Baseball Salary Predictor Selection using Decision Trees in R

Justin Bennett[1]

[1]School of Mathematical and Statistical Sciences

April 30, 2021

Abstract

Decision trees are a pivotal part of data science, establishing both classification systems and predicting by a target variable. One of the more common uses of decision trees is variable selection. The quantity of predictors that are routinely measured in baseball has increased dramatically with the introduction of different metrics to determine success in the sport. Many of these variables are hardly relevant, and likely should not be included in a regression process. In this paper, we look to explore which variables of the Lahman baseball data set possess the highest importance by implementing a random forest using the (randomForest) library in r, measuring their significance using the node depth. Further, we will compare variable importance and error by also implementing bagging, boosing.
Keywords: randomForest; decison tree; R; Lahman

Contents

1 Introduction

Baseball statistics became prominent in 2002, when the Oakland Athletics decided to use statistics instead of home runs and large salaries to get wins. Using Lahman's baseball data set, we aim to find the predictors that are most significant to determining a player's salary. This could be very useful in application when determining if a player is being underpaid or overpaid, and could be used as a potential bargaining tool.

The inherent variance in baseball games, such as left- vs. right-handedness, weather conditions, and umpire bias make it difficult to make reasonable predictions in individual games. However, it may be much more reasonable to make predictions of a player's value based on hitting variables e.g. runners batted in, hits, doubles, total bases, on-base percentage, etc.

There are some natural issues that occur with data, however. There is the issue of high correlation between variables, and the large amount of noise in the data. The major problem, however, lies with controlling for variables that may be somewhat consistent across entities but vary over time:

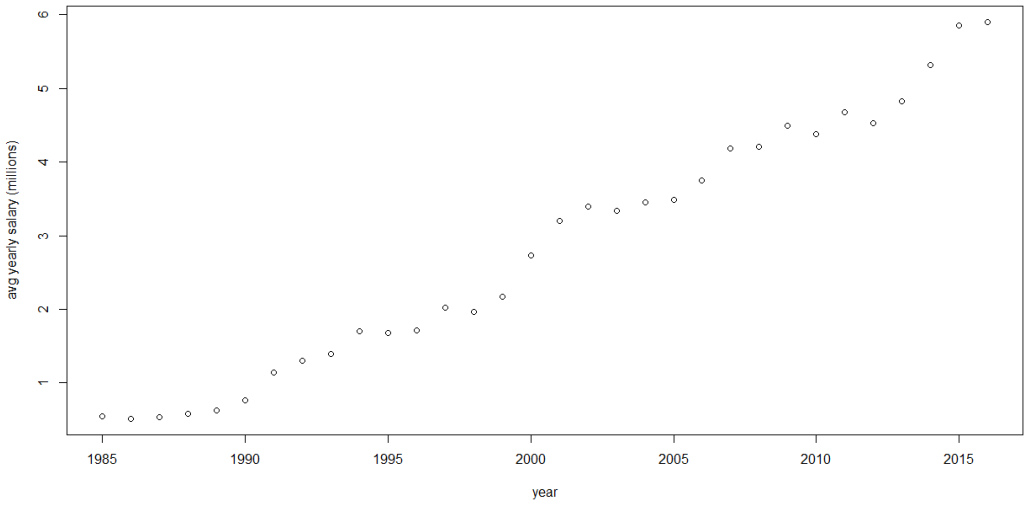

Figure 1: Time series plot of average yearly MLB player salaries

Observe, the average salary of a player in 1985 is roughly 300,000 USD while the average salary in 2016 is roughly 6,000,000 USD. There are many known ways to approach handling this issue. In this paper, we subtract the mean from the salary to by year to address this issue.

Another issue found in this paper was high correlation between some of the variables, such as hits (H) with total bases (TB), runners batted in (RBI) with runs (R), and salary with intentional base on balls (IBB), stolen bases (SB), sacrifice hits/bunts (SH).

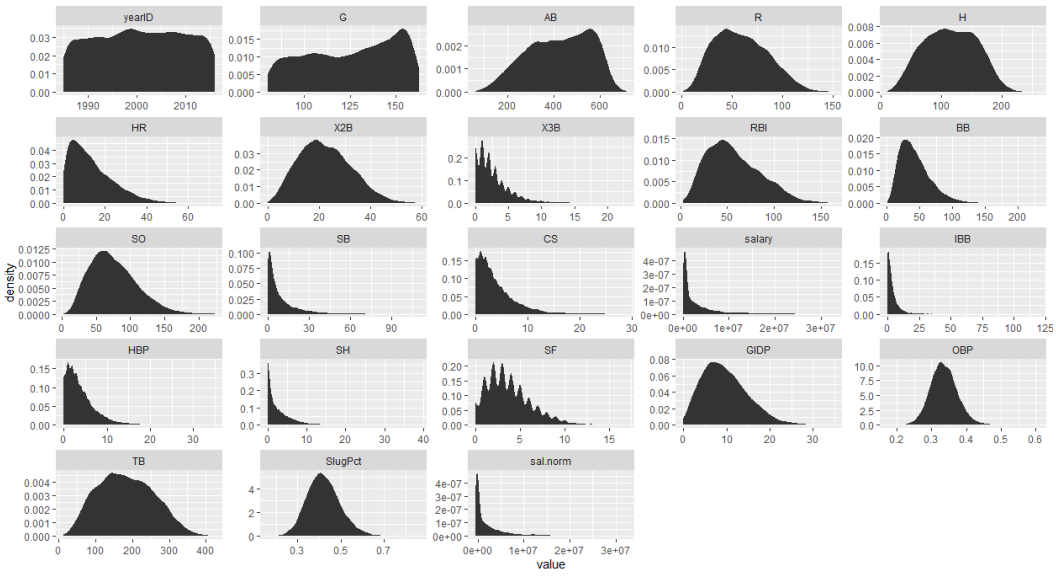

Figure 2: Density plots for predictor variables

In application, it is a common practice to remove highly correlated variables when performing regression. For this paper, we keep all variables since we are using decision trees and random forests.

In this paper, we seek to understand the value of a player through applying machine learning models, as the culmination of a year of data science training in ASU's DAT401 and DAT402.

2 Models and methodology

In this paper, we use tree-based methods for regression, stratifying the predictor space into simple, partitioned regions. It is known that Tree-based methods are simple and useful for interpretation [1], thus we choose to forego the often used predictive models desired in data science. Although predictive models are important, it should be recognized that baseball models demand power outside of numerical significance. Because of this, we find it practical to use tree-based methods to determine the important predictors of MLB player salaries. We use, comapre, and discuss major tree-based algorithms, such as random forests, bootstrap aggregation (bagging) and boosting to explore variable importance.

The data used for this analysis was Lahman's Baseball data set[2][3]. With this data, we were able to obtain a vast quantity of predictor variables and accurate statistics so that modeling could be performed.

2.1 Data

Lahman's baseball data includes many subsets, of which we chose to use *Teams, Batting, Salaries*, and *Appearances*. For more information and details of these subsets, we refer interested readers to the Lahman package description, or to [3].

Getting and cleaning the data comprised of combining select predictors from the Salaries subset (salary), *Batting* subset (G, AB, R ,H, HR, X2B, X3B, RBI, BB ,SO, SB, CS, IBB, HBP, SH, SF, GIDP, OBP, TB, SlugPct), *Teams* subset (teamID), and *Appearances* subset (Gp). The original data set had statistics from 1871 to 2016. We chose to filter the data such that we are only using player data from 1985 to 2016. The reasoning was two-fold; as a baseball fanatic, I am aware that divisional playoffs did not begin until 1985, and the egregious upward trend of average player salary by year.

Recall from the density plots that the distribution of player salary is very right-skewed. Instead of taking the log of salary, we find it useful to subtract the mean from each player's salary by year, as this alleviates both the skewness of salary, but also the discrepancy between yearly averages.

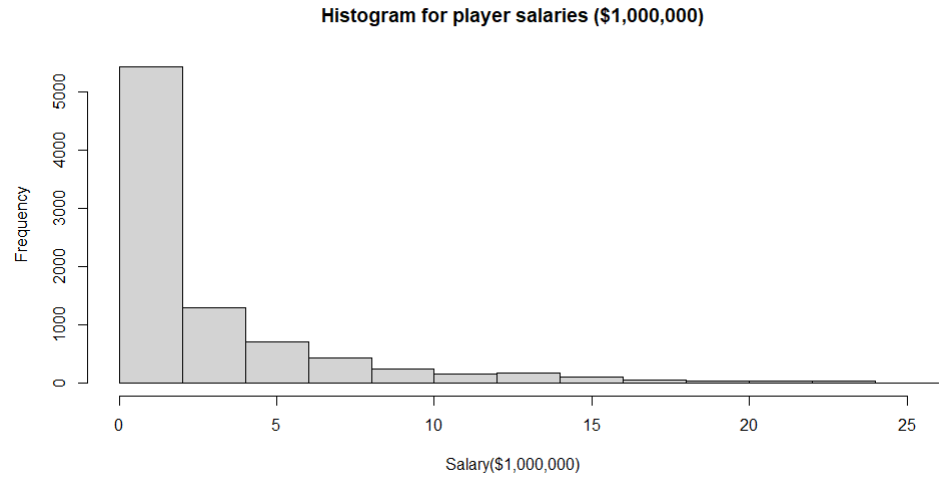

Figure 3: Player salary prior to transformation. Observe right-skewness in the data.

