IST 707 Data Analytics – Final Project

NCAA Basketball Tournament – Predicting 2017 wining scores

Authors: Prasad Kulkarni Michael Jung

Date: 06/09/2020

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 Tournament.

The Dataset

The dataset was obtained 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. 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.

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. We believe that by using three different models 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 the data files & Data Preparations.

We will create a master file by combining selected columns from multiple CSV files like Regular season, tournaments, teams, cities etc. Once the master file is 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.

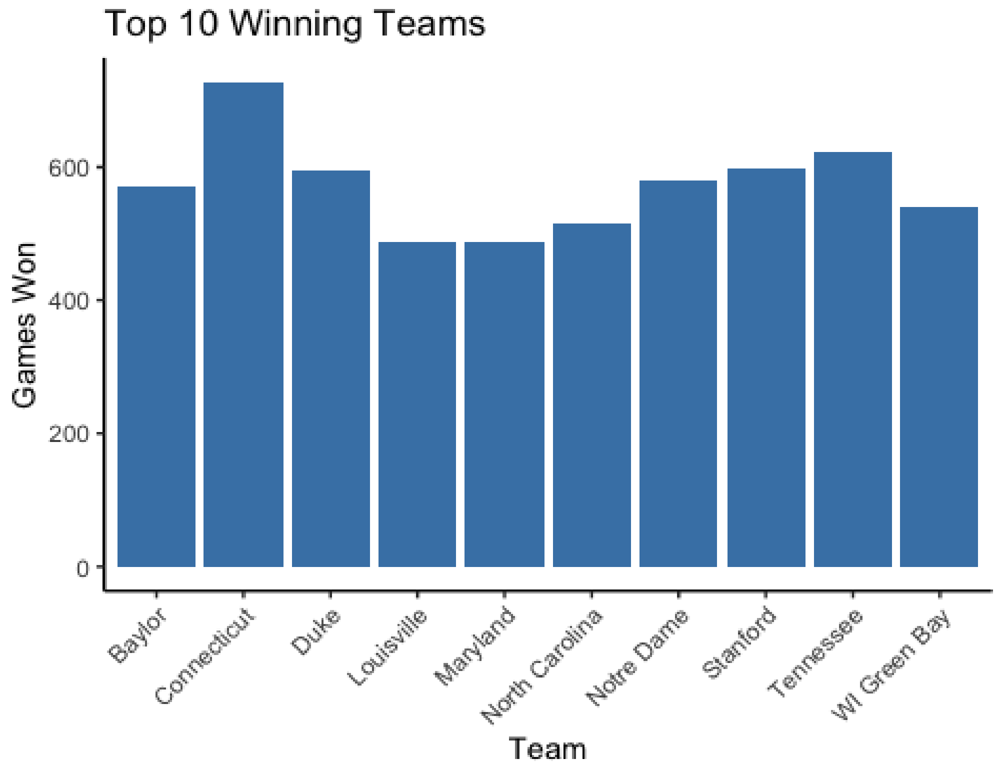
We start by selecting the winning team, their score, and the score of the losing team. These last two will be the points scored and points allowed and will be used to calculate each team's offensive rating (OR) and defensive rating (DR).

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.

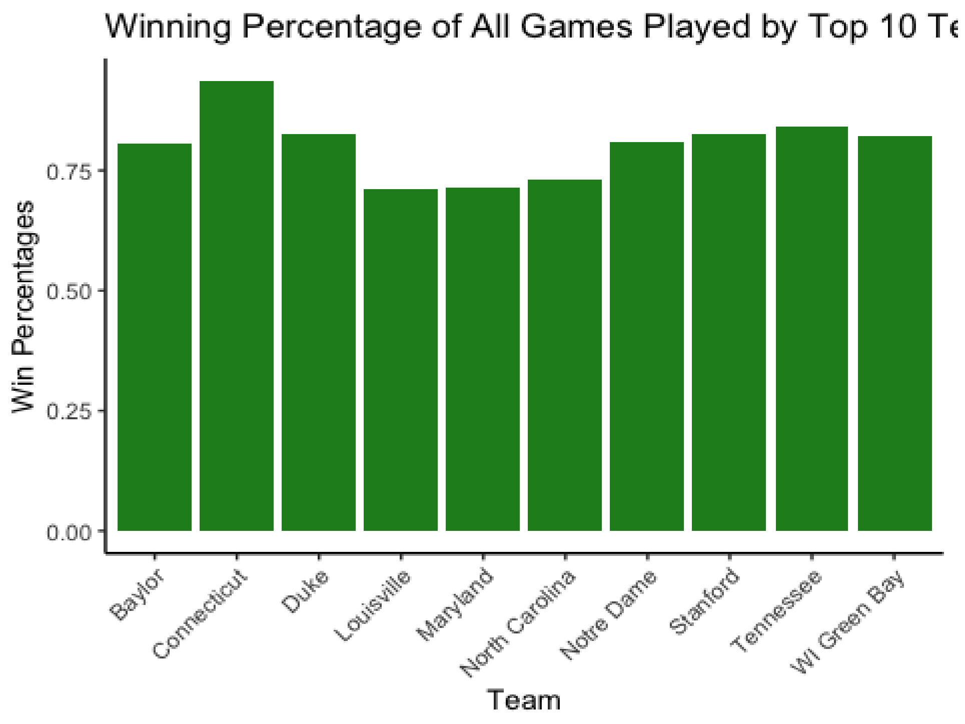
With each team's OR, DR, and point spread calculated, we'll merge it back into the master file and calculate the expected results for each match (what a team should have ideally scored given the other team's neutral stats.)

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.

