Ensemble Classification: An Analysis of the Random Forest Model

Jarod Korn
jjk131@uakron.edu

An Analysis of the Random Forest Model For Classification Prediction

Jarod Korn

Department of Statistics
University of Akron
May 2024

Contents

1 Abstract

The random forest model proposed by Dr. Leo Breiman in 2001 [Bre01] is an ensemble machine learning method for classification prediction and regression. In the following paper, we will conduct an analysis on the random forest model with a focus on how the model works, how it is applied in software, and how it performs on a set of data. To fully understand the model, we will introduce the concept of decision trees, give a summary of the CART model, explain in detail how the random forest model operates, discuss how the model is implemented in software, demonstrate the model by applying it to a set of data, evaluate the performance of the model, and finally compare the model and its performance to the simpler CART model. By the end, a reader should have an introductory understanding of decision tree methodology, be knowledgeable about the mechanics of both the CART and random forest models, an understand both the benefits and hindrances of the random forest model.

2 Decision Trees

This section is based on material learned in the Applied Analytic Decision Trees course taught at the University of Akron [Fri23]. Further reference information on decision trees was gathered from IBM [IBMa] and cross validation information from the website geeksforgeeks [Geec].

2.1 Introduction

Decision trees are a statistical/machine learning method in which a categorical/quantitative variable is predicted from a list of other categorical/quantitative variables. The variable being predicted is referred to as the response while the variables being used to predict the response variable are refereed to as predictors. Tree models use these predictors to split the data set into subsets based on a criteria. While the splitting criteria varies from model to model, the general idea is to always choose the split that leads to better predictions. These splits occur recursively, meaning that every subset has the possibility to be further split into more subsets. In decision tree terminology, the sets being split are referred to as parent nodes/parents and the resulting subsets are referred to as child nodes/children. Once a certain stopping criteria is reached, the model will no longer consider splits and the growth of the tree will cease. The child nodes at the end of the branches are now refereed to as terminal nodes. It is important to note that, due to the splitting, every data point in the original sample will fall into only one of the terminal nodes. Because of this, the terminal nodes are used to classify the data based on the highest proportion of class in that node.

2.2 Variance and Bias

With all statistical and machine learning methods, the primary goal is to always provide a good estimation of the population parameter of interest. The accuracy of these estimations is found by computing both the bias and variance of the estimator, resulting in the error estimate

$$Error = [Bias]^2 + Variance$$

Bias is the difference between the theoretical/predicted value and the true value. Essentially, the larger the difference between our estimation and the true value, the larger the bias. Bias in particular poses a problem for our estimation as the term is squared in the error estimate, meaning that small deviations in the bias result in larger changes in the error.

Variance is the measure of the spread of the data. In predictive modeling, variance refers to the change in the predictions over different training sets. That is, a model is highly variable if slightly different training sets result in much different predictions.

The interaction between bias and variance is referred to as the bias-variance trade off as, in practice, it is often not possible to reduce both at once. As such, the combinations of low bias, high variance and high bias, low variance are referred to as overfitting and under fitting respectively. As the names imply, overfitting is the tendency of a model to too close rely on training data, such that the predictions are too variable. Conversely, underfitting is the tendency of the model to not rely enough on the training set such that predictions do not actually predict the target population.

2.3 Cross-Validation

Cross-validation is a means of measuring how overfit a model is by splitting the original dataset into multiple subsets. The number of subsets the original dataset is broken into is not a fixed value and can be chosen by the analyst. Typically a split of either 2, 5, or 10 is chosen depending on the size of the dataset and available computational power. After the splits, one of the subsets is then chosen to be the testing dataset while the others are collected into the training dataset [Geec].

The training dataset is the subset or collection of subsets that will be used to build/train the model. The testing dataset is the subset that the model will be run on to test the model's performance. The benefit of this validation is that it allows analysts to compare the performance of the training and testing sets. Should the performance significantly differ between the training and testing sets, then the model is likely overfit.

Generally speaking, the training data set needs to be large enough to build the model, but not so large that the testing data set is too small. The typical range of proportions for the data sets is between a 50:50 split and a 80:20 split depending on the size of the original data set. A larger data set can afford to use a higher split ratio whereas a smaller data set can not. Because of this convention, decision tree models require a relatively large sample of data.

2.4 Prediction Measures

2.4.1 Confusion Matrix

The confusion matrix of a decision tree is a cross tabulation table of the predicted classifications vs the actual classifications from the data. The matrix is primarily used to calculate other prediction measures but it is also used to visualize the performance of the model. For illustration purposes, a confusion matrix generated from a decision tree predicting a binary response variable is of the form

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

The rows of the matrix are the predicted classifications whereas the columns are the actual classifications. Therefore, a and d are the correctly predicted classifications and b and c are the incorrectly predicted classifications. It is important to note that the confusion matrix is not restricted to binary responses and a model that has n possible classifications will have a confusion matrix of size $n_x n$ with its main diagonal being the correctly predicted classifications.

2.4.2 Misclassification Risk

Misclassification risk is the primary statistic used to measure the predictive performance of a decision tree model. It is the proportion of incorrectly classified response and can be calculated from the confusion matrix as follows

$$MisclassificationRisk = 1 - \frac{a+d}{n} = \frac{b+c}{n}$$

Where a and d are the correctly predicted classifications from the confusion matrix and n is the size of the data. Like the confusion matrix, the misclassification risk is also not restricted to binary responses.

