# Prediction of over/under 2.5 goals in the 1st German Soccer League

Dennis Gubenko

Ashish Panchal

Omar El-Maleh

Sameer Garg

Management Information Systems Department

San Diego State University

MIS 620: ELECTRONIC BUSINESS AND BIG DATA INFRASTRUCTURES

Dr. Aaron C. Elkins

December 17, 2021

**Table of Contents**

1 Executive Summary ........ 3

2 Discovery and Data Preparation ........ 6

3 Model Planning and Building ........ 12

4 Results and Performance ........ 13

5 Discussion and Recommendations ........ 14

References ........ 14

R-Code ........ **Fehler! Textmarke nicht definiert.**

## 1 Executive Summary

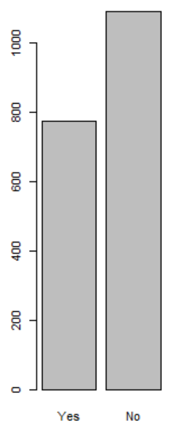
The sports betting market is very competitive and there are many different bookmakers on the market who generate good profit from the users. That means they pay out less for won bets than they earn with the stacks for lost bets over the long run. There are two reasons why this system works: The bookmakers are able to predict the matches better than the average user and they balance the odds in a way that makes the user want to bet on the games but still generates profit for the bookmakers. This is not possible with only human expertise. They use real time data analytics instead to automate and optimize the odd-building.

The goal of this project is to imitate the machine learning algorithm of the bookmakers by trying to predict, whether there will be more than 2.5 goals in each game of a whole football season. As the average amount of goals scored per game is 2.9, this threshold is used by bookmakers for a balanced distribution of the odds for this event happening. We want to test, whether our predictions can be used to generate profit on the sports betting market. If we bet $1 on the specific odd of the predicted event for every game of the season and we end up with a profit, it means that our predictions are better than the bookmakers and we can use the model to earn money with sports betting. This will probably not be possible as bookmakers do have bigger data, deeper information, and data sources, we don't have access to. We still want to see how close we can come to the data analysis of the bookmakers with our own machine learning model.

For our project we will use the historical data of 6 consecutive seasons (2014/15 – 2019/20) of the 1st German Bundesliga to build a prediction model for over/under 2.5 goals for every game, which results in a binary classification model. To test our model, we will use the 2020/21 season and see how good our prediction model performs on unseen data. Our data includes a total of 2142 matches (18 teams in the league, 9 games per matchday, 34 matchdays per season, 7 seasons), where each row represents a single match. The columns in the dataset were filtered to only those we can be potentially used for the further analysis or preprocessing. We remain with the match date, home and away team names, their scored goals, the match result, total shots, total shots on target, total fouls, total corners, yellow and red cards, and the odds for each outcome. As these are stats from the current games that we don't know before the specific game is played, we will calculate means from these columns of previous games. We distinguish between the last 6 games of a team and the entire past of a team, as well as between all games and only the ones against the specific opponent. The listing of each variable is shown in the table in Appendix 1.

First, we do a feature selection in which the multicollinearity is tested by identifying and sorting out highly correlated predictors. Then the recursive feature selection takes place, where the dataset is searched for the most relevant features that best describe the predictor variable using a 10-fold cross validation. We end up with only 7 variables left out of the initial 64.

Before we go to the model planning and building, we will take a look on the distribution of our response variable "Is_over_2.5".



We can see that most games have less than 2.5 goals per game, which results in a slight imbalance of the predicted variable which we have to keep in mind for our upcoming model building.

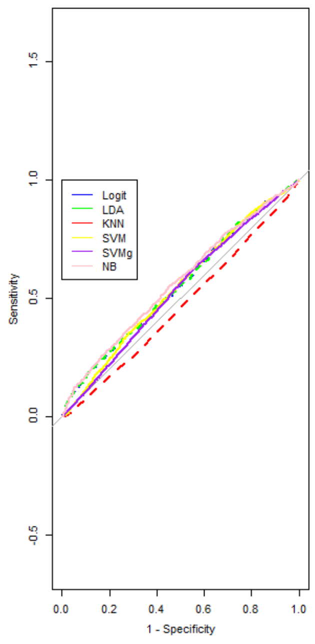
Before we dive into the model building, we set up the control function. As this is a timeseries dataset, where the date of a game actually matters, we can't simply do a cross-validation approach. The reason for that is that we can't take a game from 2020 to predict a 2016 game, as this would not reflect the reality of sports betting. Instead, we use the "timeslice"-method, where the first 306 rows are used as the training set and the next 9 games as the validation set. The next iteration takes the first 306+9 =

315 games as the training set and again the next 9 games as the validation set. In that way we always only use previous games to predict upcoming matches.