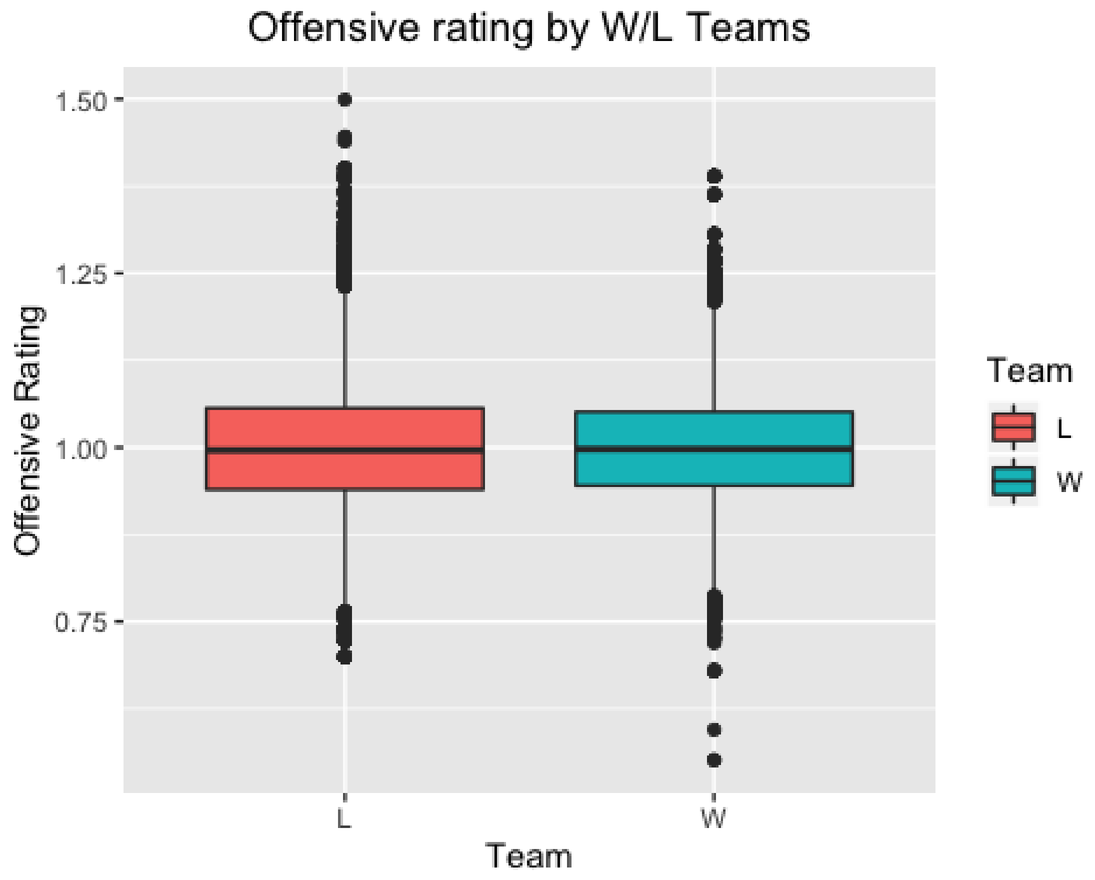


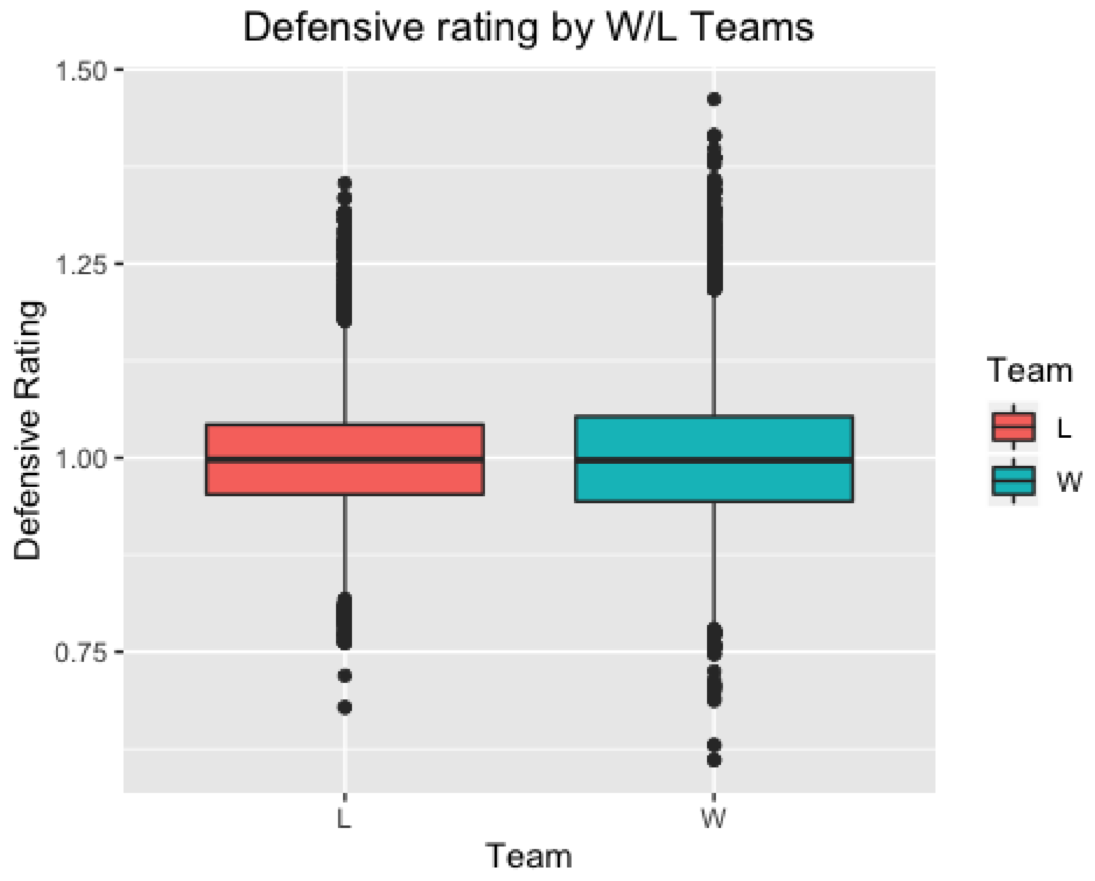
Next, we will explore the winning rates of these top teams.



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.


Offensive rating by W/L Teams
1.50
1.25
1.00
0.75
Offensive Rating
L
W
Team
Team
L
W


Defensive rating by W/L Teams
1.50
1.25
1.00
0.75
Defensive Rating
L
W
Team
Team
L
W

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 aggressive 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. Looking 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 offensive rating boxplot.

