IN-PLAY BETTING IN FOOTBALL

Henrik Sergoyan

December 5, 2017

In many sports games, spectators are interested in predicting the outcomes and placing their bets based on those predictions. Aside from traditional approaches, there are also statistical methods that help to make a good prediction. In this project, we use several statistical methods such as Logistic Regression, Decision trees, Naive Bayes classifier, KNN, Random Forest to build models that predict the outcome of a football match based on the statistics of the first half. Those models can be useful for in-play betting.

First, let us get acquainted with the data that we are going to use.

```r
library(readxl)
imb <- read_excel("Copy of IMB403-XLS-ENG.xlsx")
colnames(imb)[4]<-"RED_H"
colnames(imb)[5]<-"RED_A"
str(imb)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    1520 obs. of  11 variables:
##  $ Match Number: num  1 2 3 4 5 6 7 8 9 10 ...
##  $ HTGD        : num  0 2 3 2 0 0 1 1 1 0 ...
##  $ FGS         : num  2 1 1 1 2 0 1 1 1 0 ...
##  $ RED_H       : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ RED_A       : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ POINTS_H    : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ POINTS_A    : num  3 1 0 0 0 0 0 0 0 0 ...
##  $ TOTAL_H_P   : num  0 65 83 91 0 67 50 58 38 0 ...
##  $ TOTAL_A_P   : num  55 0 48 43 82 42 0 51 63 45 ...
##  $ Match_O     : num  1 2 2 2 1 1 2 2 2 2 ...
##  $ Match_O1    : num  0 1 1 1 0 0 1 1 1 1 ...
```

The dataset imb contains the statistics of the first half of past English Premier League (EPL) games as well as the outcomes of those games. The descriptions of variables are the following:

HTGD - Half-time goal difference (Number of goals scored by home team - Number of goals scored by away team at half-time)

FGS - First Goal Scored, FGS = 1 means home team scored the 1st goal, 0 denotes that away team scored the 1st goal, 2 denotes none of them scored goal

RED_H - Red cards conceded by the home team

RED_A - Red cards conceded by the away team

POINTS_H - Points earned by the home team in the league until that match

POINTS_A - Points earned by the away team in the league until the match

TOTAL_H_P - Total points earned by the home team in the previous season

TOTAL_A_P - Total points earned by the away team in the previous season

Match_O - Match outcome, 2 - home team win, 1 - draw, 0 - away team win

Match_O1 - Match outcome, 1- home team win, 0 - home team not win

Match_O and Match_O1 variables are numeric so let's convert these variables to a more appropriate class.

```r
imb <- imb[,-1]
imb$Match_O <- factor(imb$Match_O, levels = c(0,1,2), labels = c("Lose","Draw","Win"))
imb$Match_O1 <- factor(imb$Match_O1, levels = c(0,1), labels = c("Not win","Win"))
```

Now, let us call all the libraries that we will need when building our models.

```r
library(caret)
library(e1071)
library(rpart)
library(rpart.plot)
library(class)
library(ROCR)
library(pROC)
library(randomForest)
library(nnet)
library(reshape2)
library(ggplot2)
```

Two-class Classification

First, let's use Match_O1 variable for making predictions. That is, we want to predict whether the home team will win or not win the match.

Let's start by removing the variable Match_O and dividing the data into training and testing sets.

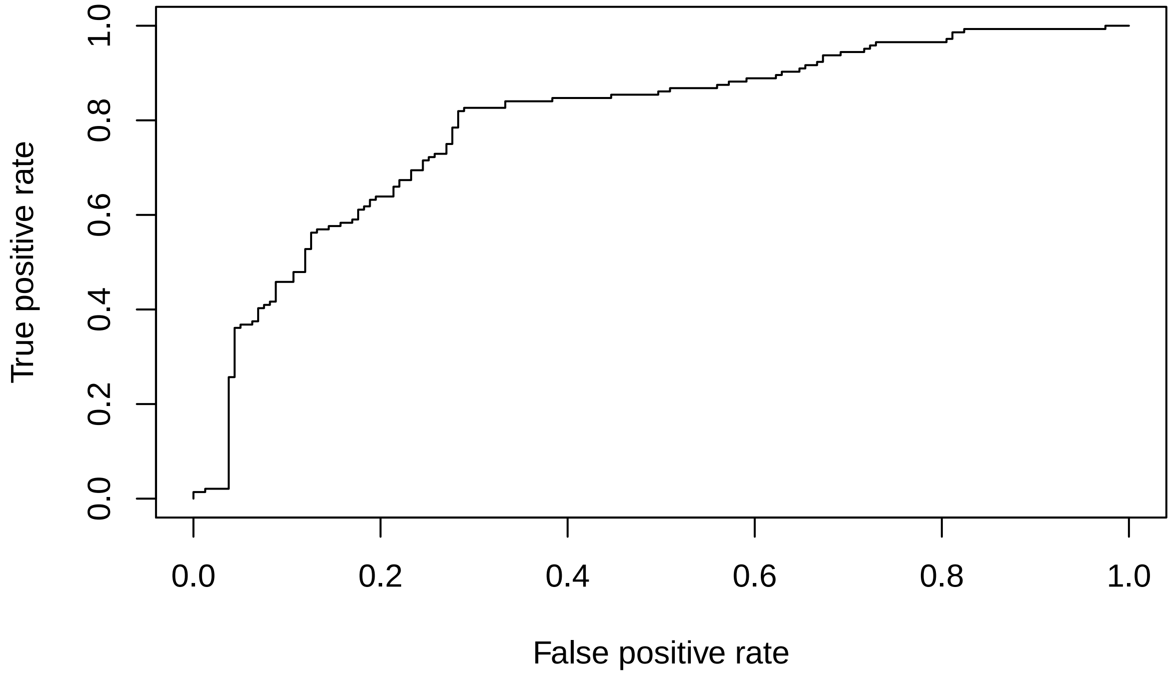
```r
set.seed(1)
trainIndex<-createDataPartition(imb$Match_O1, p=.8, list=FALSE)
Train<-imb[trainIndex,-9]
Test<-imb[-trainIndex,-9]
```

1. Naive Bayes Classifier

```r
model_nv1<-naiveBayes(Match_O1~., data=Train, laplace=1)
pred<-predict(model_nv1, newdata=Test)
confusionMatrix(data=pred, reference=Test$Match_O1, positive="Win")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Not win Win
##    Not win     114  30
##    Win          45 114
##
##                Accuracy : 0.7525
##                  95% CI : (0.6999, 0.8)
##     No Information Rate : 0.5248
##     P-Value [Acc > NIR] : 2.98e-16
##
##                   Kappa : 0.5062
##  Mcnemar's Test P-Value : 0.106
##
##             Sensitivity : 0.7917
##             Specificity : 0.7170
##          Pos Pred Value : 0.7170
##          Neg Pred Value : 0.7917
##              Prevalence : 0.4752
##          Detection Rate : 0.3762
##    Detection Prevalence : 0.5248
##       Balanced Accuracy : 0.7543
##
##        'Positive' Class : Win
##
```

```
pr1<-predict(model_nv1, newdata = Test, type="raw")
P_test<-prediction(pr1[,2], Test$Match_01)
perf<-performance(P_test,"tpr","fpr")
plot(perf)
```



