Machine Learning: Random Forests - a Review

Satyanarayana Medicherla

CEO, SSP Technologies Inc.

Consultant, Canon U.S.A. Inc, 1 Canon Park, New York - 11747

*email-* [*satyams@hotmail.com*](mailto:satyams@hotmail.com)

### Abstract

Classification is one of the most important algorithms in Machine Learning Applications. Similar to Decision Trees, Random Forests is also one of the supervised machine learning algorithms, where our data set is labeled. In Random Forests, we will construct multiple decision trees using the same training data [1]. The collection of these Decision Trees is known as Random Forest. The more decision trees we have in a Random Forest, the more accurate would be our classification result or regression result [2].

### 

### 1. Introduction

Random Forest is Ensemble technique, where we train an ensemble of learners or models. These ensemble of learners combined together would give better prediction results when compared to the individual learners. In Random Forests, these individual learners are Decision Trees. The individual decision trees would have high variance. The combination of these decision trees, the random forest would considerably reduce this variance [3].

### 2. Ensemble Learning

Ensemble Learning combines many learners because more predictors are always better than a single one [4]. All these individual learners could be of the same type or different types. The Ensemble Learning could use different techniques like Bagging, Boosting etc. We will discuss Bagging and Boosting in the following sections.

While determining the result of the combined learners, we could either use majority vote or weighted averages or we can use some kind of “predictor of predictors”, where we train another model on these predictors.

### 2.1 Bagging

Bagging stands for Bootstrap Aggregation. We would create subsets of data from the training data with replacement and train individual decision trees on these subsets. We will predict the outcome of a test example based on the majority voting for Classification problems and average for regression problems [5]. The individual learners would have high variance and when we use these together for prediction, the variance would be considerably reduced.

Since Bagging uses the sampling with replacement, it would create new data samples and some of the data elements will be repeated in sampled subsets. In Bagging any element would have same probability of getting sampled.

Bagging is very effective way to reduce the variance of a classifier. Decision trees grown fully would have a very high variance. The decision tree captures the characteristics specific to the training set D (a subset of P) and which is not common in P [3]. We will draw n different subsets of data from P with replacements. Now learn classifiers on the sampled subsets .

One of the popular instances of bagging is Random Forests - it is a set of bagged decision trees.

While building decision trees, we would use k dimensions at each split where k < d, where d is the dimensionality of D. The following expression would give us the value for k.

for classification and for regression.

#### 2.1.1 Bagged Decision Trees

As we discussed earlier, we will sample many different subsets from our training data and build Decision Trees on those subsets. These trees are called Bagged Decision Trees. The prediction is based on majority voting.

Let us say that our training data set is D with n different data elements and with d features. are drawn with replacement from D are trained on

The combination of these learners would be used for prediction

The larger the number of subsets that we sample, the better the classification result would be. So the number of subsets is completely independent of the size of the training data set n. The size of each subset is the same as the size of D.

Let us say that there are d features, we will use k < d features at every split of the decision tree.

#### 2.1.2 Out of Bag Error [3]

Random Forest uses the out of bag data for testing and there is no need for separate test data.

In other machine learning models, we split the data set into two parts e.g. 80% as training and 20% on the test data. We train our model on the 80% of the data and it is tested on the remaining 20% of the data. But in case of Random Forest, there is no such splitting is needed. It works on the out of bag error.

As we mentioned earlier, each of the sampled subset would have approximately 60% distinct elements and the rest would be duplicated. So for testing, we would use those left over samples from each subset to test the corresponding learner. While testing each learner, we would use those left over samples from that subset.

Loss Function is the sum of all losses over all the trees of the Random Forest.

The out of bag error is the average of all the errors of all our leaners on their out of bag samples.

Here is the logic for computing out of bag error for a random forest 3].

