NBA Shot Prediction

Alex Fung, Patrick Osborne, Tony Lee, Viswesh Krishnamurthy

14/03/2020

ABSTRACT

The use of various predictive metrics in sports has been occurring as long as human beings have watched each other compete. Initially this started out as simple gut feeling or a subjective assessment of the competitors – the bets usually go to the bigger fighter! In more recent times, humanity has refined its predictive power with the advent of statistics and logical decision making (famously put to use in Major League Baseball, as shown in the film Moneyball). With the advent of the information age and the possibility for large-scale data analytics and machine learning, the National Basketball Association has decided to puruse this analysis to better understand player matchups (defender vs offender) and to assess & optimize shooter performance.

BUSINESS UNDERSTANDING

Detailed statistics are already available to the NBA as these have been tracked for many years, supporting classic statistical decision making. The goal is to run both unsupervised and supervised machine learning algorithms on the statistical data available. In technical terms, we aim to deliver predictive metrics for threat/benefit level at an individual player level, as well as an interactive application that identifies the likelihood of a shot landing from a specific offender shooting from a specific position on the court, against a specific defender located a certain distance away. This will allow coaches to run limited scenarios in the predictive model, to inform both their practice routines and to assist in making strategic decisions during live games.

As an example, consider the matchup of Lebron James on offence and Serge Ibaka on defence. Let's assume that Lebron typically tries to shoot from top of the key, and is being defended by Serge Ibaka, 5 feet away. The model takes these discrete inputs and outputs a real-world percentage success of 10.5% (example). If the average shooting success rate is 30%, we can identify this as a bad shot, and encourage Lebron to pass in these situations.

DATA UNDERSTANDING

We begin with understanding each feature available in the data. The available data set is data of all shots attempted at NBA games between 2014 and 2015. For each shot attempted, the most important outcome of that attempt, whether the shot was made or missed is available. This is seen in 2 columns SHOT_RESULTS and FGM. FGM stands for "Field Goal Made". In support of this outcome, there are a number of other data points to be seen, like the player who attempted the shot and who defended the sho, how far away was the defender, how far away from the basket was the shot attempted, was the match at home or away etc. With that data understanding, let's look at the "head" of the data.

GAME_ID	MATCHUP	LOCATION	W	FINAL_MARGIN	SHOT_NUMBER	PERIOD	GAME_CLOCK
21400899	MAR 04, 2015 - CHA @ BKN	A	W	24	1	1	1:09
21400899	MAR 04, 2015 - CHA @ BKN	A	W	24	2	1	0:14
21400899	MAR 04, 2015 - CHA @ BKN	A	W	24	3	1	0:00
21400899	MAR 04, 2015 - CHA @ BKN	A	W	24	4	2	11:47
21400899	MAR 04, 2015 - CHA @ BKN	A	W	24	5	2	10:34
21400899	MAR 04, 2015 - CHA @ BKN	A	W	24	6	2	8:15

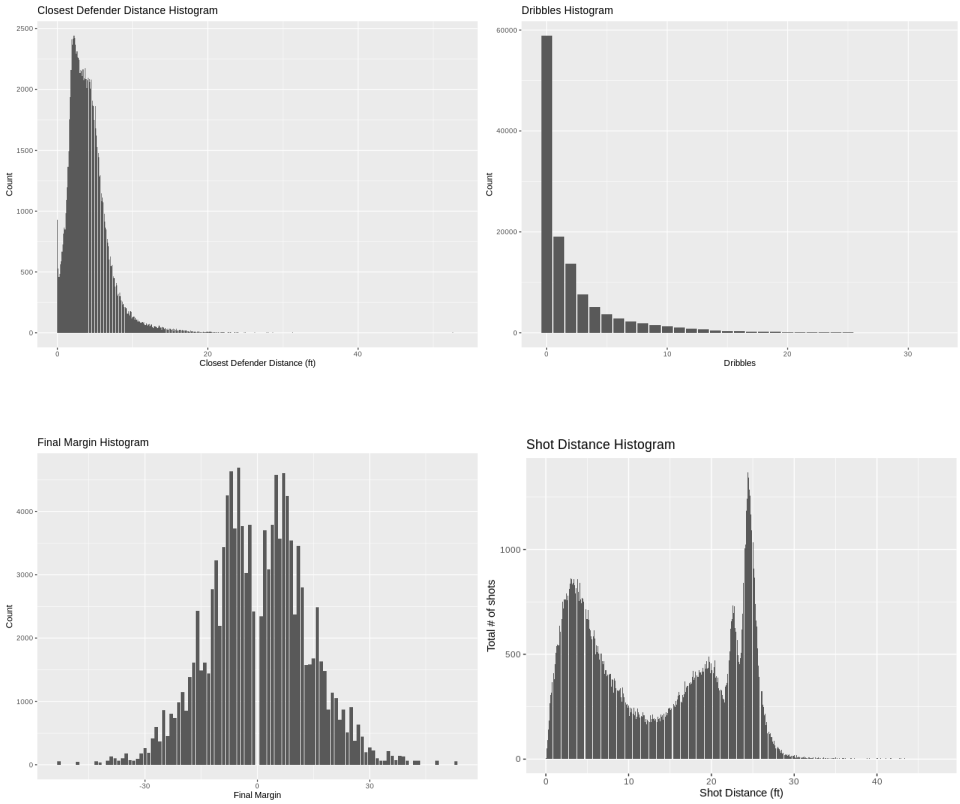
SHOT_CLOCK	DRIBBLES	TOUCH_TIME	SHOT_DIST	PTS_TYPE	SHOT_RESULT	CLOSEST_DEFENDER
10.80	2	1.90	7.70	2	made	Anderson, Alan
3.40	0	0.80	28.20	3	missed	Bogdanovic, Bojan
	3	2.70	10.10	2	missed	Bogdanovic, Bojan
10.30	2	1.90	17.20	2	missed	Brown, Markel
10.90	2	2.70	3.70	2	missed	Young, Thaddeus
9.10	2	4.40	18.40	2	missed	Williams, Deron

Table 1: Data Dictionary - NBA Data

Feature	Feature.Description
GAME_ID	A unique ID for each game
MATCHUP	Shows the date of the match and the teams the match is between
LOCATION	A - Away, H - Home. Shows the location of the match with respect to the first team in the match up column
W	Win or Loss. W means win and L means Loss, with respect to the first team in the match up column
FINAL_MARGIN	Points difference between the teams at the end of the game
SHOT_NUMBER	To be read in conjunction with the PERIOD column. Indicates the shot number in a given game period
PERIOD	Indicates the game period
GAME_CLOCK	Time elapsed since the period commenced. This datset shows the time at which the shot was attempted. Max 12 minutes per period
SHOT_CLOCK	The length of time for a given shot in seconds. Max - 24 seconds is a rule
DRIBBLES	The number of times the ball was dribbled before the shot was attempted
TOUCH_TIME	The length of time a player touched the ball
SHOT_DIST	The distance from which a shot was attempted. Distance in feet
PTS_TYPE	Points awarded if a shot was made. 2 pointer or 3 pointer shots
SHOT_RESULT	Indicates whether the shot was made or missed
CLOSEST_DEFENDER	Shows the name of the player that was the closest defender
CLOSEST_DEFENDER_PLAYER_ID	Unique ID of the closest defender
CLOSE_DEF_DIST	Distance of the closest defender in feet
FGM	An abbreviation for FIELD GOALS MADE. A proxy for SHOT_RESULT, 0 indicates missed shot and 1 indicates shot made
PTS	Points awarded for shots made
player_name	Name of the player who attempted the shot
player_id	Unique ID of the player who attempted the shot

Plots

We attempt to further understand the data using the following plots. A histogram of the "closest defender distance" shows that a majority of the shots were defended from within 5 feet of the player attempting the shot and it is safe to say that more than 90% of the shots were defended from within 10 feet. The "dribbles count" shows that most of the shots were attempted soon after getting the ball and that more than 80% of the shots were attempted within 3 dribbles. "Final Margin" histogram shows that most matches were won or lost within a 15 point margin. The "Shot distance" histogram shows that most of the shots were attempted from "top of the key" and followed by 2 to 4 feet range from the basket



DATA PREPARATION

SHOT_CLOCK

Looking at the data, some of the NA values need to be dealt with. The "SHOT_CLOCK" column has some NA values and the assumption is that the SHOT_CLOCK was equal to the GAME_CLOCK and therefore it may not be recorded. For such cases, the GAME_CLOCK is assumed to be equal to SHOT_CLOCK.

```
cleanData <- initialData
gameClock <- as.vector(second(fast_strptime(cleanData$GAME_CLOCK, "%M:%S"))) +
  as.vector(minute(fast_strptime(cleanData$GAME_CLOCK, "%M:%S"))) * 60
shotClock <- is.na(initialData$SHOT_CLOCK)
for(i in 1:length(gameClock)){
  if(shotClock[i] & gameClock[i] < 25){
    cleanData$SHOT_CLOCK[i] <- gameClock[i]
  }
}
```

Names

To further handle player names in this exercise, all names are standardized to read as "First Name" followed by "Last Name". A custom function was written to achieve this result.

```
nameformatreverse <- function(s) {
  fname <- str_extract(s, "^\\w+")
  lname <- str_extract(s, "\\w+$")
  s <- paste(lname, fname, sep = ", ")
```

```
}
```

All Shooter & Defender names are then put through the function to standardize names

```
shooterName <- cleanNoNAData$player_name
shooterName <- toupper(shooterName)
shooterName <- nameformatreverse(shooterName)
```

```
cleanNoNAData$player_name <- shooterName
cleanNoNAData$CLOSEST_DEFENDER <- toupper(cleanNoNAData$CLOSEST_DEFENDER)
cleanNoNAData$CLOSEST_DEFENDER <- gsub("[.]", "", cleanNoNAData$CLOSEST_DEFENDER)
```

Game Clock

It makes best sense to have the GAME_CLOCK expressed in seconds.

```
cleanNoNASecondsClockData <- cleanNoNAData
cleanNoNASecondsClockData$GAME_CLOCK <-
  as.vector(second(fast_strptime(cleanNoNAData$GAME_CLOCK, "%M:%S"))) +
  as.vector(minute(fast_strptime(cleanNoNAData$GAME_CLOCK, "%M:%S"))) * 60
```

Touch time

Any row that has TOUCH_TIME less than 0.1 seconds is not right and hence are omitted

```
cleanNoNASecondsClockData <- cleanNoNASecondsClockData[cleanNoNASecondsClockData$TOUCH_TIME > 0, ]
```

MODELLING

K-Means Clustering

The Elbow method is a popular, non computation intensive process of determining the most optimal number of clusters for a dataset by looking at a dropoff of variance. The other methods, eg, Bayesian Inference which we ran is more computation intensive, and produced optimal clusters that didnt agree with the visual Elbow method.Therefore, after plotting and analyzing a few different features against each other and highlighting the clusters by colouring the datapoints, we find that 3 clusters is likely the best compromise for the important features, namely, shot distance and closest defender distance.

