Predicting MLB Playoff Teams

Wayne Chong, Melissa Canas, Ryan Linow, Skyler Ataide

June 4, 2020

Introduction

In January, the United States was hit with the global pandemic known as COVID-19. The coronavirus canceled or postponed nearly every sporting event, major or minor. Now, as the country slowly initiates its reopening of essential businesses, one feature still remains on hold: live sports. As of now, the NFL aims to start the season as planned in September. The NBA and NHL have both agreed to restore their seasons with modified formats. However, the MLB and its Player's Union still cannot reach an agreement to begin their season. It has been approximately three and half months since the original start of the MLB season, and they have yet to play a single game. Thus, with a lack of baseball in our lives, we have decided to analyze the past using methods learned in class.

Over the course of a (typical) season, all 30 teams generate a massive variety of data, with hopes of reaching the postseason and ultimately winning the World Series. By analyzing the end of season statistics from all 30 teams and applying supervised machine learning techniques learned in class, our goal in this project is to predict playoff teams. As such, our research question is: Can we predict, with reasonable accuracy, which teams will make it to the 2018 MLB playoffs?

Data Overview

For our project, we will be using the Lahman's Baseball Database, which is a public database containing complete batting and pitching statistics from 1871 to 2018. The current MLB playoff format was adopted in 1994, but not implemented until 1995 (due to a strike). Therefore, our analysis will focus specifically on the data collected for the 1995 to 2018 seasons. Since our goal is to correctly predict the 2018 playoff teams, we use season statistics from 1995 to 2017 to build our model. The database also contains several data frames that summarize different aspects of the data, but we specifically use the Teams data table.

The Teams data frame contains summary statistics on every team's seasons, from 1871 to 2018. It has 2895 observations of these 48 variables:

```
##  [1] "yearID"         "lgID"           "teamID"         "franchID"
##  [5] "divID"          "Rank"           "G"              "Ghome"
##  [9] "W"              "L"              "DivWin"         "WCWin"
## [13] "LgWin"          "WSWin"          "R"              "AB"
## [17] "H"              "X2B"            "X3B"            "HR"
## [21] "BB"             "SO"             "SB"             "CS"
## [25] "HBP"            "SF"             "RA"             "ER"
## [29] "ERA"            "CG"             "SHO"            "SV"
## [33] "IPouts"         "HA"             "HRA"            "BBA"
## [37] "SOA"            "E"              "DP"             "FP"
## [41] "name"           "park"           "attendance"     "BPF"
## [45] "PPF"            "teamIDBR"       "teamIDlahman45" "teamIDretro"
```

While there are some missing values, they only exist in the data from the years before 1995, so we can ignore them. Many of the variables are irrelevant to our goal, and the data is missing a variable that is key to completing this project, so we had to adjust the raw data to our needs.

Methods

Data Cleaning and Preprocessing

First, we created our binary response variable, PlyOff. PlyOff is a factor variable with two levels – Y if a team made the playoffs and N if a team failed to reach the playoffs. We were left with 714 observations with the following playoff distribution:

```
##
##   N   Y
## 508 206
```

This means there was about a 28.85% probability of making it to the playoffs.

Next, we removed all characteristic variables from our data as they are irrelevant to our research. We also removed the two variables "Wins" and "Losses". If we were to include these covariates, we would correctly predict the best (playoff) teams 100% of the time, defeating the purpose of our project. Ultimately, we were left with 28 covariates: PlyOff, yearID, and 26 numeric predictors. Our 28 variables are the following:

```
##  [1] "yearID" "R"      "AB"     "H"      "X2B"    "X3B"    "HR"     "BB"
##  [9] "SO"     "SB"     "CS"     "HBP"    "SF"     "RA"     "ER"     "ERA"
## [17] "CG"     "SHO"    "SV"     "IPouts" "HA"     "HRA"    "BBA"    "SOA"
## [25] "E"      "DP"     "FP"     "PlyOff"
```

Split Training/Testing Data

We split our data into the training and testing data. The training data included statistics from 1995 to 2017 and the testing data included statistics from 2018. We also removed the yearID variable, as it was no longer important to the rest of the project. As such, the training data had 648 observations and the testing data had 30 observations, both containing 27 variables, including 26 predictor end of season statistics and the response variable.

Furthermore, to make our future analysis more convenient, we created vectors ytrain and ytest containing the response variables in the training and testing data sets respectively as well as design matrices xtrain and xtest containing the predictor variables in the training and testing data sets respectively.

Model Selection

We fit our training data twice using the classifiers k-NN, a basic classifier, and random forest, an ensemble method. These two classifiers were chosen because they are compatible with a binary response variable and multiple numerical prediction variables. To choose the best classifier, we compared the test error rate of both models.

