Modeling the Exposure and the Outcome: CART and Random Forest

Ashley I Naimi

June 2022

Contents

1 CART Performance

When they were introduced, it was quickly realized that classification and regression trees were a powerful tool for exploring data and making predictions. Relative to other prediction algorithms (e.g., principal component analysis, partial least squares), trees are not affected by transformations (e.g., centering and scaling) of the predictors. They are immune to the effects of predictor outliers. Unlike logistic regression, they do not require complex link functions for binary outcomes. Unlike neural networks, the relation between the predictors and the outcome is relatively easy to understand. Finally, trees are easy to use because they can be directly applied to data without requiring complex transformations to optimize predictive validity (Hastie et al., 2009, (p352)).

However, it was also soon realized that trees suffered from an important problem: relatively low accuracy.[1] The problem is that trees tend to grow too deep and learn highly irregular patterns: they overfit (i.e., undersmooth) the data. In other words, CART tends to have low bias, but very high variance.

[1] For classification trees, accuracy is captured via a confusion matrix. For regression, accuracy is defined as mean squared error.

This problem results from the fact that trees tend to encode highly irregular features of the data under a wide range of tuning parameters. We can visualize this effect by fitting a CART model to the NHEFS data under the default tuning parameters and a moderate decrease in the complexity parameter.

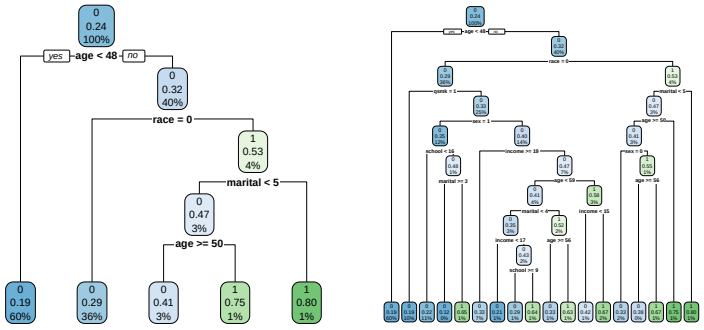

Figure 1: Classification and Regression Tree Algorithms fit to the NHEFS Data with a complexity parameter of 0.01 (left panel) and 0.005 (right panel).

As displayed in Figure 1, as soon as we lower the complexity parameter from its default by even a little amount, the number of "important" features captured by the CART tree jumps from 3 to 14. Indeed, it is very easy to create highly complex trees that fit the data at hand nearly perfectly. But in doing so, we lose the ability to make scientific generalizations from our models, which is primarily why we would do the analysis.

Soon after CART was introduced, several researchers began to make modifications to the algorithm to make it more robust to overfitting. The two that were shown to have the most important impacts on the performance of a tree were bootstrap aggregating (bagging) and the random subspace method.

2 Bootstrap Aggregation

Bootstrap aggregation, or bagging, is a simple technique meant to reduce the variability of a flexibly specified classification and regression tree. The basic premise starts with a bootstrap sample of the original data. For example, suppose we let X represent all (outcome and covariates) of the NHEFS data that we are using to fit a CART model. Suppose further that we let Tree(X) represent the tree fit to these data (i.e., as in Figure 1, left panel). We can take B bootstrap samples from X, which we denote $X_1, X_2, \ldots X_B$, which gives us a total of B trees. This multiplicity of trees is why we call the approach random forests.

Going back to the NHEFS data example, we can take nine bootstrap resamples and fit a flexible CART model to each resample, giving us a total of nine trees:

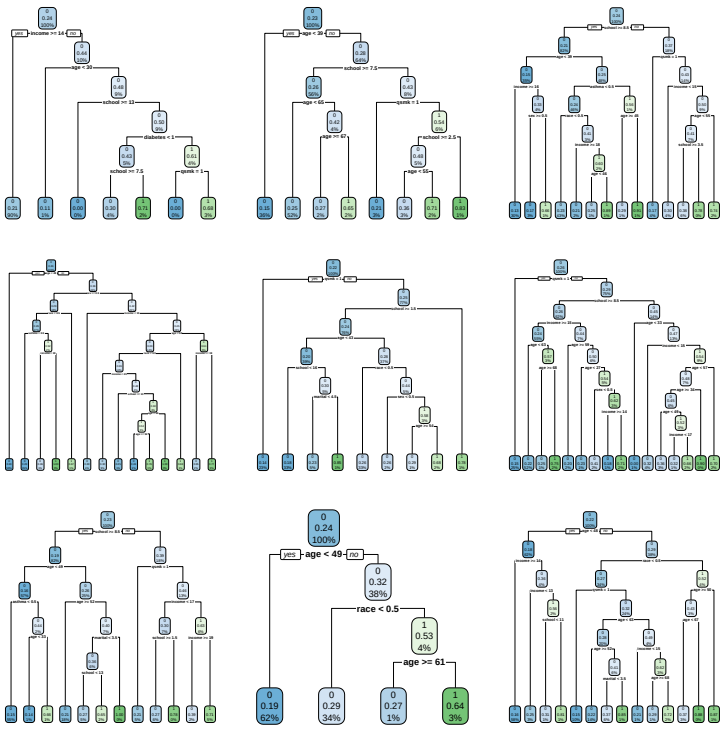

Figure 2: Nine Classification and Regression Tree Algorithms fit to bootstrap resamples of the NHEFS Data under default tuning parameters.

Note the variation in the structure of these trees, which is an indication of how sensitive CART is to (here relatively minor) perturbations in the data. This variation is not of direct interest, in the sense that we do not use the bootstrap to quantify confidence intervals.[2] Rather, random forests rely on the bootstrap to obtain numerous trees, which are then averaged over to get a single prediction. By taking the average of several trees fit to the same data, bagging is better able to deal with the volatility inherent in CART to yield a better performing algorithm.

[2] Recall, the difficulties with using the bootstrap for machine learning algorithms result from the curse of dimensionality mean that we can't actually construct honest confidence intervals under general conditions for CART-based learners.

Mathematically, we can write that a random forest is simply an aggregation of bootstrapped trees:

$$\text{RF} = \frac{1}{B}\sum_{b=1}^{B} \text{Tree}(X_b)$$

It is the averaging of numerous trees that results in a better performance of the random forest over CART. To give a more precise illustration of the mechanism here, consider the trees in Figure 2. Consider further that we might be interested in predicting the probability of high blood pressure if everyone quit smoking. From each tree, we would obtain the predicted probability of high blood pressure if `qsmk=1`, giving us a total of nine predictions, from which we

would compute the average.[3] For a classification algorithm, the final prediction (i.e., whether a person has high BP or not) will be based on the "majority vote" of all.

[3] Note that for some trees, the prediction doesn't change since `qsmk` is not in the tree.

Another benefit of using the bootstrap in this context is that we have a way of estimating the error in the algorithm. This error is calculated as follows:

Step 1: Compare the observations in each of the B bootstrapped trees to the observations in the original sample. From all B trees, select only those that do not have the first observation in the original sample. These trees are referred to as "out of bag" trees.

Step 2: Predict the event using only the out of bag trees.

Step 3: Take the difference of the out-of-bag prediction and the actual outcome for the first observation.

Step 4: Repeat for all N.

Step 5: Take the mean of all N errors.

The end result from this process is referred to as the out-of-bag error. But why is it important? First, the OOB error gives an idea of how "close" the forest predictions are to the actual predictions. Second, Brieman (1983) showed that the out-of-bag error is as accurate as doing a formal analysis with a testing and validation set (e.g., cross-validation, which we will discuss later). As a result, one need not actually use formal testing/validation with random forests.

3 Random Subspace Selection

A second feature of random forests is the use of random subspace selection, which works as follows: Instead of using **all available covariates** when fitting each tree to the bootstrapped data, the algorithm randomly chooses only a subset of the predictors. What this does is it reduces the correlation between each of the bootstrapped trees, thus allowing us to compute error rates without accounting for violations of the IID assumption.[4] It also tends to reduce the impact of a variable that is so highly predictive that it would force the tree to overfit the data whenever it is included in the tree. Thus, by repeated the fitting process a large number of times the overfitting induced by these highly predictive variables is minimized.