Modeling

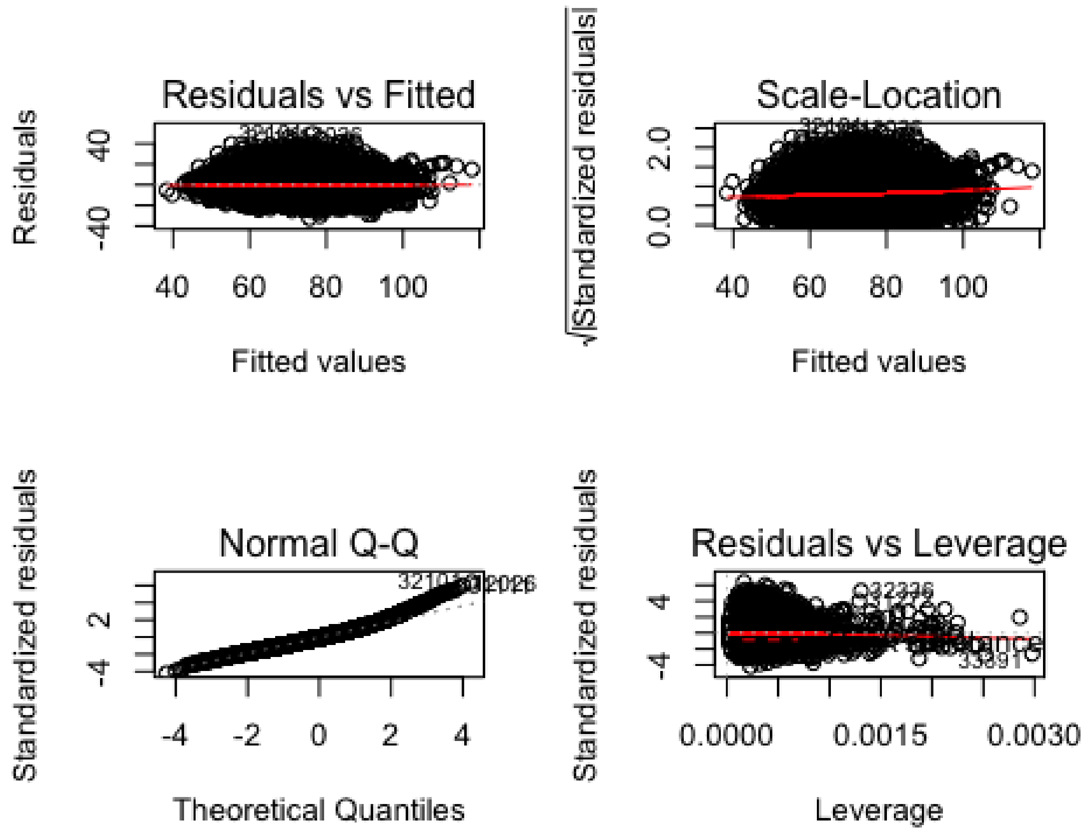
We'll now turn to building the models to predict which team is likelier to win given it's OR, DR, and point spread. It's also time to split the datasets. Because of the size of the data, we'll only use the years 2008 through 2016 as the training set, and the 2017 data as the testing set.

We will use Linear regression, Random Forest and SVM to predict the winning score and then use decision tree to outline the factors of winning teams.

Linear Regression Model

Often, many seek to predict the winning score for a particular match. While not encouraged, 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.

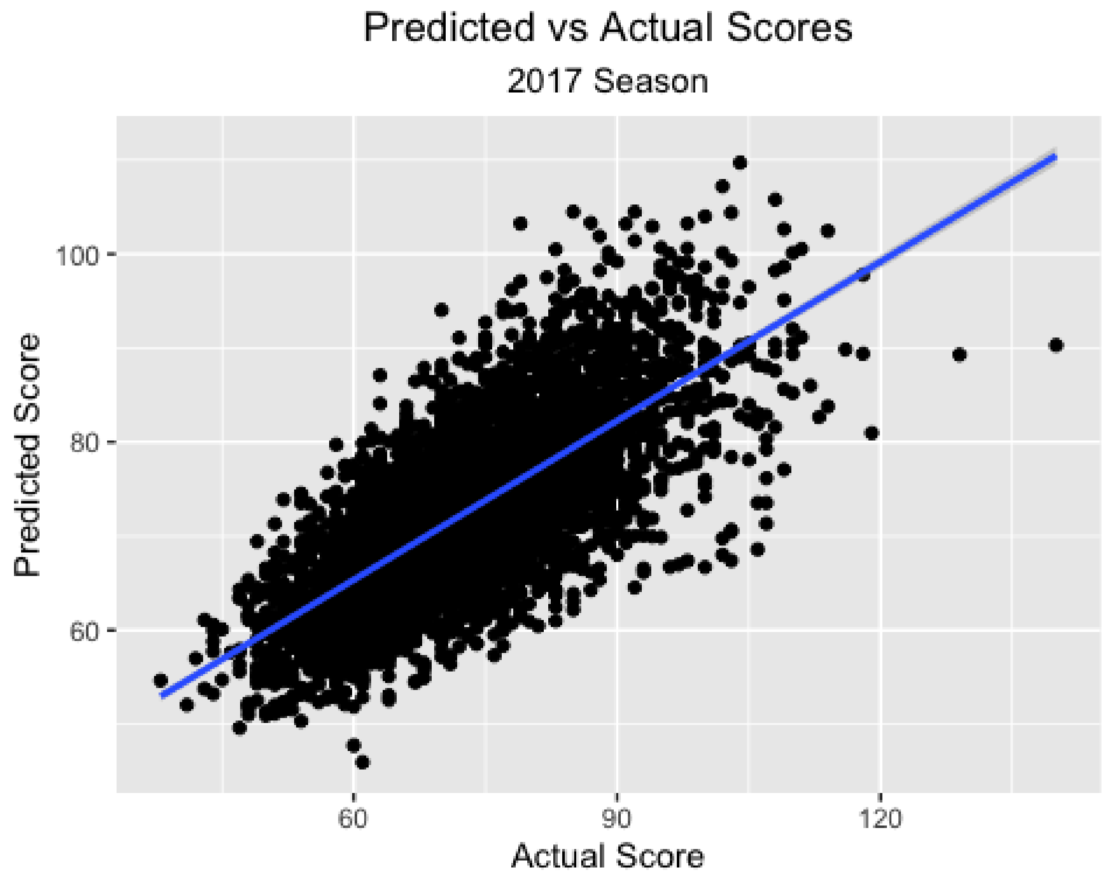
Thus, a linear regression model is built to predict the winning score based on each team's offensive rating, defensive rating, and point spread, alongside the losing team's ID.



