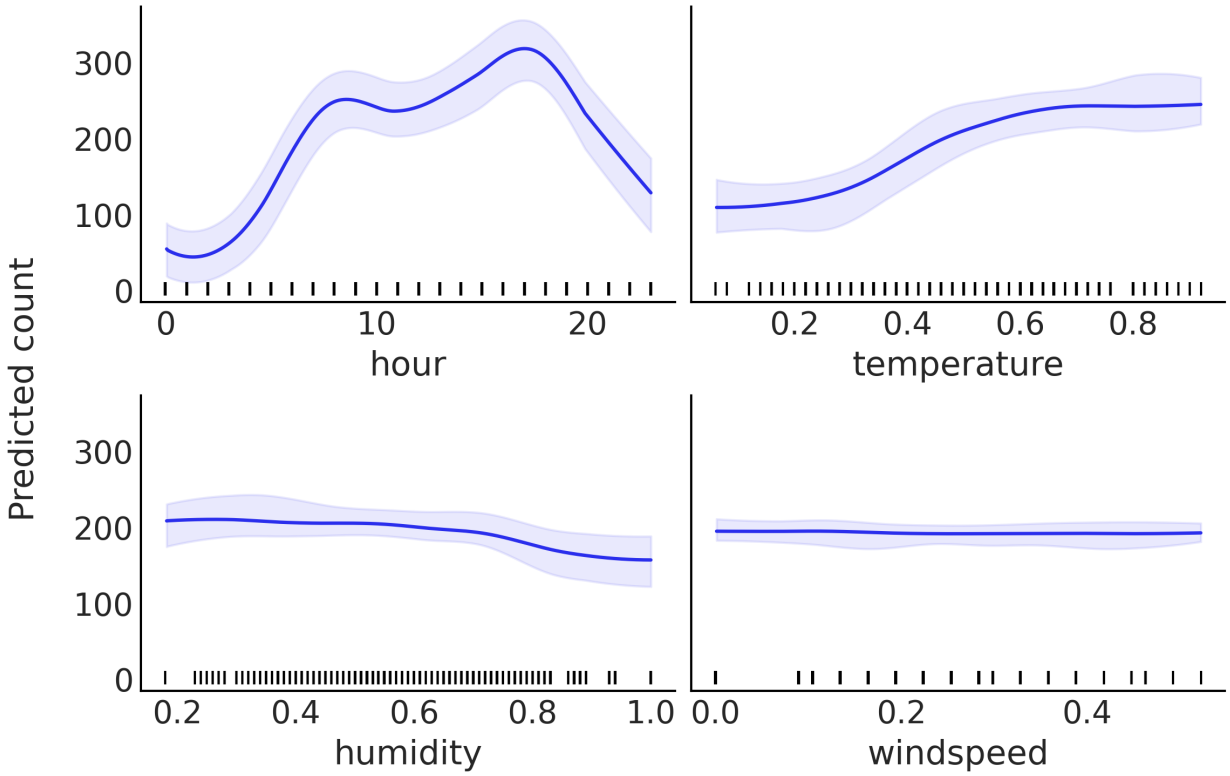

Figure 3: Partial contribution of each variable to the number of rented bikes.

4 Examples

4.1 Bikes

For our first example, we will use a dataset from the University of California Irvine's Machine Learning Repository `https://archive.ics.uci.edu/ml/datasets/bike+sharing+dataset`. As the response variable, we are going to use the number of bikes rented per hour. And for the covariates we are going to use the hour of the day, the temperature, the humidity, and the speed of the wind.

The proposed model is:

Code Block 1 PyMC model for the bikes example. We have followed the import conventions `import pymc as pm` and `import pymc_experimental as pmx`

```
with pm.Model() as model_bikes:
    μ = pmx.BART('μ', X, Y, m=50)
    σ = pm.HalfNormal('σ', Y.std())
    y = pm.Normal('y', μ, σ, observed=Y)
    idata_bikes = pm.sample()
```

Besides extending PyMC with BART random variables and the PGBART sampler, we also offer a few helper functions, one of which can be used to compute partial dependence plots [Friedman, 2001]. This allows the user to analyze the partial contribution of each variable. From Figure 3, we can conclude that:

1. The marginal contribution of the variable `hour` varies in a more complex way than for the other variables. Starting from a minimum between 0 and 3 approximately, it increases to a first peak at around 8; then it

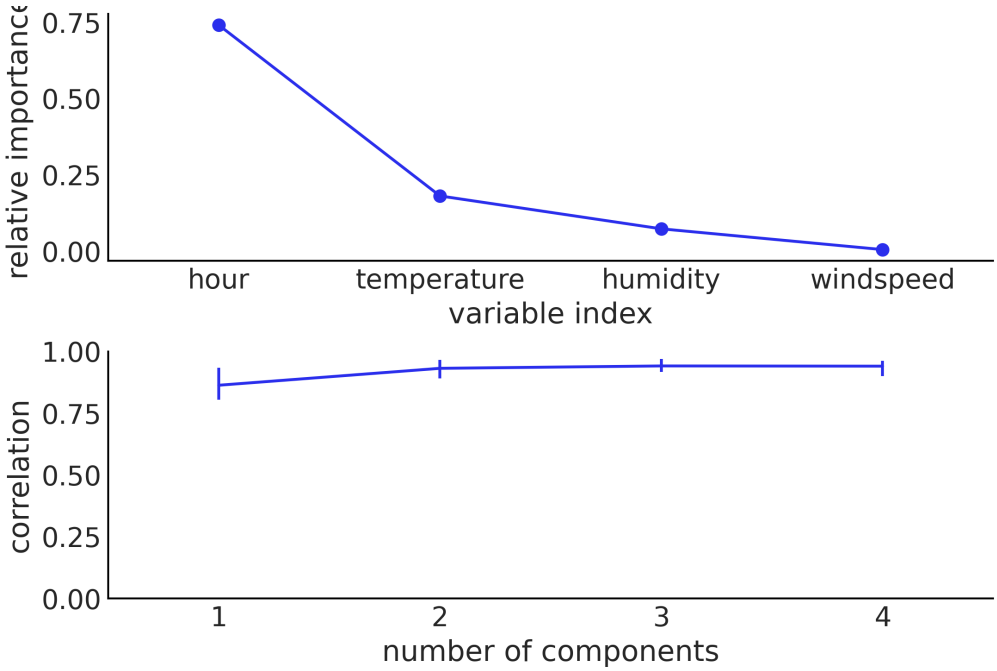

Figure 4: Variables importance (top) from each variable to the number of rented bikes, and Pearson's correlation coefficient (bottom) between BART with all variables.

decreases slightly to a new, higher peak, at around 17, and finally, it decreases. This pattern can be interpreted as a relationship between working hours and the need to rent bikes.

2. While the value of `temperature` increases, the number of rented bikes also increases, but at some point it stabilizes. This can be explained by saying that at higher temperature people are more motivated to go out, but there comes a point where this is no longer the case.
3. `Humidity` seems to contribute in a very slightly negative way, which could be interpreted as meaning that high humidity does not contribute to people wanting to ride a bike. Nevertheless, the relationship (if any) seems to be very small.
4. `Wind speed` shows a practically no contribution to the motivation to ride a bike.