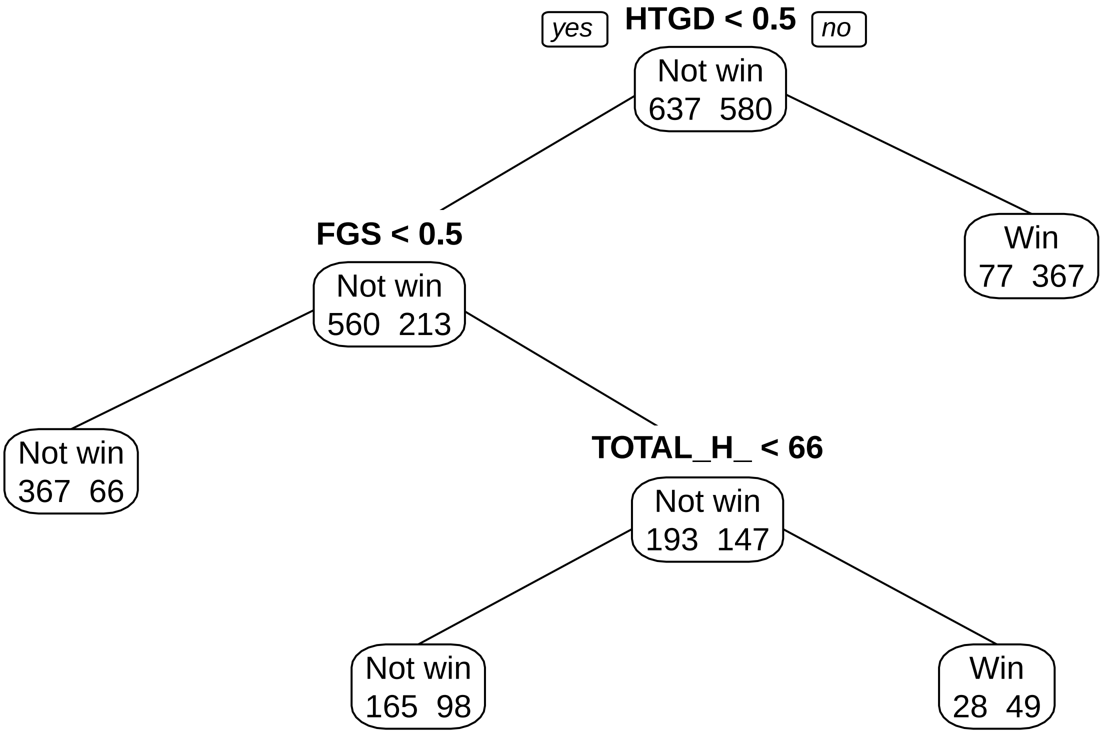
```
performance(P_test,"auc")@y.values
```

```
## [[1]]
## [1] 0.7950734
```

The overall accuracy of this model is 0.7525. Sensitivity and Specificity are 0.7917 and 0.7170 respectively. AUC, that is, the area under the curve is approximately 0.795, which is close to 1 so our model is good.

2. Decision Tree

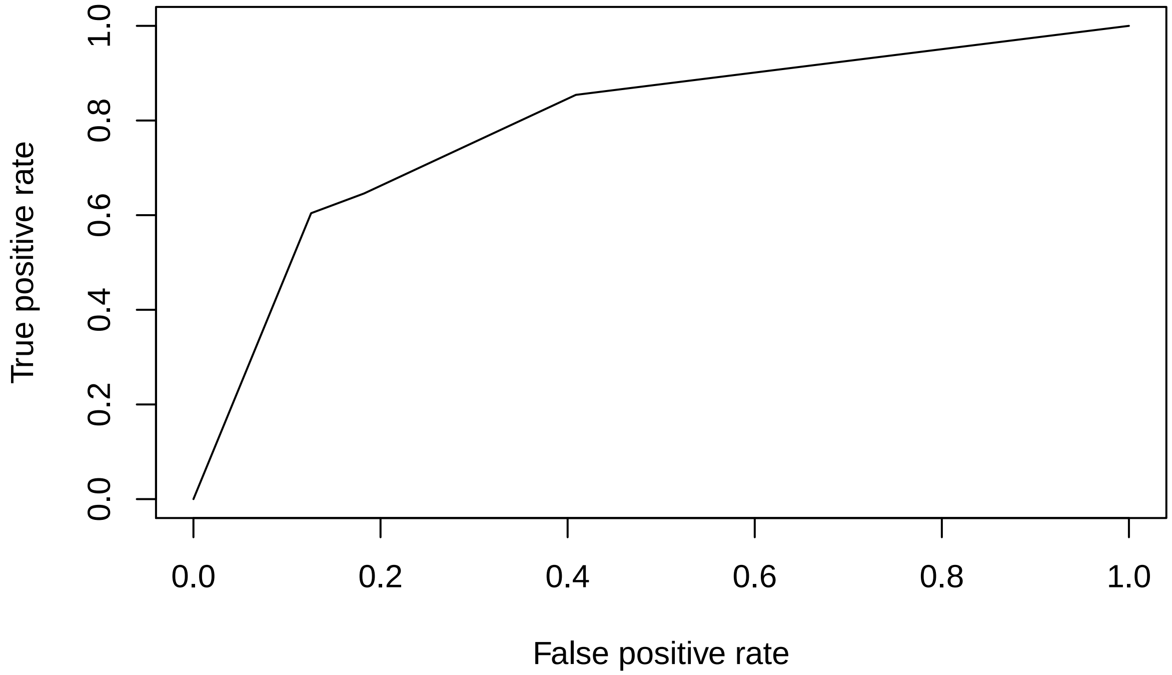
```
model_dt1<-rpart(Match_01~., data=Train)
prp(model_dt1, type=1, extra=1)
```



```
pred_class<-predict(model_dt1, Test, type="class")
confusionMatrix(pred_class, Test$Match_01)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Not win Win
##    Not win     130  51
##    Win          29  93
##
##                Accuracy : 0.736
##                  95% CI : (0.6825, 0.7847)
##     No Information Rate : 0.5248
##     P-Value [Acc > NIR] : 3.905e-14
##
##                   Kappa : 0.4668
##  Mcnemar's Test P-Value : 0.01888
##
##             Sensitivity : 0.8176
##             Specificity : 0.6458
##          Pos Pred Value : 0.7182
##          Neg Pred Value : 0.7623
##              Prevalence : 0.5248
##          Detection Rate : 0.4290
##    Detection Prevalence : 0.5974
##       Balanced Accuracy : 0.7317
##
##        'Positive' Class : Not win
##
```

```
pr1<-predict(model_dt1, Test)
P_test<-prediction(pr1[,2], Test$Match_01)
perf<-performance(P_test,"tpr","fpr")
plot(perf)
```



```
performance(P_test,"auc")@y.values
```

```
## [[1]]
## [1] 0.7912736
```

The overall accuracy of this model is 0.736. Sensitivity and Specificity are 0.8176 and 0.6458 respectively. AUC is approximately 0.791, which is again close to 1.

3. KNN

```
scaled_train<-as.data.frame(scale(Train[,-9]))
scaled_test<-as.data.frame(scale(Test[,-9]))
knn_1<-knn(scaled_train, scaled_test, Train$Match_01, k=21)
confusionMatrix(knn_1, Test$Match_01)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Not win Win
##    Not win     117  44
##    Win          42 100
##
##                Accuracy : 0.7162
##                  95% CI : (0.6618, 0.7663)
##     No Information Rate : 0.5248
##     P-Value [Acc > NIR] : 7.832e-12
##
##                   Kappa : 0.4306
##  Mcnemar's Test P-Value : 0.9141
```

```
##
##             Sensitivity : 0.7358
##             Specificity : 0.6944
##          Pos Pred Value : 0.7267
##          Neg Pred Value : 0.7042
##              Prevalence : 0.5248
##          Detection Rate : 0.3861
##    Detection Prevalence : 0.5314
##       Balanced Accuracy : 0.7151
##
##        'Positive' Class : Not win
##
```

```
knn_p<-knn(scaled_train, scaled_test, Train$Match_01, k=21, prob=T)
a<-multiclass.roc(Test$Match_01,as.ordered(knn_p))
a$auc
```

```
## Multi-class area under the curve: 0.7151
```

The overall accuracy of this model is 0.7162. Sensitivity and Specificity are 0.7358 and 0.6944 respectively. AUC is approximately 0.715.