As shown from the regression summary, all selected variables 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 autocorrelated.

However, it must be noted that the adjusted R-square is fairly 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.

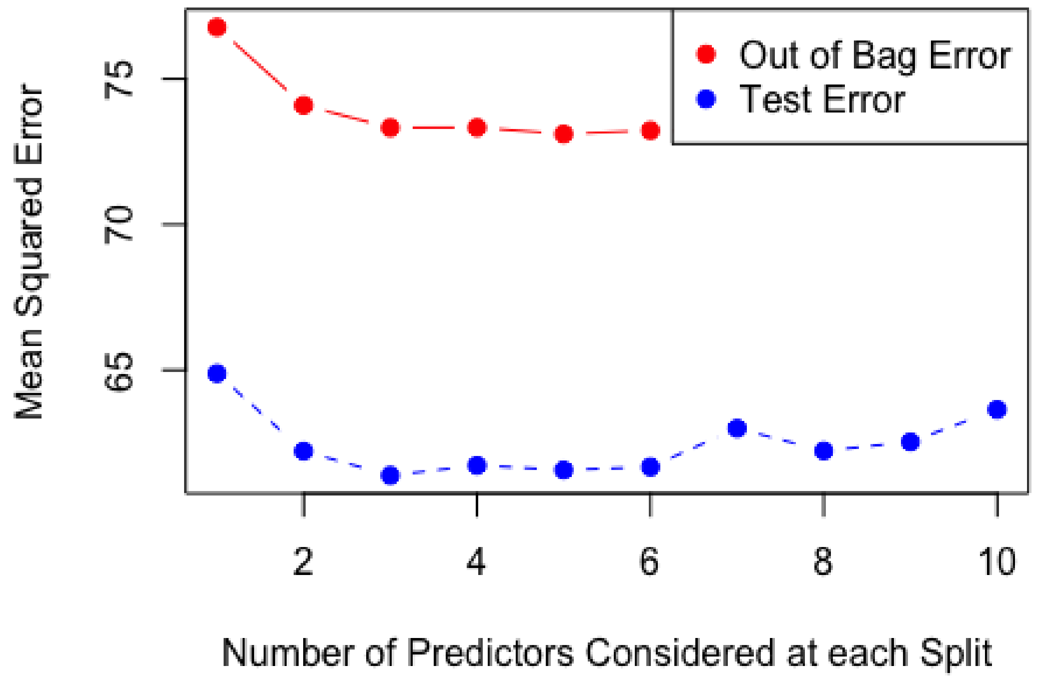
However, let's see how the actual scores compare to the predicted scores.



The root mean squared error shows that the winning scor (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 scores.

Random Forest

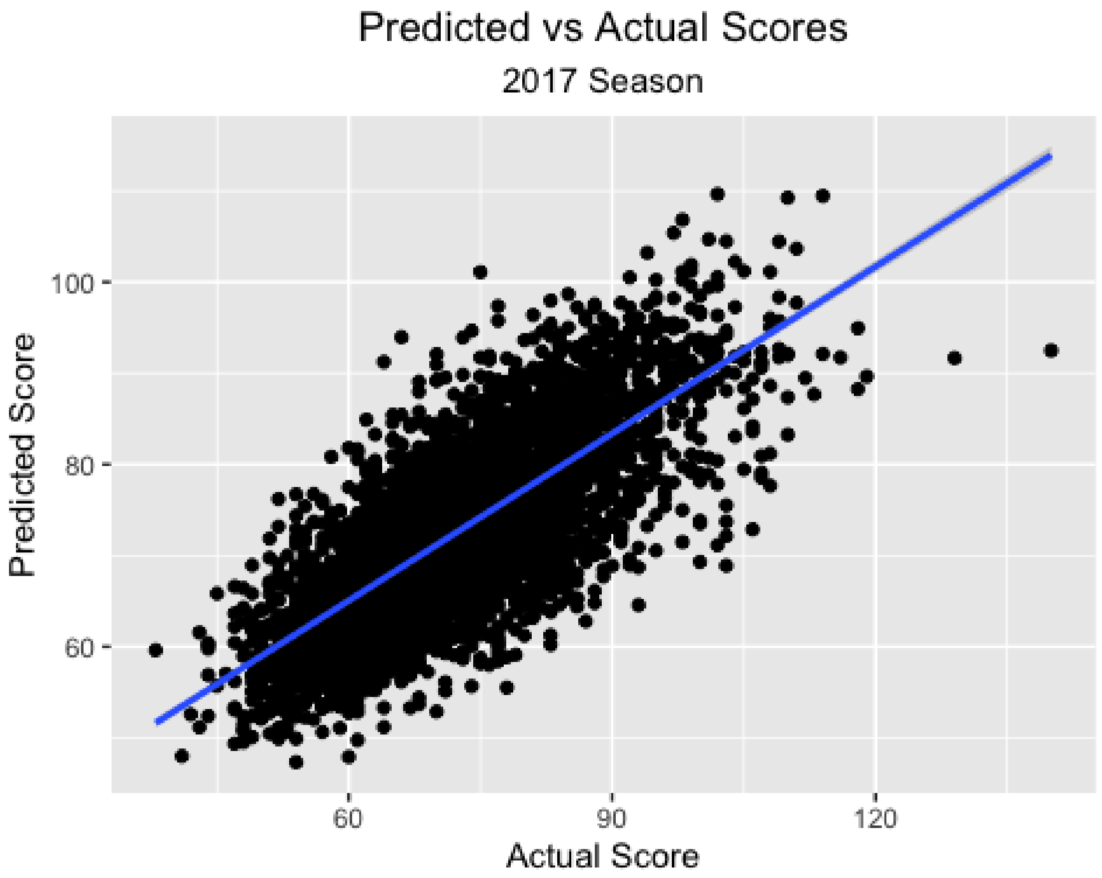
Let's try to improve the model by building a random forest. We'll first set the training parameters to check the number of variables that we'll need and the number of trees that we'll build for the algorithm. Checking this plot, we can see that the number of variables is ideally 6.



