Predict to Result of NBA Match Scores by Using Machine Learning Methods

Veli Kisa

27 04 2020

Introduction

Nba is the biggest basketball organization in the world. So there are many people who support a team in league and this leds to a large betting industry. It is too important and valuable for betting companies if a match result can be estimated closest to the real result. This project aims to predict the result of NBA match scores in regular season. Since the match scores are numeric it is more difficult than estimate match result as binary win or loss. Regression techniques will be used to estimate match scores.

Data

Data set based on box score and standing statistics from the NBA.

- teamBoxScore communicates game data from each teams perspective
- standing communicates standings data for each team every day during the season

Data link: https://www.kaggle.com/pablote/nba-enhanced-stats

```
require(readr)
X2012_18_teamBoxScore <- read_csv("~/R/nba/nbaresult/2012-18_teamBoxScore.csv")
X2012_18_standings <- read_csv("~/R/nba/nbaresult/2012-18_standings.csv")
```

We will work on 3 years data. (2016-2018).

```
require(lubridate)
X2016_18team.stats<-X2012_18_teamBoxScore[year(X2012_18_teamBoxScore$gmDate)==2016|
                                          year(X2012_18_teamBoxScore$gmDate)==2017|
                                          year(X2012_18_teamBoxScore$gmDate)==2018,]

X2016_18standings<-X2012_18_standings[year(X2012_18_standings$stDate)==2016|
                                      year(X2012_18_standings$stDate)==2017|
                                      year(X2012_18_standings$stDate)==2018,]


dim(X2016_18team.stats)
```

```
## [1] 6402  123
```

```
dim(X2016_18standings)
```

```
## [1] 12750    39
```

Data Manipulation

The first step of data manipulation is combining two stats data “team.stats” and “standings” according to information of date of match “gmDate” and name of teams “teamAbbr” (team) and “opptAbbr” (against team).

```
nba.data1<-merge(X2016_18team.stats,X2016_18standings,by.x = c("gmDate","teamAbbr"),by.y = c("stDate","
nba.data1<-merge(nba.data1,X2016_18standings,by.x=c("gmDate","opptAbbr"),by.y = c("stDate","teamAbbr"))

nba.data1<-nba.data1[,c("gmDate","teamAbbr","opptAbbr","teamLoc","teamRslt","teamDayOff",
                        "teamPTS","rank.x","gameWon.x","gameLost.x","stk.x","stkType.x",
                        "stkTot.x","gameBack.x","ptsFor.x","ptsAgnst.x","homeWin.x",
                        "homeLoss.x","awayWin.x","awayLoss.x","confWin.x","confLoss.x",
                        "lastFive.x","lastTen.x","ptsScore.x","ptsAllow.x","ptsDiff.x",
                        "opptGmPlay.x","opptGmWon.x","sos.x","rel%Indx.x","mov.x","srs.x",
                        "pw%.x","pyth%13.91.x","wpyth13.91.x","lpyth13.91.x",
                        "pyth%16.5.x","wpyth16.5.x","lpyth16.5.x","rank.y","gameWon.y",
                        "gameLost.y","stk.y","stkType.y","stkTot.y","gameBack.y","ptsFor.y",
                        "ptsAgnst.y","homeWin.y","homeLoss.y","awayWin.y","awayLoss.y",
                        "confWin.y","confLoss.y","lastFive.y","lastTen.y", "ptsScore.y",
                        "ptsAllow.y","ptsDiff.y","opptGmPlay.y","opptGmWon.y","sos.y",
                        "rel%Indx.y","mov.y","srs.y","pw%.y","pyth%13.91.y","wpyth13.91.y",
                        "lpyth13.91.y", "pyth%16.5.y","wpyth16.5.y","lpyth16.5.y"
                         )]
```

We selected only independent variables in "X2016_18standings" data to estimate target value "teamPTS". The variables of "X2016_18team.stats" data are not selected since they are a function of target value. (For example: "teamFG%" is dependent on "teamPTS")

In this part, it is good to see the graphs of home advantage situation for teams. For 3 season, barplot of win/loss situation as percentage can be seen following chunk:

```
require(dplyr)
```

```
## Loading required package: dplyr
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:lubridate':
##
##     intersect, setdiff, union
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
require(plyr)
```

```
## Loading required package: plyr
```

```
## ------------------------------------------------------------------------------
```

```
## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)
```

```
## ------------------------------------------------------------------------------
```

```
##
## Attaching package: 'plyr'
```

```
## The following objects are masked from 'package:dplyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize
```