-Initialize Error = 0

-For each data point in our training set D

- For each learn trained on our subset that does not contain

- Error = Error + Compute the error of on $(xi,yi)

- end

-End

where

which is just the count of all of our subsets that does not contain the data point

### 3. Decision Trees Vs Random Forests Examples

Let us take a very simple example of Decision Tree using the R supplied iris flower data set, which is an in built data set of R.

library(rpart)  
library(rpart.plot)  
set.seed(111)  
s <- sample(length(iris$Species),100)  
iris\_train <- iris[s,]  
iris\_test <- iris[-s,]  
  
model\_tree <- rpart(data=iris\_train,Species~.,method="class")  
model\_tree

## n= 100   
##   
## node), split, n, loss, yval, (yprob)  
## \* denotes terminal node  
##   
## 1) root 100 61 versicolor (0.30000000 0.39000000 0.31000000)   
## 2) Petal.Length< 2.45 30 0 setosa (1.00000000 0.00000000 0.00000000) \*  
## 3) Petal.Length>=2.45 70 31 versicolor (0.00000000 0.55714286 0.44285714)   
## 6) Petal.Width< 1.6 38 1 versicolor (0.00000000 0.97368421 0.02631579) \*  
## 7) Petal.Width>=1.6 32 2 virginica (0.00000000 0.06250000 0.93750000) \*

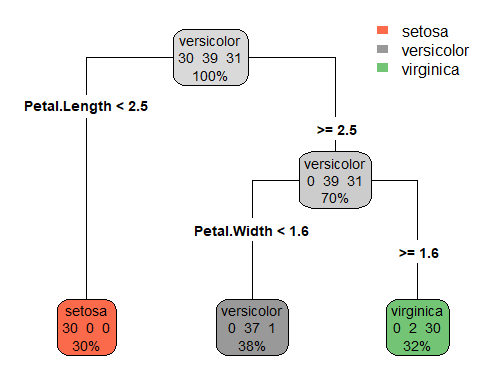
rpart.plot(model\_tree, type = 4, extra = 101)  
  
pred\_tree <- predict(model\_tree, iris\_test, type = "class")  
  
table(pred\_tree, iris\_test$Species)

##   
## pred\_tree setosa versicolor virginica  
## setosa 20 0 0  
## versicolor 0 8 2  
## virginica 0 3 17

library(caret)

## Loading required package: lattice

## Loading required package: ggplot2



confusionMatrix(table(pred\_tree, iris\_test$Species))

## Confusion Matrix and Statistics  
##   
##   
## pred\_tree setosa versicolor virginica  
## setosa 20 0 0  
## versicolor 0 8 2  
## virginica 0 3 17  
##   
## Overall Statistics  
##   
## Accuracy : 0.9   
## 95% CI : (0.7819, 0.9667)  
## No Information Rate : 0.4   
## P-Value [Acc > NIR] : 2.196e-13   
##   
## Kappa : 0.8447   
##   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: setosa Class: versicolor Class: virginica  
## Sensitivity 1.0 0.7273 0.8947  
## Specificity 1.0 0.9487 0.9032  
## Pos Pred Value 1.0 0.8000 0.8500  
## Neg Pred Value 1.0 0.9250 0.9333  
## Prevalence 0.4 0.2200 0.3800  
## Detection Rate 0.4 0.1600 0.3400  
## Detection Prevalence 0.4 0.2000 0.4000  
## Balanced Accuracy 1.0 0.8380 0.8990

library(randomForest)

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##   
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':  
##   
## margin

model\_forest <- randomForest(Species ~ ., data = iris\_train)  
model\_forest

##   
## Call:  
## randomForest(formula = Species ~ ., data = iris\_train)   
## Type of random forest: classification  
## Number of trees: 500  
## No. of variables tried at each split: 2  
##   
## OOB estimate of error rate: 5%  
## Confusion matrix:  
## setosa versicolor virginica class.error  
## setosa 30 0 0 0.00000000  
## versicolor 0 37 2 0.05128205  
## virginica 0 3 28 0.09677419

