BARP: MRP - Multilevel + BART

James Bisbee (NYU)

james.bisbee@nyu.edu

Multilevel Regression and Poststratification (MRP or, colloqially, Mister P) is the current gold standard for extrapolating opinions to smaller units of interest than the survey was originally designed to represent. MRP's performance is well-documented in papers by Lax and Phillips (2009), Warshaw and Rodden (2012), and Buttice and Highton (2013), who successfully recover state- and district-level estimates of opinion from nationally representative surveys.

However, advances in machine learning can improve on MRP by replacing the multilevel model with more sophosticated prediction algorithms. BARP is one such example that uses Bayesian Additive Regression Trees (Chipman, George, and McCulloch 2010) in lieu of the multilevel model. In a test of predictive accuracy across 89 surveys (Bisbee 2019), BARP yields consistently superior accuracy as measured by both Mean Absolute Error (MAE) and interstate correlation.

In this vignette, I walk through an applied example using the **BARP** package for **R**. I use the data included with the **BARP** package which consists of:

1. A nationally representative survey of support for gay marriage fielded in 2006.
2. Census data giving the share of the population falling into different covariate bins (i.e. the share that is a black female with a college degree between the ages of 31 and 50) for the geographic unit of interest (in this example, US states).

This vignette is designed to demonstrate the functions associated with the **BARP** package. Readers interested in the details of the method are encouraged to refer to Bisbee (2019).

The vignette proceeds as follows: **Section 0** presents a brief overview of Bayesian Additive Regression Trees. Interested readers are encouraged to refer to Chipman, George, and McCulloch (2010).

In **Section 1** I demonstrate how to implement the basic `barp` function to generate state-level opinions from nationally representative data. I show how to estimate upper and lower bounds of these values through either Bayesian credible intervals or through bootstrapping over random samples of the data.

In **Section 2** I demonstrate how to explore the partial depedencies generated by the model using the `barp_partial_dependence` function. This function allows the researcher to evaluate which covariates are most strongly associated with support of, or opposition to, a given topic. The function allows for up to three-way interaction estimation, permitting rich characterizations of how the supplied covariates predict the opinion of interest.

Section 3 evaluates the prognostic power of the covariates by examining the number of times they are used as a "splitting rule" (see section 3.1 for more details). The number of times the covariates are used is known as the "variable inclusion proportion". Following Bleich et al. (2014), I permute the outcome variable to break any connection with the covariates and re-estimate the variable inclusion proportions to generate a null distribution. Against that null distribution I evaluate the significance of variable inclusion. I also discuss methods for dealing with missing data, introduced in Kapelner and Bleich (2015).

Section 4 describes how to implement alternative prediction algorithms. Users can instantiate any one of the 43 different algorithms included in the `SuperLearner` package (Polley and Van der Laan 2015), the full list of which is viewable with a call to the `listWrappers()` function. These functions can be run in isolation or in combination, in which case the package will return both the predictions associated with each algorithm as well as those predictions generated by a weighted ensemble of all algorithms, with weights calculated with 10-fold cross validation.

Section 5 concludes.

0.0 A BART primer

This section introduces Bayesian Additive Regression Trees. Users are encouraged to refer to Chipman, George, and McCulloch (2010) for a more detailed discussion.

A single regression tree $\mathcal{T}$ approximates an unknown function f by recursively partitioning the covariate space ($\mathbf{X}$) to best organize observations (i) according to some outcome Y. The resulting bins (commonly referred to as "nodes" or "leafs") proceed until a stopping criteria is met. Each terminal node b is associated with a parameter value μ_b which combine to form the set $\mathcal{M}$. Observed values of $x \in \mathcal{X}$ are assigned to a $\mu_i \in \mathcal{M}$ by a function $g(x; \mathcal{T}, \mathcal{M})$ which is an approximation of the unknown function f. Armed with many such trees indexed by t, the researcher can predict Y via

$$Y = \sum_{t=1}^{T} g(\mathbf{X}; \mathcal{T}_t, \mathcal{M}_t) + \epsilon, \qquad \epsilon \sim \mathcal{N}(0, \sigma^2)$$

The Bayesian aspect of BART imposes priors on the parameters $(\mathcal{T}_t, \mathcal{M}_t)$ and σ^2, the details of which can be found in Chipman, George, and McCulloch (2010).

Consider a simplified example where y_i is an individual's (i) opinion on gay marriage and $\mathbf{X}$ includes information on respondent age, educational attainment, and state of residence. A regression tree might first divide the data into one group (known as a "node" or a "leaf") over 35 years of age, and the other group younger if this division most cleanly separates supports of gay marriage from opponents. The algorithm might then divide the data based on gender, then back again on age, then on state of residence, each time further separating sub-groups of respondents into supporters and opponents. Each division is called a "splitting rule" where the "splitting variable" x_j is divided at a "splitting value" c.

When the tree stops growing (based on a pre-specified depth or a minimum limit on the number of observations in each terminal node), the terminal nodes each contain a fraction of the original observations characterized by observed opinions and a particular sequence of splitting rules. So for example, a terminal node containing 10 survey respondents, 9 of whom oppose gay marriage, may have been created by selecting those over 35, those without a college degree, those living in the Northeast, those *under* 50, and those living in Massachusetts. This particular sequence of splitting rules is then assigned a parameter μ capturing the aggregate opposition to gay marriage in this group. Armed with these terminal nodes and set of parameters $\mu_i \in \mathcal{M}$, the researcher can then predict opinions using new data or use the estimated functions g to evaluate the partial dependence between a given covariate and the outcome.

Repeating this process many times using the regularization priors on $(\mathcal{T}_t, \mathcal{M}_t)$ and σ^2 yields a rich characterization of the unobservable function f. Draws from the posterior distribution of $Pr(\mathcal{T}_t, \mathcal{M}_t, \sigma^2|Y)$ leverage a Metropolis-within-Gibbs sampler which first proposes a change to the structure of $\mathcal{T}_\infty$, either by growing a terminal node, pruning two child nodes, or changing one of the splitting rules. Samples of $\mathcal{M}_1$ are then drawn from this new structure and this process repeats for each tree in $\mathcal{T}$. Finally, a new draw of σ^2 completes the sampler, providing an estimate of f.

The discussion above deals with continuous outcome variables. In applications to binary opinion data, BART can be fruitfully implemented as a linear probability method. However, if Y is coded as a factor, BART can instead be used for classification with a probit model $Pr(Y = 1|\mathbf{X} = \Phi(\sum_{t=1}^{T} g(\mathbf{X}; \mathcal{T}_t, \mathcal{M}_t))$. (Naturally, no prior is needed for σ_2 since the probit assumes $\sigma^2 = 1$.) The latent variable Z is added to the sampler, replacing Y after the additional step:

$$Z_i|y_i = 1 \sim max\Bigg\{\mathcal{N}\Bigg(\sum_{t=1}^{T} g(\mathbf{X}; \mathcal{T}_t, \mathcal{M}_t)\Bigg), 0\Bigg\} Z_i|y_i = 0 \sim min\Bigg\{\mathcal{N}\Bigg(\sum_{t=1}^{T} g(\mathbf{X}; \mathcal{T}_t, \mathcal{M}_t)\Bigg), 0\Bigg\}$$

From this brief introduction, it should be clear that BART possesses two attractive qualities. First, it allows for deep interactions that grow exponentially in the number of covariates as well as the allowed tree depth. Second, the ensemble character of the method can capture additive effects. These qualities allow for superior estimation of the unknown function *f* relative to multilevel models and relax the requirement that the research define the functional form correctly *a priori*. In the context of extrapolating public opinion through post-stratification, BART's superior predictive performance is particularly attractive, as discussed in detail in Bisbee (2019). In the ensuing sections, I demonstrate how to easily leverage BART's predictive power with the **BARP** package for **R**.

1.0 Predicting opinions with `barp`

`barp` is the main function in the **BARP** package and produces objects of class `barp` which then can be used in other functions. A `barp` object is a list containing two components. The first is a `data.frame` that gives the predicted opinion, and the lower and upper bounds for each geographic unit of interest. The second is the `bartMachine` object (Kapelner and Bleich 2013). I demonstrate the function below.

While the package was originally built with the intention of implementing Bayesian Additive Regression Trees, it has since been expanded to accommodate a number of alternative regularization algorithms via the `SuperLearner` package. These alternative algorithms will return predicted opinions in the same `data.frame` object but users will not be able to examine partial dependencies or prognostic covariate strength. I return to a brief discussion of how to run **BARP** with alternative regularization algorithms at the end of this vignette.

1.1 Installing BARP and setting available memory

BARP implements Bayesian Additive Regression Trees using the **bartMachine** package developed by Adam Kapelner and Justin Bleich (Kapelner and Bleich 2013). This requires `rJava`; installing `rJava` on a Windows PC can be tricky because some machines require users to manually set the `PATH` to their Java bin. For users confronting errors, a good start can be found here. (But Googling the specific error is always the best method.)

Once `rJava` is installed, **BARP** can be installed as follows:

```
require(devtools)
install_github('jbisbee1/BARP')
```

With **BARP** installed, the first thing to do *before* loading the package is set the memory available to Java with `options(java.parameters = "-Xmx[NUM]g")` where `[NUM]` refers to the number of gigabytes of memory to use. For common opinion datasets (i.e. $<$ 5,000 rows and $<$ 20 covariates), 3 GB should suffice.

```
options(java.parameters = "-Xmx5g")
require(BARP)
```

1.2 Loading the data and predicting opinions

We can now proceed to extrapolating opinions on gay marriage to the state level. Following Buttice and Highton (2013), I will use four individual-level covariates (age, education, and the interaction of gender and race), two state-level covariates (Republican presidential vote-share in the preceding election, and the share of the population identifying as a "religious conservative"), and two geographic indicators (state, and region). *Note* that the geographic unit of interest to the user (`geo.unit`) **must** be included in the vector of covariates. The outcome opinion in this example is opposition to gay marriage, surveyed in 2006.

The main parameters used by the `barp` function include:

1. the survey data `dat`
2. the census data `census`
3. variable names for the outcome `y` and covariates `x`
4. variable name of the geographic unit of interest `geo.unit`

The user should also specify the name of the column in the census data that lists the proportions or shares that fall into each covariate category (`proportion`). If left to the default “None”, `barp` assumes that the census data is raw and calculates the proportions by counting the number of rows for each covariate bin over the total rows per geographic unit.