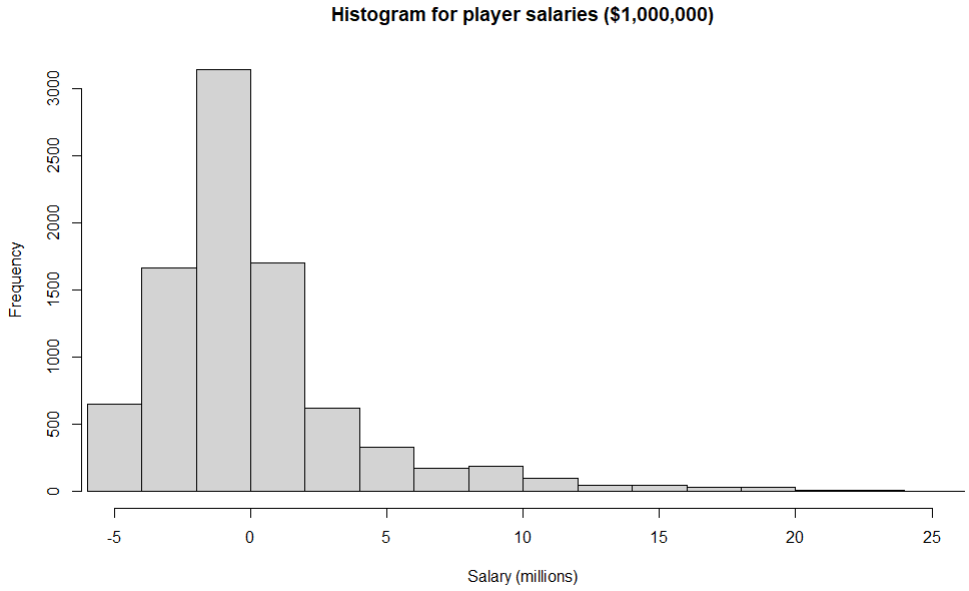

Figure 4: Player salary after transformation

The purpose of this paper is not quite to rescale, or alter the density. Since we are using tree-based models, this should not play a major factor. However, as long as we address the time-series issue of the upward trend of average player salaries, then we should be able to have the model learn better. We observe the changes in player salaries by year:

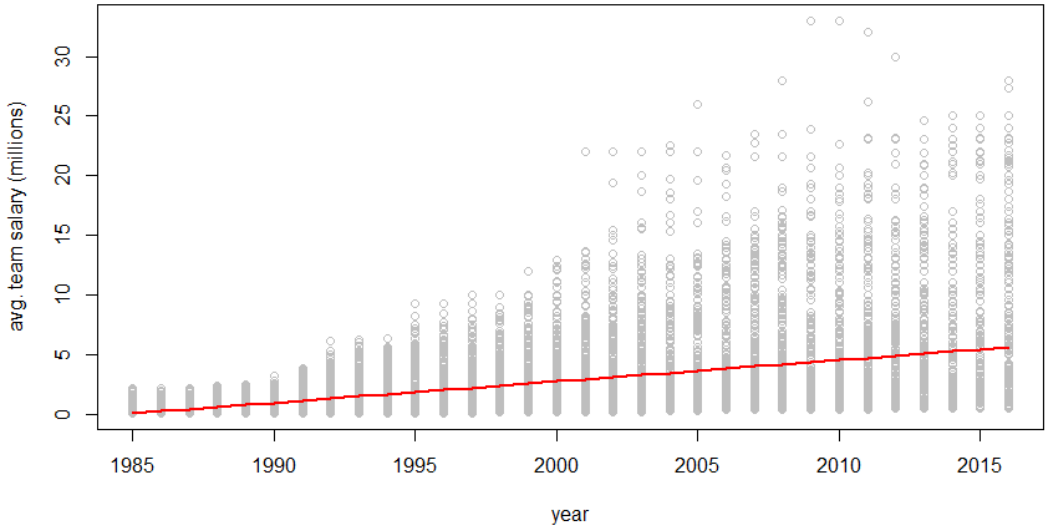

Figure 5: Player salary by year prior to transformation. The red line is the regression of salary by year.

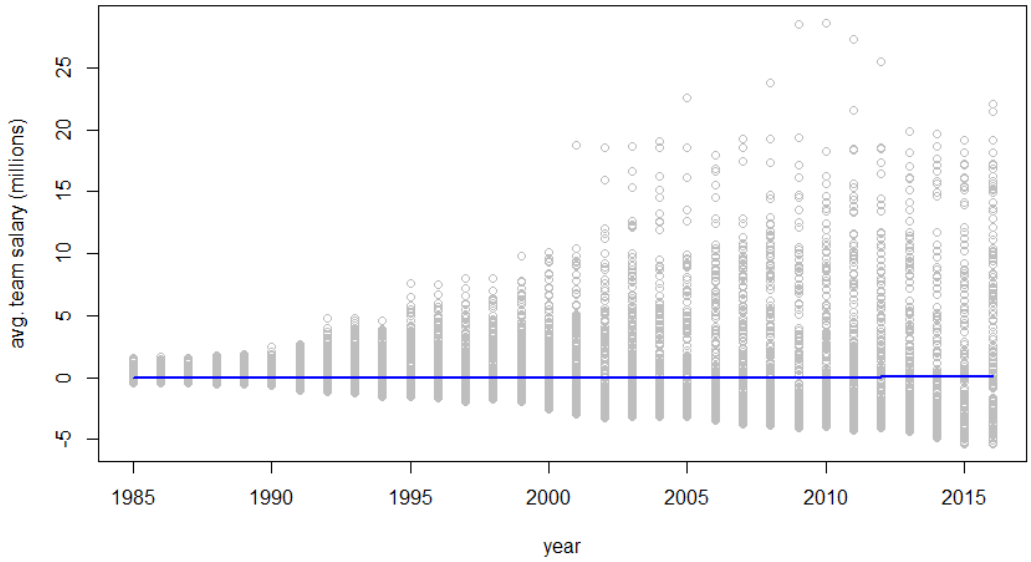

Figure 6: Player salary by year after transformation. The blue line is the regression of salary by year

Another significant observation is that by subtracting the yearly mean salary from each player, we eliminate the upward trend. Observe, the mean is zero for all years:

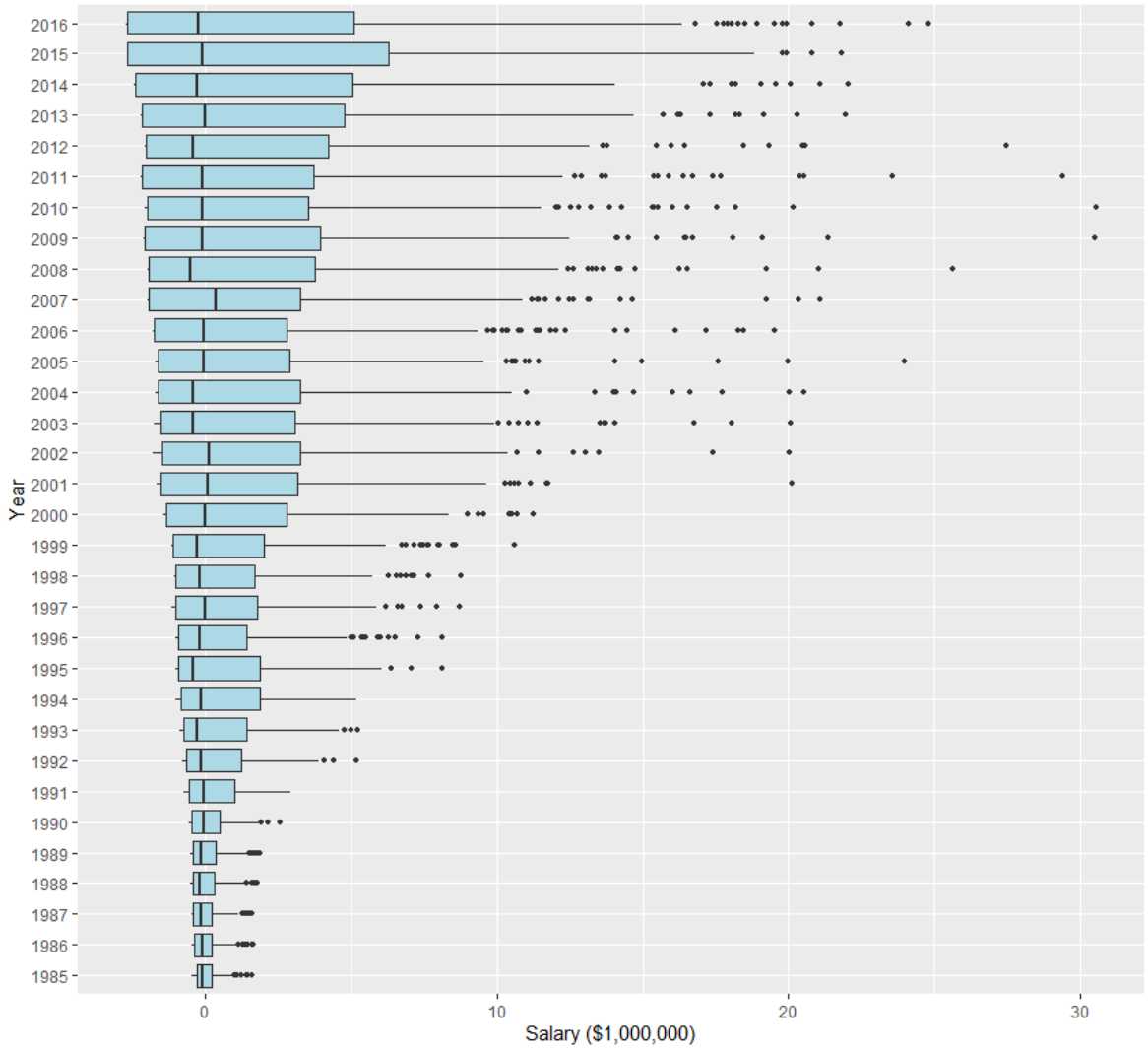

Figure 7: Player salaries by year with box-whisker plots. Dots are statistical outliers.

We did not find it appropriate to remove these outliers to train the model. In real-life context, these players are payed way above the mean for multiple reasons, i.e. teams with higher caps have the funds to pay their players much higher than other teams could. This high pay also keeps these players off the market from teams that could not otherwise afford them. More importantly, we are seeking important variables that dictate pay using tree-based models, and these guys certainly get paid a high amount for a reason. Let's explore.