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

```
##                Type of random forest: regression
##                      Number of trees: 100
## No. of variables tried at each split: 6
##
##           Mean of squared residuals: 57.72605
##                     % Var explained: 55.31
```

In this case, the random forest regression tells us that the variables chosen for the tree explain 55.31% of the variability in the model - a slight improvement over the linear regression model. Let's check with the test set.



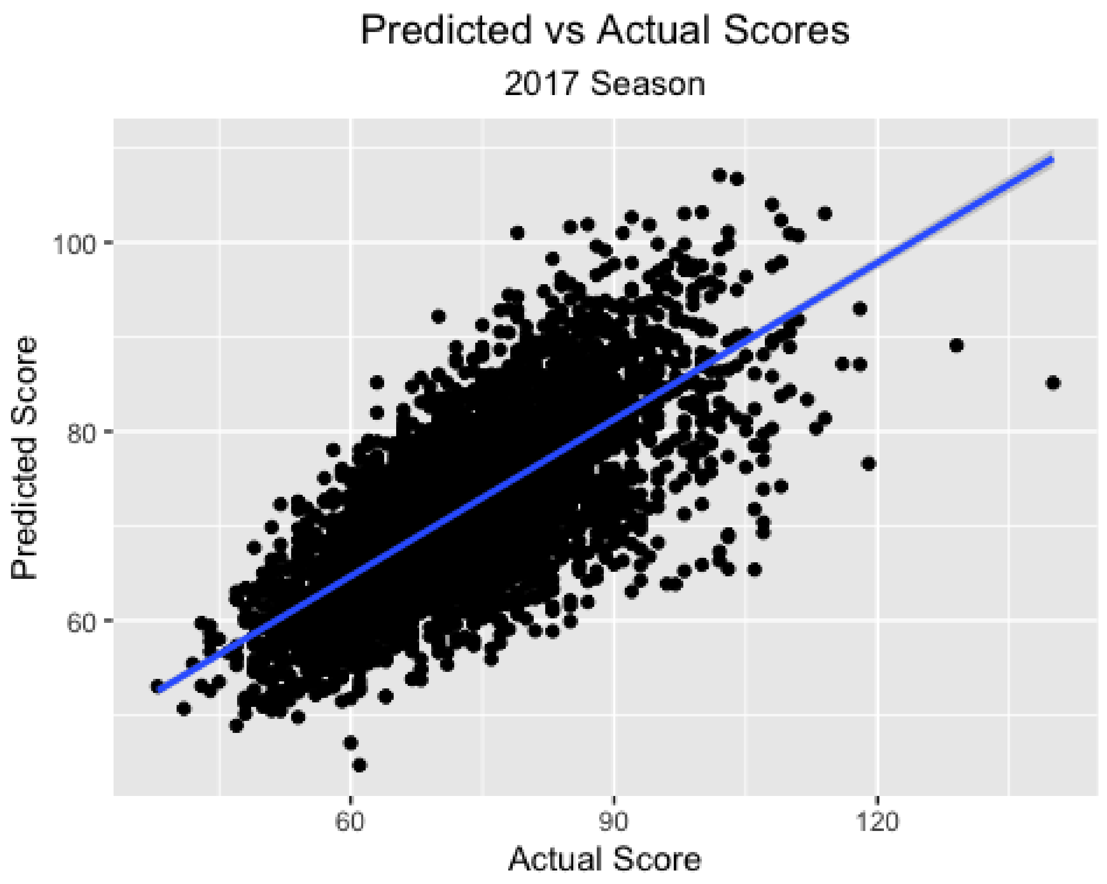
The plot seems to be almost identical, but the RMSE shows us a slight improvement of four percent, reducing the margin of error from 7.9 to 7.54.

Support Vector Machine

We'll try another model to see whether we can improve our predictions a bit more.

```
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  linear
##        cost:  1
##       gamma:  0.1
##     epsilon:  0.1
##
## Sigma:  0.5199067
##
##
## Number of Support Vectors:  40946
##
##
##
## 3-fold cross-validation on training data:
##
## Total Mean Squared Error: 60.83469
## Squared Correlation Coefficient: 0.5331637
## Mean Squared Errors:
##  60.59914 61.03063 60.8743
```

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



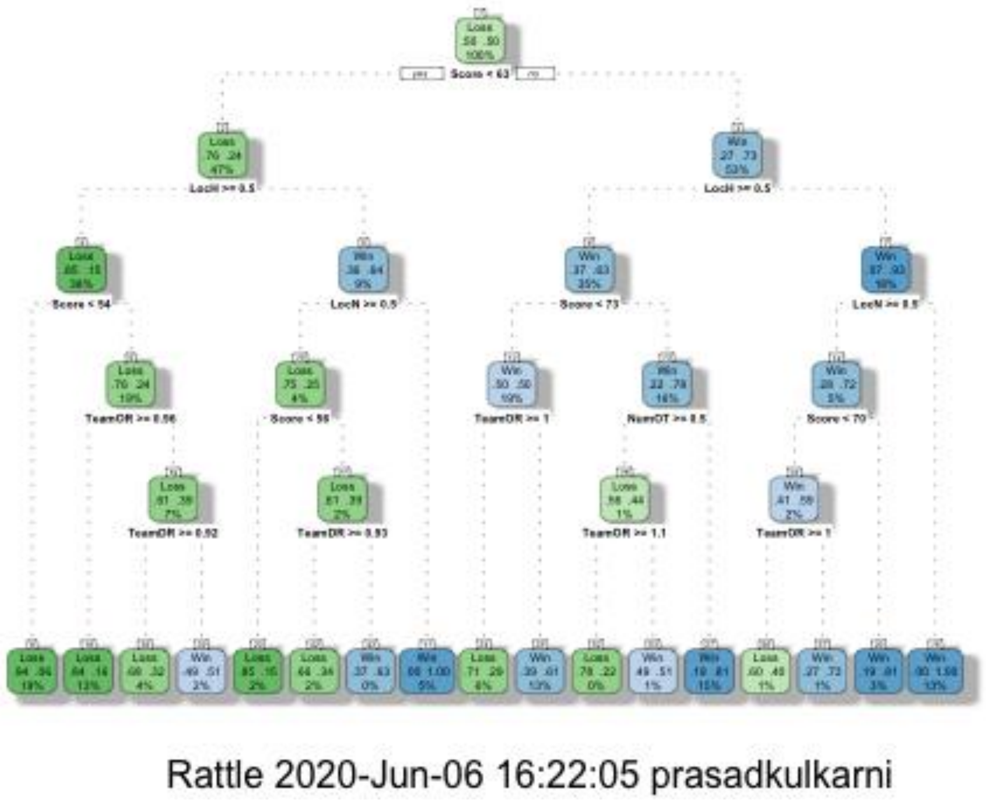
SVM is the least effective of the three models (with RMSE 7.96). Although it seems as if it predicts the same results as the random forest and linear regression, the predictions become more volatile for higher scores.