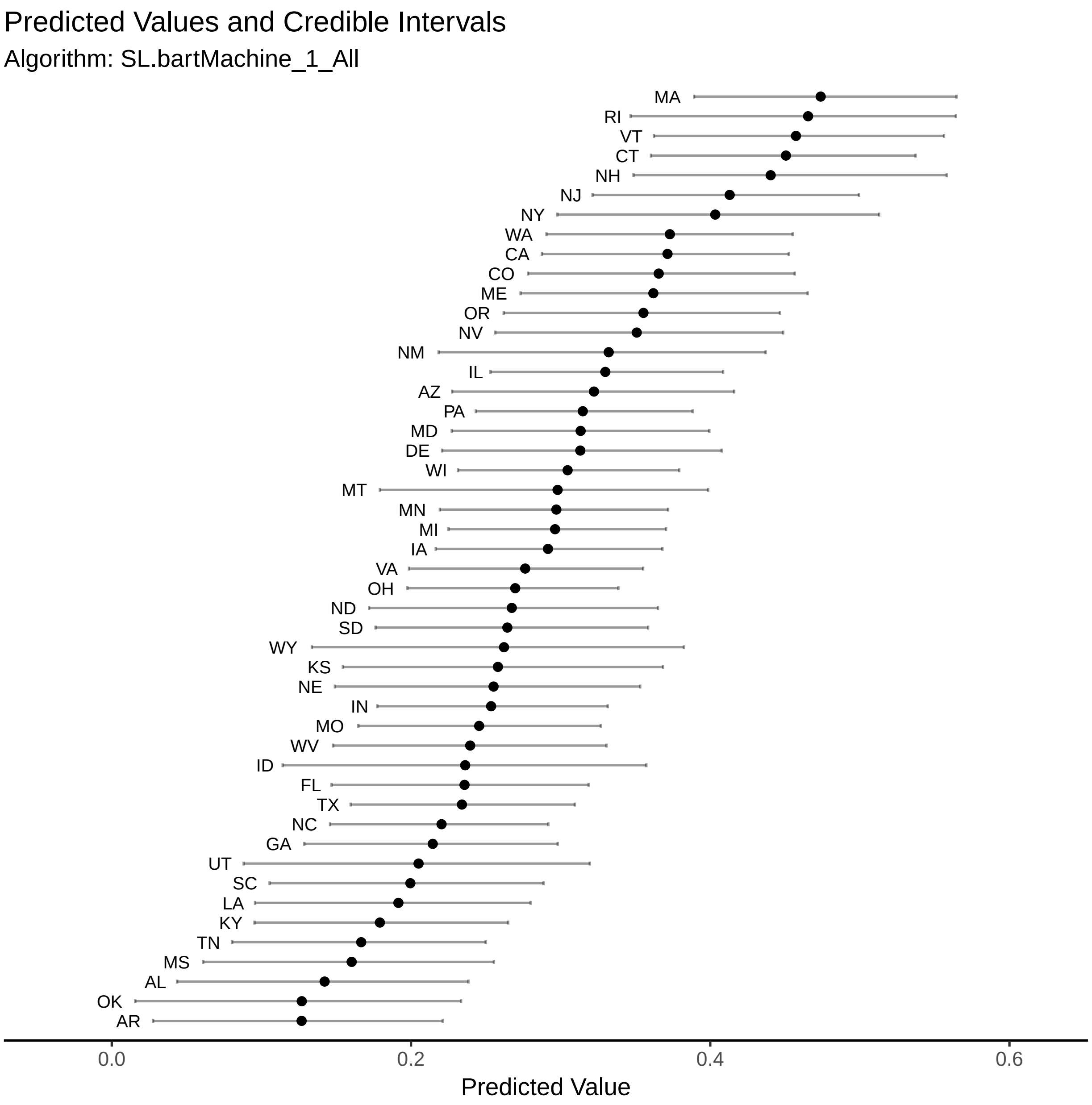
```
data("gaymar")
census06 <- census06 %>% merge(svy %>% dplyr::select(state,stateid) %>% distinct())
factors <- c("region","gXr","state")
census06[factors] <- lapply(census06[factors],factor)
svy[c(factors)] <- lapply(svy[c(factors)],factor)
barp.obj <- barp(y = "supp_gaymar",
                 x = c("age","educ","gXr",
                       "pvote","religcon",
                       "state","region"),
                 dat = svy,census = census06,
                 geo.unit = "state",
                 proportion = "n",
                 seed = 1021,
                 serialize = TRUE)
```

The resulting `barp` object summarizes the predicted opinions and bounds as a `data.frame`. Plotting the `barp` object will return either a simple plot of the predicted values and credible intervals (`evaluate_model = FALSE`, the default), or a set of convergence diagnostic plots (`evaluate_model = TRUE`). The latter plot should exhibit relative stability across the post-burn-in Markov Chain Monte Carlo (MCMC) simulations in terms of percent acceptance, number of leafs and terminal nodes, and tree depth (and σ^2 when y is not a factor).

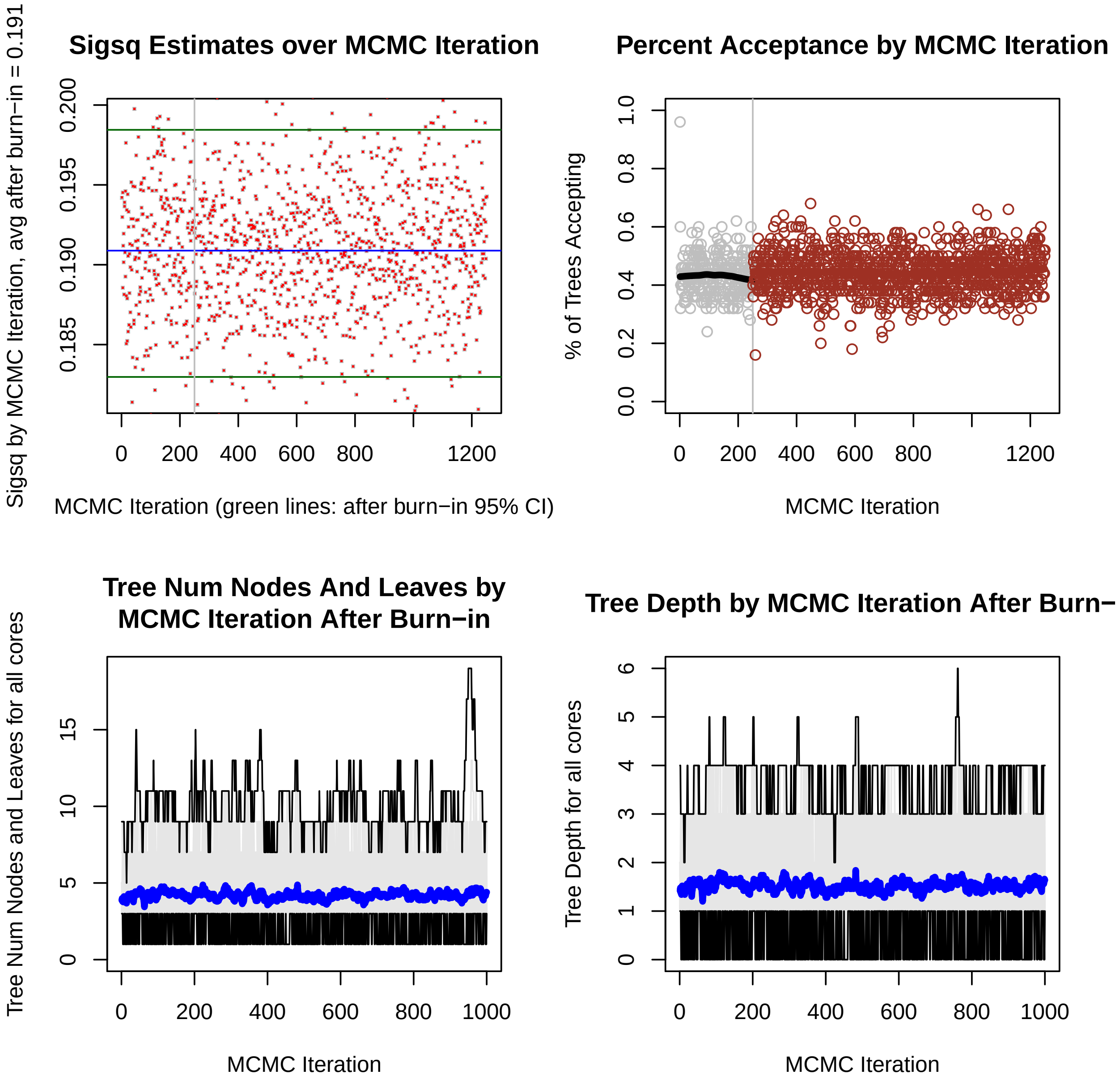
```
barp.obj$pred.opn %>% head()
```

```
##   state SL.bartMachine_1_All     opn.lb    opn.ub
## 1    AL            0.1422521 0.04367446 0.2383575
## 2    AR            0.1268787 0.02760524 0.2212013
## 3    AZ            0.3222951 0.22747162 0.4159995
## 4    CA            0.3714351 0.28748739 0.4525278
## 5    CO            0.3655661 0.27821291 0.4564940
## 6    CT            0.4506066 0.36040250 0.5372730
```

```
plot(barp.obj,algorithm = "BARP")
```



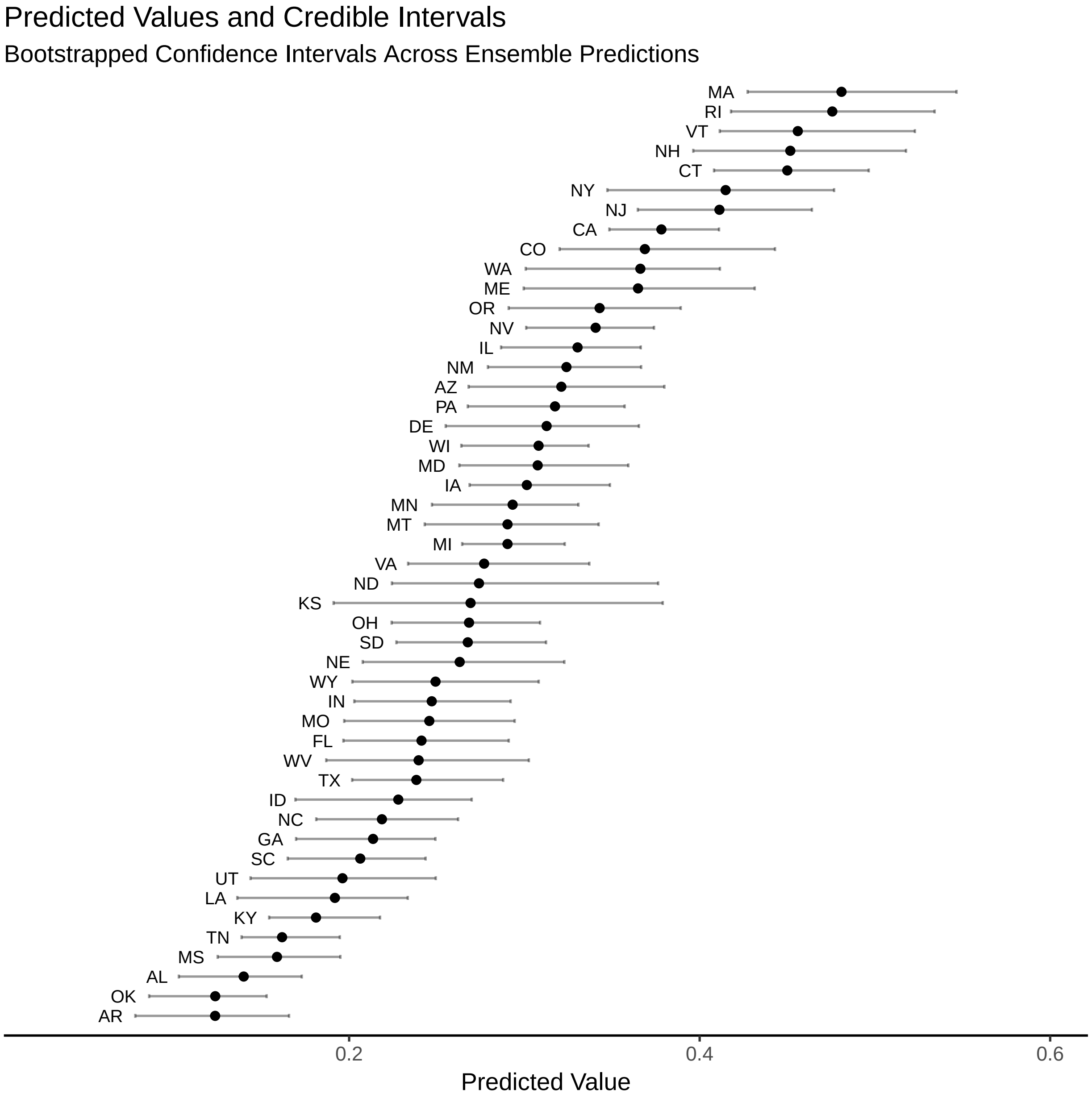
```
plot(barp.obj,evaluate_model = T)
```



Alternatively, the user can choose to calculate the upper and lower bounds using bootstrapped simulations by setting `BSSD = TRUE` and defining the number of simulations through the `nsims` parameter. (Note that doing so will multiply the compute time accordingly). The user can set the credible intervals for the bounds with `cred_int = c(0.025,0.975)`. Lastly, additional arguments can be passed to **bartMachine** including `num_trees`, `num_burn_in`, `num_iterations_after_burn_in`, `verbose`, et cetera. These parameters are non-trivial and should be adjusted based on evaluating model performance via `evaluate_model = TRUE` in the `plot` function.

```
barp.obj2 <- barp(y = "supp_gaymar",
                  x = c("age","educ","gXr",
                        "pvote","religcon",
                        "state","region"),
                  dat = svy,census = census06,
                  geo.unit = "state",
                  proportion = "n",
                  BSSD = T,
                  nsims = 50,
                  num_trees = 40,
                  num_burn_in = 250,
                  num_iterations_after_burn_in = 250,
                  setSeed = 1021)
```

```
plot(barp.obj2)
```

