Online Appendix for How to Cautiously Uncover the 'Black Box' of Machine Learning Models for Legislative Scholars

Soren Jordan*
Hannah L. Paul†
Andrew Q. Philips‡

Contents

*sorenjordanpols@gmail.com. Associate Professor, Auburn University.
†hannah.paul@colorado.eduu. Assistant Professor, University of Missouri.
‡andrew.philips@colorado.edu. Assistant Professor, University of Colorado Boulder.

1 What are Machine Learning models?

1.1 How does a Machine Learning model estimate a model?

The actual mechanics of Random Forest models have been well and accessibly summarized in greater detail elsewhere (Hastie, Tibshirani and Friedman 2013; Muchlinski et al. 2016; Montgomery and Olivella 2018; Molnar 2020), but we offer a brief description here. First, given a set of *k* predictors, $\mathbf{x}_i$, the algorithm selects a "cut-point"—a value of one of the predictors that partitions the data into two regions. Within each region, a single prediction is made. The algorithm then proceeds to find the next split in a region, makes a prediction, finds the next split, and so on. The algorithm will continue to create these splits (commonly called nodes) until some stopping criterion is satisfied, such as a minimum number of observations left in a region.

While the process above describes the construction of a single tree, known as a classification and regression tree (CART), Random Forests innovate on a single tree—which tends to overfit the data—in two major ways. First, by creating *B* bootstrap samples, on which a tree is grown on each sample, thousands of trees can be created. Thus, instead of making a single prediction from a single tree, "ensemble" predictions are made by averaging over all trees (hence the "Forest" in Random Forest). Second, by forcing the algorithm to choose from a random subset of predictors at each node, trees end up becoming decorrelated from one another. Perhaps counterintuitively, this actually *increases* predictive ability, since again we are averaging over all trees (Hastie, Tibshirani and Friedman 2013); while a single tree may be a bad predictor, the average prediction should be very accurate.

The number of trees, stopping criterion, and number of subset predictors to choose at each node are "hyperparameters" typically chosen through cross-validation in order to avoid any subjective user-specific decisions. Crucial hyperparameters to select are the number of trees to grow, the number of random predictors to force the algorithm to choose from at each node ("m-try"), and the minimum number of observations contained in the terminal nodes. We recommend the standard approach of a cross-validation procedure to select these hyperparameters (e.g., across possible combinations of the number of trees, terminal node size, and m-try, select the combination that produces the lowest mean squared error, across all trees, when out-of-bag observations are fed through). We detail how to establish these hyperparameters below.

1.2 How does a Machine Learning model make predictions?

Since they do not produce parameter estimates, tree-based models are often critiqued as "black-box" approaches. The model returns predictions, but these are generated by relatively obscured means and might be unsatisfying if we are asking classic questions like "what is the effect of a predictor on an outcome of interest?" (especially if we are accustomed to seeing tables of parametric estimates). Even when evaluating the predictions, we might still have residual questions like "why did a particular observation get predicted as it did"? Such questions are of key importance to analysts, yet they are difficult to answer with a machine learning model.

This is why tools of *interpretation*, in particular graphical approaches, are so important. As Molnar (2020, p. 20, bold in the original) puts it, "… for certain problems or tasks it is not enough to get the prediction (the **what**). The model must also explain how it came to the prediction (the **why**)." Not only has a growing literature grappled with better understanding machine learning models for the context of explanation and even causation (Kim, Khanna and Koyejo 2016; Doshi-Velez and Kim 2017; Zhao and Hastie 2019), a substantial amount of recent work has focused on doing so graphically (Friedman 2001; Goldstein et al. 2015; Apley and Zhu 2016; Ribeiro, Singh and Guestrin 2016). It is exactly these graphical tools that we recommend as methods of unpacking the "black box."

A common method of generating and validating the predictions used in graphical interpretations is to rely on the out-of-bag predictions (described in the main text), a form of out-of-sample prediction. A second method is to use another "holdout" dataset, often called test data, to make a final prediction (typically, to compare across a class of models) (Hastie, Tibshirani and Friedman 2013). Note that test data should only be used after model selection and diagnostics are complete. Still another way of making predictions uses cross-validation, which is somewhat similar to bootstrapping; for an example of this in political science, see Muchlinski et al. (2016).

