English Premier League Match Predictor

Omotoyosi Taiwo

## Introduction

Football Matches are notoriously hard to predict due to the random and low scoring nature of the games. One of the prevailing nuggets of wisdom however, is that home teams win more on average, and this has been statistically backed up. In recent years, more advanced metrics have been proposed to account for variance in playing styles and team performances. An example is xG (Expected Goals) this is a measure the possible goals teams should have scored based on shot location, however even that does not account for factors like defence engagement, the time the shot was attempted, and the current scoreline all which hypothetically affect players.

These factors and many have made soccer results really difficult to predict. Betting companies develop advanced algortithms to generate true odds for football matches even though they adjust these odds so that they always turn a profit.

The idea behind this project is to use the closing betting data, team form (using rolling averages of n previous games for various performance metrics) and 538 metrics including expected goals, team reputation (SPI), and projected scores to improve prediction.

The data I used for this project was downloaded from football-data.co.uk and fivethirtyeight.com

Note

Because 538 metrics only go back to the beginning of the 16/17 season, the models I generated were finicky and I decided to use only betting data and team form for various performance metrics like shots on target, shots allowed, corners, fouls, yellow cards, and red cards

## Preparing the Data

I use R data.table, because I love how blazing fast data.tables are and it’s one-liner approach to data manipulation, although it is quite unneccesary for the size of data involved in this project, however I plan to include other leagues in the future, and I can see the data getting big (Big data?) very quickly.

#All functions  
#Takes confusion\_matrix object and plots it with important metrics  
plot\_confusion\_matrix = function(cm ){  
 autoplot(cm, type = "heatmap")+  
 scale\_fill\_gradient(low="#D6EAF8",high = "#2E86C1")+  
 ggtitle(paste0("Accuracy = ", format(round(summary(cm)[[".estimate"]][1], 2), nsmall = 2),  
 " Sensitivity = ", format(round(summary(cm)[[".estimate"]][3], 2), nsmall = 2),  
 " Specificity = ", format(round(summary(cm)[[".estimate"]][4], 2), nsmall = 2))) +  
 theme(plot.title = element\_text(hjust = 0.5))  
 }   
  
##Custom function - Find the rolling mean of previous n elements  
shift\_froll = function(x, n){shift(frollmean(x, n= n))}

Read data for the past 20 seasons, this is how far back I could find gambling data for matches

# Read Match Stats and Betting data  
for (year in c(0:19)){  
 thisYear = print(sprintf("%02d", year))  
 nextYear = print(sprintf("%02d", year+1))  
 assign(paste0("EPL",thisYear,nextYear), fread(paste0("season-",thisYear,nextYear,"\_csv.csv")))  
}  
setnames(EPL1920,c("AvgA","AvgH","AvgD"), c("BbAvA","BbAvH","BbAvD"))#Fix closing betting averages column name changes from the 19/20 season dataset  
  
modelData = NULL  
# Seperate Modelling Data that doesn't need to be aggregated  
modelCols = c("HomeTeam","AwayTeam","FTR","BbAvH","BbAvD","BbAvA")  
relevantColumns = c("HomeTeam","AwayTeam","FTHG","FTAG","HTHG","HTAG","HS","AS","HST","AST","HF","AF","HC","AC","HY","AY","HR","AR")  
allSeasons = vector(mode="list", length = 20)  
meltedDataList = list()  
for (year in c(0:19)){  
 thisYear = print(sprintf("%02d", year))  
 nextYear = print(sprintf("%02d", year+1))  
 assign(paste0("EPL",thisYear,nextYear,"Mod"), get(paste0("EPL",thisYear,nextYear))[,..modelCols] )  
 modelData = rbind(modelData, get(paste0("EPL",thisYear,nextYear,"Mod")))  
 assign(paste0("EPL",thisYear,nextYear), get(paste0("EPL",thisYear,nextYear))[,..relevantColumns])  
 allSeasons[[year+1]] = get(paste0("EPL",thisYear,nextYear))  
}  
  
  
#Separate Predictor variables from Aggregators.  
aggregateEplCols = c("HomeTeam","AwayTeam","FTHG","FTAG","HTHG","HTAG","HS","AS","HST","AST","HF","AF","HC","AC","HY","AY","HR","AR")  
seasonMeltMeasureList = list(c("HomeTeam", "AwayTeam"), c("FTHG", "FTAG"), c("HTHG", "HTAG"), c("HS", "AS"), c("HST", "AST"), c("HF", "AF"), c("HC", "AC"),c("HY", "AY"), c("HR", "AR"))  
seasonMeltNames = c("Team","FTG","HTG","Shots","ST","Fouls","Corners","Yellow","Red")  
  