Now we move our focus to the analysis of variable importance. From the top panel of Figure 4 we can see that the variables `hour` and `temperature` are the most important covariates, and that the other two are less important. Notice this is in line with the partial dependence plots from Figure 3. This kind of plot is useful to see the relative importance of a variable, but is not very useful if we want to select a subset of the variables, that is, if we want to perform variable selection.

In order to provide a variable selection procedure from the computation of variable importance, we introduce a new plot. We can see an instance in the bottom panel of Figure 4. On the x-axis we have the number of components (variables) and on the y-axis the Pearson correlation between the predictions made by the full-model (all variables are included) and the restricted-models, i.e., those with only a subset of the variables in the full-model. The components are included following the relative variable importance order, as shown in the top panel. Thus, in the bikes example, one component means `hour`, two components means `hour, temperature`, three components `hour, temperature, humidity` and four components `hour, temperature, humidity, windspeed`, i.e., the full model. Hence, from Figure 4 we can see that even a model with a single component, `hour`, is very close to the full model. Moreover, the model with the two components `hour` and `temperature` is, on average, indistinguishable from the full model. The error bars represent the 94 % Highest Density Interval (HDI) of the posterior predictive distribution, that is, of the predictions of the model.

To generate this plot, we are making two important approximations in order to reduce the computational cost:

- We do not evaluate all possible combinations of variables, we add components, one at a time, following their relative importance.

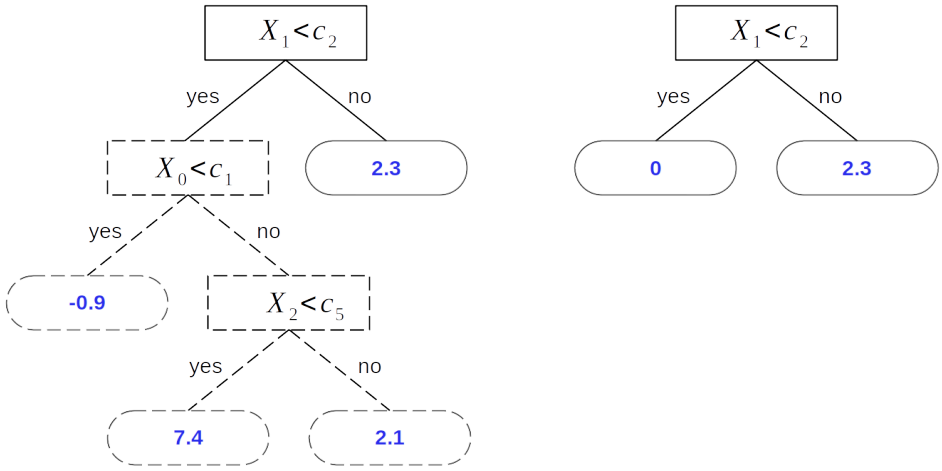

Figure 5: An example of a pruned tree during the computation of a restricted model. The tree on the left represents a tree obtained for the full model. The tree on the right represents the tree that will be used to compute a restricted model that does not include the variable X_0.

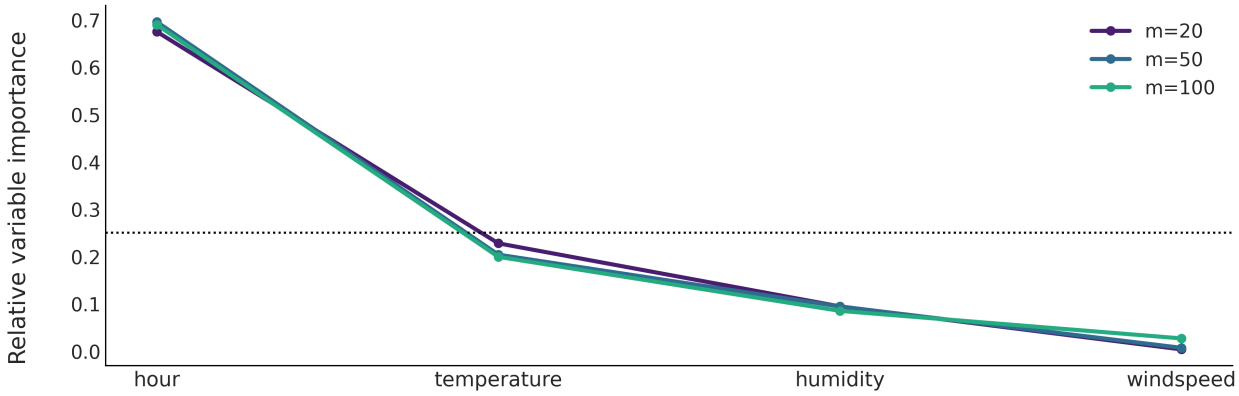

Figure 6: Variable importance for the bikes model. Hour and temperature are the most important covariates. The number of trees `m` has little effect on the computed values of the variable importance. The black dashed line represents the uniform importance variable $\left(\frac{1}{4}\right)$.

- We do not refit the model for 1 to n components; instead, we approximate the effect of removing variables by traversing trees from the posterior distribution computed for the full-model and pruning branches without the variable of interest, see Figure 5. That is, we compute predictions by replacing a splitting node with a "removed" variable with a leaf node with a value of 0. It is important to note that with this pruning procedure, we are effectively removing all variables upstream from that branch. Given that BART trees are very shallow, this approximation should be good enough.

Finally, we close this example by showing that the computation of variable importance is robust with respect to the number of trees `m`, as can be seen from Figure 6. This is due to the sparsity-inducing prior on the splitting variables, and is contrary to the original proposal by Chipman et al. [2010] where they use a low number of trees (`m`=20 or 25) for variable importance and a higher one for inference. With our implementation, it is possible to use the same value of `m` for both tasks. Thus, we recommend computing variable importance once we are sure inference is good enough for our purposes, including being sure we do not have convergence issues.