2 How do I estimate and interpret a Machine Learning model in R?

2.1 Model estimation

2.1.1 Hyperparameters

First, we can establish a series of combinations of hyperparameters to use in model estimation. This can be handled through R's `expand.grid()` function. Generally, we define a function, `makeHyper`, that makes hyperparameters for all combinations of interest:

```
makeHyper <- function(min.mtry, max.mtry, mtry.increment,
  min.node, max.node, node.increment,
  min.tree, max.tree, tree.increment) {
 combinations <- expand.grid(mtry = seq(min.mtry, max.mtry, mtry.increment),
  node.size = seq(min.node, max.node, node.increment),
  tree.size = seq(min.tree, max.tree, tree.increment),
  OOB.RMSE = NA)
 combinations$comb <- 1:dim(combinations)[1]
 combinations
}
```

`makeHyper` will return a grid of combinations for us to search over to see which combination returns the best initial fit. It takes three dimensions of arguments: `mtry`, which is the number to try at each node, `node`, which is the number of observations within a terminal node, and `tree`, which is the number of trees to grow. Be careful with values of `mtry.increment`, `node.increment`, and `tree.increment`: low values will lead to really dense grids of hyperparameters to search over, which could be extraordinarily computationally intensive. We recommend incrementing by more than 1.

Once the potential hyperparameters are established, we can then loop over the grid of combinations. For each combination we estimate a Random Forest model using `ranger()`. For each estimated model, we store the prediction error for that combination. Our goal is to find the hyperparameter combination with the minimum prediction error. Since every tree is grown on a bootstrapped sample, we can use the observations that were *not* randomly selected in a given bootstrap as a holdout sample (the "out-of-bag" sample) to help select hyperparameters. Across every combination of hyperparameters, we calculated the average mean squared error obtained by using out-of-bag observations to feed through each respective tree. The hyperparameter combination with the lowest mean squared error is the optimal combination to use.

Machine learning models are easiest to estimate in `R` by defining a dataframe that contains *only* the dependent variable and relevant independent variables (without any other variables that are not going to be used in model estimation). For instance, if we have a large dataset with many variables, `data`, we could create a smaller dataset, `data.small`, through code like

```
data.small <- data[,names(data) %in% c("[name1]", "[name2]", ...)]
```

where we would replace `name1` and `name2` with the names of the dependent and all independent variables we want in our model. From there, if we had already created a dimension of hyperparameter combinations `model.combinations` using the `makeHyper` function above, we could loop over them with

```
for(i in 1:nrow(model.combinations)) {
 the.model <- ranger(formula = y ~ .,
  data = data.small,
  num.trees = model.combinations$tree.size[i],
  mtry = model.combinations$mtry[i],
  min.node.size = model.combinations$node.size[i])
 model.combinations$OOB.RMSE[i] <- sqrt(the.model$prediction.error)
}
```

This loops over all combinations of hyperparameters in `model.combinations`, estimates the Random Forest model of `y` on all the other variables in `data.small`, and saves the prediction error for each combination in `model.combinations$OOB.RMSE`. Once this is done, we could isolate the hyperparameters with the lowest prediction error by using

```
model.combinations[which(model.combinations$OOB.RMSE ==
 min(model.combinations$OOB.RMSE)),]
```

If we wanted a way to isolate the row number (which would make it easier to pass the combination to the full model estimation function), we could use the column `comb` of the hyperparameters.

```
model.combinations[which(model.combinations$OOB.RMSE ==
 min(model.combinations$OOB.RMSE)), "comb"]
```

To then fit the model, we'd just use the same dataset and pull the hyperparameters from the grid search. Assuming the same object names from above, we would first isolate the best set of hyperparameters, `best`, and then pass the set to `randomForest`.

```
best <- model.combinations[which(model.combinations$OOB.RMSE ==
 min(model.combinations$OOB.RMSE)), "comb"]

model <- randomForest(y  ~ .,
 data = data.small,
 ntree = model.combinations[model.combinations$comb == best,]$tree.size,
 mtry = model.combinations[model.combinations$comb == best,]$mtry,
 nodesize = model.combinations[model.combinations$comb == best,]$node.size,
 importance = TRUE)
```

