Using Machine Learning to Predict Over / Under Moneylines

William Li

17/12/2020

Introduction

This exercise in machine learning will build models to predict whether or not a team beats its pregame betting spread. Point spreads are a method used by bookkeepers to handicap games. Bettors can bet the over or under on the spread. For example if a game between the Raptors and the Lakers is set at -6.5, that means the Raptors are expected to beat the Lakers by 6.5 points.

If the Raptors beat the Lakers by 6 points or less or the Raptors lose, the bettors who bet under the spread win their bets. If the Raptors win by 7 points or more the Raptors have "beat the spead" and the bettors who bet on the over will win their bets. Losing bets lose 100% of their money and winning bets win a certain number of cents to their dollar usually around $0.90 per dollar bet.

Data will be scraped from the web and the "Caret" package will be used to build machine learning models. Other complementary packages will be used to produce visuals and implement machine learning models.

Scraping Data

Data was scraped from sportsdatabase.com, a website with data from NBA games in simple HTML tables. Only data from the 2019-2020 season was scraped.

```
games.2019 <- read_html('https://sportsdatabase.com/nba/query?output=default&sdql=team%2C+site%2C+o%3At

dat <- data.frame(games.2019[4])
```

8 columns of data were selected from every NBA game in the season. team : The team playing, site : Home game or away game, o.team : The opposing team, line : The point spread before the game, margin : The margin of the game relative to the other team's score. Negative values are losses. Postive values are wins, wins : number of wins the team has, losses : number of losses the team has

```
summary(dat)
```

```
##      team               site              o.team               line       
##  Length:2286        Length:2286        Length:2286        Min.   :-18.0  
##  Class :character   Class :character   Class :character   1st Qu.: -5.5  
##  Mode  :character   Mode  :character   Mode  :character   Median :  0.0  
##                                                           Mean   :  0.0  
##                                                           3rd Qu.:  5.5  
##                                                           Max.   : 18.0  
##      streak              margin         wins           losses     
##  Min.   :-13.00000   Min.   :-49   Min.   : 0.00   Min.   : 0.00  
##  1st Qu.: -2.00000   1st Qu.: -9   1st Qu.: 8.00   1st Qu.: 8.00
##  Median :  0.00000   Median :  0   Median :17.00   Median :17.00
##  Mean   :  0.04374   Mean   :  0   Mean   :20.22   Mean   :17.89
##  3rd Qu.:  2.00000   3rd Qu.:  9   3rd Qu.:30.00   3rd Qu.:27.00
##  Max.   : 18.00000   Max.   : 49   Max.   :67.00   Max.   :49.00
```

Feature Engineering

Two new columns were created. The first is the win percentage of the team. The second is the target variable "beat.line". This is a boolean variable stating whether the team beat its pregame betting spread.