Decision Tree

However, this is only part of what ideally we would 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. Let's predict result by using the decision tree model

For this model we will further split our training data into two sets 1) 2008-2017 regular season training data 2) 2008-2016 Tournament training data. The 2017 tournament results will be used as the testing set.

Data Model using all regular season games:



Predicting outcome of 2017 tournament using this data model

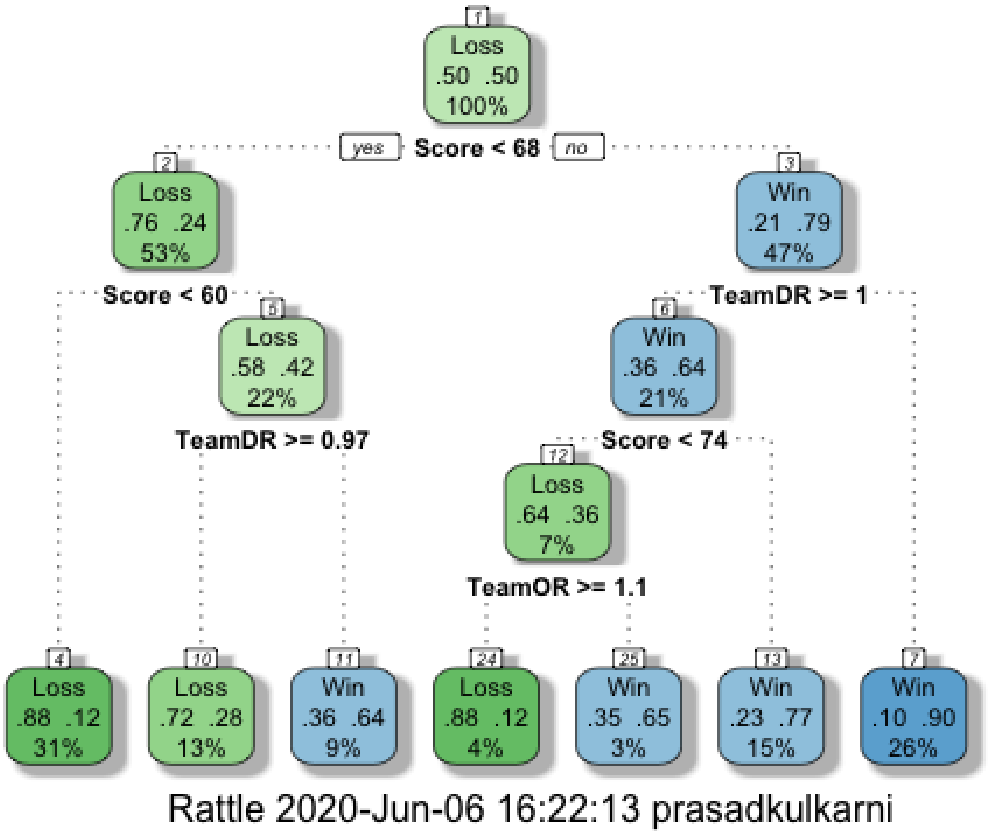
```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction Loss Win
##       Loss   50  11
##       Win    13  52
##
##                Accuracy : 0.8095
##                  95% CI : (0.73, 0.874)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 6.067e-13
##
##                   Kappa : 0.619
##
##  Mcnemar's Test P-Value : 0.8383
##
##             Sensitivity : 0.7937
##             Specificity : 0.8254
##          Pos Pred Value : 0.8197
##          Neg Pred Value : 0.8000
##              Prevalence : 0.5000
##          Detection Rate : 0.3968
##    Detection Prevalence : 0.4841
##       Balanced Accuracy : 0.8095
```

```
##
##        'Positive' Class : Loss
##
```

Using the regular season training set the model predicted 81% of the games correctly. Final Score, Home Games and the Offensive Rating of the teams were the three most important variables in determining the outcome of the game.

