Salary Prediction Using CART & ENSEMBLE METHODS

2023-11-05

PROJECT OVERVIEW

Adopted from ISLR: We will use the Hitters data in the ISLR2 package. The goal is to make a model to predict Salary.

First, I will apply CART to this dataset. I will check the trees before and after pruning and interpret the results, report the error rate.

```
library(ISLR2)
```

```
## Warning: package 'ISLR2' was built under R version 4.3.2
```

```
library(rpart)
```

```
## Warning: package 'rpart' was built under R version 4.3.3
```

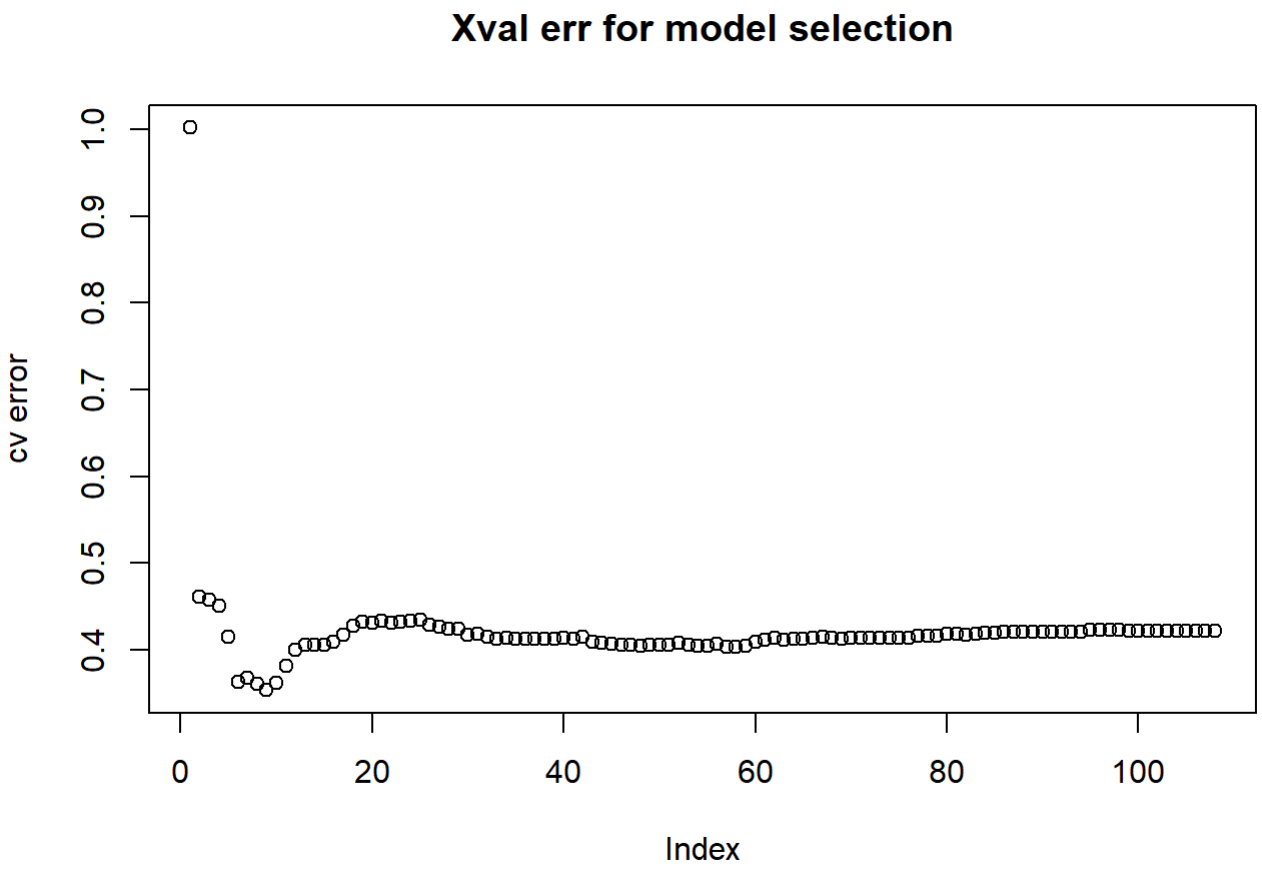
```
data("Hitters")
str(Hitters)
```

```
## 'data.frame':    322 obs. of  20 variables:
##  $ AtBat    : int  293 315 479 496 321 594 185 298 323 401 ...
##  $ Hits     : int  66 81 130 141 87 169 37 73 81 92 ...
##  $ HmRun    : int  1 7 18 20 10 4 1 0 6 17 ...
##  $ Runs     : int  30 24 66 65 39 74 23 24 26 49 ...
##  $ RBI      : int  29 38 72 78 42 51 8 24 32 66 ...
##  $ Walks    : int  14 39 76 37 30 35 21 7 8 65 ...
##  $ Years    : int  1 14 3 11 2 11 2 3 2 13 ...
##  $ CAtBat   : int  293 3449 1624 5628 396 4408 214 509 341 5206 ...
##  $ CHits    : int  66 835 457 1575 101 1133 42 108 86 1332 ...
##  $ CHmRun   : int  1 69 63 225 12 19 1 0 6 253 ...
##  $ CRuns    : int  30 321 224 828 48 501 30 41 32 784 ...
##  $ CRBI     : int  29 414 266 838 46 336 9 37 34 890 ...
##  $ CWalks   : int  14 375 263 354 33 194 24 12 8 866 ...
##  $ League   : Factor w/ 2 levels "A","N": 1 2 1 2 2 1 2 1 2 1 ...
##  $ Division : Factor w/ 2 levels "E","W": 1 2 2 1 1 2 1 2 2 1 ...
##  $ PutOuts  : int  446 632 880 200 805 282 76 121 143 0 ...
##  $ Assists  : int  33 43 82 11 40 421 127 283 290 0 ...
##  $ Errors   : int  20 10 14 3 4 25 7 9 19 0 ...
##  $ Salary   : num  NA 475 480 500 91.5 750 70 100 75 1100 ...
##  $ NewLeague: Factor w/ 2 levels "A","N": 1 2 1 2 2 1 1 1 2 1 ...
```