2.4.3 Sensitivity and Specificity

Sensitivity and specificity, like the misclassification risk, measure the performance of a decision tree model. Sensitivity is a measure of how accurately the model predicts the target classification, that is the classification of interest. To find the sensitivity, calculate the proportion of correct predictions for the target classification compared to the number of the target class in the data. For the binary response variable case, this is calculated from

$$Sensitivity = \frac{a}{a+c}$$

Specificity refers to how accurately the model predicts the non-target classification(s). To find specificity, calculate the proportion of correct predictions for the non-target classification(s) compared to the overall number of non-target class(es) in the data. For the binary response case, this is calculated from

$$Specificity = \frac{d}{b+d}$$

2.5 Model Evaluation

In general, a decision tree model is considered to have performed well if it shows little to no signs of overfitting and predicts the classifications with a desired level of accuracy. To gauge how overfit the decision tree model, analysts compare the results of the prediction measures discussed previously from the training data set to the testing data set. That is, for each data set, the prediction measures of misclassification risk, sensitivity, and specificity will be calculated. Then, if there is a significant difference between any of the predictive measures, there is an increased chance of overfitting. To gauge the accuracy of the predictions, look at the values of the misclassification risk, sensitivity, and specificity. A well performing model will have a relatively low misclassification risk and relatively high sensitivity and specificity in the context of the test data.

3 CART Model

This section is based on material learned in the Applied Analytic Decision Trees course taught at the University of Akron [Fri23]

3.1 Introduction

Classification and regression trees, CART for short, is a decision tree model developed by Leo Breiman in which only binary splits are considered, that is, parent nodes can only be split into two child nodes. Compared to other decision tree models, the reduced number of potential splits often leads to a simpler model generally requires considerably less computational power to run.

3.2 Gini Impurity and Goodness of Split

Gini impurity is the measure of how impure a sample is where the purity of a sample is the proportion of the majority class to the other classes. For example, if we had classes A, B, and C with proportions 0.9, 0.03, and 0.02, the sample would have high purity as class A has the super majority. Gini impurity is calculated as follows

$$Gini = 1 - \sum_{i=1}^{k} p_i^2$$

where p_i is the proportion of each classification in the sample.

Goodness of split is a method of measuring the change in Gini impurity from the parent node to the child nodes. A split is chosen by considering all the possible binary splits and choosing the split that maximizes the change in Gini impurity from parent to children. The goodness of split measure is calculated as follows

$$GoS = \Delta Gini = Gini_{parent} - Gini_{child_1} - Gini_{child_2}$$

3.3 Output

A typical CART algorithm will generate much the same output as other decision tree models such as CHAID or logistic regression. For the purposes of the discussion in this paper, we will only be focusing on the following output: a tree diagram, variable importance, confusion matrix, misclassification risk, sensitivity, and specificity.

3.4 Benefits and Challenges

Due to the model only considering binary splits, CART models are often much simpler than other decision tree models. The simplicity not only improves the interpretability of the model but also reduces the computational cost. The reduced computational cost is especially important as, when handling with a large set of training data for which the CART model will generate relatively quickly saving both time and energy. The biggest limitation of the CART model is that it has a high variance and low bias, meaning that the CART model is prone to overfitting.

In short, the CART models biggest strengths are sometimes its biggest weaknesses and mainly comes from the fact that more complicated and less overfit models require large amounts of data from a population which is often not practically feasible.

4 Random Forest Model

This section is based on the paper ”Random Forests” by Leo Breiman [Bre01]

4.1 Introduction

The random forest model is an ensemble machine learning method proposed by Dr. Leo Breiman in 2001 [Bre01] who, as discussed previously, also proposed the CART model. At its core, the random forest method randomly generates a fixed number of CART trees and then aggregates the classifications into a single prediction through voting. The method can be broken down into three main steps.

1. Bagging
2. Tree Generation
3. Classification Voting

4.2 Bootstrapping and Bagging

Bootstrapping is a sampling procedure invented by Bradley Efron in which a sample is randomly resampled with replacement. That is, if we have a sample from a population, we can treat the sample as if it were a population and sample it. To illustrate this, let's suppose that we have a random sample from a population with data points,

$$x_1, x_2, ..., x_n$$

If we were to bootstrap this sample, randomly chosen data points will be resampled and added to the bootstrap sample resulting in a sample that may contain multiple of the same data point and will be of the same size as the original sample. For example, if we have a dataset

$$x_1, x_2, x_3, x_4, x_5$$

bootstrapping our hypothesised sample may result in the bootstrap sample

$$x_1, x_2, x_3, x_1, x_5$$

Notice that the data point x_4 was replaced with the data point x_1 showing that the resampling was done with replacement.