We now illustrate this practically using our first example, (Howard and Owens 2020). First, we define the hyperparameters: we increment `mtry` and `node` by 2 and `tree` by 100. We then create the grid of parameters `hyperparameter.grid.HO` using `makeHyper`.

```
hyperparameter.grid.HO <- makeHyper(
 min.mtry = 2, max.mtry = 12, mtry.increment = 2,
 min.node = 2, max.node = 11, node.increment = 2,
 min.tree = 200, max.tree = 2000, tree.increment = 100)
```

We would pass a dataframe with the dependent variable and independent variables. Assuming the dataframe is `ho.rf.data.nona.nointaxn`, and the dependent variable is `bypass`, we would fit

```
for(i in 1:nrow(hyperparameter.grid.HO)) {
 the.model <- ranger(formula = as.factor(bypass) ~ .,
  data = ho.rf.data.nona.nointaxn,
  num.trees = hyperparameter.grid.HO$tree.size[i],
  mtry = hyperparameter.grid.HO$mtry[i],
  min.node.size = hyperparameter.grid.HO$node.size[i])
 hyperparameter.grid.HO$OOB.RMSE[i] <- sqrt(the.model$prediction.error)
}
```

Notice that the last line saves the prediction error from each combination. We'd then choose the combination parameter set with the lowest prediction error.

```
the.combo <- hyperparameter.grid.HO[which(hyperparameter.grid.HO$OOB.RMSE ==
 min(hyperparameter.grid.HO$OOB.RMSE)), "comb"]
```

To then fit the model, we'd just use the same dataset. Notice we're pulling the hyperparameters from the grid search.

```
full.rf.ho.nointaxn <- randomForest(as.factor(bypass)  ~ .,
 data = ho.rf.data.nona.nointaxn,
 ntree = hyper.grid.ho.nointaxn[hyper.grid.ho.nointaxn$comb ==
  the.combo,]$tree.size,
 mtry = hyper.grid.ho.nointaxn[hyper.grid.ho.nointaxn$comb ==
  the.combo,]$mtry,
 nodesize = hyper.grid.ho.nointaxn[hyper.grid.ho.nointaxn$comb ==
  the.combo,]$node.size,
 importance = TRUE)
```

2.1.2 Global interpretation

Although we specifically advocate for using graphical interpretation methods to unpack the potential inferences from machine learning models, we offer a few points on global model interpretation here.

First, it's important to communicate the cross-validation conducted for the model tuning parameters. This is akin to describing the `makeHyper` process. For instance, we would report that to set our hyperparameters (i.e., tuning parameters), we used a cross-validation procedure that was run over a grid search of different combinations of hyperparameter values. Specifically, we set the number of variables to randomly choose from at each node split ("m-try"), the minimum number of observations allowed in a terminal node, and the number of *B* trees to construct.

We can also generally assess model fit, although our emphasis in this text is on interpretation. Model fit is typically done by examining average mean squared error (MSE), especially if the problem is a regression problem, but we could also use some measure of classification accuracy if the dependent variable is categorical. This measure of fit is calculated on the out-of-bag samples. Often machine learning models entered into prediction competitions have a final "test" sample that is used to pick a winning model. We could of course compare our MSE against that of a parametric model. But since we are advocating for the use of tree-based models for learning and explanation, purely focusing on predictive accuracy is not a large part of our emphasis: we will typically always find that tree-based models are much more accurate (i.e., fit better) than parametric models. For a good example of model prediction across both parametric and machine learning models in political science, see Muchlinski et al. (2016).

