IST707 Final Project

Michael Morales

February 4, 2019

Load Libraries

library(stringr)  
library(e1071)  
library(randomForest)  
library(caret)  
library(rpart)  
library(rattle)  
library(rpart.plot)  
library(ggplot2)  
library(arules)  
library(naivebayes)  
library(factoextra)  
library(cluster)  
library(fpc)

Set WD

# Set working directory  
setwd("C:\\Users\\madmo\\OneDrive\\Syracuse\\IST707\\Final Project")  
  
# Load in the data  
worldCupData <- read.csv("WorldCups.csv")  
matchData <- read.csv("WorldCupMatches.csv")

Data Cleaning ########################################################################

# Remove ID variables from match data  
matchData <- matchData[ , -which(names(matchData) %in% c("RoundID","MatchID"))]  
  
# Remove Team Initials from match data  
matchData <- matchData[ , -which(names(matchData) %in% c("Home.Team.Initials","Away.Team.Initials"))]  
  
# Remove Date and only keep time in Datetime  
matchData$Datetime <- str\_extract(matchData$Datetime, '[0-9][0-9]:[0-9][0-9]')  
  
# Change column names in match data  
colnames(matchData) <- c("year", "time", "stage", "stadium", "city", "homeTeam", "hometeamGoals", "awayTeamGoals",   
 "awayTeam", "winConditions", "attendance", "halfTimeHomeGoals", "halfTimeAwayGoals",   
 "referee", "assistant1", "assistant2", "matchWin", "winPlace", "winWin")  
  
# Check for NA in match data  
summary(complete.cases(matchData))

## Mode FALSE TRUE   
## logical 2 850

# Remove Win Conditions Column from match data. Too many missing values.  
matchData <- matchData[ , -which(names(matchData) %in% c("winConditions"))]  
  
# Remove remaining records with missing values in match data  
matchData <- matchData[complete.cases(matchData),]  
  
# Get nationality of officials  
i <- 1  
for(row in matchData){  
 matchData$refereeNationality[i] <- gsub("[\\(\\)]", "", regmatches(matchData$referee[i],   
 gregexpr("\\(.\*?\\)",   
 matchData$referee[i]))[[1]])  
 matchData$assistant1Nationality[i] <- gsub("[\\(\\)]", "", regmatches(matchData$assistant1[i],   
 gregexpr("\\(.\*?\\)",   
 matchData$assistant1[i]))[[1]])  
 matchData$assistant2Nationality[i] <- gsub("[\\(\\)]", "", regmatches(matchData$assistant2[i],   
 gregexpr("\\(.\*?\\)",   
 matchData$assistant2[i]))[[1]])  
 i <- i + 1  
}  
  
# convert officials to char type  
matchData$referee <- as.character(matchData$referee)  
matchData$assistant1 <- as.character(matchData$assistant1)  
matchData$assistant2 <- as.character(matchData$assistant2)  
  
# remove nationality from officials names  
matchData$referee <- substr(matchData$referee,1,nchar(matchData$referee)-6)  
matchData$assistant1 <- substr(matchData$assistant1,1,nchar(matchData$assistant1)-6)  
matchData$assistant2 <- substr(matchData$assistant2,1,nchar(matchData$assistant2)-6)  
  
#Duplicate data frame to separate home and away teams  
homeDF <- matchData  
awayDF <- matchData  
  
# Get team name for each game for home and away teams  
homeDF$team <- homeDF$homeTeam  
awayDF$team <- awayDF$awayTeam  
  
# Get opponent name for each game for home and away teams  
homeDF$opponent <- homeDF$awayTeam  
awayDF$opponent <- awayDF$homeTeam  
  
# Get home or away  
homeDF$homeOrAway <- "home"  
awayDF$homeOrAway <- "away"  
  
# Remove home team and away team columns  
homeDF <- homeDF[ , -which(names(homeDF) %in% c("homeTeam", "awayTeam"))]  
awayDF <- awayDF[ , -which(names(awayDF) %in% c("homeTeam", "awayTeam"))]  
  
# iterate over home df and get results  
for(i in 1:length(homeDF$team)){  
 if(homeDF$hometeamGoals[i] > homeDF$awayTeamGoals[i]){  
 homeDF$result[i] <- "win"  
 }else if(homeDF$hometeamGoals[i] < homeDF$awayTeamGoals[i]){  
 homeDF$result[i] <- "loss"  
 }else{  
 homeDF$result[i] <- "draw"  
 }  
}  
  