2.2 Decision Trees: Regression

Decision trees can be used for both regression and classification problems. When considering which model to be most appropriate, we chose to sacrifice predicting power over interpretability and visualization by using a tree. This is because context beyond test error comes into fruition when variable selection is desired in baseball.

2.2.1 Building the Tree

We begin by growing a tree using recursive partitioning and regression trees (rpart)[4]. At each iteration in the process, the split occurs based on the predictor that results in the largest possible reduction in heterogeneity of the predicted variable (salary, in our case). Rpart makes locally optimal decisions at each stage, reducing the residual sum of squares at each iteration[1]. Observe the tree:

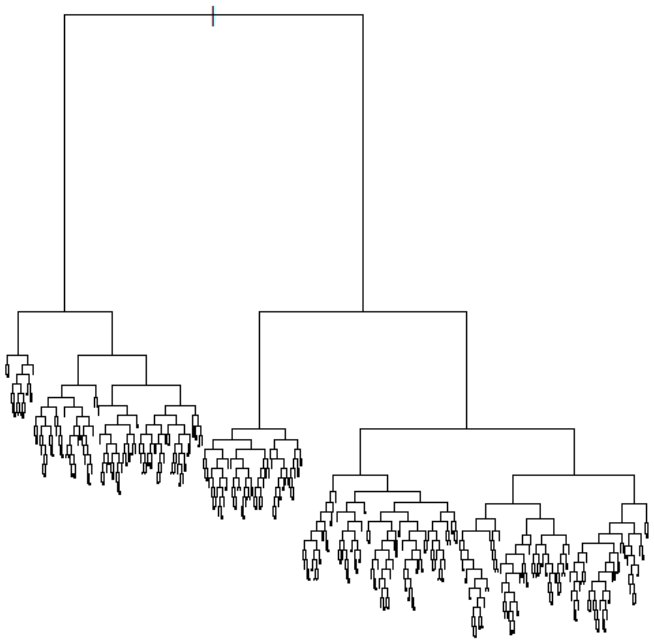

Figure 8: The unlabeled big tree of our baseball data grown by rpart.

This is not ideal, however, since a much simpler and interpretable tree would be better suited in practice.

2.2.2 Pruning

The bias-variance trade-off is one of, if not the most important concepts in statistical learning. Although low error is nice, overfitting the model may reduce its predictive power. In practice, a method supported by cross-validation would be the most useful means to reduce error rate. However, the task of calculating the cross-validation error of every subtree would be arduous to say the least. Instead, we consider a sequence of trees via cost complexity pruning[1]. In an attempt to control overfitting, we use rpart to grow the tree unrestricted and then prune it back using an appropriate criterion, $\alpha = 0.0005$ and 5 minimum splits in our case. Observe the relative error vs. complexity (as a transformation of the size of the tree):

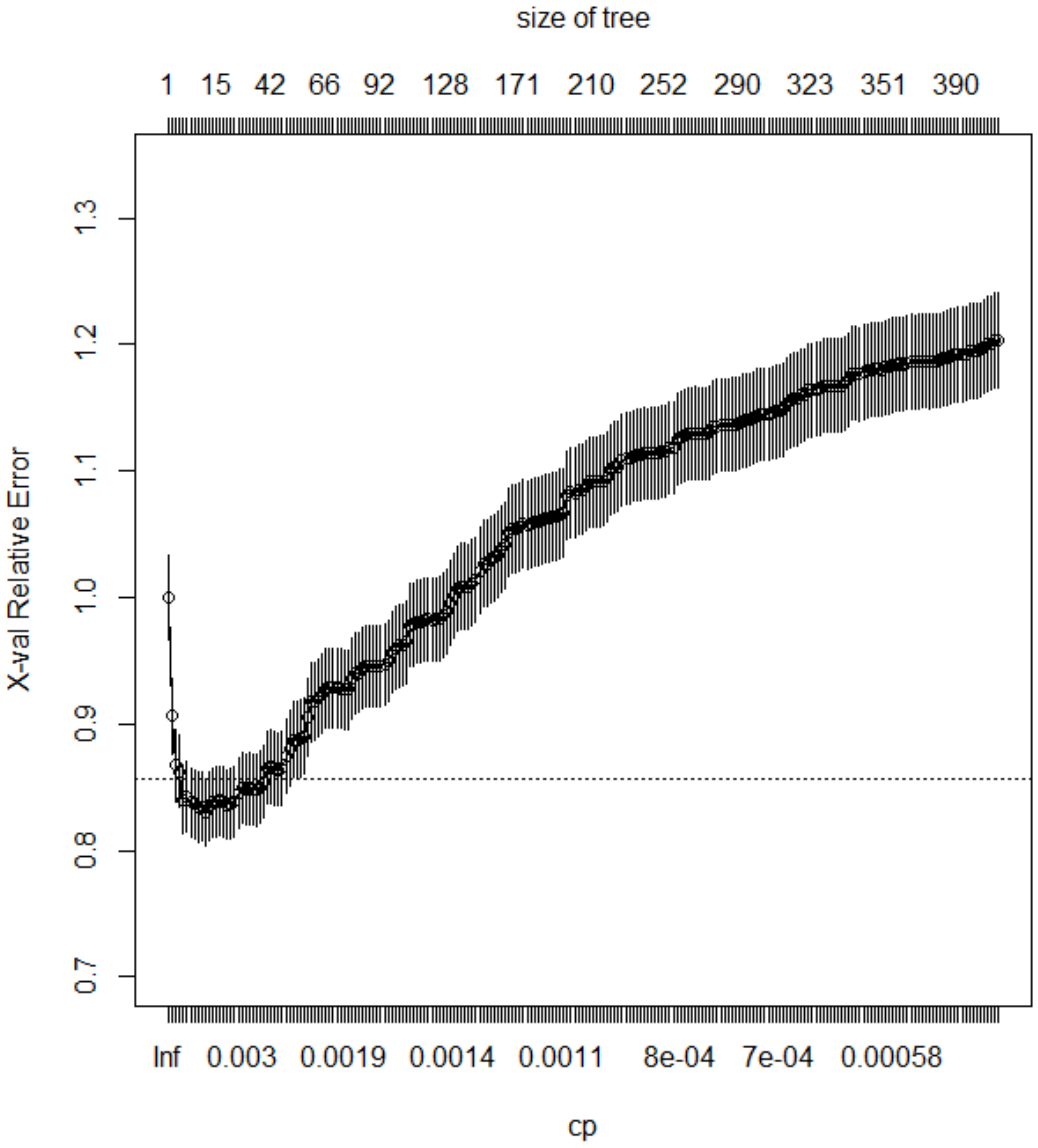

Figure 9: Relative error vs. complexity

Observe, the cost-complexity plot reveals that a tree of size 12 is roughly the best subtree of our large tree.

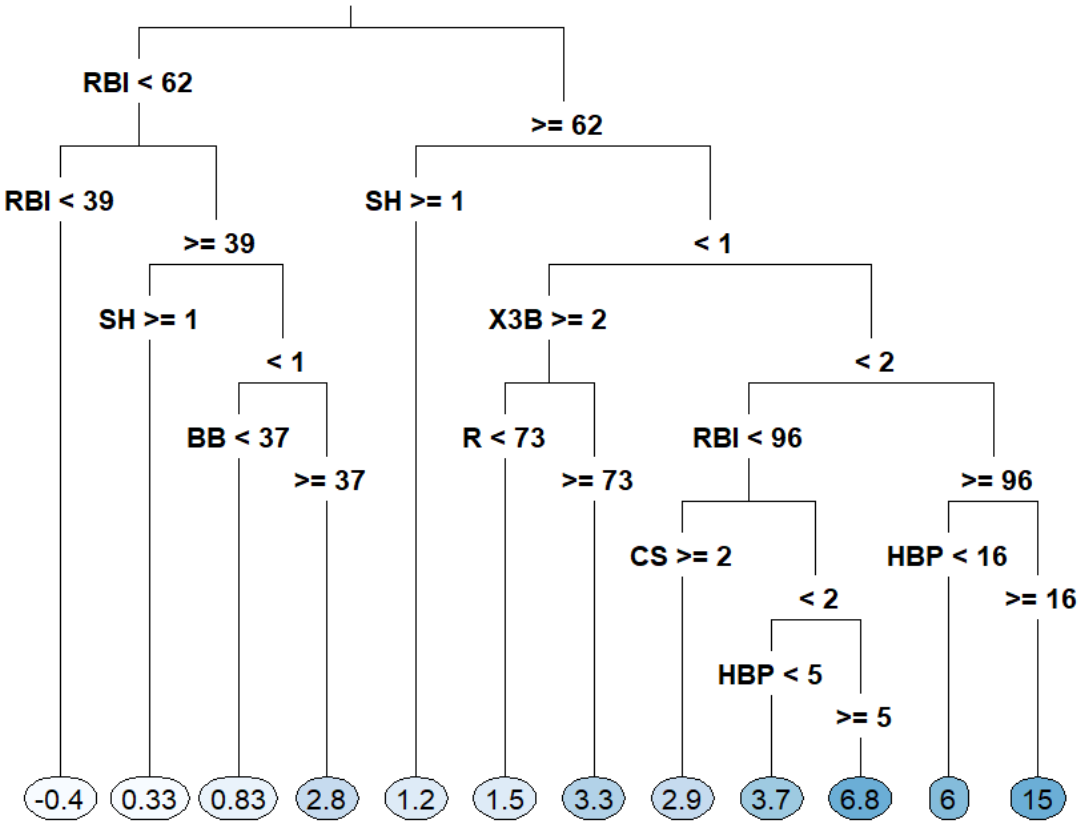

Figure 10: Pruned tree

This is a valuable tree, because it gives valuable information as to what conditions are necessary to reach certain salary thresholds. In rpart trees, this is read by moving to the left branch when the logical condition is true. For example, this model says that a player is worth 15 million USD if they:
have greater than or equal to 62 runners batted in,
have 0 (less than one) sacrifice bunts,
less than 2 triples,
greater than or equal to 96 runners batted in,
and have been hit by pitch greater than or equal to 16 times.
Sounds like now we know how to make the big bucks in the major leagues! However, we should check the prediction accuracy before moving forward.

