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 2018), 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 opposition to 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 (2018).

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. **Section 4** 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 recusively 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.

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:

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

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 (2018). 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.

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 = "-Xmx3g")
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 %>% select(state,stateid) %>% distinct())
factors <- c("region","gXr","state")
census06[factors] <- lapply(census06[factors],factor)
svy[factors] <- lapply(svy[factors],factor)
barp.obj <- barp(y = "supp_gaymar",
                 x = c("age","educ","gXr",
                       "pvote","religcon",
```

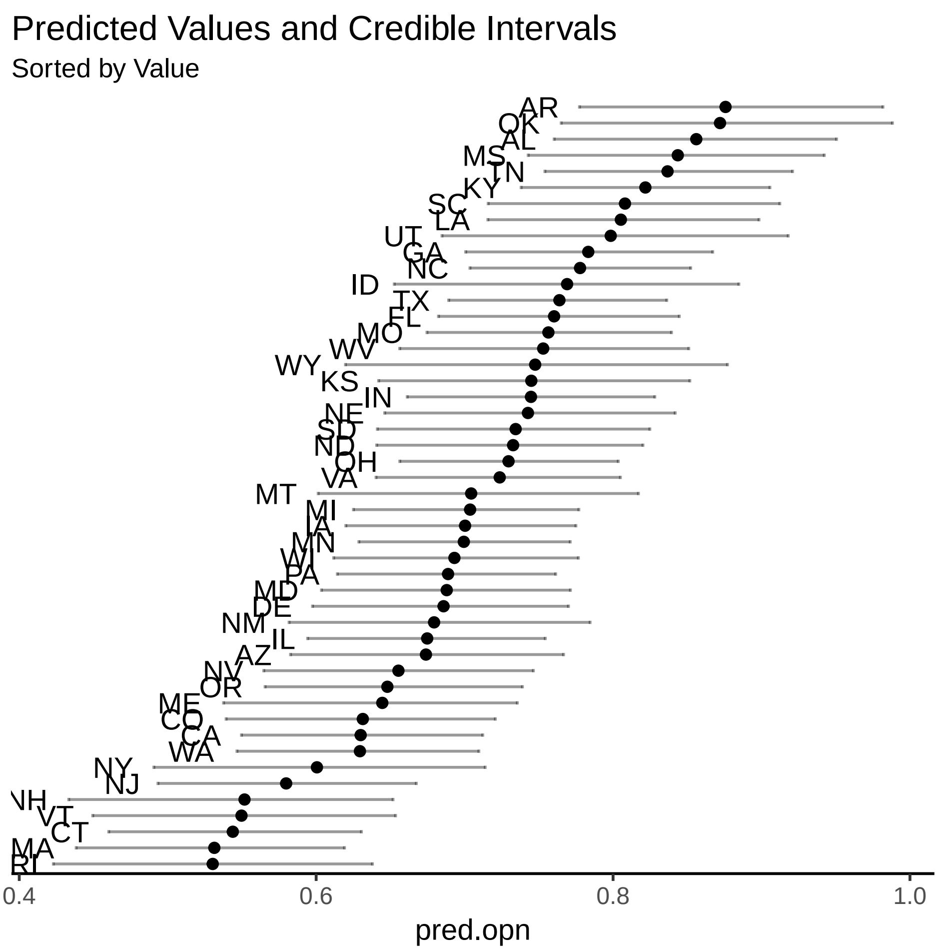
```
                    "state","region"),
               dat = svy,census = census06,
               geo.unit = "state",
               proportion = "n")
```

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).

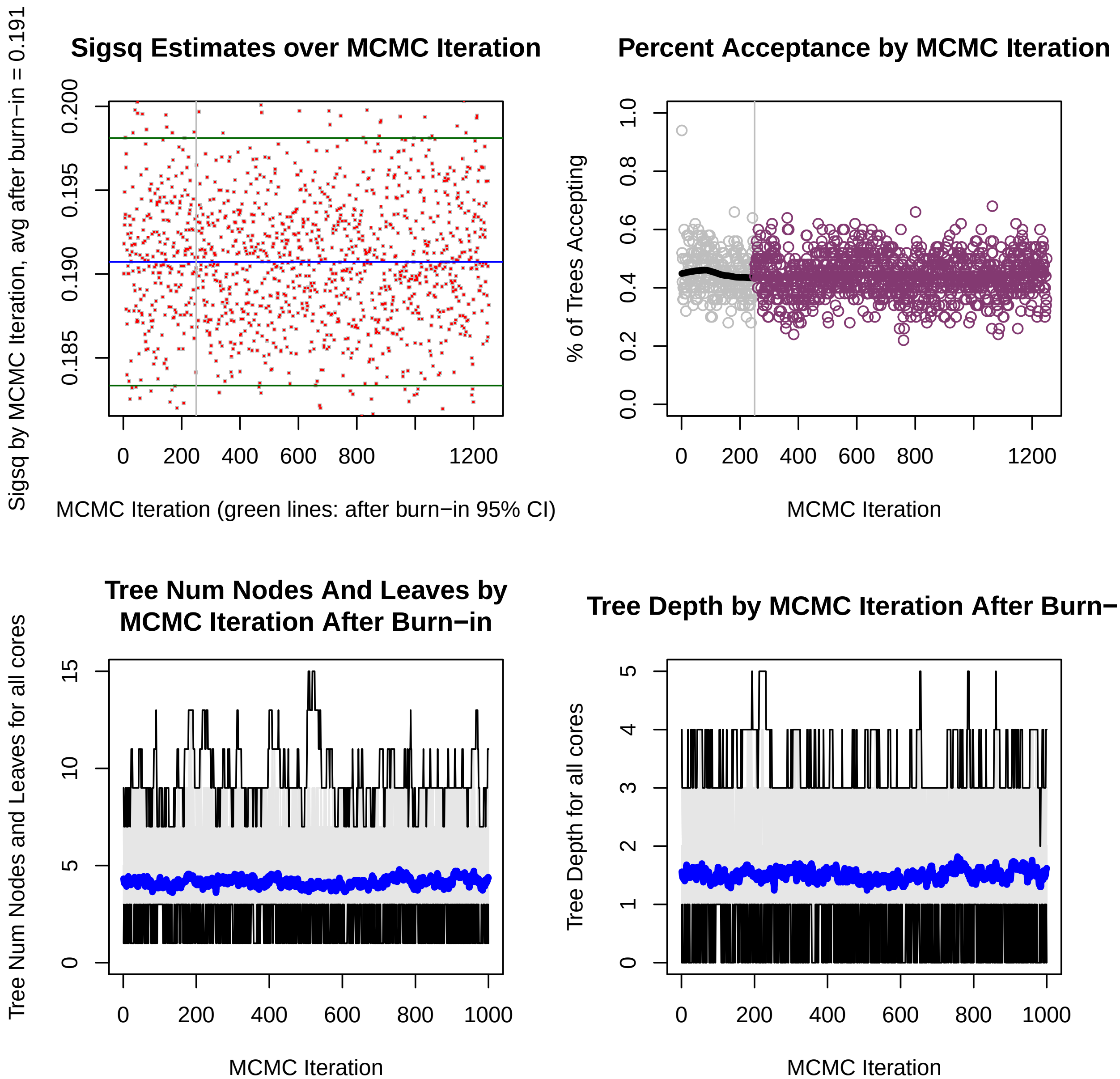
```
barp.obj$pred.opn %>% head()
```