2.1.3 Using Machine Learning with fixed effects

In the main manuscript, our replication of Howard and Owens (2020) included fixed effects for both the various Congressional sessions included in the sample, as well as the specific policy areas. Such variables are typically termed factors in the machine learning literature, since, unlike simple dummy variables (e.g., a senator voted "yes" or "no"), these represent categorical variables with more than two categories (although they are functionally estimated as a series of dummy variables: if it *is* one Congressional session, it's *not* any other Congressional session). To remain consistent with Howard and Owens, we also include these fixed effects in the same format they did. In the machine learning context, though, there is a clear drawback: in a given tree, while some of these fixed effects may be included, others will likely not be (e.g., the 110th and 107th Congress session dummies were included, but not the 108th or 109th). While this is likely much less of a problem for a machine learning model than for a parametric model—from a statistical standpoint, the final predictions or inferences are done by averaging over hundreds or even thousands of trees, meaning that all of the fixed effects were likely used for at least *some* trees—this approach fails to treat these fixed effects as, conceptually, a *single* variable, as one reviewer noted. As such, we note three different potential methods of incorporating such fixed effects into their machine learning models.

First, one might not include any fixed effects. This is likely not a good idea, as there may be variation explained by the fixed effects that is important. Moreover, given that Random Forests can easily handle more predictors than parametric models, we should probably err on the side of including more predictors, not less.

Second, one might include all fixed effects, coded as unique dummy variables. This is our current approach described above in the Howard and Owens example. Consider a categorical variable with categories A, B, C, and D that we turn into dummy variables. Inclusion of dummy variable A at a particular node in a tree, for instance, therefore partitions the data in that branch into A versus not-A (i.e., $\{B,C,D\}$). The advantage of this approach is it does not ask much of the algorithm at each node split; in our example, the algorithm picks the A/not-A dummy versus whatever other candidate predictors are possible to select at that node. The disadvantage, of course, is that we are really only splitting into one group versus all other groups at a time. In other words, we are treating each category (relative to all others) in isolation. Given, however, that we write specifically to legislative scholars, we're aware of the ubiquity of fixed effects for Congresses, states, years, or parties that are commonly used as categorical indicators: so this approach is probably the most consistent with the current legislative literature.

A third option, unique to Random Forests (compared to standard parametric approaches), involves including the variable as a single categorical variable, not as separate dummy variables as done above. If this categorical variable is selected for a given node, the algorithm will search through all possible combinations of categories in order to partition the data into two groups, in what is known as a "binary criteria split." For instance, it would try $\{A\}$ versus $\{B,C,D\}$, $\{A,B\}$ versus $\{C,D\}$, $\{A,B,C\}$ versus $\{D\}$, and so on. This approach keeps all categories together, and although it will only partition the categories into two classes, keep in mind these classes may differ substantially between different nodes and trees. The downside is this approach is computationally complex; with c categories and only a binary split, there are still $2^{c-1}-1$ possible splits to consider. Thus, this approach is not feasible if there are a large number of categories; indeed, to the best of our knowledge the `randomForest` package only allows for up to 53 distinct categories.

2.2 Graphical interpretations

2.2.1 Variable Importance Plots

Assuming that we fit the Random Forest model with `randomForest`, creating a VIP is relatively easy, as the importance measures are stored in `model$importance`. Liaw and Wiener (2018), in the `randomForest` documentation, define the importance measure. For each tree, the prediction error on the out-of-bag portion of the data is recorded (for classification, like the Howard and Owens (2020) example, this would be the classification error rate; for regression, like the Poyet and Raunio (2020) example, this would be the mean squared error, or MSE). This is done again after permuting each predictor variable. The difference between the two are then averaged over all of the trees. This measure can then be normalized by the standard deviation of the differences; this measure is accessible in `model$importanceSD`.

Variable Importance Plots, then, are most easily created in `R` following a five-step structure:

1. Save the variables and importance measures from the model.
2. Ensure the measures are explicitly numeric for plotting.
3. Create the variable labels (for publication).
4. Order the importance measure for plotting.
5. Pass the data to `ggplot`.

