IST 707 Project

Martin Alonso, Jimmy Chen

11/28/2018

# Introduction

Every year, the NCAA women’s basketball teams play thousands of matches that will lead up to the popular March Madness championship. The championship consists of 64 teams and builds up from early October until February, where the top 16 teams per each of the four regions compete for the championship, with the final game usually played in March or April. As a result, a lot of data has been gathered over the years from these matches and can be used to try and predict the next champion.

For this project, we’ve taken data from the NCAA Women’s basketball tournaments, spanning from 1998 to 2017. These data include every game played for every season, whether the game was played at home or away, which team won, the final score, number of overtimes played, and if the game was part of the regular season or March Madness.

# The Dataset

The dataset was obtain from the 2018 Women’s NCAA March Madness (<https://www.kaggle.com/c/womens-machine-learning-competition-2018/data>) Kaggle competition, and is conformed of 9 comma-separated value (CSV) files with a variety of data columns. However, for the purpose of this project, only relevant columns will be used to predict tournament results. Examples of variables that will be used include: season, day season started, day number, winning and losing teams, winning and losing team scores, seeds (for tournament games), number of overtimes played, cities were the game was played, and whether the winning team won at home, away, or on neutral ground.

The data goes all the way back to 1998 and reaches 2017. This will allow us to discover team trends over time. However, we will probably restrict ourselves to the most recent year of data to create the predictive model because sports teams tend to vary in overall greatness over time due to players, coaches, and other factors changing. By now, the 2018 data has been released and can be used to cross validate results from the predictive models with the actual competition results.

## Data Preprocessing

We will start by loading the required packages that will be used for this project and explain why they will be used. \* dplyr: data manipulation package that will allow us to view and explore the data. \* plyr: tools that allow for data manipulation. \* tidyr: package that allows us to build datasets that conform to Hadley Wickham’s tidy dataset structure. \* lubridate: package that allows date manipulation. \* ggplot2: grammar of graphics package for exploratory data analysis, used to discover potential trends in the data. \* caret: package housing different modeling algorithms. Will be used to create the random forest and SVM models. \* randomForest: package that will allow us to build a random forest model with assistance from the caret package.

# Objective

Our objective is to take these data points and try to build a model that is capable of correctly predicting the score of the winning team for the 2017 women’s basketball season, an objective that is based on the Kaggle March Madness competition held in February 2017. We believe that by using three different models - two which we have seen in class and one which we have discovered outside of class. Our goal will be able to successfully predict over 75 percent of the match outcomes. We will use data from 1998-2016 as the training set to build the model, and the 2017 season will be the testing set.

For this project, we will be using the following three models: Random Forest, Support Vector Machine, and Linear Regression.

## Load required packages

require(dplyr)  
require(plyr)  
require(tidyr)  
require(lubridate)  
require(ggplot2)  
require(caret)  
require(e1071)  
require(randomForest)

With the said packages loaded, it is time to import the datasets that we’ll use for this project. Kaggle provides nine datasets in comma-separated value format. These csvs contain information on the team cities, the cities in which games were played, NCAA tournament results, tournament seeds, tournament slots, regular season results, seasons, teams, and team names.

For the sake of this project, we will load all the csvs but we will not use all these.

## Load the data files

# Load csv files  
cities <- read.csv('WCities.csv')  
gameCities <- read.csv('WGameCities.csv')  
compactResults <- read.csv('WNCAATourneyCompactResults.csv')  
seeds <- read.csv('WNCAATourneySeeds.csv')  
slots <- read.csv('WNCAATourneySlots.csv')  
regSeasonResults <- read.csv('WRegularSeasonCompactResults.csv')  
season <- read.csv('WSeasons.csv')  
teams <- read.csv('WTeams.csv')  
teamSpellings <- read.csv('WTeamSpellings.csv')

With the files loaded, we will now move on to create a master file that will serve us when building the model.