```
##   state  pred.opn    opn.lb    opn.ub
## 1    AL 0.8560596 0.7603315 0.9505132
## 2    AR 0.8757631 0.7773733 0.9817825
## 3    AZ 0.6739561 0.5827422 0.7666447
## 4    CA 0.6299887 0.5496699 0.7122063
## 5    CO 0.6314296 0.5393474 0.7207066
## 6    CT 0.5438542 0.4602865 0.6304804
```

```
plot(barp.obj)
```



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

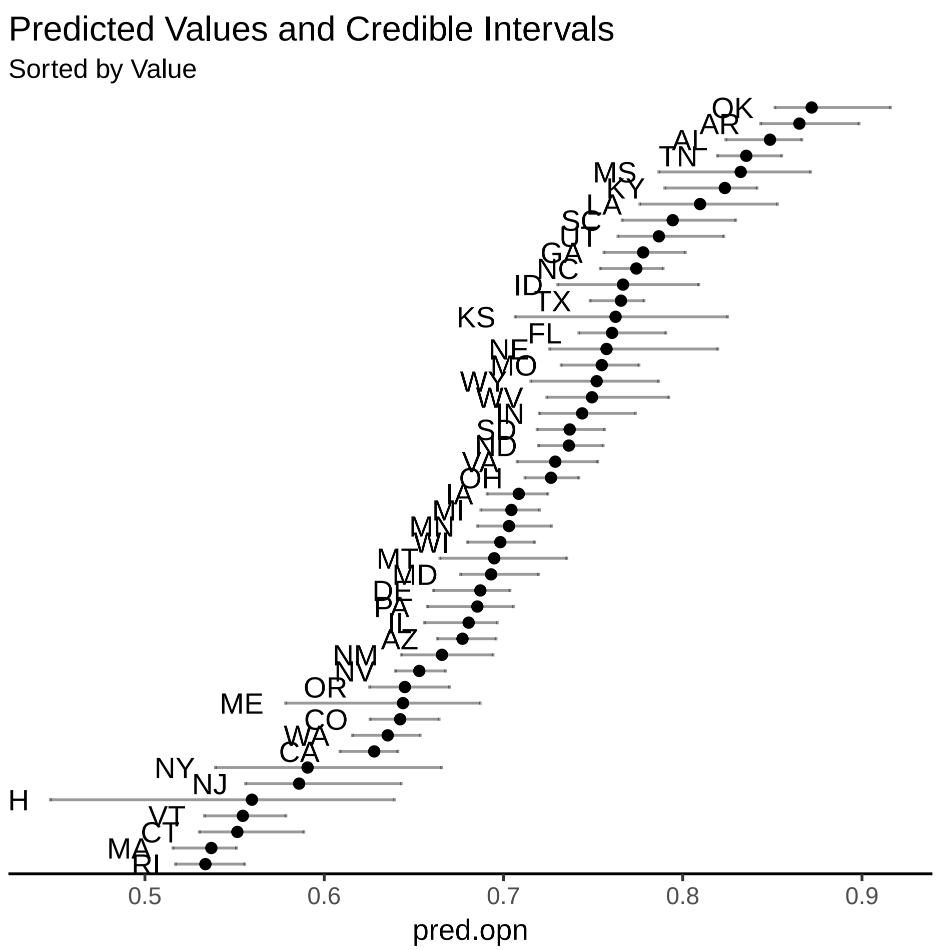


Alternatively, the user can choose to calculate the upper and lower bounds using bootstrapped simulations by setting `BSSD = TRUE` and defining the number of simulations through the `nsims` parameter. (Note that doing so will multiply the compute time accordingly). The user can set the credible intervals 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`.