The benefit of bootstrapping is that it emulates repeated sampling from a population. This is important as gathering more data from a population reduces the amount of variance in our sample, making our estimators perform better and more accurately model the population of interest. When possible, actually sampling from the population is preferable as bootstrapping operates under the assumption that the sample is representative of the population. As such, when a sample is not representative, the bootstrapping generates samples from a different population from the target population, increasing bias. Nevertheless, bootstrapping is a powerful technique as sampling is oftentimes expensive, time consuming, and inconvenient whereas bootstrapping is the opposite. [Tra]

Bagging is shorthand for bootstrap aggregation and is a procedure in which multiple bootstrap samples are generated and then compiled into a single prediction by way of voting. In the random forest model, the training dataset is bootstrapped a total of N times where N is the number of CART trees that will be generated thus requiring N many bootstrap samples. It is important to note that each bootstrap sample is independent of all other bootstrap samples, thus leading to reduced correlation between CART trees. Furthermore, the reduced correlation helps to reduce the risk of overfitting the model to the training dataset as there is no single training dataset but multiple generated from the same theoretical population.

4.3 Tree Generation

4.3.1 Random Features

For each bootstrap sample generated, a variation of the CART algorithm will be used to build a tree. This variation of CART is the same as discussed previously with the exception that predictors will be randomly chosen at each node. Doing so decreases the correlation between trees in the forest and makes the final model more robust to outliers and noise in the population.

4.3.2 Out-of-Bag (OOB)

In order to measure the performance of the model on the training dataset, roughly a third of each bootstrap sample will be set aside and used as a testing dataset for the current forest. This testing dataset is referred to as the out of bag sample (OOB). OOB is required because if the random forest model were run on the whole training dataset, the model would near perfectly predict the classifications. This is due to the implicit assumption made in bagging that the sample is representative and can thus be treated like the target population. Since the model is estimating the population, that is the training sample, with little variance and bias, the error estimates will be near zero.

4.4 Classification Voting

After the CART models are generated, the trees will be aggregated into a single predictive model. It is important to note that this aggregated model is not a decision tree but rather parts of N many trees and thus has no visual representation unlike the CART models it is based on. Nevertheless, trees are aggregated by a voting process. The details of the voting process differ depending on whether the method is regression or classification, but since the focus of this particular project is on classification, the details for the regression voting will be omitted. For classification voting, the random forest model uses the method of plurality voting. For example, if we generate N tree classifiers with N many predicted classifications for a single test data point x_i, the aggregate classification of x_i will be the class predicted most often from the N many generated trees.

4.5 Output

The generated output of the random forest algorithm is identical to CART output with the exception of tree diagrams and the inclusion of a classification plot. Like the CART algorithm, the random forest will generate a confusion matrix along with a list of predictors in terms of importance to the model. The difference between the confusion matrix of CART and random forest is that the matrix for the random forest is generated from the last OOB sample rather than the training dataset. Unlike CART, no tree diagram will be produced as drawing each tree would not only be impractical but also computationally expensive. Moreover, visualizing a handful of trees is not beneficial as each tree is uncorrelated, meaning that behavior exhibited in a single tree is not indicative of the behavior of the population. The random forest model will also produce an additional plot referred to as the classification plot. The classification plot will plot the OOB estimate along with the estimate for each class over the number of currently generated trees. The plot is useful for determining the convergence and rate of convergence of the estimates.

4.6 Benefits and Challenges

The primary benefit of the random forest model is that, due to the randomness injected from the bagging, random feature selection, and law of large numbers, they do not overfit [Bre01]. As a result, random forests predict classifications accurately with very little bias and low variance.

The primary challenge of the random forest model is that, due to the lack of a tree diagram, interpretation ability is severely diminished. Although variable importance measures indicate which features are most important for prediction, random forests can not detail how they are important. That is, without a tree diagram, an understanding of how the predictors interact with the response is not available. Another key challenge is the computational cost of the model. In order for the error estimates to converge and the risk of overfitting to be reduced, the random forest algorithm needs to generate a large enough forest. This requirement is problematic when computational power is not readily available as the cost of computing scales with the size of the dataset, number of predictors, and number of desired trees in the forest.

5 Implementation of Models

5.1 R Programming Language

For the implementation of the CART and random forest models, we have chosen to use the programming language R, an open source statistical programming language. The primary motivation for using R is that it is open source and highly versatile making it the go to choice for many statistical fields. However, the base language R does not support either the CART model nor the random forest model and so we will be using specialized packages. The the list of packages used is; randomforest, rpart, rpart.plot, and caTools.

5.2 Data

In order to give a full demonstration of the models, an example data set must be used. For this project, we will be trying to predict the outcome of a chess match. The original dataset [J] was obtained through Kaggle, a free online database hosting website, and has a variety of variables including both players names, ELO ratings for both players, number of turns the game lasted, a full list of moves played, game outcome, and more. Since most of these variables are not useful as predictors, most were omitted and some variables were combined into a single predictor. Identifying information such as player names, date, and game ID were omitted and the players ratings were combined into a single predictor by subtracting blacks rating from whites. Moreover, since the model will only be predicting games that ended in either a black or white win, draws will be omitted. After cleaning up the data set to fit our needs, our final data set contains 19108 games played with 8 variables:

- opening ply (how many moves the opening lasted)
- first move
- number of turns
- max time for both players
- difference in rating between players
- indicator of whether or not the game was rated
- how the game ended
- winner of the game

5.3 Initialization

After loading the cleaned data into R, the next step is to initialize everything that we will need before we run each model. We start off by fixing our seed for the random number generation so that we get the same results each time we run the model (We used a seed of 100). Then, we want to split the data into our training and testing data sets as described previously. Since we have a large amount of data, we are not too concerned with the split being too skewed so we will go with a 70:30 split. The last step is to name our response variable which, as discussed previously, will be the winner of the match. Now that we have set our seed, split the data, and named the response variable, we can run each model.