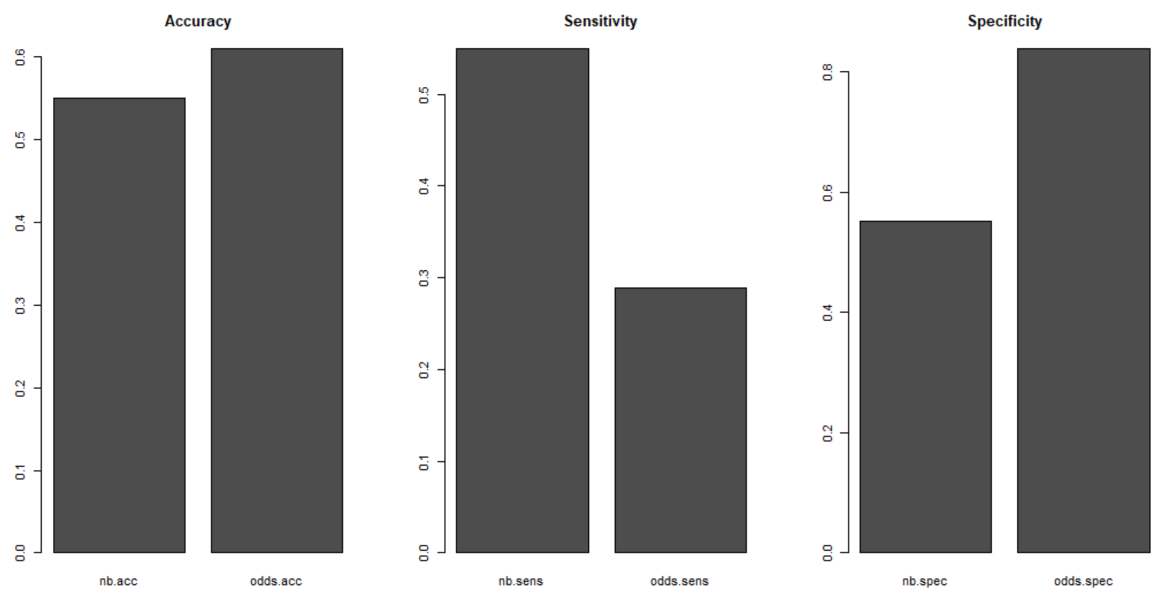
We will train the following 6 classification models to see which one performs the best:

- Logistic Regression
- Linear Discriminant Analysis
- K nearest neighbor classification
- Support Vector Machine (with and without grid)
- Naive Bayes

We used the Area under-ROC-Curve metric to decide for the best performing model. But after calculating the accuracy of each classifier, we adjusted the threshold of each classifier to optimize the accuracy. We finally ended up with the highest accuracy score of 55% with the Naïve Bayes Classifier. The image below shows how well it performed compared to the other models and it can be seen, that the accuracies are really close.



By comparing our predictions to the bookmakers, where the lower odds implicate that they are predicting that outcome, we can see that they indeed have slightly better (61%) predictions than our model (55%). What stands out, is the higher sensitivity which implicates that our model is better in predicting high scoring games.



To test the real value of our prediction accuracy, we want to test our betting strategy by betting $1 on every predicted match event and see, how much will be left on our bank account.

We lost $49.23 over the season 2020/21, so the model is not able to generate profit for us on the betting market. As our model seems to predict Over 2.5 goals better than Under 2.5, we will test another betting strategy, by only betting on over goals predictions. By doing that we still end up losing $9.78.

The conclusion of our project is that we are still far away from competing with the sports betting industry. It is also hard to compare our model to the odds, as they build their models not to predict the goals right, but to generate profit with the odds. They also include the betting behavior of the users to build the optimal odds for them. In our opinion, it is really hard to beat the market with such a simple dataset, because the bookmakers have a huge advantage in terms of the size of their data.