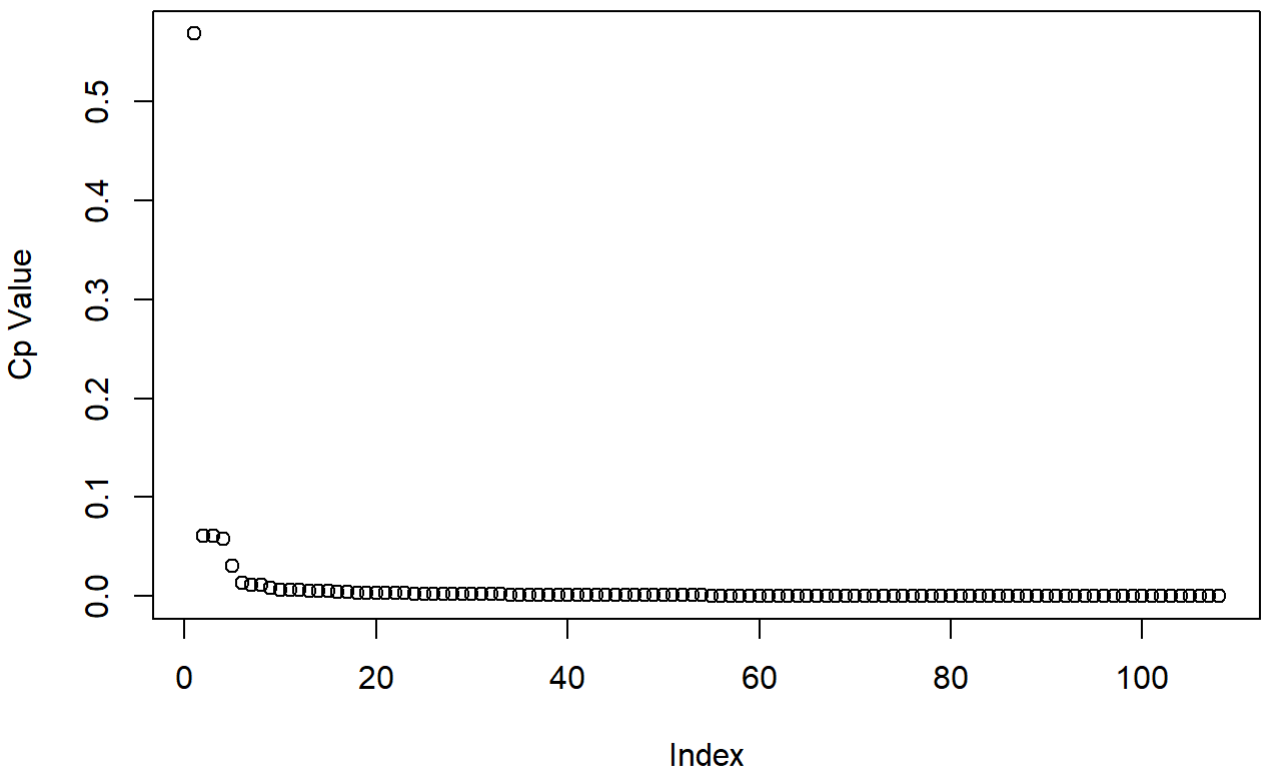
```
dim(Hitters)
```

```
## [1] 322  20
```

```
set.seed(12345)
new_Hitters <- Hitters

# Creating a new dataset without factor variables
new_Hitters <- new_Hitters[, !colnames(new_Hitters) %in% c("League", "Division", "NewLeague")]



new_Hitters <-  new_Hitters[complete.cases(new_Hitters$Salary), ]

#to remove right skew we convert salary to log salary
log_of_Salary = log(new_Hitters$Salary)

new_Hitters$Salary = log_of_Salary

colnames(new_Hitters)[colnames(new_Hitters)=="Salary"]="log_of_Salary"



model.controls <- rpart.control(minbucket = 2, minsplit = 4, xval = 10, cp = 0)
fit <- rpart(new_Hitters$log_of_Salary~., data = new_Hitters, control = model.controls)


plot(fit$cptable[,4], main = "Xval err for model selection", ylab = "cv error")
```