```r
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 = 20,
                   num_trees = 20,
                   num_burn_in = 50,
                   num_iterations_after_burn_in = 50,
                   verbose = F)
```

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



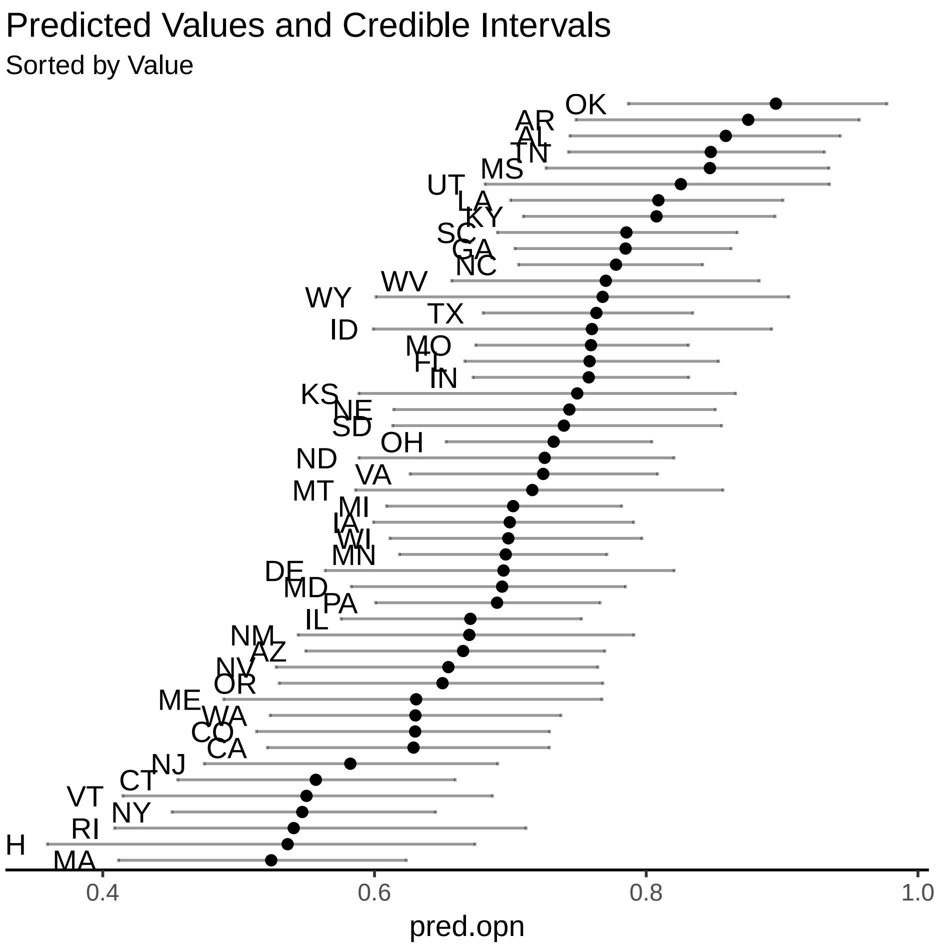
1.2.1 Classification vs regression

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",
```

```r
                    "pvote","religcon",
                    "state","region"),
              dat = svy,census = census06,
              geo.unit = "state",
              proportion = "n")
```

```r
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.

```r
barp.class$trees
```

```
## bartMachine v1.2.3 for classification
##
## training data n = 5000 and p = 62
## built in 14.2 secs on 1 core, 50 trees, 250 burn-in and 1000 post. samples
##
## confusion matrix:
##
##              predicted 1 predicted 0 model errors
## actual 1        3304.000     191.000        0.055
## actual 0        1226.000     279.000        0.815
```

```
## use errors       0.271      0.406       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 = 0.55)
```

```
barp.class2$trees
```

```
## bartMachine v1.2.3 for classification
##
## training data n = 5000 and p = 62
## built in 11.7 secs on 1 core, 50 trees, 250 burn-in and 1000 post. samples
##
## confusion matrix:
##
##            predicted 1 predicted 0 model errors
## actual 1      3182.000     313.000        0.090
## actual 0      1108.000     397.000        0.736
## use errors       0.258       0.441        0.284
```

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 user must indicate which covariates to explore (through the `vars` parameter) and define the values at which the partial dependence should be estimated (through the `levs` parameter). If left to the default NULL value, 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. The `levs` parameter must be a list of values corresponding to each variable.

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 = c("age","educ"),
                               prop_data = .2,
                               levs = list(c(1:4),c(1:4)),
                               credible_interval = c(0.025,0.975))
```

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 0.512 0.434 0.588
## 2   2    1 0.649 0.577 0.710
## 3   3    1 0.750 0.696 0.801
## 4   4    1 0.867 0.815 0.919
## 5   1    2 0.520 0.463 0.576
## 6   2    2 0.659 0.614 0.697
```

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