#For loop to aggregate each season data  
for(season in allSeasons){  
 #Melt to combine home and away results - Note separate running averages in future  
 season = melt(season, measure = seasonMeltMeasureList, value.name = seasonMeltNames)  
 meltedDataList[[length(meltedDataList)+1]] <- season  
}  
allMeltedData = rbindlist(meltedDataList)  
  
#New columns for the rolling Averages  
rollingAvgColumns = paste0(c("FTG","HTG","Shots","ST","Fouls","Corners","Yellow","Red"),"rAvg")  
#bMelted[,variable:= as.factor(variable)]  
  
#Rolling Maean and Shift  
allMeltedData[, (rollingAvgColumns):= lapply(.SD, shift\_froll, n = 5), by = c("Team","variable"), .SDcols = c("FTG","HTG","Shots","ST","Fouls","Corners","Yellow","Red")]  
  
#Fill NA values from rolling window and shifting with mean  
allMeltedData[,11:18 := na.aggregate(allMeltedData[,11:18] )]  
  
#Modelling Data  
#--------------  
#Fold data into Home and Away  
awayNames = names(allMeltedData[,11:18])#Get table names except variable  
awayNames = paste0(awayNames,"Away")#Add away to specify stats  
allHome = allMeltedData[variable == 1, 11:18] #Home Stats without variable  
allAway = allMeltedData[variable == 2, 11:18] #Away stats without the variable  
names(allAway) = awayNames  
allBound = cbind(allHome, allAway)  
  
#Column bind data to get home and away wide table  
finalData = cbind(modelData, allBound)

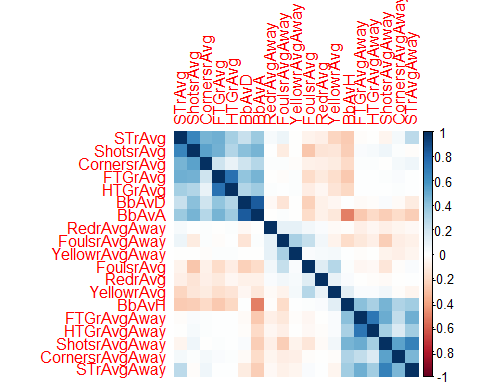
## Generate Modelling Data

After the data has been reshaped to calculate team forms by doing rolling avergaes, some more data cleaning is required.

#Tree Model  
ModelData = finalData[101:nrow(finalData),3:ncol(finalData)] #First 200 rows are pretty much the same  
ModelData = na.omit(ModelData)

Correlation plot below shows intercorrelations between predictors, This is the first step in variable analysis, and from the plots it makes sense that half time goals and full time goals have a strong correlation. BbAvH (Betting odds for the home team) also has moderate correlations with full time goals rolling average and Shots rolling average. This is intuitive because the stronger teams tend to take more shots, which increases the chances of 1 or 2 going in.

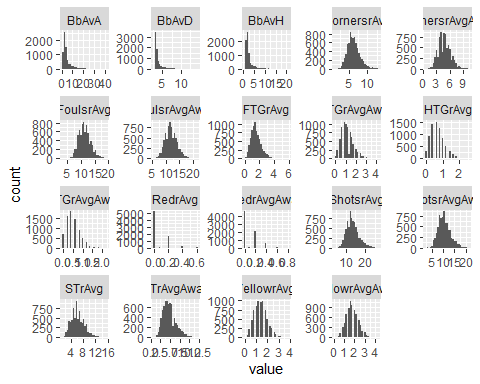
#Correlation Plot  
corrplot(cor(ModelData[,2:ncol(ModelData)]), method ="color",order = "AOE")



The histogram mosaic below shows the distribution of the various variables. Because the variables are all of different units, It is important to standardize (normalize) it, hence the variables with the highest variance dominate the PCA Analysis. I will be excluding the Red cards due to how unfrequent they are. I will log transform the betting avergaes to get the data as close to normal as possible.