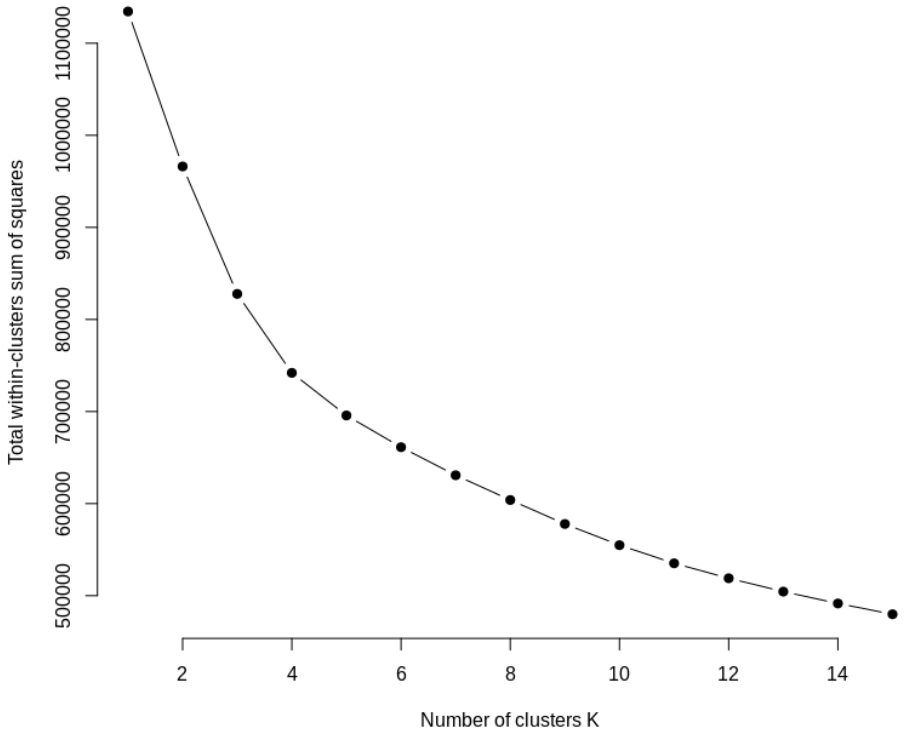
To perform clustering, only the numeric columns from the data are selected.

```
kdataunscaled <- cleanNoNASecondsClockData[, c("SHOT_NUMBER", "PERIOD",
                                               "GAME_CLOCK", "SHOT_CLOCK", "DRIBBLES",
                                               "TOUCH_TIME", "SHOT_DIST", "CLOSE_DEF_DIST")]
kdata <- scale(kdataunscaled)
```

We use the 'Elbow' method to determine the ideal number of clusters

```
set.seed(123)
# Compute and plot wss for k = 1 to k = 15
k.max <- 10
wss <- sapply(1:k.max, function(k){kmeans(kdata, k, nstart=50,iter.max = 15 )$tot.withinss})
wss
plot(1:k.max, wss,
     type="b", pch = 19, frame = FALSE,
     xlab="Number of clusters K",
     ylab="Total within-clusters sum of squares")
```

The "number of clusters" vs "sum of squares" plot helps us identify the 'Elbow' and decide on the right number of clusters. From the plot, we will try creating clusters with k = 2,3 & 4

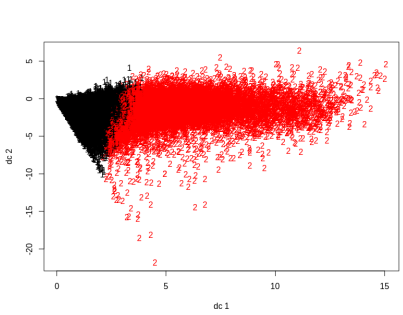


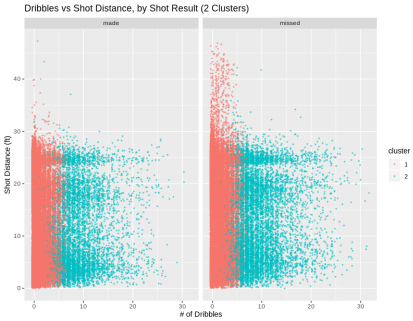
Bayesian Inference Criterion for k means to validate choice from Elbow Method

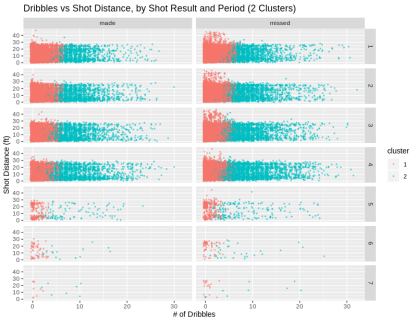
```
d_clust <- Mclust(as.matrix(kdata), G=1:10,
                  modelNames = mclust.options("emModelNames"))
d_clust$BIC
plot(d_clust)

# Let us apply kmeans for k=2 clusters
kmm.2 <- kmeans(kdata, 2, nstart = 50, iter.max = 15)
# Let us apply kmeans for k=3 clusters
kmm.3 <- kmeans(kdata, 3, nstart = 50, iter.max = 15)
# Let us apply kmeans for k=3 clusters
kmm.4 <- kmeans(kdata, 4, nstart = 50, iter.max = 15)
# We keep number of iter.max=15 to ensure the algorithm converges and nstart=50 to
# Ensure that atleat 50 random sets are choosen
kmm.2
kmm.3
kmm.4

# Plot the clusters
clusplot(kdataunscaled, kmm.3$cluster, color=TRUE, shade=TRUE, labels=2, lines=0)

# Centroid Plot against 1st 2 discriminant functions
plotcluster(kdataunscaled, kmm.2$cluster)
```

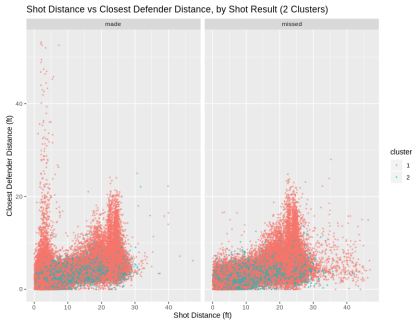
```
plotcluster(kdataunscaled, kmm.3$cluster)
plotcluster(kdataunscaled, kmm.4$cluster)
```

K2 plots

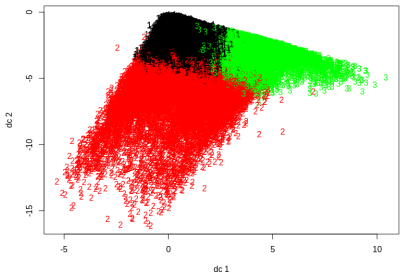


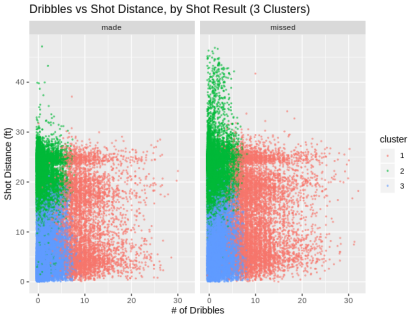


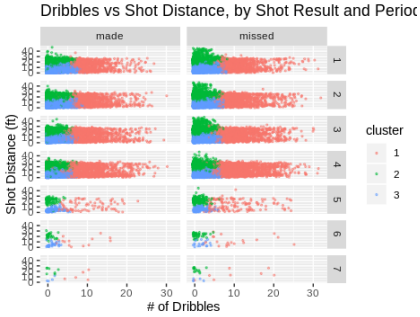


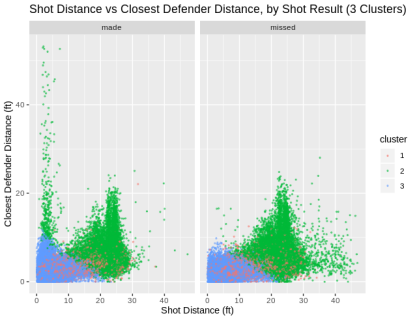


k3 plots


dc 1
dc 2

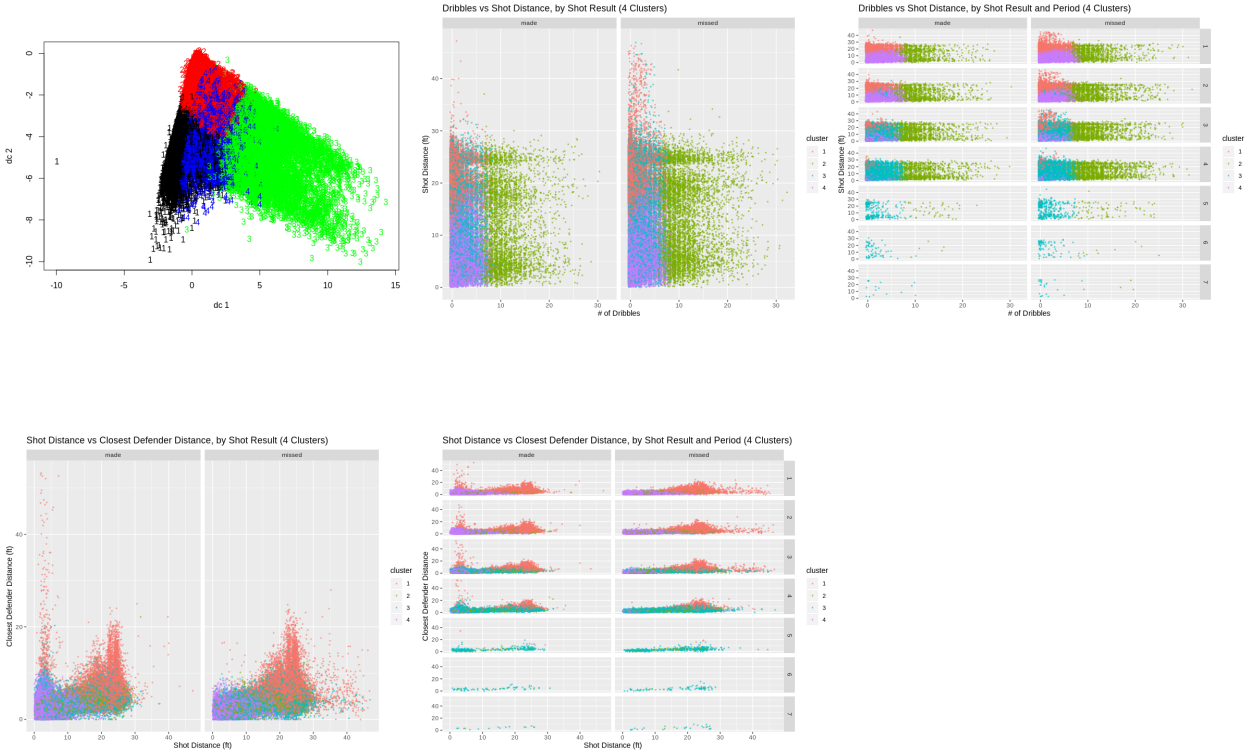

Dribbles vs Shot Distance, by Shot Result (3 Clusters)
made
missed
Shot Distance (ft)
# of Dribbles
cluster
1
2
3