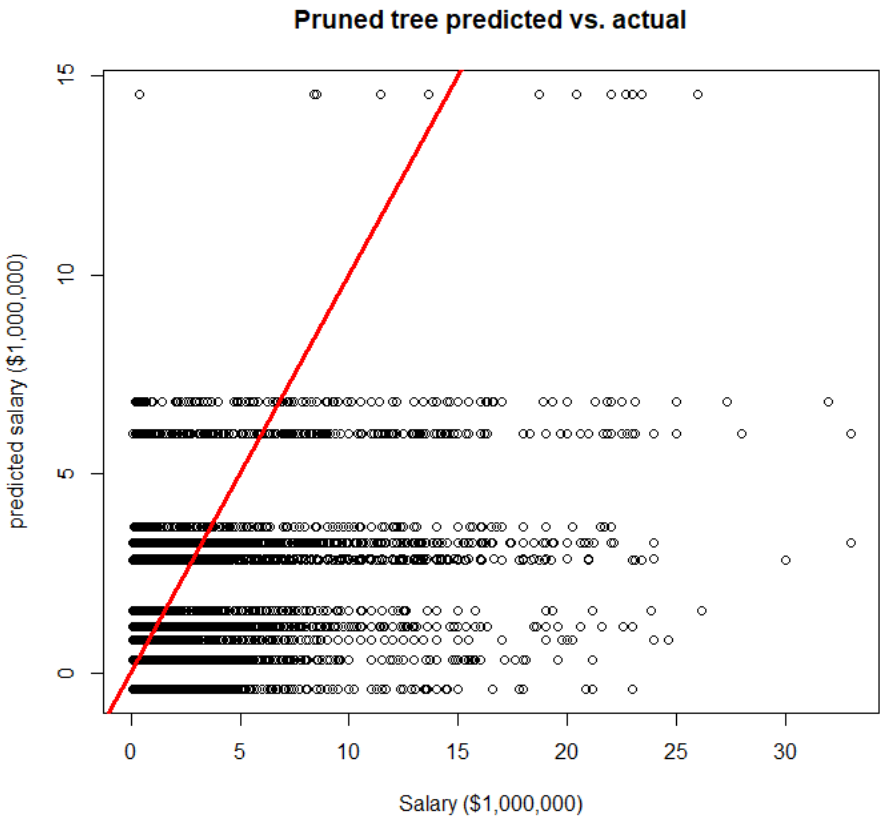

Figure 11: Pruned tree predicted vs. actual

Observe that the model's predicting power needs quite a bit of work. Tree's generally do not have the same predictive accuracy as other methods, and are also highly sensitive to changes in the data[1]. This can be addressed by aggregating many decision trees, which we do with our data via bootstrap aggregation, random forests, and boosting.

3 Results

In this section, we discuss the results obtained through our proposed methods.

3.1 Bootstrap aggregating

Circling back to the bias-variance trade-off, we observed that our decision trees has extremely high variance and very little predictive accuracy - but they were extremely *interpretable*. So how can we keep our interpretability while reducing variance? By bootstrap aggregation, or bagging.

Bagging is a method that generates a number of different bootstrapped training data sets, build a separate prediction model using each set, then average the predictions. To ensure test error is in check, cross-validation is, again, not necessary. Instead, we look at the remaining observations not used to fit a bagged tree. These are the out-of-bag (OOB) observations. Since we are using decision trees for regression, we calculate the out-of-bag mean-squared-error, which turns out to be a valid estimate of the test error for a bagged model.

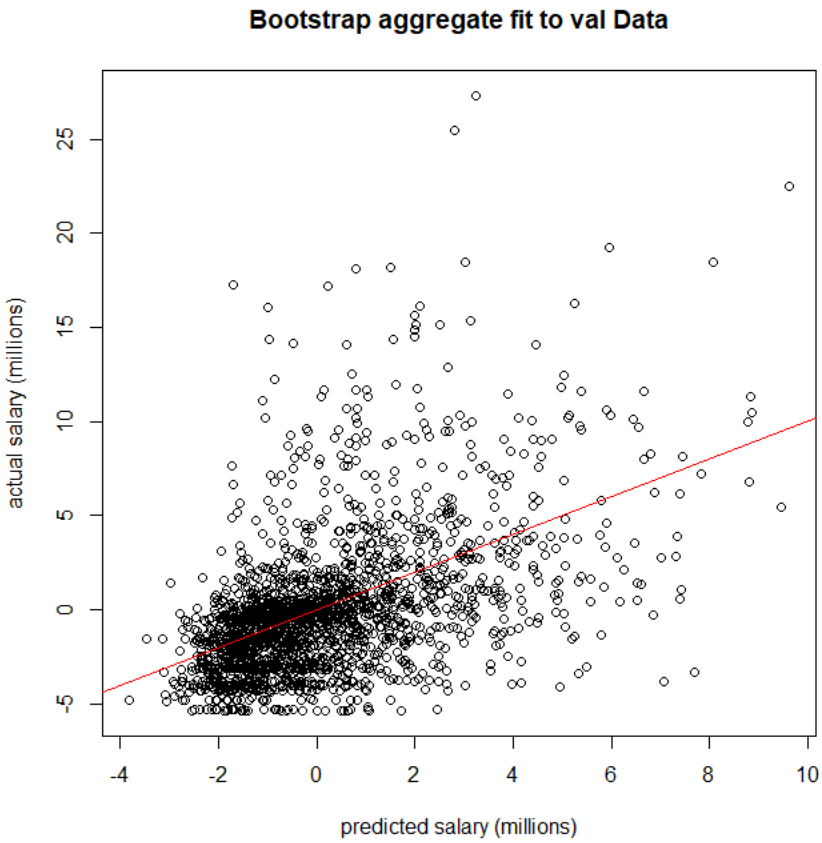

Figure 12: Predicted vs. Actual salary on validation set: Bootstrap Aggregation

Recall that we lose interpretability as we incorporate more trees into our model. However, an overall summary of importance of each predictor can be obtained by recording the total amount that the MSE is decreased due to splits over a given predictor, averaged over all of the bagged trees[1].

3.2 Boosting

Another means for improving prediction power from a decision tree is boosting.

Unlike bagging, boosting does not use bootstrap sampling, but instead each iterative tree is fit to a modified version of the original data set[1]. The boosting approach learns slowly by fitting the residuals from the given model, as opposed to the outcome. This is followed by updating the residuals, slowly improving the model in areas where performance is subpar. The rate at which improvement occurs is controlled by the shrinkage parameter, usually denoted λ. Another parameter, d, controls the interaction order of the boosted model. This parameter is also important for controlling the complexity of the boosted ensembles[1].

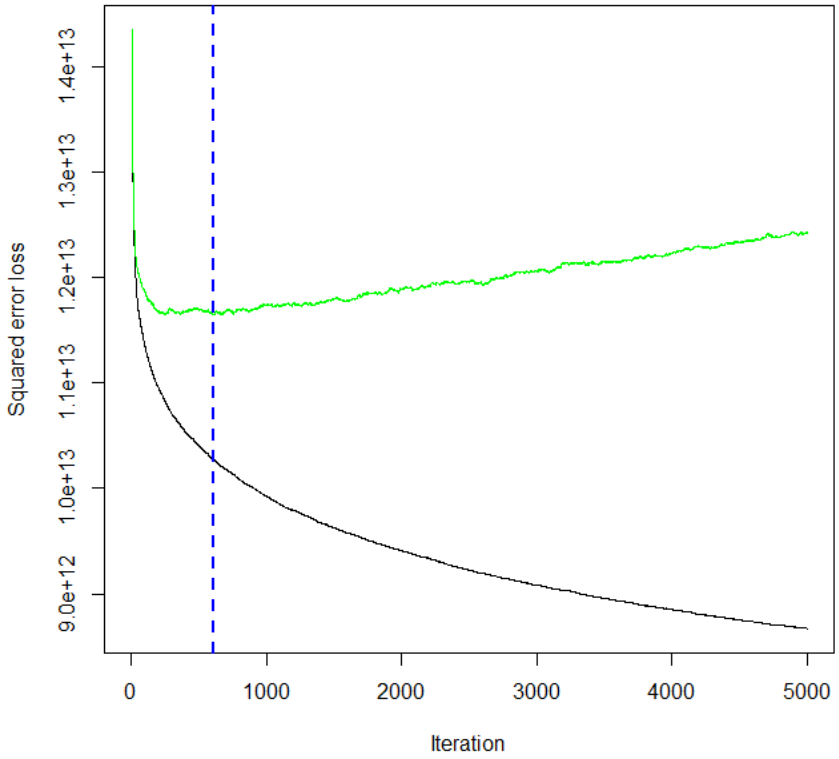

Figure 13: Best boosting value via cross-validation error

Using the *GBM* package in R, specifically *gbm.perf*, we are able to estimate the optimal number of boosting iterations for the model and plot the Figure 13. This figure gives us the optimal iteration of boosted trees, shrinkage, and interaction depth. With these parameters optimized, we then fit the boosted model to our validation data (Figure 14), and our test data (Figure 15).