## Data preparation

# Add columns to files  
regSeasonResults$gameType <- 'regularSeason'  
compactResults$gameType <- 'tournament'  
  
compactResults <- compactResults %>%   
 left\_join(seeds, by = c('Season'='Season', 'WTeamID' = 'TeamID'))  
colnames(compactResults)[10] <- 'WTeamSeed'  
  
compactResults <- compactResults %>%   
 left\_join(seeds, by = c('Season'='Season', 'LTeamID' = 'TeamID'))  
colnames(compactResults)[11] <- 'LTeamSeed'  
  
seasonNew <- gather(season, key = Season, value = Region, c(RegionW, RegionY, RegionX, RegionZ))

## Warning: attributes are not identical across measure variables;  
## they will be dropped

colnames(seasonNew) <- c('DayZero', 'Region', 'Title')  
seasonNew$DayZero <- as.Date(as.character(seasonNew$DayZero), '%m/%d/%Y')  
  
seasonNew <- seasonNew %>%   
 mutate(season = year(DayZero),   
 seed = substr(Region, 7, 7))  
  
# Create a master dataset   
master <- regSeasonResults %>% bind\_rows(compactResults)  
  
master <- master %>%   
 dplyr::left\_join(season, by = c('Season' = 'Season')) %>%   
 dplyr::select(-RegionW, -RegionX, -RegionY, -RegionZ)  
master$DayZero <- as.Date(as.character(master$DayZero), format = '%m/%d/%Y')  
master$gameDay <- master$DayNum + master$DayZero  
  
master <- master %>%   
 dplyr::mutate(WRegion = substr(WTeamSeed, 1, 1),   
 LRegion = substr(LTeamSeed, 1, 1)) %>%   
 dplyr::left\_join(seasonNew, by = c('Season'='season', 'WRegion'='seed')) %>%   
 dplyr::left\_join(seasonNew, by = c('Season'='season', 'LRegion'='seed')) %>%  
 dplyr::select(-DayZero, -WRegion, -LRegion, -DayZero.y, -Region.x, -Region.y) %>%  
 dplyr::group\_by(Season, WTeamID, LTeamID) %>%  
 dplyr::mutate(meeting = 1:n()) %>%   
 dplyr::ungroup() %>%   
 dplyr::mutate(gameId = paste0(Season, '\_', WTeamID, '\_', LTeamID, '\_', meeting)) %>%   
 dplyr::select(Season, DayZero.x, gameDay, DayNum, gameId, WTeamID, WScore, LTeamID, LScore, WLoc, NumOT, gameType,   
 WTeamSeed, LTeamSeed, WRegion = Title.x, LRegion = Title.y)  
colnames(master)[2] <- 'DayZero'

Now that we have the master file ready, we’ll group by team and calculate each team’s offensive and defensive ratings, and the average point spread by which they beat - or lose to - their opponents.

We start by selecting the winning team, their score, and the score of the losing team. These last two will be the points scored and points allowed, and will be used to calculate each team’s offensive rating (OR) and defensive rating (DR).

homeTeam <- master %>%  
 dplyr::select(Season, gameId, Team1 = WTeamID, Team2 = LTeamID, PointsScored = WScore, PointsAllowed = LScore)  
awayTeam <- master %>%   
 dplyr::select(Season, gameId, Team1 = LTeamID, Team2 = WTeamID, PointsScored = LScore, PointsAllowed = WScore)  
  
totalTeam <- homeTeam %>% bind\_rows(awayTeam) %>% arrange(gameId)  
totalTeamDedup <- totalTeam[!duplicated(totalTeam$gameId), ]  
  
# First we'll calculate average points scored and allowed by the league. This needs to be done by season to account for the offensive and defensive shifts of all the teams.   
totalTeam <- totalTeamDedup %>%   
 dplyr::group\_by(Season) %>%   
 dplyr::mutate(lgPS = mean(PointsScored), lgPA = mean(PointsAllowed)) %>%   
 dplyr::ungroup()  
  
# Then, we'll create Team1's OR, DR and point spread...   
totalTeam <- totalTeam %>%   
 dplyr::group\_by(Season, Team1) %>%   
 dplyr::mutate(T1OR = mean(PointsScored)/lgPS,   
 T1DR = mean(PointsAllowed)/lgPA,  
 T1PointSpread = mean(PointsScored) - mean(PointsAllowed)) %>%  
 # ...followed by Team2's OR and DR.  
 dplyr::ungroup() %>%   
 dplyr::group\_by(Season, Team2) %>%   
 dplyr::mutate(T2OR = mean(PointsAllowed)/lgPA,  
 T2DR = mean(PointsScored)/lgPS,  
 T2PointSpread = mean(PointsAllowed - mean(PointsScored))) %>%  
 dplyr::ungroup()

We calculate these ratings because it gives us a better idea of how good or bad the team was relative to the league’s offensive and defensive environment. The mean for both these ratings is 1, meaning that anybody below 1 is below average, while those above are above average. Point spread will give us the average margin of victory or defeat for each team.