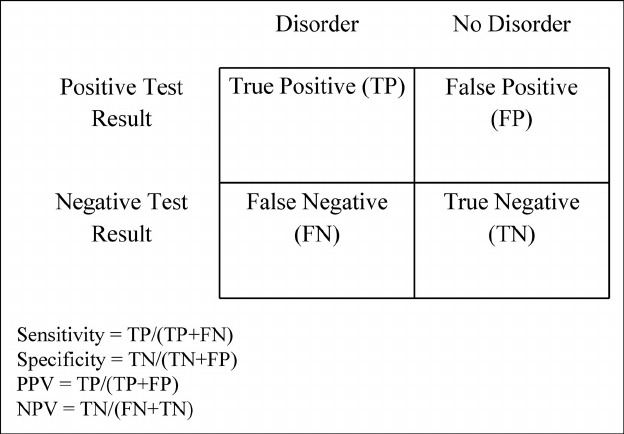
pred\_forest <- predict(model\_forest, iris\_test, type = "class")  
  
table(pred\_forest, iris\_test$Species)

##   
## pred\_forest setosa versicolor virginica  
## setosa 20 0 0  
## versicolor 0 11 2  
## virginica 0 0 17

library(caret)  
  
confusionMatrix(table(pred\_forest, iris\_test$Species))

## Confusion Matrix and Statistics  
##   
##   
## pred\_forest setosa versicolor virginica  
## setosa 20 0 0  
## versicolor 0 11 2  
## virginica 0 0 17  
##   
## Overall Statistics  
##   
## Accuracy : 0.96   
## 95% CI : (0.8629, 0.9951)  
## No Information Rate : 0.4   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.9388   
##   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: setosa Class: versicolor Class: virginica  
## Sensitivity 1.0 1.0000 0.8947  
## Specificity 1.0 0.9487 1.0000  
## Pos Pred Value 1.0 0.8462 1.0000  
## Neg Pred Value 1.0 1.0000 0.9394  
## Prevalence 0.4 0.2200 0.3800  
## Detection Rate 0.4 0.2200 0.3400  
## Detection Prevalence 0.4 0.2600 0.3400  
## Balanced Accuracy 1.0 0.9744 0.9474

Here is a quick reference for the columns displayed in Confusion Matrix above [6]



Decision Tree for Tennis Game Data Set

Accuracy is the ratio of the total correct predictions to the overall total. The total correct predictions is TP+TN and total is TP+TN+FP+FN.

As we can see that the classification percentage is much higher for Random Forest when compared to its corresponding Decision Tree.

Let us take another example and show how Random Forest performs better when compared to Decision Trees. Here we use the Red Wind Quality data set that is available on UCLA web site.[9]

setwd("C:/Coursera/R/MachineLearning")  
wine <- read.csv('winequality-red.csv')  
dim(wine)

## [1] 1599 12

wine$quality <- as.factor(wine$quality)  
str(wine$quality)

## Factor w/ 6 levels "3","4","5","6",..: 3 3 3 4 3 3 3 5 5 3 ...

set.seed(111)  
s <- sample(length(wine$quality), 1200)  
wine\_train <- wine[s, ]  
wine\_test <- wine[-s, ]  
dim(wine\_train)

## [1] 1200 12

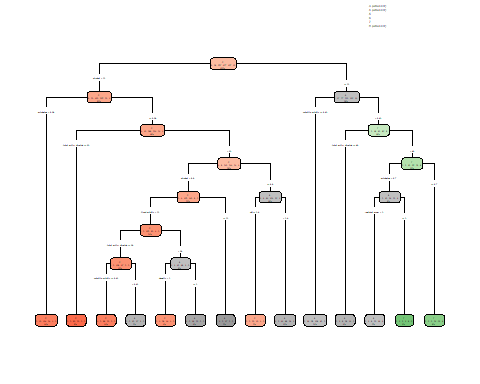
dim(wine\_test)