To improve our model, we should balance out the class imbalance with an up- or down sampling method. The inclusion of more variables, such as injured or suspended players, could also improve our accuracy.

## 2 Discovery and Data Preparation

The goal of this project is to predict the event Over/Under 2.5 goals of soccer matches in the 1st Bundesliga in Germany from the season 2020/2021 as accurately as possible. We will be using history data from the seasons 2014/2015 to 2019/2020 and try to achieve better results than the betting providers. As there are two different possible outcomes, it's a binary classification problem. Each line in the data set represents a single match in the 1st soccer Bundesliga, and since there are 18 teams in the league which are playing against each at in 34 match days, there are a total of 306 matches per game, resulting in 306 observations per year.

There are several providers of databases on European soccer matches on the Internet with different contents. The datasets from Football Data UK are particularly suitable for our project for various reasons. They have a quality information content and already contain some of the variables needed for the project. In addition, the data provide an essential contribution for the creation of the required variables, so further variables can be derived from the data. Furthermore, another advantage is that there are no missing values in the data set, so the data quality is pretty good. The individual data sets are in CSV format and can therefore be easily read into R.

First, the goal is to create the overall data set from the data sets of each single season. Since not all columns are needed, a vector with the required columns is created first. Because in the last seasons other column labels were used and additional data were added than at the beginning, a different vector must be applied for these periods. In addition, the columns are transferred to a uniform labeling. The individual data records are then merged into one data set and individual data sets are removed after this. The data set now contains 2,142 observations and 21 total columns. When checking the data for missing values, there are no missing data.

The next step is to create a function to generate the variables, which are needed to predict the outcome of the game. A distinction is made between:

- data for the home team and data for the away team
- the current form of the team in the last 6 games and the past games in total

- all matches and matches against a specific opponent

The goal is to provide the current form of the team by generating variables for the last 6 games of the respective teams and to include the form against the respective opponent and the overall form. On the other hand, variables are generated, which give information about the complete past games. This is done in 2 steps, first the respective data (goals, shots, cards, etc.) are added together, before sums or averages are calculated from them. In the form of the last 6 games, it should be noted that there are not enough past games at the beginning of the data set, so this data is omitted. Yellow and red cards are first considered separately and converted into a point system. Yellow cards are calculated with 10 points, red cards with 25 points, to be able to calculate a variable for the cards at the end for the total number of points of cards per game, thus yellow cards and red cards cumulated. For points, the same is added up and divided by the number of games to get an average number of points per game. Goals scored, goals conceded, shots and shots on goal, corners and fouls are also averaged per match. The steps are executed 2 times, once for the respective home team and once for the respective away team. This step contains all variables, which are shown in Appendix 1.

As for some matches or teams, there are no previous games (for example in the first season or for promoted teams), so there are no statistics to be computed and we just delete these games from our dataset. We end up with 1864 observations out of the initial 2142.

For the next steps it is necessary to convert the created columns into numeric columns and to round the numbers to 2 decimal places for clarity. Lines in which data are missing are also removed and are not considered in the following calculations. Columns that were necessary for the generation of the variables but are no longer needed are then sorted out. Feature selection now follows, testing for multicollinearity by identifying and sorting out highly correlated predictors (threshold = 0.75).

The following 28 variables are subsequently excluded:

[1] "HT_card_points_form_opp"

[2] "AT_Mean_card_points"