Using the Howard and Owens (2020) example, given the Random Forest model is saved as `full.rf.ho.nointaxn`, we would first frame the variable names and the importance measure

```
ho.X.vars.nointaxn.vip.dat <- data.frame(
 names = rownames(full.rf.ho.nointaxn$importance),
 MDA = full.rf.ho.nointaxn$importance[,1]/full.rf.ho.nointaxn$importanceSD)
```

We can add an explicitly numeric version of the variable back to the data:

```
ho.X.vars.nointaxn.vip.dat$MDA <-
 as.numeric(ho.X.vars.nointaxn.vip.dat$MDA.MeanDecreaseAccuracy)
```

We're now ready to plot. We can define less-ugly variable labels. For instance, for Figure 2 in the main text, we added a set of variable names through

```
ho.X.vars.nointaxn.vip.dat$Variable <- c("Polarization", "Cosponsors",
 "Bills Introduced", "Sponsor Seniority", "Time Remaining",
 "Extremity", paste0(100:113, " Congress"),
 paste0("Major Area ", c(2:10, 12:21, 99)),
 "Minority Sponsor", "Committee Chair", "Floor Leader",
 "Up for Election", "Duplicate Bill",
 "Nontrivial Bill", "Party Bill")
```

If we wanted to ignore the labels for fixed effects (or other substantively uninteresting predictors), we could replace those terms with a blank quotation to plot a blank label.

```
ho.X.vars.nointaxn.vip.dat$Variable2 <- c("Polarization", "Cosponsors",
 "Bills Introduced", "Sponsor Seniority", "Time Remaining",
 "Extremity", rep(" ", length(100:113)),
 rep(" ", length(c(2:10, 12:21, 99))),
 "Minority Sponsor", "Committee Chair", "Floor Leader",
 "Up for Election", "Duplicate Bill",
 "Nontrivial Bill", "Party Bill")
```

If we want to distinguish key theoretical variables, we could use a simple indicator. For instance, to distinguish the key variables of Howard and Owens (2020) (extremity, minority sponsor, and committee chair), we could create an indicator (in the `ho.X.vars.nointaxn.vip.dat` dataset) through

```
ho.X.vars.nointaxn.vip.dat$ho.key.var <- "no"
ho.X.vars.nointaxn.vip.dat$ho.key.var[ho.X.vars.nointaxn.vip.dat$Variable ==
 "Extremity"] <- "yes"
ho.X.vars.nointaxn.vip.dat$ho.key.var[ho.X.vars.nointaxn.vip.dat$Variable ==
 "Minority Sponsor"] <- "yes"
ho.X.vars.nointaxn.vip.dat$ho.key.var[ho.X.vars.nointaxn.vip.dat$Variable ==
 "Committee Chair"] <- "yes"
```

Finally, we sort the values by importance and pass to `ggplot`

```
ho.X.vars.nointaxn.vip.dat.sort <- ho.X.vars.nointaxn.vip.dat %>%
 arrange(-MDA)
```

In the plot, notice that the fill aesthetic is the `ho.key.var` indicator we just established.

```
ho.X.vars.nointaxn.vip.plot <- ggplot(ho.X.vars.nointaxn.vip.dat.sort,
  aes(x = reorder(Variable, -MDA), y = MDA, fill = ho.key.var)) +
 geom_bar(stat = "identity") +
 theme_minimal() +
 theme(axis.text.x = element_text(angle=75, hjust = 1),
  legend.position = "none") +
 scale_fill_manual(values = c("yes" = "black", "no" = "grey50")) +
 scale_x_discrete(labels = ho.X.vars.nointaxn.vip.dat.sort$Variable2) +
 xlab("") +
 ylab("Mean Decrease Accuracy")
```

Again, this measure is interpreted as a way of seeing the relative "importance"—in terms of predictive accuracy—of each variable in the Random Forest. Recall that this is cleverly done by permuting a predictor and seeing how much *worse* the prediction becomes relative to the unpermuted predictor. Thus, more important predictors would have a much larger decrease in accuracy when we permute them.

We could also use `varImpPlot()`, a part of the `randomForest` package, but the plots are rather crude and are inconsistent with many of the `ggplot` visualizations we typically use.