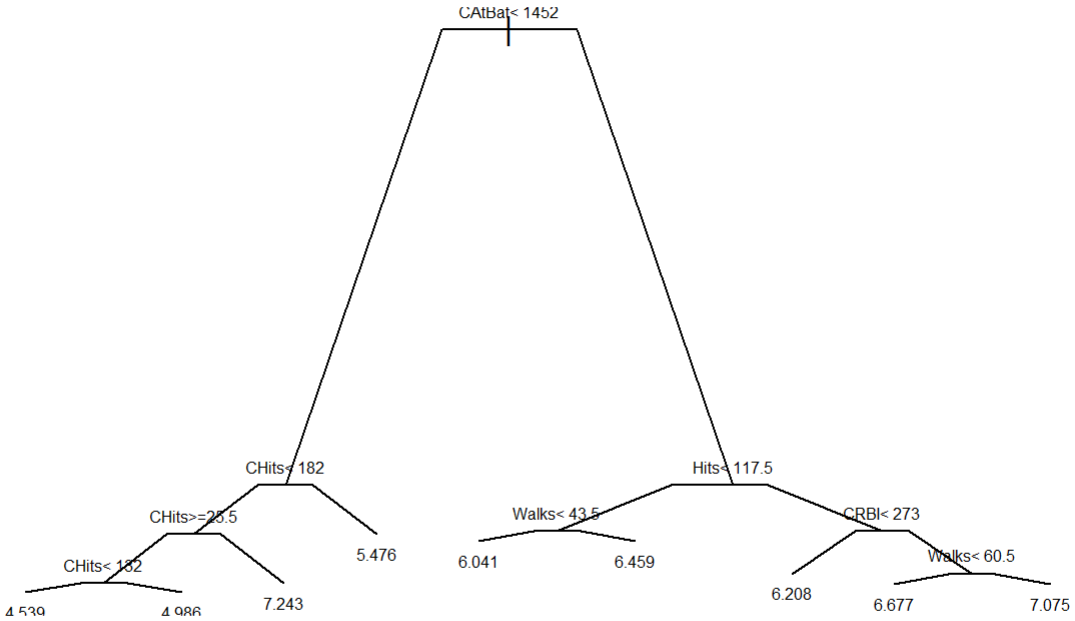



```
plot(fit$cptable[,1], main = "Cp for model selection", ylab = "Cp Value")
```



