Random Forests

Jan-Philipp Kolb

20 Mai, 2019

Random Forests

- Bagging (bootstrap aggregating) regression trees is a technique that can turn a single tree model with high variance and poor predictive power into a fairly accurate prediction function.
- Unfortunately, bagging regression trees typically suffers from tree correlation, which reduces the overall performance of the model.
- Random forests are a modification of bagging that builds a large collection of de-correlated trees and have become a very popular "out-of-the-box" learning algorithm that enjoys good predictive performance.

Preparation - random forests

- The following slides are based on UC Business Analytics R Programming Guide on random forests

```
library(rsample)      # data splitting
library(randomForest) # basic implementation
library(ranger)       # a faster implementation of randomForest
library(caret)        # an aggregator package for performing man
library(h2o)          # an extremely fast java-based platform
```

The Ames housing data

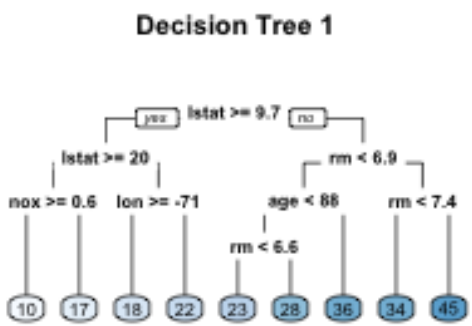
```
set.seed(123)
ames_split <- initial_split(AmesHousing::make_ames(), prop = .7)
ames_train <- training(ames_split)
ames_test  <- testing(ames_split)
```

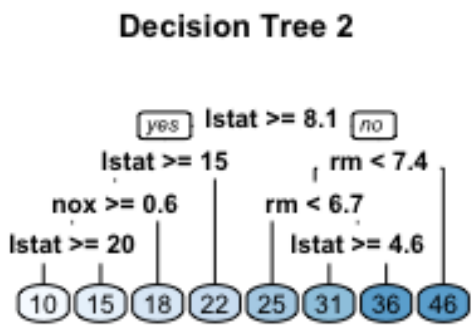
The idea of random forests

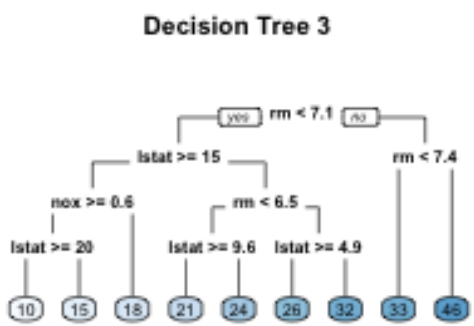
- Random forests are built on the same fundamental principles as decision trees and bagging.
- Bagging trees introduces a random component in to the tree building process that reduces the variance of a single tree's prediction and improves predictive performance.
- The trees in bagging are not completely independent of each other since all the original predictors are considered at every split of every tree.
- Trees from different bootstrap samples typically have similar structure to each other (especially at the top of the tree) due to underlying relationships.

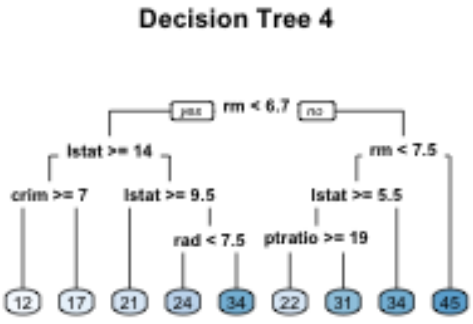
Similar trees

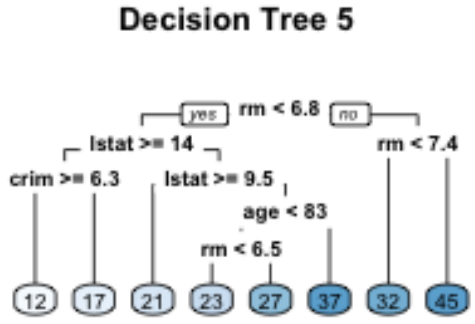
- E.g., if we create six decision trees with different bootstrapped samples of the Boston housing data, we see that the top of the trees all have a very similar structure.
- Although there are 15 predictor variables to split on, all six trees have both `lstat` and `rm` variables driving the first few splits.