With each team’s OR, DR, and point spread calculated, we’ll merge it back into the master file and calculate the expected results for each match (what a team should have ideally scored given the other team’s neutral stats.)

master <- master %>%   
 dplyr::left\_join(totalTeam, by = c('Season' = 'Season', 'gameId' = 'gameId')) %>%   
 dplyr::select(Season, gameId, DayZero, gameDay, DayNum, WTeamID, WScore, LTeamID, LScore, WLoc, NumOT, gameType, WTeamSeed,  
 LTeamSeed, WRegion, LRegion, lgPS, lgPA, WTeamOR = T1OR, WTeamDR = T1DR, WTeamPointSpread = T1PointSpread,  
 LTeamOR = T2OR, LTeamDR = T2DR, LTeamPointSpread = T2PointSpread) %>%  
 as.data.frame()  
  
head(master)

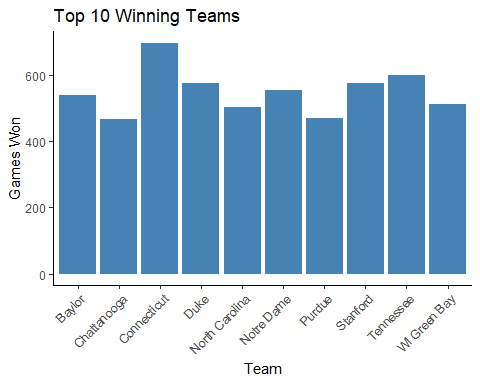
## Season gameId DayZero gameDay DayNum WTeamID WScore  
## 1 1998 1998\_3104\_3202\_1 1997-10-27 1997-11-14 18 3104 91  
## 2 1998 1998\_3163\_3221\_1 1997-10-27 1997-11-14 18 3163 87  
## 3 1998 1998\_3222\_3261\_1 1997-10-27 1997-11-14 18 3222 66  
## 4 1998 1998\_3307\_3365\_1 1997-10-27 1997-11-14 18 3307 69  
## 5 1998 1998\_3349\_3411\_1 1997-10-27 1997-11-14 18 3349 115  
## 6 1998 1998\_3435\_3172\_1 1997-10-27 1997-11-14 18 3435 65  
## LTeamID LScore WLoc NumOT gameType WTeamSeed LTeamSeed WRegion  
## 1 3202 41 H 0 regularSeason <NA> <NA> <NA>  
## 2 3221 76 H 0 regularSeason <NA> <NA> <NA>  
## 3 3261 59 H 0 regularSeason <NA> <NA> <NA>  
## 4 3365 62 H 0 regularSeason <NA> <NA> <NA>  
## 5 3411 35 H 0 regularSeason <NA> <NA> <NA>  
## 6 3172 63 H 0 regularSeason <NA> <NA> <NA>  
## LRegion lgPS lgPA WTeamOR WTeamDR WTeamPointSpread LTeamOR  
## 1 <NA> 75.35126 60.34226 1.0536161 0.9532580 21.86957 1.095036  
## 2 <NA> 75.35126 60.34226 1.1256372 0.9631926 26.69697 1.114476  
## 3 <NA> 75.35126 60.34226 0.9525756 0.9133088 16.66667 1.008138  
## 4 <NA> 75.35126 60.34226 0.9856402 0.8751362 21.46154 1.065352  
## 5 <NA> 75.35126 60.34226 1.0193528 0.9501357 19.47619 0.802242  
## 6 <NA> 75.35126 60.34226 0.9639066 0.9376339 16.05263 1.116548  
## LTeamDR LTeamPointSpread  
## 1 1.1331544 -19.307692  
## 2 1.0268574 -10.125000  
## 3 0.9842790 -13.333333  
## 4 0.9782754 -9.428571  
## 5 1.0791880 -32.909091  
## 6 0.9837260 -6.750000

We finally have a dataset that we can work on. From here on out we’ll do some additional data exploration and start building the model.