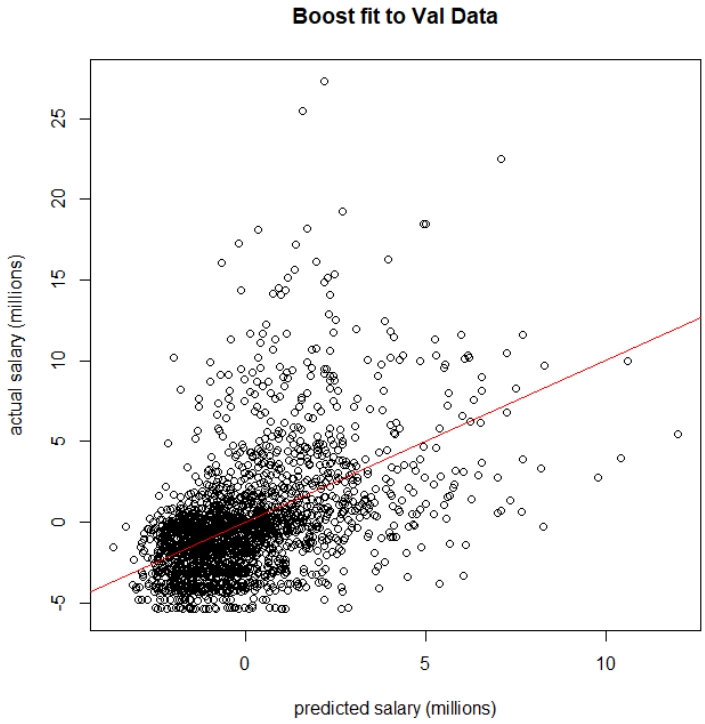

Figure 14: Predicted vs. Actual salary on validation set: Boosting

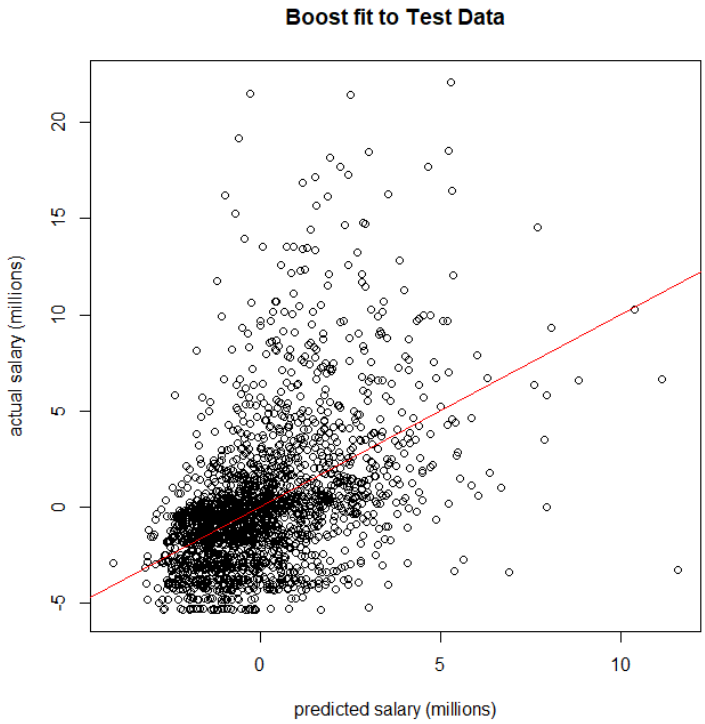

Figure 15: Predicted vs. Actual salary on validation set: Boosting

3.3 Random Forests

Random forests have much more interpretability because of their inherent randomness, although they do lack the predicting power as we will see shortly. This is fine, however, since our goal is to compare the relative influence of the predictors in these models, as well as gain insight on the importance of the predictors.

Because we split our data into three sets (train, validate, test), we begin by growing a random forest with the standard $m = \sqrt{p}$ nodes, and 500 trees. When fitting a regression to the validation set, we get Figure 12.

We see that there are many outliers in the model, particularly the players that were paid upwards of 33,000,000 USD. Also, notice how the model fitted to the validation set compared to the fit of the test set. There seems to be much more variance in the fit of the test data, which is bound to happen when fitting to out-of-bag data.

The out-of-bag (OOB) error is very high in the random forest fits. But we will see that the variable importance is certainly interpretable.

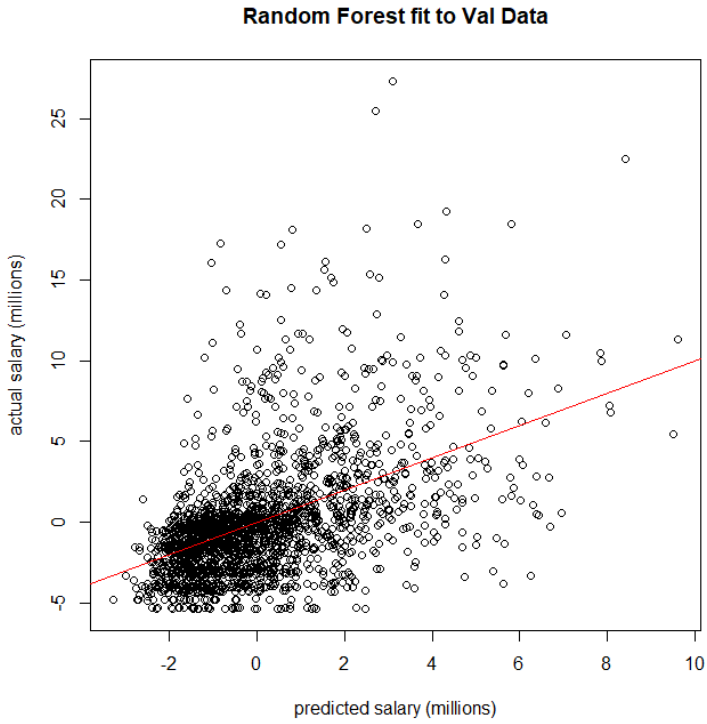

Figure 16: Predicted vs. Actual salary on validation set: Random Forest

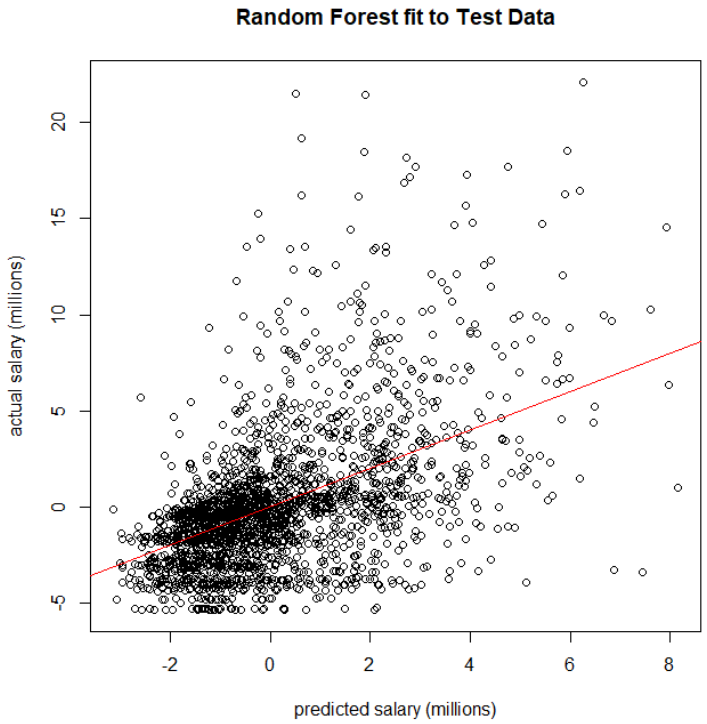

Figure 17: Predicted vs. Actual salary on Test set: Random Forest

3.4 Variable Importance

Since the splits of random forests are inherently independent, and thus much more random, we select variable importance based off of the random forest fit to the test data. We considered using the increase in MSE percentage per permutation as a method of determining importance, but decided against it since our error is so high. Instead, we decided to use how often a variable is selected closest to the root of the tree.

In both a distribution plot and multi-way importance plot, we found that base on balls (BB) and runners batted in (RBI) were the most important features in determining salary. We then compared the prediction of the model for different values of BB and RBI to obtain this matrix, which is consistent in showing that a player with RBI roughly 100 and higher with a base on balls roughly 50 or higher will generally get paid closer to 7,500,000 USD.

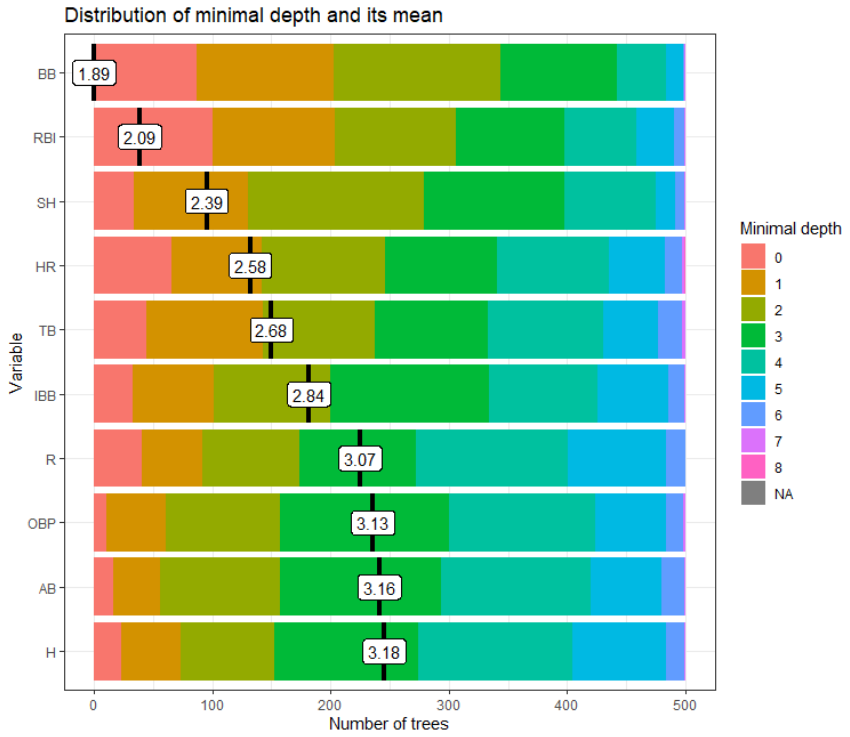

Figure 18: Minimal node depth and it's mean node depth of 10 most influential variables