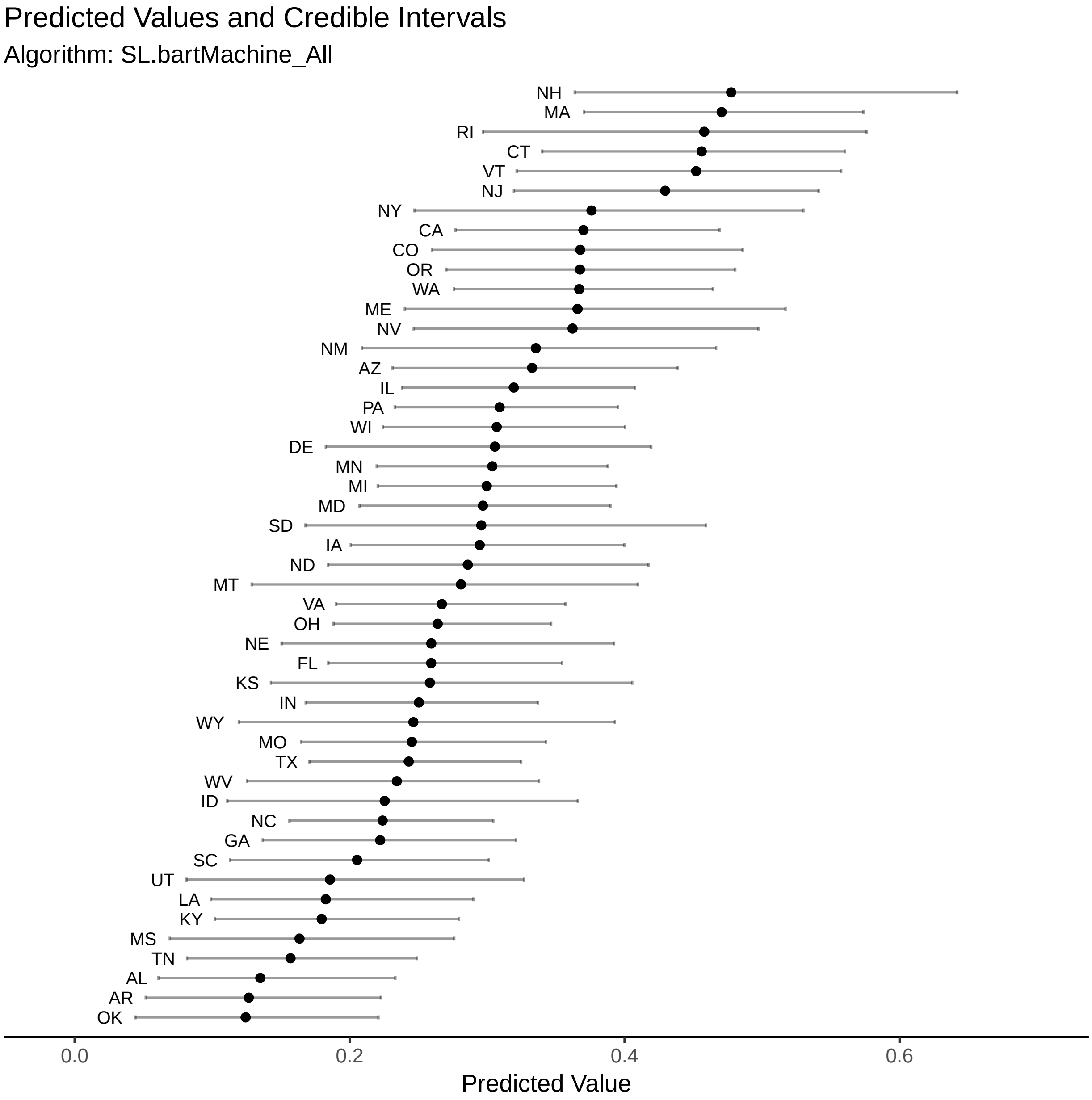


1.2.1 Classification vs regression

As mentioned in **Section 0.0**, bartMachine can handle classification tasks in addition to conventional prediction. The user can prompt bartMachine to execute its classification algorithm by making y a factor. However, care must be taken when ordering the factor as the predicted probabilities are based on the first value. Failing to account for this (somewhat unintuitive) nuance will result in predicted probabilities that are inverted.

```
svy$supp_gaymar <- factor(svy$supp_gaymar,levels = c("1","0"))
barp.class <- barp(y = "supp_gaymar",
                 x = c("age","educ","gXr",
                       "pvote","religcon",
                       "state","region"),
                 dat = svy,census = census06,
                 geo.unit = "state",
                 proportion = "n",
                 setSeed = 1021)
```

```
plot(barp.class)
```



While in most cases the difference between the default and classification routines is trivial, users can examine the confusion matrix of the classification object to gain further insight on model performance and tweak the `prob_rule_class` parameter to improve the predictive power. (Note for applications of BARP to extract representative estimates of opinion at different geographic units, these changes barely influence unit-level predictions.)

```
barp.class$trees$confusion_matrix
```

```
##              predicted 1 predicted 0 model errors
## actual 1         245.000    1260.000        0.837
## actual 0         156.000    3339.000        0.045
## use errors         0.389       0.274        0.283
```

```
barp.class2 <- barp(y = "supp_gaymar",
                    x = c("age","educ","gXr",
                          "pvote","religcon",
                          "state","region"),
                    dat = svy,census = census06,
                    geo.unit = "state",
                    proportion = "n",prob_rule_class = 0.45,setSeed = 1021)
```

```
barp.class2$trees$confusion_matrix
```

```
##              predicted 1 predicted 0 model errors
## actual 1         418.000    1087.000        0.722
## actual 0         330.000    3165.000        0.094
## use errors         0.441       0.256        0.283
```

2.0 Partial dependencies with barp

The user may also be interested in the substantive relationships between covariates and the outcome. The **BARP** package includes tools to facilitate this type of analysis.

2.1 Calculating partial dependencies

The barp_partial_dependence function estimates partial dependence for deep interactions between the covariates using the barp object. The predicted values of the outcome are estimated at different values of the covariate(s) of interest and can be analyzed to make inferential statements about the relationship between the covariates and the outcome, analogous to the coefficients and standard errors of conventional regression analysis.

The vars parameter allows the user to indicate which covariates to explore and, if desired, define the values at which the partial dependence should be estimated. If only the variable names are entered as a character vector, the levels are automatically generated by splitting the support of the variable into quantiles at c(0.05,seq(.1,.9,by = .1),.95) and taking the unique values. Otherwise, the vars parameter must be named list where the names are the variables of interest and the contents are the custom values.

The user can also choose how much of the original data to use when calculating the partial dependence through the prop_data parameter, allowing for faster calculation times at the cost of less precise estimates. The value should be a numeric value between 0 and 1, corresponding to the share of the total data. Lastly, the user can stipulate the credible interval bounds through the credible_interval parameter.

```
bpd <- barp_partial_dependence(barp = barp.obj,
                               vars = list(age = c(1:4),educ = c(1:4)),
                               prop_data = .2,
                               credible_interval = c(0.025,0.975),
                               setSeed = 1021)
```

The resulting object of class bpd contains a list of two components. The first component is a summary data.frame, which gives the predicted outcome and the lower and upper bounds for each value of each variable. The second component is a raw data.frame that gives the posterior predictions (rows) for each value of each variable (columns). The user can make inferential statements using either component of the bpd object or can rely on default plotting methods, described below in Section 3.2.

```
round(bpd$summary %>% head(),3)
```

```
##   age educ  pred    lb    ub
## 1   1    1 0.483 0.409 0.559
## 2   2    1 0.342 0.285 0.405
## 3   3    1 0.251 0.203 0.300
## 4   4    1 0.126 0.076 0.175
## 5   1    2 0.476 0.421 0.540
## 6   2    2 0.342 0.304 0.385
```

```
round(bpd$raw[,1:4] %>% head(),3)
```

```
##   age1_educ1 age2_educ1 age3_educ1 age4_educ1
## 1      0.487      0.368      0.264      0.141
## 2      0.503      0.358      0.275      0.158
## 3      0.454      0.317      0.243      0.113
## 4      0.497      0.373      0.282      0.092
## 5      0.451      0.368      0.307      0.148
## 6      0.478      0.327      0.263      0.135
```

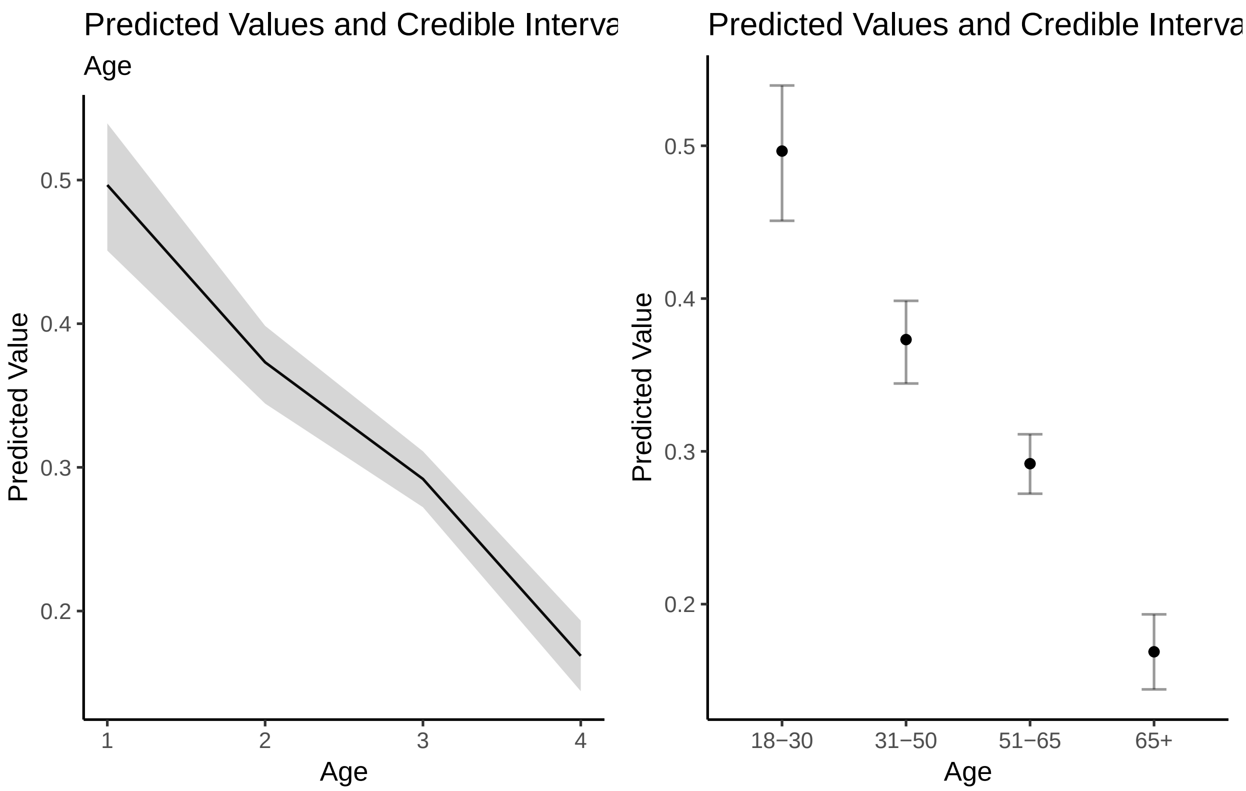
2.2 Plotting partial dependence

BARP provides a default plotting function to streamline the visualization of the partial dependence results up to three-way interactions. This function allows the user to provide variable names (through the `var_names` parameter) and level descriptions (through the `var_labs` parameter), as well as an indicator for which variables are categorical and which are not (`is_categorical`). The type of plot produced will depend on how many variables are included as well as whether or not each is categorical.

For a single variable, the plot type is a function of whether the variable is categorical or continuous, as illustrated.