```
##   age1_educ1 age2_educ1 age3_educ1 age4_educ1
## 1      0.565      0.714      0.786      0.910
## 2      0.546      0.672      0.736      0.826
## 3      0.517      0.651      0.755      0.861
## 4      0.498      0.661      0.727      0.842
## 5      0.586      0.718      0.805      0.890
## 6      0.559      0.676      0.729      0.850
```

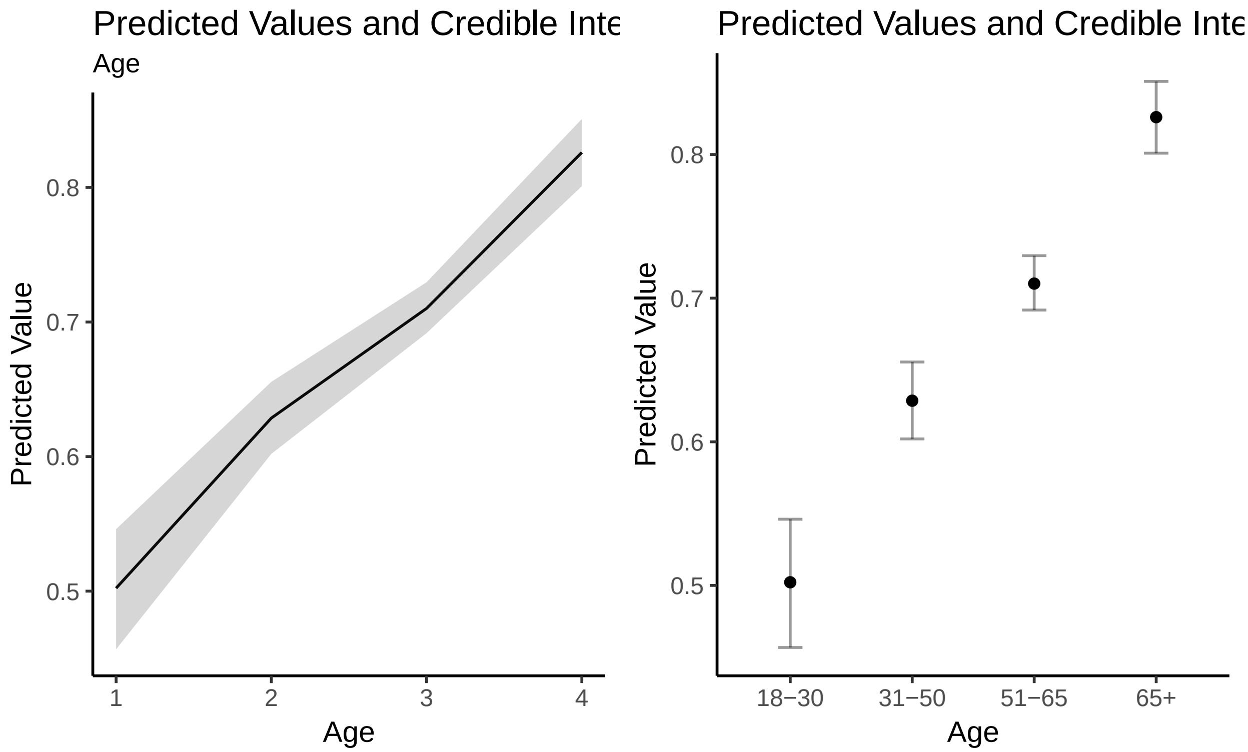
2.2 Plotting partial dependence

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.

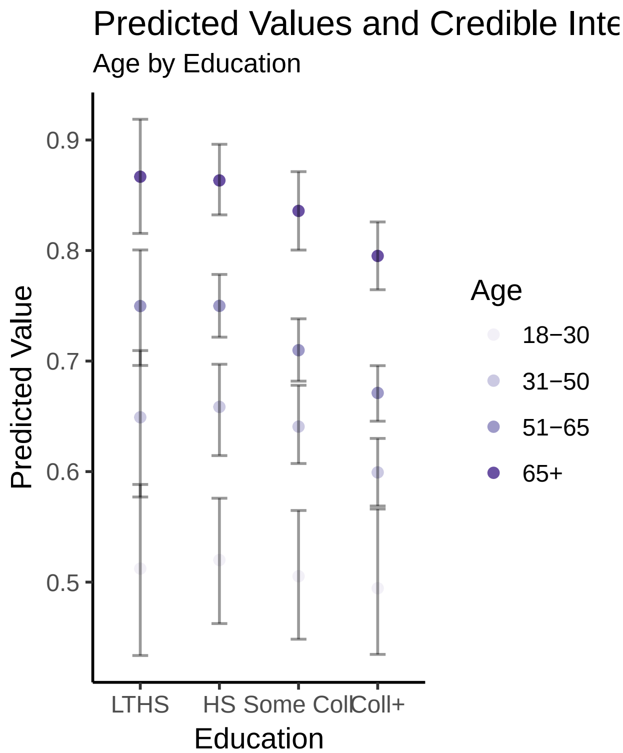
```