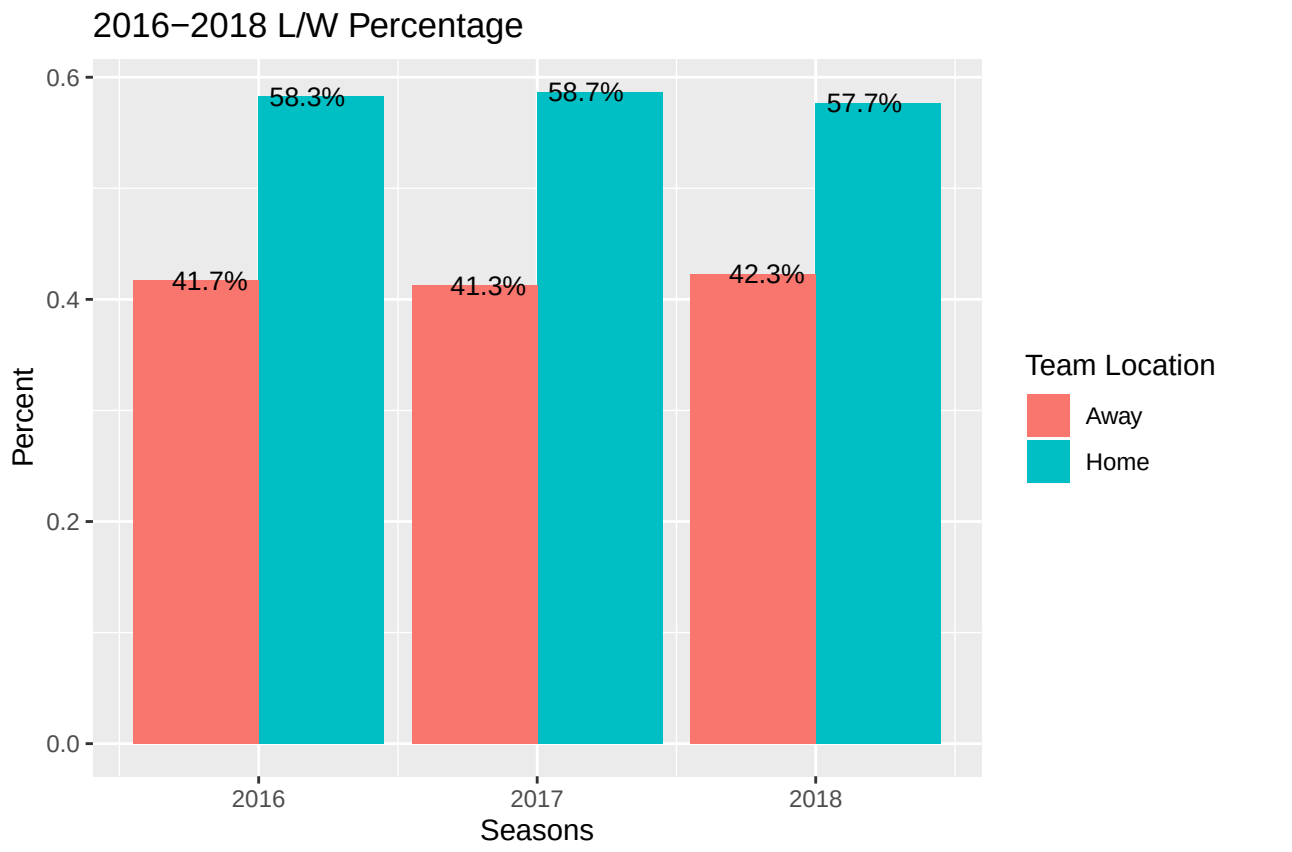
```
## The following object is masked from 'package:lubridate':
##
##     here
```

```
require(ggplot2)
```

```
## Loading required package: ggplot2
```

```
require(lubridate)
nba.data1_2<-nba.data1
nba.data1_2$gmDate<-year(nba.data1_2$gmDate)
freq.table<-dplyr::summarise(group_by(nba.data1_2,gmDate,teamRslt,teamLoc),count =n())
freq.table<-freq.table[freq.table$teamRslt=="Win",]
freq.table<-ddply(freq.table, .(gmDate), mutate, per.win = round((count / sum(count)),3))

ggplot(freq.table, aes(fill=freq.table$teamLoc, y=freq.table$per.win, x=freq.table$gmDate))+
geom_bar(position="dodge", stat="identity")+
geom_text(aes(label=scales::percent(freq.table$per.win)), position = position_dodge(.7),
          size = 3.5)+
labs(title = "2016-2018 L/W Percentage", y = "Percent", x = "Seasons",fill = "Team Location")
```