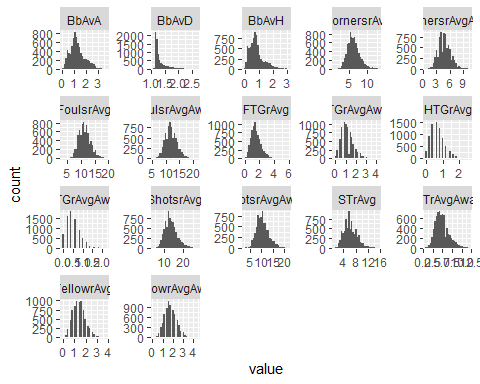
ModelData[,2:ncol(ModelData)]%>%gather()%>%ggplot(aes(value))+facet\_wrap(~key, scales = "free")+geom\_histogram()

## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.



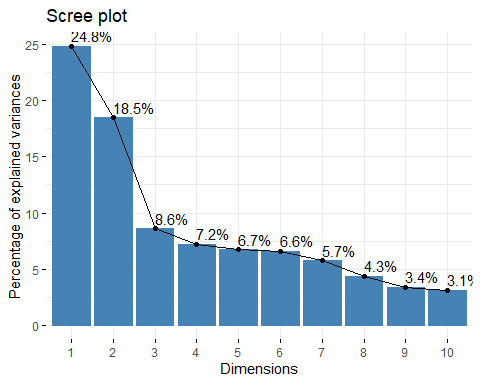
#transform model data  
ModelData[,`:=` (BbAvA=log(BbAvA), BbAvD=log(BbAvD), BbAvH=log(BbAvH), FTR= as.factor(FTR))]  
ModelData[,`:=` (RedrAvg = NULL, RedrAvgAway = NULL)]  
ModelData[,2:ncol(ModelData)]%>%gather()%>%ggplot(aes(value))+facet\_wrap(~key, scales = "free")+geom\_histogram()

## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.

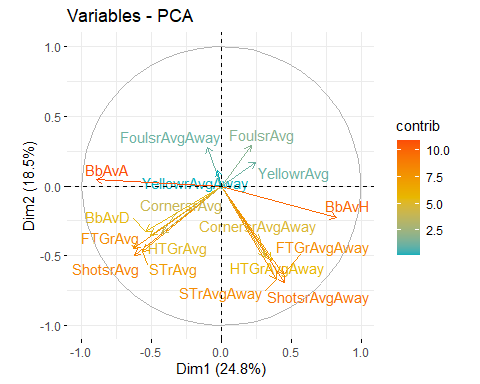


## Principal Component Analysis

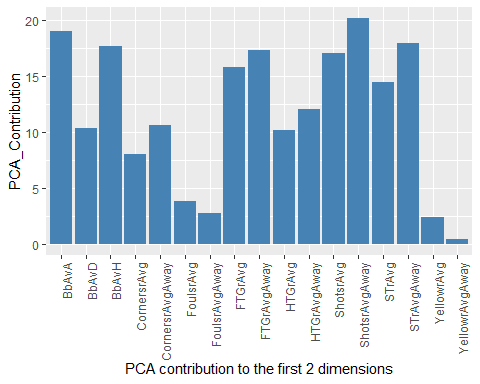
pcaEPL = prcomp(ModelData[,-1], scale=T)  
fviz\_eig(pcaEPL, addlabels = T)



#summary(pcaEPL)  
#Variables  
fviz\_pca\_var(pcaEPL,  
 col.var = "contrib", # Color by contributions to the PC  
 gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),  
 repel = TRUE # Avoid text overlapping  
 )

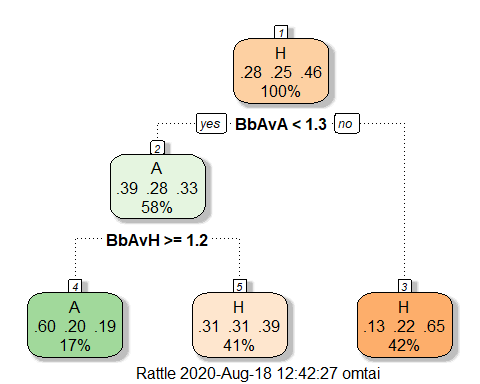


#PCA result  
eplVariablesPCA = get\_pca\_var(pcaEPL)  
#eplVariablesPCA$coord  
#eplVariablesPCA$contrib  
  
variableContribbution = as.data.frame(sort(rowSums(eplVariablesPCA$contrib[,1:2])))  
names(variableContribbution) = c("PCA\_Contribution")  
ggplot(variableContribbution, aes(x = row.names(variableContribbution), y = PCA\_Contribution))+ geom\_bar(stat = "identity", fill="steelblue")+   
 theme(axis.text.x=element\_text(angle=90, hjust = 1))+xlab("PCA contribution to the first 2 dimensions")