5.3.1 CART Model

To run the CART model in R, model building parameters must be passed to the function. For this analysis, the model was passed the response variable and all predictors, was told to use the Gini Impurity splitting criteria, and was told to only split if the goodness of split is greater than 0.05.

5.3.2 Random Forest Model

Like the CART model, building parameters need to be passed to the random forest function. For this analysis, the model was passed the response variable and all predictors, was told to generate 500 trees, and was told to asses the importance of each predictor.

6 Results

6.1 CART Output

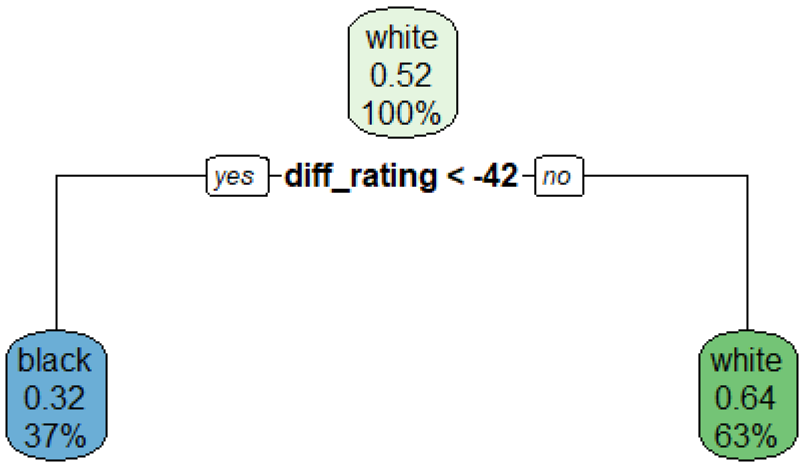
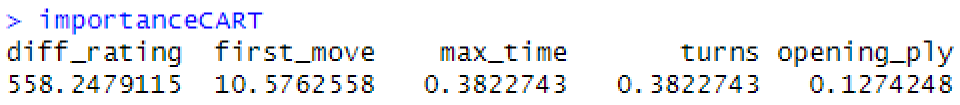


Figure 1: CART Tree Model

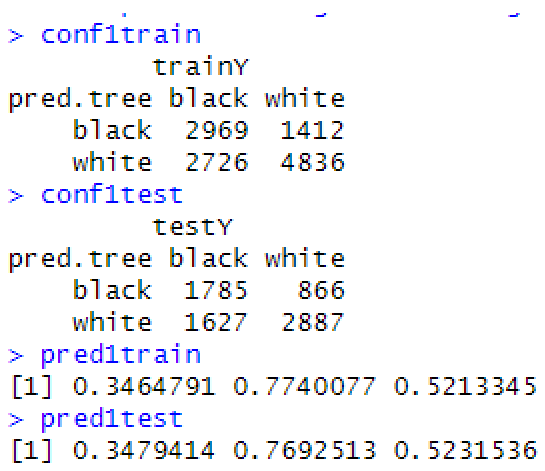
From figure 1, we can see that the CART model returned a single level tree with the split difference in ELO rating less than -42. Node #2 classifies the game as a black win, is 68% pure, and contains 37% of the total training set. Node #3 classifies the game as a white win, is 64% pure, and contains 63% of the total training set. As can be seen, the model is only single level and quite simple, giving information only on the interaction between the difference in rating and the response. From the root node, we can see that white wins 52% of games, meaning that white seems to have a slight advantage over black. This is likely due to the fact that white always makes the first move in a chess game, giving them a distinct advantage over black.

From figure 2, we can see that the list of predictors from most to least important is: difference in rating, the first move played, the max time for both players, the number of turns the game lasted, and the number of turns the opening lasted for. Furthermore, the difference in rating is by far the most important predictor with the next most important being significantly less so (558.25 vs 10.58).



```
> importanceCART
diff_rating  first_move    max_time       turns opening_ply
558.2479115  10.5762558   0.3822743   0.3822743   0.1274248
```

Figure 2: CART Variable Importance



```
> conf1train
          trainY
pred.tree black white
    black  2969  1412
    white  2726  4836
> conf1test
          testY
pred.tree black white
    black  1785   866
    white  1627  2887
> pred1train
[1] 0.3464791 0.7740077 0.5213345
> pred1test
[1] 0.3479414 0.7692513 0.5231536
```

Figure 3: CART Confusion Matrices and Prediction Measures

From figure 3 we can see that our misclassification risks for the training and testing data sets are 34.65% and 34.80%. These two values are very close showing no indication of overfitting. The values themselves, while somewhat high, are reasonable for the data and are a significant improvement from the 48% risk we would have had if there were no predictive model. Moreover, the sensitivity and specificity are similar between the sets (77.40% vs 76.93%)(52.13% vs 52.31%) further indicating that overfitting is not an issue. From the values, the sensitivity is significantly higher than the specificity, showing that the model is better at predicting white wins than black. This is likely due to the fact that we are classifying the majority of the data as a white win (63%) and white wins the majority of games by default (52%) due to the advantage of having the first move.