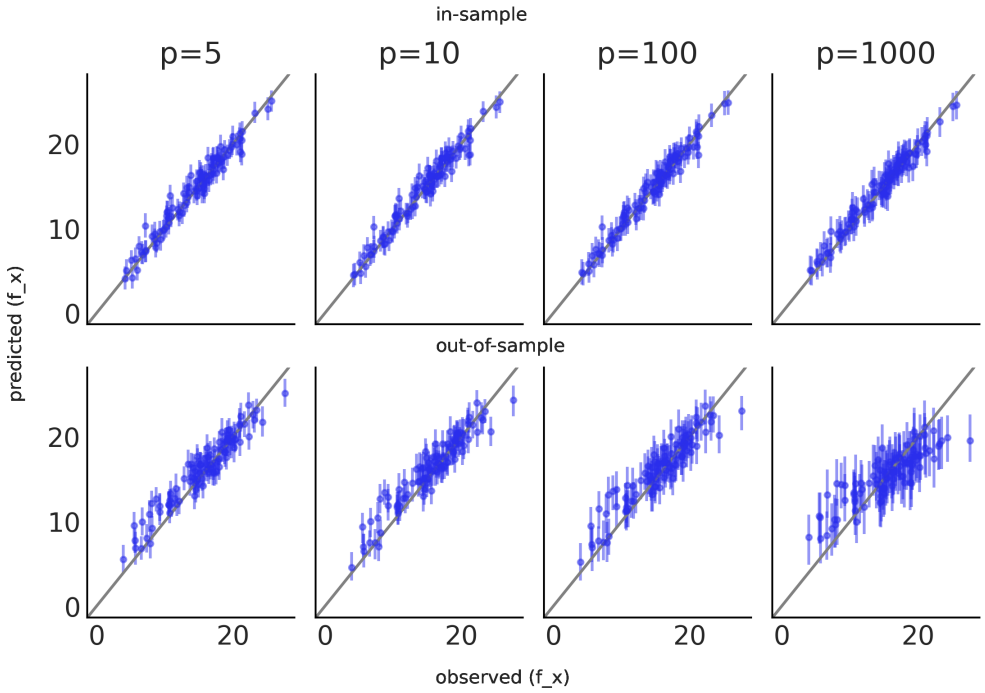

Figure 7: Inference of Friedman function with dimension $p \in \{5, 10, 100, 1000\}$

4.2 Friedman function

The second example uses the Friedman function [Chipman et al., 2010], which consists of generating data for the random variables $\boldsymbol{X} = (X_0, X_1, ..., X_p)$ where $X_0, X_1, ..., X_p$ iid $\sim \mathcal{U}(0, 1)$ and $Y = f(\boldsymbol{X}) + \epsilon = 10 \sin(\pi X_0 X_1) + 20(X_2 - 0.5)^2 + 10X_3 + 5X_4 + \epsilon$, where $\epsilon \sim N(0, 1)$.

We can see that Y only depends on the first five covariates $\boldsymbol{X}_{0:4}$. Thus, the rest of the covariates $\boldsymbol{X}_{5:p}$ are completely irrelevant. This fact, plus the nonlinearities and interactions, make finding $f(x)$ challenging for standard parametric methods, and thus a good test for BART models.

To fit the data generated with the Friedman function, we use the model in Code Block 2 with the variable `X` being an array with a variable number of features $p \in \{5, 10, 100, 1000\}$, or columns. That is, we evaluate BART for an increasing number of irrelevant features.

Code Block 2 PyMC model for the Friedman example. This model is essentially the same as that of Code Block 1

```
with pm.Model() as model_friedman:
    μ = pmx.BART('μ', X, Y, m=200)
    σ = pm.HalfNormal('σ', 1)
    y = pm.Normal('y', μ, σ, observed=Y)
    idata_friedman = pm.sample()
```

Figure 7 shows the correlation between predicted and observed data with different number of covariables $p \in \{5, 10, 100, 1000\}$, the true values are in the x-axis while the y-axis contains the in-sample predictions (top panel) and out-of-sample (bottom panel). The error bars represent the 90% HDI. The more closely the predictions are to the true function, the closer they will be to the black line at 45°. For the in-sample predictions we can see that there is a very good agreement between predicted and observed data even when the number of irrelevant features is much larger than the relevant ones. As expected, out-of-sample predictions are worse than in-sample ones. But even in this scenario, and when using a sparsifying prior, BART predictions are robust to the number of irrelevant features as previously observed Linero [2018]. We notice that for extreme cases when the number of irrelevant covariates is very

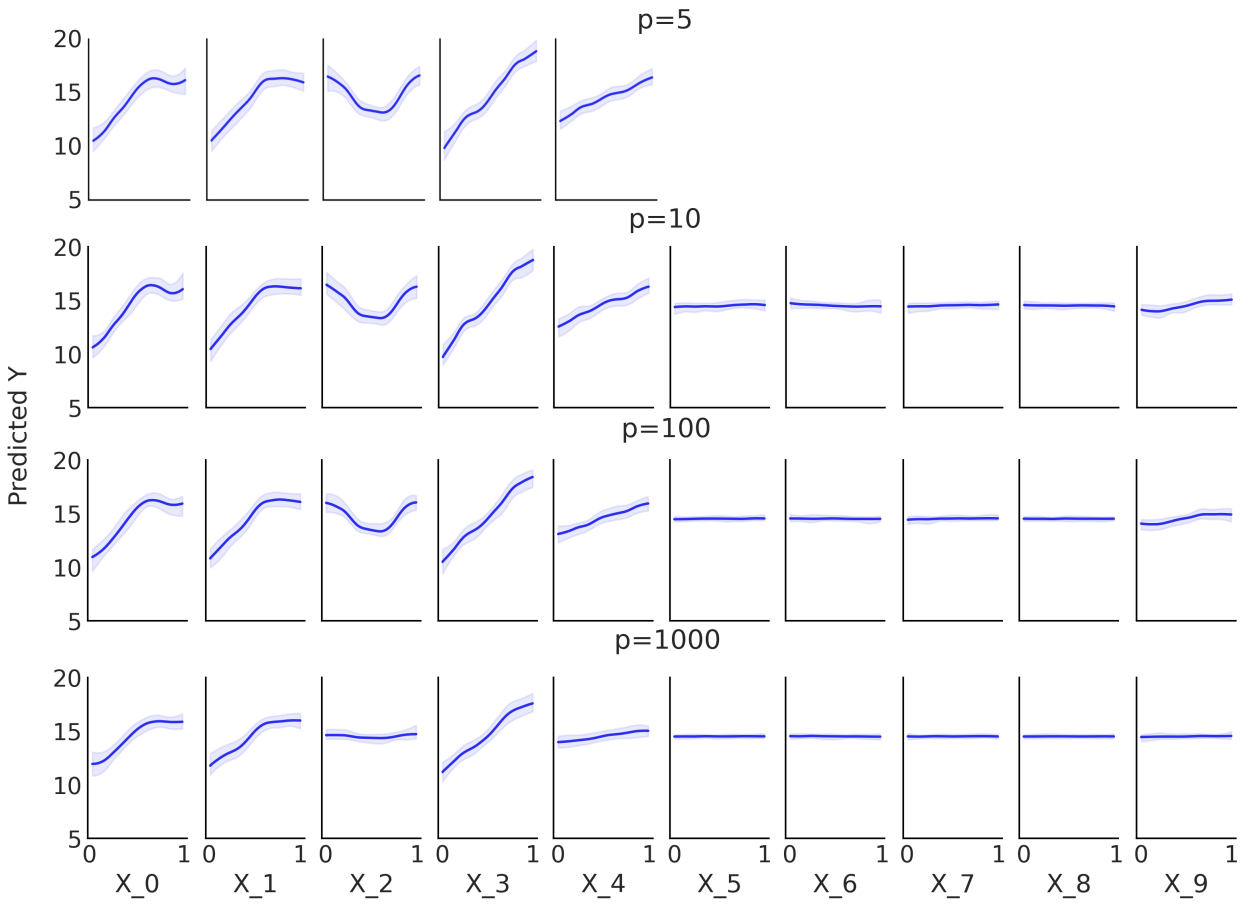

Figure 8: Partial dependence plots with a number of predictors $p \in \{5, 10, 100, 1000\}$.

large compared to the relevant ones, like when p=1000 in this example, it may be wise to do a first run to check the variable importance and if the non-relevant variables represent a very large fraction of the total number of covariates then a second run keeping only the most relevant covariates.