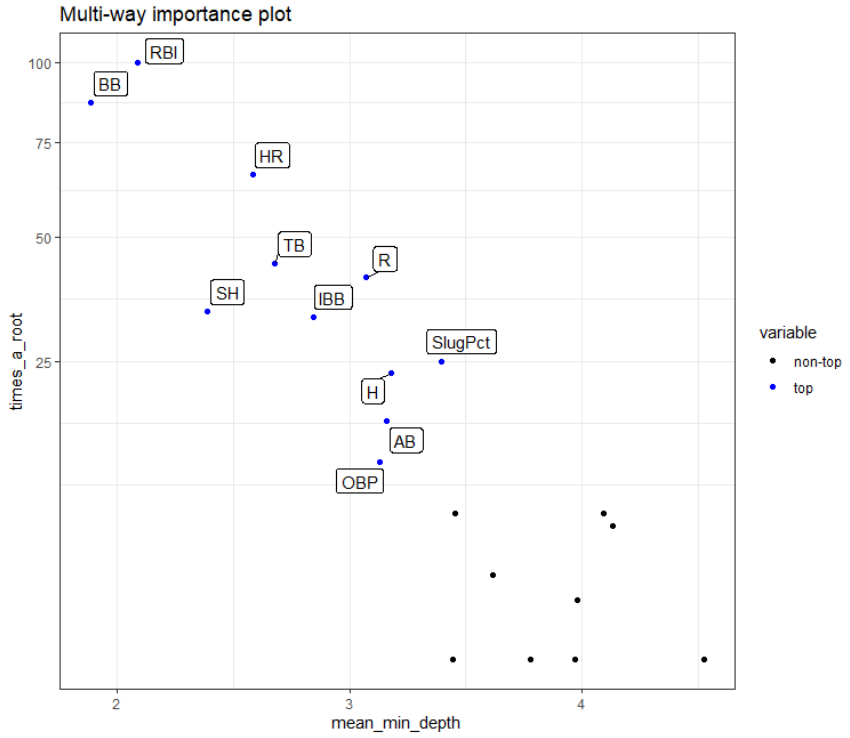

Figure 19: Multiway variable importance plot with the top 11 variables

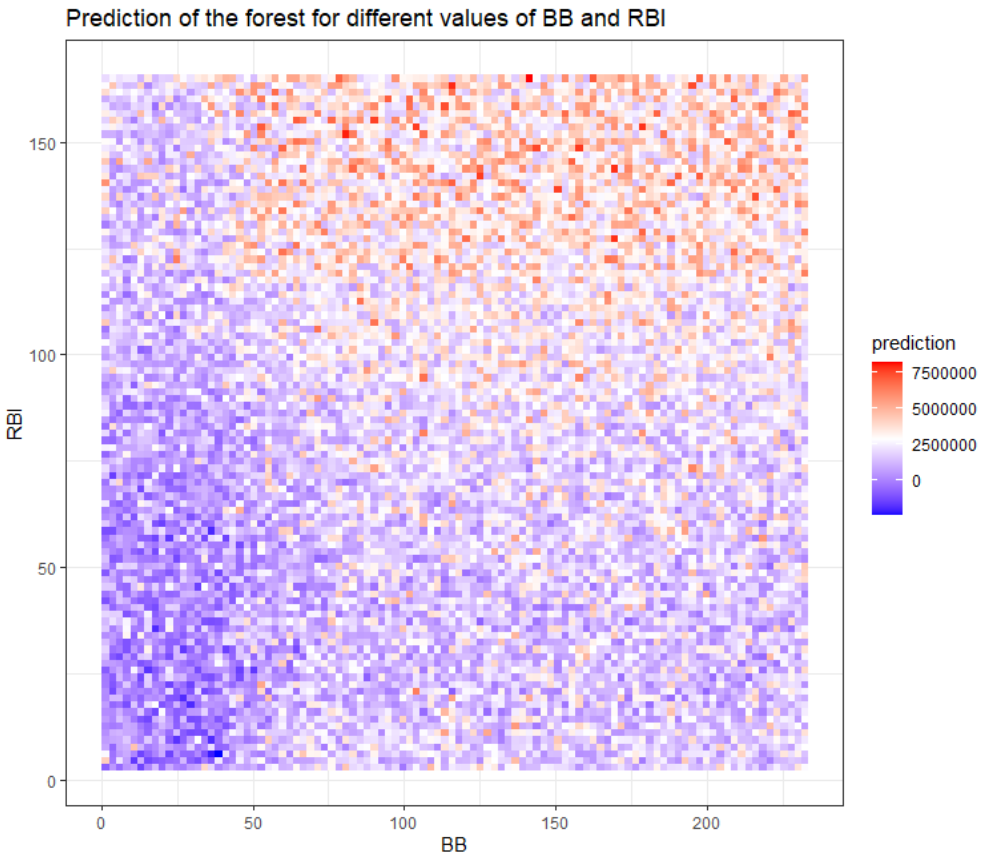

Figure 20: Prediction matrix with base-on-balls (BB) and runners-batted-in (RBI) as the predictors

4 Discussion and Conclusions

We have seen from the different models that each has it's own advantage. At each split, each decision tree only takes a bootstrapped sample of predictors, which prevents trees from using the exact same variables. Thus, random forest error rates are lower than trees, making the model more robust to outliers in the data[5][6]. Although the importance of features in random forests can be extracted, random forests, used for regression in particular, lack explanatory power. The complex nature of random forests makes it quite difficult to understand the decision mechanism for each decision made. With many different subsets of predictors in each case, even though we can learn the more significant features, it is near impossible to gain an adequate understanding of a broader decision-making framework for interpreting points of interest for player salaries in the way a logistic regression model could, or a simpler decision tree facilitate.

5 Acknowledgement

I would like to thank both ASU's Dr. Marko Samara and Dr. Robert McCulloch. Dr. Samara mentored me in probability and Bayesian inference, and Dr. McCulloch mentored me in statistical learning. Being a mathematics major, this was certainly a step outside of my comfort zone. I have learned much from both classes, not only the scientific knowledge in theory and application, but also the importance of discussions, collaborations, etc.

References

[1] G. James et al., An Introduction to Statistical Learning: with Applications in R, Springer Texts in Statistics, Springer Science+Business Media New York. 2013.

[2] Dalzell, C et al., Lahman: Sean 'Lahman' Baseball Database, v9.0, 2021 https://CRAN.R-project.org/package=Lahman.

[3] Lahman, S., Lahman's Baseball Database, 1871-2020, 2021 version,http://www.seanlahman.com/baseball-archive/statistics/

[4] Breiman, L et al., Classification and Regression Trees. Wadsworth 1984.

[5] Breiman, L. Random Forests. Machine Learning 45, 5–32 2001. https://doi.org/10.1023/A:1010933404324

[6] Hastie et al., The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Springer Texts in Statistics. 2009

[7] Friedman, J.H., Greedy Function Approximation: A Gradient Boosting Machine, Annals of Statistics 29(5):1189-1232. 2001