Dribbles vs Shot Distance, by Shot Result and Period
made
missed
Shot Distance (ft)
# of Dribbles
cluster
1
2
3


Shot Distance vs Closest Defender Distance, by Shot Result (3 Clusters)
made
missed
Closest Defender Distance (ft)
Shot Distance (ft)
cluster
1
2
3


Shot Distance vs Closest Defender Distance, by Shot Result and Period (3
made
missed
Closest Defender Distance
Shot Distance (ft)
cluster
1
2
3

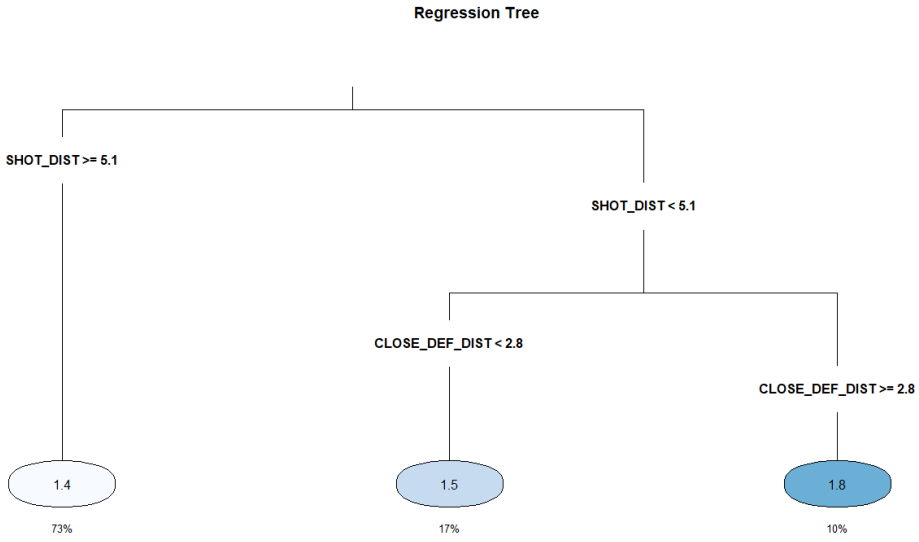
k4 plots



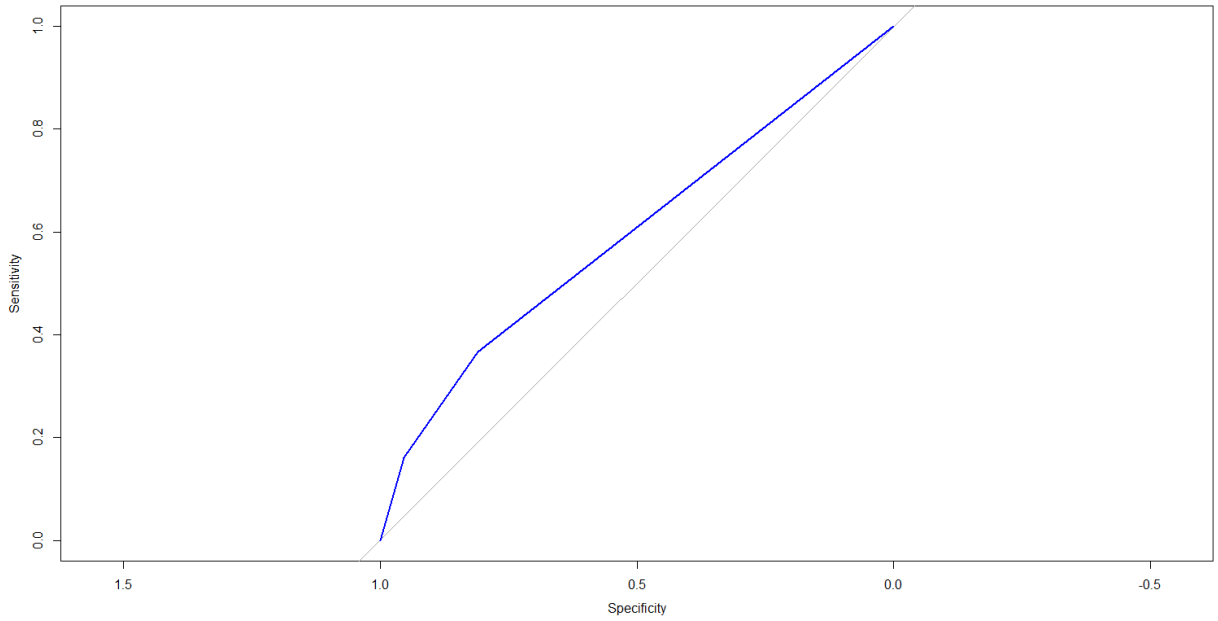
We chose a number of supervised machine learning models to predict the number of field goals made. We ran, evaluated, and compared the performance of the following models: * Decision Tree * Logistic Regression * GBM * GBM with PCA Before running the models, we converted the predictors and response variable to their correct data type. For example, numeric factors such as SHOT_DIST were explicitly converted into numeric, while categorical factors such as FGM were explicitly converted into categorical factors. Because the binary response variables of, 0 and 1, showcased a relatively balanced dataset, with the split being roughly 55%/45%, we did not need to undersample or oversample the data.
We also split the dataset into a training, and testing set. The split was set at 70% training, and 30% training, ensuring an equal

Decision Tree

The Decision tree algorithm is an algorithm that uses a tree-like data structure to make either predictions for regression, or classification problems. Given the business problem and context, a categorical variable decision tree was chosen as we wanted to classify, given the available data, whether or not an attempted shot made became a FGM (Field Goal Made); in this particular situation, there would be two categories for the response variable: either 0, denoting an attempted shot that missed, or 1, denoting an attempted shot that resulted in a field goal. A decision tree is suitable supervised machine learning algorithm because it is fairly easy to explain and visualize. For example, the following figure below is the generated decision tree diagram. Shot Distance, titled as SHOT_DIST, as well as the distance to the closest defender, titled as CLOSE_DEF_DIST, were the two most important variables in the decision tree, and of which the decisions are based upon.

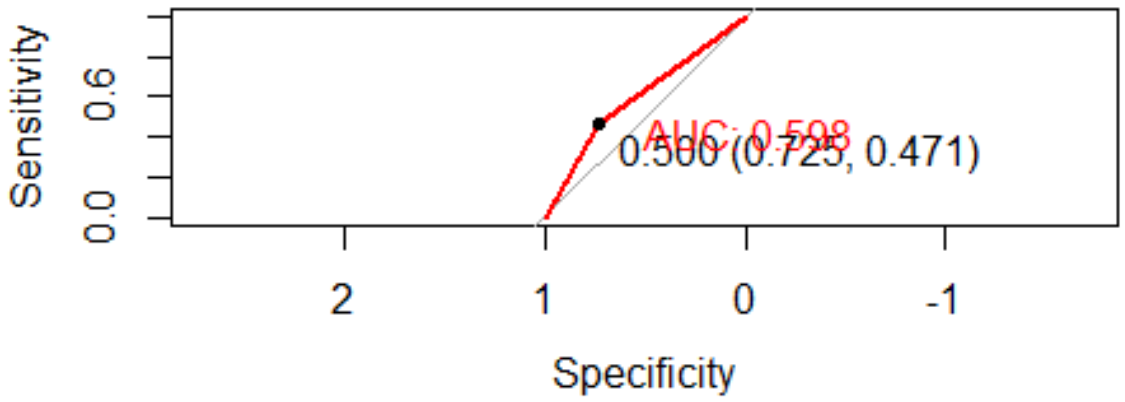


The ROC curve, and AUC of the decision tree algorithm showed similar results to the other models, as seen in the figure below. Nevertheless, it was decided not to use the decision tree algorithm as the model only showed 3 raw probabilities given the different permutations of SHOT_DIST and CLOSE_DEF_DIST. This would not have looked good on the Shiny app, as we wanted to show more probabilities on a more granular level, as there were other predictor variables that were not being used in the final model.



Logistic Regression

Logistic Regression is a parametric model used for binary classification, and it is based off the logit (logistic) function, hence the name. There are a few assumptions that were made with logistic regression: * There is a large enough dataset to make accurate predictions * Observations are independent of one another * Response variable is binary (0 or 1). * Predictor variables are related to logit function * Minimal multicollinearity among the predictor variables All assumptions except (4) and (5) are true, and due to the lack of time on our part, we were unable to ascertain whether the last two assumptions were, indeed, true. The results for the logistic regression showed similar results to other models. Listed below is the ROC curve/AUC results of the logistic regression model.