bpd1 <- barp_partial_dependence(barp = barp.obj,
                                vars = "age",
                                prop_data = .2)
```

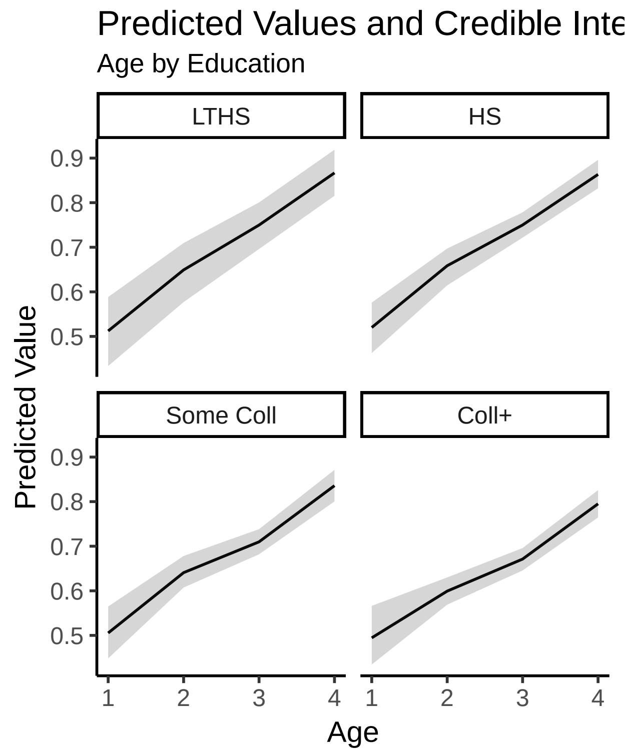
```
p1 <- plot(bpd1,var_names = "Age",
           is_categorical = F)
p2 <- plot(bpd1,var_names = "Age",
           var_labs = list(c("18-30","31-50","51-65","65+")),
           is_categorical = T)
grid.arrange(p1,p2,ncol = 2)
```



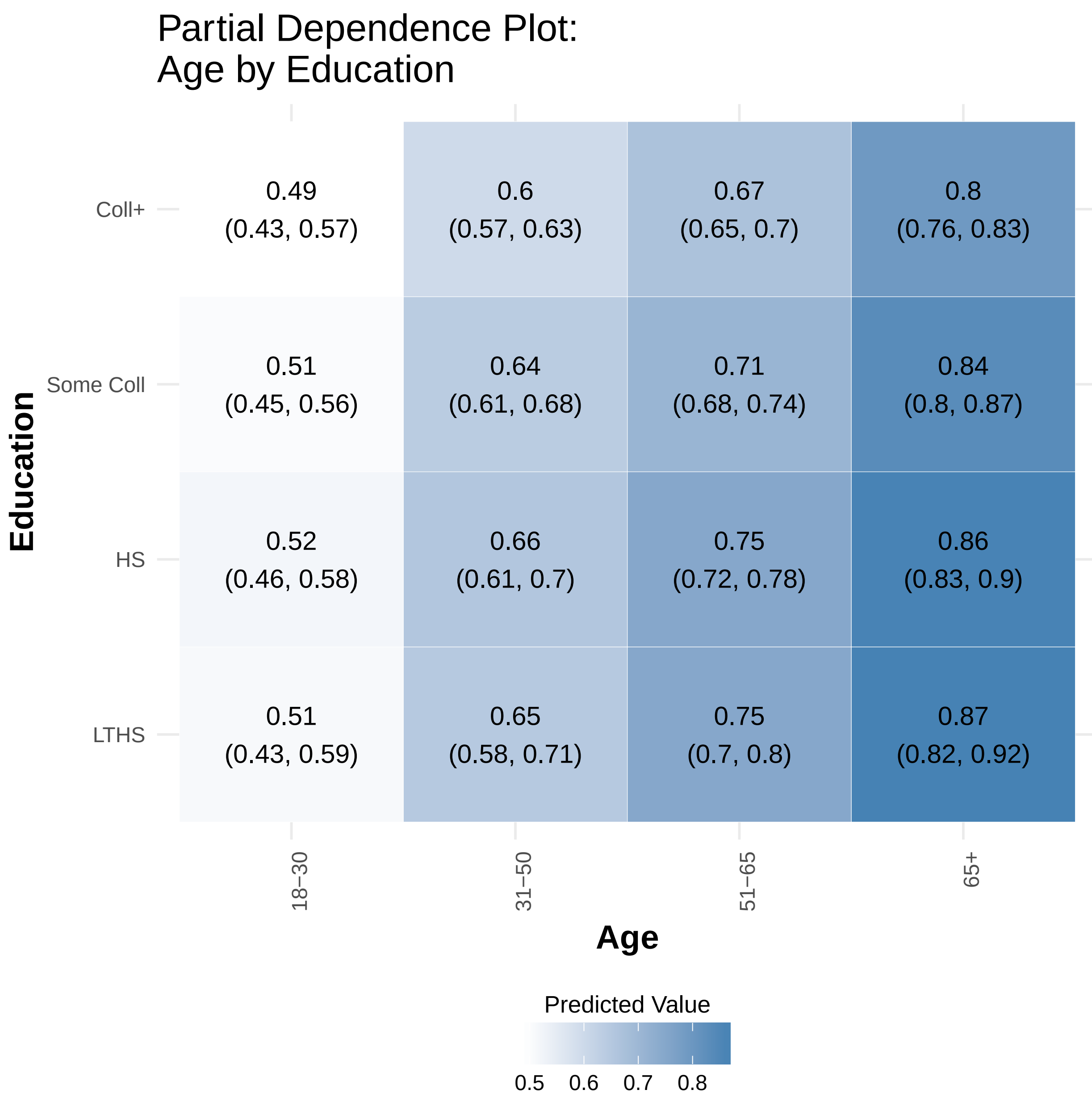
For an interaction analysis of two variables, there may be one of three different plots produced.