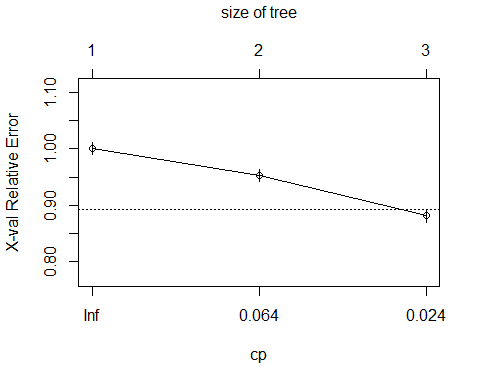


## CART Tree with k-Fold cross validation

#Classification Tree  
EPLTree <- rpart(FTR~ .-FoulsrAvg -FoulsrAvgAway -YellowrAvg -YellowrAvgAway , data= ModelData, method = "class")  
fancyRpartPlot(EPLTree)

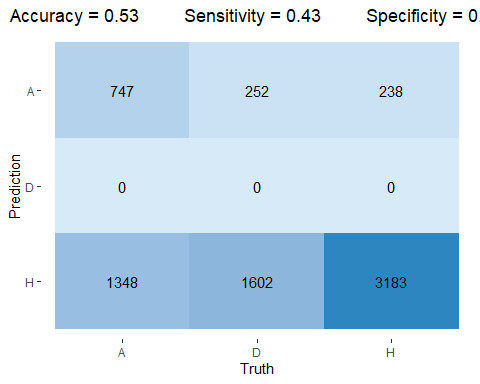


#printcp(EPLTree)  
plotcp(EPLTree)

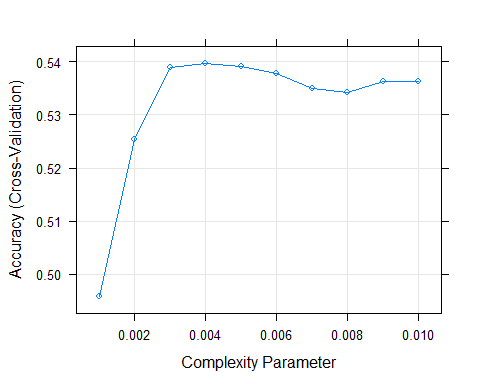


#Predicion - All data  
ModelData$predictFT = predict(EPLTree, type = "class")  
cm <- conf\_mat(ModelData, FTR, predictFT)  
plot\_confusion\_matrix(cm)

## Scale for 'fill' is already present. Adding another scale for 'fill', which  
## will replace the existing scale.



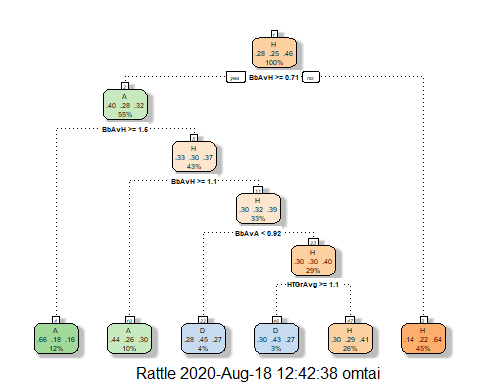
#Collect Train and Test Data  
set.seed(3000)  
trainIndex = createDataPartition(ModelData$FTR, p = 0.7, list = F, times = 1)   
Train = ModelData[trainIndex,]  
Test = ModelData[-trainIndex,]  
  
#Cross Validation  
numFolds <- trainControl(method="cv", number=10, repeats = 10)  
cpGrid <- expand.grid(.cp=seq(0.001,0.01,0.001))#cp paramaeters to test as numbers from 0.0005 to 0.05, in increments of 0.01.  
treeTuning = train(FTR~.-FoulsrAvg -FoulsrAvgAway -YellowrAvg -YellowrAvgAway, data = Train, method="rpart", trControl=numFolds, tuneGrid = cpGrid) #cp = 0.0155  
plot(treeTuning)