Figure 8 shows a partial dependence plot for the BART model in Code Block 2, with different number of covariables $p \in \{5, 10, 100, 1000\}$. We can see that the variables $\boldsymbol{X}_{5:10}$ have almost zero contribution to the response variable Y[3], while the first 5 variables $\boldsymbol{X}_{0:4}$ have a larger effect on Y. We can also see that, as we increase p, these observations are quite robust, although the response becomes flatter. This effect is more clear for X_2, specially for $p = 1000$.

Next, we compute variable importance, using $p = 10$ and the model in Code Block 2. We observe in Figure 9 that the first five variables $\boldsymbol{X}_{0:4}$ are the more important ones. This is expected from the construction of Y, as we already mentioned that the variables $\boldsymbol{X}_{5:p}$ are unrelated to the response variable Y. This is in agreement with Figure 8. Additionally, from Figure 9 we can see that variable importance is virtually insensitive to the values of `m` when `m` >= 50. With fewer trees, 10 or 20, the values of the importance variables have more dispersion, but even in that situation, the 5 first variables are the most important ones. Again, we can see a qualitative agreement with Figure 8.

Our results are similar to those of Chipman et al. [2010], with the important difference that, in our case, the variables $\boldsymbol{X}_{5:p}$ have even less importance (almost nil), which is a better result. This is in line with the results of Linero [2018], which introduced the use of a Dirichlet prior for the splitting variables.

In Figure 10 we can observe that, as the number of irrelevant features increases, the relative importance of the first five covariates decreases, as expected, because the total importance of 1 has to be distributed among more covariates. But we can also see that the relative importance is robust with respect to an increase in the number of non-relevant

[3]With the rest of the variables $\boldsymbol{X}_{10:p}$, not shown here but following the same flat pattern

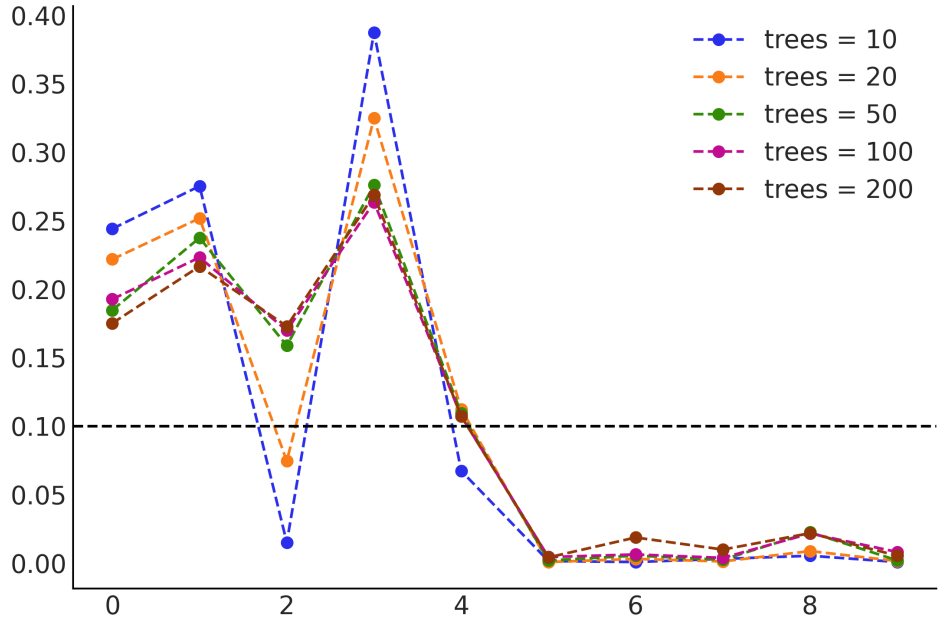

Figure 9: Relatively variable importance of $\boldsymbol{X}_{0:9}$ for number of trees `m` $\in \{10, 20, 50, 100, 200\}$. The values are normalized so that the total variable importance sums up to 1. The black dashed line represents the uniform importance variable $\left(\frac{1}{10}\right)$.

covariates, because on average the variable importance for the non-relevant covariates is smaller than for the relevant ones.

To evaluate the impact of the hyperparameter `alpha`, which controls the prior for the depth of the trees, we use the model described in Code Block 2 for $p = 10$, iterating through `m` and `alpha`, for $m \in 10, 20, 50, 100, 200$ and `alpha` $\in 0.1, 0.25, 0.5$. In Figure 11 we can see that the effect of the `alpha` parameter is overall small. Deeper trees are needed to represent higher order interactions, so the effect of `alpha` could be more profound in those cases. However, we notice that in practice, properly accounting for higher order interactions can be difficult and requires larger and larger sample size as the order of the interactions increases. Thus, regularizing trees to account for interactions of lower order should be beneficial in practice. For all these reasons, we decided to set `alpha=0.25` as the default value; this choice implies that trees deeper than 2 have a probability of 0.0625.

4.3 Cox processes

We now show how to use BART for a 1D discretized non-homogeneous Poisson process. We use the classical coal mine disaster dataset. The same example, but using Gaussian Processes, can be found in [Martin, 2018] and the documentation of GPstuff package [Vanhatalo et al., 2013].

The data consist of timestamps for when the disaster occurred. To be able to fit this data using a regression model, we bin the data as shown in Code Block 3, where the values `X` are the date of the disasters and `Y` are the number of accidents for that date. Because this is a simple function, we use `m`=20.

The BART model for this example is in Code Block 4.

In Figure 12, the blue line represents the mean accident rate, and the dark and light blue bands represent the HDI 50% and 94% respectively. A notable decrease in accidents can be observed between the years 1880 and 1900.

4.4 Heteroscedasticity

So far we have seen examples of BART used to model the mean function, but we can also use it to model a non-constant variance. To exemplify such a scenario, we are going to make use of the marketing dataset from [Kassambara, 2019]. We have the budget spent on YouTube advertisement vs the effect on sales. We decided to model the mean as a linear model, with a square root transformation, and let BART be in charge of the standard deviation. The model is:

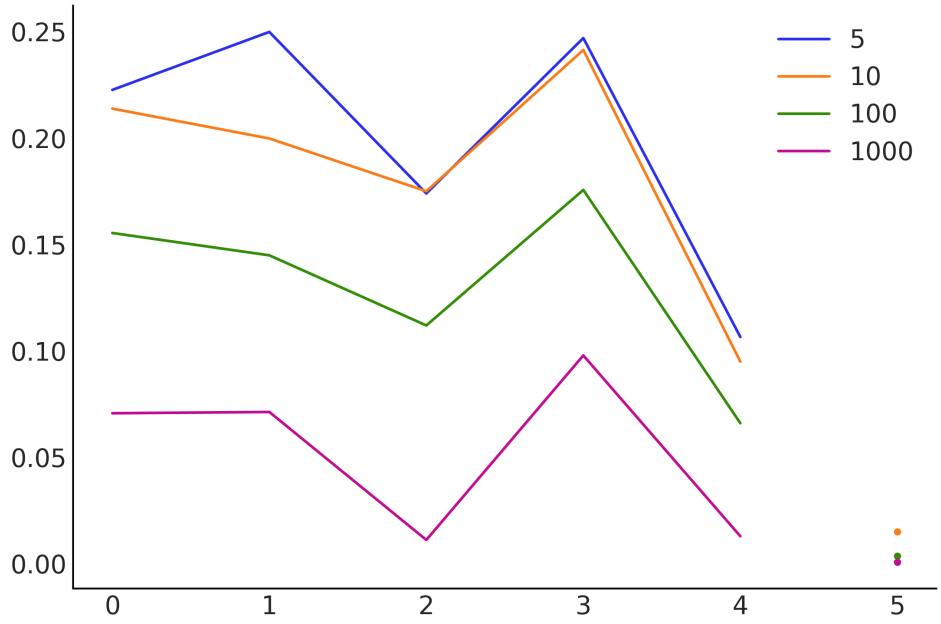

Figure 10: Relative variable importance for $p \in \{5, 10, 100, 1000\}$. The values are normalized so that the total variable importance sums up to 1. The mean of the importance of the variables $\boldsymbol{X}_{5:p}$ is represented with dots.

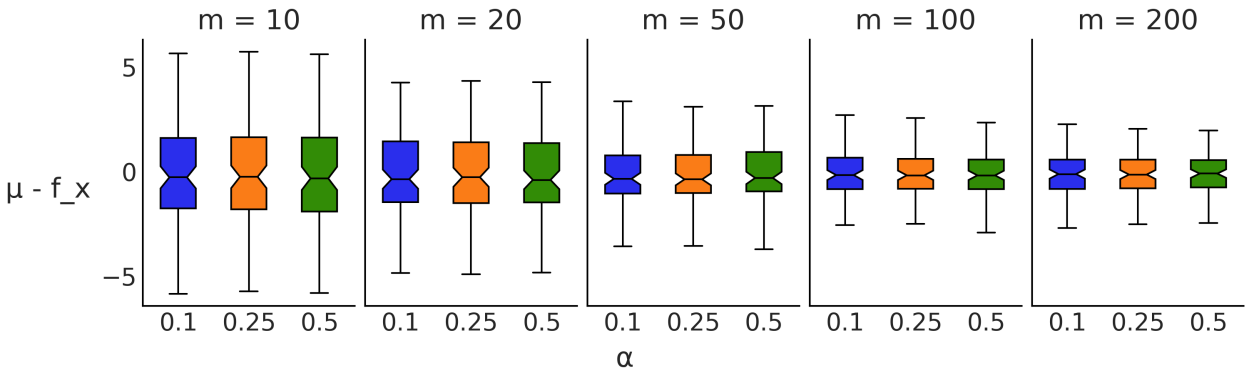

Figure 11: Box plots for the difference between the true Friedman function and the predictions for $m \in \{10, 20, 50, 100, 200\}$ in the panels from left to right. In each panel, $\alpha \in \{0.1, 0.25, 0.5\}$ is plotted in blue, orange, and green, respectively.

Code Block 3 Preprocessing of the coal dataset.

```
# discretize data
years = int(coal.max() - coal.min())
bins = years // 4
hist, x_edges = np.histogram(coal, bins=bins)
# compute the location of the centers of the discretized data
x_centers = x_edges[:-1] + (x_edges[1] - x_edges[0]) / 2
# X needs to be 2D for BART
X = x_centers[:, None]
# express data as the rate number of disaster per year
Y = hist / 4
```

Code Block 4 PyMC model for the coal mining dataset. Compared with the model in Code Block 1, the main differences are: the use of a Poisson likelihood and exponential inverse link function `np.exp()`, and a smaller number of trees `m=20`.

```
with pm.Model() as model_coal:
    μ = pmx.BART("μ", X=X, Y=Y, m=20)
    y = pm.Poisson("y", mu=np.exp(μ), observed=y_data)
    idata_coal = pm.sample()
```

In Figure 13 we see the mean function as a black line, the 94% HDI of the mean as a darker blue band, and the 94% HDI of the standard deviation as a lighter blue band.

5 Defining the number of trees

Finding a good value for the number of tress remains important when using BART in practice. This number should be large enough so that the sum of trees provides an adequate function to explain the data, but not so large that the function

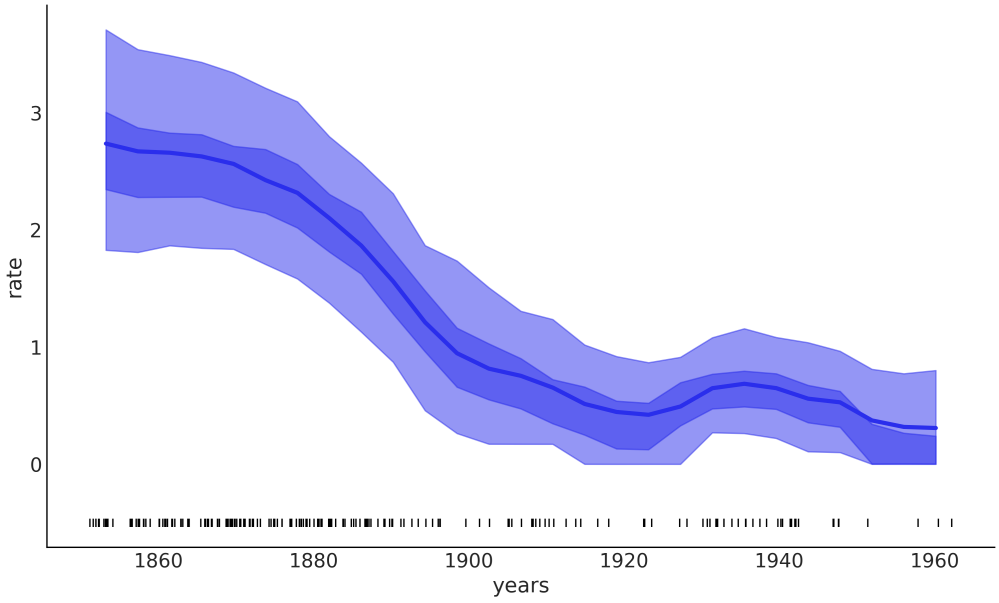

Figure 12: Intensity estimation for the coal mining disaster dataset.