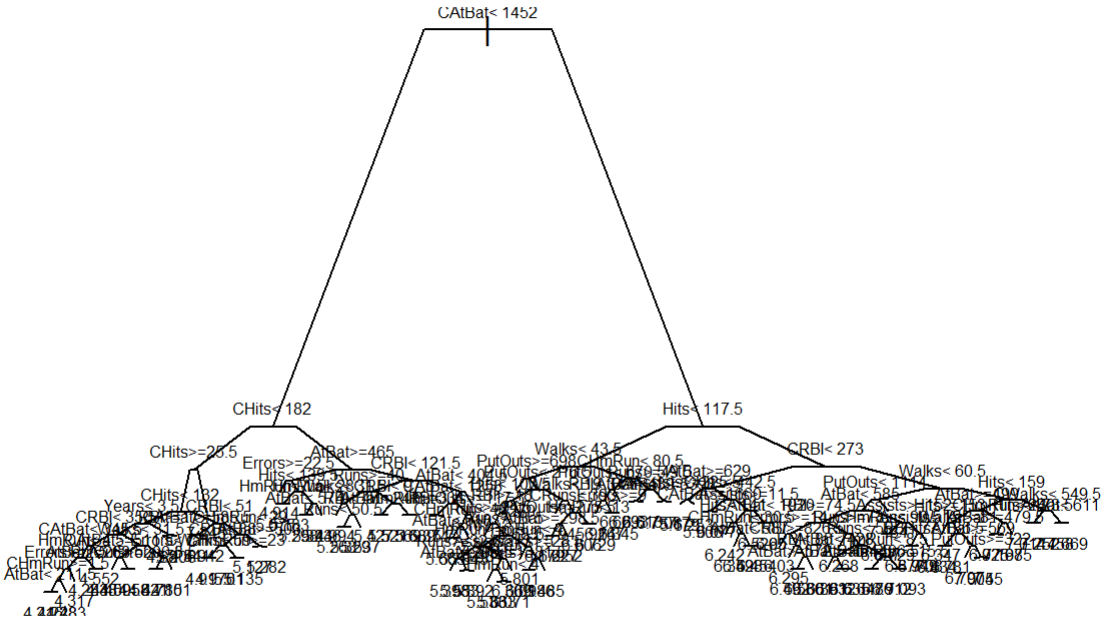

```
min_cp = which.min(fit$cptable[,4])
pruned.fit <- prune(fit, cp = fit$cptable[min_cp, 1])


plot(pruned.fit, branch = .3, compress = T, main = "Pruned Tree")
text(pruned.fit, cex = .5)
```

Pruned Tree




```
plot(fit, branch = .3, compress = T, main = "Full Tree")
text(fit, cex = .5)
```

Full Tree



```
# making a prediction
pred_fulltree <- predict(fit, newdata = new_Hitters)
pred_pruned <- predict(pruned.fit, newdata = new_Hitters)
mse_error_fulltree <- mean((pred_fulltree - new_Hitters$log_of_Salary)^2)

mse_error_pruned <- mean((pred_pruned - new_Hitters$log_of_Salary)^2)

mse_error_fulltree
```

```
## [1] 0.02135087
```

```
mse_error_pruned
```

```
## [1] 0.1449258
```

So interpreting the results, we see that the Full tree is highly detailed and is a complex decission tree. It overfits the data in the model and also captures noise which leads to poor generalization to new and unseen data. It is too specific in making prediction bases on each unique variation in the data. This is why the MSE for the Full tree is less than that of pruned tree.

Whereas, the pruned tree is a more simplified and concise as compared to the Full tree. The branches which does not have much significance are removed. This makes it easier to interpret the model and also the chances of overfitting is reduced. This is why the MSE for the pruned tree is more than that of the Full tree as it is more concise rather than being precise.

Now comparing the MSEs: The MSE for the unpruned tree (0.02135087) is relatively low, suggesting that it has a good fit to the training data. However, there is a risk that it may not generalize well to new, unseen data due to its complexity.

The MSE for the pruned tree (0.1449258) is higher than that of the unpruned tree, indicating that the pruned tree may not fit the training data as closely. However, this reduced complexity may result in better generalization and predictive performance on new data.”

In conclusion the pruned tree is likely a better choice for making predictions.