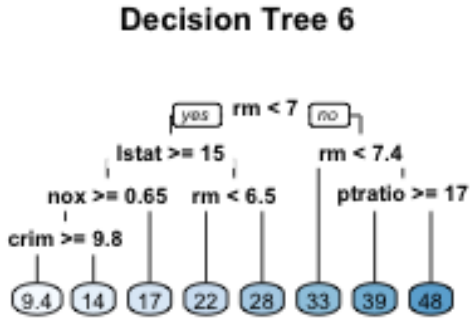












Tree correlation

- This characteristic is known as tree correlation and prevents bagging from optimally reducing variance of the predictive values.
- In order to reduce variance further, we need to minimize the amount of correlation between the trees.
- This can be achieved by injecting more randomness into the tree-growing process. Random forests achieve this in two ways:

1. Bootstrap:

- Similar to bagging, each tree is grown to a bootstrap resampled data set, which makes them different and somewhat decorrelates them.

2. Split-variable randomization:

- Each time a split is to be performed, the search for the split variable is limited to a random subset of m of the p variables.
- For regression trees, typical default values are $m = \frac{p}{3}$ but this should be considered a tuning parameter.
- When $m = p$, the randomization amounts to using only step 1 and is

Basic algorithm

The basic algorithm for a regression random forest can be generalized to the following:

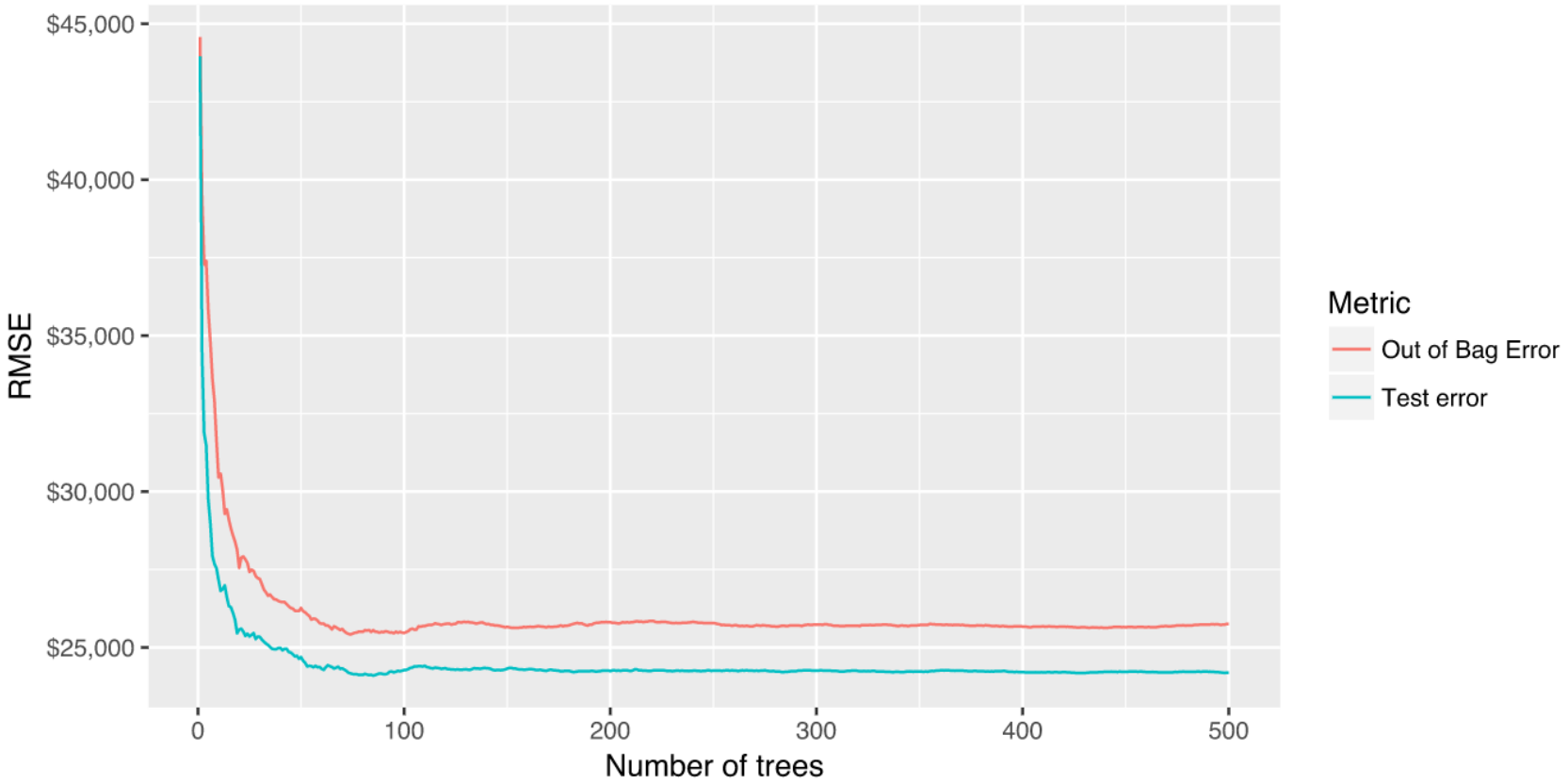
```
1.  Given training data set
2.  Select number of trees to build (ntrees)
3.  for i = 1 to ntrees do
4.  |  Generate a bootstrap sample of the original data
5.  |  Grow a regression tree to the bootstrapped data
6.  |  for each split do
7.  |  | Select m variables at random from all p variables
8.  |  | Pick the best variable/split-point among the m
9.  |  | Split the node into two child nodes
10. |  end
11. | Use typical tree model stopping criteria to determine when
12. end
```