2.2.2 Partial Dependence Plots

Assuming that we fit the Random Forest model with `randomForest`, creating a PDP is also relatively easy. PDPs show the effect of a predictor while averaging over the effects of other predictors. It serves as an intuitive depiction of the relationship between a predictor of interest and the dependent variable, while basically “controlling” for all other effects of other predictors. This is especially straightforward using the `partial` function from the pdp package. Partial Dependence Plots, then, are most easily created in `R` following a three-step structure:

1. Define the prediction function.
2. Create the partials using `partial`.
3. Pass the data to `ggplot`.

Using the Howard and Owens (2020) example, we would first define the prediction function. Since this is a classification problem (bypassing the committee is a dummy indicator), we need to write a prediction function that isolates the probabilities. We could write a more generic function, but to illustrate the point precisely, if the Random Forest results are in `full.rf.ho.nointaxn`, we could write:

```
ho.X.vars.nointaxn.pdppred <- function(object, newdata) {
 predict(full.rf.ho.nointaxn, newdata, type = 'prob')[, 2]
}
```

For our second example, Poyet and Raunio (2020), it’s a regression problem (number of speeches is much closer to continuous), so we need to write a prediction function that just calculates the predicted values:

```
pr.X.vars.nointaxn.pdppred <- function(object, newdata) {
 predict(full.rf.pr.nointaxn, newdata)
}
```

With the prediction function established, we can return the predictions really easily. Returning to our first example, if we wanted the PDP for member extremity, we would run:

```
the.pdp.ext <- partial(full.rf.ho.nointaxn,
 pred.var = "extremity100_scaled",
 pred.fun = ho.X.vars.nointaxn.pdppred)
```

The first argument is the model, the second is the variable for which we want the PDP (`extremity100_scaled`), and the third is the prediction function we just wrote. Finally, we just plot the resulting dataframe (`the.pdp.ext`):

```
ext.pdp <- ggplot() +
 stat_summary(data = the.pdp.ext, aes(x = extremity100_scaled, y = yhat),
  fun = mean, geom = "line", col = "black", size = 2) +
 theme_minimal() +
 xlab("Extremity") +
 ylab("Predicted Value") +
 geom_rug(data = ho.rf.data.nona.nointaxn, aes(x = extremity100_scaled))
```

If we wanted to retain the standard deviation of the partials (for something like confidence intervals), we could write another prediction function:

```
ho.X.vars.nointaxn.pdppred.sd <- function(object, newdata) {
 preds <- predict(object, newdata, type = ’prob’)[, 2]
 c("mean" = mean(preds),
  "lower" = mean(preds) - sd(preds),
  "upper" = mean(preds) + sd(preds))
}
```

We could then aggregate these partials for plotting:

```
the.pdp.ext.sd.wide <- data.frame(extremity100_scaled =
  the.pdp.ext.sd$extremity100_scaled[the.pdp.ext.sd$yhat.id == "mean"],
 lower = the.pdp.ext.sd$yhat[the.pdp.ext.sd$yhat.id == "lower"],
 mean = the.pdp.ext.sd$yhat[the.pdp.ext.sd$yhat.id == "mean"],
 upper = the.pdp.ext.sd$yhat[the.pdp.ext.sd$yhat.id == "upper"])

ext.pdp.sd <- ggplot(data = the.pdp.ext.sd.wide,
  aes(x = extremity100_scaled, y = mean)) +
 geom_line(lwd = 2) +
 geom_ribbon(aes(ymin = lower, ymax = upper, alpha = 0.2,
  linetype = "dashed")) +
 theme_minimal() +
 theme(legend.position = "none") +
 xlab("Extremity") +
 ylab("Predicted Value")
```

There’s no agreement on whether adding these pseudo-confidence-intervals is accepted practice. Evans and Murphy (2019) explicitly recommend it as an extension to drawing the PDP, and, of course, there’s nothing wrong with shading or indicating an area where some percent of the predictions fall. However, it *would* be incorrect to refer to these intervals as real “confidence intervals,” and it would *certainly* be incorrect to use them to test a hypothesis. We reiterate again: the PDP is just averaging over many predictions, so it is not equipped to test a hypothesis in the standard statistical-significance framework. Thus, we only recommend adding these shaded areas if the analyst is explicitly clear that they are purely illustrative.