# Exploratory data analysis

To get an idea of the strong teams in the league, the top ten teams based on the total amount of matches won are plotted below.

# Generating win count by teams  
teamWins <- count(master, "WTeamID")  
teamWins <- merge(teamWins, teams, by.x = "WTeamID", by.y = "TeamID")  
teamWins <- teamWins[order(-teamWins$freq),]  
teamWins <- teamWins[1:10,]  
  
ggplot(data = teamWins, aes(x = TeamName, y = freq)) +   
 geom\_bar(stat = "identity", fill = "steelblue") +  
 labs(title = "Top 10 Winning Teams", x = "Team", y = "Games Won") +  
 theme\_classic() +  
 theme(axis.text.x = element\_text(angle = 45, hjust = 1))

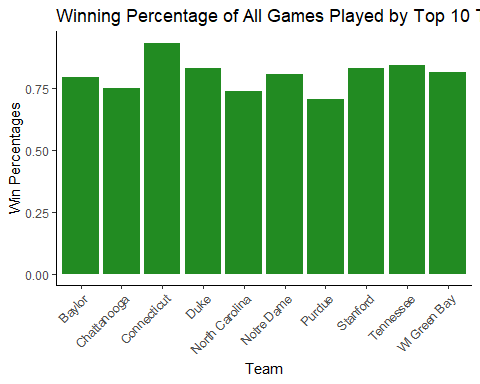


head(teamWins)

## WTeamID freq TeamName  
## 58 3163 696 Connecticut  
## 290 3397 600 Tennessee  
## 283 3390 575 Stanford  
## 76 3181 574 Duke  
## 217 3323 552 Notre Dame  
## 21 3124 539 Baylor

Next, we will explore the winning rates of these top teams.

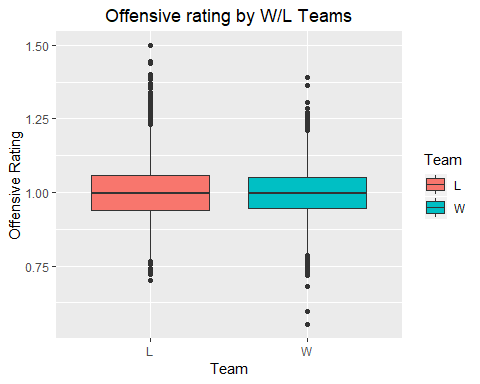
# Generating W/L proportions of top 10 teams  
teamWins <- count(master, "WTeamID")  
teamLoses <- count(master, "LTeamID")  
  
teamWL <- merge(teamWins, teamLoses, by.x = "WTeamID", by.y = "LTeamID")  
teamWL <- teamWL[order(-teamWL$freq.x),]  
teamWL <- teamWL[1:10,]  
  
teamWL <- merge(teamWL, teams, by.x = "WTeamID", by.y = "TeamID")  
teamWL <- teamWL[order(-teamWL$freq.x),]  
  
teamWL$WinPercentage <- (teamWL$freq.x)/(teamWL$freq.x + teamWL$freq.y)  
teamWL$LosePercentage <- (teamWL$freq.y)/(teamWL$freq.x + teamWL$freq.y)  
colnames(teamWL) <- c("TeamID", "Wins", "Loses", "TeamName", "WinPercentage", "LosePercentage")  
  
ggplot(data = teamWL, aes(x = TeamName, y = WinPercentage)) +   
 geom\_bar(stat = "identity", fill = "forestgreen") +  
 labs(title = "Winning Percentage of All Games Played by Top 10 Teams", x = "Team", y = "Win Percentages") +  
 theme\_classic() +  
 theme(axis.text.x = element\_text(angle = 45, hjust = 1))



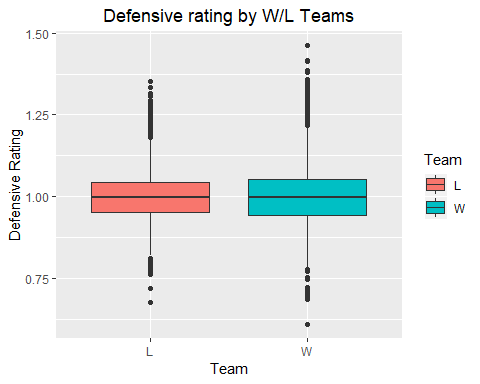
As shown above, the top winning teams all have significantly higher win rate. The winning teams average a 75 winning percentage.