In this model using regular season data, a score of 62.5 was used to determine the winner and loser at the first branch in the decision tree

Data Model using all tournament games:



Predicting outcome of 2017 tournament using this data model

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Loss Win
##       Loss   49   9
##       Win    14  54
##
##                Accuracy : 0.8175
##                  95% CI : (0.7388, 0.8806)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 1.393e-13
##
##                   Kappa : 0.6349
##
##  Mcnemar's Test P-Value : 0.4042
##
##             Sensitivity : 0.7778
##             Specificity : 0.8571
##          Pos Pred Value : 0.8448
##          Neg Pred Value : 0.7941
##              Prevalence : 0.5000
##          Detection Rate : 0.3889
##    Detection Prevalence : 0.4603
##       Balanced Accuracy : 0.8175
##
##        'Positive' Class : Loss
##
```

Using the 2008-2016 tournament training set the model predicted 82% of the 2017 tournament games correctly. Final Score, Offensive Rating and the Defensive Rating of the teams were the three most important variables in determining the outcome of the game. I is worth mentioning that home team advantage was not as important for tournament games as it was for regular season games.

In this second model, using only tournament data, a score of 67.5 was utilized for the first branch of the decision tree.

Conclusions

Out of the three models used for predicting the winning score Random forest seem to work best with lowest RMSE value of 7.54 SVM had a RMSE of 7.96, with the linear regression having a RMSE of 7.89.

While predicting the winning teams using decision tree model, we could achieve 82% accuracy by using only tournament data to train the model.

The final conclusion we've drawn is that, even though we have utilized a lot of data for our analysis, it would be interesting to see the impact of 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 would likely increase the accuracy of our results.

Appendix

R code used for data preparation, modelling, analysis and plotting

```
# Load csv files
cities <- read.csv('/Users/prasadkulkarni/Documents/Syracuse
University/IST707/Assignments/Project Proposal/Data/WCities.csv')
gameCities <- read.csv('/Users/prasadkulkarni/Documents/Syracuse
University/IST707/Assignments/Project Proposal/Data/WGameCities.csv')
compactResults <- read.csv('/Users/prasadkulkarni/Documents/Syracuse
University/IST707/Assignments/Project
Proposal/Data/WNCAATourneyCompactResults.csv')
seeds <- read.csv('/Users/prasadkulkarni/Documents/Syracuse
University/IST707/Assignments/Project Proposal/Data/WNCAATourneySeeds.csv')
slots <- read.csv('/Users/prasadkulkarni/Documents/Syracuse
University/IST707/Assignments/Project Proposal/Data/WNCAATourneySlots.csv')
regSeasonResults <- read.csv('/Users/prasadkulkarni/Documents/Syracuse
University/IST707/Assignments/Project
Proposal/Data/WRegularSeasonCompactResults.csv')
season <- read.csv('/Users/prasadkulkarni/Documents/Syracuse
University/IST707/Assignments/Project Proposal/Data/WSeasons.csv')
teams <- read.csv('/Users/prasadkulkarni/Documents/Syracuse
University/IST707/Assignments/Project Proposal/Data/WTeams.csv')
teamSpellings <- read.csv('/Users/prasadkulkarni/Documents/Syracuse
University/IST707/Assignments/Project Proposal/Data/WTeamSpellings.csv')

# 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
```

```r
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"

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()

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
LTeamID
## 1   1998 1998_3104_3202_1 1997-10-27 1997-11-14     18    3104     91
3202
## 2   1998 1998_3163_3221_1 1997-10-27 1997-11-14     18    3163     87
3221
## 3   1998 1998_3222_3261_1 1997-10-27 1997-11-14     18    3222     66
3261
## 4   1998 1998_3307_3365_1 1997-10-27 1997-11-14     18    3307     69
3365
## 5   1998 1998_3349_3411_1 1997-10-27 1997-11-14     18    3349    115
3411
## 6   1998 1998_3435_3172_1 1997-10-27 1997-11-14     18    3435     65
3172
##   LScore WLoc NumOT      gameType WTeamSeed LTeamSeed WRegion LRegion
lgPS
## 1     41    H     0 regularSeason      <NA>      <NA>    <NA>    <NA>
75.35126
## 2     76    H     0 regularSeason      <NA>      <NA>    <NA>    <NA>
75.35126
## 3     59    H     0 regularSeason      <NA>      <NA>    <NA>    <NA>
75.35126
## 4     62    H     0 regularSeason      <NA>      <NA>    <NA>    <NA>
75.35126
## 5     35    H     0 regularSeason      <NA>      <NA>    <NA>    <NA>
75.35126
## 6     63    H     0 regularSeason      <NA>      <NA>    <NA>    <NA>
75.35126
```

```
##       lgPA   WTeamOR   WTeamDR WTeamPointSpread  LTeamOR   LTeamDR
## 1 60.34226 1.0536161 0.9532580         21.86957 1.095036 1.1331544
## 2 60.34226 1.1256372 0.9631926         26.69697 1.114476 1.0268574
## 3 60.34226 0.9525756 0.9133088         16.66667 1.008138 0.9842790
## 4 60.34226 0.9856402 0.8751362         21.46154 1.065352 0.9782754
## 5 60.34226 1.0193528 0.9501357         19.47619 0.802242 1.0791880
## 6 60.34226 0.9639066 0.9376339         16.05263 1.116548 0.9837260
##   LTeamPointSpread
## 1       -19.307692
## 2       -10.125000
## 3       -13.333333
## 4        -9.428571
## 5       -32.909091
## 6        -6.750000

# 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  728 Connecticut
## 290    3397  624   Tennessee
## 283    3390  597    Stanford
## 76     3181  596        Duke
## 217    3323  581  Notre Dame
## 21     3124  570      Baylor

# 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")
```

```r
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))

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 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)

# 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
## -32.603  -5.189  -0.564   4.449  49.594
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)      -2.526e+01  1.286e+00 -19.641  < 2e-16 ***
## LTeamID          -1.593e-03  3.435e-04  -4.638 3.53e-06 ***
## LScore            4.986e-01  4.111e-03 121.288  < 2e-16 ***
## WTeamOR          -6.616e+00  1.812e+01  -0.365  0.71502
## WTeamDR           3.106e+01  1.451e+01   2.141  0.03226 *
## LTeamOR           1.222e+01  1.523e+01   0.803  0.42217
## LTeamDR           9.420e+00  1.903e+01   0.495  0.62059
## WTeamPointSpread  1.074e+00  2.559e-01   4.198 2.70e-05 ***
## LTeamPointSpread -8.310e-01  2.685e-01  -3.094  0.00197 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.759 on 47058 degrees of freedom
## Multiple R-squared:  0.534,  Adjusted R-squared:  0.5339
## F-statistic:  6741 on 8 and 47058 DF,  p-value: < 2.2e-16

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

predictLm <- data.frame(predict = predict(fit.reg, newdata = testSet))
LmResults <- testSet %>% dplyr::bind_cols(predictLm)