4. Random Forest

```
set.seed(1)
model_rf<-randomForest(Match_01~., data = Train, ntree = 50, do.trace = T, importance = T)
```

```
## ntree      OOB      1      2
##     1:  29.55% 28.44% 30.70%
##     2:  30.82% 30.85% 30.79%
##     3:  30.70% 29.21% 32.33%
##     4:  29.47% 28.74% 30.27%
##     5:  29.16% 26.10% 32.50%
##     6:  29.54% 27.87% 31.37%
##     7:  29.15% 29.46% 28.80%
##     8:  28.66% 28.37% 28.98%
##     9:  28.17% 29.05% 27.19%
##    10:  28.86% 27.73% 30.09%
##    11:  28.39% 27.06% 29.86%
##    12:  27.99% 26.34% 29.81%
##    13:  28.42% 26.93% 30.05%
##    14:  27.77% 26.53% 29.14%
##    15:  28.10% 26.37% 30.00%
##    16:  25.80% 22.92% 28.97%
##    17:  26.79% 24.02% 29.83%
##    18:  26.87% 24.02% 30.00%
##    19:  27.12% 23.55% 31.03%
##    20:  27.12% 24.96% 29.48%
##    21:  26.13% 23.23% 29.31%
##    22:  26.62% 24.80% 28.62%
##    23:  28.02% 24.96% 31.38%
##    24:  26.21% 23.55% 29.14%
##    25:  26.38% 23.39% 29.66%
##    26:  25.88% 22.92% 29.14%
##    27:  26.13% 23.08% 29.48%
##    28:  25.72% 22.14% 29.66%
##    29:  25.64% 22.14% 29.48%
##    30:  25.88% 22.61% 29.48%
##    31:  25.31% 21.51% 29.48%
##    32:  25.06% 21.51% 28.97%
##    33:  25.47% 21.82% 29.48%
##    34:  25.39% 21.51% 29.66%
##    35:  25.55% 21.35% 30.17%
##    36:  25.23% 20.88% 30.00%
##    37:  25.80% 21.51% 30.52%
##    38:  25.31% 21.19% 29.83%
##    39:  25.47% 21.51% 29.83%
##    40:  25.39% 21.51% 29.66%
##    41:  25.88% 21.98% 30.17%
##    42:  25.80% 21.66% 30.34%
##    43:  25.31% 20.72% 30.34%
##    44:  25.23% 21.04% 29.83%
##    45:  25.06% 21.04% 29.48%
##    46:  24.98% 20.72% 29.66%
##    47:  25.31% 21.35% 29.66%
##    48:  24.90% 20.57% 29.66%
##    49:  25.06% 20.88% 29.66%
##    50:  25.23% 21.04% 29.83%
```

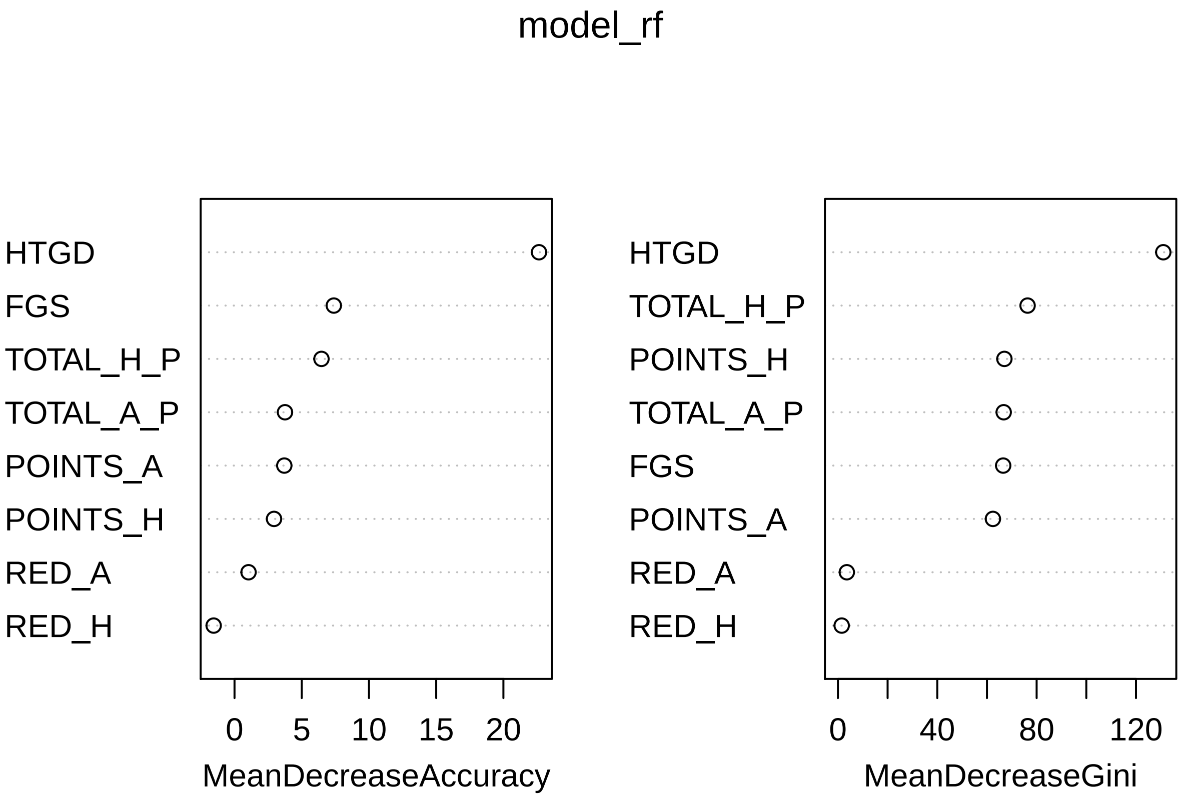
```
model_rf$importance
```

```
##                  Not win           Win MeanDecreaseAccuracy
## HTGD       0.1347968962  0.1199558919         0.1274683265
## FGS        0.0245354889  0.0293139099         0.0266574932
## RED_H     -0.0004511467 -0.0003838796        -0.0004015439
## RED_A      0.0001787597  0.0005418918         0.0003721536
## POINTS_H   0.0076371841  0.0118929990         0.0096404258
## POINTS_A   0.0089519583  0.0095579591         0.0092202779
## TOTAL_H_P  0.0268579432  0.0118218741         0.0195729630
## TOTAL_A_P  0.0104827750  0.0089658762         0.0100201623
##           MeanDecreaseGini
## HTGD            130.983906
## FGS              66.524693
## RED_H             1.522967
## RED_A             3.582612
## POINTS_H         67.018060
## POINTS_A         62.395427
## TOTAL_H_P        76.354555
## TOTAL_A_P        66.734106
```

```
model_rf$err.rate
```

```
##              OOB   Not win       Win
##  [1,] 0.2954545 0.2844444 0.3069767
##  [2,] 0.3082192 0.3085106 0.3079096
##  [3,] 0.3070078 0.2921109 0.3232558
##  [4,] 0.2947053 0.2873563 0.3027140
##  [5,] 0.2916283 0.2610229 0.3250000
##  [6,] 0.2954145 0.2787162 0.3136531
##  [7,] 0.2914530 0.2945990 0.2880143
##  [8,] 0.2865546 0.2836538 0.2897527
##  [9,] 0.2816667 0.2904762 0.2719298
## [10,] 0.2885572 0.2773376 0.3008696
## [11,] 0.2839404 0.2705696 0.2986111
## [12,] 0.2799339 0.2634069 0.2980936
## [13,] 0.2841845 0.2692913 0.3005181
## [14,] 0.2777321 0.2653061 0.2913793
## [15,] 0.2810189 0.2637363 0.3000000
## [16,] 0.2580115 0.2291994 0.2896552
## [17,] 0.2678718 0.2401884 0.2982759
## [18,] 0.2686935 0.2401884 0.3000000
## [19,] 0.2711586 0.2354788 0.3103448
## [20,] 0.2711586 0.2496075 0.2948276
## [21,] 0.2612983 0.2323391 0.2931034
## [22,] 0.2662284 0.2480377 0.2862069
## [23,] 0.2801972 0.2496075 0.3137931
## [24,] 0.2621200 0.2354788 0.2913793
## [25,] 0.2637634 0.2339089 0.2965517
## [26,] 0.2588332 0.2291994 0.2913793
## [27,] 0.2612983 0.2307692 0.2948276
## [28,] 0.2571898 0.2213501 0.2965517
## [29,] 0.2563681 0.2213501 0.2948276
## [30,] 0.2588332 0.2260597 0.2948276
## [31,] 0.2530813 0.2150706 0.2948276
## [32,] 0.2506163 0.2150706 0.2896552
## [33,] 0.2547247 0.2182104 0.2948276
## [34,] 0.2539030 0.2150706 0.2965517
## [35,] 0.2555464 0.2135008 0.3017241
## [36,] 0.2522597 0.2087912 0.3000000
## [37,] 0.2580115 0.2150706 0.3051724
## [38,] 0.2530813 0.2119309 0.2982759
## [39,] 0.2547247 0.2150706 0.2982759
## [40,] 0.2539030 0.2150706 0.2965517
## [41,] 0.2588332 0.2197802 0.3017241
## [42,] 0.2580115 0.2166405 0.3034483
## [43,] 0.2530813 0.2072214 0.3034483
## [44,] 0.2522597 0.2103611 0.2982759
## [45,] 0.2506163 0.2103611 0.2948276
## [46,] 0.2497946 0.2072214 0.2965517
## [47,] 0.2530813 0.2135008 0.2965517
## [48,] 0.2489729 0.2056515 0.2965517
## [49,] 0.2506163 0.2087912 0.2965517
## [50,] 0.2522597 0.2103611 0.2982759
```

```
varImpPlot(model_rf)
```