Now, I will apply bagging to this dataset and report the error rate.

```
library(rpart)
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.3.2
```

```
## randomForest 4.7-1.1
```

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

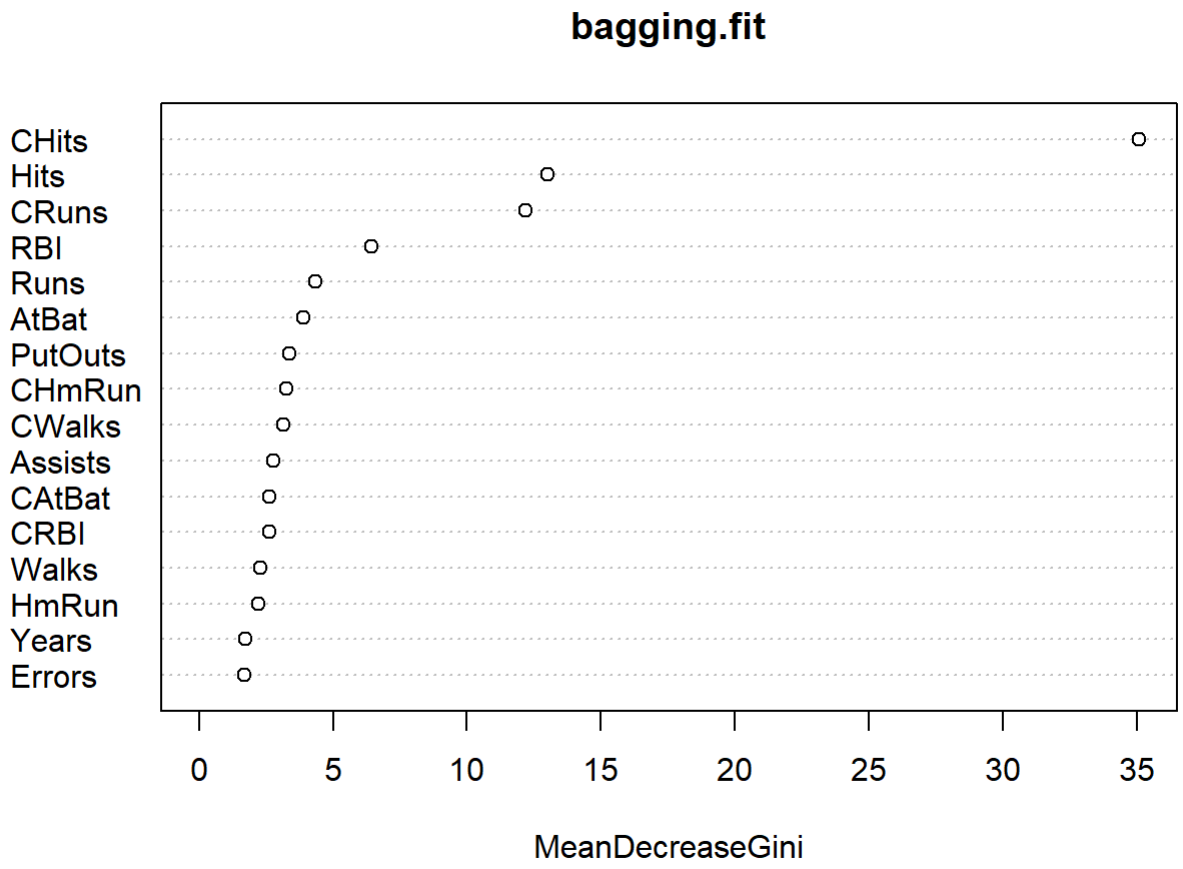
```
library(ISLR2)


mean(Hitters$Salary)
```

```
## [1] NA
```

```
N_Hitters <- Hitters

# Creating a new dataset without factor variables
Hitters1 <- N_Hitters[, !colnames(N_Hitters) %in% c("League", "Division", "NewLeague")]



Hitters2 <-  Hitters1[complete.cases(Hitters1$Salary), ]

which(names(Hitters2) == "Salary")
```

```
## [1] 17
```

```
High <- ifelse(Hitters2$Salary<=535, "No", "Yes")
New_Hitters <- data.frame(Hitters2[,-17], High)

set.seed(12345)
test_indis <- sample(1:nrow(New_Hitters), .20*nrow(New_Hitters))
test <- New_Hitters[test_indis, ]
training <- New_Hitters[-test_indis, ]

test$High <- as.factor(test$High)
y_true <- as.numeric(test$High)-1
training$High <- as.factor(training$High)

bagging.fit <- randomForest(High~., data = training, n.tree = 1000, mtry = 16)

varImpPlot(bagging.fit)
```



```
importance(bagging.fit)
```

```
##         MeanDecreaseGini
## AtBat           3.913952
## Hits           13.017617
## HmRun           2.228971
## Runs            4.355525
## RBI             6.427871
## Walks           2.275149
## Years           1.741391
## CAtBat          2.642867
## CHits          35.100603
## CHmRun          3.257626
## CRuns          12.177700
## CRBI            2.641634
## CWalks          3.155612
## PutOuts         3.356391
## Assists         2.771706
## Errors          1.707518
```

```
y_hat <- predict(bagging.fit, newdata = test, type = "response")
y_hat <- as.numeric(y_hat)-1
misclass_bag <- sum(abs(y_true- y_hat))/length(y_hat)
misclass_bag
```

```
## [1] 0.2115385
```

Therefore, the missclassification rate is 0.2115385.