## [1] 399 12

library(rpart)  
  
tm <- rpart(quality~., wine\_train, method = "class")  
  
library(rpart.plot)  
  
tm

## n= 1200   
##   
## node), split, n, loss, yval, (yprob)  
## \* denotes terminal node  
##   
## 1) root 1200 693 5 (0.0067 0.032 0.42 0.4 0.13 0.01)   
## 2) alcohol< 10.525 746 316 5 (0.008 0.029 0.58 0.34 0.043 0.0013)   
## 4) sulphates< 0.535 173 39 5 (0.012 0.069 0.77 0.14 0.0058 0) \*  
## 5) sulphates>=0.535 573 277 5 (0.007 0.017 0.52 0.4 0.054 0.0017)   
## 10) total.sulfur.dioxide>=82.5 105 22 5 (0 0.019 0.79 0.19 0 0) \*  
## 11) total.sulfur.dioxide< 82.5 468 255 5 (0.0085 0.017 0.46 0.45 0.066 0.0021)   
## 22) alcohol< 9.85 275 123 5 (0.011 0.015 0.55 0.4 0.022 0)   
## 44) fixed.acidity< 10.85 252 102 5 (0.0079 0.012 0.6 0.37 0.016 0)   
## 88) total.sulfur.dioxide>=28.5 161 53 5 (0.0062 0.012 0.67 0.29 0.019 0)   
## 176) volatile.acidity>=0.405 133 35 5 (0.0075 0.015 0.74 0.23 0.015 0) \*  
## 177) volatile.acidity< 0.405 28 11 6 (0 0 0.36 0.61 0.036 0) \*  
## 89) total.sulfur.dioxide< 28.5 91 45 6 (0.011 0.011 0.46 0.51 0.011 0)   
## 178) density< 0.99716 44 16 5 (0 0.023 0.64 0.32 0.023 0) \*  
## 179) density>=0.99716 47 15 6 (0.021 0 0.3 0.68 0 0) \*  
## 45) fixed.acidity>=10.85 23 6 6 (0.043 0.043 0.087 0.74 0.087 0) \*  
## 23) alcohol>=9.85 193 92 6 (0.0052 0.021 0.32 0.52 0.13 0.0052)   
## 46) pH>=3.445 35 15 5 (0 0.057 0.57 0.34 0.029 0) \*  
## 47) pH< 3.445 158 69 6 (0.0063 0.013 0.26 0.56 0.15 0.0063) \*  
## 3) alcohol>=10.525 454 232 6 (0.0044 0.037 0.17 0.49 0.28 0.024)   
## 6) volatile.acidity>=0.425 256 117 6 (0.0078 0.055 0.23 0.54 0.16 0.012) \*  
## 7) volatile.acidity< 0.425 198 113 7 (0 0.015 0.096 0.42 0.43 0.04)   
## 14) total.sulfur.dioxide>=47.5 49 21 6 (0 0.02 0.18 0.57 0.22 0) \*  
## 15) total.sulfur.dioxide< 47.5 149 75 7 (0 0.013 0.067 0.37 0.5 0.054)   
## 30) sulphates< 0.695 73 39 6 (0 0.027 0.14 0.47 0.32 0.055)   
## 60) residual.sugar< 2.95 60 27 6 (0 0.017 0.13 0.55 0.23 0.067) \*  
## 61) residual.sugar>=2.95 13 4 7 (0 0.077 0.15 0.077 0.69 0) \*  
## 31) sulphates>=0.695 76 25 7 (0 0 0 0.28 0.67 0.053) \*

rpart.plot(tm, type = 4, extra = 101)



pred\_tree <- predict(tm, wine\_test, type = "class")  
  
table(pred\_tree, wine\_test$quality)

##   
## pred\_tree 3 4 5 6 7 8  
## 3 0 0 0 0 0 0  
## 4 0 0 0 0 0 0  
## 5 1 8 112 33 1 0  
## 6 1 6 58 111 23 5  
## 7 0 0 4 17 18 1  
## 8 0 0 0 0 0 0

library(caret)  
  
confusionMatrix(table(pred\_tree, wine\_test$quality))