Code Block 5 PyMC model for the marketing example. Notice, how BART is now used to model the standard deviation of the Normal response.

```
with pm.Model() as model_marketing:
    α = pm.HalfNormal("α", 50)
    β = pm.HalfNormal("β", 5)
    μ = pm.Deterministic("μ", np.sqrt(α + β * X[:, 0]))
    σ_ = pmx.BART("σ_", X, Y, m=50)
    σ = pm.Deterministic("σ", np.abs(σ_))
    y = pm.Normal("y", μ, σ, observed=Y)
    idata_marketing = pm.sample()
```

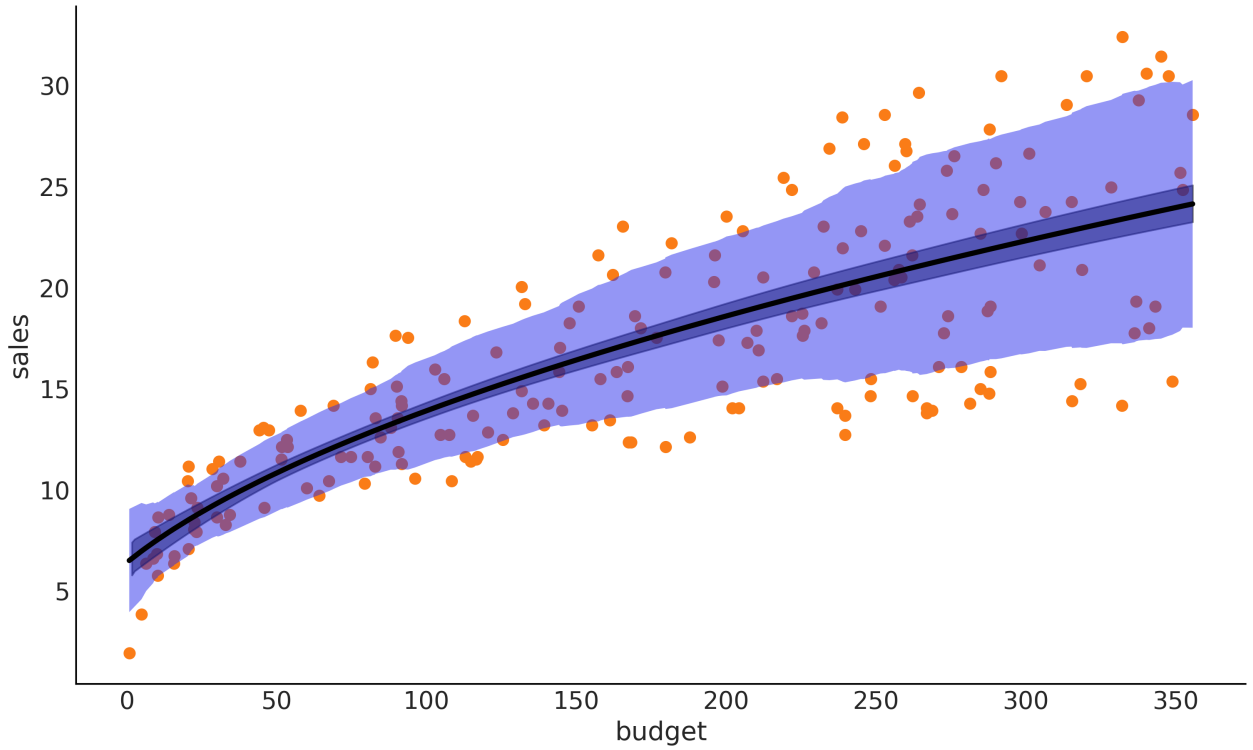

Figure 13: Non-constant variance estimation for the marketing dataset.

becomes too flexible. Overfitting for large values of `m` is mitigated by shrinking the leaf node values towards zero as `m` increases. Another practical reason to avoid overshooting `m` is the computational cost of BART, which increases with `m`.

First, Chipman et al. [2010] and others later, have reported that usually the number of trees should be between 20 and 200. For implementations of BART without sparsifying prior over the splitting variables, authors recommend a lower value of `m` when computing variable importance than when doing inference. In our implementation we see that the computation of variable importance is robust with respect to the value of `m` (see for example Figures 6 and 9) and in general we recommend using the same value of `m` for both inference and variable importance assessment.

One way to tune `m` is to perform K-fold cross validation, as recommended by Chipman et al. [2010]. Another option is to approximate cross validation by using Pareto-smoothed importance sampling leave-one-out cross validation (PSIS-LOO-CV), sometimes also referred to as LOO, Vehtari et al. [2017, 2021b]. The main advantage of PSIS-LOO-CV is that we only need to fit the model once for each value of `m`. It has been reported that PSIS-LOO-CV can lead to overfitting Linero and Yang [2018] when used to select `m`. We have observed that in many problems PSIS-LOO-CV keep increasing as `m` increases, even for values of $m > 200$. Nevertheless, we consider that PSIS-LOO-CV can still be a useful guide to selecting `m`, if we take into account its uncertainty. In Figure 14, we can see that the value of the

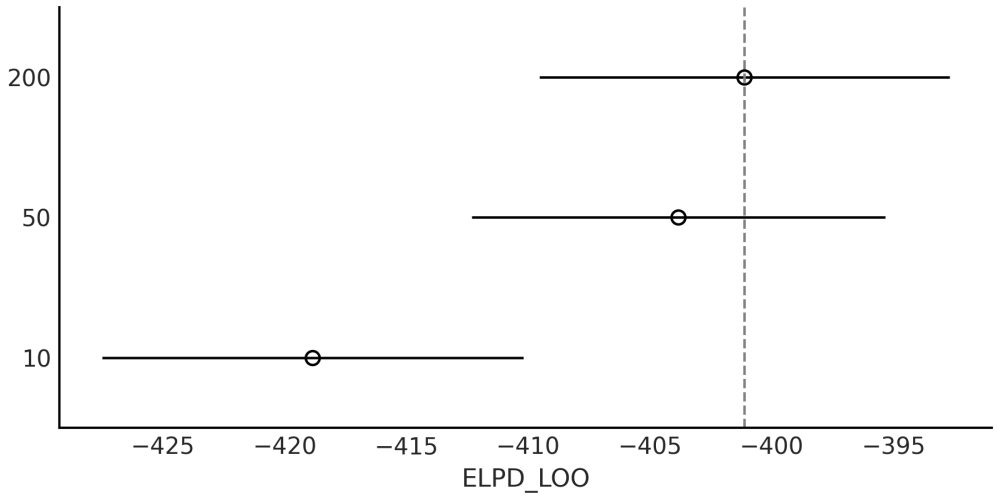

Figure 14: Model comparison using PSIS-LOO-CV for the model and data shown in the first row of Figure 2.

expected log pointwise predictive density (ELPD), as estimated by PSIS-LOO-CV, follows the order $200 > 50 > 10$, indicating that we should pick `m`=200. But if we also consider the uncertainty of the estimated ELPD, we can see that there is considerable overlap between 50 and 200, and in such cases we would pick `m`=50. This is a pattern we have observed in many cases when running our experiments with BART. We notice that it can also happen that the ELPD, as estimated by PSIS-LOO-CV, increases and then decreases with `m`.

6 Discussion and Conclusion

We have introduced a probabilistic programming version of BART by extending PyMC. It effectively combines the flexibility of BART with the power of the state-of-the-art probabilistic programming framework PyMC. Through a series of empirical evaluations, we have given an overview of how BART can be used for inference of non-linear regression models and to assess variable importance or perform variable selection.

When working with Bayesian models, there is a series of related tasks that need to be addressed besides inference itself [Kumar et al., 2019, Martin et al., 2021]; these include convergence assessment, model criticism, and model comparison. Due to the tight integration of PyMC with ArviZ all these tasks are immediately available also for BART models.