The second step of manipulating is assign to numeric values to the name of team abbrevation and result situation win or loss.

```
require(dplyr)
nba.data1<-nba.data1%>%
  mutate(teamAbbr=case_when(
    teamAbbr==unique(sort(nba.data1$teamAbbr))[1] ~ 1,
    teamAbbr==unique(sort(nba.data1$teamAbbr))[2] ~ 2,
    teamAbbr==unique(sort(nba.data1$teamAbbr))[3] ~ 3,
    teamAbbr==unique(sort(nba.data1$teamAbbr))[4] ~ 4,
    teamAbbr==unique(sort(nba.data1$teamAbbr))[5] ~ 5,
    teamAbbr==unique(sort(nba.data1$teamAbbr))[6] ~ 6,
    teamAbbr==unique(sort(nba.data1$teamAbbr))[7] ~ 7,
    teamAbbr==unique(sort(nba.data1$teamAbbr))[8] ~ 8,
    teamAbbr==unique(sort(nba.data1$teamAbbr))[9] ~ 9,
    teamAbbr==unique(sort(nba.data1$teamAbbr))[10] ~ 10,
    teamAbbr==unique(sort(nba.data1$teamAbbr))[11] ~ 11,
    teamAbbr==unique(sort(nba.data1$teamAbbr))[12] ~ 12,
    teamAbbr==unique(sort(nba.data1$teamAbbr))[13] ~ 13,
    teamAbbr==unique(sort(nba.data1$teamAbbr))[14] ~ 14,
    teamAbbr==unique(sort(nba.data1$teamAbbr))[15] ~ 15,
    teamAbbr==unique(sort(nba.data1$teamAbbr))[16] ~ 16,
    teamAbbr==unique(sort(nba.data1$teamAbbr))[17] ~ 17,
    teamAbbr==unique(sort(nba.data1$teamAbbr))[18] ~ 18,
    teamAbbr==unique(sort(nba.data1$teamAbbr))[19] ~ 19,
    teamAbbr==unique(sort(nba.data1$teamAbbr))[20] ~ 20,
    teamAbbr==unique(sort(nba.data1$teamAbbr))[21] ~ 21,
    teamAbbr==unique(sort(nba.data1$teamAbbr))[22] ~ 22,
    teamAbbr==unique(sort(nba.data1$teamAbbr))[23] ~ 23,
    teamAbbr==unique(sort(nba.data1$teamAbbr))[24] ~ 24,
    teamAbbr==unique(sort(nba.data1$teamAbbr))[25] ~ 25,
    teamAbbr==unique(sort(nba.data1$teamAbbr))[26] ~ 26,
    teamAbbr==unique(sort(nba.data1$teamAbbr))[27] ~ 27,
    teamAbbr==unique(sort(nba.data1$teamAbbr))[28] ~ 28,
    teamAbbr==unique(sort(nba.data1$teamAbbr))[29] ~ 29,
    teamAbbr==unique(sort(nba.data1$teamAbbr))[30] ~ 30
  ))

nba.data1<-nba.data1%>%
  mutate(opptAbbr=case_when(
    opptAbbr==unique(sort(nba.data1$opptAbbr))[1] ~ 1,
    opptAbbr==unique(sort(nba.data1$opptAbbr))[2] ~ 2,
    opptAbbr==unique(sort(nba.data1$opptAbbr))[3] ~ 3,
    opptAbbr==unique(sort(nba.data1$opptAbbr))[4] ~ 4,
    opptAbbr==unique(sort(nba.data1$opptAbbr))[5] ~ 5,
    opptAbbr==unique(sort(nba.data1$opptAbbr))[6] ~ 6,
    opptAbbr==unique(sort(nba.data1$opptAbbr))[7] ~ 7,
    opptAbbr==unique(sort(nba.data1$opptAbbr))[8] ~ 8,
    opptAbbr==unique(sort(nba.data1$opptAbbr))[9] ~ 9,
    opptAbbr==unique(sort(nba.data1$opptAbbr))[10] ~ 10,
    opptAbbr==unique(sort(nba.data1$opptAbbr))[11] ~ 11,
    opptAbbr==unique(sort(nba.data1$opptAbbr))[12] ~ 12,
    opptAbbr==unique(sort(nba.data1$opptAbbr))[13] ~ 13,
    opptAbbr==unique(sort(nba.data1$opptAbbr))[14] ~ 14,
    opptAbbr==unique(sort(nba.data1$opptAbbr))[15] ~ 15,
    opptAbbr==unique(sort(nba.data1$opptAbbr))[16] ~ 16,
```

```
    opptAbbr==unique(sort(nba.data1$opptAbbr))[17] ~ 17,
    opptAbbr==unique(sort(nba.data1$opptAbbr))[18] ~ 18,
    opptAbbr==unique(sort(nba.data1$opptAbbr))[19] ~ 19,
    opptAbbr==unique(sort(nba.data1$opptAbbr))[20] ~ 20,
    opptAbbr==unique(sort(nba.data1$opptAbbr))[21] ~ 21,
    opptAbbr==unique(sort(nba.data1$opptAbbr))[22] ~ 22,
    opptAbbr==unique(sort(nba.data1$opptAbbr))[23] ~ 23,
    opptAbbr==unique(sort(nba.data1$opptAbbr))[24] ~ 24,
    opptAbbr==unique(sort(nba.data1$opptAbbr))[25] ~ 25,
    opptAbbr==unique(sort(nba.data1$opptAbbr))[26] ~ 26,
    opptAbbr==unique(sort(nba.data1$opptAbbr))[27] ~ 27,
    opptAbbr==unique(sort(nba.data1$opptAbbr))[28] ~ 28,
    opptAbbr==unique(sort(nba.data1$opptAbbr))[29] ~ 29,
    opptAbbr==unique(sort(nba.data1$opptAbbr))[30] ~ 30
  ))

nba.data1<-nba.data1%>%
  mutate(teamLoc=case_when(
    teamLoc=="Home"  ~ 1,
    teamLoc=="Away" ~ 2))

nba.data1<-nba.data1%>%
  mutate(teamRslt=case_when(
    teamRslt=="Win"  ~ 1,
    teamRslt=="Loss" ~ 2))

nba.data1<-nba.data1%>%
  mutate(stkType.x=case_when(
    stkType.x=="win"  ~ 1,
    stkType.x=="loss" ~ 2))

nba.data1<-nba.data1%>%
  mutate(stkType.y=case_when(
    stkType.y=="win"  ~ 1,
    stkType.y=="loss" ~ 2))

nba.data1[1:5,1:10]
```

```
##       gmDate teamAbbr opptAbbr teamLoc teamRslt teamDayOff teamPTS rank.x
## 1 2016-01-01       28        4       1        1          2     104      2
## 2 2016-01-01       20        5       2        2          3      81     12
## 3 2016-01-01       16        7       1        1          3     106      5
## 4 2016-01-01       23       14       2        2          2      84     15
## 5 2016-01-01        7       16       2        2          2      82      5
##   gameWon.x gameLost.x
## 1        21         13
## 2        15         19
## 3        19         13
## 4         3         32
## 5        19         14
```