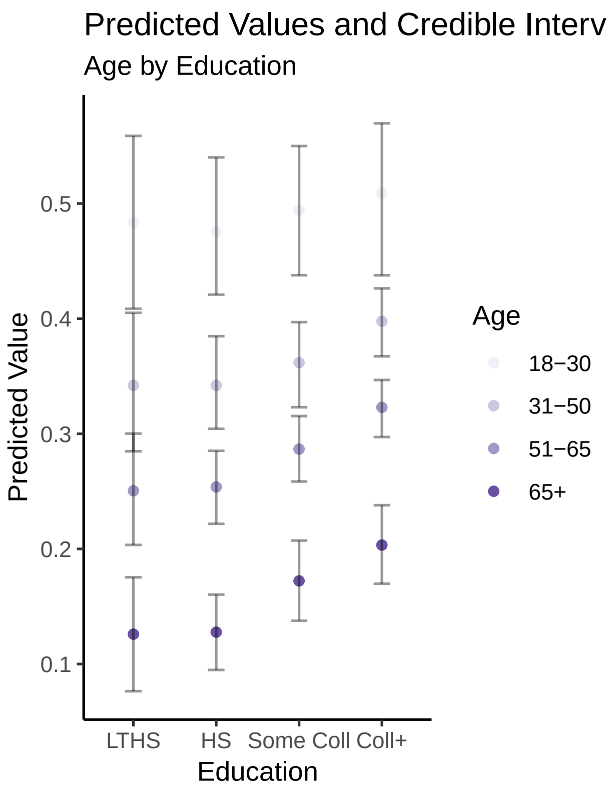
```
bpd1 <- barp_partial_dependence(barp = barp.obj,
                                vars = list(age = 1:4),
                                prop_data = .2,
                                setSeed = 1021)
```

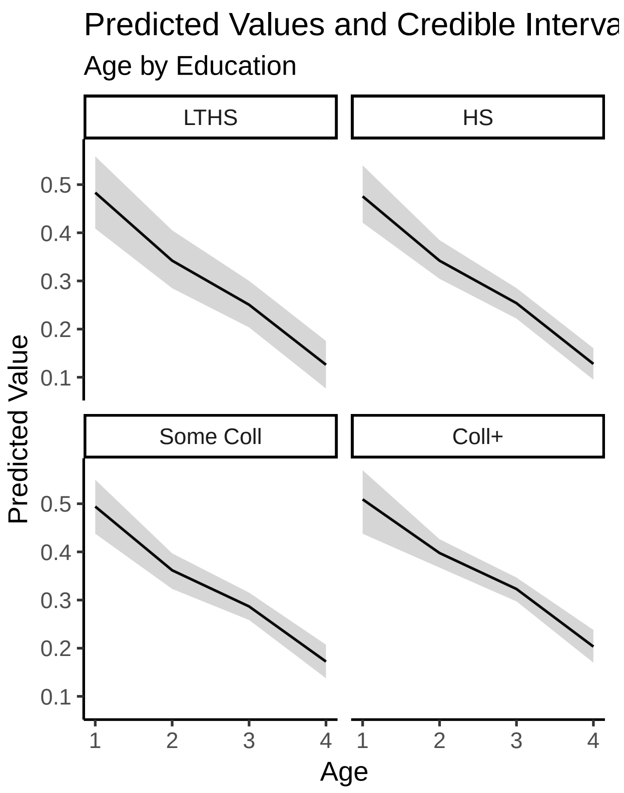
```
p1 <- plot(bpd1,var_names = "Age",
           is_categorical = F)
p2 <- plot(bpd1,var_names = "Age",
           var_labs = list(c("18-30","31-50","51-65","65+")),
           is_categorical = T)
grid.arrange(p1,p2,ncol = 2)
```



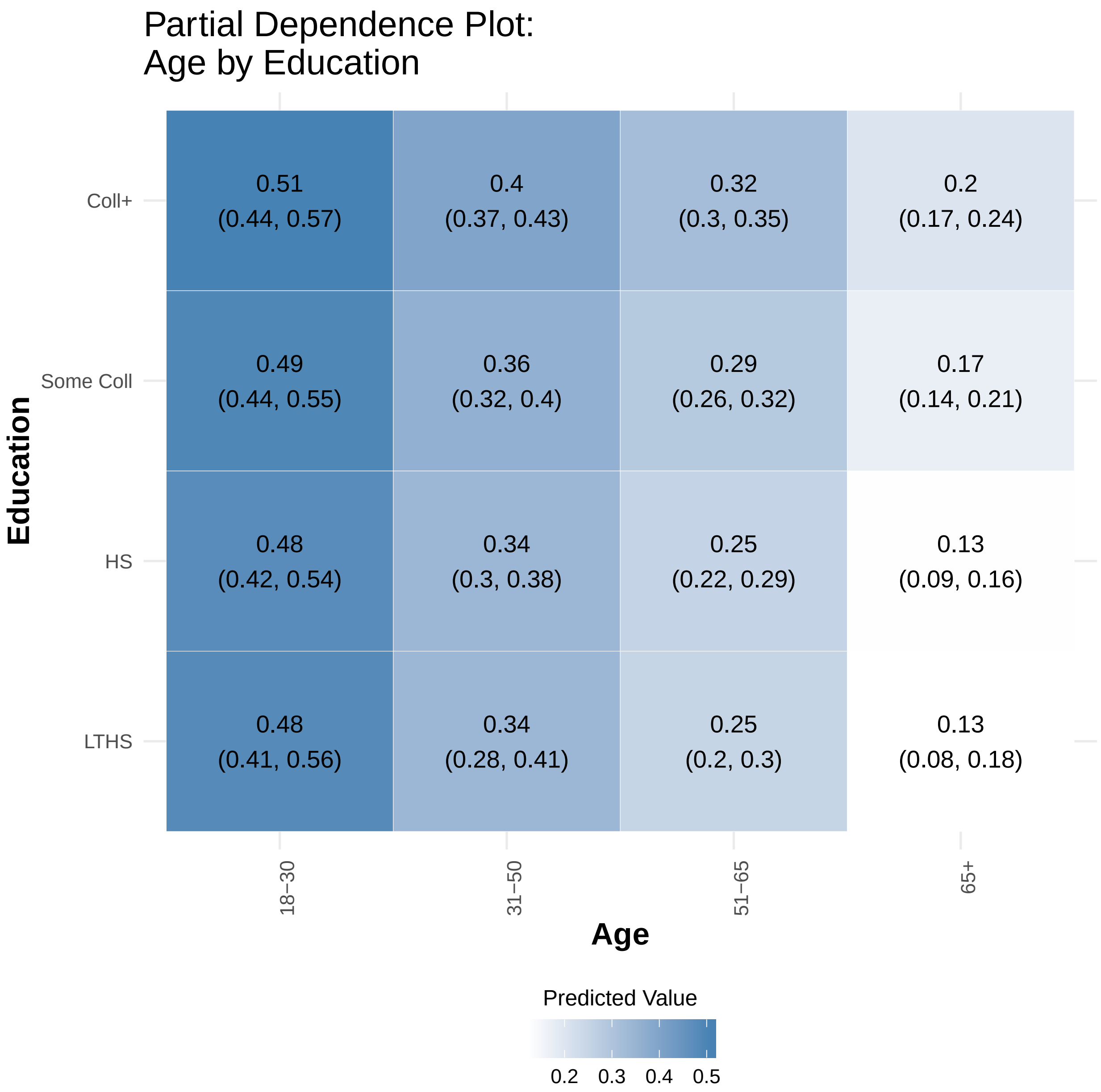
For an interaction analysis of two variables, there may be one of three different plots produced.

```
p1 <- plot(bpd,
     var_names = c("Age","Education"),
     var_labs = list(c("18-30","31-50","51-65","65+"),
                     c("LTHS","HS","Some Coll","Coll+")),
     is_categorical = c(T,T))
p2 <- plot(bpd,
     var_names = c("Age","Education"),
     var_labs = list(c("18-30","31-50","51-65","65+"),
                     c("LTHS","HS","Some Coll","Coll+")),
     is_categorical = c(F,T))
grid.arrange(p1,p2,ncol = 2)
```





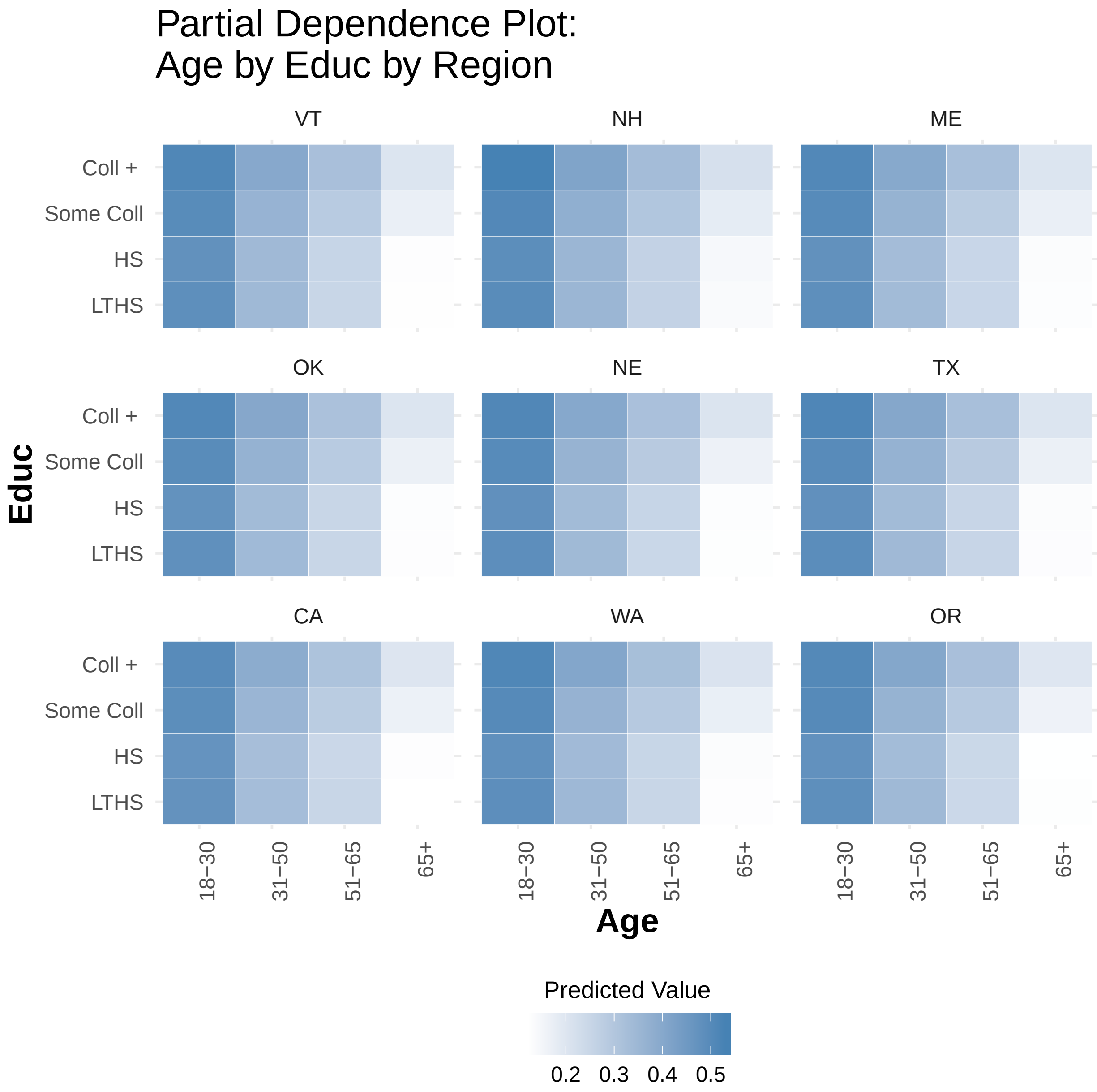
```r
plot(bpd,
     var_names = c("Age","Education"),
     var_labs = list(c("18-30","31-50","51-65","65+"),
                     c("LTHS","HS","Some Coll","Coll+")),
     is_categorical = c(F,F))
```



Finally, for a three-way interaction, the plot will be a grid of heatmaps, regardless of variable type. The third variable will always be the one used to organize the grid while the first and second variables will be the x- and y-axes respectively. Users should limit the number of levels in the `barp_partial_dependence` function for visual clarity.

```
bpd3 <- barp_partial_dependence(barp.obj = barp.obj,
                                vars = list(age = 1:4,educ = 1:4,
                                            state = c("VT","NH","ME",
                                                      "OK","NE","TX",
                                                      "CA","WA","OR")),
                                prop_data = .2,
                                setSeed = 1021)
```

```
plot(bpd3,
     var_names = c("Age","Educ","Region"),
     var_labs = list(c("18-30","31-50","51-65","65+"),
                     c("LTHS","HS","Some Coll","Coll + "),
                     c("VT","NH","ME",
                       "OK","NE","TX",
                       "CA","WA","OR")))
```



3.0 Covariate importance using `barp`

The partial dependence analysis informs the user of different covariates' relationships to the opinion of interest. Instead, however, researchers may be interested in which covariates matter most to the BART model. One method of assessing covariate importance is through examination of how often a covariate is used, either in a tree or as a splitting rule.