treeTuning

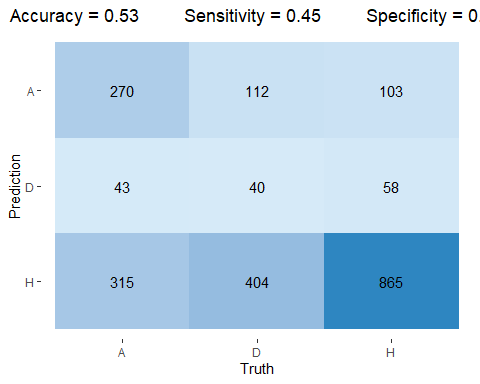
## CART   
##   
## 5160 samples  
## 18 predictor  
## 3 classes: 'A', 'D', 'H'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 4643, 4645, 4644, 4644, 4645, 4644, ...   
## Resampling results across tuning parameters:  
##   
## cp Accuracy Kappa   
## 0.001 0.4957548 0.1831466  
## 0.002 0.5254008 0.2082387  
## 0.003 0.5389611 0.2224991  
## 0.004 0.5397359 0.2221366  
## 0.005 0.5391538 0.2207803  
## 0.006 0.5377968 0.2180318  
## 0.007 0.5348872 0.2123516  
## 0.008 0.5341116 0.2143880  
## 0.009 0.5362434 0.2173852  
## 0.010 0.5362442 0.2164838  
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was cp = 0.004.

#Classification Tree - With Validation Set  
EPLTree = rpart(FTR~ .-FoulsrAvg -FoulsrAvgAway -YellowrAvg -YellowrAvgAway, data= Train, method = "class", cp = 0.004)  
fancyRpartPlot(EPLTree)



#Predicion - Test data  
Test$predictFT = predict(EPLTree, type = "class", newdata = Test)  
cm = conf\_mat(Test, FTR, predictFT)  
plot\_confusion\_matrix(cm)

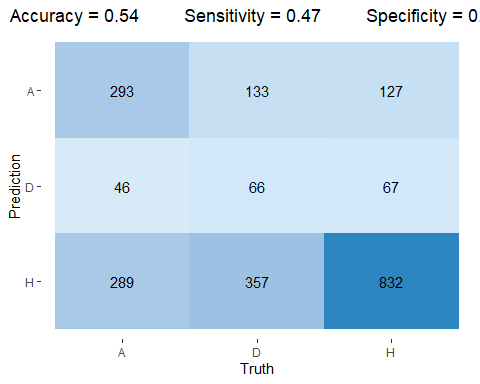
## Scale for 'fill' is already present. Adding another scale for 'fill', which  
## will replace the existing scale.



## Random Forest

EPLForest = randomForest(FTR~., data= Train, mtry =4, ntree = 1000)  
Test$predictFT = predict(EPLForest, type = "class", newdata = Test)  
cm <- conf\_mat(Test, FTR, predictFT)  
plot\_confusion\_matrix(cm)

## Scale for 'fill' is already present. Adding another scale for 'fill', which  
## will replace the existing scale.



# Tune Parameters  
# control = trainControl(method="repeatedcv", number=10, repeats=3, search="grid")  
# tunegrid = expand.grid(.mtry =c(1:5))  
# rfGrid <- train(FTR~., data=Train, method="rf", metric="Accuracy", tuneGrid=tunegrid, trControl = control, ntree = 500)  
# print(rfGrid)  
# plot(rfGrid)

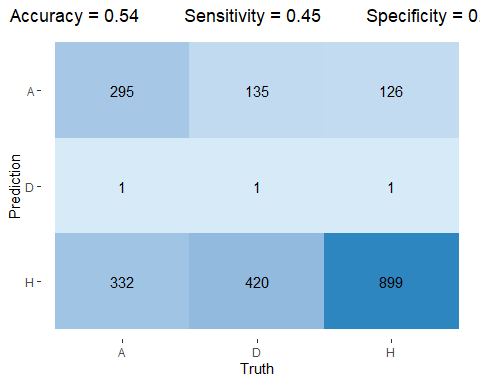
## Multinomial Logistic Regression

glmModel = multinom(FTR~.-FoulsrAvg -FoulsrAvgAway -YellowrAvg -YellowrAvgAway, data= Train)

## # weights: 51 (32 variable)  
## initial value 5668.839410   
## iter 10 value 5169.756715  
## iter 20 value 5026.583400  
## iter 30 value 4952.566469  
## final value 4949.762794   
## converged

#summary(glmModel)  
Test$predictFT = predict(glmModel,Test, "class")  
cm <- conf\_mat(Test, FTR, predictFT)  
plot\_confusion\_matrix(cm)

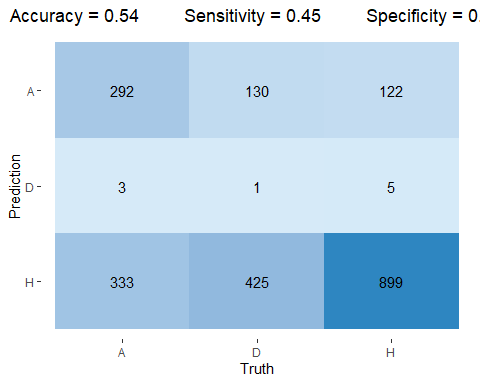
## Scale for 'fill' is already present. Adding another scale for 'fill', which  
## will replace the existing scale.