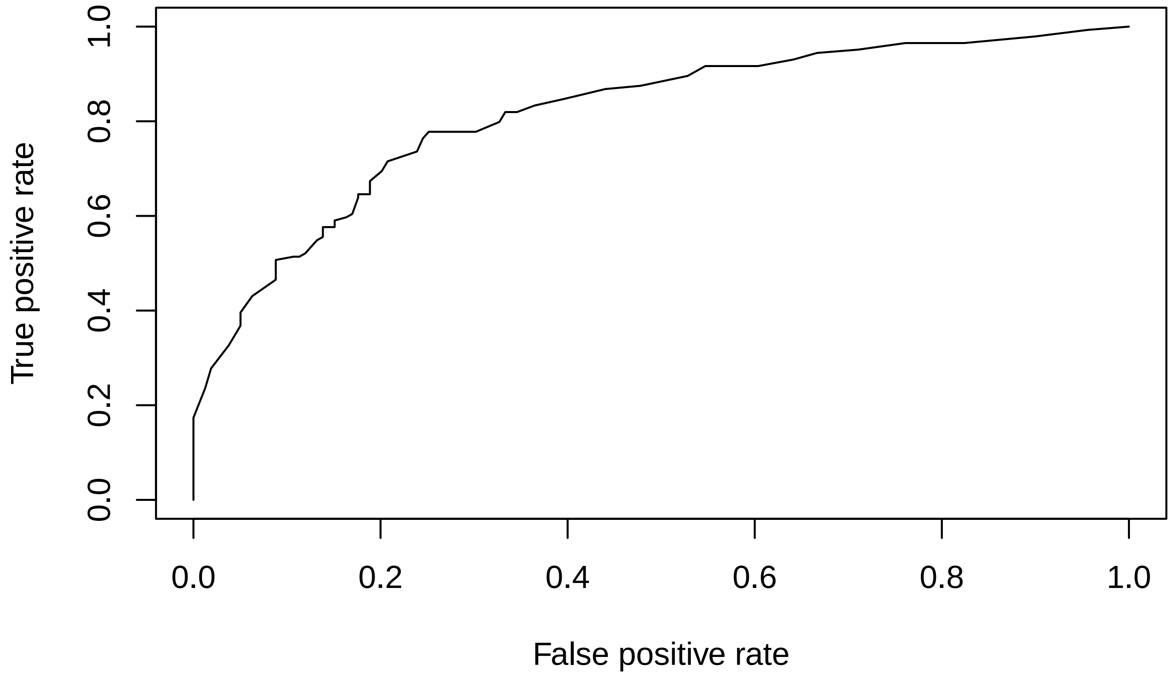
```
#The most important one is the HTGD as the average decrease in accuracy when it does not participate in

pred_class_rf<-predict(model_rf, newdata=Test, type="class")
confusionMatrix(pred_class_rf, Test$Match_01, positive = "Win")
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction Not win Win
##    Not win     129  47
##    Win          30  97
##
##                Accuracy : 0.7459
##                  95% CI : (0.6929, 0.7939)
##     No Information Rate : 0.5248
##     P-Value [Acc > NIR] : 2.205e-15
##
##                   Kappa : 0.4876
##  Mcnemar's Test P-Value : 0.06825
##
##             Sensitivity : 0.6736
##             Specificity : 0.8113
##          Pos Pred Value : 0.7638
##          Neg Pred Value : 0.7330
##              Prevalence : 0.4752
##          Detection Rate : 0.3201
##    Detection Prevalence : 0.4191
##       Balanced Accuracy : 0.7425
##
```

```
##        'Positive' Class : Win
##
```

```
pred_prob_rf<-predict(model_rf, newdata = Test, type = "prob")
p_test_rf<-prediction(pred_prob_rf[,2], Test$Match_01)
perf_rf<-performance(p_test_rf, "tpr", "fpr")
plot(perf_rf)
```



```
performance(p_test_rf, "auc")@y.values
```

```
## [[1]]
## [1] 0.8157102
```

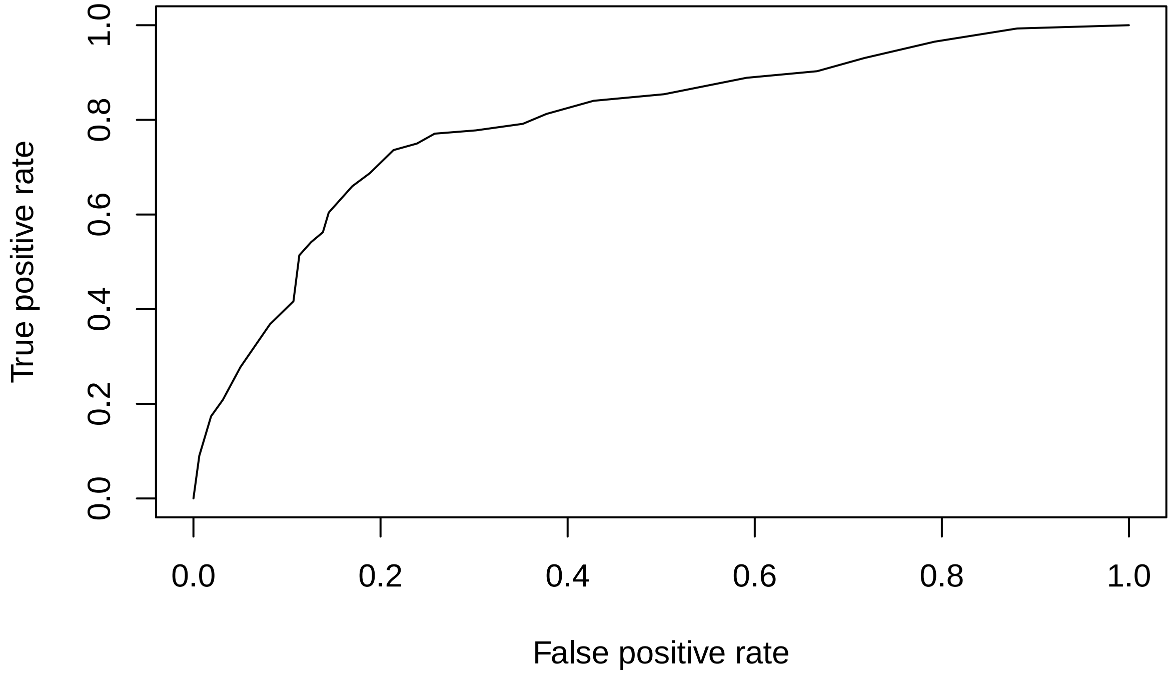
The overall accuracy of this model is 0.7459. Sensitivity and Specificity are 0.6736 and 0.8113 respectively. AUC is approximately 0.815, which is again close to 1.

Now, let's run the Random Forest with Specified MTry value to see how it changes the model.

```
set.seed(1)
trc<-trainControl(method = "cv", number = 10)
mtry_grid<-expand.grid(mtry=c(5:25))
model_mtry<-train(Match_01~., data = Train, trControl=trc, method = "rf", ntree=25, tuneGrid=mtry_grid)
#model_mtry$results
set.seed(1)
pred_class_mtry<-predict(model_mtry, newdata=Test, type="raw")
confusionMatrix(pred_class_mtry, Test$Match_01, positive = "Win")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Not win Win
##    Not win     125  38
##    Win          34 106
##
##                Accuracy : 0.7624
##                  95% CI : (0.7104, 0.8092)
##     No Information Rate : 0.5248
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.523
##  Mcnemar's Test P-Value : 0.7237
##
##             Sensitivity : 0.7361
##             Specificity : 0.7862
##          Pos Pred Value : 0.7571
##          Neg Pred Value : 0.7669
##              Prevalence : 0.4752
##          Detection Rate : 0.3498
##    Detection Prevalence : 0.4620
##       Balanced Accuracy : 0.7611
##
##        'Positive' Class : Win
##
```

```
pred_prob_mtry<-predict(model_mtry, newdata=Test, type="prob")
p_test_mtry<-prediction(pred_prob_mtry[,2], Test$Match_01)
perf_mtry<-performance(p_test_mtry, "tpr", "fpr")
plot(perf_mtry)
```



```
performance(p_test_mtry, "auc")@y.values
```

```
## [[1]]
## [1] 0.7952699
```

The overall accuracy of this model is 0.7624. Sensitivity and Specificity are 0.7361 and 0.7862 respectively. AUC is approximately 0.795.