Here,I will create a plot displaying the test error resulting from random forests on this data set for a more comprehensive range of values for mtry and ntree.

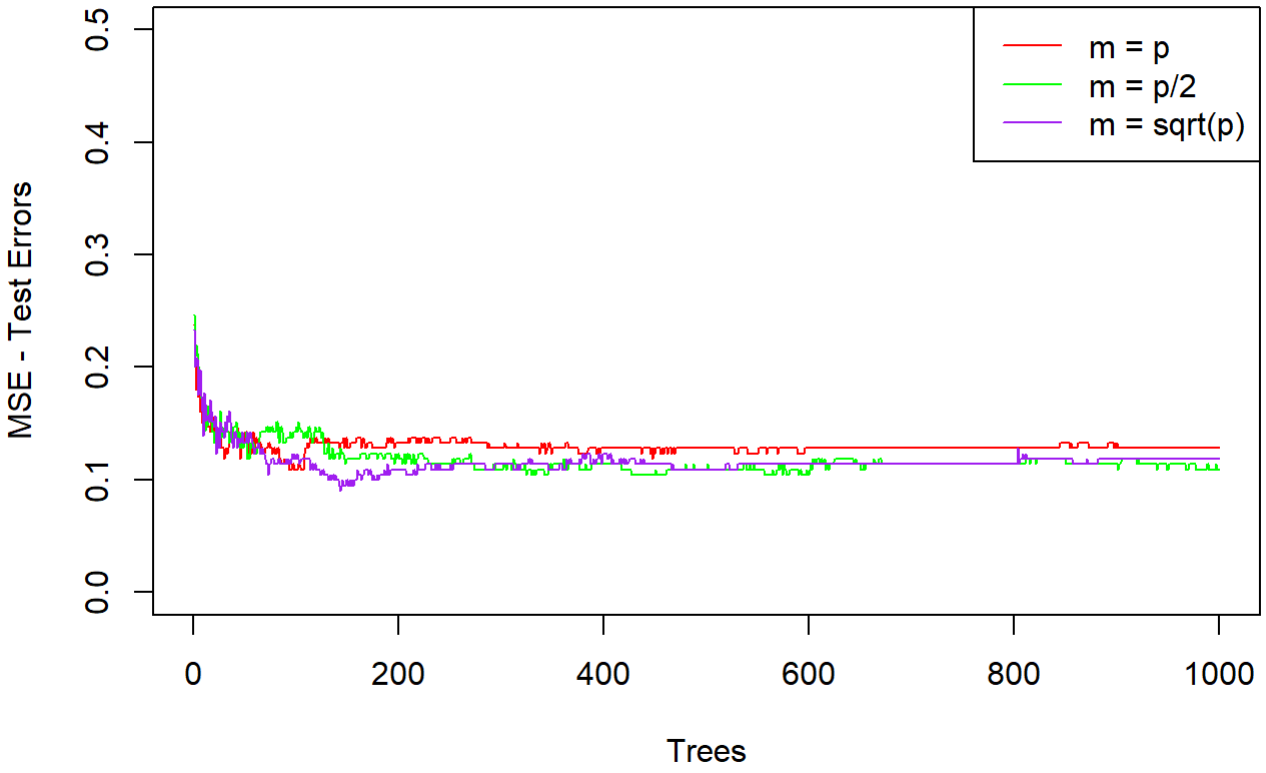
```
library(randomForest)
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.3.3
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:randomForest':
##
##     margin
```

```r
#Total columns in New_Hitters is 17 so p = 16
p <- 16


N_Hitters <- Hitters


Hitters1 <- N_Hitters[, !colnames(N_Hitters) %in% c("League", "Division", "NewLeague")]

Hitters2 <- Hitters1[complete.cases(Hitters1$Salary), ]

High <- ifelse(Hitters2$Salary <= 535, "No", "Yes")
New_Hitters <- data.frame(Hitters2[, -17], High)

set.seed(12345)
test_indis <- sample(1:nrow(New_Hitters), 0.20 * nrow(New_Hitters))
test <- New_Hitters[test_indis, ]
training <- New_Hitters[-test_indis, ]

test$High <- as.factor(test$High)
y_true <- as.numeric(test$High) - 1
training$High <- as.factor(training$High)

#considering n.tree = 1000

# Random Forest models
rf_model1 <- randomForest(High ~ ., data = training, ntree = 1000, mtry = ncol(training) - 1)
rf_model2 <- randomForest(High ~ ., data = training, ntree = 1000, mtry = (ncol(training) - 1) / 2)
rf_model3 <- randomForest(High ~ ., data = training, ntree = 1000, mtry = sqrt(ncol(training) - 1))


mse_error1 <- rf_model1$err.rate[, "OOB"]
mse_error2 <- rf_model2$err.rate[, "OOB"]
mse_error3 <- rf_model3$err.rate[, "OOB"]




plot(1:1000, mse_error1, col = "red", type = "l", xlab = "Trees", ylab = "MSE - Test Errors", ylim = c
(0,0.5))
lines(1:1000, mse_error2, col = "green", type = "l")
lines(1:1000, mse_error3, col = "purple", type = "l")

legend("topright", c("m = p", "m = p/2", "m = sqrt(p)"), col = c("red", "green", "purple"), cex = 1, lt
y = 1)
```