In terms of other metrics like accuracy, balanced accuracy, sensitivity, and specificity, the logistic regression model also showed similar results to the other models. Unfortunately, logistic regression took the longest amount of time to train compared to all the other models, at 2.5 hours. Listed below are the other metrics, as well as the runtime of the logistic regression model.

```
library(dplyr)
library(caTools)
library(pROC)
library(caret)
library(e1071)

# Read the data
shotDataRaw <- read.csv('../../data/shot_longs_clean_noNA_secondsclock.csv', header = TRUE, na.strings =

#columns to keep
shotData <- shotDataRaw[
  c(
    'LOCATION',
    'PERIOD',
    'GAME_CLOCK',
    'SHOT_CLOCK',
    'DRIBBLES',
    'TOUCH_TIME',
    'SHOT_DIST',
    'PTS_TYPE',
    'CLOSEST_DEFENDER',
    #'CLOSEST_DEFENDER_PLAYER_ID',
```

```
    'CLOSE_DEF_DIST',
    'player_name',
    'FGM'
  )
]

#scaling not required for glm (logistic regression)
#kdataunscaled <- shotData[, c("PERIOD", "GAME_CLOCK", "SHOT_CLOCK", "DRIBBLES", "TOUCH_TIME", "SHOT_DI
#kdata <- scale(kdataunscaled)

#make sure columns are set as factor, ordered, or numerical
shotData$LOCATION <- as.factor(shotData$LOCATION)
shotData$PERIOD <- as.factor(shotData$PERIOD)
shotData$GAME_CLOCK <- as.numeric(shotData$GAME_CLOCK)
shotData$SHOT_CLOCK <- as.numeric(shotData$SHOT_CLOCK)
shotData$DRIBBLES <- as.numeric(shotData$DRIBBLES)
shotData$TOUCH_TIME <- as.numeric(shotData$TOUCH_TIME)
shotData$SHOT_DIST <- as.numeric(shotData$SHOT_DIST)
shotData$PTS_TYPE <- as.factor(shotData$PTS_TYPE)
shotData$CLOSEST_DEFENDER <- as.factor(shotData$CLOSEST_DEFENDER)
shotData$CLOSE_DEF_DIST <- as.numeric(shotData$CLOSE_DEF_DIST)
shotData$player_name <- as.factor(shotData$player_name)

#glm is a bit weird, doesn't accept 1 or 0, so we will convert FGM into "yes" or "no"
shotData$FGM <- as.factor(
  ifelse(shotData$FGM == 0, "no", "yes")
)


#no need to under/oversample, because FGM of 0 is close to 55%, and FGM of 1 is close to 45%
#so pretty balanced dataset!

#split the data into training and testing datasets
set.seed(123)
shotSample = sample.split(shotData$FGM, SplitRatio = 0.70)
shotTrain = subset(shotData, shotSample == TRUE)
shotTest = subset(shotData, shotSample == FALSE)


#set a train control
#use cross validation
glm.trainControl = trainControl(
  method = "cv",
  number = 5,
  #Estimate class probabilities
  classProbs = TRUE,
  #Evaluate performance using the following function
  summaryFunction = twoClassSummary,
  allowParallel = TRUE,
  verbose = TRUE
)

#no need to tuneGrid because logistic regression using glm has no parameters
```

```
#train model
set.seed(123)
ptm_rf <- proc.time()
model_glm <- train(
  FGM ~ .,
  data = shotTrain,
  method = 'glm',
  trControl = glm.trainControl
)
proc.time() - ptm_rf

#make prediction against testData with the new model
print(model_glm)
pred.model_glm.prob = predict(model_glm, newdata = shotTest, type="prob")
pred.model_glm.raw = predict(model_glm, newdata = shotTest)

roc.model_glm = pROC::roc(
  shotTest$FGM,
  as.vector(ifelse(pred.model_glm.prob[,"yes"] > 0.5, 1, 0))
)
auc.model_glm = pROC::auc(roc.model_glm)
print(auc.model_glm)

#plot ROC curve
plot.roc(roc.model_glm, print.auc = TRUE, col = 'red', print.thres = "best")

#generate confusion matrix, as well as other metrics such as accuracy, balanced accuracy
confusionMatrix(data = pred.model_glm.raw, shotTest$FGM)

#summary of model
summary(model_glm)

# Save the model into a file
save(model_glm, file="model_glm.rda")
```

Stochastic Gradient Boosting (GBM)

Stochastic Gradient Boosting is a relatively complex supervised learning algorithm. The algorithm continuously iterates through several trees, at one at a time, so that it can boost the performance of its weakest learners.

When initially training the GBM model, we first decided to keep CLOSEST_DEFENDER_PLAYER_ID, and player_name, when training the GBM model, but the model itself was fairly big, and in terms of variable importance, both features were not that important unless if the players had a large enough data sample to draw meaningful conclusions from. For example, looking at the figure of output variable importance, one could see the top CLOSEST_DEFENDER and player_name names were highly regarded players from the 2015 season, or at least players who had a lot of playtime.

```
library(dplyr)
library(gbm)
library(caTools)
library(pROC)
library(doParallel)
library(caret)
library(MLmetrics)
```

```r
# Read the data
shotDataRaw <- read.csv('./source/data/shot_logs_clean_noNA_secondsclock.csv', header = TRUE, na.string

#columns to keep
shotData <- shotDataRaw[
  c(
    'LOCATION',
    'PERIOD',
    'GAME_CLOCK',
    'SHOT_CLOCK',
    'DRIBBLES',
    'TOUCH_TIME',
    'SHOT_DIST',
    'PTS_TYPE',
    'CLOSEST_DEFENDER',
    #'CLOSEST_DEFENDER_PLAYER_ID',
    'CLOSE_DEF_DIST',
    'player_name',
    'FGM'
  )
]

#scaling not required for gbm
#kdataunscaled <- shotData[, c("PERIOD", "GAME_CLOCK", "SHOT_CLOCK", "DRIBBLES", "TOUCH_TIME", "SHOT_DI
#kdata <- scale(kdataunscaled)

#make sure columns are set as factor, ordered, or numerical
shotData$LOCATION <- as.factor(shotData$LOCATION)
shotData$PERIOD <- as.factor(shotData$PERIOD)
shotData$GAME_CLOCK <- as.numeric(shotData$GAME_CLOCK)
shotData$SHOT_CLOCK <- as.numeric(shotData$SHOT_CLOCK)
shotData$DRIBBLES <- as.numeric(shotData$DRIBBLES)
shotData$TOUCH_TIME <- as.numeric(shotData$TOUCH_TIME)
shotData$SHOT_DIST <- as.numeric(shotData$SHOT_DIST)
shotData$PTS_TYPE <- as.factor(shotData$PTS_TYPE)
shotData$CLOSEST_DEFENDER <- as.factor(shotData$CLOSEST_DEFENDER)
shotData$CLOSE_DEF_DIST <- as.numeric(shotData$CLOSE_DEF_DIST)

#gbm is a bit weird, doesn't accept 1 or 0, so we will convert FGM into "yes" or "no"
shotData$FGM <- as.factor(
  ifelse(shotData$FGM == 0, "no", "yes")
)


#split the data into training and testing datasets
set.seed(123)
shotSample = sample.split(shotData$FGM, SplitRatio = 0.70)
shotTrain = subset(shotData, shotSample == TRUE)
shotTest = subset(shotData, shotSample == FALSE)

#set trainControl
#5-fold cross validation
gbm.trainControl = trainControl(
```

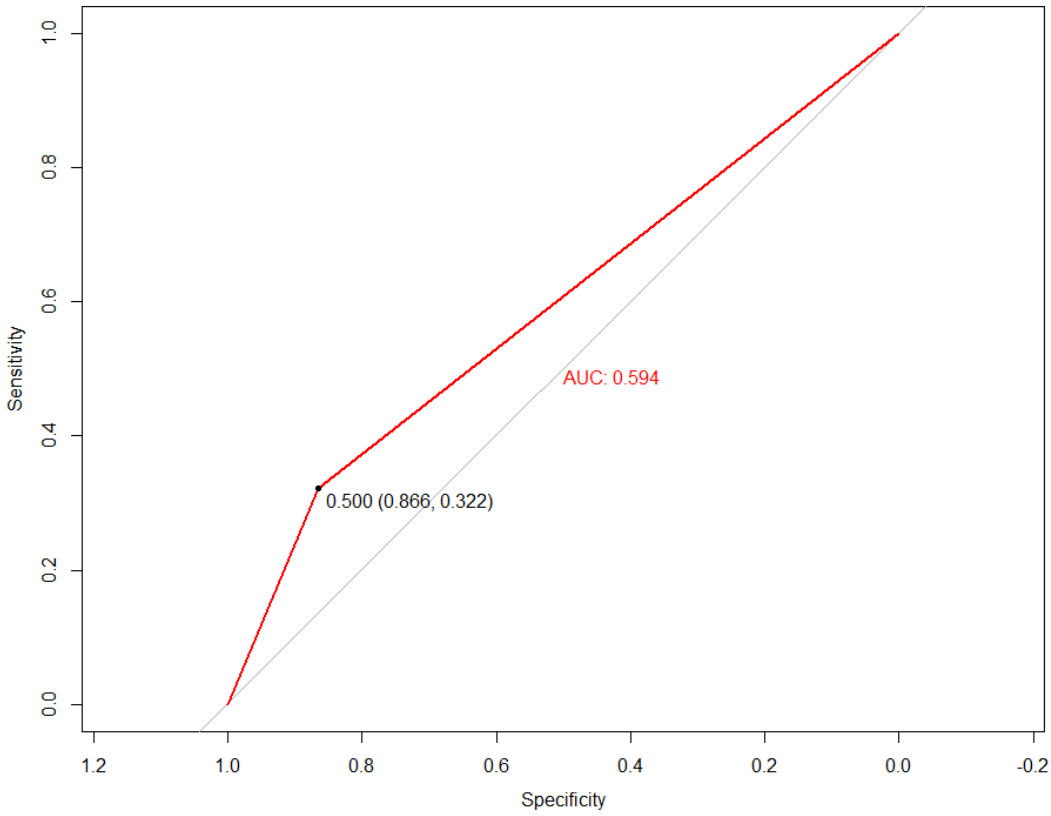
```
  method = "cv",
  number = 5,
  # Estimate class probabilities
  classProbs = TRUE,
  # Evaluate performance using the following function
  summaryFunction = twoClassSummary,
  allowParallel = TRUE,
  verbose = TRUE
)

#tuneGrid for GBM
gbmGrid <- expand.grid(
  #interaction.depth = c(10, 20),
  #n.trees = c(50, 100, 250),
  interaction.depth = c(5),
  n.trees = c(40),
  n.minobsinnode = 10,
  shrinkage = .1
)

#train model
set.seed(123)
ptm_rf <- proc.time()
model_gbm <- train(
  FGM ~ .,
  #data = data[trainSlices[[1]],],
  data = shotTrain,
  #data = train_data,
  method = "gbm",
  #family="gaussian",
  #distribution = "gaussian",
  trControl = gbm.trainControl,
  #tuneLength = 5
  tuneGrid = gbmGrid
)
proc.time() - ptm_rf

#make predictions aginst testData with the new model
print(model_gbm)
pred.model_gbm.prob = predict(model_gbm, newdata = shotTest, type="prob")
pred.model_gbm.raw = predict(model_gbm, newdata = shotTest)


roc.model_gbm = pROC::roc(
  shotTest$FGM,
  as.vector(ifelse(pred.model_gbm.prob[,"yes"] >0.5, 1,0))
)
auc.model_gbm = pROC::auc(roc.model_gbm)
print(auc.model_gbm)

#plot ROC curve
plot.roc(roc.model_gbm, print.auc = TRUE, col = 'red' , print.thres = "best" )
```

```r
#generate confusion matrix, as well as other metrics such as accuracy, balanced accuracy
confusionMatrix(data = pred.model_gbm.raw, shotTest$FGM)

#summary of model
summary(model_gbm)

# Save the model into a file
save(model_gbm, file="gbm.rda")
```

```
Confusion Matrix and Statistics

          Reference
Prediction    no   yes
       no  17477 11287
       yes  2703  5362

               Accuracy : 0.6201
                 95% CI : (0.6152, 0.6251)
    No Information Rate : 0.5479
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.197

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.8661
            Specificity : 0.3221
         Pos Pred Value : 0.6076
         Neg Pred Value : 0.6648
             Prevalence : 0.5479
         Detection Rate : 0.4745
   Detection Prevalence : 0.7810
      Balanced Accuracy : 0.5941

       'Positive' Class : no
```