Appendices

A R Code

```
library(Lahman)
library(caret)
library(dplyr)
library(ggplot2)


rm(list=ls())
#-------------------------------------------------------------------------------
#-------------------------------------------------------------------------------
### GET/CLEAN/EXPLORE DATA ###
#-------------------------------------------------------------------------------
#-------------------------------------------------------------------------------
data(Teams); data(Batting); data(Salaries); data(Appearances)
head(Batting)
head(Teams)
head(Salaries)
head(Appearances)

stats <- battingStats(data = Lahman::Batting,
              idvars = c("playerID", "yearID", "stint", "teamID", "lgID"),
              cbind = TRUE)

ba <- Batting %>%
  filter(yearID >= 1985) %>%
  left_join(select(Salaries, playerID, yearID, teamID, salary),
            by=c('playerID', 'yearID', 'teamID')) %>%
  left_join(select(stats, playerID, yearID, teamID, OBP, TB, SlugPct),
            by=c('playerID', 'yearID', 'teamID')) %>%
  left_join(select(Appearances, playerID, yearID, teamID, G_p),
            by=c('playerID', 'yearID', 'teamID'))



ba1 <- ba %>%
  filter(G_p == 0, salary > 0, AB > 20, G >= 80) %>%
  select(playerID, yearID, teamID, G, AB, R, H, HR, X2B, X3B, RBI, BB, SO, SB, CS, salary,
         IBB, HBP, SH, SF, GIDP, OBP, TB, SlugPct)

### uncomment option how to handle na ###
#ba$salary[is.na(ba$salary)] <- median(ba$salary, na.rm=TRUE) #na to median
ba1 <- na.omit(ba1) #delete all rows with na
#ba[is.na(ba)] = 0 #change all na to 0

### initialize matrix ###
salmatrix <- ba1 %>%
```

```
  select(playerID, yearID, teamID, salary)

### put total and count by year into a matrix ###
nyears = 1985:2016
totalmat = matrix(0, length(nyears), 3)
# for (i in 1:length(nyears)){
#   sum1 = sum(ba1$yearID == nyears[i])
#   for (j in 1:sum1){
#     totalmat[i,1] = totalmat[i,1] + ba1$salary[j]
#     totalmat[i,2] = totalmat[i,2] + 1
#     totalmat[i,3] = totalmat[i,1]/totalmat[i,2]
#   }
# }
sum1 = 0
n = 1
for (i in 1:length(nyears)){
  sum1 = sum1 + sum(ba1$yearID == nyears[i])
  for (j in n:sum1){
    totalmat[i,1] = totalmat[i,1] + ba1$salary[j]
    totalmat[i,2] = totalmat[i,2] + 1
    totalmat[i,3] = totalmat[i,1]/totalmat[i,2]
    n = sum1
  }
}

### label columns of totalmat ###
colnames(totalmat) = c("total", "count", "average")
totalmat <- data.frame(totalmat)


#-------------------------------------------------------------------------------
#-------------------------------------------------------------------------------
### SALARY OPTIONS: choose option ###
#-------------------------------------------------------------------------------
#-------------------------------------------------------------------------------

### display density of salary before changing ###
hist(ba1$salary/1e6,
     main = "Histogram for player salaries ($1,000,000)",
     xlab = "Salary($1,000,000)",
     xlim=c(0,25))

### choose salary option 1 2 or 3 ###
option = 3

### option 1: merge normalized salary with data ###
### normalize salary in salmatrix ###
if(option == 1){j = 1
for (i in 1:nrow(salmatrix)){
```

```
  if (salmatrix[i,2] != nyears[j])
    j = j+1
  salmatrix[i,4] = salmatrix[i,4] / totalmat[j, 1]
}

ba1 <- ba1 %>%
  left_join(select(salmatrix, playerID, yearID, teamID, salary),
            by=c('playerID', 'yearID', 'teamID'))

ba1 <- ba1 %>%
  rename(
    salary = salary.x,
    sal.norm = salary.y
)

}
### option 2: normalize salary by log salary ###
if(option == 2){
  ba1$salary = log(ba1$salary)
}
### option 3: subtract mean from salary ###
### subtract mean from salary in salmatrix ###
if(option == 3){
  j = 1
for (i in 1:nrow(salmatrix)){
  if (salmatrix[i,2] != nyears[j])
    j = j+1
  salmatrix[i,4] = salmatrix[i,4] - totalmat[j, 3]
}

ba1 <- ba1 %>%
  left_join(select(salmatrix, playerID, yearID, teamID, salary),
            by=c('playerID', 'yearID', 'teamID'))

ba1 <- ba1 %>%
  rename(
    salary = salary.x,
    sal.norm = salary.y
  )
}

hist(ba1$sal.norm/1e6,
     main = "Histogram for player salaries ($1,000,000)",
     xlab = "Salary (millions)",
     xlim=c(-5,25))

densityplot(ba1$sal.norm/1e6,
            main = "Density of player salaries ($1,000,000)",
            xlab = "Salary (millions)",
```

```
            xlim=c(-5,25))

### note: explore with time fixed effects ###
#-------------------------------------------------------------------------

### visualize predictor distribution ###
avg = aggregate(ba1$salary/1e6 ~ yearID, ba1, mean)
as.data.frame(avg)
plot(avg,
     xlab = "year", ylab = "avg yearly salary (millions)")

ba1 <- select(ba1, -"playerID", -"teamID")
### more visuals ###
library(reshape2)
melt.ba1 <- melt(ba1)
head(melt.ba1)
library(ggplot2)
ggplot(data = melt.ba1, aes(x = value)) +
  stat_density() +
  facet_wrap(~variable, scales = "free")
boxplot(ba1)


### OPTIONAL: LOOK AT TEAM BACKGROUND, UNCOMMENT ###
### Team background ###
franchise <- c(`ANA`= "LAA",`ARI`= "ARI",`ATL`= "ATL",`BAL`= "BAL",`BOS`= "BOS",`CAL`= "LAA"
               `CHA`= "CHA",`CHN`= "CHN",`CIN`= "CIN",`CLE`= "CLE",`COL`= "COL",`DET`= "DET"
               `FLO`= "MIA",`HOU`= "HOU",`KCA`= "KCA",`LAA`= "LAA",`LAN`= "LAN",`MIA`= "MIA"
               `MIL`= "MIL",`MIN`= "MIN",`ML4`= "MIL",`MON`= "WAS",`NYA`= "NYA",`NYM`= "NYM"
               `NYN`= "NYN",`OAK`= "OAK",`PHI`= "PHI",`PIT`= "PIT",`SDN`= "SDN",`SEA`= "SEA"
               `SFG`= "SFN",`SFN`= "SFN",`SLN`= "SLN",`TBA`= "TBA",`TEX`= "TEX",`TOR`= "TOR"
               `WAS`= "WAS")

Salaries$franchise <- unname(franchise[Salaries$teamID])

avg_team_salaries <- Salaries %>%
  group_by(yearID, franchise, lgID) %>%
  summarise(salary= mean(salary)/1e6) %>%
  filter(!(franchise == "CLE" & lgID == "NL"))

x <- avg_team_salaries$yearID; y <- avg_team_salaries$salary
plot(x,y, col='grey',
     xlab = "year", ylab = "avg. team salary (millions)")

x <- ba1$yearID; y <- ba1$sal.norm
plot(x,y/1e6, col='grey',
     xlab = "year", ylab = "avg. team salary (millions)")

#regression
```

```
mvp = lm(y~x, ba1)
yhat <- mvp$fitted
lines(x,yhat/1e6, col='blue', lwd=2)

# player salaries
ggplot(ba1, aes(x = factor(yearID), y = sal.norm/1e6)) +
  geom_boxplot(fill = "lightblue", outlier.size = 1) +
  labs(x = "Year", y = "Salary ($1,000,000)") +coord_flip()


#------------------------------------------------------------------------
#------------------------------------------------------------------------
### PRUNE PLOT FIT ###
#------------------------------------------------------------------------
#------------------------------------------------------------------------
library(rpart)
library(rpart.plot)

ddf=ba1[,c(2,3,4,5,6,7,8,9,10,11,12,13,15,16,17,18,19,20,21,22,23)]
set.seed(36)
big.tree = rpart(ba1$sal.norm~.,method = "anova" ,data=ddf,
                 control=rpart.control(minsplit=5,cp=.0005))
nbig = length(unique(big.tree$where))
cat("size of big tree: ",nbig,"\n")

###look at CV results
plot(big.tree)
plotcp(big.tree)

### get nice tree from CV results
iibest = which.min(big.tree$cptable[,"xerror"]) #which has the lowest error
bestcp=big.tree$cptable[iibest,"CP"]
bestsize = big.tree$cptable[iibest,"nsplit"]+1

#prune to good tree
best.tree = prune(big.tree,cp=bestcp)
nbest = length(unique(best.tree$where))
cat("size of best tree: ", nbest,"\n")

### plot tree
plot(best.tree,uniform=TRUE)
text(best.tree,digits=4,use.n=TRUE)

### using rpart.plot
rpart.plot(best.tree,split.cex=1.0,cex=1.3,type=3,extra=0)

### get fit ###
yhat = predict(best.tree)
plot(ba1$salary/1e6,yhat,
```

```
     main = "Pruned tree predicted vs. actual",
     xlab = "Salary ($1,000,000)",
     ylab = "predicted salary ($1,000,000)")
abline(0,1,col="red",lwd=3)

#------------------------------------------------------------------------
#------------------------------------------------------------------------
### BAGGING, BOOSTING AND RF ###
### VARIABLE IMPORTANCE ###
#------------------------------------------------------------------------
#------------------------------------------------------------------------

ddf1=ba1[,c(2,3,4,5,6,7,8,9,10,11,12,13,15,16,17,18,19,20,21,22,23)]
### Bagging, Boosting and Random Forest ###
set.seed(32)
n=nrow(ddf1)
n1=floor(n/2)
n2=floor(n/4)
n3=n-n1-n2
ii = sample(1:n,n)
batrain= ddf1[ii[1:n1],]
baval = ddf1[ii[n1+1:n2],]
batest = ddf1[ii[n1+n2+1:n3],]


library(randomForest)
library(tree)
### bagging: mtry = 20 indicates all predictors considered ###
rffit.bag = randomForest(sal.norm~.,data=batrain,mtry=20,
                         ntree=250, importance = TRUE)
plot(rffit.bag)
varImpPlot(rffit.bag)
vip::vip(rffit.bag, num_features = 20L,
         geom = c("point"))

### how well did bagged model perform on val set? ###
yhat.bag = predict(rffit.bag, newdata=baval)
plot(yhat.bag/1e6, baval$sal.norm/1e6,
     main = "Bootstrap aggregate fit to val Data",
     xlab = "predicted salary (millions)",
     ylab = "actual salary (millions)")
abline(0,1, col = 'red')
mse <- mean((yhat.bag - baval$sal.norm)^2)
cat("out of sample rmse on val set: ",sqrt(mse),"\n") #3186133