Next, let’s analyze the performance of the winning teams and the losing teams. Offensive ratings and defensive ratings had been calculated for each winning and losing team. These ratings are on a percentage scale, meaning if a team has a performance rating above 1, they are above average, and below average if their rating is under 1.

Wratings <- master %>%   
 dplyr::select(WTeamID, WTeamOR, WTeamDR) %>%   
 dplyr::mutate(TeamFlag = 'W') %>%   
 dplyr::select(TeamFlag, TeamId = WTeamID, TeamOR = WTeamOR, TeamDR = WTeamDR)  
Lratings <- master %>%   
 dplyr::select(LTeamID, LTeamOR, LTeamDR) %>%   
 dplyr::mutate(TeamFlag = 'L') %>%   
 dplyr::select(TeamFlag, TeamId = LTeamID, TeamOR = LTeamOR, TeamDR = LTeamDR)  
  
ratings <- Wratings %>% bind\_rows(Lratings)  
  
# Comparing offensive ratings  
ggplot(data = ratings, aes(x = as.factor(TeamFlag), y = TeamOR, fill = as.factor(TeamFlag))) +   
 geom\_boxplot() +  
 ggtitle("Offensive rating by W/L Teams") +   
 labs(x = 'Team', y = 'Offensive Rating', fill = 'Team') +  
 theme(plot.title = element\_text(hjust = 0.5))



# Comparing defensive ratings  
ggplot(data = ratings, aes(x = as.factor(TeamFlag), y = TeamDR, fill = as.factor(TeamFlag))) +   
 geom\_boxplot() +  
 ggtitle("Defensive rating by W/L Teams") +   
 labs(x = 'Team', y = 'Defensive Rating', fill = 'Team') +  
 theme(plot.title = element\_text(hjust = 0.5))



The boxplots above shown some interesting information about the teams’ performance ratings. First, we want to point out that, on average, losing teams tend to have higher offensive ratings. Perhaps the reason for this is because they are highly aggresive offensively, leaving their defense open and allowing opposing teams to score. In the end, the old mantra might prove to be true: defense wins championships. And, if we look at the boxplot for defensive ratings, we find that indeed the spread of defensive ratings for winning teams is larger, which may indeed support the hypothesis drawn from the first boxplot.

# Modeling

We’ll now turn to building the models to predict which team is likelier to win given it’s OR, DR, and point spread. It’s also time to split the datasets. Because of the size of the data, we’ll only use the years 2008 through 2016 as the training set, and the 2017 data as the testing set.

# The training set will consist of games played before 2017, while the testing set will contain games played in 2017  
trainSet <- master %>%   
 dplyr::filter(Season >= 2008 & Season < 2017)  
  
testSet <- master %>%   
 dplyr::filter(Season == 2017)

Though not a conventional split of the data, we want to see if we are able to predict a season worth of games.

## Linear Regression Model

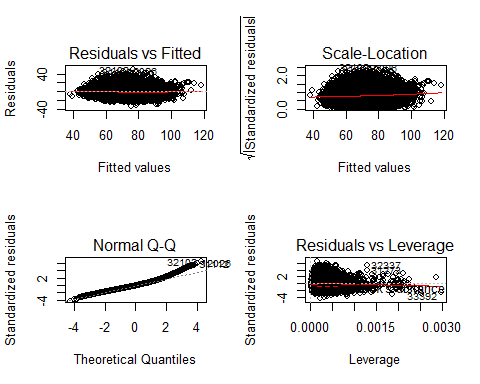
Often, many seek to predict the winning score for a particular match. While not encourage, some people make bets on how much higher a team’s score will be - their spread. Similarly, people generally want to know the winning score beforehand to know what they should expect.

Thus, a linear regression model is built to predict the winning score based on each team’s offensive rating, defensive rating, and point spread, along side the losing team’s ID.