```
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)
```






```
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))
```

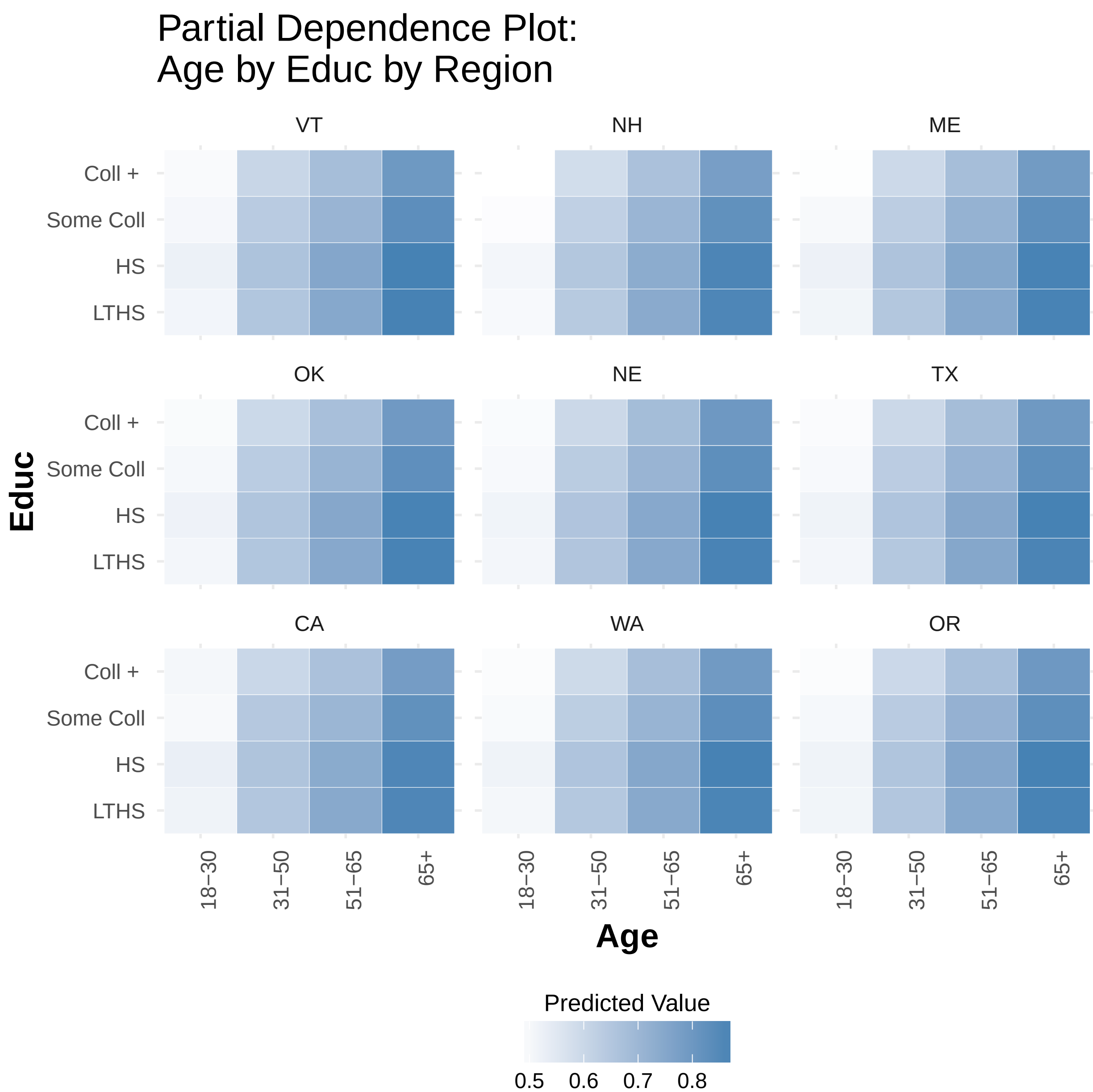


Finally, for a three-way interaction, the plot will be a grid of heatmaps, regardless of variable type. The third variable will always be the one used to organize the grid while the first and second variables will be the x- and y-axes respectively. Users should limit the number of levels in the `barp_partial_dependence`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

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

```
plot(bpd3,
     var_names = c("Age","Educ","Region"),
     var_labs = list(c("18-30","31-50","51-65","65+"),
```

```r
                  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.

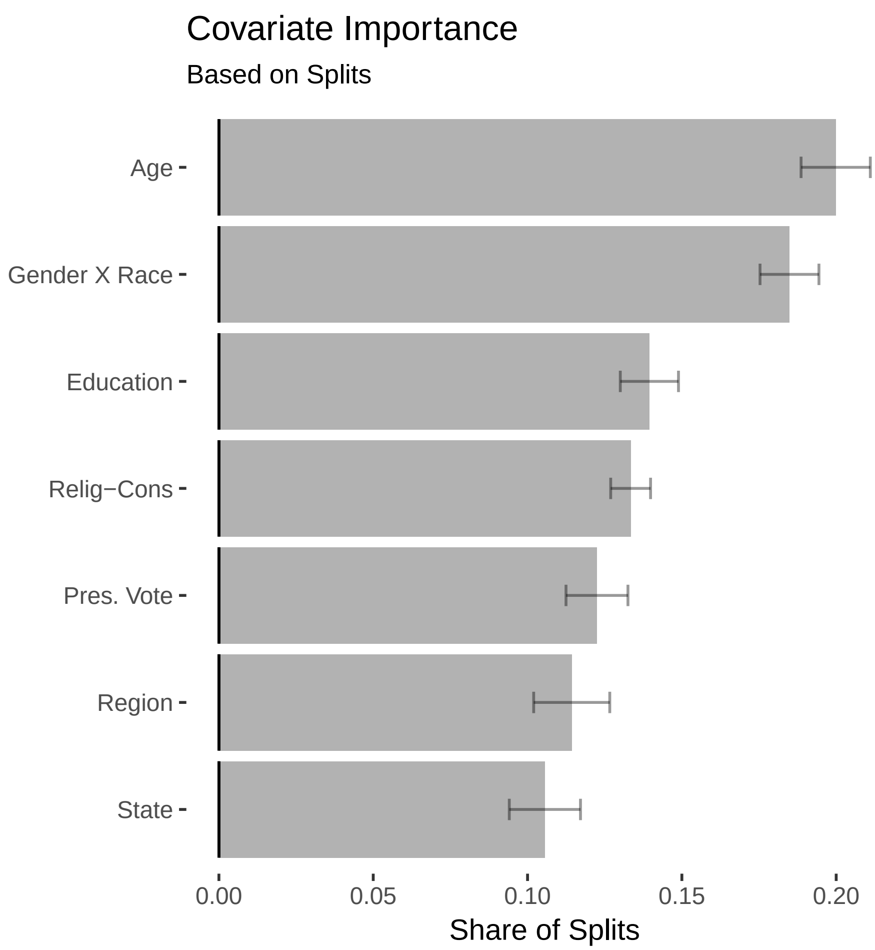
```
barpcov <- barp_prognostic_covs(barp.obj,
                                perm_test = F,
                                interactions = F,
                                num_reps = 10,
                                num_trees = 20,
                                type = "splits")
```

