Introduction to Decision Trees and Random Forests and Their Applications in R

Jiajun Zhang

August 1, 2019

1 Introduction

Random forest is a well-known technique that has been widely used in machine learning, performing both regression and classification. For regression analysis, there is always a trade-off whether we should use linear regression or random forest. Also, the random forest has some unique features which are different than linear regression. In this paper, I will introduce the method of decision trees and random forests and their applications in R.

2 Decision Trees

Without further interpreting the use of random forest, it is essential to know the concept of a regression or classification tree since the random forest is an ensemble method that consists of many individual regression or classification trees.[2] As the name implies, classification trees separate data into classes where the response variable is categorical; similarly, regression trees apply to the cases where the response variable is numerical. Considered the dataset **iris** in R. By looking at the structure of this dataset, it contains both numerical variable measured in cm and categorical variable with 3 levels.

```
data(iris)
str(iris)
```

```
## 'data.frame': 150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Regression Tree

Assuming that our interested predictor variable is the sepal length and it is numeric, we can produce a regression tree with other explanatory variables from the dataset, as shown in Figure 1. It is clear to interpret the result by tracing the line from top to bottom based on the criterion at each split, until we reach the least bottom node.

```
reg_tree=rpart(Sepal.Length~., data=iris, method="anova")
rpart.plot(reg_tree, type=3, digits=3)
```

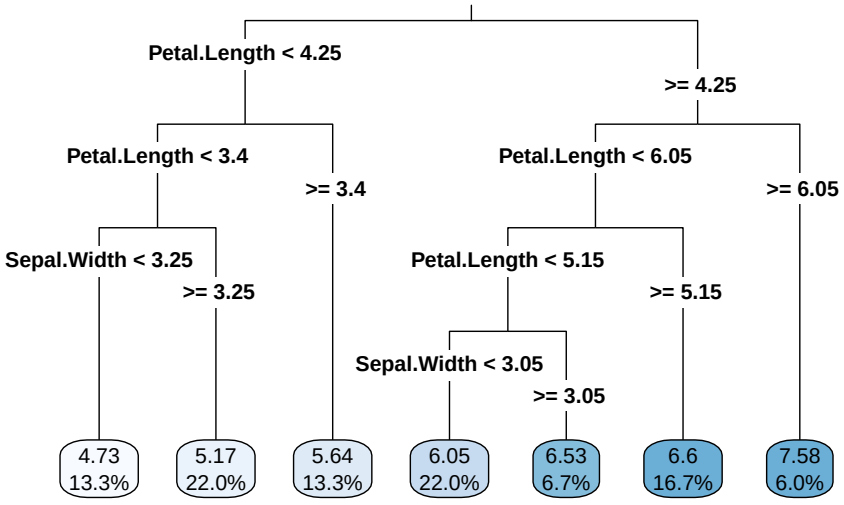

Figure 1: Single Regression Tree

Generally, the beginning of the regression tree is known as the root, while the terminal node is named as the leaf. Also, it is worth mentioning that there are two numbers in each leaf. The numbers with a percent sign represent the percentage of total observations. For example, in the leftmost leaf, it shows that 13.3% of total 150 observations are in that leaf. However, the numbers without a percent sign imply the predictive numbers of the response variable. These numbers are calculated by taking the average of the values of the response variable within the particular subset of data. For instance, in our **iris** dataset, we have an estimation of sepal length of 4.73cm if our data have a petal length less than 3.4cm and sepal width less than 3.25cm.

Classification Tree

Meanwhile, if we are interested in predicting the type of flowers, we will need to build a classification tree which takes the categorical variable of *species* as the response. The syntax of building a classification tree

is not much different than a regression tree. And the interpretation of the graph is almost the same except the leaves are now shown as the type of categories. For example, as shown in Figure 2, the tree suggests the flower is Versicolor if we have data with petal length at least 2.45cm and petal width less than 1.75cm.