# Linear Regression to predict the winning score  
  
# Fitting the regression  
fit.reg <- lm(WScore ~ LTeamID + LScore + WTeamOR + WTeamDR + LTeamOR + LTeamDR + WTeamPointSpread + LTeamPointSpread, data = trainSet)  
summary(fit.reg)

##   
## Call:  
## lm(formula = WScore ~ LTeamID + LScore + WTeamOR + WTeamDR +   
## LTeamOR + LTeamDR + WTeamPointSpread + LTeamPointSpread,   
## data = trainSet)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -32.602 -5.190 -0.565 4.448 49.595   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) -2.525e+01 1.286e+00 -19.637 < 2e-16 \*\*\*  
## LTeamID -1.595e-03 3.435e-04 -4.643 3.45e-06 \*\*\*  
## LScore 4.986e-01 4.111e-03 121.295 < 2e-16 \*\*\*  
## WTeamOR -6.398e+00 1.812e+01 -0.353 0.72395   
## WTeamDR 3.089e+01 1.450e+01 2.130 0.03318 \*   
## LTeamOR 1.196e+01 1.522e+01 0.786 0.43202   
## LTeamDR 9.750e+00 1.902e+01 0.513 0.60820   
## WTeamPointSpread 1.071e+00 2.558e-01 4.187 2.83e-05 \*\*\*  
## LTeamPointSpread -8.263e-01 2.684e-01 -3.078 0.00208 \*\*   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 7.759 on 47059 degrees of freedom  
## Multiple R-squared: 0.534, Adjusted R-squared: 0.5339   
## F-statistic: 6741 on 8 and 47059 DF, p-value: < 2.2e-16

# Visualizing some detail about the regression  
layout(matrix(c(1,2,3,4),2,2))  
plot(fit.reg)



As shown from the regression summary, all variable are significant in determining the winning score of the match. It’s important to point out that the winning team’s offensive rating is the least significant of the coefficients, while the winning team’s defensive rating, and the losing team’s offensive rating seem to be more predictive of the final score. From the residual plots, we can see that the data points in the regression are randomly distributed, which implies that the variables are not correlated.

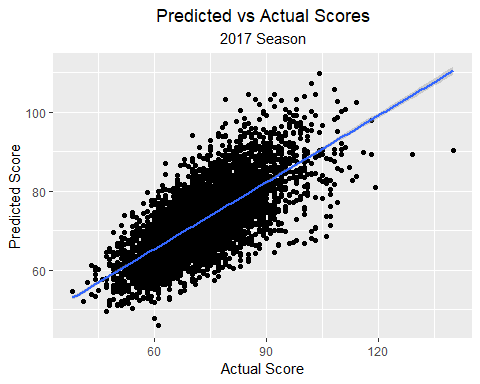
However, it must be noted that the adjusted R-square is farily low for the model at 0.5339. While this is not a high R-square value, it is understandable that the variables in the regression only account for 53.39% of the winning score. Sports are highly variable in nature, and are therefore hard to predict. Thus, the R-square value is acceptable in this instance.

However, let’s see how the actual scores compare to the predicted scores.

predictLm <- data.frame('predict' = predict(fit.reg, newdata = testSet))  
LmResults <- testSet %>%   
 dplyr::bind\_cols(predictLm)  
  
# Root mean squeared error   
RMSE(LmResults$predict, LmResults$WScore)

## [1] 7.899277

# Plot the results for comparison  
ggplot(LmResults, aes(x = WScore, y = predict)) +  
 geom\_point() +   
 geom\_smooth(method = 'lm') +   
 labs(x = 'Actual Score', y = 'Predicted Score') +  
 ggtitle('Predicted vs Actual Scores', subtitle = '2017 Season') +  
 theme(plot.title = element\_text(hjust = 0.5),  
 plot.subtitle = element\_text(hjust = 0.5))



The root mean squared error shows that the the WScore is overall off by 7.9 points in either direction which, as we can, see by the graph, is not bad. There’s a cluster of scores and a positive relationship when comparing these scores. The model’s predictions appear to be in line with the actual score but they are not 100 percent spot on. We’ll see if we can do better using other methods.

