Predicting wine quality using Random Forests

*February 4, 2016*

By [Teja Kodali](https://www.r-bloggers.com/author/teja-kodali/" \o "Posts by Teja Kodali)

[inShare](javascript:void(0);)

(This article was first published on **[DataScience+](http://datascienceplus.com/predicting-wine-quality-using-random-forests/)**, and kindly contributed to [R-bloggers)](https://www.r-bloggers.com/)

**177**

**SHARES**

[Share](https://www.facebook.com/sharer.php?u=https%3A%2F%2Fwww.r-bloggers.com%2Fpredicting-wine-quality-using-random-forests%2F)[Tweet](https://twitter.com/intent/tweet?text=Predicting%20wine%20quality%20using%20Random%20Forests&url=https://www.r-bloggers.com/predicting-wine-quality-using-random-forests/&via=Rbloggers)

Hello everyone! In this article I will show you how to run the random forest algorithm in R. We will use the [wine quality data set (white)](https://archive.ics.uci.edu/ml/datasets/Wine+Quality) from the UCI Machine Learning Repository.

What is the Random Forest Algorithm?

In a previous post, I outlined how to build decision trees in R. While decision trees are easy to interpret, they tend to be rather simplistic and are often outperformed by other algorithms. Random Forests are one way to improve the performance of decision trees. The algorithm starts by building out trees similar to the way a normal decision tree algorithm works. However, every time a split has to made, it uses only a small random subset of features to make the split instead of the full set of features (usually (sqrt[]{p}), where p is the number of predictors). It builds multiple trees using the same process, and then takes the average of all the trees to arrive at the final model. This works by reducing the amount of correlation between trees, and thus helping reduce the variance of the final tree. The simplest way to understand this is (as explained in Introduction to Statistical Learning): if you have some numbers (Z\_1, Z\_2,…,Z\_n) with a variance of (sigma^2), then their mean (overline{Z}) will have variance (sigma^2/n).

Exploring Data Analysis

Let us read in the data and explore it. We can read in the data directly from the page using the read.table function.

url <- '<https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-white.csv>'

wine <- read.table(url)

head(wine)

*fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide total.sulfur.dioxide density pH sulphates alcohol quality*

*1 7.0 0.27 0.36 20.7 0.045 45 170 1.0010 3.00 0.45 8.8 6*

*2 6.3 0.30 0.34 1.6 0.049 14 132 0.9940 3.30 0.49 9.5 6*

*3 8.1 0.28 0.40 6.9 0.050 30 97 0.9951 3.26 0.44 10.1 6*

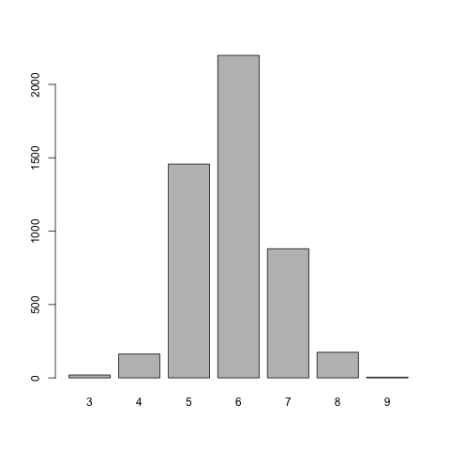
*4 7.2 0.23 0.32 8.5 0.058 47 186 0.9956 3.19 0.40 9.9 6*

*5 7.2 0.23 0.32 8.5 0.058 47 186 0.9956 3.19 0.40 9.9 6*

*6 8.1 0.28 0.40 6.9 0.050 30 97 0.9951 3.26 0.44 10.1 6*

Let us look at the distribution of the wine quality. We can use barplot for this.

barplot(table(wine$quality))

The barplot:  
[](https://i0.wp.com/datascienceplus.com/wp-content/uploads/2016/02/graph1.png)

As we can see, there are a lot of wines with a quality of 6 as compared to the others. The dataset description states – there are a lot more normal wines than excellent or poor ones. For the purpose of this discussion, let’s classify the wines into good, bad, and normal based on their quality.

wine$taste <- ifelse(wine$quality < 6, 'bad', 'good')

wine$taste[wine$quality == 6] <- 'normal'

wine$taste <- as.factor(wine$taste)

This will classify all wines into bad, normal, or good, depending on whether their quality is less than, equal to, or greater than 6 respectively. Let’s look at the distribution again.

table(wine$taste)

*bad good normal*

*1640 1060 2198*

Before we build our model, let’s separate our data into testing and training sets.

set.seed(123)

samp <- sample(nrow(wine), 0.6 \* nrow(wine))

train <- wine[samp, ]

test <- wine[-samp, ]

This will place 60% of the observations in the original dataset into train and the remaining 40% of the observations into test.

Building the model

Now, we are ready to build our model. We will need the randomForest library for this.

library(randomForest)

model <- randomForest(taste ~ . - quality, data = train)

We can use ntree and mtry to specify the total number of trees to build (default = 500), and the number of predictors to randomly sample at each split respectively. Let’s take a look at the model.

model

*Call:*

*randomForest(formula = taste ~ . - quality, data = train)*

*Type of random forest: classification*

*Number of trees: 500*

*No. of variables tried at each split: 3*

*OOB estimate of error rate: 29.54%*

*Confusion matrix:*

*bad good normal class.error*

*bad 683 16 274 0.2980473*

*good 16 404 229 0.3775039*

*normal 222 111 983 0.2530395*

We can see that 500 trees were built, and the model randomly sampled 3 predictors at each split. It also shows a matrix containing prediction vs actual, as well as classification error for each class. Let’s test the model on the test data set.

pred <- predict(model, newdata = test)

table(pred, test$taste)

*pred bad good normal*

*bad 482 10 130*

*good 14 252 85*

*normal 171 149 667*

We can test the accuracy as follows:

(482 + 252 + 667) / nrow(test)

*0.7147959*

There we have it! We achieved ~71.5% accuracy with a very simple model. It could be further improved by feature selection, and possibly by trying different values of mtry.

That brings us to the end of the article. I hope you enjoyed it! As always, if you have questions or feedback, feel free to reach out to me on [Twitter](https://twitter.com/tejaykodali) or leave a comment below!