6.2 Random Forest Output

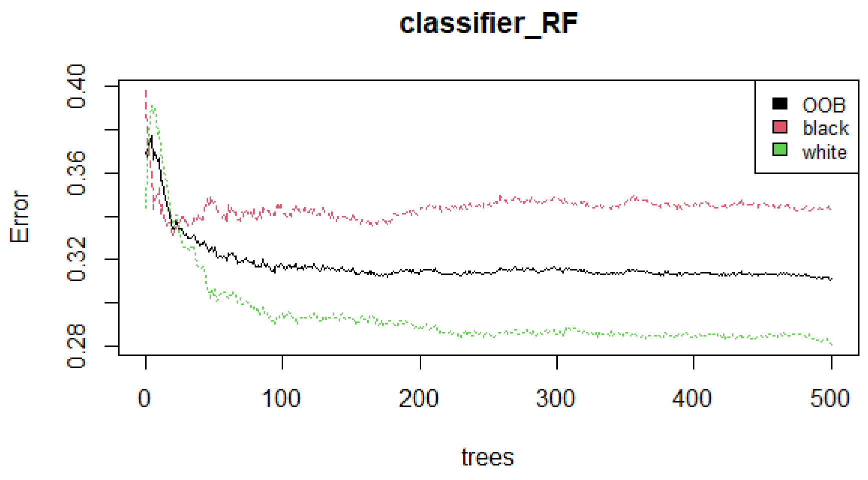
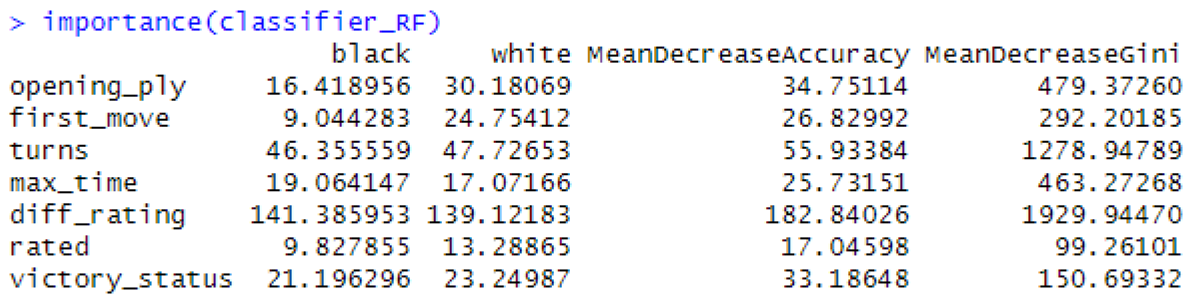


Figure 4: Classification Error Plot

Figure 4 shows the classification error plot for the generated random forest model. As can be seen in the plot, as the number of trees grows, the errors estimates converge, as required. Looking at the plot, the errors appear to stabilize around the 100 tree mark indicating that we may have generated more trees (500) than required. However, as the error estimates tend to overestimate the actual error and thus going past the convergence point is necessary [Bre01]. Nonetheless, the plot indicates that the number of trees generated was more than necessary and can be reduced ,with caution, if desired.



```
> importance(classifier_RF)
                   black     white MeanDecreaseAccuracy MeanDecreaseGini
opening_ply    16.418956  30.18069             34.75114        479.37260
first_move      9.044283  24.75412             26.82992        292.20185
turns          46.355559  47.72653             55.93384       1278.94789
max_time       19.064147  17.07166             25.73151        463.27268
diff_rating   141.385953 139.12183            182.84026       1929.94470
rated           9.827855  13.28865             17.04598         99.26101
victory_status 21.196296  23.24987             33.18648        150.69332
```

Figure 5: Random Forest Variable Importance

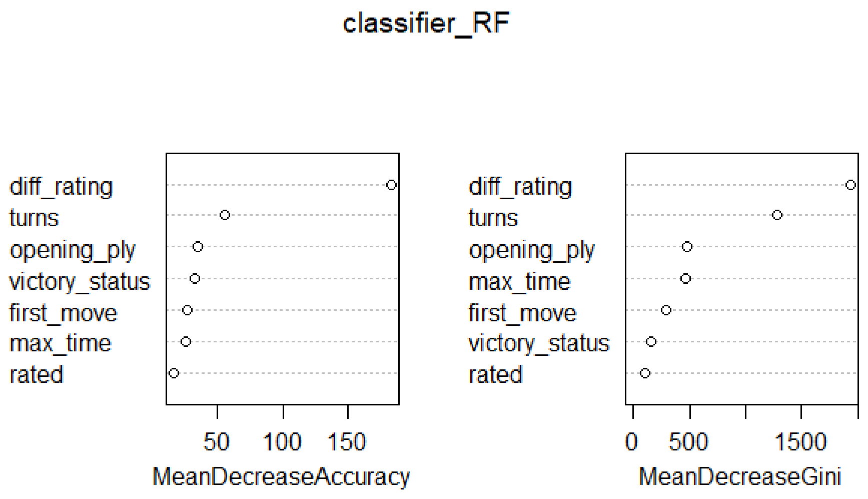
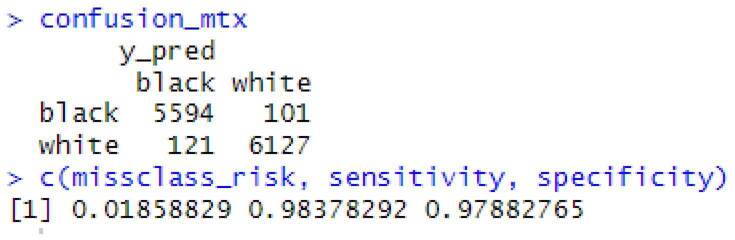