# iterate over away df and get results  
for(i in 1:length(awayDF$team)){  
 if(awayDF$awayTeamGoals[i] > awayDF$hometeamGoals[i]){  
 awayDF$result[i] <- "win"  
 }else if(awayDF$awayTeamGoals[i] < awayDF$hometeamGoals[i]){  
 awayDF$result[i] <- "loss"  
 }else{  
 awayDF$result[i] <- "draw"  
 }  
}  
  
# Rename goals columns  
names(homeDF)[names(homeDF) == 'hometeamGoals'] <- 'goalsFor'  
names(homeDF)[names(homeDF) == 'awayTeamGoals'] <- 'goalsAgainst'  
names(homeDF)[names(homeDF) == 'halfTimeHomeGoals'] <- 'goalsForHalfTime'  
names(homeDF)[names(homeDF) == 'halfTimeAwayGoals'] <- 'goalsAgainstHalfTime'  
  
names(awayDF)[names(awayDF) == 'awayTeamGoals'] <- 'goalsFor'  
names(awayDF)[names(awayDF) == 'hometeamGoals'] <- 'goalsAgainst'  
names(awayDF)[names(awayDF) == 'halfTimeAwayGoals'] <- 'goalsForHalfTime'  
names(awayDF)[names(awayDF) == 'halfTimeHomeGoals'] <- 'goalsAgainstHalfTime'  
  
# combine homeDF and awayDF  
df <- rbind(homeDF, awayDF)  
  
# add empty host column to df  
df$host <- ""  
  
# add host nation to each world cup  
for(i in 1:length(df$team)){  
 if(df$year[i] == 1930){  
 df$host[i] <- "Uruguay"  
 }else if(df$year[i] == 1934){  
 df$host[i] <- "Italy"  
 }else if(df$year[i] == 1938){  
 df$host[i] <- "France"  
 }else if(df$year[i] == 1950){  
 df$host[i] <- "Brazil"  
 }else if(df$year[i] == 1954){  
 df$host[i] <- "Switzerland"  
 }else if(df$year[i] == 1958){  
 df$host[i] <- "Sweden"  
 }else if(df$year[i] == 1962){  
 df$host[i] <- "Chile"  
 }else if(df$year[i] == 1966){  
 df$host[i] <- "England"  
 }else if(df$year[i] == 1970){  
 df$host[i] <- "Mexico"  
 }else if(df$year[i] == 1974){  
 df$host[i] <- "Germany"  
 }else if(df$year[i] == 1978){  
 df$host[i] <- "Argentina"  
 }else if(df$year[i] == 1982){  
 df$host[i] <- "Spain"  
 }else if(df$year[i] == 1986){  
 df$host[i] <- "Mexico"  
 }else if(df$year[i] == 1990){  
 df$host[i] <- "Italy"  
 }else if(df$year[i] == 1994){  
 df$host[i] <- "United States"  
 }else if(df$year[i] == 1998){  
 df$host[i] <- "France"  
 }else if(df$year[i] == 2002){  
 df$host[i] <- "Japan and South Korea"  
 }else if(df$year[i] == 2006){  
 df$host[i] <- "Germany"  
 }else if(df$year[i] == 2010){  
 df$host[i] <- "South Africa"  
 }else if(df$year[i] == 2014){  
 df$host[i] <- "Brazil"  
 }else{  
 df$host[i] <- "Russia"  
 }  
}  
  
# add whether each country was host nation  
for(i in 1:length(df$team)){  
 if(df$host[i] == df$team[i]){  
 df$isHostCountry[i] <- "yes"  
 }else{  
 df$isHostCountry[i] <- "no"  
 }  
}  
  
# remove city and stadium columns from df (redundant with host attributes)  
df <- df[ , -which(names(homeDF) %in% c("stadium", "city"))]  
  
# get correct data types  
df$year <- as.factor(df$year)  
df$time <- as.factor(df$time)  
df$referee <- as.factor(df$referee)  
df$assistant1 <- as.factor(df$assistant1)  
df$assistant2 <- as.factor(df$assistant2)  
df$refereeNationality <- as.factor(df$refereeNationality)  
df$assistant1Nationality <- as.factor(df$assistant1Nationality)  
df$assistant2Nationality <- as.factor(df$assistant2Nationality)  
df$homeOrAway <- as.factor(df$homeOrAway)  
df$result <- as.factor(df$result)  
df$winPlace <- as.factor(df$winPlace)  
df$winWin <- as.factor(df$winWin)  
df$host <- as.factor(df$host)  
df$isHostCountry <- as.factor(df$isHostCountry)  
df$result <- as.factor(df$result)  
  
# remove unnecessary columns  
df <- df[ , -which(names(df) %in% c("referee", "assistant1", "assistant2", "matchWin", "team", "opponent"))]