As we can see, the overall accuracy, as well as the Sensitivity, slightly increased. However, there was a slight decrease in Specificity and AUC.

5. Binomial Logistic Regression

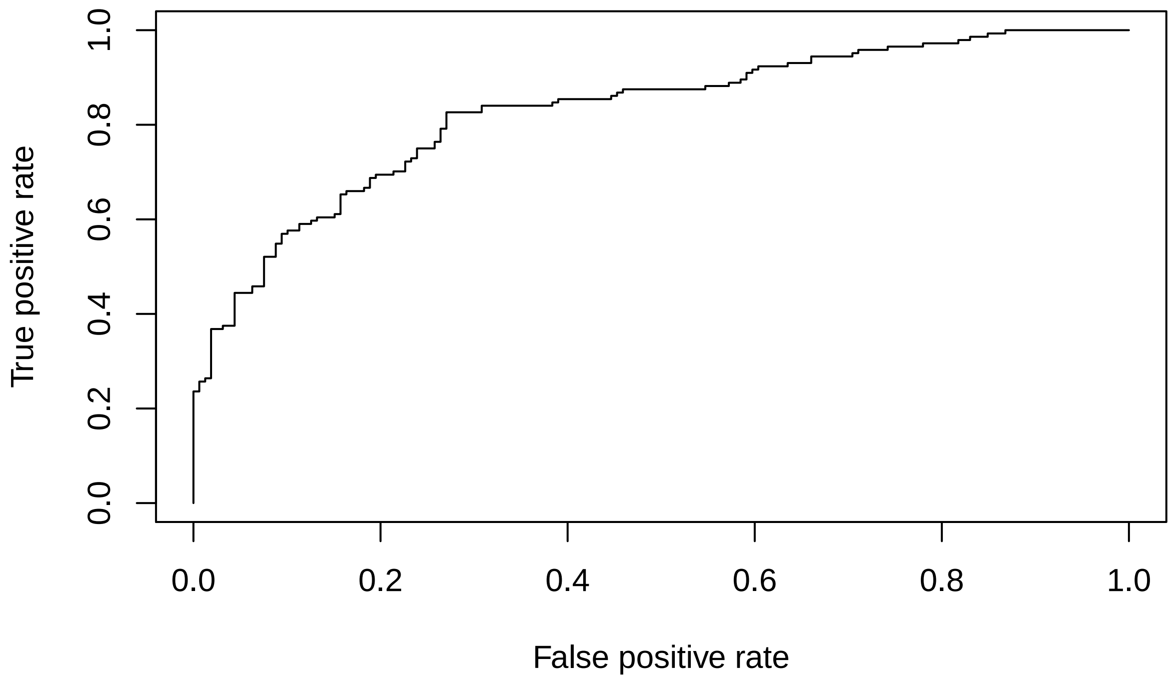
```
model_blg<-glm(formula=Match_01~., data=Train, family = "binomial")
pr1<-predict(model_blg, newdata = Test, type="response")
table(Test$Match_01, pr1>0.5)
```

```
##
##           FALSE TRUE
##   Not win   126   33
##   Win        44  100
```

```
pr_class<-ifelse(pr1>0.5, "Win", "Not win")
caret::confusionMatrix(pr_class, Test$Match_01, positive="Win")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Not win Win
##    Not win     126  44
##    Win          33 100
##
##                Accuracy : 0.7459
##                  95% CI : (0.6929, 0.7939)
##     No Information Rate : 0.5248
##     P-Value [Acc > NIR] : 2.205e-15
##
##                   Kappa : 0.4887
##  Mcnemar's Test P-Value : 0.2545
##
##             Sensitivity : 0.6944
##             Specificity : 0.7925
##          Pos Pred Value : 0.7519
##          Neg Pred Value : 0.7412
##              Prevalence : 0.4752
##          Detection Rate : 0.3300
##    Detection Prevalence : 0.4389
##       Balanced Accuracy : 0.7434
##
##        'Positive' Class : Win
##
```

```
P_Test<-prediction(pr1, Test$Match_01)
perf<- performance(P_Test, "tpr", "fpr")
plot(perf)
```



```
performance(P_test,"auc")@y.values
```

```
## [[1]]
## [1] 0.7912736
```

The overall accuracy of this model is 0.7459. Sensitivity and Specificity are 0.6944 and 0.7925 respectively. AUC is approximately 0.791, which is again close to 1.

To sum up, Random Forest and Naive Bayes Classifier gave the highest overall accuracy and AUC, while Decision tree and Logistic Regression gave the highest Sensitivity and Specificity respectively. However, the differences between the results of all five methods were not that significant.

Multi-class Classification

For multi-class classification, we will use Match_O variable for making predictions. That is, we will try to predict whether the home team will win, draw or lose the match.

Again, let's start by dividing the data into training and testing sets - but this time removing the variable Match_O1 from the data.

```
set.seed(1)
trainIndex2<-createDataPartition(imb$Match_O, p=.8, list=FALSE)
Train2<-imb[trainIndex2,-10]
Test2<-imb[-trainIndex2,-10]
```

1. Naive Bayes Classifier

```
model_nv<-naiveBayes(Match_O~., data=Train2, laplace=1)
pred<-predict(model_nv, newdata=Test2)
confusionMatrix(data=pred, reference=Test2$Match_O, positive="Win")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Lose Draw Win
##       Lose   61   23  13
##       Draw    2   24  13
##       Win    15   33 118
##
## Overall Statistics
##
##                Accuracy : 0.6722
##                  95% CI : (0.6161, 0.7249)
##     No Information Rate : 0.4768
##     P-Value [Acc > NIR] : 5.804e-12
##
##                   Kappa : 0.4719
##  Mcnemar's Test P-Value : 7.572e-06
##
## Statistics by Class:
##
##                      Class: Lose Class: Draw Class: Win
## Sensitivity               0.7821     0.30000     0.8194
## Specificity               0.8393     0.93243     0.6962
## Pos Pred Value            0.6289     0.61538     0.7108
## Neg Pred Value            0.9171     0.78707     0.8088
## Prevalence                0.2583     0.26490     0.4768
## Detection Rate            0.2020     0.07947     0.3907
## Detection Prevalence      0.3212     0.12914     0.5497
## Balanced Accuracy         0.8107     0.61622     0.7578
```

```r
pred_prob<-predict(model_nv, newdata = Test2, type = "raw")
b<-c()
for(i in 1:3)
{
  a<-multiclass.roc(Test2$Match_O, pred_prob[,i])
  b[i]<-a$auc
}
b
```

```
## [1] 0.7916244 0.6543099 0.8200907
```

The overall accuracy of this model is 0.6722. However, let's check the Sensitivity and Specificity of our model as well.