Moreover, as we discuss below, a better strategy may exist that would also help illustrate the heterogeneity in the predictions (i.e. the region we would ultimately “shade” on the PDP). If we want to envision the heterogeneity of the PDP, we might actually plot individual conditional expectations, rather than trying to plot the standard deviations of the combined predictions. This is the Individual Conditional Expectation plot discussed in the next section.

If we wanted a two-way PDP, like an interaction, all we have to do is pass multiple variables to `partial`. For instance, if we wanted it based on extremity and minority party status, we would pass:

```
the.pdp.ext.min <- partial(full.rf.ho.nointaxn,
 pred.var = c("extremity100_scaled", "minority_fac"),
 pred.fun = ho.X.vars.nointaxn.pdppred, chull = TRUE)
```

From there, we would plot the resulting partials as normal. Such interactive PDPs can uncover "hidden" interactions between predictors; for a good example of this see Funk, Paul and Philips (2021). One word of caution with interactive PDPs is that we probably only want to create PDPs for interactive scenarios for which the data were actually observed; Random Forests perform badly when extrapolating to data points that are extreme or atypical values never observed in the dataset (Greenwell 2017). For this reason, we should only show PDPs inside the "convex hull", which can be done using the option `chull = TRUE`.

2.2.3 Individual Conditional Expectation Plots

Creating an ICE plot is extraordinarily straightforward. In the above, drawing the PDP required that we average over the predictions for every observation along each value of our variable of interest; recall that PDPs hold the value of all control variables, $\mathbf{x}_{ic}$ at their actual values for each observation as we vary our variable of interest, x_s, and create prediction functions $\hat{f}^{(i)}$ for each observation. From this we take the average to create the PDP. In contrast, an ICE plot simply draws the actual prediction functions $\hat{f}^{(i)}$ (the "individual" conditional expectations), rather than taking the average. Using the same partials as above, contained in `the.pdp.ext`, we draw the individual lines by instructing `ggplot` to group the lines by their identifier in the `the.pdp.ext` dataset: `yhat.id`. Notice the additional `group` aesthetic passed through `geom_line`:

```
ext.ice <- ggplot() +
 geom_line(data = the.pdp.ext, aes(x = extremity100_scaled, y = yhat,
  group = yhat.id), alpha = 0.2) +
 stat_summary(data = the.pdp.ext, aes(x = extremity100_scaled, y = yhat),
  fun = mean, geom = "line", col = "black", size = 2) +
 theme_minimal() +
 xlab("Extremity") +
 ylab("Predicted Value") +
 geom_rug(data = ho.rf.data.nona.nointaxn, aes(x = extremity100_scaled))
```

Something to be attentive to is the number of conditional expectations to draw, since many datasets will have too many observations to practically fit on one plot, requiring that we only draw a subset. To draw a smaller set of lines, the easiest way is to create a smaller version of the partial predictions. As with any random action, best practice is to set a seed for replication.

```
set.seed(77281)
lines.to.draw <- 500

the.ext.yhats <- sample(1:max(the.pdp.ext$yhat.id),
 lines.to.draw, replace = FALSE)
the.pdp.ext.small <- the.pdp.ext[the.pdp.ext$yhat.id %in% the.ext.yhats,]
```

There are two extensions of ICE plots. One is a “centered” ICE plot, or c-ICE (Goldstein et al. 2015), whereby some observed location x^* along x_s is chosen, and all lines are forced to run through that point. The c-ICE plots for plotting along a single predictor variable are given as:

$$\hat{f}^{(i)}_{cICE} = \hat{f}^{(i)}(x_s) - \hat{f}(x^*, \mathbf{x}_{ic}) \tag{1}$$

where $\hat{f}^{(i)}_s$ is the original ICE curve and $\hat{f}(x^*, \mathbf{x}_{ic})$ is the prediction for location x^* for observation i. This has the effect of ‘anchoring’ (or centering) all trends to run through a single common point (Molnar 2020). Goldstein et al. (2015) suggest using either the minimum or maximum observed value of x_s as this anchor. c-ICE plots are valuable since they can help uncover heterogeneity and clustering of certain observations, which can be hard to see if the observations have different average levels across x_s. In other words, by centering all curves, it can be easier to see differences in slopes across observations that might otherwise be obscured by differences in levels.