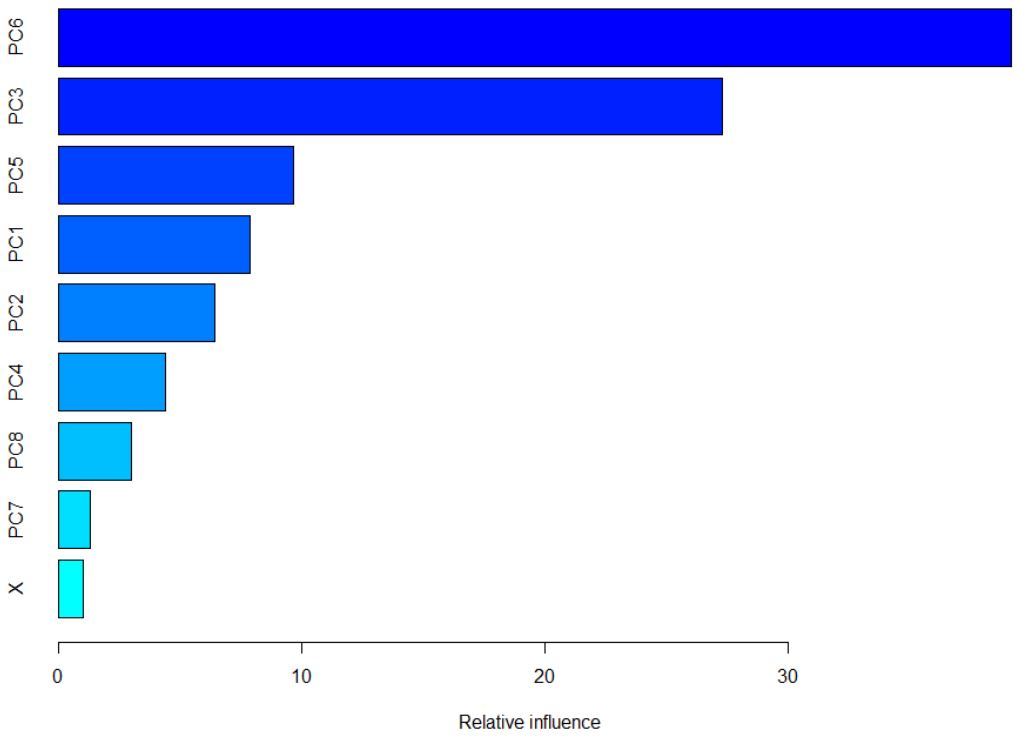
Also, including the features CLOSEST_DEFENDER_PLYAER_ID and player_name made the model size unnecessarily big for little to no improvement in the prediction of FGM. In relation to the size of the models, it also took a lot longer to train the model as well, at about 4 hours instead of the usual 1 hour. For hyper parameter optimization, we found that GBM produced the best results at with an interaction depth of 5, 40 trees, minimum of 10 observations in each node, and a shrinkage of 0.1. Using such hyper parameters offered a relatively fast training time of around 10 minutes

Listed below is the ROC curve, and AUC value for GBM without CLOSEST_DEFENDER_PLAYER_ID, and player_name predictor variables. Also listed below is the console output of training the GBM model, showing some of the metrics from the test dataset. Overall, the GBM model did the best compared to the other models, although not by a huge margin. It had an accuracy of 62.01%, a sensitivity of 0.8661 which were the highest of all the models. Unfortunately, its specificity was at the lower end compared to other models at 0.3221

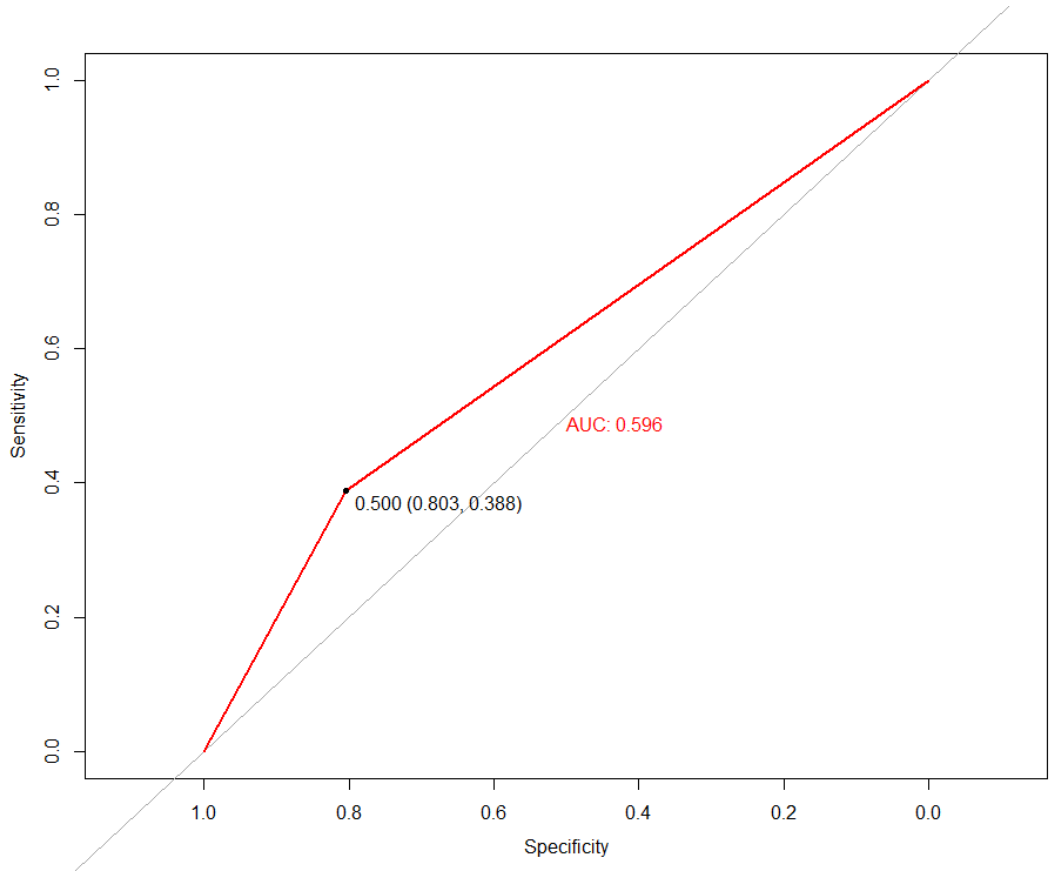


Stochastic Gradient Boosting (GBM) with PCA

We also chose to perform GBM with the results generated from PCA. We used all 8 of the components for the GBM model. Listed below was the variable importance of each component:



Listed below are the metrics of the GBM with PCA. Overall the GBM with PCA model was fairly competitive with the previous GBM model, as it scored 61.58% accuracy, and a 0.8035 sensitivity. Its specificity was a bit better than the previous GBM model at 0.3884. In terms of runtime both GBM models were fairly similar with a training duration of around 10 minutes.




```
> library(dplyr)
> library(caTools)
> library(pROC)
> library(caret)
> library(e1071)
>
> # Read the data
> shotDataRaw <- read.csv('../../data/shot_logs_pca.csv', header = TRUE, na.strings = c('NA','','#NA'))
> shotData <- shotDataRaw
>
> #scaling not required for gbm
> #kdataunscaled <- shotData[, c("PERIOD", "GAME_CLOCK", "SHOT_CLOCK", "DRIBBLES", "TOUCH_TIME", "SHOT_
> #kdata <- scale(kdataunscaled)
>
> #make sure columns are set as factor, ordered, or numerical
> shotData$PC1 <- as.numeric(shotData$PC1)
> shotData$PC2 <- as.numeric(shotData$PC2)
> shotData$PC3 <- as.numeric(shotData$PC3)
> shotData$PC4 <- as.numeric(shotData$PC4)
> shotData$PC5 <- as.numeric(shotData$PC5)
> shotData$PC6 <- as.numeric(shotData$PC6)
> shotData$PC7 <- as.numeric(shotData$PC7)
> shotData$PC8 <- as.numeric(shotData$PC8)
>
> #glm is a bit weird, doesn't accept 1 or 0, so we will convert FGM into "yes" or "no"
> shotData$FGM <- as.factor(
```

```
+   ifelse(shotData$FGM == 0, "no", "yes")
+ )
>
>
> #no need to under/oversample, because FGM of 0 is close to 55%, and FGM of 1 is close to 45%
> #so pretty balanced dataset!
>
> #split the data into training and testing datasets
> set.seed(123)
> shotSample = sample.split(shotData$FGM, SplitRatio = 0.70)
> shotTrain = subset(shotData, shotSample == TRUE)
> shotTest = subset(shotData, shotSample == FALSE)
>
>
> #set trainControl
> #5-fold cross validation
> gbm.trainControl = trainControl(
+   method = "cv",
+   number = 5,
+   # Estimate class probabilities
+   classProbs = TRUE,
+   # Evaluate performance using the following function
+   summaryFunction = twoClassSummary,
+   allowParallel = TRUE,
+   verbose = TRUE
+ )
>
> #tuneGrid for GBM
> gbmGrid <- expand.grid(
+   #interaction.depth = c(10, 20),
+   #n.trees = c(50, 100, 250),
+   interaction.depth = c(1, 2, 5, 10),
+   n.trees = c(25, 50, 100, 150, 200),
+   n.minobsinnode = 10,
+   shrinkage = .1
+ )
>
> #train model
> set.seed(123)
> ptm_rf <- proc.time()
> model_gbm <- train(
+   FGM ~ .,
+   #data = data[trainSlices[[1]],],
+   data = shotTrain,
+   #data = train_data,
+   method = "gbm",
+   #family="gaussian",
+   #distribution = "gaussian",
+   trControl = gbm.trainControl,
+   #tuneLength = 5
+   tuneGrid = gbmGrid
+ )
+ Fold1: shrinkage=0.1, interaction.depth= 1, n.minobsinnode=10, n.trees=200
```

```
Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.3734             nan     0.1000    0.0017
     2        1.3702             nan     0.1000    0.0015
     3        1.3675             nan     0.1000    0.0014
     4        1.3648             nan     0.1000    0.0013
     5        1.3622             nan     0.1000    0.0012
     6        1.3601             nan     0.1000    0.0010
     7        1.3582             nan     0.1000    0.0010
     8        1.3564             nan     0.1000    0.0009
     9        1.3548             nan     0.1000    0.0008
    10        1.3533             nan     0.1000    0.0007
    20        1.3432             nan     0.1000    0.0003
    40        1.3330             nan     0.1000    0.0002
    60        1.3272             nan     0.1000    0.0001
    80        1.3230             nan     0.1000    0.0000
   100        1.3200             nan     0.1000    0.0001
   120        1.3176             nan     0.1000    0.0000
   140        1.3159             nan     0.1000    0.0000
   160        1.3145             nan     0.1000    0.0000
   180        1.3134             nan     0.1000    0.0000
   200        1.3124             nan     0.1000    0.0000

- Fold1: shrinkage=0.1, interaction.depth= 1, n.minobsinnode=10, n.trees=200
+ Fold1: shrinkage=0.1, interaction.depth= 2, n.minobsinnode=10, n.trees=200
Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.3718             nan     0.1000    0.0026
     2        1.3673             nan     0.1000    0.0022
     3        1.3635             nan     0.1000    0.0018
     4        1.3598             nan     0.1000    0.0017
     5        1.3567             nan     0.1000    0.0014
     6        1.3540             nan     0.1000    0.0012
     7        1.3512             nan     0.1000    0.0013
     8        1.3489             nan     0.1000    0.0011
     9        1.3471             nan     0.1000    0.0009
    10        1.3452             nan     0.1000    0.0009
    20        1.3322             nan     0.1000    0.0004
    40        1.3192             nan     0.1000    0.0002
    60        1.3125             nan     0.1000    0.0001
    80        1.3087             nan     0.1000    0.0000
   100        1.3061             nan     0.1000    0.0000
   120        1.3042             nan     0.1000    0.0000
   140        1.3026             nan     0.1000    0.0000
   160        1.3016             nan     0.1000    0.0000
   180        1.3006             nan     0.1000   -0.0000
   200        1.2998             nan     0.1000   -0.0000

- Fold1: shrinkage=0.1, interaction.depth= 2, n.minobsinnode=10, n.trees=200
+ Fold1: shrinkage=0.1, interaction.depth= 5, n.minobsinnode=10, n.trees=200
Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.3689             nan     0.1000    0.0039
     2        1.3624             nan     0.1000    0.0031
     3        1.3570             nan     0.1000    0.0028
     4        1.3517             nan     0.1000    0.0024
```

```
     5        1.3473             nan     0.1000    0.0021
     6        1.3435             nan     0.1000    0.0018
     7        1.3401             nan     0.1000    0.0016
     8        1.3371             nan     0.1000    0.0013
     9        1.3345             nan     0.1000    0.0012
    10        1.3321             nan     0.1000    0.0011
    20        1.3166             nan     0.1000    0.0005
    40        1.3036             nan     0.1000    0.0000
    60        1.2985             nan     0.1000   -0.0000
    80        1.2952             nan     0.1000   -0.0000
   100        1.2928             nan     0.1000    0.0000
   120        1.2906             nan     0.1000   -0.0000
   140        1.2886             nan     0.1000    0.0000
   160        1.2866             nan     0.1000   -0.0000
   180        1.2848             nan     0.1000   -0.0000
   200        1.2830             nan     0.1000   -0.0000

- Fold1: shrinkage=0.1, interaction.depth= 5, n.minobsinnode=10, n.trees=200
+ Fold1: shrinkage=0.1, interaction.depth=10, n.minobsinnode=10, n.trees=200
Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.3676             nan     0.1000    0.0044
     2        1.3594             nan     0.1000    0.0039
     3        1.3528             nan     0.1000    0.0032
     4        1.3471             nan     0.1000    0.0026
     5        1.3427             nan     0.1000    0.0021
     6        1.3381             nan     0.1000    0.0022
     7        1.3344             nan     0.1000    0.0017
     8        1.3311             nan     0.1000    0.0015
     9        1.3280             nan     0.1000    0.0014
    10        1.3252             nan     0.1000    0.0013
    20        1.3084             nan     0.1000    0.0003
    40        1.2957             nan     0.1000    0.0001
    60        1.2899             nan     0.1000   -0.0000
    80        1.2855             nan     0.1000    0.0000
   100        1.2819             nan     0.1000   -0.0000
   120        1.2780             nan     0.1000   -0.0001
   140        1.2741             nan     0.1000   -0.0000
   160        1.2705             nan     0.1000   -0.0001
   180        1.2670             nan     0.1000   -0.0000
   200        1.2633             nan     0.1000   -0.0000

- Fold1: shrinkage=0.1, interaction.depth=10, n.minobsinnode=10, n.trees=200
+ Fold2: shrinkage=0.1, interaction.depth= 1, n.minobsinnode=10, n.trees=200
Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.3731             nan     0.1000    0.0019
     2        1.3699             nan     0.1000    0.0016
     3        1.3668             nan     0.1000    0.0016
     4        1.3641             nan     0.1000    0.0013
     5        1.3615             nan     0.1000    0.0013
     6        1.3592             nan     0.1000    0.0011
     7        1.3571             nan     0.1000    0.0010
     8        1.3553             nan     0.1000    0.0009
     9        1.3536             nan     0.1000    0.0008
```

```
    10        1.3519             nan     0.1000    0.0008
    20        1.3415             nan     0.1000    0.0003
    40        1.3317             nan     0.1000    0.0002
    60        1.3256             nan     0.1000    0.0001
    80        1.3213             nan     0.1000    0.0001
   100        1.3181             nan     0.1000    0.0001
   120        1.3158             nan     0.1000    0.0000
   140        1.3140             nan     0.1000    0.0000
   160        1.3126             nan     0.1000    0.0000
   180        1.3114             nan     0.1000    0.0000
   200        1.3105             nan     0.1000   -0.0000

- Fold2: shrinkage=0.1, interaction.depth= 1, n.minobsinnode=10, n.trees=200
+ Fold2: shrinkage=0.1, interaction.depth= 2, n.minobsinnode=10, n.trees=200
Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.3716             nan     0.1000    0.0026
     2        1.3670             nan     0.1000    0.0023
     3        1.3628             nan     0.1000    0.0021
     4        1.3594             nan     0.1000    0.0017
     5        1.3562             nan     0.1000    0.0015
     6        1.3529             nan     0.1000    0.0015
     7        1.3506             nan     0.1000    0.0011
     8        1.3483             nan     0.1000    0.0011
     9        1.3462             nan     0.1000    0.0010
    10        1.3443             nan     0.1000    0.0009
    20        1.3313             nan     0.1000    0.0004
    40        1.3182             nan     0.1000    0.0002
    60        1.3117             nan     0.1000    0.0001
    80        1.3073             nan     0.1000    0.0001
   100        1.3045             nan     0.1000    0.0000
   120        1.3026             nan     0.1000    0.0000
   140        1.3007             nan     0.1000    0.0000
   160        1.2994             nan     0.1000    0.0000
   180        1.2985             nan     0.1000   -0.0000
   200        1.2976             nan     0.1000   -0.0000

- Fold2: shrinkage=0.1, interaction.depth= 2, n.minobsinnode=10, n.trees=200
+ Fold2: shrinkage=0.1, interaction.depth= 5, n.minobsinnode=10, n.trees=200
Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.3691             nan     0.1000    0.0039
     2        1.3628             nan     0.1000    0.0031
     3        1.3569             nan     0.1000    0.0029
     4        1.3519             nan     0.1000    0.0025
     5        1.3479             nan     0.1000    0.0020
     6        1.3441             nan     0.1000    0.0017
     7        1.3408             nan     0.1000    0.0016
     8        1.3378             nan     0.1000    0.0014
     9        1.3352             nan     0.1000    0.0012
    10        1.3328             nan     0.1000    0.0012
    20        1.3169             nan     0.1000    0.0004