- Since the algorithm randomly selects a bootstrap sample to train on and predictors to use at each split, tree correlation will be lessened

OOB error vs. test set error

- Similar to bagging, a natural benefit of the bootstrap resampling process is that random forests have an out-of-bag (OOB) sample that provides an efficient and reasonable approximation of the test error.
- This provides a built-in validation set without any extra work on your part, and you do not need to sacrifice any of your training data to use for validation.
- This makes identifying the number of trees required to stablize the error rate during tuning more efficient;
- As illustrated below some difference between the OOB error and test error are expected.

Random forest out-of-bag error versus validation error



Scoring models - metrics

- Many packages do not keep track of which observations were part of the OOB sample for a given tree and which were not.
- If you are comparing multiple models to one-another, you'd want to score each on the same validation set to compare performance.
- It is possible to compute certain metrics such as root mean squared logarithmic error (RMSLE) on the OOB sample, but it is not built in to all packages.
- So if you are looking to compare multiple models or use a slightly less traditional loss function you will likely want to still perform cross validation.

Advantages & Disadvantages

Advantages - random forrests

- Typically have very good performance
- Remarkably good "out-of-the box" - very little tuning required
- Built-in validation set - don't need to sacrifice data for extra validation
- No pre-processing required
- Robust to outliers

Disadvantages - random forrests

- Can become slow on large data sets
- Although accurate, often cannot compete with advanced boosting algorithms
- Less interpretable

Basic implementation

- There are over 20 random forest packages in R.
- To demonstrate the basic implementation we illustrate the use of the `randomForest` package, the oldest and most well known implementation of the Random Forest algorithm in R.
- As your data set grows in size `randomForest` does not scale well (although you can parallelize with `foreach`).
- To explore and compare a variety of tuning parameters we can also find more effective packages.
- The packages `ranger` and `h2o` packages will be presented in the tuning section.

randomForest::randomForest

- `randomForest` can use the formula or separate x, y matrix notation for specifying our model.
- Below we apply the default `randomForest` model using the formulaic specification.
- The default random forest performs 500 trees and $\frac{\text{features}}{3} = 26$ randomly selected predictor variables at each split.
- Averaging across all 500 trees provides an OOB MSE=659550782 (RMSE=25682).