Model Building

k-NN

We first used 5-fold cross validation on the training data to find the best value of neighbors.

The best number of neighbors from this process turned out to be:

```
## [1] 25
```

Then, we used this model to predict our response variable in the test data.

After comparing the predictions to the true values, the following test error rate was obtained:

```
## [1] 0.3333333
```

Random Forest

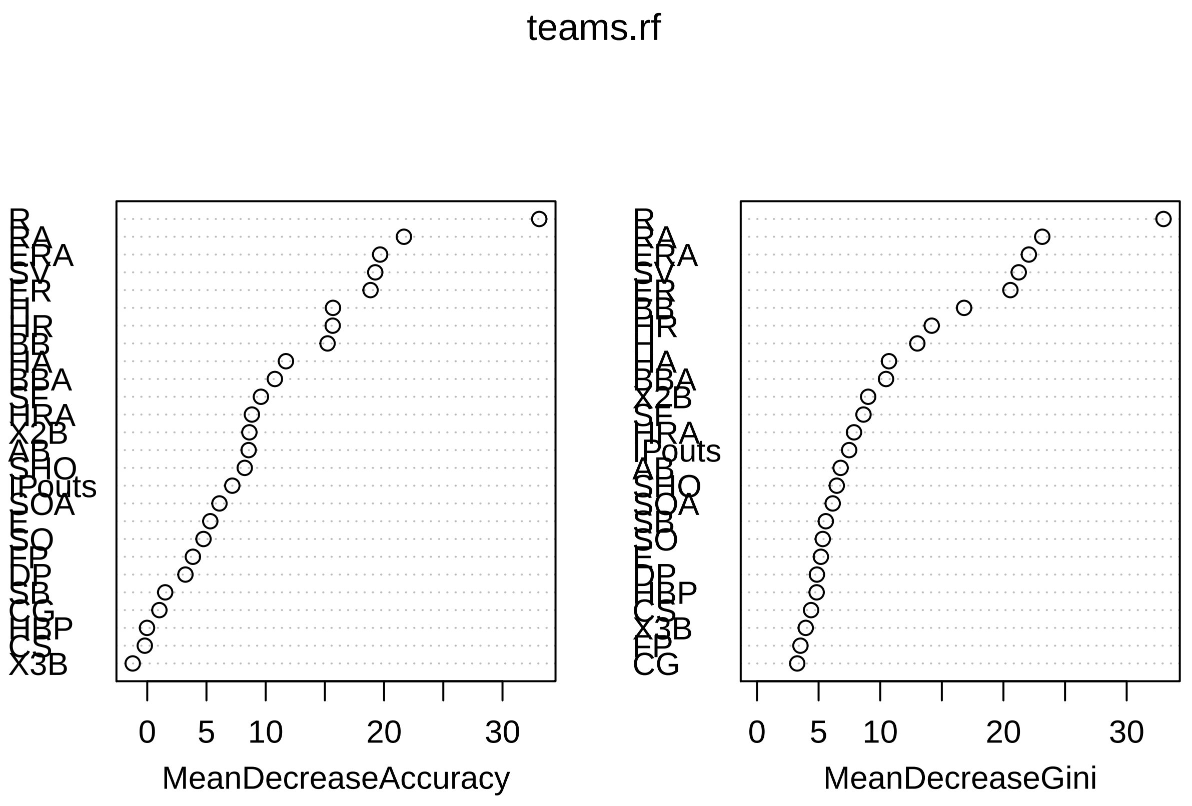
When fitting our training data using random forest, we used an $m \approx \sqrt{p}$ and 500 trees, meaning 5 predictors were considered for each split of each tree.

Then, we used this model to predict our response variable in the test data.

After comparing the predictions to the true values, the following test error rate was obtained:

```
## [1] 0.1
```

We also found that the variable runs (representing the number of home runs hit by batters) is the most predictive variable of whether a team made it to the playoffs. This is shown in the output of the importance function, suggesting that runs is indeed the most important variable in terms of model accuracy and Gini index.



Conclusions

The purpose of this data analysis was to develop an adequate model that could accurately predict whether or not an MLB team would make it to the playoffs, using historical data of the team's previous performance. Through cross validation, we found that the k-NN classifier had a 33% test error rate. The confusion matrix tells us that this model predicted that none of the teams would make the playoffs in 2018.

```
##          true
## predicted  N  Y
##         N 20 10
##         Y  0  0
```

In the first row, where the model predicted that the teams will not make it to the playoffs, a total of 30 teams are shown (which is all of them). Clearly, this cannot be correct. We believe this to mean that k-NN does not provide a model capable of predicting playoff teams.