Data Visualization ########################################################################

#Exploring trends in match results

plot(df$stage, df$attendance, col="red")

plot(df$isHostCountry, df$result) #much more likely to win if host country == yes

plot(df$isHostCountry, df$winPlace)

plot(df$isHostCountry, df$winWin)

plot(df$homeOrAway, df$result)

plot(df$result, df$goalsFor)

#Exploring Financial Results

plot(worldCupData$year, worldCupData$fifarev)

plot(worldCupData$year, worldCupData$hostcost)

plot(worldCupData$year, worldCupData$cost\_inf)

plot(worldCupData$year, worldCupData$rev\_inf)

plot(worldCupData$year, worldCupData$prizes)

worldCupData$prizes\_inf - worldCupData$hostcost

Association Rules ########################################################################

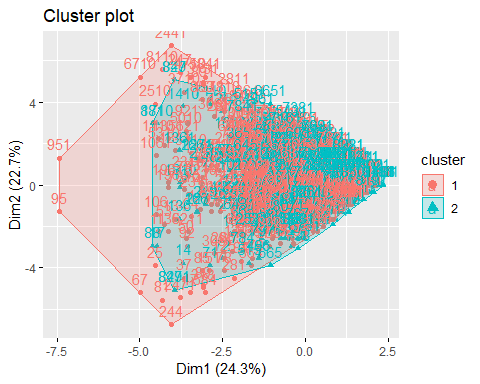
# get df for association rules  
ruleDF <- df  
  
# set all but attendance as factor  
ruleDF$goalsAgainst <- as.factor(ruleDF$goalsAgainst)  
ruleDF$goalsFor <- as.factor(ruleDF$goalsFor)  
ruleDF$goalsAgainstHalfTime <- as.factor(ruleDF$goalsAgainstHalfTime)  
ruleDF$goalsForHalfTime <- as.factor(ruleDF$goalsForHalfTime)  
  
# discretize attendance variable  
ruleDF$attendance <- as.factor(ifelse(ruleDF$attendance <= 30000,'low',ifelse(ruleDF$attendance <= 61381, 'average', 'high')))  
  
#Get association rules  
rules <- apriori(ruleDF, parameter = list(conf = 0.99, maxlen = 2), control = list(verbose=F))  
arules::inspect(rules)

## lhs rhs support confidence lift count  
## [1] {winWin=1} => {winPlace=1} 0.1058824 1.0000000 3.346457 180  
## [2] {host=Brazil} => {assistant2Nationality=BRA} 0.1176471 1.0000000 1.019185 200  
## [3] {host=Brazil} => {assistant1Nationality=BEL} 0.1176471 1.0000000 1.017964 200  
## [4] {host=Brazil} => {refereeNationality=URU} 0.1176471 1.0000000 1.015532 200  
## [5] {host=Germany} => {assistant2Nationality=BRA} 0.1200000 1.0000000 1.019185 204  
## [6] {host=Germany} => {assistant1Nationality=BEL} 0.1200000 1.0000000 1.017964 204  
## [7] {host=Germany} => {refereeNationality=URU} 0.1200000 1.0000000 1.015532 204  
## [8] {time=21:00} => {assistant2Nationality=BRA} 0.1282353 1.0000000 1.019185 218  
## [9] {time=21:00} => {assistant1Nationality=BEL} 0.1282353 1.0000000 1.017964 218  
## [10] {time=21:00} => {refereeNationality=URU} 0.1282353 1.0000000 1.015532 218  
## [11] {time=16:00} => {assistant2Nationality=BRA} 0.1482353 0.9921260 1.011160 252  
## [12] {time=16:00} => {assistant1Nationality=BEL} 0.1482353 0.9921260 1.009949 252  
## [13] {time=16:00} => {refereeNationality=URU} 0.1482353 0.9921260 1.007535 252  
## [14] {goalsAgainst=2} => {assistant2Nationality=BRA} 0.2005882 0.9941691 1.013242 341  
## [15] {goalsAgainst=2} => {assistant1Nationality=BEL} 0.2005882 0.9941691 1.012028 341  
## [16] {goalsAgainst=2} => {refereeNationality=URU} 0.2011765 0.9970845 1.012571 342  
## [17] {goalsFor=2} => {assistant2Nationality=BRA} 0.2005882 0.9941691 1.013242 341  
## [18] {goalsFor=2} => {assistant1Nationality=BEL} 0.2005882 0.9941691 1.012028 341  
## [19] {goalsFor=2} => {refereeNationality=URU} 0.2011765 0.9970845 1.012571 342  
## [20] {result=draw} => {assistant2Nationality=BRA} 0.2235294 1.0000000 1.019185 380  
## [21] {result=draw} => {assistant1Nationality=BEL} 0.2235294 1.0000000 1.017964 380  
## [22] {result=draw} => {refereeNationality=URU} 0.2235294 1.0000000 1.015532 380  
## [23] {attendance=low} => {isHostCountry=no} 0.2494118 0.9906542 1.049291 424  
## [24] {goalsFor=0} => {goalsForHalfTime=0} 0.2882353 1.0000000 1.694915 490  
## [25] {goalsAgainst=0} => {goalsAgainstHalfTime=0} 0.2882353 1.0000000 1.694915 490  
## [26] {attendance=average} => {assistant2Nationality=BRA} 0.4952941 0.9929245 1.011973 842  
## [27] {attendance=average} => {assistant1Nationality=BEL} 0.4964706 0.9952830 1.013162 844  
## [28] {attendance=average} => {refereeNationality=URU} 0.4952941 0.9929245 1.008346 842  
## [29] {winPlace=0} => {winWin=0} 0.7011765 1.0000000 1.118421 1192  
## [30] {winPlace=0} => {refereeNationality=URU} 0.6952941 0.9916107 1.007012 1182  
## [31] {assistant2Nationality=BRA} => {assistant1Nationality=BEL} 0.9800000 0.9988010 1.016743 1666  
## [32] {assistant1Nationality=BEL} => {assistant2Nationality=BRA} 0.9800000 0.9976048 1.016743 1666  
## [33] {assistant2Nationality=BRA} => {refereeNationality=URU} 0.9811765 1.0000000 1.015532 1668  
## [34] {refereeNationality=URU} => {assistant2Nationality=BRA} 0.9811765 0.9964158 1.015532 1668  
## [35] {assistant1Nationality=BEL} => {refereeNationality=URU} 0.9800000 0.9976048 1.013099 1666  
## [36] {refereeNationality=URU} => {assistant1Nationality=BEL} 0.9800000 0.9952210 1.013099 1666