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

```
##        age  educ pvote religcon state region   gXr
## [1,] 0.209 0.145 0.135    0.130 0.097  0.096 0.187
## [2,] 0.212 0.139 0.123    0.137 0.101  0.113 0.177
## [3,] 0.211 0.131 0.108    0.129 0.124  0.114 0.182
## [4,] 0.189 0.148 0.111    0.130 0.114  0.117 0.192
## [5,] 0.197 0.157 0.127    0.124 0.107  0.123 0.165
## [6,] 0.194 0.138 0.129    0.130 0.122  0.095 0.192
```

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,
     var_names = c("Age","Education",
                   "Pres. Vote","Relig-Cons",
                   "State","Region",
                   "Gender X Race"))
```



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 = 10,
                                num_trees = 20,
                                type = "splits")
```

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

```
##        age  educ pvote religcon state region   gXr
## [1,] 0.133 0.125 0.140    0.134 0.174  0.139 0.155
## [2,] 0.149 0.140 0.159    0.142 0.161  0.114 0.136
## [3,] 0.136 0.154 0.128    0.128 0.164  0.132 0.158
```

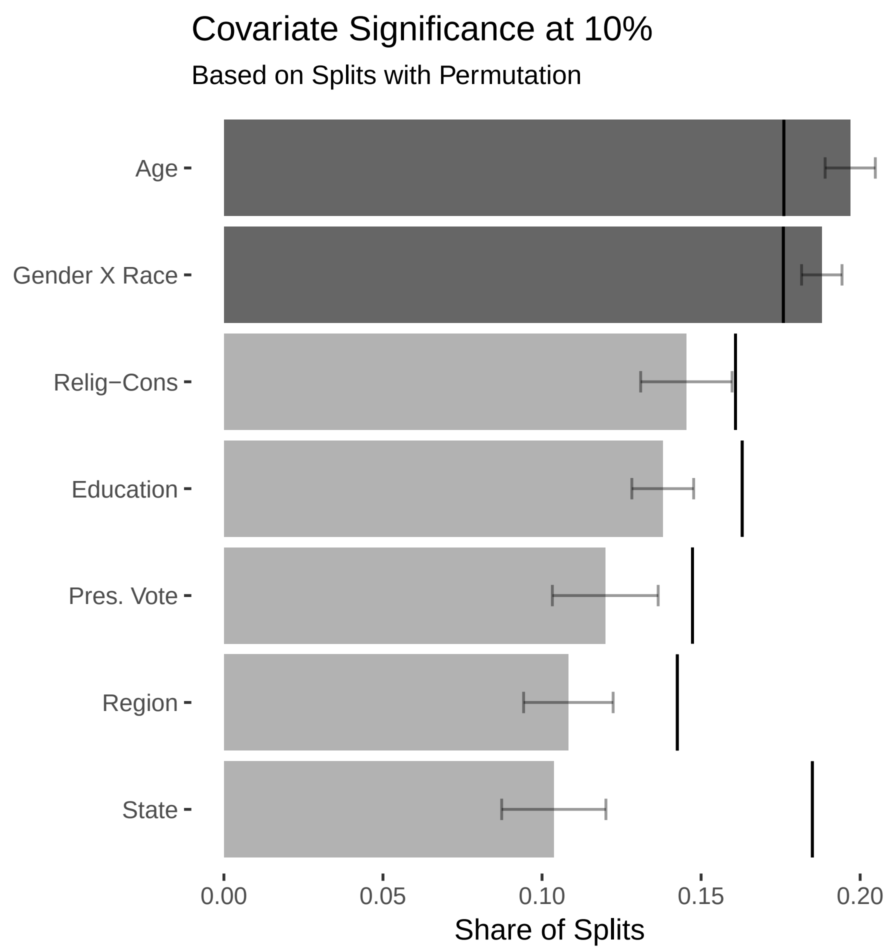
```
## [4,] 0.120 0.140 0.139    0.170 0.176  0.098 0.157
## [5,] 0.156 0.132 0.127    0.111 0.163  0.116 0.194
## [6,] 0.151 0.101 0.147    0.153 0.160  0.155 0.132
```

```r
round(barpcov_perm$p_vals,3)
```

```
##      age     educ    pvote religcon    state   region      gXr
##    0.000    0.703    0.713    0.327    1.000    0.832    0.030
```

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.