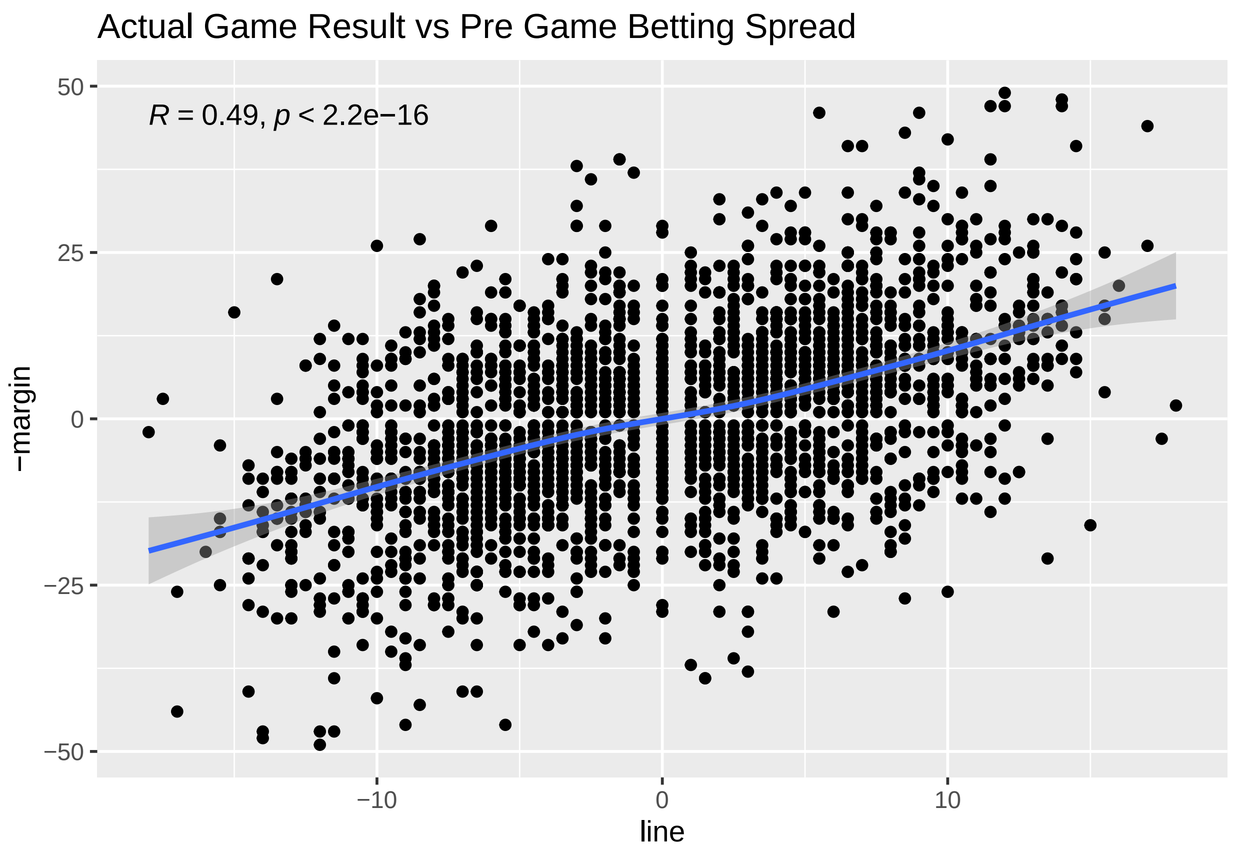
```
# adding new columns
dat.b <- mutate(dat,
                win.p = wins/(wins+losses),
                beat.line=ifelse(margin > line,'yes','no'),
                margin=NULL)

# replacing nan values of win.p
dat.b$win.p[is.nan(dat.b$win.p)] <- 0.5
```

Data Profiling

A simple plot of the spread vs the margin of the actual score. It shows a linear relationship with a correlation of 0.49 which is not bad. Bookkeepers do generally set the spread accurately.

```
ggplot(data = dat,aes(x=line, y = -margin)) + geom_point() +
  ggtitle('Actual Game Result vs Pre Game Betting Spread') +
  stat_cor(method="pearson") +
  geom_smooth()
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



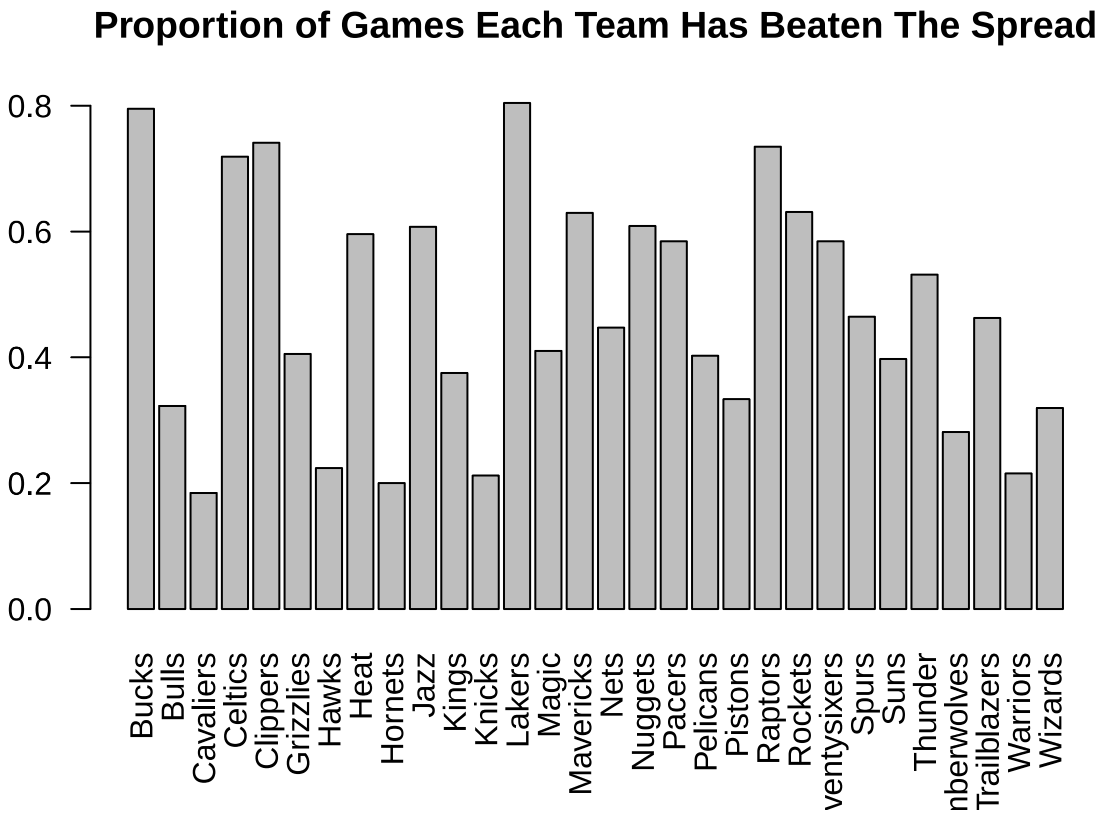
Also the equation of the linear model between the pregame betting spread and the betting line has an intercept of zero and slope of 1. There is no bias in the estimation of the final score by the betting spread.