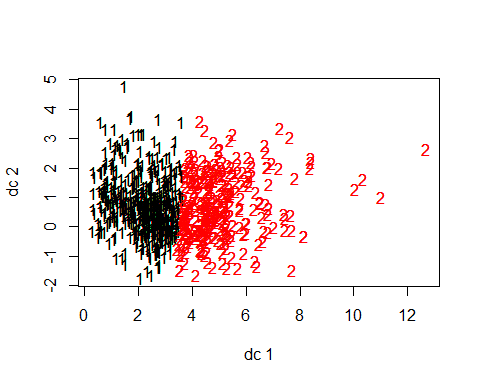
k-Means Clustering ########################################################################

#Set seed  
set.seed(474)  
#Set number of clusters  
k <- 2  
  
#make a DF with only numeric variables  
kmeansDF <- df[ , which(names(df) %in% c("year", "goalsFor", "goalsAgainst", "attendance", "goalsForHalfTime", "goalsAgainstHalfTime", "winPlace", "winWin"))]  
  
#make all variables as.numeric  
kmeansDF$year <- as.numeric(kmeansDF$year)  
kmeansDF$goalsFor <- as.numeric(kmeansDF$goalsFor)  
kmeansDF$goalsAgainst <- as.numeric(kmeansDF$goalsAgainst)  
kmeansDF$goalsFor <- as.numeric(kmeansDF$goalsFor)  
kmeansDF$attendance <- as.numeric(kmeansDF$attendance)  
kmeansDF$goalsForHalfTime <- as.numeric(kmeansDF$goalsForHalfTime)  
kmeansDF$goalsAgainstHalfTime <- as.numeric(kmeansDF$goalsAgainstHalfTime)  
kmeansDF$winPlace <- as.numeric(kmeansDF$winPlace)  
kmeansDF$winWin <- as.numeric(kmeansDF$winWin)  
kmeansResult <- kmeans(kmeansDF, k)  
  
#Observe clusters  
#(kmeansResult$cluster)

plot <- fviz\_cluster(kmeansResult, kmeansDF)  
plot



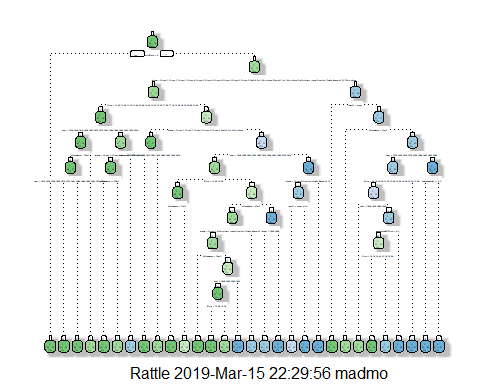
plotcluster(kmeansDF, kmeansResult$cluster)