Figure 6: Random Forest Variable Importance Plot

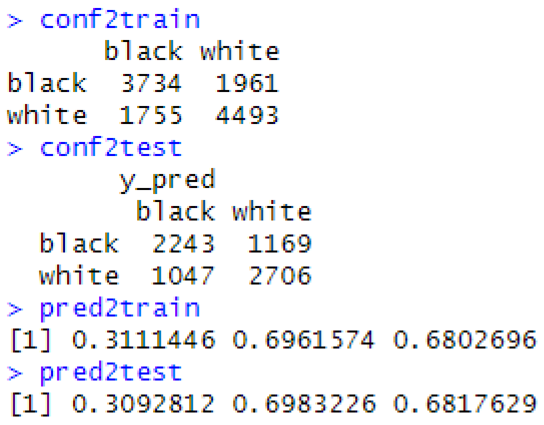
Figures 5 and 6 both show the variable importance for the model. Figure 5 shows the values for the importance and figure 6 is a visualization of figure 5. Since the CART model only relied on Gini impurity, we will only be discussing the Gini importance. From these figures we can see that the three most important variables for the model are: difference in player rating, number of turns the game lasted, number of turns the opening lasted for. Moreover, from figure 5 we can see that the number of turns the opening lasted for and the first move played were significantly more important for white than for black. This is further evidence to suggest that white having the first move is important as both opening.ply and first.move relate to the start of the game where white has the temporal advantage.



```
> confusion_mtx
       y_pred
        black white
  black  5594   101
  white   121  6127
> c(missclass_risk, sensitivity, specificity)
[1] 0.01858829 0.98378292 0.97882765
```

Figure 7: Necessity of OOB Demonstration

Figure 7 is a demonstration of the concept discussed previously in section 4.3.2 that the random forest model will near perfectly predict the training dataset. From the figure, we can see that the misclassification risk is nearly zero and that both the sensitivity and specificity are nearly one. All three measures indicate that the model is almost perfectly predicting the training set, illustrating the need for the OOB samples.

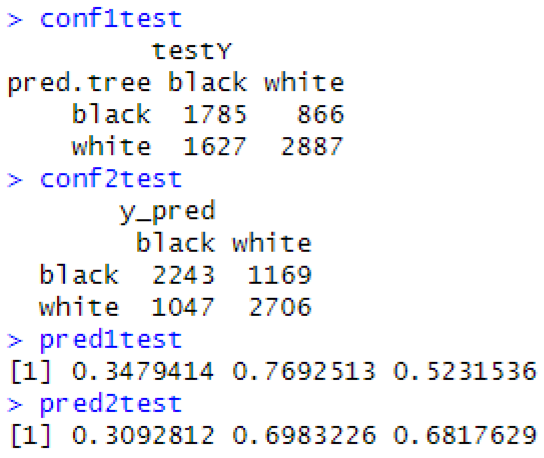


```
> conf2train
      black white
black  3734  1961
white  1755  4493
> conf2test
       y_pred
        black white
  black  2243  1169
  white  1047  2706
> pred2train
[1] 0.3111446 0.6961574 0.6802696
> pred2test
[1] 0.3092812 0.6983226 0.6817629
```

Figure 8: Random Forest Confusion Matrices and Prediction Measures

From figure 8, we can see that the misclassification risks, sensitivity, and specificity for the final OOB and testing sets are nearly identical (31.11% vs 30.93%)(69.62% vs 69.83%)(68.03% vs 68.18%) indicating that there are no problems with overfitting. Looking at the values of the estimates, the misclassification risk is relatively small and the sensitivity/specificity are relatively high (30.93%, 69.83%, 68.18%). Moreover, there is not a significant difference between the sensitivity and the specificity (69.83% vs 68.18%) indicating that the model is predicting both classes equally.

6.3 Comparison of Output



```
> conf1test
          testY
pred.tree black white
    black  1785   866
    white  1627  2887
> conf2test
        y_pred
         black white
  black  2243  1169
  white  1047  2706
> pred1test
[1] 0.3479414 0.7692513 0.5231536
> pred2test
[1] 0.3092812 0.6983226 0.6817629
```

Figure 9: Comparison of Predictions

Figure 9 compares the predictions of the CART and random forest models by combining figures 3 and 8. From the comparison, we can see that the random forest model yielded better predictions than the CART model. Looking at the misclassification risks, the CART model had a somewhat higher risk than the random forest model (34.79% vs 30.93%) showing that the random forest predicted wins more accurately. Although the difference in accuracy is somewhat small (roughly 4% improvement), in the context of a chess game the difference is significant. Being able to predict games with even just 4% more accuracy leads to many more games being correctly predicted in the extremely large target population (over 10^{120} possible chess games).