As BART proceeds, it attempts to divide the data to most cleanly separate observations along the outcome variable. For example, the fastest way to separate subjects who support gay marriage from those who oppose it is likely to be to divide the data into two groups – those under 50 years of age and those over. This splitting rule may then be applied to educational attainment and then to gender. Alternatively, counting the number of times a variable appears in the trees across posterior samples yields a similar measure. **BARP's** `barp_prognostic_covs` function defaults to the splitting rule (through the `type = 'splits'` parameter).

Over the course of post-burn-in Markov Chain Monte Carlo (MCMC) iterations and over the branches of a decision tree, a variable may be chosen as a splitting rule or included in a tree a certain number of times. The Variable Inclusion Proportion (VIP) of a given covariate is the share of total splitting rules (or total trees) in which the covariate is chosen. This proportion is a measure of covariate importance in the model. For more information, please refer to Bleich et al. (2014).

3.1 Average variable inclusion proportions

To assess the relative covariate importance, **BARP** includes a `barp_prognostic_covs` function. This function will return the observed VIPs for all covariates averaged over a number of runs set by the user through the `num_reps` parameter. The user can also set the number of trees through the `num_trees` parameter which may differ from the number of trees used in the original `barp` command. If the user's goal is predictive accuracy, more trees allow for more flexibility. When evaluating covariate importance, however, limiting the number of trees can improve estimation because each covariate must compete with all others to be included (Chipman, George, and McCulloch 2010). If not specified by the user, `num_trees` defaults to 20.

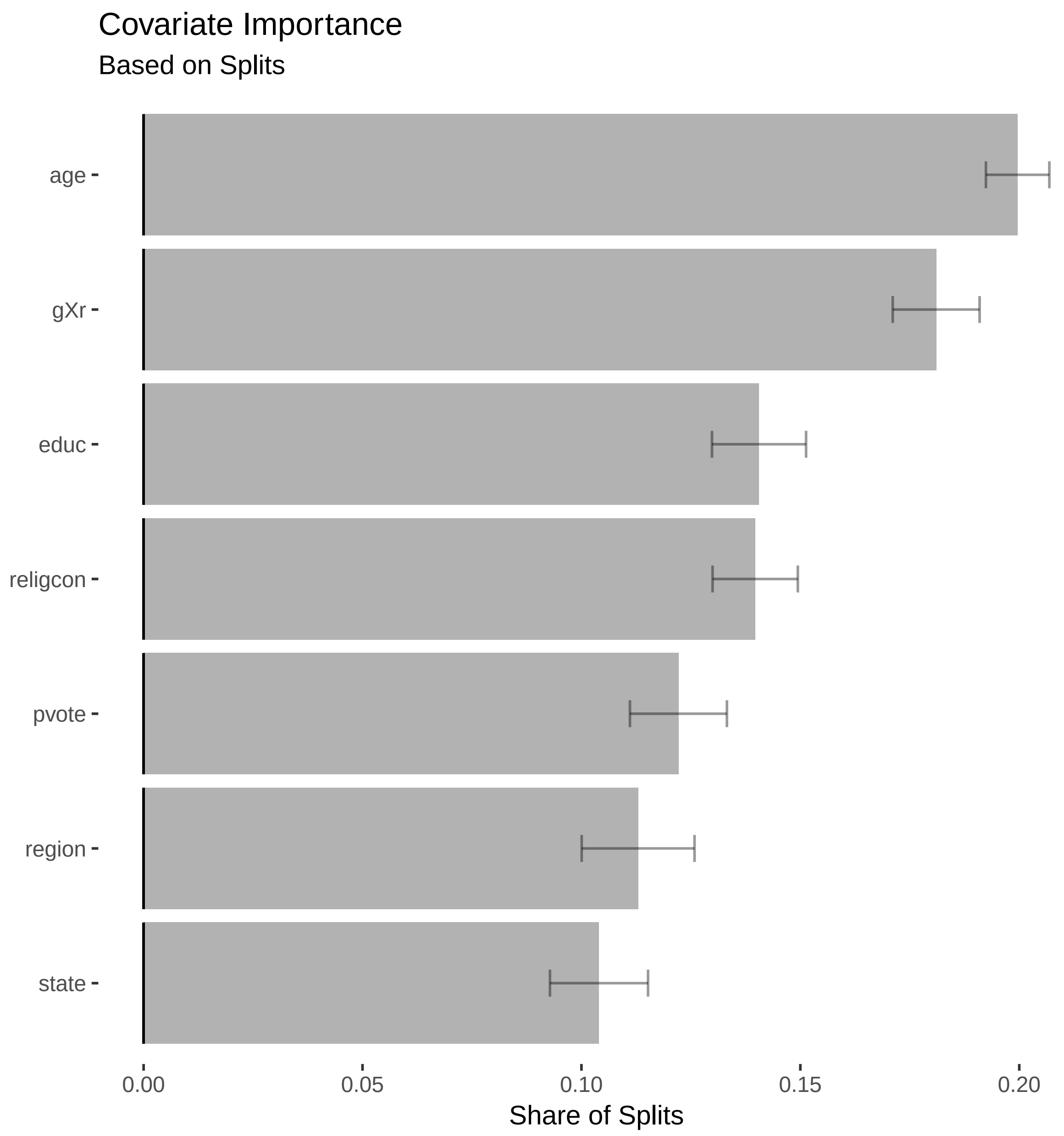
```
barpcov <- barp_prognostic_covs(barp.obj,
                                perm_test = F,
                                interactions = F,
                                num_reps = 30,
                                num_trees = 20,
                                type = "splits",
                                setSeed = 1021)
```

```
round(barpcov$covariate_importance %>% head(),3)
```

```
##        age  educ pvote religcon   gXr region state
## [1,] 0.204 0.129 0.126    0.132 0.189  0.126 0.094
## [2,] 0.196 0.137 0.126    0.150 0.175  0.118 0.098
## [3,] 0.215 0.133 0.106    0.134 0.192  0.098 0.123
## [4,] 0.199 0.138 0.137    0.132 0.163  0.144 0.086
## [5,] 0.203 0.140 0.141    0.143 0.166  0.117 0.090
## [6,] 0.199 0.161 0.115    0.144 0.168  0.098 0.115
```

The `barp_prognostic_covs` function returns an object of class `barpcov` which, if run without a permutation test, contains a single matrix with the number of rows equal to the `num_reps` parameter and the number of columns equal to the number of variables. Plotting this object will produce a horizontal bar chart ordered by variable importance as measured by VIPs. An optional parameter `var_names` allows the user to replace the default variable names with more descriptive labels.

```
plot(barpcov)
```



3.2 Permutation tests

Although the average of the VIPs for all covariates can give the user an idea of which covariates are most important, it does not allow for statistical inference. By randomly permuting y a permutation test breaks the relationship between all covariates and the outcome variable. The newly permuted VIPs represent a null distribution to compare with the observed VIPs. The user can estimate each variable's statistical significance by setting `perm_test = TRUE` and defining the number of permutation simulations to run though the `num_permute` parameter.

```
barpcov_perm <- barp_prognostic_covs(barp.obj,
                                perm_test = T,
                                num_permute = 30,
                                interactions = F,
                                num_reps = 30,
                                num_trees = 20,
                                type = "splits")
```

```
round(barpcov_perm$permutation_test %>% head(),3)
```

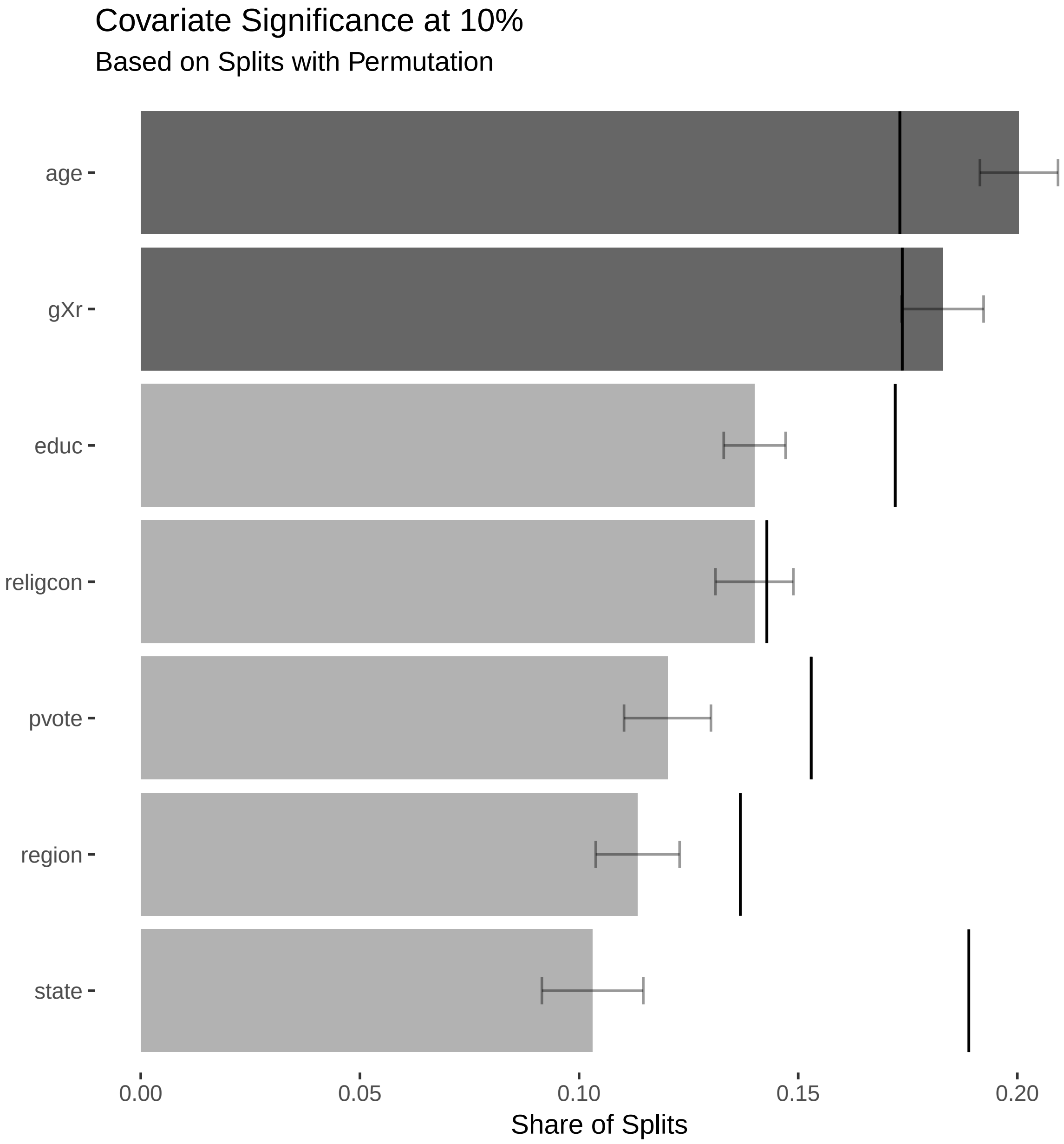
```
##        age  educ pvote religcon   gXr region state
## [1,] 0.149 0.153 0.138    0.124 0.140  0.135 0.162
## [2,] 0.134 0.153 0.148    0.133 0.143  0.121 0.168
## [3,] 0.133 0.172 0.132    0.137 0.128  0.122 0.177
## [4,] 0.164 0.119 0.120    0.123 0.165  0.128 0.181
## [5,] 0.138 0.131 0.121    0.114 0.193  0.127 0.175
## [6,] 0.146 0.146 0.101    0.180 0.173  0.105 0.148
```

```
round(barpcov_perm$p_vals[order(barpcov_perm$p_vals)],3)
```

```
##      age      gXr religcon    pvote     educ   region    state
##    0.020    0.030    0.149    0.653    0.693    0.772    1.000
```

The `barpcov`-class object now includes a matrix summarizing the permutation test results and a vector of p-values capturing the proportion of a variable's permutation test VIPs that fall below the average observed VIP. The plot command now colors the results by significance at a user-specified level through `sig_level` (defaults to 0.05), and overlays the VIP value at this level as a vertical black line.