The test error is portrayed as a function of the number of trees in the contour plot. On the plot, the x-axis denotes the number of trees (ntree), and the y-axis represents the count of predictors available for splitting at each inner tree node or our (mtry).

We made a comparison for mtry = p/2 where p = 16, representing half the number of predictors. This is evaluated against other mtry values like mtry = p and mtry = √p to ascertain whether utilizing half of the predictors for splitting leads to improved or diminished performance compared to alternative mtry values.

Each distinctive contour line on the plot corresponds to a different configuration of mtry and ntree, with the line color indicating the associated test error. Here m = p is red, m = p/2 is green and √p is purple

We see that the Random forests for m = p/2 and m = √p exhibit a marginal enhancement over bagging (m = p) after a certain number of n.trees. This can be seen in regions of the plot where reducing the number of predictors available for splitting (mtry) results in lower test errors as compared to scenarios where all predictors are utilized (mtry = p), supporting the idea that a subset of predictors can outperform using all predictors.

We also see that the, contour lines associated with m = √p (square root of the number of predictors) show a modest improvement in test error over bagging with m = p, suggesting that using a smaller subset of predictors can be beneficial.

Now, similar to bagging, we know that random forests exhibit a lack of overfitting when the number of trees (ntree) is increased. In practice, a sufficiently large value for the number of trees is chosen to allow the error rate to settle down, indicating that further increases in the number of trees may not significantly impact performance.

Now, applying boosting to this data using different shrinkage parameters. Then, I will tune the model and report the test error.

```
set.seed(123)
library(gbm)
```

```
## Warning: package 'gbm' was built under R version 4.3.3
```

```
## Loaded gbm 2.1.9
```

```
## This version of gbm is no longer under development. Consider transitioning to gbm3, https://github.com/gbm-developers/gbm3
```

```
train <- training;
train$High <- as.numeric(training$High)-1
testing <- test;
testing$High <- as.numeric(test$High)-1
y_true <- as.numeric(test$High) - 1

shrink <- c(.1, .4, .6, .8)
iteration <- 1000
error <- c()
for (i in 1:length(shrink)){
    fit <- gbm(High~., data = train, n.trees = iteration, shrinkage = shrink[i], interaction.depth = 3, distribution = "adaboost")
    temp <- c()
    for (j in 1:iteration){
        y_hat <- predict(fit, newdata = testing, n.trees = j, type =        "response")
        misclass <- sum(abs(y_true - y_hat))/length(y_hat)
        temp <- c(temp, misclass)
    }
    error <- cbind(error, temp) # max_iter x length(shrink)
}

colnames(error) <- paste("shrinkage", shrink, sep = ":")

error[1000,]
```

```
## shrinkage:0.1 shrinkage:0.4 shrinkage:0.6 shrinkage:0.8
##     0.2436379     0.2619358     0.2504945     0.2347090
```

```
plot(error[,1], type = "l", main = "Error Profiles", ylab = "error", xlab = "boosting iterations", ylim = c(.07, .5))
lines(error[,2], col = "red")
lines(error[,3], col = "blue")
lines(error[,4], col = "green")
```