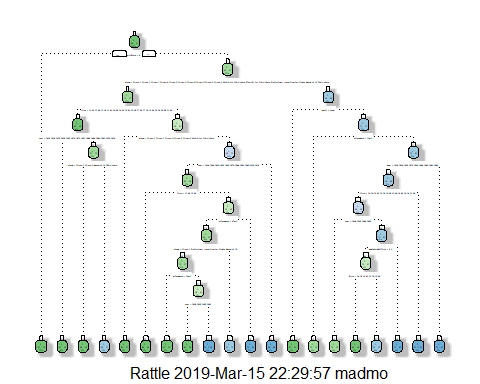
Decision Trees ########################################################################

# Set the random seed  
set.seed(474)  
# Split data set  
trainRows <- sample(1:nrow(df),0.80\*nrow(df))  
train <- df[trainRows, ]  
test <- df[-trainRows, ]  
  
#create a function to display results of decision tree for winWin  
get\_resultsWin <- function(clfWin){  
 #get predictions from the win tree  
 predictions <- predict(clfWin, test, type = "class")  
 #visualize the tree  
 fancyRpartPlot(clfWin)  
 #Get cross validation table  
 cvWin <- table(predictions, test$winWin)  
 print(cvWin)  
 #Get accuracy of win model  
 accuracy <- ((cvWin[1,1] + cvWin[2,2])/length(predictions))  
 print(accuracy)  
}  
  
#create a function to display results of decision tree for winPlace  
get\_resultsPlace <- function(clfPlace){  
 #get predictions from place the tree  
 predictions <- predict(clfPlace, test, type = "class")  
 #visualize the tree  
 fancyRpartPlot(clfPlace)  
 #Get cross validation table  
 cvPlace <- table(predictions, test$winPlace)  
 print(cvPlace)  
 #Get accuracy of place model  
 accuracy <- ((cvPlace[1,1] + cvPlace[2,2])/length(predictions))  
 print(accuracy)  
}  
  
#Create a function to prune win tree  
prune\_treeWin <- function(clfWin){  
 ptreeWin <- prune(clfWin, cp= clfWin$cptable[which.min(clfWin$cptable[,"xerror"]),"CP"])  
 get\_resultsWin(ptreeWin)  
}  
  
Winformula <- formula(winWin~., data = train)  
minSplit <- 1  
myCp = -1  
  
#Create a function to prune place tree  
prune\_treePlace <- function(clfPlace){  
 ptreePlace <- prune(clfPlace, cp= clfPlace$cptable[which.min(clfPlace$cptable[,"xerror"]),"CP"])  
 get\_resultsPlace(ptreePlace)  
}  
   
Placeformula <- formula(winPlace ~., data = train)  
minSplit <- 1  
myCp = -1  
  
  
#Create a function to easily test different tunings on the win tree  
get\_treeWin <- function(formula, minSplit, myCp){  
 myClfWin <- rpart(formula, data = train, method = "class", control = c(minsplit = minSplit, cp = myCp))  
 print("Full Win Tree Results")  
 get\_resultsWin(clfWin = myClfWin)  
 print("Pruned Win Tree Results")  
 prune\_treeWin(clfWin = myClfWin)  
}  
  
#Create a function to easily test different tunings on the place tree  
get\_treePlace <- function(formula, minSplit, myCp){  
 myClfPlace <- rpart(formula, data = train, method = "class", control = c(minsplit = minSplit, cp = myCp))  
 print("Full Place Tree Results")  
 get\_resultsPlace(clfPlace = myClfPlace)  
 print("Pruned Place Tree Results")  
 prune\_treePlace(clfPlace = myClfPlace)  
}  
  
#fit the win decision tree and view results  
get\_treeWin(Winformula, minSplit, myCp)

## [1] "Full Win Tree Results"



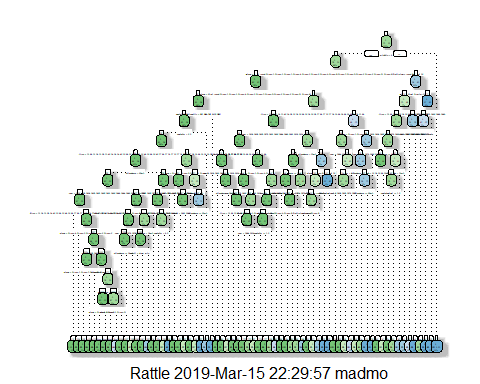
##   
## predictions 0 1  
## 0 299 10  
## 1 11 20  
## [1] 0.9382353  
## [1] "Pruned Win Tree Results"