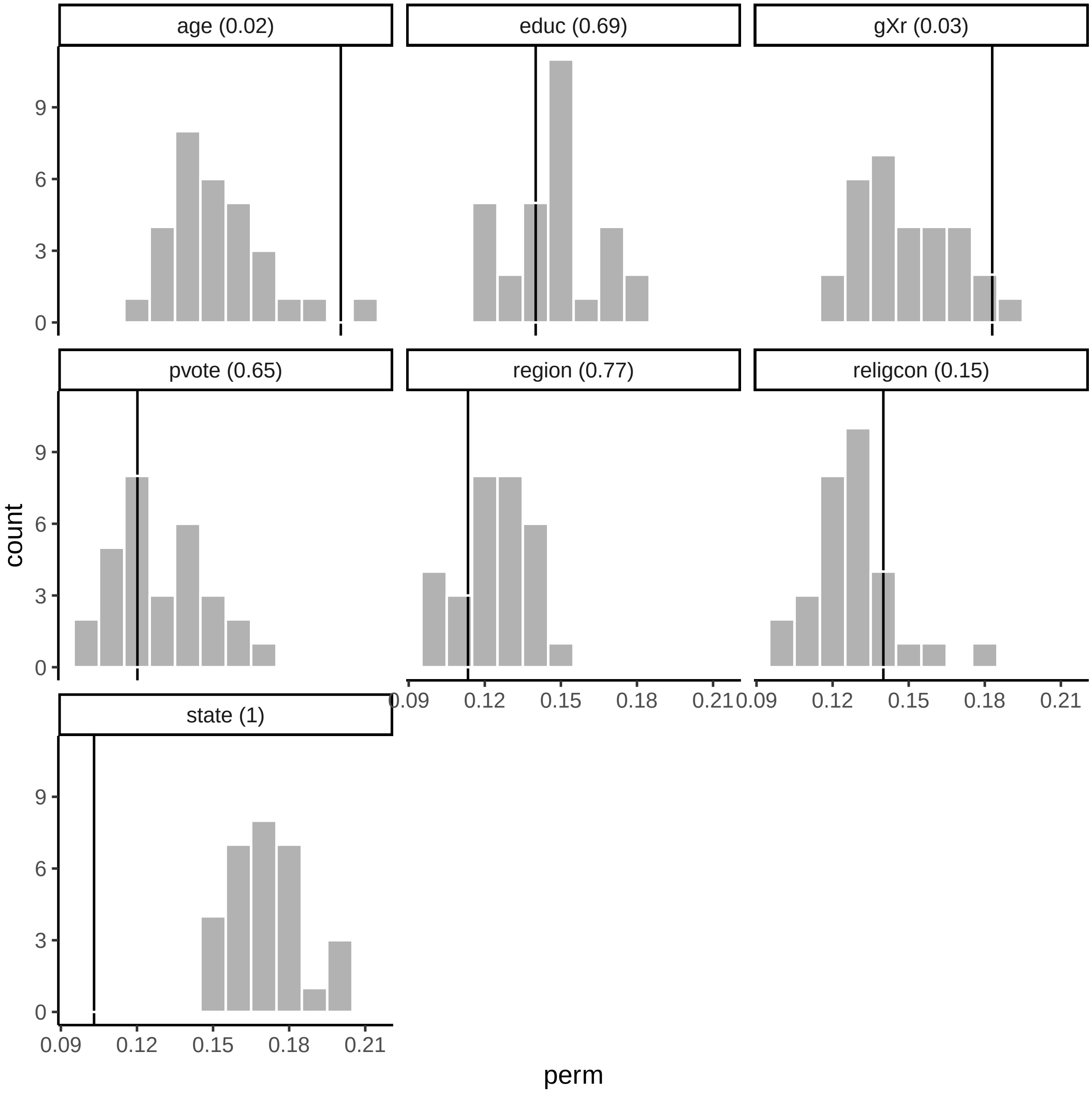
```
plot(barpcov_perm,
     sig_level = 0.10,topn = 20)
```



Alternatively, and as with all **BARP** outputs, the user may customize her own visualization using the raw data. In this example, I plot the histogram of the permutation results for each variable and overlay the average VIPs as vertical lines.

```
require(tidyr)
toplot <- gather(as_data_frame(barpcov_perm$permutation_test),variable,perm)
pvals <- data_frame(variable = names(barpcov_perm$p_vals),pvals = barpcov_perm$p_vals)
pvals <- pvals[order(pvals$pvals),]
pvals <- pvals[1:min(12,nrow(pvals)),]
avgVIPs <- apply(barpcov_perm$covariate_importance,2,mean)
avgVIPs <- data_frame(variable = names(avgVIPs),means = avgVIPs)
toplot <- pvals %>% left_join(toplot) %>% left_join(avgVIPs)
ordered <- pvals$variable
toplot <- toplot %>% arrange(match(variable,ordered))
```

```
toplot$variable <- paste0(toplot$variable," (",round(toplot$pvals,2),")")
ggplot(toplot, aes(x=perm))+
  geom_vline(aes(xintercept = means), colour="black") +
  geom_histogram(binwidth=0.01,colour = "white",fill = rgb(0,0,0,.3)) +
  facet_wrap(~variable) +
  theme_classic()
```

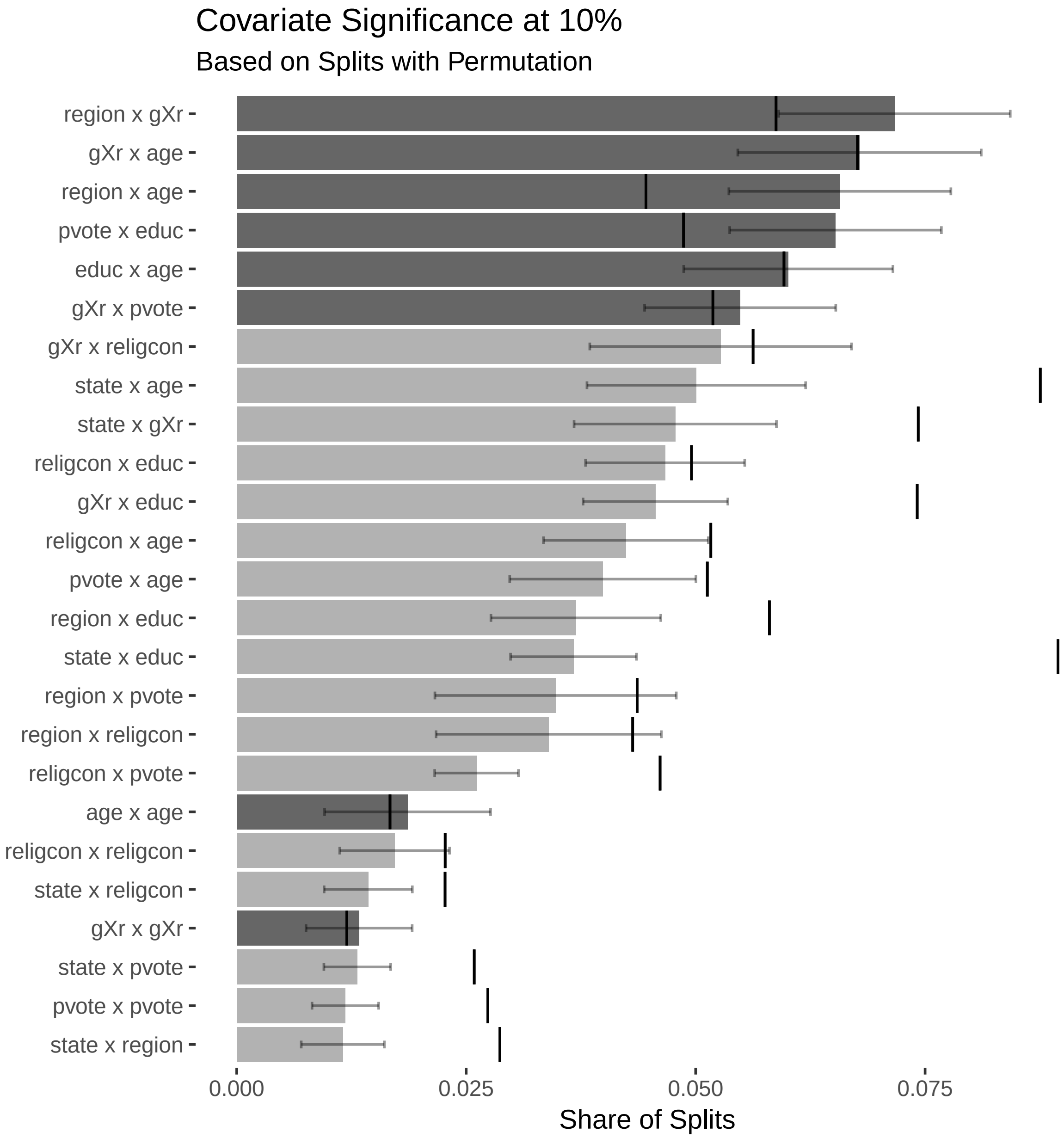


3.3 Interactions

These methods can also be applied to interaction terms.

```
barpcov_int <- barp_prognostic_covs(barp.obj,
                                    perm_test = T,
                                    num_permute = 30,
                                    interactions = T,
                                    num_reps = 30,
                                    num_trees = 20,
                                    type = "splits")
```

```
plot(barpcov_int,
     topn = 25,
     sig_level = 0.10)
```



Covariate Significance at 10%

Based on Splits with Permutation

3.4 Missing Data

The goal of extrapolating representative data from non-representative surveys is fundamentally a missing data problem where the more sparsely populated bins constitute the missing data. However, unlike well-known imputation methods such as **Amelia** (see Honaker et al. (2011) for details) which attempt to recover missing data at the unit of the original data, MRP-style methods predict opinions at the level of grouped bins of individuals where the bins are defined by observable covariates.