# Root mean squared error
RMSE(LmResults$predict, LmResults$WScore)

## [1] 7.899331

# 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))

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))
}

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"))

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: 57.72605
##                     % Var explained: 55.31

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] 7.547944

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))
```

```r
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:  0.5199067
##
##
## Number of Support Vectors:  40946
##
##
##
## 3-fold cross-validation on training data:
##
## Total Mean Squared Error: 60.83469
## Squared Correlation Coefficient: 0.5331637
## Mean Squared Errors:
##  60.59914 61.03063 60.8743

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] 7.96071
```

```
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))

# Create a Decision Tree to Determine the winner of the 2017 tournament

# Install the partykit library to create fancy decision trees.
library(partykit)

## Loading required package: grid

## Loading required package: libcoin

## Loading required package: mvtnorm

library(rattle)

## Loading required package: tibble

## Loading required package: bitops

## Rattle: A free graphical interface for data science with R.
## Version 5.4.0 Copyright (c) 2006-2020 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.

##
## Attaching package: 'rattle'

## The following object is masked from 'package:randomForest':
##
##     importance

library(rpart)

# Process the master file to prepare for decision tree analysis Split the
data
# set to put winning team data and losing team data on separate lines Create
the
# Win team lines
dt_master_data1 <- master[, c("Season", "WTeamID", "WScore", "WLoc", "NumOT",
"lgPS",
    "lgPA", "WTeamOR", "WTeamDR", "gameType")]
names(dt_master_data1) <- c("Season", "Team", "Score", "Loc", "NumOT",
"lgPS", "lgPA",
    "TeamOR", "TeamDR", "gameType")
dt_master_data1$Result <- "Win"

# Create the Loss team rows
dt_master_data2 <- master[, c("Season", "LTeamID", "LScore", "WLoc", "NumOT",
"lgPS",
    "lgPA", "LTeamOR", "LTeamDR", "gameType")]
names(dt_master_data2) <- c("Season", "Team", "Score", "Loc", "NumOT",
"lgPS", "lgPA",
    "TeamOR", "TeamDR", "gameType")
dt_master_data2$Result <- "Loss"
# Swap home and away team for losing team (only winning team location was
given)
dt_master_data2$Loc[dt_master_data2$Loc == "H"] <- "A"
dt_master_data2$Loc[dt_master_data2$Loc == "A"] <- "H"

# Combine winning and losing team rows
dt_master_data <- rbind(dt_master_data1, dt_master_data2)

# Convert columns to factors, where applicable. dt_master_data$Season =
# as.factor(dt_master_data$Season)
dt_master_data$Team = as.factor(dt_master_data$Team)
dt_master_data$Loc = as.factor(dt_master_data$Loc)
dt_master_data$Result = as.factor(dt_master_data$Result)
dt_master_data$gameType = as.factor(dt_master_data$gameType)

# Split into training and testing data sets Train Set 1 is all regular season
# games 2008-2017
dt_train_data1 <- dt_master_data %>% dplyr::filter(Season >= 2008 & Season <=
2017 &
    gameType == "regularSeason")

# Train Set 2 is all tournament games 2008-2016
dt_train_data2 <- dt_master_data %>% dplyr::filter(Season >= 2008 & Season <=
2016 &
    gameType == "tournament")

# Test Set 1 is all 2017 tournament games
dt_test_data1 <- dt_master_data %>% dplyr::filter(Season == 2017 & gameType
== "tournament")


# Create the model using all regular season games 2008-2017
Tgrid <- expand.grid(cp = seq(0, 0.02, 0.01))
Tcontrol <- rpart.control(minsplit = 20, maxdepth = 5)

dt_model1 <- train(Result ~ ., data = dt_train_data1, method = "rpart",
metric = "Accuracy",
    tuneGrid = Tgrid, control = Tcontrol)

# plot model variable importance
importance <- varImp(dt_model1)
plot(importance, top = 5)

# Plot the model accuracy vs complexity parameter
plot(dt_model1)

# Create a fancy plot of the decision tree
fancyRpartPlot(dt_model1$finalModel)
```

```
# Predict the outcomes for the 2017 tournament
dt_model1_predict <- predict(dt_model1, dt_test_data1)

confusionMatrix(dt_model1_predict, dt_test_data1$Result)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction Loss Win
##       Loss   50  11
##       Win    13  52
##
##                Accuracy : 0.8095
##                  95% CI : (0.73, 0.874)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 6.067e-13
##
##                   Kappa : 0.619
##
##  Mcnemar's Test P-Value : 0.8383
##
##             Sensitivity : 0.7937
##             Specificity : 0.8254
##          Pos Pred Value : 0.8197
##          Neg Pred Value : 0.8000
##              Prevalence : 0.5000
##          Detection Rate : 0.3968
##    Detection Prevalence : 0.4841
##       Balanced Accuracy : 0.8095
##
##        'Positive' Class : Loss
##

# Create the model using all tournament games 2008-2016
dt_model2 <- train(Result ~ ., data = dt_train_data2, method = "rpart",
metric = "Accuracy",
    tuneGrid = Tgrid, control = Tcontrol)

# plot model variable importance
importance <- varImp(dt_model2)
plot(importance, top = 5)

# Plot the model accuracy vs complexity parameter
plot(dt_model2)

# Create a fancy plot of the decision tree
fancyRpartPlot(dt_model2$finalModel)

# Predict the outcomes for the 2017 tournament
dt_model2_predict <- predict(dt_model2, dt_test_data1)

confusionMatrix(dt_model2_predict, dt_test_data1$Result)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction Loss Win
##       Loss   49   9
##       Win    14  54
## 
##                Accuracy : 0.8175
##                  95% CI : (0.7388, 0.8806)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 1.393e-13
## 
##                   Kappa : 0.6349
## 
##  Mcnemar's Test P-Value : 0.4042
## 
##             Sensitivity : 0.7778
##             Specificity : 0.8571
##          Pos Pred Value : 0.8448
##          Neg Pred Value : 0.7941
##              Prevalence : 0.5000
##          Detection Rate : 0.3889
##    Detection Prevalence : 0.4603
##       Balanced Accuracy : 0.8175
## 
##        'Positive' Class : Loss
## 
```