The specificity of CART was lower than the random forest (52.32% vs 68.18%) whereas the sensitivity of CART was higher than the random forest (76.93% vs 69.83%). Therefore, the CART model is better at predicting wins for white whereas the random forest model is evenly split between white and black.

7 Conclusion

Through both theory and practice, we have seen that the random forest model is a powerful predictive tool for classifications which offers many improvements over similar predictive models with a few critical limitations. The key improvement of the random forest model is the reduced risk of overfitting. Due to the injection of randomness from bagging and random feature selection, the model will not rely on any one training set, reducing the risk the overfitting by decreasing the variability of the predictions. As such, random forests tend to yield more accurate predictions when compared to other tree models. However, random forests come with two major limitations, computational cost and reduced interpretation ability. Because random forests require large sets of data with many generate trees, the algorithm can be very computationally expensive. This is a major restriction as the cost scales with the size of the dataset and number of desired trees, parameters which should be maximized whenever possible for better predictions. For example, in the process of running the random forest model with 500 trees on the chess game dataset of 19108 observations, the algorithm took over 30 seconds to run and required over a gigabyte of RAM. When testing the algorithm for a larger number of trees (2000) these requirements increased to multiple minutes of computation and over four gigabytes of available RAM. Additionally, the reduced interpretation ability is a major limitation as analysts often want to understand the behavior of data. While the random forest model provides information about the importance of predictors, the nature of how the predictors interact with the response is not attainable. In contexts where detecting these interactions are vital, the random forest model significantly under performs compared to simpler, non-ensemble methods. In the context of the chess game analysis, this limitation as not as noticeable due to the simplicity of the generated CART model. Even so, there was a notable distinction in the interpretability of the CART model versus the random forest model. Should the CART model have been more complex, this discrepancy would have been much more apparent.

8 Appendix

8.1 R Code

```
rm(list = ls())
graphics.off()

# ------------------------------------------------------------------

# Loading packages
library(caTools)
library(randomForest)
library(rpart)
library(rpart.plot)

# Load Data
D = read.csv("lichessgames.csv")
nrow(D)
```

```
# set seed
set.seed(100)

# Splitting data in train and test data
split <- sample.split(D, SplitRatio = 0.7)

train <- subset(D, split == "TRUE")
nrow(train)
test <- subset(D, split == "FALSE")
nrow(test)

# Responce variables
trainY = as.factor(train$winner)
testY = as.factor(test$winner)

# ------------------------------------------------------------------

# CART Method
# ------------------------------------------------------------------

# Fit CART
fit.tree = rpart(winner ~ ., data=train,
                 method = "class",
                 parms = list(split = "gini"),
                 cp = 0.05)
fit.tree

# Plot CART tree
rpart.plot(fit.tree)

# Variable Importance
importanceCART = sort(fit.tree$variable.importance,decreasing=TRUE)
importanceCART

# Predict Training Data
pred.tree = predict(fit.tree, train, type = "class")
confusion_mtx = table(pred.tree,trainY)
missclass_risk = (sum(confusion_mtx)-sum(diag(confusion_mtx)))/sum(confusion_mtx)
sensitivity = confusion_mtx[2,2]/sum(confusion_mtx[,2])
specificity = confusion_mtx[1,1]/sum(confusion_mtx[,1])

conf1train = confusion_mtx
pred1train = c(missclass_risk, sensitivity, specificity)

# Predict Testing Data
pred.tree = predict(fit.tree, test, type = "class")
```

```
confusion_mtx = table(pred.tree,testY)
missclass_risk = (sum(confusion_mtx)-sum(diag(confusion_mtx)))/sum(confusion_mtx)
sensitivity = confusion_mtx[2,2]/sum(confusion_mtx[,2])
specificity = confusion_mtx[1,1]/sum(confusion_mtx[,1])

conf1test = confusion_mtx
pred1test = c(missclass_risk, sensitivity, specificity)

# Compare Training and Testing
conf1train
conf1test

pred1train
pred1test

# -------------------------------------------------------------------

# Random Forest Method
# -------------------------------------------------------------------
# Fitting Random Forest to the train dataset
classifier_RF = randomForest(x = train[-8],
                             y = trainY,
                             ntree = 500,
                             importance = TRUE)

classifier_RF

# Plotting model
plot(classifier_RF)
legend("topright", colnames(classifier_RF$err.rate),col=1:4,cex=0.8,fill=1:4)

# Importance plot
importance(classifier_RF)

# Variable importance plot
varImpPlot(classifier_RF)

# Training data set almost perfectly predicts training data set
# Predicting the Train set results
y_pred = predict(classifier_RF, newdata = train[-8])

# Confusion Matrix
confusion_mtx = table(train[, 8], y_pred)
missclass_risk = (sum(confusion_mtx)-sum(diag(confusion_mtx)))/sum(confusion_mtx)
sensitivity = confusion_mtx[2,2]/sum(confusion_mtx[,2])
specificity = confusion_mtx[1,1]/sum(confusion_mtx[,1])
```

```
confusion_mtx
c(missclass_risk, sensitivity, specificity)

# Predict "Training" Data
confusion_mtx = classifier_RF$confusion[,-3]
missclass_risk = (sum(confusion_mtx)-sum(diag(confusion_mtx)))/sum(confusion_mtx)
sensitivity = confusion_mtx[2,2]/sum(confusion_mtx[,2])
specificity = confusion_mtx[1,1]/sum(confusion_mtx[,1])

conf2train = confusion_mtx
pred2train = c(missclass_risk, sensitivity, specificity)

# Predict Testing Data
y_pred = predict(classifier_RF, newdata = test[-8])
confusion_mtx = table(test[, 8], y_pred)
missclass_risk = (sum(confusion_mtx)-sum(diag(confusion_mtx)))/sum(confusion_mtx)
sensitivity = confusion_mtx[2,2]/sum(confusion_mtx[,2])
specificity = confusion_mtx[1,1]/sum(confusion_mtx[,1])

conf2test = confusion_mtx
pred2test = c(missclass_risk, sensitivity, specificity)

conf2train
conf2test

pred2train
pred2test

# -------------------------------------------------------------------

# Compare CART and Forest

conf1test
conf2test

pred1test
pred2test
```