Nevertheless, there may be situations, particularly in panel data, where observations are missing at the individual respondent level in a survey. **BARP** allows users to implement a variety of imputation methods via `use_missing_data`, including using the simple average of the vector of observations (`replace_missing_data_with_x_j_bar`), using a linear model (`impute_missingness_with_x_j_bar_for_lm`), or to treat the missing data as covariates (`use_missing_data_dummies_as_covars`). A future update of the **BARP** package will add in functionality for the random forest imputation method (`impute_missingness_with_rf_impute`). See Kapelner and Bleich (2015) for more details on how Bayesian Additive Regression Trees treat missing data.

```
svy$age[sample(1:nrow(svy),nrow(svy)*.01,replace = F)] <- NA
svy$educ[sample(1:nrow(svy),nrow(svy)*.01,replace = F)] <- NA

barp.covs <- barp(y = "supp_gaymar",
                 x = c("age","educ","gXr",
                       "pvote","religcon",
                       "state","region"),
                 dat = svy,census = census06,
                 geo.unit = "state",
                 proportion = "n",
                 use_missing_data = TRUE,
                 use_missing_data_dummies_as_covars = TRUE,
                 setSeed = 1021)

barp.xjbar <- barp(y = "supp_gaymar",
                 x = c("age","educ","gXr",
                       "pvote","religcon",
                       "state","region"),
                 dat = svy,census = census06,
                 geo.unit = "state",
                 proportion = "n",
                 use_missing_data = TRUE,
                 replace_missing_data_with_x_j_bar = TRUE,
                 setSeed = 1021)

barp.xjlm <- barp(y = "supp_gaymar",
                 x = c("age","educ","gXr",
                       "pvote","religcon",
                       "state","region"),
                 dat = svy,census = census06,
                 geo.unit = "state",
                 proportion = "n",
                 use_missing_data = TRUE,
                 impute_missingness_with_x_j_bar_for_lm = TRUE,
                 setSeed = 1021)
```

```
cor.mat <- cor(cbind(barp.obj$pred.opn$SL.bartMachine_1_All,
          barp.xjbar$pred.opn$SL.bartMachine_1_All,
          barp.xjlm$pred.opn$SL.bartMachine_1_All,
          barp.covs$pred.opn$SL.bartMachine_1_All))
colnames(cor.mat) <- rownames(cor.mat) <- c("No Missing",
                                            "Impute Xj Bar",
                                            "Impute Xj LM",
                                            "Missing as Covs")
round(cor.mat,3)
```

```
##                  No Missing Impute Xj Bar Impute Xj LM Missing as Covs
## No Missing            1.000         0.987        0.993           0.989
## Impute Xj Bar         0.987         1.000        0.993           0.991
## Impute Xj LM          0.993         0.993        1.000           0.993
## Missing as Covs       0.989         0.991        0.993           1.000
```

4.0 Alternative Regularization Methods

The updated **BARP** package allows users to replace or add additional regularization algorithms via the `method` parameter. These algorithms can be one of the 43 methods included in the `SuperLearner` package (Polley and Van der Laan 2015) although users must ensure that the associated dependencies are installed. A detailed introduction to this package can be found here.

The easiest implementation of these alternative algorithms is to include one or more as a character vector. In this case, I use the vanilla implementation of `SL.glmnet`.

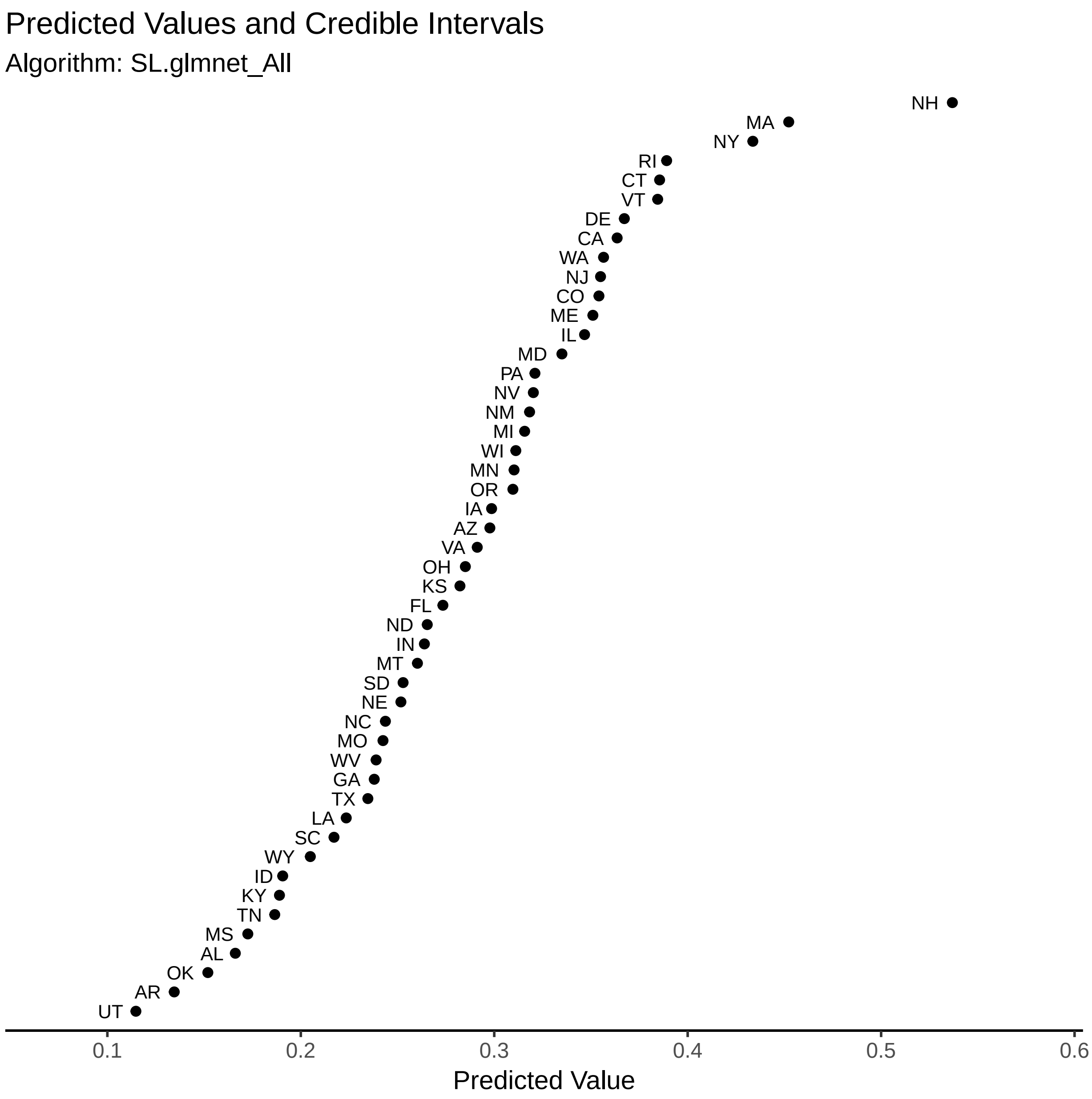
```
data("gaymar")
census06 <- census06 %>% merge(svy %>% dplyr::select(state,stateid) %>% distinct())

barp.objSL <- barp(y = "supp_gaymar",
                  x = c("age","educ","gXr",
                        "pvote","religcon",
                        "state","region"),
                  dat = svy,census = census06,
                  algorithm = c("SL.glmnet"),
                  geo.unit = "state",
                  proportion = "n",
                  setSeed = 1021)
```

```
barp.objSL$pred.opn %>% head()
```

```
##    state SL.glmnet_All opn.lb opn.ub
## AL    AL     0.1661456     NA     NA
## AR    AR     0.1345653     NA     NA
## AZ    AZ     0.2977428     NA     NA
## CA    CA     0.3635293     NA     NA
## CO    CO     0.3541205     NA     NA
## CT    CT     0.3854912     NA     NA
```

```
plot(barp.objSL,algorithm = "SL.glmnet")
```



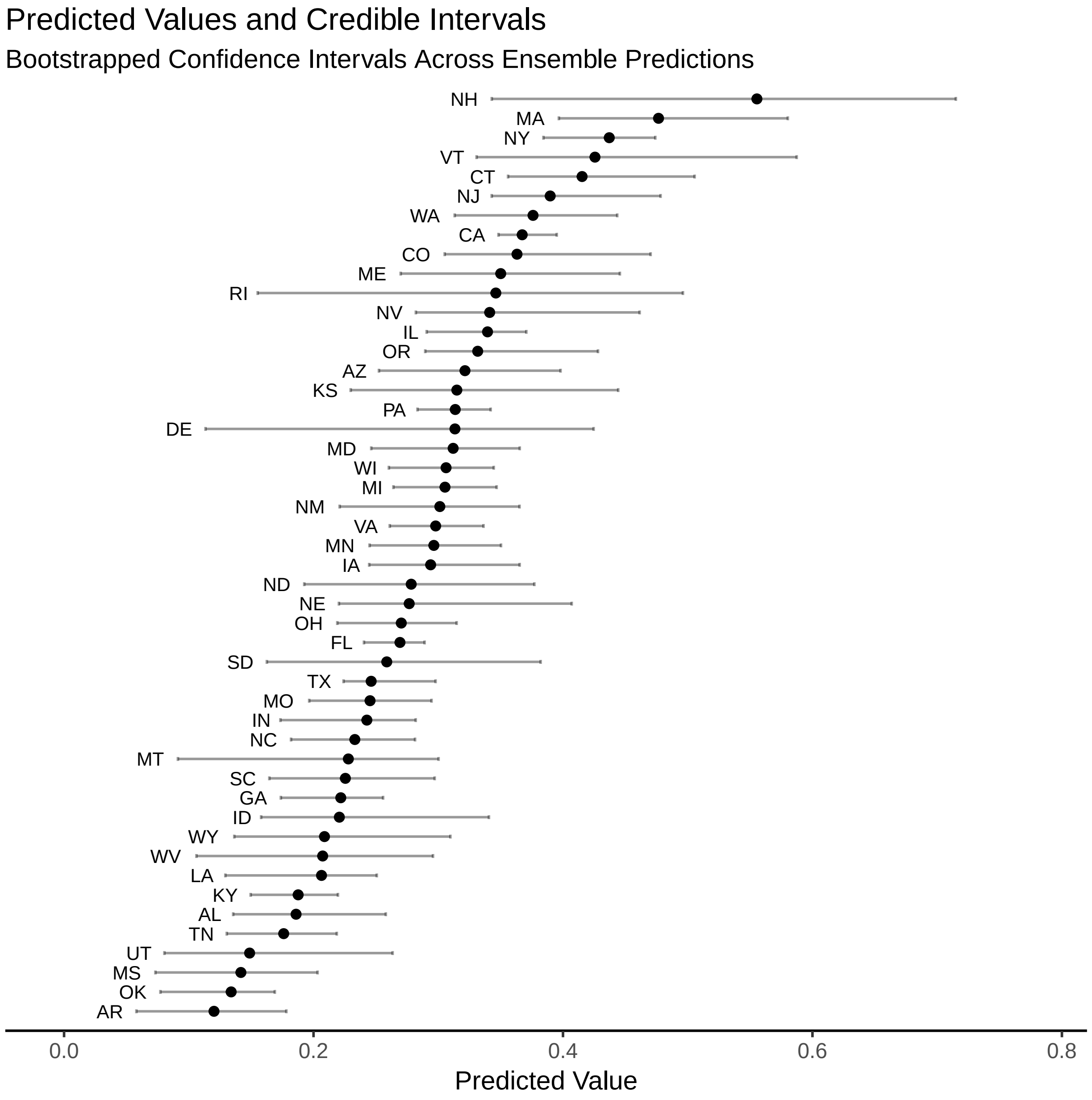
Unlike with `barp` objects, other algorithms do not provide uncertainty estimates without turning on bootstraps with `BSSD = TRUE`.

```
barp.objSLBS <- barp(y = "supp_gaymar",
                 x = c("age","educ","gXr",
                       "pvote","religcon",
                       "state","region"),
                 dat = svy,census = census06,
                 algorithm = c("SL.glmnet"),
                 geo.unit = "state",
                 proportion = "n",
                 BSSD = TRUE,
                 nsims = 50,
                 setSeed = 1021)
```

```r
barp.objSLBS$pred.opn %>% head()
```

```
##   state  pred.opn     opn.lb    opn.ub
## 1    AL 0.1860024 0.13558079 0.2579651
## 2    AR 0.1201974 0.05798779 0.1782284
## 3    AZ 0.3214124 0.25244014 0.3980291
## 4    CA 0.3672889 0.34826682 0.3950198
## 5    CO 0.3631236 0.30516150 0.4702434
## 6    CT 0.4153072 0.35595841 0.5055253
```

```r
plot(barp.objSLBS)
```