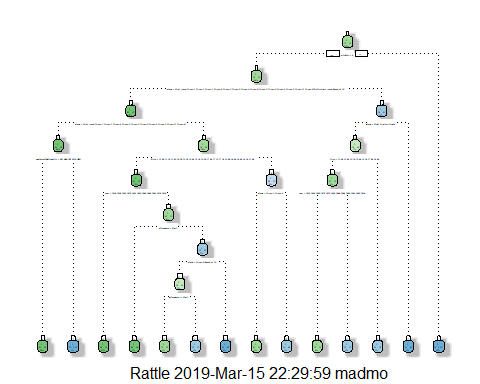
##   
## predictions 0 1  
## 0 299 10  
## 1 11 20  
## [1] 0.9382353

#fit the place decision tree and view results  
get\_treePlace(Placeformula, minSplit, myCp)

## [1] "Full Place Tree Results"



##   
## predictions 0 1  
## 0 221 30  
## 1 18 71  
## [1] 0.8588235  
## [1] "Pruned Place Tree Results"



##   
## predictions 0 1  
## 0 225 41  
## 1 14 60  
## [1] 0.8382353

Naive Bayes ########################################################################

# Set the random seed  
set.seed(474)  
# Split data set  
trainRows <- sample(1:nrow(df),0.80\*nrow(df))  
train <- df[trainRows, ]  
test <- df[-trainRows, ]  
  
#Win NB  
NB\_objectWin <- naive\_bayes(winWin~., data=train)  
NB\_predictionWin <-predict(NB\_objectWin, test[ , -which(names(test) %in% c("winPlace"))], type = c("class"))  
#head(predict(NB\_objectWin, test, type = "class"))  
table(NB\_predictionWin,test$winWin)

##   
## NB\_predictionWin 0 1  
## 0 291 16  
## 1 19 14

#Place NB  
NB\_objectPlace <- naive\_bayes(winPlace~., data=train)  
NB\_predictionPlace <-predict(NB\_objectPlace, test[ , -which(names(test) %in% c("winPlace"))], type = c("class"))  
#head(predict(NB\_objectPlace, test, type = "class"))  
table(NB\_predictionPlace,test$winPlace)

##   
## NB\_predictionPlace 0 1  
## 0 212 37  
## 1 27 64

Random Forest ########################################################################

# Set the random seed  
set.seed(474)  
# Split data set  
trainRows <- sample(1:nrow(df),0.80\*nrow(df))  
train <- df[trainRows, ]  
test <- df[-trainRows, ]  
  
n <- names(train)  
  
#Win  
fWin <- as.formula(paste("winWin ~", paste(n[!n %in% "winWin"], collapse = " + ")))  
  
#Place  
fPlace <- as.formula(paste("winPlace ~", paste(n[!n %in% "winPlace"], collapse = " + ")))  
  
# Set up RF Win  
rfWin <- randomForest(fWin, data = train)  
print(rfWin)

##   
## Call:  
## randomForest(formula = fWin, data = train)   
## Type of random forest: classification  
## Number of trees: 500  
## No. of variables tried at each split: 4  
##   
## OOB estimate of error rate: 5.66%  
## Confusion matrix:  
## 0 1 class.error  
## 0 1175 35 0.02892562  
## 1 42 108 0.28000000

# Set up RF Place  
rfPlace <- randomForest(fPlace, data = train)  
print(rfPlace)

##   
## Call:  
## randomForest(formula = fPlace, data = train)   
## Type of random forest: classification  
## Number of trees: 500  
## No. of variables tried at each split: 4  
##   
## OOB estimate of error rate: 12.65%  
## Confusion matrix:  
## 0 1 class.error  
## 0 896 57 0.05981112  
## 1 115 292 0.28255528

# Make predictions - Win  
winPredictions <- predict(rfWin, test)   
cvWin = (table(winPredictions, test$winWin))  
print(cvWin)

##   
## winPredictions 0 1  
## 0 300 5  
## 1 10 25

winAccuracy <- ((cvWin[1,1] + cvWin[2,2])/length(test$winWin))  
print(winAccuracy)

## [1] 0.9558824

# Make predictions - Place  
placePredictions <- predict(rfPlace, test)   
cvPlace = (table(placePredictions, test$winPlace))  
print(cvPlace)

##   
## placePredictions 0 1  
## 0 225 28  
## 1 14 73

placeAccuracy <- ((cvPlace[1,1] + cvPlace[2,2])/length(test$winPlace))  
print(placeAccuracy)