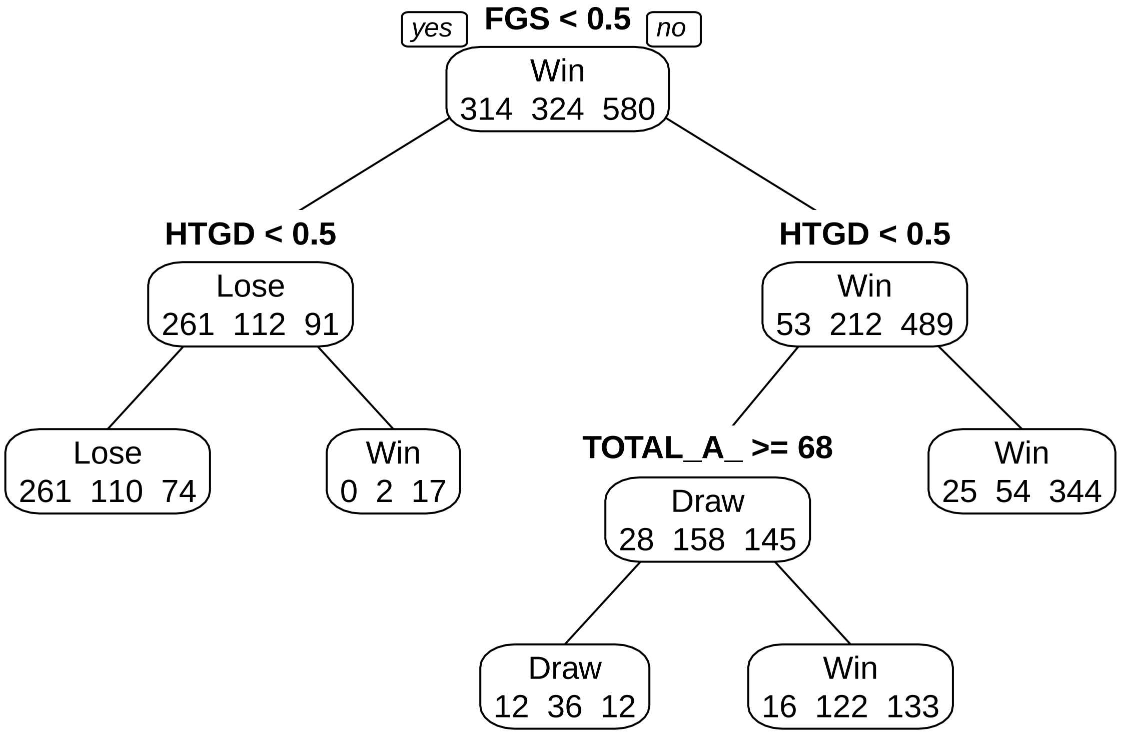
1. For class Lose, the Sensitivity and Specificity are 0.7821 and 0.8393 respectively.
2. For class Draw, the Sensitivity and Specificity are 0.30000 and 0.93243 respectively.
3. For class Win, the Sensitivity and Specificity are 0.8194 and 0.6962 respectively.

The performance of the model(AUC curve value) for class:

1. Lose is 0.7916
2. Draw is 0.6543
3. Win is 0.8200

2. Decision Tree

```r
model_dt<-rpart(Match_O~., data=Train2)
prp(model_dt, type=1, extra=1)
```



```r
pred_class<-predict(model_dt, Test2, type="class")
confusionMatrix(pred_class, Test2$Match_O)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Lose Draw Win
##       Lose   63   27  13
##       Draw    6    8   5
##       Win     9   45 126
##
## Overall Statistics
##
##                Accuracy : 0.6523
##                  95% CI : (0.5957, 0.706)
##     No Information Rate : 0.4768
##     P-Value [Acc > NIR] : 6.109e-10
##
##                   Kappa : 0.431
##  Mcnemar's Test P-Value : 5.425e-10
##
## Statistics by Class:
##
##                      Class: Lose Class: Draw Class: Win
## Sensitivity               0.8077     0.10000     0.8750
## Specificity               0.8214     0.95045     0.6582
## Pos Pred Value            0.6117     0.42105     0.7000
## Neg Pred Value            0.9246     0.74558     0.8525
## Prevalence                0.2583     0.26490     0.4768
## Detection Rate            0.2086     0.02649     0.4172
## Detection Prevalence      0.3411     0.06291     0.5960
## Balanced Accuracy         0.8146     0.52523     0.7666
```

```
pred_prob<-predict(model_dt, newdata = Test2)
b<-c()
for(i in 1:3)
{
  a<-multiclass.roc(Test2$Match_O, pred_prob[,i])
  b[i]<-a$auc
}
b
```

```
## [1] 0.6935697 0.6781940 0.8084513
```

The overall accuracy of the model is 0.6523. Now, the Sensitivity and Specificity.

1. For class Lose, the Sensitivity and Specificity are 0.8077 and 0.8214 respectively.
2. For class Draw, the Sensitivity and Specificity are 0.10000 and 0.95045 respectively.
3. For class Win, the Sensitivity and Specificity are 0.8750 and 0.6582 respectively.

The performance of the model(AUC curve value) for class:

1. Lose is 0.6935
2. Draw is 0.6781
3. Win is 0.8084

    3. KNN

```
scaled_train2<-as.data.frame(scale(Train2[,-9]))
scaled_test2<-as.data.frame(scale(Test2[,-9]))
knn<-knn(scaled_train2, scaled_test2, Train2$Match_O, k=21)
confusionMatrix(knn, Test2$Match_O)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Lose Draw Win
##       Lose   59   23  11
##       Draw    4   27  21
##       Win    15   30 112
##
## Overall Statistics
##
##                Accuracy : 0.6556
##                  95% CI : (0.5991, 0.7091)
##     No Information Rate : 0.4768
##     P-Value [Acc > NIR] : 2.915e-10
##
##                   Kappa : 0.4507
##  Mcnemar's Test P-Value : 0.001386
##
## Statistics by Class:
##
##                      Class: Lose Class: Draw Class: Win
## Sensitivity               0.7564      0.3375     0.7778
## Specificity               0.8482      0.8874     0.7152
## Pos Pred Value            0.6344      0.5192     0.7134
## Neg Pred Value             0.9091      0.7880     0.7793
## Prevalence                 0.2583      0.2649     0.4768
## Detection Rate             0.1954      0.0894     0.3709
## Detection Prevalence       0.3079      0.1722     0.5199
## Balanced Accuracy          0.8023      0.6124     0.7465
```

```
knn_p<-knn(scaled_train2, scaled_test2, Train2$Match_O, k=21, prob=T)
a<-multiclass.roc(Test2$Match_O, attr(knn_p,"prob"))
a$auc
```

```
## Multi-class area under the curve: 0.6271
```

The overall accuracy of the model is 0.6556.

1. For class Lose, the Sensitivity and Specificity are 0.7564 and 0.8482 respectively.
2. For class Draw, the Sensitivity and Specificity are 0.3375 and 0.8874 respectively.
3. For class Win, the Sensitivity and Specificity are 0.7778 and 0.7152 respectively.

The Multi-class area under the curve is approximately 0.6271.

4. Random Forest

```
set.seed(1)
model_rf1<-randomForest(Match_O~., data = Train2, ntree = 50, do.trace = T)
```

```
## ntree      OOB      1      2      3
##     1:  45.14% 36.67% 69.30% 36.36%
##     2:  44.18% 41.54% 64.40% 33.94%
##     3:  47.39% 43.21% 69.84% 36.47%
##     4:  46.15% 41.57% 67.28% 36.83%
##     5:  45.15% 45.80% 68.88% 31.73%
##     6:  45.89% 46.26% 72.00% 31.10%
##     7:  43.00% 42.48% 69.16% 28.68%
##     8:  43.12% 41.75% 72.12% 27.84%
##     9:  41.80% 41.48% 68.14% 27.40%
##    10:  40.96% 39.30% 66.98% 27.48%
##    11:  40.33% 37.90% 67.19% 26.74%
##    12:  39.80% 36.94% 66.04% 26.74%
##    13:  39.21% 37.90% 64.71% 25.65%
##    14:  39.18% 34.08% 67.49% 26.12%
##    15:  39.67% 35.35% 68.73% 25.78%
##    16:  38.54% 33.76% 69.14% 24.01%
##    17:  38.67% 33.76% 68.83% 24.48%
##    18:  37.93% 33.44% 67.59% 23.79%
##    19:  37.60% 33.44% 65.74% 24.14%
##    20:  37.27% 31.53% 66.67% 23.97%
##    21:  37.60% 30.89% 69.75% 23.28%
##    22:  37.60% 32.17% 68.83% 23.10%
##    23:  36.95% 30.89% 67.90% 22.93%
##    24:  37.19% 31.53% 69.44% 22.24%
##    25:  37.44% 30.57% 70.06% 22.93%
##    26:  38.01% 29.94% 71.30% 23.79%
##    27:  37.52% 29.30% 71.60% 22.93%
##    28:  37.36% 29.62% 70.37% 23.10%
##    29:  37.68% 29.62% 70.68% 23.62%
##    30:  37.85% 28.98% 71.91% 23.62%
##    31:  37.19% 29.30% 69.44% 23.45%
##    32:  36.86% 28.34% 68.21% 23.97%
##    33:  37.85% 30.25% 69.75% 24.14%
##    34:  36.54% 28.34% 67.90% 23.45%
##    35:  36.45% 29.94% 68.21% 22.24%
##    36:  36.62% 29.62% 69.14% 22.24%
##    37:  36.21% 28.34% 68.83% 22.24%
##    38:  36.12% 27.71% 70.06% 21.72%
##    39:  37.27% 28.03% 73.46% 22.07%
##    40:  36.37% 26.75% 72.22% 21.55%
##    41:  37.19% 27.71% 72.22% 22.76%
##    42:  36.78% 28.66% 70.99% 22.07%
##    43:  36.70% 28.98% 71.60% 21.38%
##    44:  36.37% 28.66% 70.68% 21.38%
##    45:  35.88% 28.66% 70.06% 20.69%
##    46:  36.04% 27.71% 71.91% 20.52%
##    47:  36.04% 27.39% 72.53% 20.34%
##    48:  35.88% 26.11% 72.84% 20.52%
##    49:  35.80% 25.48% 73.15% 20.52%
##    50:  35.96% 26.43% 73.15% 20.34%
```

```
pred_class_rf1<-predict(model_rf1, newdata=Test2, type="class")
confusionMatrix(pred_class_rf1, Test2$Match_O, positive = "Win")
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction Lose Draw Win
##       Lose   59   18  10
##       Draw    9   24  11
##       Win    10   38 123
## 
## Overall Statistics
## 
##                Accuracy : 0.6821
##                  95% CI : (0.6263, 0.7343)
##     No Information Rate : 0.4768
##     P-Value [Acc > NIR] : 4.642e-13
## 
##                   Kappa : 0.4848
##  Mcnemar's Test P-Value : 0.0004662
## 
## Statistics by Class:
## 
##                      Class: Lose Class: Draw Class: Win
## Sensitivity               0.7564     0.30000     0.8542
## Specificity               0.8750     0.90991     0.6962
## Pos Pred Value            0.6782     0.54545     0.7193
## Neg Pred Value            0.9116     0.78295     0.8397
## Prevalence                0.2583     0.26490     0.4768
## Detection Rate            0.1954     0.07947     0.4073
## Detection Prevalence      0.2881     0.14570     0.5662
## Balanced Accuracy         0.8157     0.60495     0.7752
```

```
pred_prob_rf1<-predict(model_rf1, newdata = Test2, type = "prob")
b<-c()
```

```
for(i in 1:3)
{
  a<-multiclass.roc(Test2$Match_O, pred_prob_rf1[,i])
  b[i]<-a$auc
}
b
```

```
## [1] 0.8319407 0.6598387 0.8340489
```