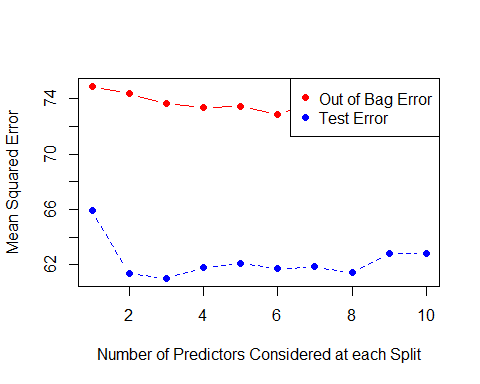
## Random Forest

Let’s try to improve the model by building a random forest. We’ll first set the training parameters to check the number of variables that we’ll need and the number of trees that we’ll build for the algorithm.

oob.err <- double(10)  
test.err <- double(10)  
  
for(mtry in 1:10){  
 rf <- randomForest(WScore ~ LTeamID + LScore + WTeamOR + WTeamDR + LTeamOR + LTeamDR + WTeamPointSpread + LTeamPointSpread + lgPS + lgPA, data = trainSet, mtry = mtry, ntree = 10)   
 #Error of all trees  
 oob.err[mtry] = rf$mse[10]   
   
 #Predictions on Test Set for each Tree  
 pred <- predict(rf, testSet)   
 test.err[mtry]= with(testSet, mean((WScore - pred)^2))  
}

Checking this plot, we can see that the number of variables is ideally 6.

matplot(1:mtry , cbind(oob.err,test.err), pch=19 , col=c("red", "blue"), type="b", ylab = "Mean Squared Error", xlab = "Number of Predictors Considered at each Split")   
legend("topright", legend = c("Out of Bag Error", "Test Error"), pch = 19, col = c("red", "blue"))



We’ll once again train the model, accounting for mtry = 6, and use that value to generate the random forest. However, we will now grow 100 trees instead of ten.

rfTrain <- randomForest(WScore ~ LTeamID + LScore + WTeamOR + WTeamDR + LTeamOR + LTeamDR + WTeamPointSpread + LTeamPointSpread + lgPS + lgPA, data = trainSet, mtry = 6, ntree = 100)  
  
rfTrain

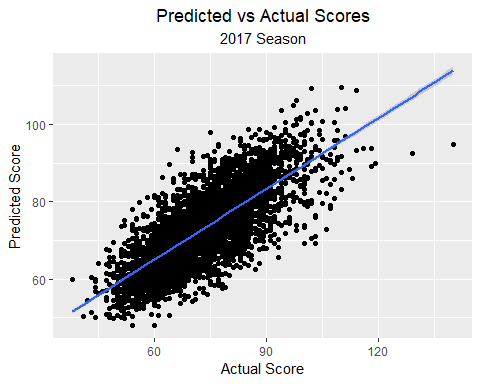
##   
## Call:  
## randomForest(formula = WScore ~ LTeamID + LScore + WTeamOR + WTeamDR + LTeamOR + LTeamDR + WTeamPointSpread + LTeamPointSpread + lgPS + lgPA, data = trainSet, mtry = 6, ntree = 100)   
## Type of random forest: regression  
## Number of trees: 100  
## No. of variables tried at each split: 6  
##   
## Mean of squared residuals: 57.93208  
## % Var explained: 55.15

In this case, the random forest regression is tells us that the variables chosen for the tree explain 55.16% of the variability in the model - a slight improvement over the previous model. Let’s check with the test set.

rfPred <- predict(rfTrain, newdata = testSet)  
rfPred <- data.frame('predict' = rfPred)  
  
rfResults <- testSet %>% bind\_cols(rfPred)  
  
# And now to check the RMSE and the plot  
RMSE(rfResults$predict, rfResults$WScore)

## [1] 7.53142

ggplot(rfResults, aes(x = WScore, y = predict)) +  
 geom\_point() +   
 geom\_smooth(method = 'lm') +   
 labs(x = 'Actual Score', y = 'Predicted Score') +  
 ggtitle('Predicted vs Actual Scores', subtitle = '2017 Season') +  
 theme(plot.title = element\_text(hjust = 0.5),  
 plot.subtitle = element\_text(hjust = 0.5))



The plot seems to be almost identical, but the RMSE shows us a slight improvement of four percent, reducing the margin of error from 7.9 to 7.58.