9 References

All References

[Bre94] Leo Breiman. “Bagging Predictors”. In: *University of California* (1994), pp. 1–20.

[Bre01] Leo Breiman. “Random Forests”. In: *University of California* (2001), pp. 1–33.

[Fri23] Mark Fridline. *Applied Analytic Decision Trees*. Department of Statistics. University of Akron, 2023.

[Geea] GeeksforGeeks. *Bias and Variance in Machine Learning*. URL: `https://www.geeksforgeeks.org/bias-vs-variance-in-machine-learning/`. (accessed: 04.25.24).

[Geeb] GeeksforGeeks. *CART (Classification And Regression Tree)*. URL: `https://www.geeksforgeeks.org/cart-classification-and-regression-tree-in-machine-learning/`. (accessed: 04.25.24).

[Geec] GeeksforGeeks. *Cross Validation in Machine Learning*. URL: `https://www.geeksforgeeks.org/cross-validation-machine-learning/`. (accessed: 04.25.24).

[Geed] GeeksforGeeks. *Random Forest Approach in R Programming*. URL: `https://www.geeksforgeeks.org/random-forest-approach-in-r-programming/`. (accessed: 04.25.24).

[Gui] Carmelia Guild. *Classification and Regression Trees (CART) in R*. URL: `https://rpubs.com/camguild/803096`. (accessed: 04.25.24).

[IBMa] IBM. *What is a Decision Tree?* URL: `https://www.ibm.com/topics/decision-trees`. (accessed: 04.25.24).

[IBMb] IBM. *What is bagging?* URL: `https://www.ibm.com/topics/bagging`. (accessed: 04.25.24).

[IBMc] IBM. *What is random forest?* URL: `https://www.ibm.com/topics/random-forest`. (accessed: 04.25.24).

[J] Mitchel J. *Chess Game Dataset (Lichess)*. URL: `https://www.kaggle.com/datasets/datasnaek/chess`. (accessed: 04.25.24).

[Tra] Jack Trainer. *Bootstrapping in Statistics*. URL: `https://www.lancaster.ac.uk/stor-i-student-sites/jack-trainer/bootstrapping-in-statistics/`. (accessed: 04.25.24).

CART References

[Fri23] Mark Fridline. *Applied Analytic Decision Trees*. Department of Statistics. University of Akron, 2023.

[Geeb] GeeksforGeeks. *CART (Classification And Regression Tree)*. URL: `https://www.geeksforgeeks.org/cart-classification-and-regression-tree-in-machine-learning/`. (accessed: 04.25.24).

Random Forest References

[Bre94] Leo Breiman. “Bagging Predictors”. In: *University of California* (1994), pp. 1–20.

[Bre01] Leo Breiman. "Random Forests". In: *University of California* (2001), pp. 1–33.

[IBMb] IBM. *What is bagging?* URL: `https://www.ibm.com/topics/bagging`. (accessed: 04.25.24).

[IBMc] IBM. *What is random forest?* URL: `https://www.ibm.com/topics/random-forest`. (accessed: 04.25.24).

[Tra] Jack Trainer. *Bootstrapping in Statistics*. URL: `https://www.lancaster.ac.uk/stor-i-student-sites/jack-trainer/bootstrapping-in-statistics/`. (accessed: 04.25.24).

R References

[Geed] GeeksforGeeks. *Random Forest Approach in R Programming*. URL: `https://www.geeksforgeeks.org/random-forest-approach-in-r-programming/`. (accessed: 04.25.24).

[Gui] Carmelia Guild. *Classification and Regression Trees (CART) in R*. URL: `https://rpubs.com/camguild/803096`. (accessed: 04.25.24).

Data Source

[J] Mitchel J. *Chess Game Dataset (Lichess)*. URL: `https://www.kaggle.com/datasets/datasnaek/chess`. (accessed: 04.25.24).