## [1] 0.8764706

k-Nearest Neighbor ########################################################################

# copy df  
dfOrig <- df  
  
# get all numeric variables  
df$year <- as.numeric(df$year)  
df$time <- as.numeric(df$time)  
df$stage <- as.numeric(df$stage)  
df$winWin <- as.numeric(df$winWin)  
df$winPlace <- as.numeric(df$winPlace)  
df$refereeNationality <- as.numeric(df$refereeNationality)  
df$assistant1Nationality <- as.numeric(df$assistant1Nationality)  
df$assistant2Nationality <- as.numeric(df$assistant2Nationality)  
df$homeOrAway <- as.numeric(df$homeOrAway)  
df$result <- as.numeric(df$result)  
df$host <- as.numeric(df$host)  
df$isHostCountry <- as.numeric(df$isHostCountry)  
  
# Set the random seed  
set.seed(474)  
# Split data set  
trainRows <- sample(1:nrow(df),0.80\*nrow(df))  
train <- df[trainRows, ]  
test <- df[-trainRows, ]  
  
  
# Set k  
k <- 4  
  
# Fit the Win model  
kNN\_fitWin <- class::knn(train=train, test=test, cl=train$winWin, k = k, prob=TRUE)  
#print(kNN\_fitWin)

## Check the win classification accuracy  
cvWin = (table(kNN\_fitWin, test$winWin))  
print(cvWin)

##   
## kNN\_fitWin 1 2  
## 1 286 21  
## 2 24 9

winAccuracy <- ((cvWin[1,1] + cvWin[2,2])/length(test$winWin))  
print(winAccuracy)

## [1] 0.8676471

# Fit the Place model  
kNN\_fitPlace <- class::knn(train=train, test=test, cl=train$winPlace, k = k, prob=TRUE)  
#print(kNN\_fitPlace)

## Check the place classification accuracy  
cvPlace = (table(kNN\_fitPlace, test$winPlace))  
print(cvPlace)

##   
## kNN\_fitPlace 1 2  
## 1 188 52  
## 2 51 49

placeAccuracy <- ((cvPlace[1,1] + cvPlace[2,2])/length(test$winPlace))  
print(placeAccuracy)

## [1] 0.6970588

Support Vector Machine ########################################################################

df <- dfOrig  
  
# Set random seed  
set.seed(474)  
# Split data set  
trainRows <- sample(1:nrow(df),0.80\*nrow(df))  
train <- df[trainRows, ]  
test <- df[-trainRows, ]  
  
# Set up Win SVM  
clfWin <- svm(winWin~., data=train)  
print(clfWin)

##   
## Call:  
## svm(formula = winWin ~ ., data = train)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: radial   
## cost: 1   
## gamma: 0.008264463   
##   
## Number of Support Vectors: 329

# Get Predictions  
(winPredictions <- predict(clfWin, test, type="class"))

## 7 12 16 19 26 29 41 55 61 65 69 72 92 110 111   
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## 112 118 124 125 127 138 139 140 147 148 149 163 170 175 185   
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## 189 199 200 201 202 205 206 213 216 217 228 241 250 252 254   
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## 255 258 259 260 264 267 268 275 277 282 284 285 290 291 293   
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## 295 296 297 298 301 304 307 309 310 316 325 330 342 345 350   
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## 351 352 358 359 369 372 373 387 390 405 407 415 423 424 433   
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## 434 446 447 456 457 461 462 467 468 473 486 487 492 497 500   
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## 502 504 509 512 521 528 529 534 550 553 559 570 571 572 578   
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## 580 581 582 585 586 594 596 600 608 614 618 626 643 644 653   
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## 656 671 672 678 683 689 711 713 719 720 723 726 728 733 738   
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## 747 751 752 756 773 777 781 792 797 804 805 807 813 814 816   
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## 821 825 826 829 831 832 847 850 1100 1510 1810 2710 3710 4110 4810   
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## 5310 5610 5710 6210 6410 6510 7810 8210 881 901 951 991 1021 1031 1061   
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## 1121 1161 1231 1311 1331 1341 1381 1461 1481 1511 1591 1611 1621 1641 1711   
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## 1731 1741 1771 1821 1851 1891 1911 1951 1991 2001 2041 2061 2131 2141 2161   
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## 2251 2281 2351 2491 2551 2611 2631 2671 2741 2861 2901 2961 3021 3081 3121   
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## 3411 3421 3461 3521 3541 3581 3621 3631 3751 3771 3811 3841 3901 3911 3981   
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## 4101 4221 4311 4391 4421 4491 4501 4531 4591 4651 4671 4711 4731 4801 4811   
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## 4831 5131 5171 5201 5271 5281 5361 5381 5411 5471 5491 5541 5581 5651 5691   
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## 5731 5761 5771 5821 5881 5891 5941 5971 6061 6111 6211 6281 6341 6451 6471   
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## 6491 6501 6551 6621 6701 6731 6781 6791 6841 6851 6921 6941 6951 7061 7331   
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## 7341 7411 7441 7461 7511 7531 7571 7611 7661 7691 7771 7861 7871 7921 8001   
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## 8031 8041 8051 8101 8141 8191 8261 8311 8441 8511   
## 0 0 0 0 0 0 0 0 0 0   
## Levels: 0 1