The next step is to label what variables type are.

```
nba.data1$teamAbbr<-as.factor(nba.data1$teamAbbr)
nba.data1$opptAbbr<-as.factor(nba.data1$opptAbbr)
```

```
nba.data1$teamLoc<-as.factor(nba.data1$teamLoc)
nba.data1$teamRslt<-as.factor(nba.data1$teamRslt)
nba.data1$stkType.x<-as.factor(nba.data1$stkType.x)
nba.data1$stkType.y<-as.factor(nba.data1$stkType.y)
```

Now we will reduce 3 variables of the same meaning to 2 variables on "stk.x","stkType.x" and "stkTot.x".

```
nba.data1[1:3,c("teamAbbr","stk.x","stkType.x","stkTot.x","opptAbbr","stk.y","stkType.y","stkTot.y")]
```

```
##   teamAbbr stk.x stkType.x stkTot.x opptAbbr stk.y stkType.y stkTot.y
## 1       28    W2         1        2        4    L2         2        2
## 2       20    L1         2        1        5    W3         1        3
## 3       16    W1         1        1        7    L1         2        1
```

```
nba.data1<-nba.data1[,-c(which(colnames(nba.data1)=="stk.x"|colnames(nba.data1)=="stk.y"))]
```

Building of Model

Because target variable "teamPTS" is numeric , we will use the regression techniques. Most common algorithms to predict numerical value: Linear Regression, Decision Trees, Neural Networks and K-Nearest Neighbors.

