The random forest method is based on the regression tree method; in essence, the forest is simply a collection of trees.

So what is the regression tree method? It’s a mode of statistical modeling that predicts based on mean values in each bin, rather than assigning coefficients to certain variables. This allows predictions to be more flexible and reduces the harm of covariance between different variables. The tree is generated according to the following process. Imagine a dataset of NBA players. The dependent variable will be points per game in the 2019-2020 season. The independent variables will be years in the league and draft position. Imagine that the dataset resembles the following:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | PPG | Years | Draft |
| LeBron James | 25.3 | 17 | 1 |
| Kemba Walker | 20.4 | 9 | 9 |
| LaMarcus Aldridge | 18.9 | 14 | 2 |
| Chris Paul | 17.6 | 15 | 4 |
| Marc Gasol | 7.5 | 12 | 48 |
| Kentavious Caldwell-Pope | 9.3 | 7 | 8 |

The first step would be to consider the entire dataset, and split it into two groups based on years and/or draft position. Here, let’s say we partition by years of experience. The split criteria is made according to the following hypothetical: if the mean in each bin were the fitted value for each observation in the bin, which split would minimize the SSE of the dataset? The bins need not be the same size; the task is simply to minimize, by choice of splitting criteria in any of the independent variable, the total SSE. Hypothetically, let us say that splitting according to years > 12 minimizes the SSE. Now we have two bins, one containing LeBron, LaMarcus, and Chris Paul, and the other with Kemba, KCP, and Marc Gasol. Now, we examine one of the bins, and we split it again. We could split the second bin by draft position; for players with 12 or less years in the league, we divide them by draft position < 10. Now LeBron, LaMarcus, and Chris Paul are in Bin 1; Kemba and KCP are in Bin 2; and Marc Gasol is in Bin 3.

For each of our existing bins, we could go on binning and binning until we had as many bins as we had observations. Of course, this would not be particularly helpful for making predictions on our data; we would want our bins to have multiple observations. But at the same time, we want there to be enough bins that we capture the variation in our data. The best regression tree will strike the balance between having too many bins (and being severely overfitted) and having too few bins (and being inflexible).

The question, of course, becomes: how do we strike this balance? One way to do so is to create a very large, granularly binned tree, and then prune it back by assigning a complexity cost. When we’ve created a complex tree, it will have a certain number of “terminal nodes”; in other words, a certain number of bins. We can assign a penalty for having terminal nodes by reframing our splitting criterion. Instead of trying to minimize SSE, we could try to minimize SSE + penalty\*terminal nodes. This is a similar concept to the one which underlies the LASSO technique. Of course, now the question becomes, what size penalty yields the best regression tree? To answer this, we could create optimal regression trees for a given spectrum of penalties, so that each penalty value corresponds to a regression tree. Then we could evaluate those trees to find the best one and therefore, the optimal penalty value.

Another method is to optimize the minimum bin size in addition to the penalty cost. Smaller minimum bin sizes reduce the impact of outliers. Or, you could optimize the maximum depth (i.e., the number of splits between the root node and any terminal node).

Regression trees are great. They are easy to interpret, they highlight the most important variables by their splitting choice, and they allow for predictions even in cases of missing data. And, although it may take some time to generate the tree itself, making predictions from an existing tree is extraordinarily simple and fast. But, the trees can vary wildly depending on the sample; in other words, they tend to be over-reliant on the sample in ways that a standard OLS regression is not. This implies another problem, of course: these trees do not always make good predictions.

To reduce this variance (and therefore, to make better predictions), you could bootstrap multiple trees. (This is referred to as “bagging.”) In other words, rather than generating one tree from your training set, you could bootstrap 50 or so samples from your training data, and generate a tree for each sample. Then, you could average your results. The resulting tree would have much less variance than a single tree generated only from the training set.

Notes:

* Running the random forest with sampling and a feature\_frac of 0.7 produced more or less a flat-line fit at about 0.003.
* Running the random forest with sampling and a feature\_frac of 1 produced a similar result.
* Running the random forest with no sampling and a feature\_frac of 0.7 produced a reasonable fit.
* Allowing the each tree to predict on the complete dataset, not just the sample, produced a very good fit.

**References**

<http://uc-r.github.io/regression_trees>