```
class_tree=rpart(Species~., data=iris, method="class")
rpart.plot(class_tree, type=3, digits=3)
```

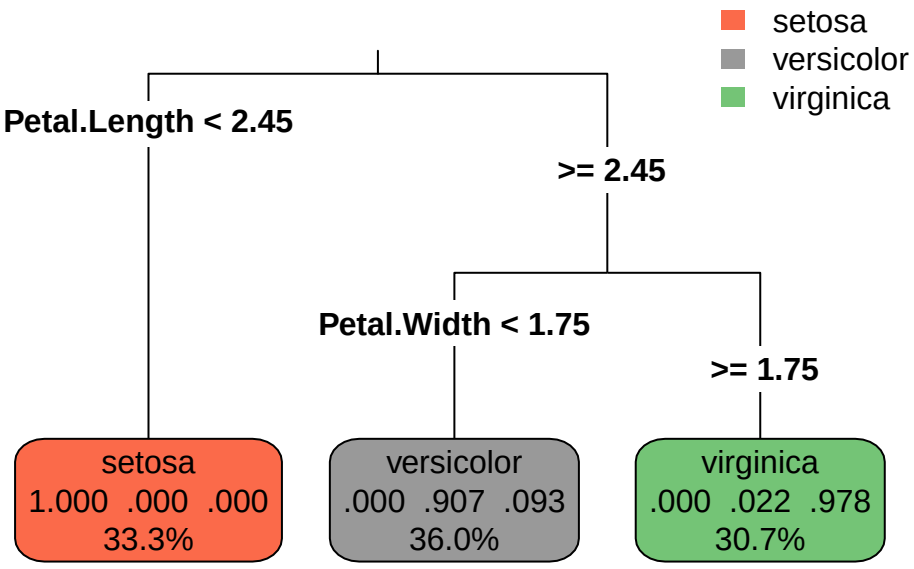

Figure 2: Single Classification Tree

Now we know how a single decision tree works for both numerical and categorical response variables, we can begin to consider modelling with random forest. Practically, decision trees are easy to build and easy to interpret. But frequently, they are not flexible and accurate to use when having a different set of new samples. However, random forests take advantage of simplicity from the decision trees, also with more flexibility in practical uses. Yet, building a single decision tree in a random forest is somehow different than before.

3 Random Forest

Bootstrapping

Other than using the entire dataset, we will use different bootstrap datasets to build different decision trees. Bootstrapping is a sampling technique that has also been widely in statistics, allowing us to randomly sample data with replacement from the entire dataset. That says some rows of observations can appear more than once in our new dataset built with bootstrapping. Then, after obtaining a bootstrap dataset, we will need to use it to construct a single decision tree. However, throughout a decision tree from the root to the leaf, we will use only a random subset of variables at each step. That implies, it is possible that some of the explanatory variables are not used; likewise, some might be used more than once. After building many individual decision trees, we aggregate the results from all the trees to make a prediction. Generally speaking, the above procedure is often referred to as bagging since it shortens for bootstrap aggregating. In short, random forests are created by many bagged decision trees where the trees contain a subset of features at each split.

The benefit of using a bootstrapped sample is that we are only considering a subset of the variables at each step, which results in a wide variety of trees. This makes random forests more effective in practical uses, and it provides a better prediction than individual trees if we have a different set of samples.

Back to our **iris** example, we now want to build a random forest using both response variables of *sepal length* and *species*. This process can be easily handled in R since the function `randomForest()` automatically done all the procedures for us. As previously discussed, bagging implies that we use random select samples instead of the original dataset. In order to keep the output consistent throughout the analysis, I will use a `set.seed()` function, which allows us to get the same random select samples every time.

Random Forest with Regreesion Trees

Now, we have a random forest model, and then we might want to determine how good is the model. One thing we can look at is the out-of-bag error. It is a method of measuring the prediction error of random forests.[1] When we are randomly selecting the data from the entire dataset, there are some entries which are included in the bootstrapped dataset. Those are considered as the out-of-bag dataset. And we will use these data to run through all the decision trees to determine how accurate our random forest is, by calculating the proportion of out-of-bag data which are incorrectly classified.

```
set.seed(123)
RF_reg=randomForest(Sepal.Length~., data=iris, ntree=500)
RF_reg

##
## Call:
```

```
##  randomForest(formula = Sepal.Length ~ ., data = iris, ntree = 500)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 0.1364625
##                     % Var explained: 79.97
```

From the output above, it suggests that the mean squared residuals is 0.136 and the percentage of variance explained is 79.97. The percentage of variance explained measures how well out-of-bag predictions explain the target variance. Since the number is around 80%, we might think this model explains the data reasonably good. Well, we can compare this model to a multiple linear regression model to see which performs better.

```
linear_reg=lm(Sepal.Length~., data=iris)
c(Rsquared=summary(linear_reg)$r.squared)

##  Rsquared
## 0.8673123

c(MSE=mean(linear_reg$residuals^2))

##        MSE
## 0.09037657
```

After fitting a linear regression with the same parameters, we obtain an R^2 value of 86.7% and a mean squared residuals of 0.09. Based on these two results, it is suggested that linear regression tends to do a better job with the response of sepal length.

Random Forest with Classification Trees

```
RF_class=randomForest(Species~., data=iris, ntree=500)
RF_class

##
## Call:
##  randomForest(formula = Species ~ ., data = iris, ntree = 500)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
```

```
##
##         OOB estimate of  error rate: 4%
## Confusion matrix:
##            setosa versicolor virginica class.error
## setosa         50          0         0        0.00
## versicolor      0         47         3        0.06
## virginica       0          3        47        0.06
```

However, one advantage of the random forest is that we can still use the method if our interested variable is categorical. On the other hand, linear regression does not have that performance. So, fitting a random forest with the response of *species*, we obtain an out-of-bag error rate of 4% which is pretty satisfying. This result provides that 96% of the time, the out-of-bag dataset is correctly classified throughout the decision trees. Also, the confusion matrix in the output tells us that which classes and the amount of incorrectly labels through all the decision trees.

Model Improvements