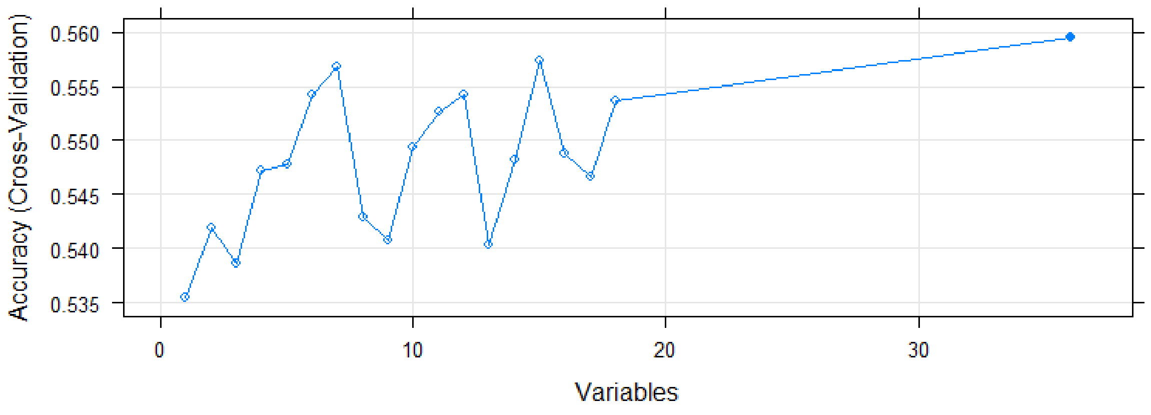
[3] "HT_shots_form_opp"

```
[4] "HT_Mean_card_points_opp"
[5] "AT_Mean_shots"
[6] "HT_Form_Points_opp"
[7] "AT_Form_Scored_opp"
[8] "AT_Mean_Conceded_opp"
[9] "AT_card_points_form"
[10] "AT_Mean_Points"
[11] "HT_Form_Conceded_opp"
[12] "HT_Mean_Points"
[13] "HT_Mean_shots"
[14] "AT_shots_form_opp"
[15] "HT_card_points_form"
[16] "HT_Form_Scored_opp"
[17] "AT_Form_Points_opp"
[18] "AT_Mean_Conceded"
[19] "AT_Form_Conceded_opp"
[20] "HT_Mean_Conceded"
[21] "HT_Mean_Scored_opp"
[22] "HT_Mean_Points_opp"
[23] "HT_Form_Points"
[24] "HT_shots_form"
[25] "HT_Mean_card_points"
[26] "AT_shots_form"
[27] "AT_corners_form_opp"
[28] "HT_fouls_form_opp"
```

Now the recursive feature selection takes place, where a seed of 101 is set, so that the result is reproducible. The recursive feature selection searches the data set for the most important features that describe the predictor variable the most. Up to 18 variables are to be retained, testing which variables achieve the highest accuracy. The result of the recursive feature elimination, including accuracy and kappa as well as the predictors are shown in the following tables:

| **Recursive feature selection** (Outer resampling method: Cross-Validated 10-fold) Resampling performance over subset size: | | | | |
|---|---|---|---|---|
| Variables | Accuracy | Kappa | AccuracySD | KappaSD |
| 1 | 0.5354 | -0.021815 | 0.03028 | 0.03814 |
| 2 | 0.5419 | 0.026737 | 0.02280 | 0.05708 |
| 3 | 0.5386 | 0.014818 | 0.03844 | 0.07840 |
| 4 | 0.5472 | 0.035047 | 0.04943 | 0.10873 |
| 5 | 0.5478 | 0.033851 | 0.04162 | 0.08736 |
| 6 | 0.5542 | 0.045540 | 0.02898 | 0.06501 |
| 7 | 0.5569 | 0.047681 | 0.02784 | 0.06328 |
| 8 | 0.5429 | 0.015032 | 0.02496 | 0.04747 |
| 9 | 0.5408 | 0.015594 | 0.03404 | 0.07169 |
| 10 | 0.5494 | 0.030326 | 0.03138 | 0.06565 |
| 11 | 0.5526 | 0.040512 | 0.03072 | 0.06429 |
| 12 | 0.5542 | 0.040265 | 0.02414 | 0.04684 |
| 13 | 0.5402 | 0.009789 | 0.02296 | 0.04354 |
| 14 | 0.5482 | 0.026003 | 0.03090 | 0.06387 |
| 15 | 0.5574 | 0.042399 | 0.03927 | 0.08489 |
| 16 | 0.5488 | 0.022353 | 0.02783 | 0.06119 |
| 17 | 0.5467 | 0.019690 | 0.03132 | 0.06626 |
| 18 | 0.5537 | 0.028714 | 0.03400 | 0.07504 |
| 36 | 0.5574 | 0.042399 | 0.03927 | 0.08489 |

| **Predictors** | | | |
|---|---|---|---|
| [1] | AT_Mean_shots_on_target_opp | [2] | HT_fouls_form |
| [3] | HT_shots_on_target_form | [4] | AT_Mean_fouls_opp |
| [5] | AT_Form_Conceded | [6] | AT_Mean_corners |
| [7] | HT_Mean_Scored | | |