# Get confusion matrix  
(winCV <- table(winPredictions, test$winWin))

##   
## winPredictions 0 1  
## 0 310 30  
## 1 0 0

print(winCV)

##   
## winPredictions 0 1  
## 0 310 30  
## 1 0 0

winAccuracy <- ((winCV[1,1] + winCV[2,2])/length(test$winWin))  
print(winAccuracy)

## [1] 0.9117647

# Set up Place SVM  
clfPlace <- svm(winPlace~., data=train)  
print(clfPlace)

##   
## Call:  
## svm(formula = winPlace ~ ., data = train)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: radial   
## cost: 1   
## gamma: 0.008264463   
##   
## Number of Support Vectors: 779

# Get Predictions  
(placePredictions <- predict(clfPlace, test, type="class"))

## 7 12 16 19 26 29 41 55 61 65 69 72 92 110 111   
## 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0   
## 112 118 124 125 127 138 139 140 147 148 149 163 170 175 185   
## 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0   
## 189 199 200 201 202 205 206 213 216 217 228 241 250 252 254   
## 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0   
## 255 258 259 260 264 267 268 275 277 282 284 285 290 291 293   
## 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0   
## 295 296 297 298 301 304 307 309 310 316 325 330 342 345 350   
## 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0   
## 351 352 358 359 369 372 373 387 390 405 407 415 423 424 433   
## 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0   
## 434 446 447 456 457 461 462 467 468 473 486 487 492 497 500   
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## 502 504 509 512 521 528 529 534 550 553 559 570 571 572 578   
## 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1   
## 580 581 582 585 586 594 596 600 608 614 618 626 643 644 653   
## 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0   
## 656 671 672 678 683 689 711 713 719 720 723 726 728 733 738   
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## 747 751 752 756 773 777 781 792 797 804 805 807 813 814 816   
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## 821 825 826 829 831 832 847 850 1100 1510 1810 2710 3710 4110 4810   
## 0 0 1 1 0 0 0 0 0 0 1 0 0 0 1   
## 5310 5610 5710 6210 6410 6510 7810 8210 881 901 951 991 1021 1031 1061   
## 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0   
## 1121 1161 1231 1311 1331 1341 1381 1461 1481 1511 1591 1611 1621 1641 1711   
## 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0   
## 1731 1741 1771 1821 1851 1891 1911 1951 1991 2001 2041 2061 2131 2141 2161   
## 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0   
## 2251 2281 2351 2491 2551 2611 2631 2671 2741 2861 2901 2961 3021 3081 3121   
## 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0   
## 3411 3421 3461 3521 3541 3581 3621 3631 3751 3771 3811 3841 3901 3911 3981   
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## 4101 4221 4311 4391 4421 4491 4501 4531 4591 4651 4671 4711 4731 4801 4811   
## 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## 4831 5131 5171 5201 5271 5281 5361 5381 5411 5471 5491 5541 5581 5651 5691   
## 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0   
## 5731 5761 5771 5821 5881 5891 5941 5971 6061 6111 6211 6281 6341 6451 6471   
## 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0   
## 6491 6501 6551 6621 6701 6731 6781 6791 6841 6851 6921 6941 6951 7061 7331   
## 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0   
## 7341 7411 7441 7461 7511 7531 7571 7611 7661 7691 7771 7861 7871 7921 8001   
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   
## 8031 8041 8051 8101 8141 8191 8261 8311 8441 8511   
## 0 0 0 0 0 0 1 0 0 0   
## Levels: 0 1

# Get confusion matrix  
(placeCV <- table(placePredictions, test$winPlace))

##   
## placePredictions 0 1  
## 0 239 69  
## 1 0 32

print(placeCV)

##   
## placePredictions 0 1  
## 0 239 69  
## 1 0 32

placeAccuracy <- ((placeCV[1,1] + placeCV[2,2])/length(test$winPlace))  
print(placeAccuracy)

## [1] 0.7970588