```
lm(-dat$margin ~ dat$line)
```

```
##
## Call:
## lm(formula = -dat$margin ~ dat$line)
##
## Coefficients:
## (Intercept)     dat$line
##   7.431e-17    1.006e+00
```

Some teams beat the spread more often than others. Good teams tend to beat the spread more whereas the worst teams tend to underperform relative to the spread.

```
dat.b$bool <- ifelse(dat.b$beat.line == 'yes',1,0)
ag <- aggregate(bool ~ team, data = dat.b, mean)
ag <- ag[order(ag$bool,decreasing = TRUE),]
barplot(xtabs(bool ~ team, data = ag), las=2,
        main='Proportion of Games Each Team Has Beaten The Spread')
```



Data Partitioning

The data is then partitioned into two sets, one for training and then one for testing, to evaluate the results.

```
# training set
training.index <- createDataPartition(dat.b$team, p = 0.8, list = FALSE)
dat.b.training <- dat.b[training_index, ]

# testing set
dat.b.testing <- dat.b[-training_index, ]
```

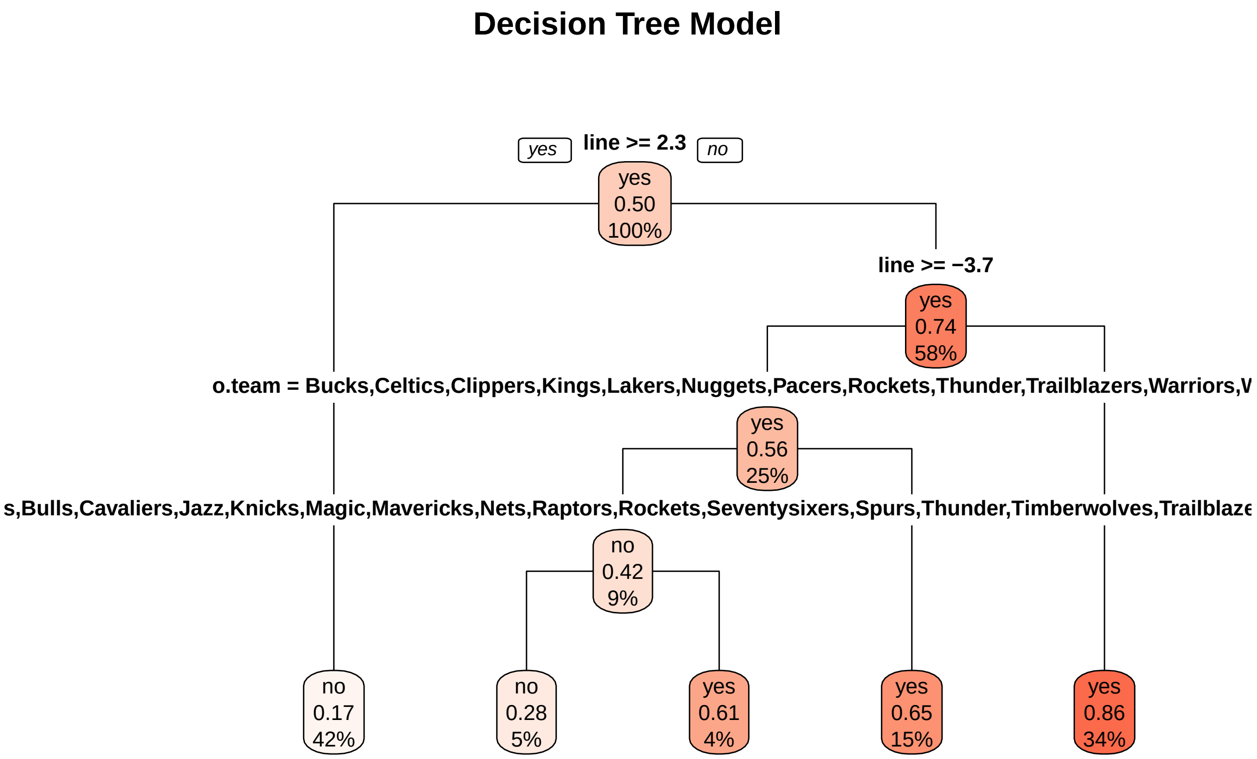
Decision Tree Model

The first model that was created is the decision tree model. This model creates binary cuts in the dataset by using decision rules. It then evaluates these cuts using gini coefficients. The parameters that can best discriminate between the two target classes while satisfying the model restrictions is chosen to be used.