## Confusion Matrix and Statistics  
##   
##   
## pred\_tree 3 4 5 6 7 8  
## 3 0 0 0 0 0 0  
## 4 0 0 0 0 0 0  
## 5 1 8 112 33 1 0  
## 6 1 6 58 111 23 5  
## 7 0 0 4 17 18 1  
## 8 0 0 0 0 0 0  
##   
## Overall Statistics  
##   
## Accuracy : 0.604   
## 95% CI : (0.5541, 0.6523)  
## No Information Rate : 0.4361   
## P-Value [Acc > NIR] : 1.188e-11   
##   
## Kappa : 0.3548   
##   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8  
## Sensitivity 0.000000 0.00000 0.6437 0.6894 0.42857 0.00000  
## Specificity 1.000000 1.00000 0.8089 0.6092 0.93838 1.00000  
## Pos Pred Value NaN NaN 0.7226 0.5441 0.45000 NaN  
## Neg Pred Value 0.994987 0.96491 0.7459 0.7436 0.93315 0.98496  
## Prevalence 0.005013 0.03509 0.4361 0.4035 0.10526 0.01504  
## Detection Rate 0.000000 0.00000 0.2807 0.2782 0.04511 0.00000  
## Detection Prevalence 0.000000 0.00000 0.3885 0.5113 0.10025 0.00000  
## Balanced Accuracy 0.500000 0.50000 0.7263 0.6493 0.68347 0.50000

library(randomForest)  
model\_forest\_wine <- randomForest(quality ~ ., data = wine\_train)  
  
pred\_forest\_wine <- predict(model\_forest\_wine, newdata = wine\_test,type="class")  
  
table(pred\_forest\_wine,wine\_test$quality)

##   
## pred\_forest\_wine 3 4 5 6 7 8  
## 3 0 0 0 0 0 0  
## 4 0 0 0 0 0 0  
## 5 2 13 135 35 2 0  
## 6 0 1 37 118 15 4  
## 7 0 0 2 8 25 2  
## 8 0 0 0 0 0 0

confusionMatrix(table(pred\_forest\_wine,wine\_test$quality))

## Confusion Matrix and Statistics  
##   
##   
## pred\_forest\_wine 3 4 5 6 7 8  
## 3 0 0 0 0 0 0  
## 4 0 0 0 0 0 0  
## 5 2 13 135 35 2 0  
## 6 0 1 37 118 15 4  
## 7 0 0 2 8 25 2  
## 8 0 0 0 0 0 0  
##   
## Overall Statistics  
##   
## Accuracy : 0.6967   
## 95% CI : (0.649, 0.7415)  
## No Information Rate : 0.4361   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.5019   
##   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8  
## Sensitivity 0.000000 0.00000 0.7759 0.7329 0.59524 0.00000  
## Specificity 1.000000 1.00000 0.7689 0.7605 0.96639 1.00000  
## Pos Pred Value NaN NaN 0.7219 0.6743 0.67568 NaN  
## Neg Pred Value 0.994987 0.96491 0.8160 0.8080 0.95304 0.98496  
## Prevalence 0.005013 0.03509 0.4361 0.4035 0.10526 0.01504  
## Detection Rate 0.000000 0.00000 0.3383 0.2957 0.06266 0.00000  
## Detection Prevalence 0.000000 0.00000 0.4687 0.4386 0.09273 0.00000  
## Balanced Accuracy 0.500000 0.50000 0.7724 0.7467 0.78081 0.50000

As we can see that the classification performance is much higher for Random Forest when compared to its Decision Tree model.

### 4. Boosting [8]

Boosting is another technique where the subsets are created and a model is trained on that subset sequentially.