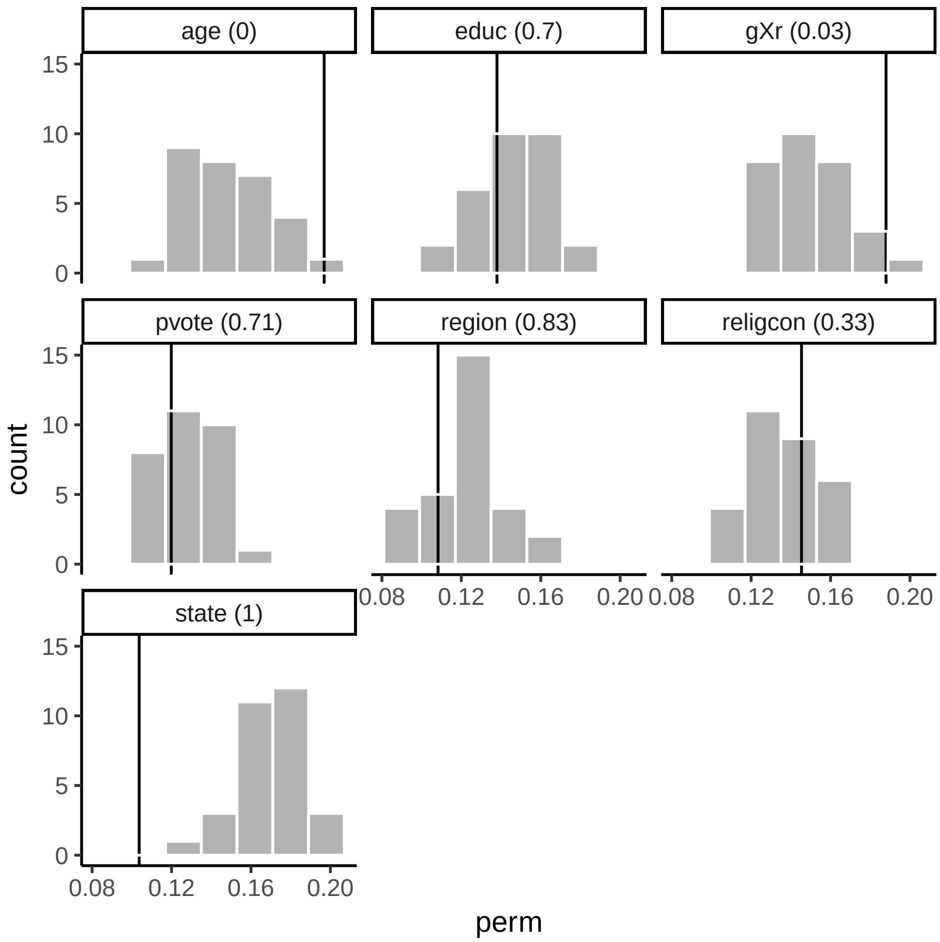
```r
plot(barpcov_perm,
     var_names = c("Age","Education","Pres. Vote","Relig-Cons","State","Region","Gender X Race"),
     sig_level = 0.10)
```



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.

```r
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$p_vals)
avgVIPs <- apply(barpcov_perm$covariate_importance,2,mean)
```

```r
avgVIPs <- data_frame(variable = names(avgVIPs),means = avgVIPs)
toplot <- toplot %>% left_join(pvals) %>% left_join(avgVIPs)
toplot$variable <- paste0(toplot$variable," (",round(toplot$pvals,2),")")
ggplot(toplot, aes(x=perm))+
  geom_vline(aes(xintercept = means), colour="black") +
  geom_histogram(binwidth=0.018,colour = "white",fill = rgb(0,0,0,.3)) +
  facet_wrap(~variable) +
  theme_classic()
```

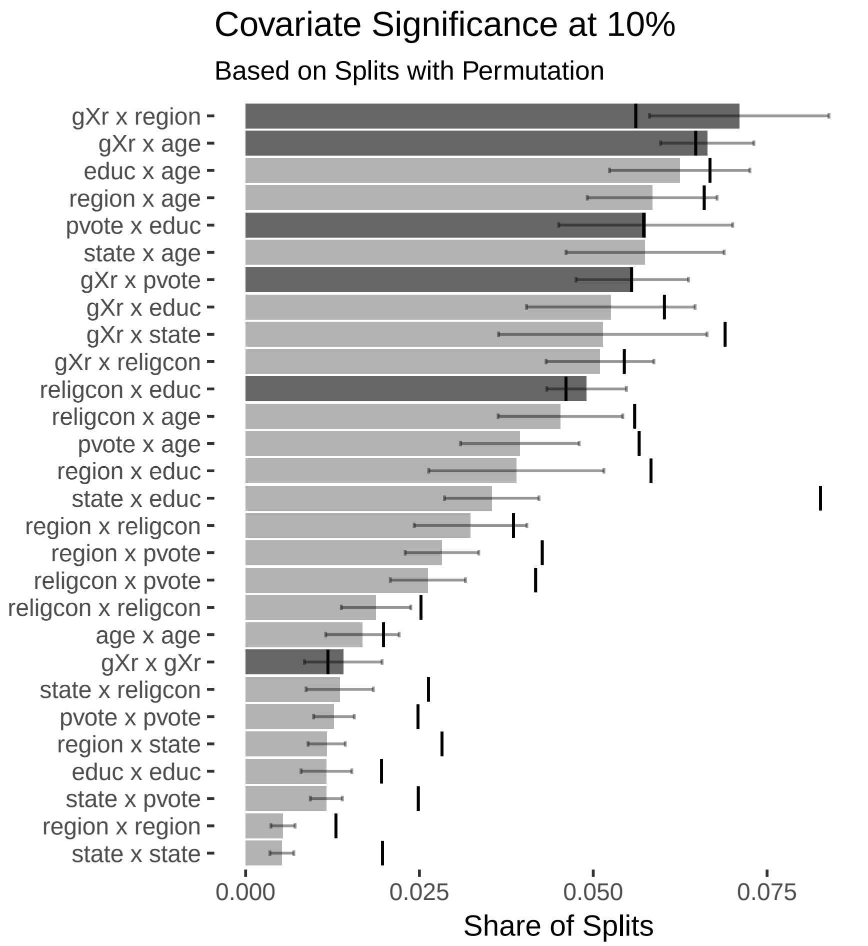


3.3 Interactions

These methods can also be applied to interaction terms.

```r
barpcov_int <- barp_prognostic_covs(barp.obj,
                                perm_test = T,
                                num_permute = 30,
                                interactions = T,
                                num_reps = 10,
                                num_trees = 20,
                                type = "splits")
```

```r
plot(barpcov_int,
     sig_level = 0.10)
```



4.0 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. 2018. "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.

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

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.

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.