Untitled

Jeff Chen

10/26/2018

Random Forests

How do we know anything for sure? Virtually every aspect of life has some uncertainty tied in. When a hurricane approaches the US Eastern Seaboard, forecasters often map the *cone of uncertainty* that provides the possible range of motion of a storm based on the results of many forecasted simulations. In presidential elections, often times the most polling results are ones that ensemble or average the results of many other similarly conducted polls. The reliance on predictions from a group of models with the same aims may well improve prediction accuracy. In statistical learning, average the results of multiple models is known as *ensemble learning* or *ensembling* for short.

Single models may imposes biases on data and may be well-suited in specific situations. Ensemble methods combine the results of many models to obtain more stable results. For example, the curve in graph #1 can be approximated using a decision tree algorithm. The result of a single tree only loosely fits the curve in a jagged fashion (#2). That one tree may impose biases on the data, perhaps through how the tree is pruned or the assumption that the jagged approximation is appropriate, which may then translate into greater variance in predictions. One could imagine that the structure of that one tree may have happened by chance, and under different situations, the fit could be better.

Bootstrapping can help. Recall from elementary statistics that bootstrapping is defined as any statistical process that involves sampling records with replacement. By bootstrapping a sample, we treat a sample like a population, we can expose and characterize the qualities of an estimator under various scenarios already available in the data, which in turn produces an empirical probability distribution for predictions using the estimator. We can bootstrap the decision tree by (1) sampling the data with replacement up to the full size of the sample, then (2) run the decision tree. The result of repeating the process 50 times is (graph #3) produces a result that appears to be more organic and more accurate. This process of *bootstrapping* and *aggregating* the results is referred to as *bagging*.

Applying bagging to decision trees may not necessarily be enough to develop a well-balance prediction. In the social sciences and public policy, it is generally assumed that a model's specification is a choice left to the analyst; However, it may also be a source of methodological bias.

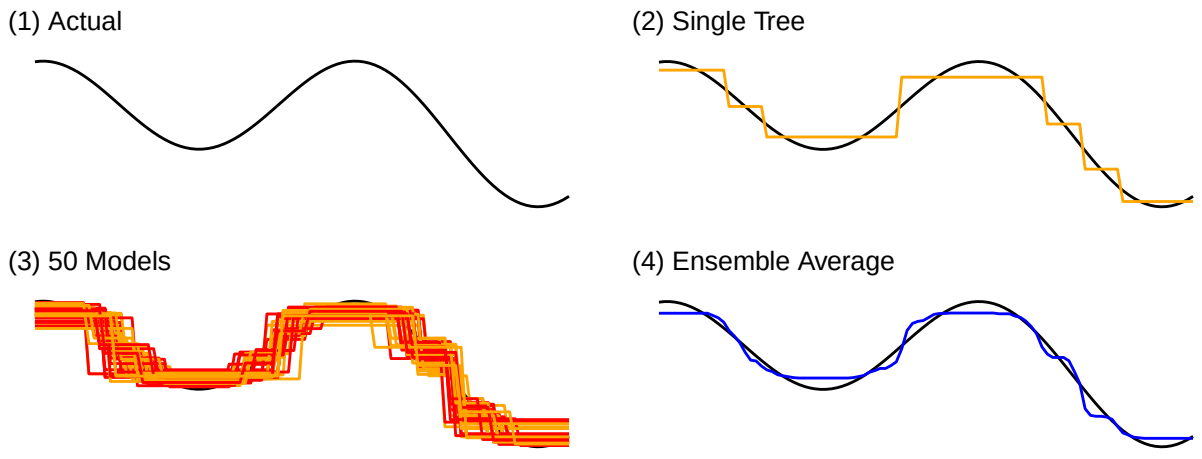


Figure 1: Comparison of results of applying a single model to fit a curve versus an ensemble of models.