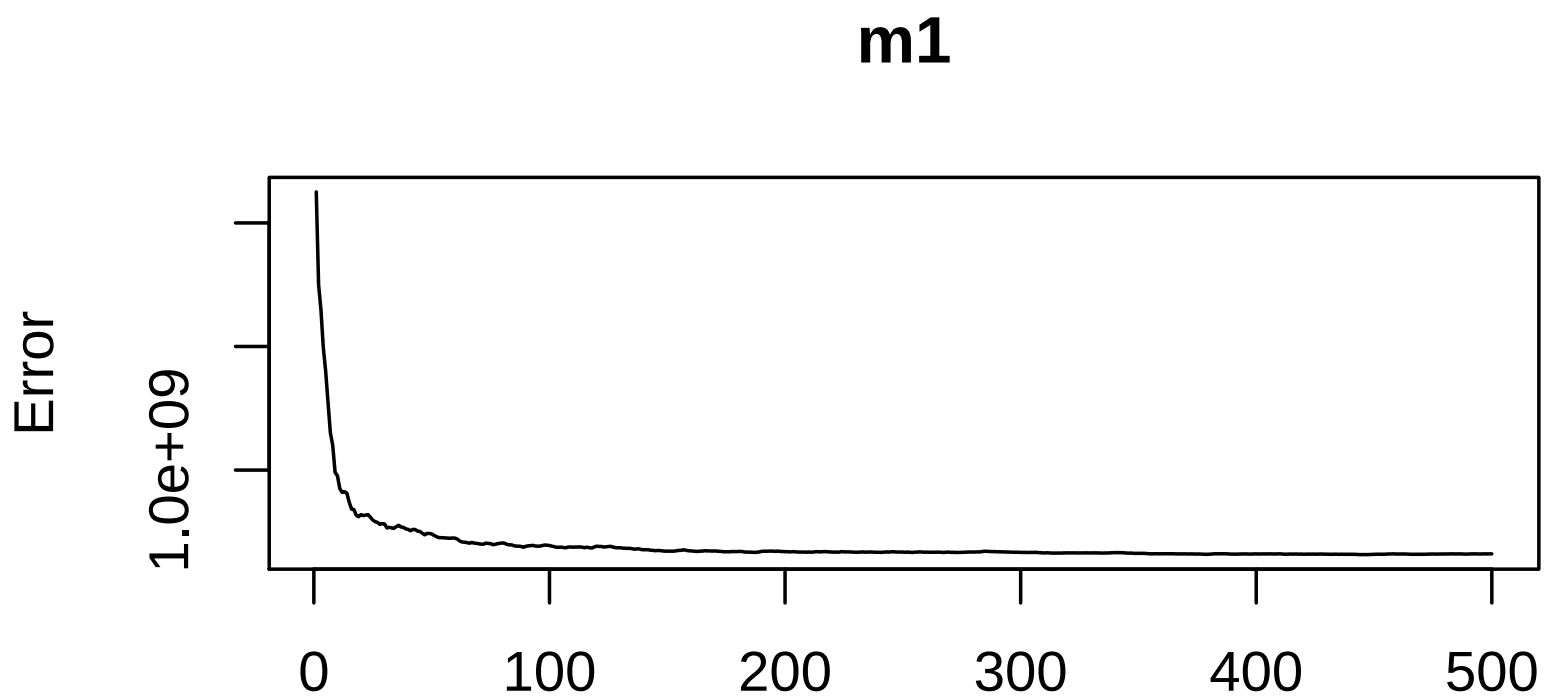
```
set.seed(123)
# default RF model
m1 <- randomForest(formula = Sale_Price ~ .,data=ames_train)
m1

##
## Call:
##  randomForest(formula = Sale_Price ~ ., data = ames_train)
##                Type of random forest: regression
```

Plotting the model

- Plotting the model will illustrate the error rate as we average across more trees and shows that our error rate stabalizes with around 100 trees but continues to decrease slowly until around 300 or so trees.

```
plot(m1)
```



The plotted error rate

- The plotted error rate above is based on the OOB sample error and can be accessed directly at `m1$mse`.
- We can find which number of trees providing the lowest error rate, which is 344 trees providing an average home sales price error of $25,673.

```
which.min(m1$mse)
```

```
## [1] 447
```

```
sqrt(m1$mse[which.min(m1$mse)])
```

```
## [1] 25648.78
```

A validation set to measure predictive accuracy

- `randomForest` also allows us to use a validation set to measure predictive accuracy if we did not want to use the OOB samples.
- Here we split our training set further to create a training and validation set.
- We then supply the validation data in the `xtest` and `ytest` arguments.