The overall accuracy of the model is 0.6854.

1. For class Lose, the Sensitivity and Specificity are 0.7692 and 0.8705 respectively.
2. For class Draw, the Sensitivity and Specificity are 0.30000 and 0.91441 respectively.
3. For class Win, the Sensitivity and Specificity are 0.8542 and 0.7025 respectively.

The performance of the model(AUC curve value) for class:

1. Lose is 0.8319
2. Draw is 0.6598
3. Win is 0.8340

5. Multinomial Logistic Regression

```
model_mlg<-multinom(Match_O ~ ., data=Train2)
```

```
## # weights:  30 (18 variable)
## initial  value 1338.109768
## iter  10 value 1099.397036
## iter  20 value 926.488689
## final  value 925.629271
## converged
```

```
summary(model_mlg)
```

```
## Call:
## multinom(formula = Match_O ~ ., data = Train2)
##
## Coefficients:
##      (Intercept)      HTGD      FGS     RED_H     RED_A   POINTS_H
## Draw -0.10299072 0.3083572 1.455401  0.175234 0.5739015 0.02043808
## Win   0.02503953 1.4381921 1.253050 -0.841165 1.0883924 0.03151897
##         POINTS_A     TOTAL_H_P   TOTAL_A_P
## Draw -0.01808147 -7.984577e-06 -0.01050688
## Win  -0.03244672  1.120005e-02 -0.01569240
##
## Std. Errors:
##      (Intercept)      HTGD       FGS     RED_H     RED_A    POINTS_H
## Draw   0.2980464 0.1167939 0.1581197 0.6159345 0.6158713 0.009857105
## Win    0.3094882 0.1310654 0.1582047 0.7710637 0.6198324 0.010130479
##         POINTS_A   TOTAL_H_P   TOTAL_A_P
## Draw 0.008772543 0.004042852 0.004249293
## Win  0.009447670 0.004283722 0.004393337
##
## Residual Deviance: 1851.259
## AIC: 1887.259
```

A one-unit increase in the variable HTGD is associated with the increase in the log odds of having draw vs away team win (ln(P(Match_O=Draw)/P(Match_O=Lose))) in the amount of 0.31.

A one-unit increase in the variable HTGD is associated with the increase in the log odds of having home team win vs away team win (ln(P(Match_O=WIn)/P(Match_O=Lose))) in the amount of 1.44.

A one-unit increase in the variable RED_A is associated with the increase in the log odds of having home team win vs away team win in the amount of 1.09.

A one-unit increase in the variable RED_H is associated with the decrease in the log odds of having home team win vs away team win in the amount of 0.84.

```
#Calculating Z-score
z<-summary(model_mlg)$coefficients/summary(model_mlg)$standard.errors
z
```

```
##      (Intercept)       HTGD      FGS      RED_H    RED_A POINTS_H  POINTS_A
## Draw -0.34555261  2.640184 9.204425  0.2845011 0.931853 2.073436 -2.061143
## Win   0.08090625 10.973085 7.920437 -1.0909152 1.755946 3.111301 -3.434362
##         TOTAL_H_P TOTAL_A_P
## Draw -0.001974986 -2.472618
## Win   2.614561590 -3.571863
```

```
#2-tailed z test for calculating p-value
p<- (1-pnorm(abs(z),0,1))*2

exp(coef(model_mlg))
```

```
##      (Intercept)     HTGD      FGS     RED_H    RED_A POINTS_H POINTS_A
## Draw   0.9021353 1.361187 4.286204 1.1915250 1.775179 1.020648 0.982081
## Win    1.0253557 4.213072 3.501006 0.4312079 2.969497 1.032021 0.968074
##      TOTAL_H_P TOTAL_A_P
## Draw  0.999992 0.9895481
## Win   1.011263 0.9844301
```

```
# changes Log odds to odds,
```

One unit increase in HTGD increases odds of having draw vs away team win (P(Match_O=Draw)/P(Match_O=Lose)) by 36 %. One unit increase in HTGD increases odds of having home team win vs away team win (P(Match_O=Win)/P(Match_O=Lose)) by 321 %.

```
#Let's see probabilities of first 10 cases
head(pp<-fitted(model_mlg))
```

```
##          Lose       Draw       Win
## 1 0.068407135 0.60254755 0.3290453
## 2 0.007365033 0.05179368 0.9408413
## 3 0.003057036 0.01799215 0.9789508
## 4 0.010473900 0.04772662 0.9417995
## 5 0.087160460 0.61032949 0.3025100
## 6 0.030545103 0.16070544 0.8087495
```

```
head(pp,10)
```

```
##           Lose       Draw       Win
## 1  0.068407135 0.60254755 0.3290453
## 2  0.007365033 0.05179368 0.9408413
## 3  0.003057036 0.01799215 0.9789508
## 4  0.010473900 0.04772662 0.9417995
## 5  0.087160460 0.61032949 0.3025100
## 6  0.030545103 0.16070544 0.8087495
## 7  0.058525244 0.18017206 0.7613027
## 8  0.081121366 0.22018710 0.6986915
## 9  0.483487579 0.27184320 0.2446692
## 10 0.424144320 0.30440679 0.2714489
```

```
pred_class <- predict (model_mlg, Test2)
confusionMatrix(pred_class, Test2$Match_O)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Lose Draw Win
##       Lose   55   21  12
##       Draw    7   25  13
##       Win    16   34 119
##
## Overall Statistics
##
##                Accuracy : 0.6589
##                  95% CI : (0.6025, 0.7123)
##     No Information Rate : 0.4768
##     P-Value [Acc > NIR] : 1.371e-10
##
##                   Kappa : 0.4485
##  Mcnemar's Test P-Value : 0.0007222
##
## Statistics by Class:
##
##                      Class: Lose Class: Draw Class: Win
## Sensitivity               0.7051     0.31250     0.8264
## Specificity               0.8527     0.90991     0.6835
## Pos Pred Value            0.6250     0.55556     0.7041
## Neg Pred Value            0.8925     0.78599     0.8120
## Prevalence                0.2583     0.26490     0.4768
## Detection Rate            0.1821     0.08278     0.3940
## Detection Prevalence      0.2914     0.14901     0.5596
## Balanced Accuracy         0.7789     0.61120     0.7550
```

```
pr_prob<-predict(model_mlg, newdata = Test2, type="probs")
b<-c()
for(i in 1:3)
{
  a<-multiclass.roc(Test2$Match_O, pr_prob[,i])
  b[i]<-a$auc
}
b
```

```
## [1] 0.8471072 0.6889408 0.8379934
```

The overall accuracy of the model is 0.6589 .