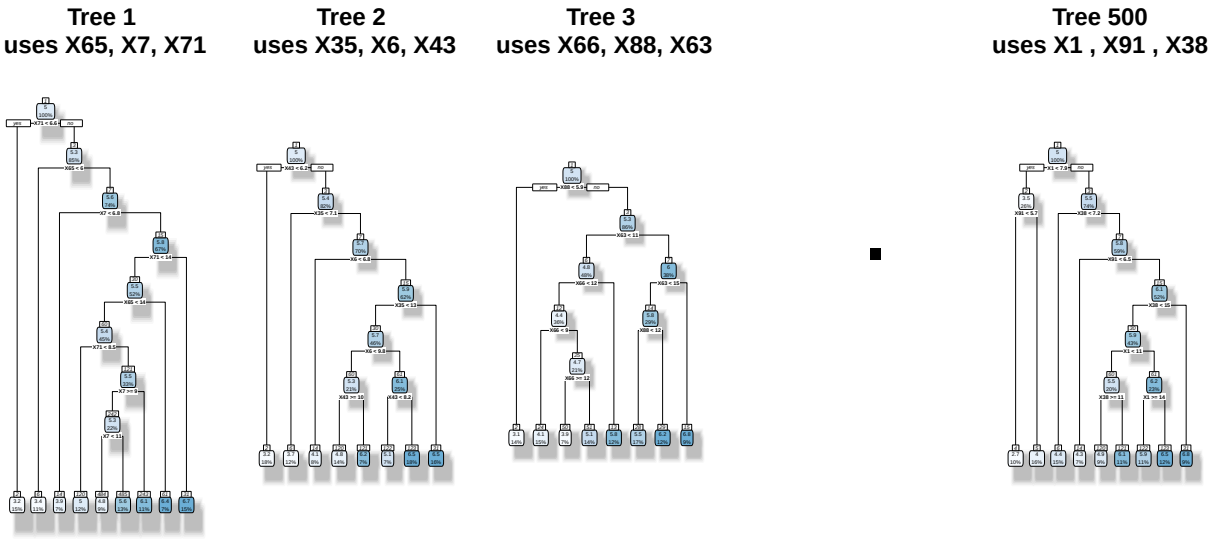


Figure 2: Random Forests construct hundreds of trees sampling from both observations and features, then combine the trees into one prediction through voting.

Random forests can help. The technique is an extension of decision trees using a modified form of bootstrapping and ensemble methods to mitigate overfitting and bias issues.[1] Not only are individual records bootstrapped, but input features are bootstrapped such that if K variables are in the training set, then k variables are randomly selected to be considered in a model such that $k < K$. Each bootstrap sample is exhaustively grown using decision tree learning and is left as an unpruned tree. The resulting predictions of hundreds of trees are ensembled. The logic is described below.

Pseudo-code

```
Let S = training sample, K = number of input features
  1. Randomly sample S cases with replacement from the original data.
  2. Given K features, select k features at random where k < K.
  3. With a sample of s and k features, grow the tree to its fullest complexity.
  4. Predict the outcome for all records.
  5. Out-Of-Bag (OOB). Set aside the predictions for records not in the s cases.
Repeat steps 1 through 5 for a large number of times saving the result after each tree.
Vote and average the results of the tree to obtain predictions.
Calculate OOB error using the stored OOB predictions.
```

The *Out-Of-Bag* (OOB) sample is a natural artifact of bootstrapping: approximately one-third of observations are naturally left un-selected, which can be used as the basis of calculating each tree's error and the overall model error. Think of it as a convenient built in test sample.

How about interpretation? Unlike decision trees, it is not a simple task to deduce rules or criteria that describe the target variable. Instead, random forests use *variable importance*, which, like for a decision tree, measures the contribution of a feature to the homogeneity of a classifier. Unlike decision trees, variable importance for a Random Forest is calculated as the mean decrease in the Gini coefficient of a split relative to the Gini coefficient of the root node. Gini coefficients measures homogeneity on a scale of 0 to 1, where 0 is perfect homogeneity and 1 is perfect heterogeneity. The Gini changes are summed for each variable and normalized.

[1] Breiman (2001)

Tuning

Whereas methods like regression have a closed form solution, Random Forest require tuning as optimal models need to be searched for under different conditions. The principal tuning parameters include: Number of features and number of trees.

- *Number of input features.* As k number of parameters need to be selected in each sampling round, the value of k needs to minimize the error on the OOB predictions.
- *Number of trees* influences the stability the Variable Importance metric that is commonly used to infer variable influence in decision tree learning. More trees help to stabilize the Variable Importance estimate. To determine the number of trees, keep adding trees to a sample until the OOB error for a randomly select set of trees is approximately equal to that of the ensemble.

DIY: Revisiting activity classification

Like decision trees, much of Random Forests rely on easy to use methods made available through the `randomForest` library. There are a couple of ways to run the algorithm, including:

`randomForest(formula, data, method, mtry, ntree)`

where: - `formula` is an object containing the specification to be estimated. Note that - `data` is a data frame. - `mtry` is the number of variables to be randomly sampled per iteration. Default is $\sqrt{k}$ for classification trees. - `ntree` is the number of trees. Default is 500.

Using the same formula as the `rpart()` function, we can train a naive Random Forest and check the OOB error. Approximately 75.6% of observations in the OOB sample were correctly classified using 2 randomly selected variables in each of the 500 trees.

```
#Load randomForest library
  library(randomForest)

#Run Random Forest
  spec <- as.formula("factor(activity) ~ accel + avg50 + max50 + min50 + sd50")
  fit.rf <- randomForest(spec, data = train[runif(nrow(train))< 0.1,], mtry = 2, ntree = 500)

#Check OOB error
  fit.rf
```

```
##
## Call:
##  randomForest(formula = spec, data = train[runif(nrow(train)) <      0.1, ], mtry = 2, ntree = 500)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 2.58%
## Confusion matrix:
##         idle  run stairs walk class.error
## idle   19120    0      1  139 0.007268951
## run        0 2116      1    5 0.002827521
## stairs    16    0    394  335 0.471140940
## walk     273    3     26 8518 0.034240363
```