[4] Recall, the independent and identically distributed (IID) assumption is required for the validity of most estimation methods, including the mean estimator, which is used to compute the OOB error.

4 Random Forests via `ranger`

In R, there are several implementations of the random forest algorithm for a wide range of data. These include `RandomForest` and `ranger`, which implement more "classical" random forests (i.e., for regression and classification), as well as `quantregForest` and `randomForestSRC`, which implement versions that predict quantiles of a distribution (e.g, survival curves), and not just the expected value.

Arguably, (IMO) the `ranger` package is the best of these given its speed and simplicity. To install `ranger`, we use the standard approach. In the `ranger` function, there are several options that we should be aware of. These options are effectively tuning parameters that will change the behavior of the algorithm. Among them include:

Table 1: Select tuning parameters for the 'ranger' function in R. Source: R help.

Parameter	Default	Interpretation
num.trees	500	Number of bootstraps from the data; the size of the forest
mtry	round($\sqrt{p}$)	Number of variables randomly selected for subspace (where p is total number)
importance	none	Whether a variable importance measure will be computed.
splitrule	gini/variance	Impurity measure used to determine whether a split should occur
min.node.size	1	Smallest number of observations in each node.
probability	F	For binary outcomes, determines whether probability or classificaiton should be computed.
quantreg	F	For continuous outcomes, determines whether quantile forest should be fit.

To implement `ranger`, we can use the following code:

```
library(tidyverse)
library(ranger)
```

```
nhefs <- read_csv(here("data", "nhefs.csv")) %>%
    mutate(wt_delta = as.numeric(wt82_71 >
        median(wt82_71)), age = scale(age),
        sbp = scale(sbp), dbp = scale(dbp),
        price71 = scale(price71), tax71 = scale(tax71)) %>%
    select(-wt82_71)
```

```
## Rows: 1394 Columns: 11
## -- Column specification ------------------------
## Delimiter: ","
## dbl (11): seqn, qsmk, sex, age, income, sbp...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
head(nhefs)
```

```
## # A tibble: 6 x 11
##    seqn  qsmk   sex age[,1] income sbp[,1]
##   <dbl> <dbl> <dbl>   <dbl>  <dbl>   <dbl>
## 1   233     0     0 -0.112      19   2.50
## 2   235     0     0 -0.614      18  -0.284
## 3   244     0     1  1.06       15  -0.712
## 4   245     0     0  2.06       15   1.06
## 5   252     0     0 -0.279      18  -0.551
## 6   257     0     1 -0.0286     11   0.680
## # i 5 more variables: dbp <dbl[,1]>,
## #   price71 <dbl[,1]>, tax71 <dbl[,1]>,
## #   race <dbl>, wt_delta <dbl>
```

```
# implement ranger set random seed
set.seed(123)
rf_example <- ranger(factor(wt_delta) ~ qsmk +
    sex + age + income + sbp + dbp + price71 +
```

```
    tax71 + race, num.trees = 500, mtry = 3,
    min.node.size = 50, data = nhefs)

rf_example
```

```
## Ranger result
##
## Call:
##  ranger(factor(wt_delta) ~ qsmk + sex + age + income + sbp + dbp +      price71 + tax71 + race, num.
##
## Type:                             Classification
## Number of trees:                  500
## Sample size:                      1394
## Number of independent variables:  9
## Mtry:                             3
## Target node size:                 50
## Variable importance mode:         none
## Splitrule:                        gini
## OOB prediction error:             43.19 %
```

In subsequent sections, we will see how we can incorporate the ranger function into a meta-learner (super learner), and search over a grid of parameters for the optimal model. We will also see how this meta-learner can then be included into a double robust estimator to quantify causal effect estimates.

References

L Brieman. Out-of-bag estimation. Technical report, University of California, Berkeley, ftp://ftp.stat.berkeley.edu/pub/users/breiman/ OOBestimation, 1983.

Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, New York, NY, 2009.