### try sqrt m ###
rffit = randomForest(sal.norm~.,data=batrain,mtry=4,
                     ntree=250, importance = TRUE)
plot(rffit)
```

```
varImpPlot(rffit)
vip::vip(rffit, num_features = 20L,
         geom = c("point"))

### how well did rF model perform on val set? ###
yhat.rF = predict(rffit, newdata=baval)
plot(yhat.rF/1e6, baval$sal.norm/1e6,
     main = "Random Forest fit to Val Data",
     xlab = "predicted salary (millions)",
     ylab = "actual salary (millions)")
abline(0,1, col = 'red')
mse <- mean((yhat.rF - baval$sal.norm)^2)
cat("out of sample rmse on val set: ",sqrt(mse),"\n") #3172855


### try boost ###
library(gbm)
set.seed(1)
boostfit = gbm(sal.norm~.,data=batrain,distribution="gaussian",
               interaction.depth=4,n.trees=5000,shrinkage=.2, cv.folds = 5)

bestboost = gbm(sal.norm~., data=batrain, distribution = "gaussian",
               n.trees=5000, cv.folds = 10)
best.sal <- gbm.perf(bestboost, method = "cv")

yhat.boost=predict(boostfit,newdata=baval,n.trees=5000)
yhat.best=predict(bestboost,newdata=baval,.n.trees=best.sal)
plot(yhat.best/1e6, baval$sal.norm/1e6,
     main = "Boost fit to Val Data",
     xlab = "predicted salary (millions)",
     ylab = "actual salary (millions)")
abline(0,1,col="red")
mse.boost <- mean((yhat.boost - baval$sal.norm)^2)
mse.best <- mean((yhat.best - baval$sal.norm)^2)
cat("out of sample rmse on val set: ",sqrt(mse.boost),"\n") #3697313
cat("out of sample rmse on val set: ",sqrt(mse.best),"\n") #3214346




plot(yhat.best/1e6, batest$sal.norm/1e6,
     main = "Boost fit to Val Data",
     xlab = "predicted salary (millions)",
     ylab = "actual salary (millions)")
abline(0,1,col="red")

#plot (out-of-sample) fits
y = baval$sal.norm
bag = yhat.bag
```

```
boost = yhat.best
pairs(cbind(y,yhat.best,yhat.boost))
pairs(cbind(y,bag,boost))
print(cor(cbind(baval$sal.norm,yhat.bag,yhat.boost)))

#combine train and validation set and refit using boosting.
batrainval = rbind(batrain,baval)

#boost refit
boostfit2 = gbm(sal.norm~.,data=batrainval,distribution="gaussian",
                n.trees=5000, cv.folds = 10)
best.sal2 <- gbm.perf(boostfit2, method = "cv")

yhat.boosttest=predict(boostfit2,newdata=batest,n.trees=best.sal2)

#plot test y vs test predictions
plot(yhat.boosttest/1e6, batest$sal.norm/1e6,
     main = "Boost fit to Test Data",
     xlab = "predicted salary (millions)",
     ylab = "actual salary (millions)")
abline(0,1,col="red")
rmse = sqrt(mean((batest$sal.norm-yhat.boosttest)^2))
cat("rmse on test for boosting: ",rmse,"\n") #3299392

#variable importance from boosting
summary(boostfit2, cBars = 11)

#refit random forests on train-val
rffit2 = randomForest(sal.norm~.,data=batrainval,mtry=4,ntree=500,
                      importance = TRUE)
rftestpred = predict(rffit2,newdata=batest)

plot(rftestpred/1e6, batest$sal.norm/1e6,
     main = "Random Forest fit to Test Data",
     xlab = "predicted salary (millions)",
     ylab = "actual salary (millions)")
abline(0,1,col="red")
mse = mean((batest$sal.norm-rftestpred)^2)
cat("rmse on test for random forests: ",sqrt(mse),"\n") #3193474

#variable importance from Random Forests
library(randomForestExplainer)
varImpPlot(rffit2)
vip::vip(rffit2, num_features = 20L,
         geom = c("point"))

plot_min_depth_distribution(rffit2)
plot_multi_way_importance(rffit2)
plot_predict_interaction(rffit2, ba1, 'BB', 'RBI')
```

```r
### hyper grid to find lowest error ###
library(ranger)
hyper_grid <- expand.grid(
  mtry       = seq(4, 20, by = 2),
  node_size  = seq(3, 31, by = 2),
  sampe_size = c(.55, .632, .70),
  OOB_RMSE   = 0
)

nrow(hyper_grid)

for(i in 1:nrow(hyper_grid)) {

  # train model
  model <- ranger(
    formula         = sal.norm ~ .,
    data            = ba1,
    num.trees       = 500,
    mtry            = hyper_grid$mtry[i],
    min.node.size   = hyper_grid$node_size[i],
    sample.fraction = hyper_grid$sampe_size[i],
    seed            = 41
  )

  # add OOB error to grid
  hyper_grid$OOB_RMSE[i] <- sqrt(model$prediction.error)
}

hyper_grid %>%
  dplyr::arrange(OOB_RMSE) %>%
  head(50)

#------------------------------------------------------------------------
#------------------------------------------------------------------------
### KNN FUNCTION ### (ASU's Robert McCulloch's KNN code)
#------------------------------------------------------------------------
#------------------------------------------------------------------------
library(kknn)
mse=function(y,yhat) {return(sum((y-yhat)^2))}
doknn=function(x,y,xp,k) {
  kdo=k[1]
  train = data.frame(x,y=y)
  test = data.frame(xp); names(test) = names(train)[1:(ncol(train)-1)]
  near  = kknn(y~.,train,test,k=kdo,kernel='optimal')
  return(near$fitted)
}
#-------------------------------------------------
docv = function(x,y,set,predfun,loss,nfold=10,doran=TRUE,verbose=TRUE,...)
  #x,y training data
```

```
  #set each row gives settings for predfun
  #predfun predicts on xp given (x,y)
  #loss: measure of fit
  #nfold: number of folds (e.g. 5 or 10)
  #doran: should you shuffle the data
{
  #a little error checking
  if(!(is.matrix(x) | is.data.frame(x))) {cat('error in docv: x is not a matrix or data fra
  if(!(is.vector(y))) {cat('error in docv: y is not a vector\n'); return(0)}
  if(!(length(y)==nrow(x))) {cat('error in docv: length(y) != nrow(x)\n'); return(0)}

  #shuffle the data
  nset = nrow(set); n=length(y) #get dimensions
  if(n==nfold) doran=FALSE #no need to shuffle if you are doing them all.
  cat('in docv: nset,n,nfold: ',nset,n,nfold,'\n')
  lossv = rep(0,nset) #return values
  if(doran) {ii = sample(1:n,n); y=y[ii]; x=x[ii,,drop=FALSE]} #shuffle rows

  #loop over folds and settings
  fs = round(n/nfold) # fold size
  for(i in 1:nfold) { #fold loop
    bot=(i-1)*fs+1; top=ifelse(i==nfold,n,i*fs); ii =bot:top
    if(verbose) cat('on fold: ',i,', range: ',bot,':',top,'\n')
    xin = x[-ii,,drop=FALSE]; yin=y[-ii]; xout=x[ii,,drop=FALSE]; yout=y[ii]
    for(k in 1:nset) { #setting loop
      yhat = predfun(xin,yin,xout,set[k,],...)
      lossv[k]=lossv[k]+loss(yout,yhat)
    }
  }

  return(lossv)
}
#------------------------------------------------
#cv version for knn
docvknn = function(x,y,k,nfold=10,doran=TRUE,verbose=TRUE) {
  return(docv(x,y,matrix(k,ncol=1),doknn,mse,nfold=nfold,doran=doran,verbose=verbose))
}


#---------------------------------------------------------------------
#---------------------------------------------------------------------
### CROSS VALIDATION FOR PREDICTING SALARY FROM IMPORTANT VARIABLES ###
#---------------------------------------------------------------------
#---------------------------------------------------------------------
### (call KNNfunc aobve)


#get top 4 predictors
x = cbind(ba1$G, ba1$OBP, ba1$H, ba1$BB)
```

```
colnames(x) = c("G", "OBP", "H","BB")
y = ba1$salary
mmsc=function(x) {return((x-min(x))/(max(x)-min(x)))}
xs = apply(x,2,mmsc) #apply scaling function to each column of x

#plot y vs each x
par(mfrow=c(1,4)) #four plot frames
plot(x[,1],y,xlab="G",ylab="salary")
plot(x[,2],y,xlab="OBP",ylab="salary")
plot(x[,3],y,xlab="H",ylab="salary")
plot(x[,4],y,xlab="BB",ylab="salary")

#run cross val once
par(mfrow=c(1,1))
set.seed(99)
kv = 2:20 #k values to try
n = length(y)
cvtemp = docvknn(xs,y,kv,nfold=5)
cvtemp = sqrt(cvtemp/n) #docvknn returns sum of squares
plot(kv,cvtemp)

#run cross val several times
set.seed(99)
cvmean = rep(0,length(kv)) #will keep average rmse here
ndocv = 15 #number of CV splits to try
n=length(y)
cvmat = matrix(0,length(kv),ndocv) #keep results for each split
for(i in 1:ndocv) {
  cvtemp = docvknn(xs,y,kv,nfold=5)
  cvmean = cvmean + cvtemp
  cvmat[,i] = sqrt(cvtemp/n)
}
cvmean = cvmean/ndocv
cvmean = sqrt(cvmean/n)
plot(kv,cvmean,type="n",ylim=range(cvmat),xlab="k",cex.lab=1.5)
for(i in 1:ndocv)
  lines(kv,cvmat[,i],col=i,lty=3) #plot each result
lines(kv,cvmean,type="b",col="black",lwd=5) #plot average result

#refit using all the data and k=15
ddf = data.frame(y,xs)
near15 = kknn(y~.,ddf,ddf,k=15,kernel = "optimal")
lmf = lm(y~.,ddf)
fmat = cbind(y,near15$fitted,lmf$fitted)
colnames(fmat)=c("y","kNN5","linear")
pairs(fmat)
print(cor(fmat))
```

```
#predict salary of my favorite Texas Ranger, Michael Young circa 2008
#155 G, .339 OBP, 183 H, 55 BB
x1=155; x2=.339; x3=183; x4=55
x1s = (x1-min(x[,1]))/(max(x[,1])-min(x[,1]))
x2s = (x2-min(x[,2]))/(max(x[,2])-min(x[,2]))
x3s = (x3-min(x[,3]))/(max(x[,3])-min(x[,3]))
x4s = (x4-min(x[,4]))/(max(x[,4])-min(x[,4]))
near = kknn(y~.,ddf,data.frame(G=x1s,OBP=x2s, H=x3s, BB=x4s),k=15,kernel = "optimal")
texas <- near$fitted
cat("knn predicted value: ",texas,"\n")
err <-  abs(texas - 6174974)/(abs(texas+6174974)/2)
cat("Michael Young was paid 6,174,974 in 2008. That's an error of: ",err,"\n")

#what does a linear model predict?
print(predict(lmf,data.frame(G=x1s,OBP=x2s, H=x3s, BB=x4s)))
#let's check we did the scaling right
lmtemp = lm(salary~G+OBP+H+BB,data=ba1)
print(predict(lmtemp,data.frame(G=155,OBP=.339, H=183, BB=55)))
#-------------------------------------------------
```