Derivative ICE (d-ICE) plots are another variant of ICE plots (Goldstein et al. 2015). d-ICE plots show the partial derivative of each ICE curve with respect to the predictor variable of interest, and can be used to probe for ‘hidden’ interactions between this predictor and the other control variables. If there are no interactive effects, the d-ICE plot will look like a single line which shows the partial derivative (i.e., the effect is constant across all observations). However, if there are interactive effects, they will appear as heterogeneous partial derivative lines in the plot. Derivatives cannot be analytically derived but are instead numerically approximated (Goldstein et al. 2015), so creating d-ICE plots often takes a long time. Both c-ICE and d-ICE plots can be created using the `ICEbox` package (Goldstein, Kapelner and Bleich 2017).

References

Apley, Daniel W and Jingyu Zhu. 2016. "Visualizing the effects of predictor variables in black box supervised learning models." *arXiv preprint arXiv:1612.08468* .

Doshi-Velez, Finale and Been Kim. 2017. "Towards a rigorous science of interpretable machine learning." *arXiv preprint arXiv:1702.08608* .

Evans, Jeffrey S. and Melanie A. Murphy. 2019. *rfUtilities: Random Forests Model Selection and Performance Evaluation*. R package version 2.1-5.
URL: *https://CRAN.R-project.org/package=rfUtilities*

Friedman, Jerome H. 2001. "Greedy function approximation: a gradient boosting machine." *Annals of statistics* pp. 1189–1232.

Funk, Kendall D, Hannah L Paul and Andrew Q Philips. 2021. "Point break: using machine learning to uncover a critical mass in women's representation." *Political Science Research and Methods* pp. 1–19.

Goldstein, Alex, Adam Kapelner and Justin Bleich. 2017. "Package 'ICEbox'.".

Goldstein, Alex, Adam Kapelner, Justin Bleich and Emil Pitkin. 2015. "Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation." *Journal of Computational and Graphical Statistics* 24(1):44–65.

Greenwell, Brandon M. 2017. "pdp: an R Package for constructing partial dependence plots." *The R Journal* 9(1):421–436.

Hastie, Trevor, Robert Tibshirani and Jerome Friedman. 2013. *The elements of statistical learning: Data mining, inference, and prediction*. Second ed. Springer Science & Business Media.

Howard, Nicholas O. and Mark E. Owens. 2020. "Circumventing Legislative Committees: The US Senate." *Legislative Studies Quarterly* 45:495–526.

Kim, Been, Rajiv Khanna and Oluwasanmi O Koyejo. 2016. "Examples are not enough, learn to criticize! criticism for interpretability." *Advances in neural information processing systems* 29.

Liaw, Andy and Matthew Wiener. 2018. "Breiman and Cutler's Random Forests for Classification and Regression." *R Documentation for package 'randomForest'* pp. 1–29.

Molnar, Christoph. 2020. *Interpretable Machine Learning*. Lulu. com.

Montgomery, Jacob M and Santiago Olivella. 2018. "Tree-Based Models for Political Science Data." *American Journal of Political Science* 62(3):729–744.

Muchlinski, David, David Siroky, Jingrui He and Matthew Kocher. 2016. "Comparing random forest with logistic regression for predicting class-imbalanced civil war onset data." *Political Analysis* 24(1):87–103.

Poyet, Corentin and Tapio Raunio. 2020. “Reconsidering the Electoral Connection of Speeches: The Impact of Electoral Vulnerability on Legislative Speechmaking in a Preferential Voting System.” *Legislative Studies Quarterly* .

Ribeiro, Marco Tulio, Sameer Singh and Carlos Guestrin. 2016. ” Why should i trust you?” Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. pp. 1135–1144.

Zhao, Qingyuan and Trevor Hastie. 2019. “Causal interpretations of black-box models.” *Journal of Business & Economic Statistics* pp. 1–10.