The random forest classifier produced a much smaller test error, and therefore is our preferred model to predict the 2018 playoff teams. The 10% test error rate tells us that teams were incorrectly labelled 10% of the time. Specifically, the confusion matrix tells us that one team was inaccurately labeled as not making it to the playoffs when it did and two teams were inaccurately labelled as making it to the playoffs when they did not.

```
##     truth
## pred  N  Y
##    N 18  1
##    Y  2  9
```

Since this model more accurately predicted the teams that would make it to the playoffs, and the teams that would not, we believe that using a random forest classifier produced a successful model.

Because our model used data from a relatively long range of time, we were able to create a model with proficient accuracy. Many of the predictions our model made were accurate, with only a few exceptions. This model could be a valuable resource for sports analysts attempting to predict future seasons' outcomes, as well for team managers in understanding which specific stats will improve the likelihood of making it to the playoffs.

Appendix

Data Cleaning and Preprocessing

```r
#load data
library(devtools)
install_github("cdalzell/Lahman")

library(Lahman)
library(tidyverse)

#create dataframe
teamdata = filter(Teams, yearID >= 1995) %>%
  mutate(PlyOff=as.factor(ifelse(DivWin=='Y'|WCWin=='Y','Y','N'))) %>%
  select(yearID,R,AB,H,X2B,X3B,HR,BB,SO,SB,CS,HBP,SF,RA,ER,
         ERA,CG,SHO,SV,IPouts,HA,HRA,BBA,SOA,E,DP,FP,PlyOff)
```

Split Training/Testing Data

```r
teams.train = filter(teamdata, yearID<=2017) %>% select(-yearID)
teams.test = filter(teamdata, yearID==2018) %>% select(-yearID)

xtrain = teams.train %>% select(-PlyOff)
xtrain = scale(xtrain, center=TRUE, scale=TRUE)
ytrain = teams.train$PlyOff

meanvec = attr(xtrain,'scaled:center')
sdvec = attr(xtrain,'scaled:scale')

xtest = teams.test %>% select(-PlyOff)
ytest = teams.test$PlyOff
```

k-NN

```r
library(reshape2)
library(plyr)
library(dplyr)
library(class)

#5-fold cross validation
do.chunk <- function(chunkid, folddef, Xdat, Ydat,k){
  train = (folddef!=chunkid)
  xtr = Xdat[train,]
  ytr = Ydat[train]
  xvl = Xdat[!train,]
  yvl = Ydat[!train]
  predytr = knn(train=xtr, test=xtr, cl=ytr,k )
  predyvl = knn(train=xtr, test=xvl, cl=ytr,k)
  data.frame(fold=chunkid, train.error=mean(predytr!=ytr), val.error=mean(predyvl!=yvl))
}

nfold = 5

set.seed(0520201)
folds = cut(1:nrow(teams.train), breaks=nfold, labels=FALSE) %>% sample()
folds
error.folds = NULL
allK = 1:50

set.seed(0520202)
for (j in allK){
  tmp = ldply(1:nfold, do.chunk, folddef=folds, Xdat=xtrain, Ydat=ytrain,k=j)
  tmp$neighbors = j
  error.folds = rbind(error.folds, tmp)
}
dim(error.folds)
head(error.folds)
```

```r
errors = melt(error.folds, id.vars=c('fold', 'neighbors'), value.name='error')
val.error.means = errors %>%
  filter(variable=='val.error') %>%
  group_by(neighbors, variable) %>%
  summarise_each(funs(mean), error) %>%
  ungroup() %>%
  filter(error==min(error))

#find best number of neighbors
numneighbor = max(val.error.means$neighbors)
numneighbor

#predict response
set.seed(0520203)
knn.pred = knn(train=xtrain, test=xtest, cl=ytrain, k=numneighbor)

#confusion matrix
conf.knn = table(predicted=knn.pred, true=ytest)
conf.knn

#test accuracy rate
sum(diag(conf.knn)/sum(conf.knn))
#test error rate
1-sum(diag(conf.knn)/sum(conf.knn))
```

Random Forest

```r
library(randomForest)
library(gbm)
library(tree)

#fit model
teams.rf = randomForest(PlyOff~., data=teams.train, mtry=5, ntree=500, importance=TRUE)
teams.rf

#predict response
set.seed(0520204)
rf.pred = predict(teams.rf, newdata=teams.test)

#confusion matrix
conf.rf = table(pred=rf.pred, truth=ytest)
conf.rf

#test accuracy rate
sum(diag(conf.rf)/sum(conf.rf))
#test error rate
1-sum(diag(conf.rf)/sum(conf.rf))

#important variables
importance(teams.rf)
varImpPlot(teams.rf)
```