```
plot(RF_class, main="Percentage Error for Different Numbers of Trees")
legend("topright", col=c(2,3,4), lty=c(2,2,2),
       c("Setosa", "Virginica", "Versicolor"), bty="n", cex=0.8)
```

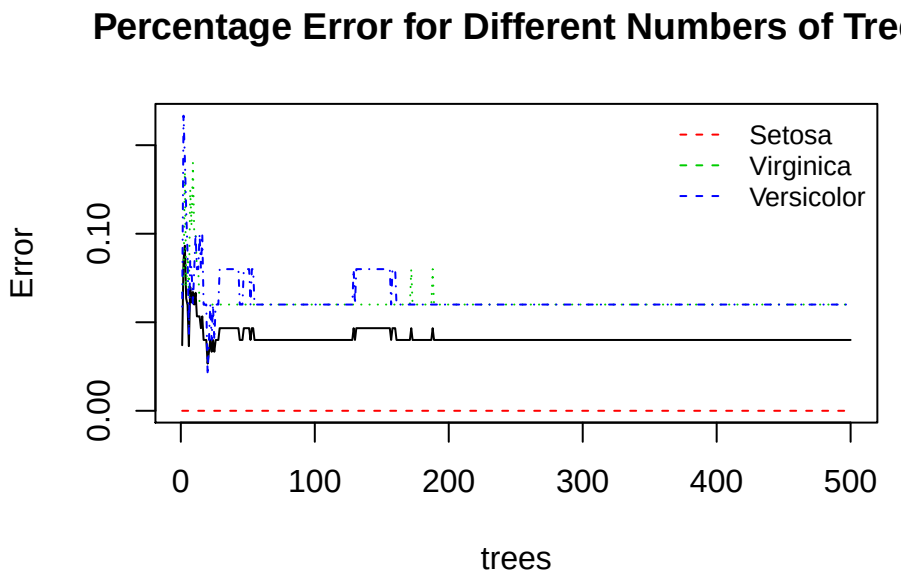

Figure 3: OOB Error Plot

```
tuneRF(iris[,-5], iris[,5], stepFactor=0.5, plot=TRUE, ntreeTry=200)
```

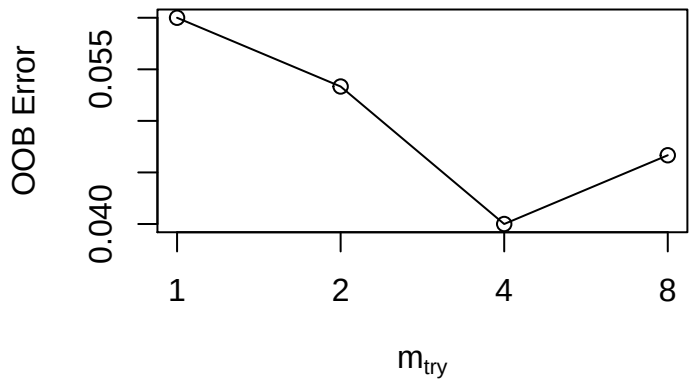

Figure 4: Tune Number of Variables at Each Split

```
RF_class1=randomForest(Species~., data=iris, ntree=200, mtry=4)
RF_class1

##
## Call:
##  randomForest(formula = Species ~ ., data = iris, ntree = 200,      mtry = 4)
##                Type of random forest: classification
##                      Number of trees: 200
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 4.67%
## Confusion matrix:
##            setosa versicolor virginica class.error
## setosa         50          0         0        0.00
## versicolor      0         47         3        0.06
## virginica       0          4        46        0.08
```

Sometimes, the model we obtain from random forests might not be as desirable as we want. However, we can tune parameters in random forests to see whether there is an improvement to the model. For example, Figure 3 suggests a graph of out-of-bag error rates for different species. It shows that we are not able to

improve this error rate after about 200 trees since all the lines remain constant after that. Moreover, Figure 4 indicates the out-of-bag error rate with the different number of variables tried at each split. It suggests that we will have the lowest error rate if we try four variables at each split. So, we use the alternative parameters and construct a new random forest. In this case, our model does not seem to be improved with the new parameters. This issue is probably because the **iris** dataset is considerably small and with few variables. But, ideally, random forest models can be improved by tuning the parameters.

4 Conclusion

In this paper, we have seen that building a model with decision trees is simple, and the model is easy to interpret. However, random forests provide better predictions with more flexibility in practical uses than decision trees. Also, we have performed both methods in R and drawn a comparison between linear regressions and random forests. While linear regression is known as making quantitative predictions, the random forest is able to perform both quantitative and qualitative predictions. Although the algorithm of random forests is more complicated than the decision tree, it tends to have better performances with more accurate predictions.

References

[1] Andy Liaw, Matthew Wiener, et al. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[2] Dan Steinberg and Phillip Colla. Cart: classification and regression trees. *The top ten algorithms in data mining*, 9:179, 2009.