```
set.seed(123)

dt.model <- train(x = dat.b.training[,c(1:8)],
                  y = factor(dat.b.training$beat.line),
                  method = "rpart",
                  tuneLength = 10,
                  metric = "ROC",
                  trControl = trainControl(method = "cv",
                                           number = 10,
                                           classProbs = TRUE,
                                           summaryFunction = twoClassSummary))
```

Below is a visualization of the tree of the final model.

```
# plot of decision tree
rpart.plot(dt.model$finalModel,
           main = "Decision Tree Model",
           box.palette = "Reds",
           type=1)
```


Decision Tree Model

yes
line >= 2.3
no
yes 0.50 100%
line >= −3.7
yes 0.74 58%
o.team = Bucks,Celtics,Clippers,Kings,Lakers,Nuggets,Pacers,Rockets,Thunder,Trailblazers,Warriors,W
yes 0.56 25%
s,Bulls,Cavaliers,Jazz,Knicks,Magic,Mavericks,Nets,Raptors,Rockets,Seventysixers,Spurs,Thunder,Timberwolves,Trailblaze
no 0.42 9%
no 0.17 42%
no 0.28 5%
yes 0.61 4%
yes 0.65 15%
yes 0.86 34%

The model produces surprisingly impressive results. The accuracy of 73% means that the model would predict whether a team beats the spread 73% of the time. Due to the bookkeepers edge one would need to win around 53% of over / under bets to make a profit. To be able to win over / under bets 73% of the time is incredible.

```
# Confusion Matrix

dt.pred <- predict(dt.model, dat.b.testing)
dt.cm <- confusionMatrix(table(dat.b.testing$beat.line,dt.pred))
dt.cm
```

```
## Confusion Matrix and Statistics
```

```
##
##      dt.pred
##        no yes
##   no  169  72
##   yes  44 160
##
##                Accuracy : 0.7393
##                  95% CI : (0.6959, 0.7795)
##     No Information Rate : 0.5213
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.4805
##
##  Mcnemar’s Test P-Value : 0.01218
##
##             Sensitivity : 0.7934
##             Specificity : 0.6897
##          Pos Pred Value : 0.7012
##          Neg Pred Value : 0.7843
##              Prevalence : 0.4787
##          Detection Rate : 0.3798
##    Detection Prevalence : 0.5416
##       Balanced Accuracy : 0.7415
##
##        ’Positive’ Class : no
##
```

More models have been built below. For the sake of simplicity and brevity their function and implementation will not be described. These models are more complex and the scope of how it works is beyond this course.

Naive Bayes Model