1. For class Lose, the Sensitivity and Specificity are 0.7051 and 0.8527 respectively.
2. For class Draw, the Sensitivity and Specificity are 0.31250 and 0.90991 respectively.
3. For class Win, the Sensitivity and Specificity are 0.8264 and 0.6835 respectively.

The performance of the model(AUC curve value) for class:

1. Lose is 0.8471
2. Draw is 0.6889
3. Win is 0.8379

```
##visualization

newmodel<-multinom(Match_O ~ HTGD +FGS, data=Train2)
```

```
## # weights:  12 (6 variable)
## initial  value 1338.109768
## iter  10 value 963.110151
## final  value 963.058238
## converged
```

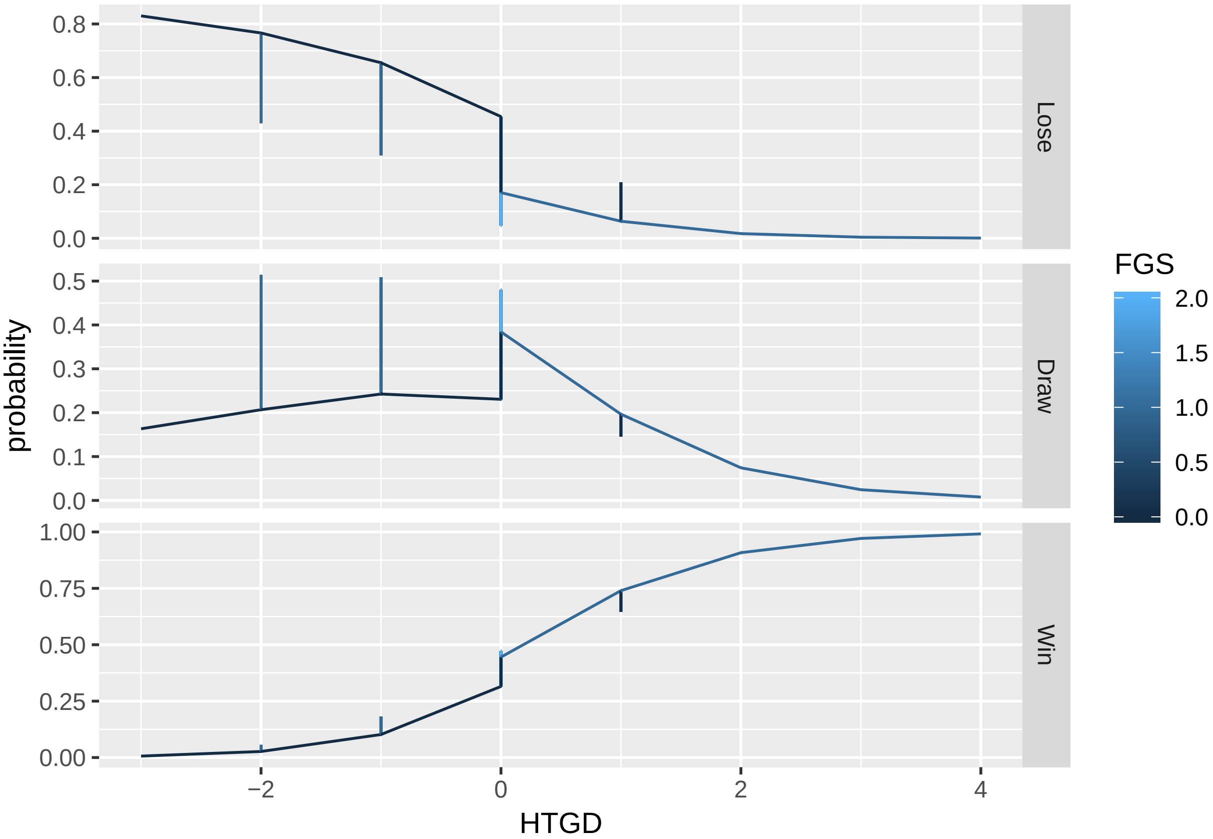
```
## store the predicted probabilities for each value of HTGD and FGS
pp.fgs <- cbind(Test2, predict(newmodel, newdata = Test2, type = "probs", se = TRUE))
by(pp.fgs[,10:12], pp.fgs$FGS,colMeans)
```

```
## pp.fgs$FGS: 0
##      Lose      Draw       Win
## 0.5748265 0.2315018 0.1936717
## ------------------------------------------------------------
## pp.fgs$FGS: 1
##       Lose       Draw        Win
## 0.07417985 0.19632709 0.72949305
## ------------------------------------------------------------
## pp.fgs$FGS: 2
##       Lose       Draw        Win
## 0.04806769 0.48024189 0.47169042
```

```
pp.fgs<-pp.fgs[,c(1,2,10:12)]

#melting for ggplot
lpp<-melt(pp.fgs,id.vars=c("HTGD", "FGS"), value.name="probability")

## plot predicted probabilities across htgd values for each level of fgs
##facetted by Match_O
ggplot(lpp, aes(x = HTGD, y = probability, colour = FGS)) + geom_line() + facet_grid(variable ~., scales
```



The graph shows the probability of each outcome of the match based on half time goal difference between teams and which team scored the first goal. For example, the probability of the win of home team increases and reaches 1 when HTGD gets more than 3 (after first half home team wins by three or more goals).

We also collected some recent data from English Premier League matches and tried to check the accuracy of multinomial regression model for those matches.

```
test<-read_excel("EPL-test.xlsx")
colnames(test)[4]<-"RED_H"
colnames(test)[5]<-"RED_A"
test <- test[,-1]
test$Match_O <- factor(test$Match_O, levels = c(0,1,2), labels = c("Lose","Draw","Win"))
pred_class <- predict (model_mlg, test)
confusionMatrix(pred_class, test$Match_O)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Lose Draw Win
##       Lose    2    0   0
##       Draw    0    1   0
##       Win     0    1   4
##
## Overall Statistics
##
##                Accuracy : 0.875
##                  95% CI : (0.4735, 0.9968)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 0.03516
##
##                   Kappa : 0.7895
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Lose Class: Draw Class: Win
## Sensitivity                 1.00      0.5000      1.000
## Specificity                 1.00      1.0000      0.750
## Pos Pred Value              1.00      1.0000      0.800
## Neg Pred Value              1.00      0.8571      1.000
## Prevalence                  0.25      0.2500      0.500
## Detection Rate              0.25      0.1250      0.500
## Detection Prevalence        0.25      0.1250      0.625
## Balanced Accuracy           1.00      0.7500      0.875
```

```
pr_prob<-predict(model_mlg, newdata = test, type="probs")
b<-c()
for(i in 1:3)
{
  a<-multiclass.roc(test$Match_O, pr_prob[,i])
  b[i]<-a$auc
}
b
```

```
## [1] 0.875 0.875 1.000
```

As you can see, our model predicted 7 out of 8 matches' outcome correctly.

In conclusion, in the case of the Multi-class Classification, we obtain the highest overall accuracy, which is 0.6854, by Random Forest method.

The highest Sensitivity, Specificity and AUC for Class Lose are 0.8077, 0.8705 and 0.8471, and we obtain them using Decision tree, Random Forest and Multinomial Logistic Regression respectfully.

The highest Sensitivity, Specificity and AUC for Class Draw are 0.3375, 0.95045 and 0.6889, and we obtain them using Knn, Decision tree and Multinomial Logistic Regression respectfully.

The highest Sensitivity, Specificity and AUC for Class Win are 0.8750, 0.7152 and 0.8379, and we obtain them using Decision tree, KNN and Multinomial Logistic Regression respectfully.

So, after the first half of a football match, if you want to place a bet on the Home team winning and want to test your prediction, then use the Decision tree model, and in the case of betting on the home team not winning - KNN model. The Decision Tree method is also useful when betting on the home team losing or the match not ending in a tie. And in case of betting that the home team will not lose, the Random Forest method will be the most useful one.