The graph shows that the accuracy increases sharply with the inclusion of the first variables up until 7 variables. After the accuracy goes down again and only exceeds the accuracy of 7 predictors whit inclusion of all 36 variables. The final dataset contains the following variables:

```
[1] "Is_over_2.5"           "AT_Mean_shots_on_target_opp"
[3] "HT_fouls_form"           "HT_shots_on_target_form"
[5] "AT_Mean_fouls_opp"        "AT_Form_Conceded"
[7] "AT_Mean_corners"          "HT_Mean_Scored"
```

## 3 Model Planning and Building

First, the descriptive statistics are performed. After evaluating the dependent variable “Is_over_2.5”, it shows that the matches resulted 775 times in over 2.5 goals and 1089 times in under 2.5 goals. This imbalance must be kept in mind. The following table shows the descriptive statistics of the independent variables.

| | [2] | [3] | [4] | [5] | [6] | [7] | [8] |
|---|---|---|---|---|---|---|---|
| Min. | 0.000 | 46.00 | 6.00 | 4.00 | 0.000 | 3.00 | 0.530 |
| 1st Qu. | 3.600 | 68.00 | 22.00 | 12.17 | 7.000 | 4.28 | 1.190 |
| Median | 4.520 | 78.00 | 27.00 | 14.00 | 9.000 | 4.57 | 1.350 |
| Mean | 4.738 | 78.92 | 28.06 | 14.06 | 8.883 | 4.77 | 1.447 |
| 3rd Qu. | 5.750 | 89.00 | 33.00 | 15.67 | 11.000 | 5.20 | 1.640 |
| Max. | 14.000 | 124.00 | 57.00 | 29.00 | 19.000 | 8.12 | 2.800 |

Now the models are trained. First a control function is set up with timeslice to make rolling forecasts for every matchday (9 games) based on the whole data before the respective matchday. Doing that, we make sure that we always only use data that is prior to the predicted games. In the real world, you can’t predict a 2016 game with stats from a game in 2020, so we have to consider that in our model.

The following 6 classification models are used:

- Logistic Regression
- Linear Discriminant Analysis
- K nearest neighbors classification
- Support Vector Machine (with and without grid)
- Naive Bayes Classifier

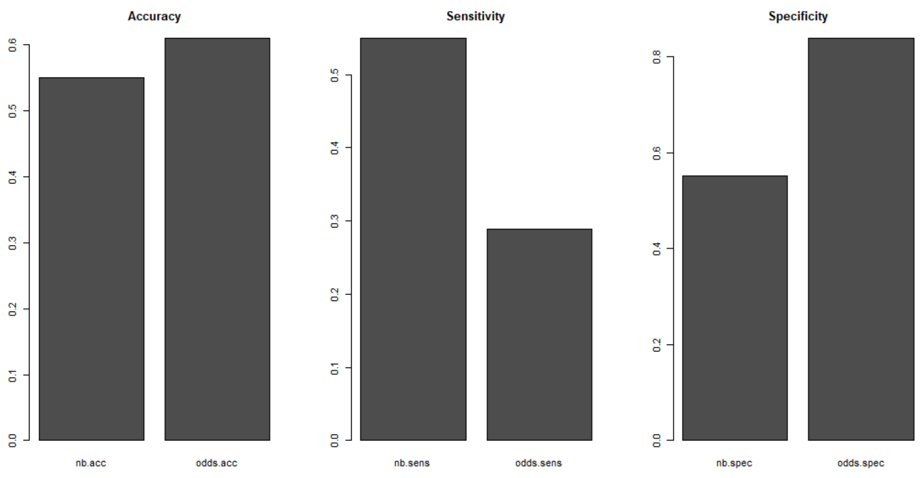
After building the confusion matrix for each model, we will extract the optimal threshold for each model and make new predictions, to increase the accuracy of our models.