Before modelling, we need to determine high related independent variables with target value. To decide which independent variables effect the target variable more we should use dimension reduction techniques. Because of type of mixed independent variables (quantitive and qualitative), some reduction methods can either not suitable or avaliable such as LDA, Pearson's Correlation, ANOVA and Chi-Square.

We set Random Forest and Multiple Factor Analysis methods to reduction tehniques for our data. Because of implementing conveniently Random Forest is useful for feature selection in addition to being effective classifier. And Multiple Factor Analysis method works effectively in case the independent variables are in various groups. In this project we will use Random Forest method to select appropiate variables.

Dimension Reduction

As we have a target value, using random forest will be appropiate because that is one of the supervised dimension reduction techique .

```
require(randomForest)
names(nba.data1) <- make.names(names(nba.data1)) #there was an unappropiate column name for R
rf=randomForest(teamPTS~.,nba.data1[,-1],importance=T) #gmDate didn't take in model
i=importance(rf)
```

Since it is not a building model we did not divide dataset into training and test data.

We will sort independent variables according to importance. Then we will put the most important top 20,25,30,35 and 40 variables into our model by using loop iteration on next step.

Modelling

For numeric target value, we will apply regression machine learning techniques; Random Forest(RF), Generalised Linear Model (GLM), Generalized Boosted Regression Modeling(GBM) and Neural Networks (NN). In order to determine optimum number of independent variable, we need to calculate training and test data for each model separately.

To compare the errors of ML techniques with different number of independent variables in order of importance, we need to create an empty tracking matrix.

```
error.table <- matrix(NA, nrow=8, ncol=5)
rownames(error.table) <- c("train.error.RF","test.error.RF",
                           "train.error.GLM","test.error.GLM",
                           "train.error.NN","test.error.NN",
                           "train.error.GBM","test.error.GBM")
colnames(error.table) <- c("i=20","i=25","i=30","i=35","i=40")
```

Random Forest (RF)

We create the "best" vector to define most top 20,25,30,35 and 40 important independent variables. In Random Forest method, we specify the number of tree as 100.

```
set.seed(1021)
k=1
for (t in seq(20,40,5)){
best=rownames(i)[order(i[,"%IncMSE"],decreasing=T)[1:t]]
nba.data2<-nba.data1[,c(match(best, names(nba.data1)),7)] #7 is for adding "teamPTS" column
sp <- sample(1:nrow(nba.data2),0.8*nrow(nba.data2))
tr <- nba.data2[sp,]
ts <- nba.data2[-sp,]
model_rf<-randomForest(teamPTS~.,data=tr,ntree=100)
predict_rf.test<-predict(model_rf,ts,type='response')
predict_rf.train<-predict(model_rf,tr,type='response')
error.table[1,k]<-mean(abs((round(predict_rf.train))-(tr$teamPTS)))
error.table[2,k]<-mean(abs((round(predict_rf.test))-(ts$teamPTS)))
k=k+1
  }
```

Generalised Linear Model (GLM)

For GLM,assigning the types of independent variables precisely is essential for better modelling. We already have specified types of variables as factor or numeric on previous section. Also, our target variable "teamPTS" is non-negative so we choose the Gamma distribution family rather than Gaussian distribution family.

```
set.seed(1021)
k=1
for (t in seq(20,40,5)){
best=rownames(i)[order(i[,"%IncMSE"],decreasing=T)[1:t]]
nba.data2<-nba.data1[,c(match(best, names(nba.data1)),7)] #7 is for adding "teamPTS" column
sp <- sample(1:nrow(nba.data2),0.8*nrow(nba.data2))
tr <- nba.data2[sp,]
ts <- nba.data2[-sp,]
model_glm<-glm(teamPTS~.,data=tr,family=Gamma(link = "log"))
predict_glm.test<-predict(model_glm,ts,type='response')
predict_glm.train<-predict(model_glm,tr,type='response')
error.table[3,k]<-mean(abs((round(predict_glm.train))-(tr$teamPTS)))
error.table[4,k]<-mean(abs((round(predict_glm.test))-(ts$teamPTS)))
k=k+1
}
```

Neural Networks (NN)