We hope that our contribution will help more users adopt BART models as part of their toolbox, and we invite others researchers and developers to further improve this implementation in the future, as well as work on similar implementations in other probabilistic programming languages.

A Sampling from BART

We use a sequential Monte Carlo sampler based on the Particle Gibbs method introduced by Lakshminarayanan et al. [2015], but with some modifications in order to be able to define a generalized version of BART, see Algorithm 1. PyMC will automatically assign this sampler to a `pmx.BART` distribution, and if other random variables are present in the model it will assign other samplers to those variables. This makes our implementation of BART flexible enough to easily accommodate a large family of BART models, and even a combination of BART and other models, like, for example, linear regression. In Algorithm 1 uppercase letters represent arrays, that is G is an array of m trees. The first three lines are run once to initialize the algorithm, after that the main loop body (lines 5 to 27) constitutes one step of the sampling algorithm for a BART variable, which can be interleaved with one step of another sampling algorithm, like NUTS, for other variables. To reduce the computational cost at each step, we update a subset of the m trees $m_b \leq m$, by default 10% of m. To simplify the description of the algorithm, we have defined a few functions, that we now describe.

initialize_trees : We set all trees to the mean of Y divided by `m`. That is, we set the sum of trees, μ, at the mean of Y.

initialize_particles : In order to propose a new G_i tree, we generate N particle-trees by sampling from the prior. One of these particles is just the tree we want to replace, G_i. This ensures that there is a non-zero probability of

Algorithm 1 The PGBART sampler

Require: X, Y, m, n ▷ n is the number of particles
$G \leftarrow$ `initialize_trees`(m)
$\mu \leftarrow \sum_{i=1}^{m} G_i^{\mu}$ ▷ compute the sum of trees
$w_t \leftarrow 0$
$m_b \leq m$
for i:=1 to m_b **do**
  $\mu_{-i} \leftarrow \mu - G_i^{\mu}$ ▷ the sum of trees without the tree to replace
  P $\leftarrow$ `initialize_particles`(n, w_t)
  while trees grow **do**
    for j:=3 to n **do**
      $P_j \leftarrow$ `grow_tree`(Pj) ▷ attempt to grow a particle-tree
      $P_j^l \leftarrow \log p(\mu_{-i} + \mathrm{P}_j^{\mu} \mid g_{-i}, \theta, Y, X)$ ▷ compute conditional log-likelihood
      $P_j^w \leftarrow P_j^w + P_j^l - P_{j-1}^l$ ▷ update weights
    end for
    $w, \bar{W} \leftarrow$ `normalize_weights`$(P_{3:N})$
    $P_{3:N} \leftarrow$`resample`$(P_{3:N}, \bar{W})$
    $P_{3:N}^w \leftarrow w$ ▷ assign the same weight to all particles
  end while
  $\bar{W} \leftarrow \frac{P_j^l}{\sum_j^N P_j^l}$ ▷ use the normalized log-likelihoods as weights
  $G_i \leftarrow$ `sample`$(P, \bar{W})$ ▷ sample a new tree from all the particles
  $w_t \leftarrow G_i^l - \log(N)$ ▷ compute particle weight next round
  $\mu \leftarrow \mu_{-i} + G_i$
  if tuning **then**
    update α_{vec}
  else
    update `variable_inclusion`
  end if
end for

keeping the current tree, instead of accepting a new one. Another particle is generated by resampling the values of the leaf nodes, while keeping the tree structure $\mathcal{T}_i$ unmodified. We find this simple solution to be effective at avoiding getting trapped at local minima, possible by allowing small perturbations around already found solutions. The rest of the particles are grown starting from scratch (see `grow_tree` function below) instead of being perturbations of an existing tree. This helps to explore larger regions.

normalize_weights : We normalize the weights, so they are between 0 and 1, and they sum up to 1; this is $\bar{W}$. We also compute w, which is the sum of unnormalized weights, divided by the number of particles. After resampling, all particles will have the same w weight.

resample : Based on, $\bar{W}$ we resample all but the first two particles. This will remove particles with low probability and retain those with higher probability. The number of particles is kept constant, meaning some particles may be repeated.

grow_tree function attempts to grow a tree based on the following criteria:

node depth : The probability that a node at depth $d = (0, 1, 2, ...)$ is non-terminal is given by α^d. This prior was proposed and studied by Rockova and Saha [2018]. It is recommended that $\alpha \in [0, 0.5)$, the default is 0.25.

splitting variable : We compute the distribution over the splitting variables from the data. We begin with a flat categorical distribution, α_{vec}, i.e., all covariates have the same chance of being used as splitting variables. During the tuning phase, we continuously update α_{vec} based on counting the splitting variables in the accepted trees. After the tuning phase, we fix this distribution and use it to sample from the splitting variables.

splitting values : Uniform over the observed values.

leaf values : We use $\mathcal{N}(\mu_{\text{pred}}, \varepsilon^2)$, where μ_{pred} is computed as the mean of the current sum of trees divided by the number of trees `m`. ε is computed from Y, being $\varepsilon = \frac{3}{\sqrt{m}}$ for binomial data and $\varepsilon = \frac{Y_{\text{std}}}{\sqrt{m}}$ for data other than binomial. When testing our implementation, we noticed that inference can be sensitive to this

value. Hence, during tuning new particles are created with a scale factor kf, that multiplies ε. We find that sampling $kf \sim \mathcal{U}(0.33, 0.75)$ provides good results across very different datasets. After tuning new particles, inherit the value of kf from the particle to replace.

References

J. Bleich, A. Kapelner, E. I. George, and S. T. Jensen. Variable selection for bart: An application to gene regulation. The Annals of Applied Statistics, 8(3), Sep 2014. ISSN 1932-6157. doi: 10.1214/14-aoas755. URL http://dx.doi.org/10.1214/14-AOAS755.

C. J. Carlson. embarcadero: Species distribution modelling with bayesian additive regression trees in r. Methods in Ecology and Evolution, 11(7):850–858, 2020. doi: https://doi.org/10.1111/2041-210X.13389. URL https://besjournals.onlinelibrary.wiley.com/doi/abs/10.1111/2041-210X.13389.

X. Chen, M. O. Harhay, G. Tong, and F. Li. A bayesian machine learning approach for estimating heterogeneous survivor causal effects: Applications to a critical care trial, 2022. URL https://arxiv.org/abs/2204.06657.

H. A. Chipman, E. I. George, and R. E. McCulloch. BART: Bayesian additive regression trees. The Annals of Applied Statistics, 4(1):266–298, Mar. 2010. ISSN 1932-6157. doi: 10.1214/09-AOAS285. URL http://projecteuclid.org/euclid.aoas/1273584455.

R. S. de Souza, A. Krone-Martins, V. Carruba, R. de Cassia Domingos, E. E. O. Ishida, S. Alijbaae, M. H. Espinoza, and W. Barletta. Probabilistic modeling of asteroid diameters from gaia DR2 errors. Research Notes of the AAS, 5 (8):199, aug 2021. doi: 10.3847/2515-5172/ac205e. URL https://doi.org/10.3847/2515-5172/ac205e.