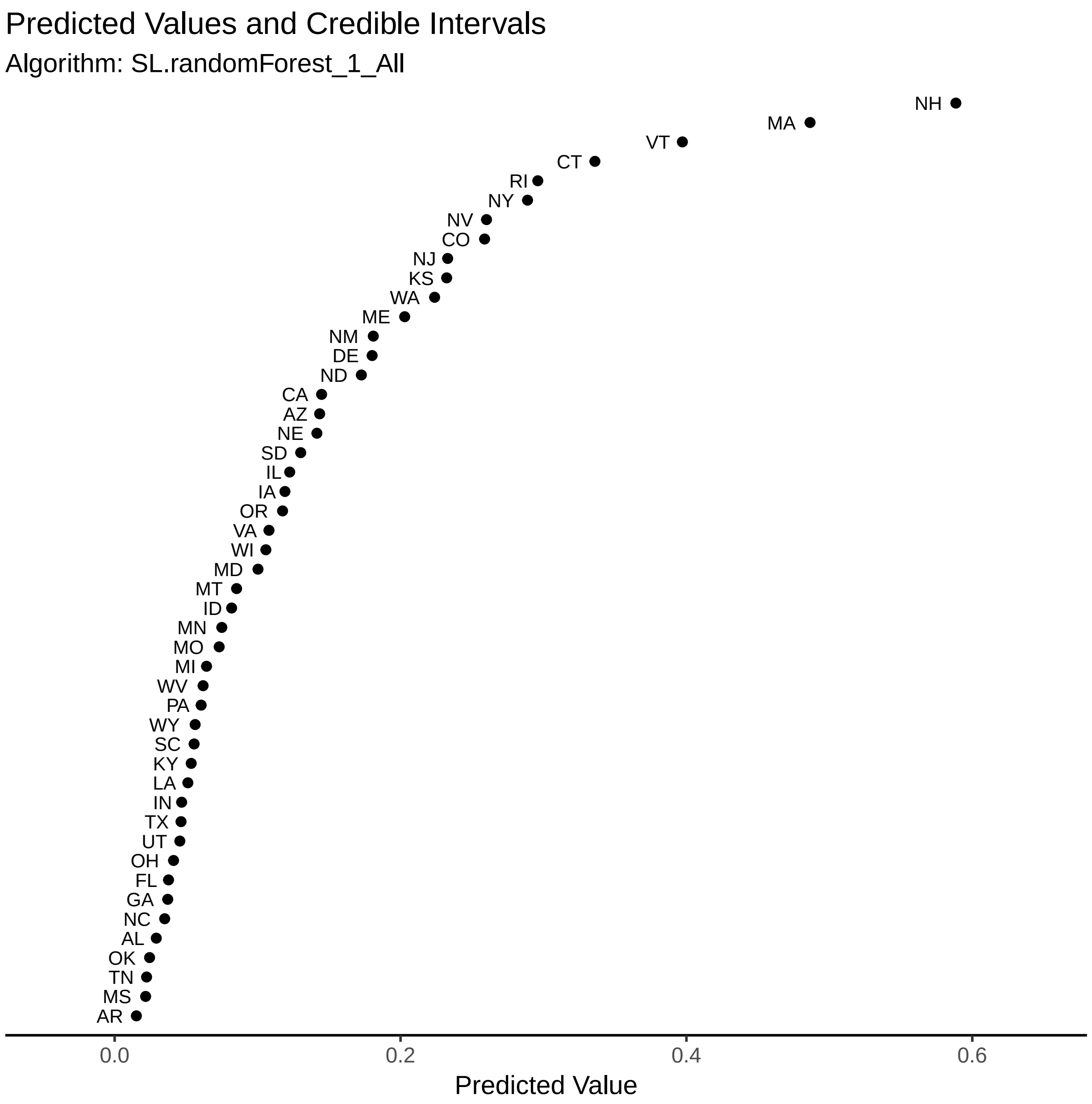
These algorithms can be customized as described in the `SuperLearner` documentation. For example, if we wanted to run a random forest with 150 trees, we would do the following:

```
rf_new <- create.Learner("SL.randomForest",params = list(ntree = 150))
barp.objRF <- barp(y = "supp_gaymar",
                 x = c("age","educ","gXr",
                       "pvote","religcon",
                       "state","region"),
                 dat = svy,census = census06,
                 algorithm = c(rf_new$names),
                 geo.unit = "state",
                 proportion = "n",
                 setSeed = 1021)
```

```
barp.objRF$pred.opn %>% head()
```

```
##    state SL.randomForest_1_All opn.lb opn.ub
## AL    AL            0.02913993     NA     NA
## AR    AR            0.01521739     NA     NA
## AZ    AZ            0.14342199     NA     NA
## CA    CA            0.14478368     NA     NA
## CO    CO            0.25880952     NA     NA
## CT    CT            0.33599271     NA     NA
```

```
plot(barp.objRF)
```



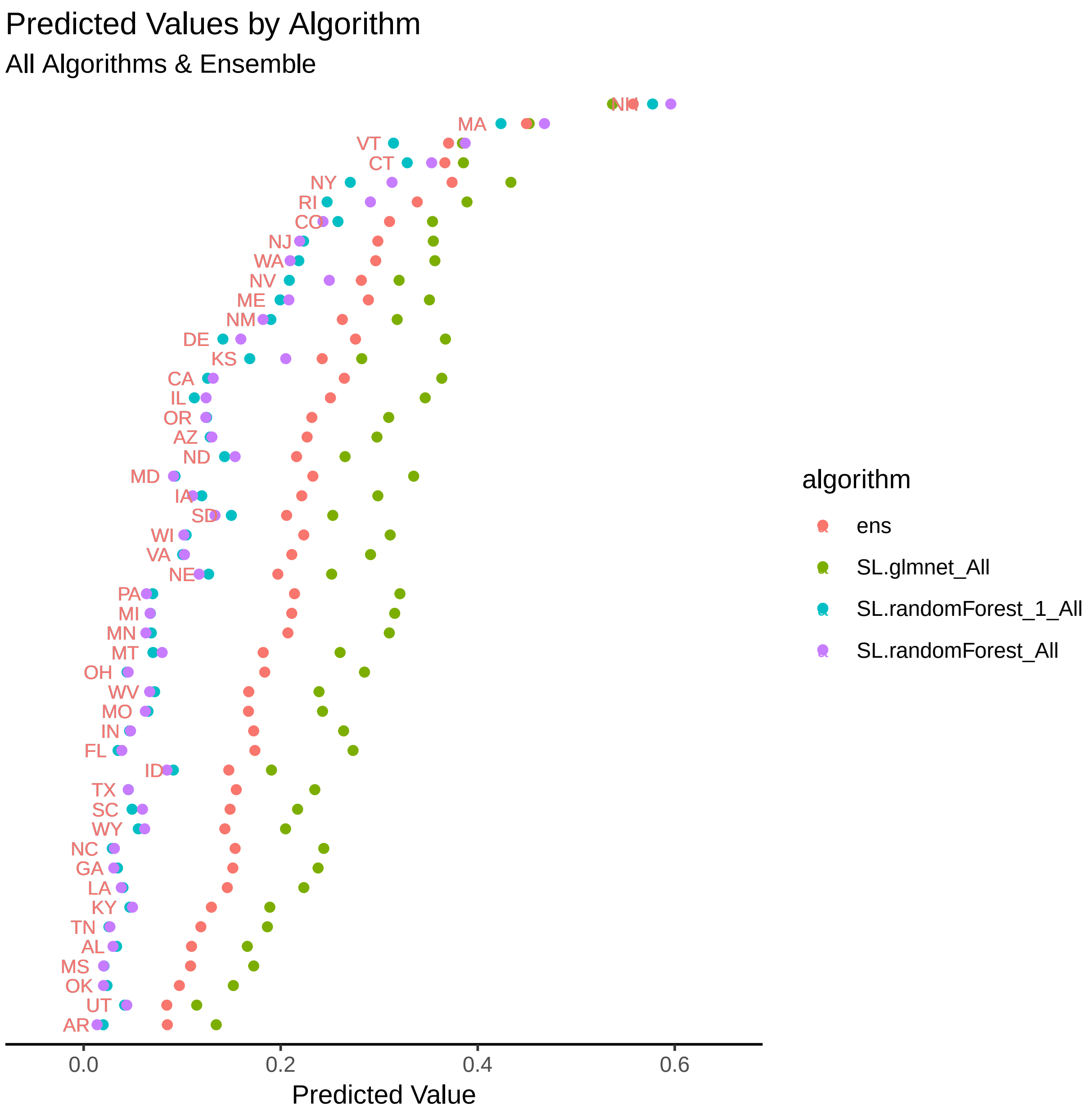
Finally, we can test multiple algorithms at once and examine which perform the best via cross validation by feeding in a vector of algorithm names to the `algorithm` parameter.

```
barp.objEns <- barp(y = "supp_gaymar",
                 x = c("age","educ","gXr",
                       "pvote","religcon",
                       "state","region"),
                 dat = svy,census = census06,
                 algorithm = c(rf_new$names,"SL.glmnet","SL.randomForest"),
                 geo.unit = "state",
                 proportion = "n",
                 setSeed = 1021)
```

```
barp.objEns$pred.opn %>% head()
```

```
##    state SL.randomForest_1_All SL.glmnet_All SL.randomForest_All
## AL    AL            0.03346351     0.1661456          0.02982427
## AR    AR            0.01981159     0.1345653          0.01366304
## AZ    AZ            0.12843085     0.2977428          0.13021941
## CA    CA            0.12583631     0.3635293          0.13147678
## CO    CO            0.25814103     0.3541205          0.24294918
## CT    CT            0.32847602     0.3854912          0.35322040
##           ens     opn.lb    opn.ub
## AL 0.10949208 0.02681255 0.1637349
## AR 0.08493234 0.01357357 0.1308434
## AZ 0.22680901 0.12184227 0.2934351
## CA 0.26460884 0.12359648 0.3572818
## CO 0.31048271 0.23437961 0.3644853
## CT 0.36670956 0.32045860 0.3985555
```

```
plot(barp.objEns)
```



As illustrated, the `SL.randomForest` algorithm returns very similar predictions when using the default number of trees (1,000) versus 150. We can further examine how these algorithms perform by looking at their cross-validated risk via the `risk` object in the output. The first column returns the performance metric (typically AUC for classification, NNLS for regression,although these can be adjusted by the user) and the second provides the weight associated with this algorithm in estimating the ensemble predictions.

```
barp.objEns$risk
```

```
##                             AUC      coef
## SL.randomForest_1_All 0.3560406 0.2097273
## SL.glmnet_All         0.3309479 0.5788130
## SL.randomForest_All   0.3529118 0.2114597
```

Some of these algorithms require bespoke packages which will need to be installed before running `barp`. In addition, others may only work for classification or regression and will give an error if applied to the incorrect data type.

5.0 Conclusion

This vignette has introduced and demonstrated the features of the **R** package **BARP**. The purpose of this package is to improve on the estimation of opinion at narrower levels of geography than originally represented in a survey. The package includes several helper functions designed to facilitate exploration of the model, both in terms of performance and in terms of covariates. I invite comments on bugs, corrections, improvements, and any other suggestions.

References

Bisbee, James. 2019. "BARP: Improving Mister P Using Bayesian Additive Regression Trees." *Working Paper*. jamesbisbee.com/research.

Bleich, Justin, Adam Kapelner, Edward I George, and Shane T Jensen. 2014. "Variable Selection for Bart: An Application to Gene Regulation." *The Annals of Applied Statistics*. JSTOR, 1750–81.

Buttice, Matthew K., and Benjamin Highton. 2013. "How Does Multilevel Regression and Poststratification Perform with Conventional National Surveys?" *Political Analysis* 21 (4): 449–67.

Chipman, Hugh A., Edward I. George, and Robert E. McCulloch. 2010. "BART: Bayesian Additive Regression Trees." *The Annals of Applied Statistics*, 266–98.

Honaker, James, Gary King, Matthew Blackwell, and others. 2011. "Amelia Ii: A Program for Missing Data." *Journal of Statistical Software* 45 (7): 1–47.

Kapelner, Adam, and Justin Bleich. 2013. "Bartmachine: A Powerful Tool for Machine Learning." *Stat* 1050: 8.

———. 2015. "Prediction with Missing Data via Bayesian Additive Regression Trees." *Canadian Journal of Statistics* 43 (2). Wiley Online Library: 224–39.

Lax, Jeffrey R., and Justin H. Phillips. 2009. "How Should We Estimate Public Opinion in the States?" *American Journal of Political Science* 53 (1): 107–21.

Polley, EC, and MJ Van der Laan. 2015. "SuperLearner: Super Learner Prediction.(Package Version 2.0-15)." *Vienna, Austria: R Foundation for Statistical Computing.*

Warshaw, Christopher, and Jonathan Rodden. 2012. "How Should We Measure District-Level Public Opinion on Individual Issues?" *The Journal of Politics* 74 (01): 203–19.