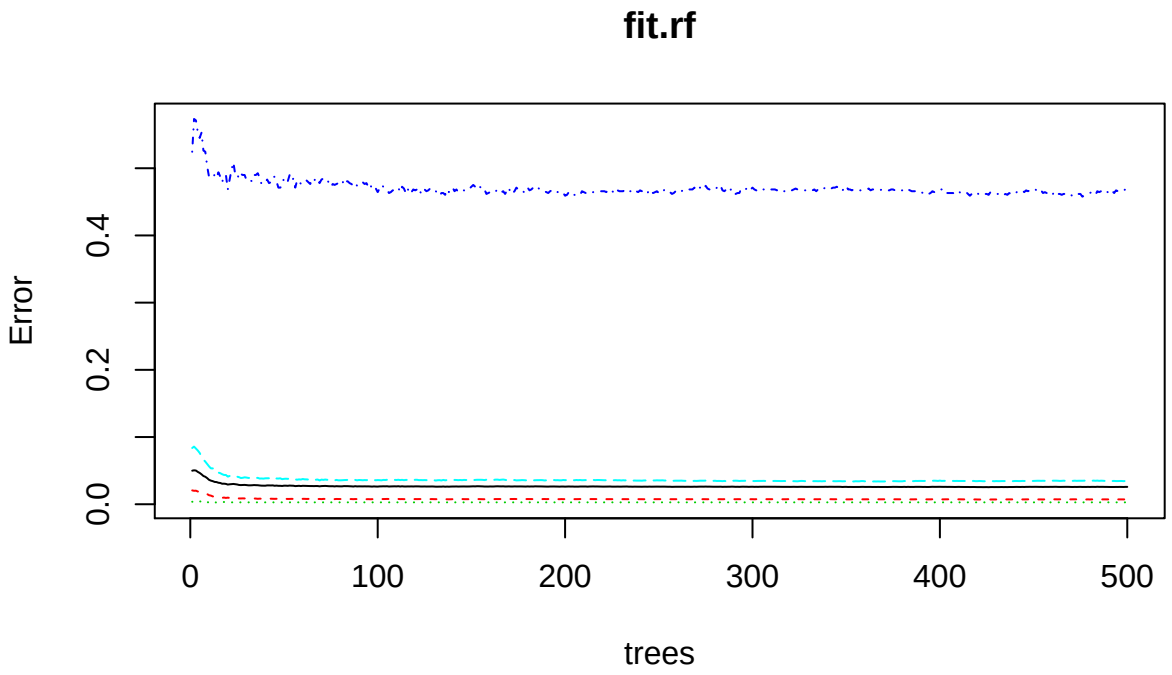
Using the `importance()` function, we can see the Mean Decrease Gini, which calculates the mean of Gini coefficients. However, the values themselves do not have any meaning outside of a comparison with other Gini measures.

```
importance(fit.rf)
```

```
##       MeanDecreaseGini
## accel         520.6098
## avg50        5913.1996
## max50        3847.8694
## min50        1864.1431
## sd50         4137.7752
```

By default, the `randomForests` library sets the number of trees to equal 500. By plotting the fit object, we can see how OOB error and the confidence interval converges asymptotically as more trees are added to the ensemble. Otherwise stated, more trees will help up to a certain point and the default is likely more than enough.

```
  plot(fit.rf)
```



As we know that $n = 500$ trees is more than enough, we will now need to tune the tree for the number of variables. To tune the algorithm, we will use the `tuneRF()` method. The method searches for the optimal number of variables per split by incrementally adding variables. While it's a useful function, it is relatively verbose. In addition to the target and input features, a number of other parameters need to be specified:

`tuneRF(x, y, ntreeTry,  mtryStart, stepFactor, improve, trace, plot)`

where: - `x` is a data frame or matrix of input features. - `ntreeTry` is the number of trees used in each iteration of tuning. - `mtryStart` is the number of variables to start. - `stepFactor` is the number of additional variables tested per iteration. - `improve` is the minimum relative improvement in OOB error for the search to go on. - `trace` is a boolean that indicates where to print the search progress. - `plot` is a boolean that indicates whether to plot the search results.

Below, we conduct a search from `mtryStart = 1` with a `stepFactor = 2`. The search result indicates that 2 variables per tree are optimal.

```
#Search for optimal number of input features
  fit.tune <- tuneRF(x = train[,c("avg50", "max50", "min50", "sd50")],
                     y =  factor(train$activity),
                     ntreeTry = 500,
```

```
                      mtryStart = 1,
                      stepFactor = 2,
                      improve = 0.001,
                      plot = TRUE)

#Extract best parameter
  tune.param <- fit.tune[fit.tune[, 2] == min(fit.tune[, 2]), 1]
```

Normally, we can plug the tuned parameter back into the `randomForest()` method and re-train the algorithm, but it unnecessary in this case as the default model already uses the same parameters. When applied to the test set, we see that the mean F1-statistic is much improved – or a whole 10-percentage point increase.

```
#Predict classes in test
  yhat <- predict(fit.rf, test)

#Calculate mean F1
  meanF1(test$activity, yhat)
```

This result does not mean that Random Forests will always turn better results, but rather multiple techniques should be tested when tackling prediction problems. Also, remember the policy goal: is the objective to predict or to explain? If a little of both, then it is worth understanding the value of increased accuracy at the cost of interpretability.