```r
# Naive Bayes

nb.model <- train(x = dat.b.training[,c(1:8)],
                       y = factor(dat.b.training$beat.line),
                       method = "nb",
                       tuneLength = 10,
                       metric = "ROC",
                       trControl = trainControl(method = "cv",
                                                number = 10,
                                                classProbs = TRUE,
                                                summaryFunction = twoClassSummary))
```

```r
nb.pred <- predict(nb.model, dat.b.testing)
```

```r
confusionMatrix(table(nb.pred , dat.b.testing$beat.line))
```

```
## Confusion Matrix and Statistics
##
##
```

```
## nb.pred  no yes
##     no  177  51
##     yes  64 153
##
##                Accuracy : 0.7416
##                  95% CI : (0.6983, 0.7816)
##     No Information Rate : 0.5416
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.4821
##
##  Mcnemar’s Test P-Value : 0.2631
##
##             Sensitivity : 0.7344
##             Specificity : 0.7500
##          Pos Pred Value : 0.7763
##          Neg Pred Value : 0.7051
##              Prevalence : 0.5416
##          Detection Rate : 0.3978
##    Detection Prevalence : 0.5124
##       Balanced Accuracy : 0.7422
##
##        ’Positive’ Class : no
##
```

Random Forest Model

```
# Random Forest Model
rv.model <- train(x = dat.b.training[,c(1:8)],
                       y = factor(dat.b.training$beat.line),
                       method = "ranger",
                       importance = "impurity",
                       tuneLength = 10,
                       metric = "ROC",
                       trControl = trainControl(method = "cv",
                                                number = 10,
                                                classProbs = TRUE,
                                                summaryFunction = twoClassSummary))
```

```
pred <- predict(rv.model, dat.b.testing)
confusionMatrix(table(pred, dat.b.testing$beat.line))
```

```
## Confusion Matrix and Statistics
##
##
## pred   no yes
##   no  170  48
##   yes  71 156
##
##                Accuracy : 0.7326
##                  95% CI : (0.6889, 0.7732)
##     No Information Rate : 0.5416
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.4661
##
##  Mcnemar's Test P-Value : 0.04372
##
##             Sensitivity : 0.7054
##             Specificity : 0.7647
##          Pos Pred Value : 0.7798
##          Neg Pred Value : 0.6872
##              Prevalence : 0.5416
##          Detection Rate : 0.3820
##    Detection Prevalence : 0.4899
##       Balanced Accuracy : 0.7351
##
##        'Positive' Class : no
##
```

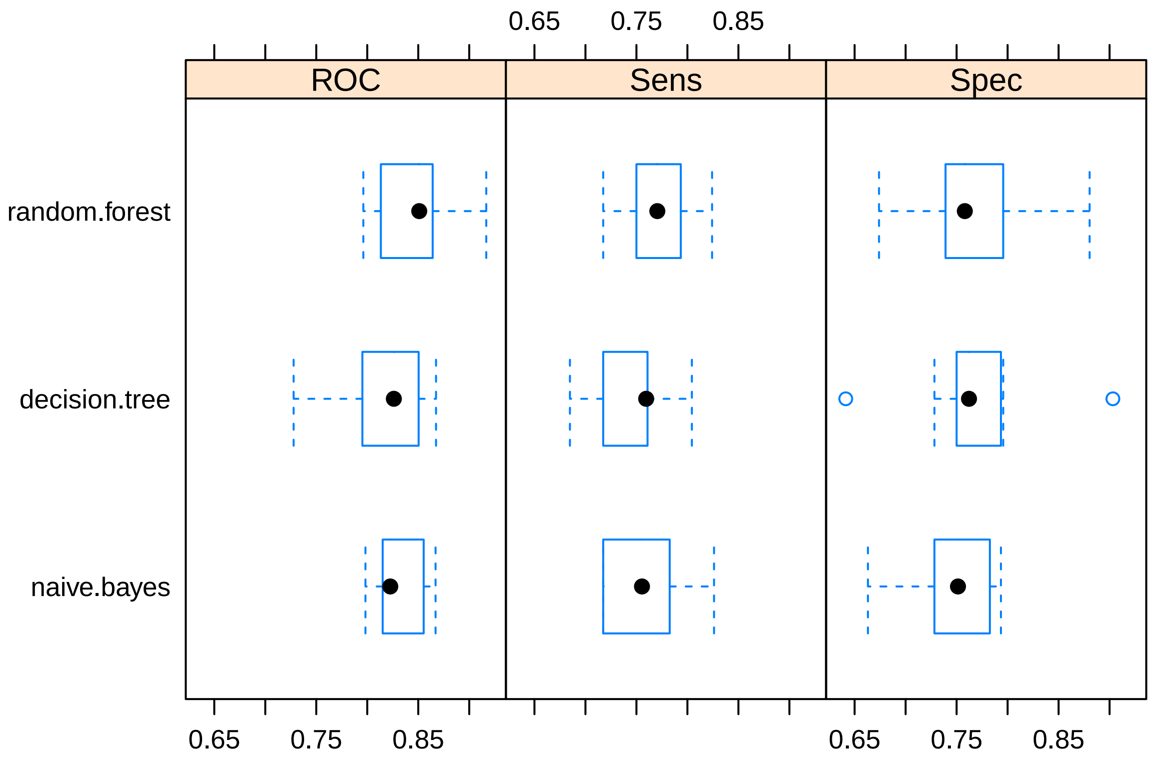
Comparision of Models

For the sake of brevity, we will not go into how to select a machine learning model. Below are some key metrics for each model. I will let the reader determine which model should be used.