## Linear Discriminant Analysis and Quadratic Discriminat Analysis

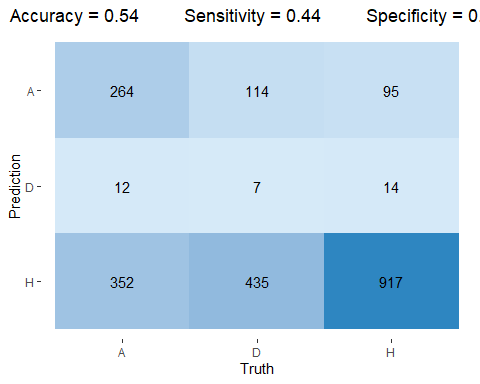
#Log transform BbAvH for lda to normalize  
ldaTrain = copy(Train)  
#ldaTrain[,BbAvH:=log(BbAvH)]  
#ldaTrain[,BbAvD:=log(BbAvD)]  
  
ldaTest = copy(Test)  
#ldaTest[,BbAvH:=log(BbAvH)]  
#ldaTest[,BbAvD:=log(BbAvD)]  
#ldaData = rbind(ldaTest,ldaTrain)  
  
ldaModel = lda(FTR~BbAvH+BbAvA+BbAvD+FTGrAvg+STrAvg+CornersrAvg+FTGrAvgAway+STrAvgAway+CornersrAvgAway, data = ldaTrain)  
library(nnet)  
ldaFT = predict(ldaModel, ldaTest)  
ldaTest$predictFT = ldaFT$class  
cm <- conf\_mat(ldaTest, FTR, predictFT)  
plot\_confusion\_matrix(cm)

## Scale for 'fill' is already present. Adding another scale for 'fill', which  
## will replace the existing scale.



qdaModel = qda(FTR~BbAvH+FTGrAvg+STrAvg+CornersrAvg+FTGrAvgAway+STrAvgAway+CornersrAvgAway, data = ldaTrain)  
qdaFT = predict(qdaModel, ldaTest)  
ldaTest$predictFT = qdaFT$class  
cm <- conf\_mat(ldaTest, FTR, predictFT)  
plot\_confusion\_matrix(cm)

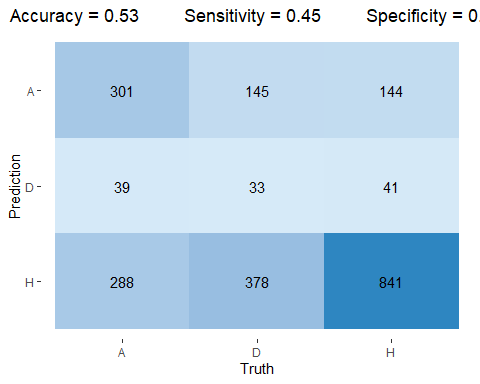
## Scale for 'fill' is already present. Adding another scale for 'fill', which  
## will replace the existing scale.



## Naive Bayes Classifier

naiveModel = naiveBayes(FTR~BbAvH+FTGrAvg+STrAvg+CornersrAvg+FTGrAvgAway+STrAvgAway+CornersrAvgAway, data = ldaTrain)  
ldaTest$predictFT = predict(naiveModel, type = "class", newdata = ldaTest)  
cm <- conf\_mat(ldaTest, FTR, predictFT)  
plot\_confusion\_matrix(cm)

## Scale for 'fill' is already present. Adding another scale for 'fill', which  
## will replace the existing scale.



## Conclusion

After trying various models for prediction, I couldn’t do better than a 54% out of sample prediction accuracy. The betting odds dominate the models as expected, due to the fact that the complex algorithms used by the multi-billion dollar gambling industry already accounts for team performance metrics. In the future, I plan to incorporate other variables into my models like, team standings from previous seasons, transfer activity in net spending, number of team injuries at time of match, etc.

The easiest part of this was programming the models, The most challenging aspect of the project was the data preparation, and the other metrics I plan to include in the future will be even more challenging due to the data coming from several different sources.