## Support Vector Machine

We’ll try one last model to see whether we can improve our predictions a bit more.

svmTrain <- svm(WScore ~ LTeamID + LScore + WTeamOR + WTeamDR + LTeamOR + LTeamDR + WTeamPointSpread + LTeamPointSpread + lgPS + lgPA, data = trainSet, type = 'eps', kernel = 'linear', cross = 3, probability = TRUE)

## Warning in cret$cresults \* scale.factor: Recycling array of length 1 in vector-array arithmetic is deprecated.  
## Use c() or as.vector() instead.

summary(svmTrain)

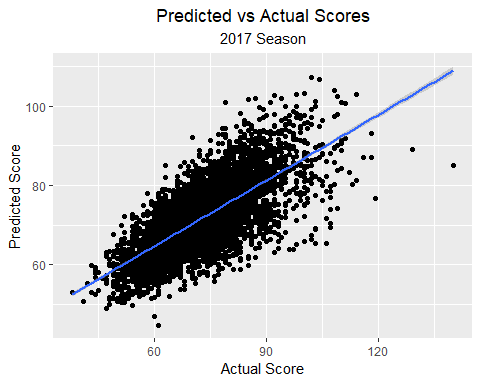
##   
## Call:  
## svm(formula = WScore ~ LTeamID + LScore + WTeamOR + WTeamDR +   
## LTeamOR + LTeamDR + WTeamPointSpread + LTeamPointSpread +   
## lgPS + lgPA, data = trainSet, type = "eps", kernel = "linear",   
## cross = 3, probability = TRUE)  
##   
##   
## Parameters:  
## SVM-Type: eps-regression   
## SVM-Kernel: linear   
## cost: 1   
## gamma: 0.1   
## epsilon: 0.1   
##   
## Sigma: 0.5199105   
##   
##   
## Number of Support Vectors: 40945  
##   
##   
##   
## 3-fold cross-validation on training data:  
##   
## Total Mean Squared Error: 60.85354   
## Squared Correlation Coefficient: 0.533058   
## Mean Squared Errors:  
## 61.82611 58.69903 62.03541

The model looks good. Time to check the results on the testing set.

testSet2 <- testSet2 <- testSet %>% dplyr::select(LTeamID, LScore, WTeamOR, WTeamDR, LTeamOR, LTeamDR, WTeamPointSpread, LTeamPointSpread, lgPS, lgPA)  
svmPred <- predict(svmTrain, newdata = testSet2)  
svmPred <- data.frame('predict' = svmPred)  
  
svmResults <- testSet %>% bind\_cols(svmPred)  
  
RMSE(svmResults$predict, svmResults$WScore)

## [1] 7.960987

ggplot(svmResults, aes(x = WScore, y = predict)) +  
 geom\_point() +   
 geom\_smooth(method = 'lm') +   
 labs(x = 'Actual Score', y = 'Predicted Score') +  
 ggtitle('Predicted vs Actual Scores', subtitle = '2017 Season') +  
 theme(plot.title = element\_text(hjust = 0.5),  
 plot.subtitle = element\_text(hjust = 0.5))



It turns out that the SVM is the worst of the three models because, even though it seems as it predicts the same results as the random forest and linear regression, for higher scores, the predictions become more volatile.

# Conclusions

Though these are just three of the models that we tested, we talked about building more models that could potentially improve the prediction. Models discussed were the popular Kaggle algorithms Generalized Additive Models and eXtreme Gradient Boosting, but due to time constraints we couldn’t use them. However, we believe that it would be interesting to revisit these algorithms in the future. Another conclusion that we’ve drawn is that, even though we have a lot of data, it would be interesting to see other additions to the dataset. Shooting percentage, fouls per game, number of triples and rebounds, are among the statistics that could improve the model, as they could give us a clearer idea on how a team both attacks and defends. Including more data into the training of the model could increase the accuracy of our results. Finally, of our three models, we were surprised that the random forest model was the most succesful of the three. Initially, we thought the best model would be the SVM, followed closely by the linear regression. It turns out that it was the other way round, with the SVM having a RMSE of 7.96, the linear regression a RMSE of 7.89, and the random forest a RMSE of 7.58. However, this is only part of what ideally we would’ve want to achieve since, what is more useful is to determine which side actually won and this is what in the end should be our aim. For the meantime, we are content that we could predict the winning team’s score to within 7.5 points of their actual score.