```
models <- list(decision.tree = dt.model,
                 naive.bayes = nb.model,
                 random.forest = rv.model)

models.resampling <- resamples(models)
summary(models.resampling)
```

```
##
## Call:
## summary.resamples(object = models.resampling)
##
## Models: decision.tree, naive.bayes, random.forest
## Number of resamples: 10
##
## ROC
##                    Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## decision.tree 0.7277883 0.7992483 0.8260771 0.8167894 0.8469400 0.8674229    0
## naive.bayes   0.7982042 0.8164865 0.8225565 0.8310067 0.8509551 0.8669503    0
## random.forest 0.7961366 0.8171078 0.8509681 0.8471833 0.8636483 0.9166866    0
##
## Sens
##                    Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## decision.tree 0.6847826 0.7228261 0.7595557 0.7461419 0.7608696 0.8043478    0
## naive.bayes   0.7173913 0.7193622 0.7554348 0.7560081 0.7771739 0.8260870    0
## random.forest 0.7173913 0.7527174 0.7704849 0.7691233 0.7880435 0.8241758    0
##
## Spec
##                    Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## decision.tree 0.6413043 0.7500000 0.7621552 0.7658018 0.7880435 0.9032258    0
## naive.bayes   0.6630435 0.7289914 0.7513441 0.7475339 0.7771739 0.7934783    0
## random.forest 0.6739130 0.7418478 0.7580645 0.7692263 0.7924264 0.8804348    0
```

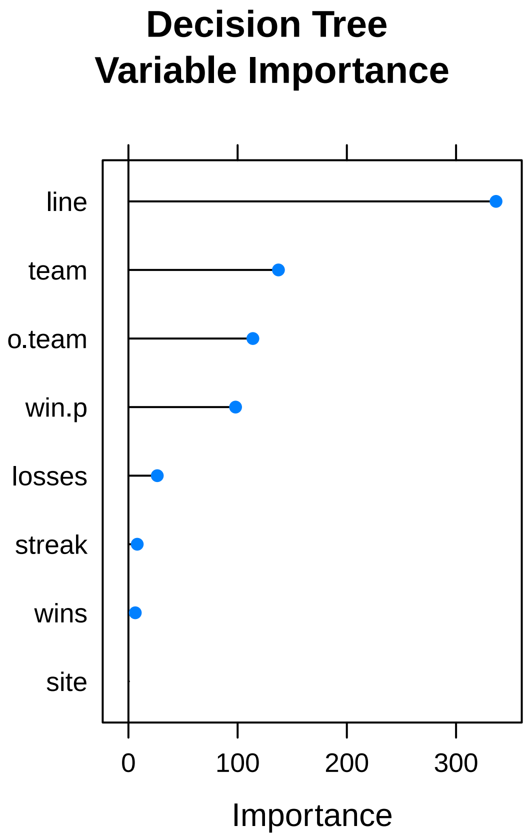
```
bwplot(models.resampling)
```

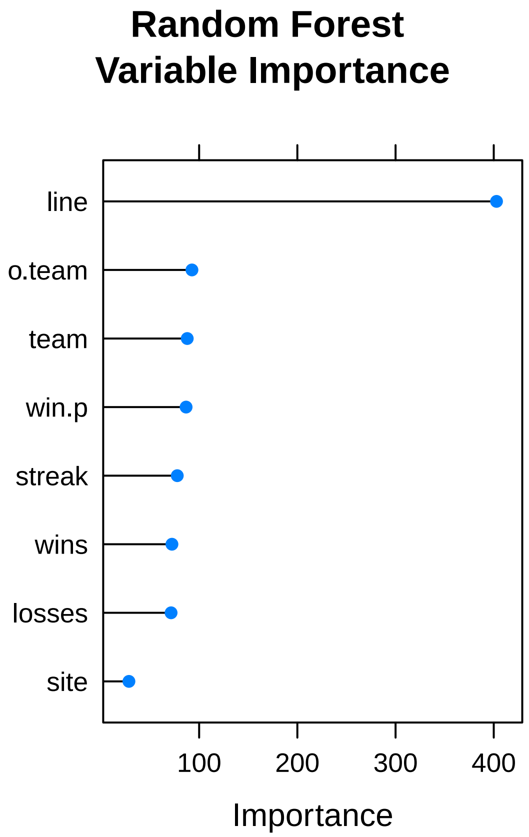


Variable Importance

A plot of the importance of each variable below. Removing the variables of the least importance should be considered. Naive Bayes models do not have this feature. In both models, “line” or the point spread is by far the most important feature.