#### 4.1 AdaBoost

AdaBoost is Adaptive Boosting, where training a model on the training set, the model is tested using the original training set. Then it will increase the weights on each of the wrongly classified training samples. Now a new subset is created considering the weights assigned to the wrongly classified samples. Now the chances of appearing these misclassified samples is very high when compared to the correctly classified samples. A model is trained on this new set and test is conducted on the new training set. This process is continued until the desired test accuracy is achieved.

In case of Bagging, all the learners are given equal importance for prediction. But in case of Boosting, weights are assigned to each learner and the result is obtained by weighted average of the N learners. The assigned weights are proportional to the accuracy of the individual learner.

#### 4.2 Gradient Boost [10]

Gradient Boosting is also an Ensemble Learning. This is similar to Adaboost where we train learners sequentially.

In Gradient Boosting for regression problems, we first build a regression tree. The average for the target variable is computed for the entire training set and then the residuals are computed for each sample. We build a decision tree for these residuals. For all the leaf nodes, we compute the average of the elements of that leaf node and subtract the average from all the leaf node elements. We will build a decision tree on these residuals/errors as training data. When a new tree is constructed on the residuals, we will predict the prices as follows.

where is the learning rate. Again we will compute the new residuals and build a new tree for these residuals. We will continue this process for a specified number iterations. We will use all the trees in the ensemble for prediction in the sequence. If our final ensemble contains n decision trees, the predicted price would be given by

### 5. Gradient Boosted Decision Tree Example [11]

require(gbm)

## Loading required package: gbm

## Loaded gbm 2.1.5

require(MASS)#package with the boston housing dataset

## Loading required package: MASS

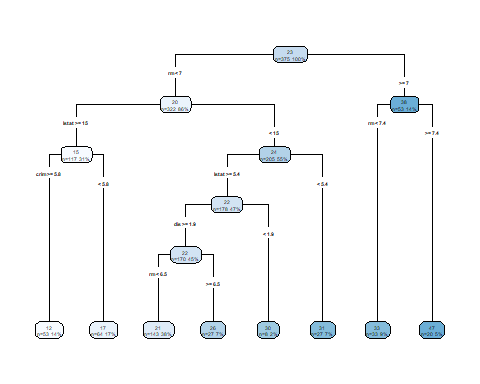
length(Boston$medv)

## [1] 506

#separating training and test data  
  
set.seed(111)  
s=sample(length(Boston$medv),size=375)  
Boston\_train = Boston[s,]  
Boston\_test = Boston[-s,]  
  
  
library(rpart)  
  
tm <- rpart(medv~., Boston\_train, method="anova")  
  
library(rpart.plot)  
  
tm

## n= 375   
##   
## node), split, n, deviance, yval  
## \* denotes terminal node  
##   
## 1) root 375 32571.4100 22.88880   
## 2) rm< 6.9715 322 14226.5400 20.33851   
## 4) lstat>=14.745 117 2472.1520 14.78205   
## 8) crim>=5.7819 53 813.2192 11.79623 \*  
## 9) crim< 5.7819 64 795.1386 17.25469 \*  
## 5) lstat< 14.745 205 6080.4600 23.50976   
## 10) lstat>=5.41 178 3234.8630 22.39382   
## 20) dis>=1.8748 170 1626.8090 22.02882   
## 40) rm< 6.5445 143 961.8120 21.33217 \*  
## 41) rm>=6.5445 27 228.0207 25.71852 \*  
## 21) dis< 1.8748 8 1104.1400 30.15000 \*  
## 11) lstat< 5.41 27 1162.5800 30.86667 \*  
## 3) rm>=6.9715 53 3526.8350 38.38302   
## 6) rm< 7.437 33 865.8824 33.24848 \*  
## 7) rm>=7.437 20 355.4695 46.85500 \*

rpart.plot(tm, type = 4, extra = 101)



pred\_tree <- predict(tm, Boston\_test)  
  
prices <- data.frame(Boston\_test$medv,pred\_tree)  
colnames(prices) <- c("Actual Price","Predicted Price")  
head(prices)