#Therefore,

the test errors with each shrinkage parameters is as follows: #shrinkage:0.1 shrinkage:0.4 shrinkage:0.6 shrinkage:0.8 # 0.2436379 0.2619358 0.2504945 0.2347090

Data Preparation

I began by preprocessing the Hitters dataset. The dataset contains several categorical variables such as “League,” “Division,” and “NewLeague,” which I excluded to focus on numerical predictors. Additionally, I removed any records with missing salary information to ensure that the dataset was complete. To address the right skew in the salary distribution, I applied a logarithmic transformation to the Salary variable, which I renamed as “log_of_Salary.” This transformation helped stabilize the variance and improve the model’s predictive accuracy.

Applying CART (Classification and Regression Trees)

Building the Full Tree

I first applied the CART algorithm to the dataset to build a decision tree model. The tree was constructed using the rpart package in R, with specific control parameters to avoid early stopping. I plotted the complexity parameter (Cp) and cross-validation error to determine the optimal tree size. The full tree, as expected, was highly detailed, capturing even minor variations in the data, which led to overfitting. This overfitting was reflected in the mean squared error (MSE) on the training data.

Pruning the Tree

To mitigate overfitting, I pruned the tree by selecting the optimal Cp value corresponding to the minimum cross-validation error. The pruned tree was simpler and less complex than the full tree, which made it easier to interpret. Although the MSE of the pruned tree was higher than that of the full tree, this trade-off suggested that the pruned tree would generalize better

to new, unseen data.

Applying Bagging

Next, I applied the Bagging technique using the randomForest package in R. Bagging involves generating multiple bootstrap samples from the training data and fitting a decision tree to each sample. The predictions from these trees are then averaged to produce a final prediction. I divided the dataset into training and test sets and built a Bagging model using 1000 trees. The model's performance was evaluated by calculating the misclassification rate on the test set, which was found to be 0.2115. This result indicated that Bagging effectively reduced overfitting and variance compared to the single CART model.

Applying Random Forests

Exploring Different mtry Values Building on the Bagging technique, I applied Random Forests, which introduces an additional layer of randomness by selecting a random subset of predictors (mtry) at each split in the tree. I experimented with three different mtry values: $m=p$(using all predictors), $m=p/2$ (using half of the predictors), and $m=p$(using the square root of the number of predictors). I constructed three Random Forest models with 1000 trees each and compared their performance by plotting the out-of-bag (OOB) error as a function of the number of trees.

COMPARISONS AND CONTRAST

A) CART vs. Bagging vs. Random Forests:

CART Full Tree vs. Pruned Tree:

The full tree in CART (Classification and Regression Trees) showed a low mean squared error (mse_error_fulltree) of 0.02135087, indicating a risk of overfitting. Pruning the tree led to a higher mean squared error (mse_error_pruned) of 0.1449258, suggesting a balance between complexity and predictive accuracy.

Bagging:

Bagging exhibited a misclassification rate of 0.2115385, indicating a reduction in overfitting and variance. However, some misclassification remains.

Random Forests:

A comprehensive exploration of the test error landscape revealed insights into the impact of the number of trees and predictor subsets.

Random forests with m = p/2 and m = $\sqrt{p}$ showed improvement over bagging in specific regions, supporting the effectiveness of utilizing a subset of predictors. The stability of the error rate with an increasing number of trees aligns with the expected behavior of random forests, avoiding overfitting.

Random Forests vs. Boosting:

Random Forests:

Random forests built multiple trees concurrently, leveraging the power of ensembles and introducing randomness for improved generalization.

Boosting:

Boosting, with different shrinkage parameters, sequentially built trees, focusing on rectifying the weaknesses of prior models. Larger shrinkage values resulted in lower test errors, emphasizing the importance of a slower learning rate in boosting.

In conclusion, we can say that the choice between CART, bagging, random forests, and boosting depends on the nuances of our dataset. While bagging and random forests collectively address overfitting, the personalized exploration of test errors and misclassification rates provides valuable insights. The consideration of factors such as the number of trees, predictor subsets, and learning rates is crucial for tailoring our approach to achieve optimal predictive performance in our unique scenario.