J. H. Friedman. Greedy function approximation: A gradient boosting machine. The Annals of Statistics, 29(5):1189 – 1232, 2001. doi: 10.1214/aos/1013203451. URL https://doi.org/10.1214/aos/1013203451.

J. Hill, A. Linero, and J. Murray. Bayesian additive regression trees: A review and look forward. Annual Review of Statistics and Its Application, 7(1):251–278, 2020. doi: 10.1146/annurev-statistics-031219-041110. URL https://doi.org/10.1146/annurev-statistics-031219-041110.

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

M. D. Hoffman and A. Gelman. The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo. Journal of Machine Learning Research, 15(1):1593–1623, 2014.

S. Hoyer and J. Hamman. Xarray: N-D Labeled Arrays and Datasets in Python. Journal of Open Research Software, 5 (1), Apr. 2017. ISSN 2049-9647. doi: 10.5334/jors.148.

L. Hu, J. Ji, R. D. Ennis, and J. W. Hogan. A flexible approach for causal inference with multiple treatments and clustered survival outcomes, 2022. URL https://arxiv.org/abs/2202.08318.

A. Kassambara. datarium: Data Bank for Statistical Analysis and Visualization, May 2019. URL https://CRAN.R-project.org/package=datarium.

T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing. Jupyter notebooks – a publishing format for reproducible computational workflows. In F. Loizides and B. Schmidt, editors, Positioning and Power in Academic Publishing: Players, Agents and Agendas, pages 87 – 90. IOS Press, 2016.

G. Kropat, F. Bochud, M. Jaboyedoff, J.-P. Laedermann, C. Murith, M. Palacios, and S. Baechler. Improved predictive mapping of indoor radon concentrations using ensemble regression trees based on automatic clustering of geological units. Journal of environmental radioactivity, 147:51–62, 2015.

M. Kuhn and K. Johnson. Applied Predictive Modeling. Springer, New York, 1st ed. 2013, corr. 2nd printing 2018 edition edition, May 2013. ISBN 978-1-4614-6848-6.

R. Kumar, C. Carroll, A. Hartikainen, and O. A. Martin. Arviz a unified library for exploratory analysis of Bayesian models in python. Journal of Open Source Software, 4(33):1143, 2019. ISSN 2475-9066. doi: 10.21105/joss.01143.

B. Lakshminarayanan, D. M. Roy, and Y. W. Teh. Particle gibbs for bayesian additive regression trees, 2015.

M. Leonti, S. Cabras, C. S. Weckerle, M. N. Solinas, and L. Casu. The causal dependence of present plant knowledge on herbals—contemporary medicinal plant use in campania (italy) compared to matthioli (1568). Journal of Ethnopharmacology, 130(2):379–391, 2010.

Z. Li, S. Liu, W. Conaty, Q.-H. Zhu, P. Moncuquet, W. Stiller, and I. Wilson. Genomic prediction of cotton fibre quality and yield traits using Bayesian regression methods. Heredity, pages 1–10, May 2022. ISSN 1365-2540. doi: 10.1038/s41437-022-00537-x. URL https://www.nature.com/articles/s41437-022-00537-x. Publisher: Nature Publishing Group.

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

A. R. Linero. Generalized bayesian additive regression trees models: Beyond conditional conjugacy, 2022. URL https://arxiv.org/abs/2202.09924.

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

O. Martin. Bayesian Analysis with Python: Introduction to Statistical Modeling and Probabilistic Programming Using PyMC3 and ArviZ, 2nd Edition. Packt Publishing, 2 edition edition, Dec. 2018.

O. Martin, A. Hartikainen, C. Carroll, and O. Abril-Pla. Arviz, May 2022. URL https://doi.org/10.5281/zenodo.6547007.

O. A. Martin, R. Kumar, and J. Lao. Bayesian Modeling and Computation in Python. Chapman and Hall/CRC, Boca Raton, 1st edition, 2021. ISBN 978-0-3678-9436-8.

V. Rockova and E. Saha. On theory for bart, 2018.

J. Salvatier, T. V. Wiecki, and C. Fonnesbeck. Probabilistic programming in python using PyMC3. PeerJ Computer Science, 2:e55, apr 2016. doi: 10.7717/peerj-cs.55. URL https://doi.org/10.7717/peerj-cs.55.

R. Sparapani, C. Spanbauer, and R. McCulloch. Nonparametric machine learning and efficient computation with bayesian additive regression trees: The bart r package. Journal of Statistical Software, 97(1):1–66, 2021. doi: 10.18637/jss.v097.i01. URL https://www.jstatsoft.org/index.php/jss/article/view/v097i01.

K. M. Steele and M. H. Schwartz. Causal effects of motor control on gait kinematics after orthopedic surgery in cerebral palsy: a machine-learning approach. medRxiv, 2022. doi: 10.1101/2022.01.04.21268561. URL https://www.medrxiv.org/content/early/2022/01/05/2022.01.04.21268561.

Y. V. Tan and J. Roy. Bayesian additive regression trees and the general bart model. Statistics in Medicine, 38(25): 5048–5069, 2019. doi: https://doi.org/10.1002/sim.8347. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/sim.8347.

J. Vanhatalo, J. Riihimäki, J. Hartikainen, P. Jylänki, V. Tolvanen, and A. Vehtari. GPstuff: Bayesian modeling with Gaussian processes. J. Mach. Learn. Res., 2013.

A. Vehtari, A. Gelman, and J. Gabry. Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. Statistics and Computing, 27(5):1413–1432, Sept. 2017. ISSN 1573-1375. doi: 10.1007/s11222-016-9696-4. URL https://doi.org/10.1007/s11222-016-9696-4.

A. Vehtari, A. Gelman, D. Simpson, B. Carpenter, and P.-C. Bürkner. Rank-Normalization, Folding, and Localization: An Improved $\widehat{R}$ for Assessing Convergence of MCMC (with Discussion). Bayesian Analysis, 16(2):667 – 718, 2021a. doi: 10.1214/20-BA1221. URL https://doi.org/10.1214/20-BA1221.

A. Vehtari, D. Simpson, A. Gelman, Y. Yao, and J. Gabry. Pareto Smoothed Importance Sampling. arXiv:1507.02646 [stat], Feb. 2021b. URL http://arxiv.org/abs/1507.02646. arXiv: 1507.02646 version: 7.

T. Wiecki, J. Salvatier, A. Patil, M. Kochurov, R. Vieira, B. Engels, J. Lao, C. Carroll, O. A. Martin, M. Osthege, B. T. Willard, A. Seyboldt, A. Rochford, rpgoldman, L. Paz, K. Meyer, P. Coyle, M. E. Gorelli, R. Kumar, O. Abril-Pla, T. Yoshioka, G. Ho, T. Kluyver, K. Beauchamp, A. Andorra, D. Pananos, E. Spaak, B. Edwards, E. Ma, and L. M. Domenzain. pymc-devs/pymc:, June 2022. URL https://doi.org/10.5281/zenodo.6611676.

Z.-H. Zhou. Ensemble Methods: Foundations and Algorithms. CRC press, Boca Raton, FL, 1 edition edition, June 2012. ISBN 978-1-4398-3003-1.