## Actual Price Predicted Price  
## 1 24.0 30.86667  
## 4 33.4 33.24848  
## 15 18.2 21.33217  
## 20 18.2 21.33217  
## 24 14.5 17.25469  
## 31 12.7 17.25469

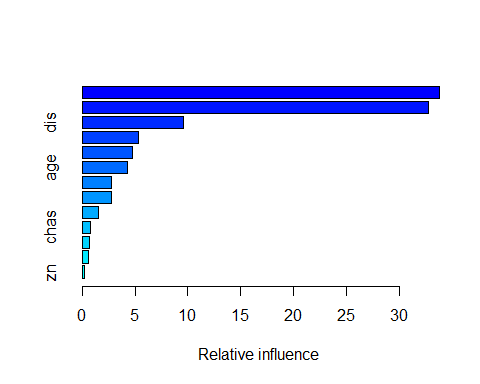
sum(abs(Boston\_test$medv-pred\_tree))

## [1] 463.936

Boston\_boost\_model=gbm(medv ~ . ,data = Boston\_train,,distribution = "gaussian",n.trees = 10000,  
 shrinkage = 0.01, interaction.depth = 4)  
Boston\_boost\_model

## gbm(formula = medv ~ ., distribution = "gaussian", data = Boston\_train,   
## n.trees = 10000, interaction.depth = 4, shrinkage = 0.01)  
## A gradient boosted model with gaussian loss function.  
## 10000 iterations were performed.  
## There were 13 predictors of which 13 had non-zero influence.

summary(Boston\_boost\_model) # Variable Importance and a plot of Variable Importance



## var rel.inf  
## rm rm 33.8115512  
## lstat lstat 32.7458809  
## dis dis 9.6131263  
## crim crim 5.3462089  
## nox nox 4.7956804  
## age age 4.3023827  
## black black 2.7679855  
## ptratio ptratio 2.7384966  
## tax tax 1.5108186  
## chas chas 0.7942549  
## indus indus 0.7148887  
## rad rad 0.6253510  
## zn zn 0.2333744

pred\_boost <- predict(Boston\_boost\_model,Boston\_test,n.trees=10000)  
  
prices <- data.frame(Boston\_test$medv,pred\_boost)  
colnames(prices) <- c("Actual Price","Predicted Price")  
head(prices)

## Actual Price Predicted Price  
## 1 24.0 26.24990  
## 2 33.4 35.45936  
## 3 18.2 18.76506  
## 4 18.2 18.98793  
## 5 14.5 15.96198  
## 6 12.7 12.42299

sum(abs(Boston\_test$medv-pred\_boost))

## [1] 285.7389

The actual and predicted prices for the first 5 houses in the test set are very close. The total error for our DecisionTree model is 464 and it is only 286 for our Boosted Decision Trees Model.

### References

1. <https://towardsdatascience.com/understanding-random-forest-58381e0602d2>
2. <https://syncedreview.com/2017/10/24/how-random-forest-algorithm-works-in-machine-learning/>
3. <http://www.cs.cornell.edu/courses/cs4780/2017sp/lectures/lecturenote17.html>
4. <https://www.youtube.com/watch?v=Yvn3--rIdZg>
5. <https://www.youtube.com/watch?v=LsK-xG1cLYA>
6. <https://www.researchgate.net/figure/Calculation-of-sensitivity-specificity-and-positive-and-negative-predictive_fig1_49650721>
7. <https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/>
8. <https://www.youtube.com/watch?v=J4Wdy0Wc_xQ>
9. <https://www.kdnuggets.com/2017/11/difference-bagging-boosting.html>
10. <https://towardsdatascience.com/machine-learning-part-18-boosting-algorithms-gradient-boosting-in-python-ef5ae6965be4>
11. <https://www.r-bloggers.com/gradient-boosting-in-r/>