```
par(mfrow=c(2,2))

dt.importance <- varImp(dt.model, scale = FALSE)
p1 <- plot(dt.importance, main = "Decision Tree \nVariable Importance")

rv.importance <- varImp(rv.model, scale = FALSE)
p2 <- plot(rv.importance, main = "Random Forest \nVariable Importance")

grid.arrange(p1,p2,ncol=2)
```





Analysis of Results

The models produce surprisingly impressive results. Each model's accuracy is between 70% and 80%. As it was mentioned earlier, due to the bookkeepers edge one would need to beat the spread around 53% of the timeto make a profit. To be able to win over 70% of bets is incredible.

For this reason I am skeptical of these models. Unless I have discovered a systematic way to beat bookkeepers using a simple machine learning model, these results cannot be valid.

Perhaps there is an error in the methodology. Using randomly selected data from an entire season to predict games in the same season means that there will be games from the future used to predict past games. This is not realistic as during a season we do not have data from games in the future.

A more realistic model would be predicting games from the current season with data from past seasons.

Shiny App Implementation

An online application was then made with the R package, Shiny, to allow users to input data about basketball games and then have predictions be made on the probability of the team beating the spread The code for the app is below.

```
# load r data because we do not want to build the model every time
load('.RData')

ui <- fluidPage(
  titlePanel(’Over / Under Moneyline Prediction’),
  sidebarPanel(" Select model to use on the right then input data on the teams and prediction will be ma
  sidebarPanel(
  radioButtons("model", "Model:",
                     names(models), inline=TRUE),
  selectInput(’team’,’Team’,unique(dat$team)),
  selectInput(’site’,’Site’,unique(dat$site)),
  selectInput(’o.team’,’Opposing Team’,unique(dat$team)),
  numericInput("line", "Line:", 0, step=0.5),
  numericInput("streak", "Streak:", 0, step=1),
  numericInput("wins", "Wins:", 0, min = 0, step=1),
  numericInput("losses", "Losses:", 0, min = 0, step=1),
  ),
  h4(’Probablity of Beating Moneyline’),
  tableOutput("data")
)

server <- function(input, output, session) {
  output$data <- renderTable({
    pred.i <- data.frame (team = input$team,
                          site = input$site,
                          o.team = input$o.team,
                          line = input$line,
                          streak = input$streak,
                          wins = input$wins,
                          losses = input$losses,
                          win.p = ifelse(is.nan(input$wins/(input$wins+input$losses)),
                                         0.5,
                                         input$wins/(input$wins+input$losses)))
    data.frame(predict(models[input$model],pred.i,type=’prob’))

  })

}

shinyApp(ui, server, options = list(height = 500))
```

The application was hosted using shinyapps.io See application below. The app will not display in the pdf version, see the html for the app or visit: https://william-li.shinyapps.io/nbamoneylineprediction/

```
knitr::include_app("https://william-li.shinyapps.io/nbamoneylineprediction/")
```

```
## PhantomJS not found. You can install it with webshot::install_phantomjs(). If it is installed, pleas
```