    40        1.3032             nan     0.1000    0.0002
    60        1.2978             nan     0.1000    0.0000
    80        1.2944             nan     0.1000   -0.0000
```

```
   100        1.2918             nan     0.1000   -0.0000
   120        1.2896             nan     0.1000   -0.0001
   140        1.2876             nan     0.1000   -0.0000
   160        1.2855             nan     0.1000   -0.0000
   180        1.2836             nan     0.1000   -0.0000
   200        1.2819             nan     0.1000   -0.0000

- Fold2: shrinkage=0.1, interaction.depth= 5, n.minobsinnode=10, n.trees=200
+ Fold2: shrinkage=0.1, interaction.depth=10, n.minobsinnode=10, n.trees=200
Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.3678             nan     0.1000    0.0044
     2        1.3595             nan     0.1000    0.0039
     3        1.3526             nan     0.1000    0.0032
     4        1.3465             nan     0.1000    0.0028
     5        1.3415             nan     0.1000    0.0025
     6        1.3370             nan     0.1000    0.0021
     7        1.3330             nan     0.1000    0.0018
     8        1.3297             nan     0.1000    0.0015
     9        1.3266             nan     0.1000    0.0014
    10        1.3237             nan     0.1000    0.0013
    20        1.3069             nan     0.1000    0.0004
    40        1.2941             nan     0.1000    0.0000
    60        1.2879             nan     0.1000    0.0000
    80        1.2831             nan     0.1000   -0.0000
   100        1.2792             nan     0.1000   -0.0001
   120        1.2754             nan     0.1000   -0.0001
   140        1.2717             nan     0.1000   -0.0000
   160        1.2681             nan     0.1000   -0.0001
   180        1.2647             nan     0.1000   -0.0001
   200        1.2610             nan     0.1000   -0.0000

- Fold2: shrinkage=0.1, interaction.depth=10, n.minobsinnode=10, n.trees=200
+ Fold3: shrinkage=0.1, interaction.depth= 1, n.minobsinnode=10, n.trees=200
Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.3732             nan     0.1000    0.0019
     2        1.3700             nan     0.1000    0.0015
     3        1.3669             nan     0.1000    0.0016
     4        1.3640             nan     0.1000    0.0013
     5        1.3616             nan     0.1000    0.0012
     6        1.3593             nan     0.1000    0.0011
     7        1.3573             nan     0.1000    0.0010
     8        1.3554             nan     0.1000    0.0009
     9        1.3537             nan     0.1000    0.0009
    10        1.3520             nan     0.1000    0.0007
    20        1.3413             nan     0.1000    0.0003
    40        1.3309             nan     0.1000    0.0002
    60        1.3248             nan     0.1000    0.0001
    80        1.3205             nan     0.1000    0.0001
   100        1.3174             nan     0.1000    0.0000
   120        1.3151             nan     0.1000    0.0000
   140        1.3134             nan     0.1000    0.0000
   160        1.3122             nan     0.1000    0.0000
   180        1.3111             nan     0.1000    0.0000
```

```
   200        1.3102             nan     0.1000   -0.0000

- Fold3: shrinkage=0.1, interaction.depth= 1, n.minobsinnode=10, n.trees=200
+ Fold3: shrinkage=0.1, interaction.depth= 2, n.minobsinnode=10, n.trees=200
Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.3717             nan     0.1000    0.0027
     2        1.3669             nan     0.1000    0.0023
     3        1.3626             nan     0.1000    0.0021
     4        1.3590             nan     0.1000    0.0017
     5        1.3560             nan     0.1000    0.0015
     6        1.3529             nan     0.1000    0.0014
     7        1.3504             nan     0.1000    0.0012
     8        1.3480             nan     0.1000    0.0011
     9        1.3458             nan     0.1000    0.0010
    10        1.3440             nan     0.1000    0.0009
    20        1.3307             nan     0.1000    0.0006
    40        1.3175             nan     0.1000    0.0001
    60        1.3103             nan     0.1000    0.0001
    80        1.3061             nan     0.1000    0.0000
   100        1.3036             nan     0.1000    0.0000
   120        1.3018             nan     0.1000   -0.0000
   140        1.3003             nan     0.1000   -0.0000
   160        1.2992             nan     0.1000    0.0000
   180        1.2983             nan     0.1000   -0.0000
   200        1.2975             nan     0.1000   -0.0000

- Fold3: shrinkage=0.1, interaction.depth= 2, n.minobsinnode=10, n.trees=200
+ Fold3: shrinkage=0.1, interaction.depth= 5, n.minobsinnode=10, n.trees=200
Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.3688             nan     0.1000    0.0040
     2        1.3621             nan     0.1000    0.0034
     3        1.3562             nan     0.1000    0.0028
     4        1.3513             nan     0.1000    0.0023
     5        1.3471             nan     0.1000    0.0020
     6        1.3433             nan     0.1000    0.0018
     7        1.3398             nan     0.1000    0.0016
     8        1.3370             nan     0.1000    0.0014
     9        1.3340             nan     0.1000    0.0013
    10        1.3314             nan     0.1000    0.0013
    20        1.3154             nan     0.1000    0.0004
    40        1.3026             nan     0.1000    0.0001
    60        1.2968             nan     0.1000   -0.0000
    80        1.2938             nan     0.1000   -0.0000
   100        1.2912             nan     0.1000   -0.0000
   120        1.2891             nan     0.1000   -0.0000
   140        1.2871             nan     0.1000   -0.0000
   160        1.2848             nan     0.1000   -0.0000
   180        1.2827             nan     0.1000   -0.0000
   200        1.2809             nan     0.1000   -0.0000

- Fold3: shrinkage=0.1, interaction.depth= 5, n.minobsinnode=10, n.trees=200
+ Fold3: shrinkage=0.1, interaction.depth=10, n.minobsinnode=10, n.trees=200
Iter   TrainDeviance   ValidDeviance   StepSize   Improve
```

```
     1        1.3673             nan     0.1000    0.0046
     2        1.3591             nan     0.1000    0.0040
     3        1.3523             nan     0.1000    0.0032
     4        1.3465             nan     0.1000    0.0028
     5        1.3414             nan     0.1000    0.0023
     6        1.3367             nan     0.1000    0.0022
     7        1.3327             nan     0.1000    0.0018
     8        1.3294             nan     0.1000    0.0015
     9        1.3263             nan     0.1000    0.0014
    10        1.3234             nan     0.1000    0.0013
    20        1.3059             nan     0.1000    0.0004
    40        1.2936             nan     0.1000   -0.0000
    60        1.2877             nan     0.1000    0.0000
    80        1.2834             nan     0.1000   -0.0001
   100        1.2792             nan     0.1000   -0.0000
   120        1.2750             nan     0.1000    0.0000
   140        1.2716             nan     0.1000   -0.0000
   160        1.2679             nan     0.1000   -0.0001
   180        1.2644             nan     0.1000   -0.0001
   200        1.2608             nan     0.1000   -0.0000

- Fold3: shrinkage=0.1, interaction.depth=10, n.minobsinnode=10, n.trees=200
+ Fold4: shrinkage=0.1, interaction.depth= 1, n.minobsinnode=10, n.trees=200
Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.3734             nan     0.1000    0.0018
     2        1.3702             nan     0.1000    0.0015
     3        1.3672             nan     0.1000    0.0016
     4        1.3646             nan     0.1000    0.0012
     5        1.3621             nan     0.1000    0.0012
     6        1.3599             nan     0.1000    0.0011
     7        1.3579             nan     0.1000    0.0010
     8        1.3559             nan     0.1000    0.0009
     9        1.3542             nan     0.1000    0.0008
    10        1.3526             nan     0.1000    0.0008
    20        1.3424             nan     0.1000    0.0003
    40        1.3320             nan     0.1000    0.0001
    60        1.3260             nan     0.1000    0.0001
    80        1.3217             nan     0.1000    0.0001
   100        1.3186             nan     0.1000    0.0000
   120        1.3163             nan     0.1000    0.0000
   140        1.3145             nan     0.1000    0.0000
   160        1.3132             nan     0.1000    0.0000
   180        1.3120             nan     0.1000    0.0000
   200        1.3112             nan     0.1000   -0.0000

- Fold4: shrinkage=0.1, interaction.depth= 1, n.minobsinnode=10, n.trees=200
+ Fold4: shrinkage=0.1, interaction.depth= 2, n.minobsinnode=10, n.trees=200
Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.3718             nan     0.1000    0.0025
     2        1.3667             nan     0.1000    0.0025
     3        1.3628             nan     0.1000    0.0019
     4        1.3592             nan     0.1000    0.0017
     5        1.3560             nan     0.1000    0.0015
```

```
     6        1.3532             nan     0.1000    0.0013
     7        1.3508             nan     0.1000    0.0012
     8        1.3485             nan     0.1000    0.0010
     9        1.3463             nan     0.1000    0.0010
    10        1.3444             nan     0.1000    0.0009
    20        1.3313             nan     0.1000    0.0004
    40        1.3181             nan     0.1000    0.0002
    60        1.3112             nan     0.1000    0.0001
    80        1.3072             nan     0.1000    0.0000
   100        1.3046             nan     0.1000    0.0000
   120        1.3027             nan     0.1000    0.0000
   140        1.3014             nan     0.1000   -0.0000
   160        1.3000             nan     0.1000   -0.0000
   180        1.2991             nan     0.1000   -0.0000
   200        1.2982             nan     0.1000   -0.0000

- Fold4: shrinkage=0.1, interaction.depth= 2, n.minobsinnode=10, n.trees=200
+ Fold4: shrinkage=0.1, interaction.depth= 5, n.minobsinnode=10, n.trees=200
Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.3691             nan     0.1000    0.0039
     2        1.3627             nan     0.1000    0.0032
     3        1.3569             nan     0.1000    0.0028
     4        1.3518             nan     0.1000    0.0025
     5        1.3476             nan     0.1000    0.0020
     6        1.3439             nan     0.1000    0.0017
     7        1.3408             nan     0.1000    0.0015
     8        1.3379             nan     0.1000    0.0013
     9        1.3350             nan     0.1000    0.0013
    10        1.3323             nan     0.1000    0.0012
    20        1.3165             nan     0.1000    0.0005
    40        1.3032             nan     0.1000    0.0002
    60        1.2980             nan     0.1000   -0.0000
    80        1.2950             nan     0.1000   -0.0000
   100        1.2924             nan     0.1000    0.0000
   120        1.2905             nan     0.1000   -0.0000
   140        1.2887             nan     0.1000   -0.0000
   160        1.2867             nan     0.1000   -0.0000
   180        1.2846             nan     0.1000   -0.0000
   200        1.2827             nan     0.1000   -0.0000

- Fold4: shrinkage=0.1, interaction.depth= 5, n.minobsinnode=10, n.trees=200
+ Fold4: shrinkage=0.1, interaction.depth=10, n.minobsinnode=10, n.trees=200
Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.3675             nan     0.1000    0.0046
     2        1.3595             nan     0.1000    0.0039
     3        1.3523             nan     0.1000    0.0033
     4        1.3464             nan     0.1000    0.0027
     5        1.3411             nan     0.1000    0.0024
     6        1.3365             nan     0.1000    0.0022
     7        1.3327             nan     0.1000    0.0018
     8        1.3292             nan     0.1000    0.0016
     9        1.3263             nan     0.1000    0.0013
    10        1.3236             nan     0.1000    0.0011
```

```
    20        1.3068             nan     0.1000    0.0005
    40        1.2943             nan     0.1000    0.0000
    60        1.2881             nan     0.1000    0.0000
    80        1.2836             nan     0.1000   -0.0000
   100        1.2793             nan     0.1000   -0.0001
   120        1.2752             nan     0.1000   -0.0000
   140        1.2714             nan     0.1000    0.0000
   160        1.2678             nan     0.1000   -0.0000
   180        1.2642             nan     0.1000   -0.0001
   200        1.2606             nan     0.1000   -0.0000

- Fold4: shrinkage=0.1, interaction.depth=10, n.minobsinnode=10, n.trees=200
+ Fold5: shrinkage=0.1, interaction.depth= 1, n.minobsinnode=10, n.trees=200
Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.3734             nan     0.1000    0.0018
     2        1.3702             nan     0.1000    0.0015
     3        1.3671             nan     0.1000    0.0015
     4        1.3645             nan     0.1000    0.0012
     5        1.3621             nan     0.1000    0.0012
     6        1.3600             nan     0.1000    0.0009
     7        1.3581             nan     0.1000    0.0010
     8        1.3562             nan     0.1000    0.0009
     9        1.3546             nan     0.1000    0.0008
    10        1.3530             nan     0.1000    0.0008
    20        1.3427             nan     0.1000    0.0003
    40        1.3326             nan     0.1000    0.0002
    60        1.3267             nan     0.1000    0.0001
    80        1.3227             nan     0.1000    0.0001
   100        1.3196             nan     0.1000    0.0000
   120        1.3174             nan     0.1000    0.0000
   140        1.3157             nan     0.1000    0.0000
   160        1.3143             nan     0.1000    0.0000
   180        1.3133             nan     0.1000    0.0000
   200        1.3124             nan     0.1000   -0.0000

- Fold5: shrinkage=0.1, interaction.depth= 1, n.minobsinnode=10, n.trees=200
+ Fold5: shrinkage=0.1, interaction.depth= 2, n.minobsinnode=10, n.trees=200
Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.3720             nan     0.1000    0.0024
     2        1.3672             nan     0.1000    0.0023
     3        1.3633             nan     0.1000    0.0019
     4        1.3595             nan     0.1000    0.0018
     5        1.3563             nan     0.1000    0.0015
     6        1.3538             nan     0.1000    0.0013
     7        1.3513             nan     0.1000    0.0011
     8        1.3490             nan     0.1000    0.0011
     9        1.3470             nan     0.1000    0.0009
    10        1.3450             nan     0.1000    0.0009
    20        1.3323             nan     0.1000    0.0004
    40        1.3196             nan     0.1000    0.0001
    60        1.3127             nan     0.1000    0.0001
    80        1.3087             nan     0.1000    0.0001
   100        1.3062             nan     0.1000   -0.0000
```

```
   120        1.3044             nan     0.1000    0.0000
   140        1.3030             nan     0.1000   -0.0000
   160        1.3021             nan     0.1000   -0.0000
   180        1.3013             nan     0.1000   -0.0000
   200        1.3003             nan     0.1000    0.0000

- Fold5: shrinkage=0.1, interaction.depth= 2, n.minobsinnode=10, n.trees=200
+ Fold5: shrinkage=0.1, interaction.depth= 5, n.minobsinnode=10, n.trees=200
Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.3695             nan     0.1000    0.0037
     2        1.3630             nan     0.1000    0.0032
     3        1.3574             nan     0.1000    0.0027
     4        1.3525             nan     0.1000    0.0024
     5        1.3480             nan     0.1000    0.0022
     6        1.3445             nan     0.1000    0.0016
     7        1.3415             nan     0.1000    0.0013
     8        1.3382             nan     0.1000    0.0016
     9        1.3355             nan     0.1000    0.0012
    10        1.3330             nan     0.1000    0.0012
    20        1.3172             nan     0.1000    0.0006
    40        1.3042             nan     0.1000    0.0001
    60        1.2990             nan     0.1000    0.0000
    80        1.2959             nan     0.1000   -0.0000
   100        1.2932             nan     0.1000   -0.0001
   120        1.2909             nan     0.1000    0.0000
   140        1.2888             nan     0.1000   -0.0000
   160        1.2868             nan     0.1000   -0.0000
   180        1.2848             nan     0.1000   -0.0000
   200        1.2829             nan     0.1000   -0.0000

- Fold5: shrinkage=0.1, interaction.depth= 5, n.minobsinnode=10, n.trees=200
+ Fold5: shrinkage=0.1, interaction.depth=10, n.minobsinnode=10, n.trees=200
Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.3674             nan     0.1000    0.0046
     2        1.3596             nan     0.1000    0.0036
     3        1.3528             nan     0.1000    0.0032
     4        1.3468             nan     0.1000    0.0028
     5        1.3419             nan     0.1000    0.0023
     6        1.3378             nan     0.1000    0.0020
     7        1.3341             nan     0.1000    0.0018
     8        1.3306             nan     0.1000    0.0015
     9        1.3274             nan     0.1000    0.0014
    10        1.3245             nan     0.1000    0.0013
    20        1.3079             nan     0.1000    0.0004
    40        1.2960             nan     0.1000    0.0000
    60        1.2903             nan     0.1000   -0.0000
    80        1.2857             nan     0.1000   -0.0001
   100        1.2812             nan     0.1000   -0.0001
   120        1.2774             nan     0.1000   -0.0000
   140        1.2739             nan     0.1000   -0.0000
   160        1.2701             nan     0.1000   -0.0000
   180        1.2663             nan     0.1000   -0.0000
   200        1.2629             nan     0.1000   -0.0001
```

```
- Fold5: shrinkage=0.1, interaction.depth=10, n.minobsinnode=10, n.trees=200
Aggregating results
Selecting tuning parameters
Fitting n.trees = 100, interaction.depth = 5, shrinkage = 0.1, n.minobsinnode = 10 on full training set
Iter   TrainDeviance   ValidDeviance   StepSize   Improve
     1        1.3695             nan     0.1000    0.0037
     2        1.3625             nan     0.1000    0.0035
     3        1.3568             nan     0.1000    0.0028
     4        1.3518             nan     0.1000    0.0025
     5        1.3474             nan     0.1000    0.0021
     6        1.3440             nan     0.1000    0.0017
     7        1.3408             nan     0.1000    0.0015
     8        1.3378             nan     0.1000    0.0013
     9        1.3350             nan     0.1000    0.0013
    10        1.3328             nan     0.1000    0.0010
    20        1.3178             nan     0.1000    0.0004
    40        1.3046             nan     0.1000    0.0001
    60        1.2993             nan     0.1000    0.0000
    80        1.2965             nan     0.1000    0.0000
   100        1.2944             nan     0.1000   -0.0000

Warning message:
In train.default(x, y, weights = w, ...) :
  The metric "Accuracy" was not in the result set. ROC will be used instead.
> proc.time() - ptm_rf
   user  system elapsed
 217.28    0.20  217.50
>
> #make predictions aginst testData with the new model
> print(model_gbm)
Stochastic Gradient Boosting

85937 samples
    9 predictor
    2 classes: 'no', 'yes'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 68749, 68751, 68749, 68750, 68749
Resampling results across tuning parameters:

  interaction.depth  n.trees  ROC        Sens       Spec
   1                   25      0.6143768  0.8101849  0.3479370
   1                   50      0.6218183  0.7722348  0.4061879
   1                  100      0.6286359  0.7611705  0.4292260
   1                  150      0.6308180  0.7564134  0.4355067
   1                  200      0.6320101  0.7534404  0.4391103
   2                   25      0.6223853  0.8157916  0.3592370
   2                   50      0.6294636  0.7975916  0.3916188
   2                  100      0.6333549  0.7873979  0.4075264
   2                  150      0.6345888  0.7856353  0.4097659
   2                  200      0.6346425  0.7863786  0.4087106
   5                   25      0.6293912  0.8298290  0.3543464
```

```
   5                  50      0.6344025  0.8118839  0.3804988
   5                 100      0.6361413  0.8078915  0.3871914
   5                 150      0.6359683  0.8056829  0.3900229
   5                 200      0.6355339  0.8072756  0.3872944
  10                  25      0.6327044  0.8366461  0.3499189
  10                  50      0.6351102  0.8202513  0.3708203
  10                 100      0.6351320  0.8144112  0.3771010
  10                 150      0.6339422  0.8109921  0.3791602
  10                 200      0.6331116  0.8096541  0.3803187

Tuning parameter 'shrinkage' was held constant at a value of 0.1

Tuning parameter 'n.minobsinnode' was held constant at a value of 10
ROC was used to select the optimal model using the largest value.
The final values used for the model were n.trees = 100, interaction.depth =
 5, shrinkage = 0.1 and n.minobsinnode = 10.
> pred.model_gbm.prob = predict(model_gbm, newdata = shotTest, type="prob")
> pred.model_gbm.raw = predict(model_gbm, newdata = shotTest)
>
>
> roc.model_gbm = pROC::roc(
+   shotTest$FGM,
+   as.vector(ifelse(pred.model_gbm.prob[,"yes"] >0.5, 1,0))
+ )
Setting levels: control = no, case = yes
Setting direction: controls < cases
> auc.model_gbm = pROC::auc(roc.model_gbm)
> print(auc.model_gbm)
Area under the curve: 0.596
>
> #plot ROC curve
> plot.roc(roc.model_gbm, print.auc = TRUE, col = 'red' , print.thres = "best" )
>
> #generate confusion matrix, as well as other metrics such as accuracy, balanced accuracy
> confusionMatrix(data = pred.model_gbm.raw, shotTest$FGM)
Confusion Matrix and Statistics

          Reference
Prediction    no   yes
       no  16214 10182
       yes  3966  6467

               Accuracy : 0.6158
                 95% CI : (0.6109, 0.6208)
    No Information Rate : 0.5479
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.1984

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.8035
            Specificity : 0.3884
```

```
         Pos Pred Value : 0.6143
         Neg Pred Value : 0.6199
             Prevalence : 0.5479
         Detection Rate : 0.4403
   Detection Prevalence : 0.7167
      Balanced Accuracy : 0.5960

       'Positive' Class : no

>
> #summary of model
> summary(model_gbm)
    var   rel.inf
PC6 PC6 39.162726
PC3 PC3 27.261858
PC5 PC5  9.652336
PC1 PC1  7.877662
PC2 PC2  6.407543
PC4 PC4  4.380920
PC8 PC8  2.955187
PC7 PC7  1.287303
X     X  1.014465
>
> # Save the model into a file
> save(model_gbm, file="gbm_pca.rda")
```

EVALUATION