Neural networks is a non-linear model which can solve both classification and regression tasks.That helps us cluster and classify.NN helps to group unlabeled data according to similarities among the example inputs, and

it classifies data when they have a labeled dataset to train on. In neural networks method, we need to specify number of units in the hidden layer as size, and decay parameter describes the weight of decay in model.

```
require(nnet)
set.seed(1021)
k=1
for (t in seq(20,40,5)){
best=rownames(i)[order(i[,"%IncMSE"],decreasing=T)[1:t]]
nba.data2<-nba.data1[,c(match(best, names(nba.data1)),7)] #7 is for adding "teamPTS" column
sp <- sample(1:nrow(nba.data2),0.8*nrow(nba.data2))
tr <- nba.data2[sp,]
ts <- nba.data2[-sp,]
model_nn<-nnet(teamPTS~.,data=tr,linout=TRUE,
           trace=FALSE,
           size=6,
           decay=0.01,
           maxit=2000)
predict_nn.test<-predict(model_nn,ts)
predict_nn.train<-predict(model_nn,tr)
error.table[5,k]<-mean(abs((round(predict_nn.train))-(tr$teamPTS)))
error.table[6,k]<-mean(abs((round(predict_nn.test))-(ts$teamPTS)))
k=k+1
}
```

Generalized Boosted Regression Modeling (GBM)

GBM is also known as MART (Multiple Additive Regression Trees) and GBRT (Gradient Boosted Regression Trees).

Whereas random forests build an ensemble of deep independent trees, GBMs build an ensemble of shallow and weak successive trees with each tree learning and improving on the previous. In GBM function we have to choose a family known as loss function: * “gaussian” family uses for minimizing squared error * “laplace” family uses for minimizing absolute loss * “bernoulli” family uses for logistic regression for 0-1 out-comes * “poisson” family uses count outcomes; requires the response to be a positive integer. We will choose the “laplace” family in our GBM function.

```
require(gbm)
set.seed(1021)
k=1
for (t in seq(20,40,5)){
best=rownames(i)[order(i[,"%IncMSE"],decreasing=T)[1:t]]
nba.data2<-nba.data1[,c(match(best, names(nba.data1)),7)] #7 is for adding "teamPTS" column
sp <- sample(1:nrow(nba.data2),0.8*nrow(nba.data2))
tr <- nba.data2[sp,]
ts <- nba.data2[-sp,]
model.gbm<-gbm(teamPTS~., data=tr, distribution = "laplace", n.trees = 100,
    bag.fraction = 0.75, cv.folds = 5, interaction.depth = 3)
predict_gbm.test<-predict(model.gbm,ts)
predict_gbm.train<-predict(model.gbm,tr)
error.table[7,k]<-mean(abs((round(predict_gbm.train))-(tr$teamPTS)))
error.table[8,k]<-mean(abs((round(predict_gbm.test))-(ts$teamPTS)))
k=k+1
}
```

```
## Using 100 trees...
## Using 100 trees...
```

```
## Using 98 trees...
## Using 98 trees...
## Using 100 trees...
## Using 100 trees...
## Using 99 trees...
## Using 99 trees...
## Using 87 trees...
## Using 87 trees...
```

Conclusion

We try to estimate result of match scores by using regression techniques rather than binary match result as win or loss because if we model this task well, we can reach easily other outcomes such as overall score, over/under on specific total scored (exp: under 240 pts). Calculation of these outcomes accurately plays essential role for betting company.

The training and test errors for ML techniques;

```
error.table
```

```
##                       i=20     i=25     i=30     i=35     i=40
## train.error.RF  3.175335 3.141844 3.111308 3.125493 3.069740
## test.error.RF   7.563780 7.597638 7.750394 7.329921 7.771654
## train.error.GLM 7.442080 7.483452 7.431442 7.417258 7.392435
## test.error.GLM  7.549606 7.251181 7.415748 7.464567 7.514173
## train.error.NN  7.379827 7.246454 7.350276 6.982467 7.897163
## test.error.NN   7.540157 7.538583 7.414173 8.153543 8.423622
## train.error.GBM 7.090426 7.154846 7.111899 7.190701 7.125887
## test.error.GBM  7.584252 7.371654 7.538583 7.237795 7.818110
```

When we check the result of error table, it can be seen that GLM model has the lowest **test error** 7.223622 with i=25 independent variables while Random Forests model has the lowest **training error** 3.078211 with i=40 independent variable.

According to the lowest training error, we can say that Random Forest is the best model to compute training error. But for lowest test error, almost all techniques have the similar result.

#References

[1] https://towardsdatascience.com/understanding-gradient-boosting-machines-9be756fe76ab