## 4 Results and Performance

After looking at the confusion matrices of every model with the new thresholds, the Naïve Bayes Classifier had the highest accuracy score. By changing the probability threshold from 0.5 to 0.4557247, the overall accuracy dropped from 57% to 55% as well as the Specificity from 70% to 55%, but we were able to increase the sensitivity from 38% to 55%, which results in more balanced predictions. The confusion matrix of our predictions shows that were better at predicting under 2.5 goals.

```
          Reference
Prediction Yes  No
       Yes 627 739
       No  515 909
```

The next step is to calculate the accuracy of the Bookmakers. Their predictions are basically the lower of the two odds. In the following visualization we can see that the bookmakers have better predictions than our model (61% vs 55%), but we manage to predict the Over 2.5 games better than the bookmakers. This is surprising, as we internally are better in predicting Under 2.5 games, but compared to the bookmakers we are better in Over 2.5 games.



Now we want to test the economic efficiency of our outcome probabilities and compare it with the bookmaker odds. The betting strategy is as following: always bet on the prediction. We bet 1$ hypothetically on every predicted match event. If the prediction is right, we multiply the specific odd for the outcome with 1$ and add it to our "bank account". At the end we see, how much money we have won or lost.

It comes out that we lose $49.23 with our betting strategy, with a total stack of 306$ (every game of the season). As we seem to predict Over 2.5 game better than the bookmakers, we try out only betting on these predictions. We end up losing only $9.78, which still is not enough for being a usable model in the real world.

## 5 Discussion and Recommendations

Overall, it can be summarized that the model is not suitable for predicting the Over/Under 2.5 goals event. The problem lies in the fact that bookmakers have a huge advantage when it comes to the size of their data and the number of predictors they have. They can use real life data like the user behavior to optimize the odds for them to generate the most profit possible. We can still try to improve our model, by considering which players are in the starting eleven, the club's injury history, and which players are suspended due to red cards. For example, it can be assumed that if a team has the best players ready and in the starting lineup, that the performance will be better. This is not reflected in our model because there is no information about the starting lineup. In addition, the strength of the players could be used with ratings, for example from the soccer manager game FIFA, to consider the strength of the team and generate an overall score for the team. Also we should have balanced out the distribution of the dependent variable with an up- or downsampling, what would probably improve the accuracy of our model.

All in all it can be said, that it is an interesting idea to investigate how far we can push our model accuracy and economic efficiency by doing further research, data preparation, variable tuning or model testing. With enough time invested, I think it is possible to come close to a model that is able to compete with the sports betting market.

## References

Football results, Statistics &amp; Soccer Betting Odds Data. (n.d.). Retrieved December 1, 2021, from http://www.football-data.co.uk/data.php.

**R-Code**

```
library(dplyr)

library(tidyverse)

library(ggplot2)

library(caret)

library(e1071)

library(psych)

library(corrplot)

library(corrgram)

library(pROC)

library(leaps)

library(rpart)

library(tree)

library(rpart.plot)

library(doParallel)

library(MLmetrics)

library(klaR)


#Build the dataframe----

season0 <- read.csv("D1 (14).csv", stringsAsFactors = T)

season1 <- read.csv("D1 (12).csv", stringsAsFactors = T)

season2 <- read.csv("D1 (11).csv", stringsAsFactors = T)

season3 <- read.csv("D1 (10).csv", stringsAsFactors = T)

season4 <- read.csv("D1 (9).csv", stringsAsFactors = T)

season5 <- read.csv("D1 (8).csv", stringsAsFactors = T)

season6 <- read.csv("D1 (7).csv", stringsAsFactors = T)
```

```
cols_1 <- c("Date", "HomeTeam", "AwayTeam","FTHG",          "FTAG","FTR"    ,"HS"    ,"AS"
        ,"HST","AST","HF","AF","HC","AC","HY","AY","HR","AR","BbAvH","BbAvD","BbAvA","BbAv.2.5
","BbAv.2.5.1")
cols_2 <- c("Date", "HomeTeam", "AwayTeam","FTHG",          "FTAG","FTR"    ,"HS"    ,"AS"
        ,"HST","AST","HF","AF","HC","AC","HY","AY","HR","AR","AvgH","AvgD","AvgA","Avg.2.5","Av
g.2.5.1")

season0 <- season0[cols_1]
season1 <- season1[cols_1]
season2 <- season2[cols_1]
season3 <- season3[cols_1]
season4 <- season4[cols_1]
season5 <- season5[cols_2]
season6 <- season6[cols_2]

season0 <- rename(season0, H_Odds = BbAvH, D_Odds = BbAvD, A_Odds = BbAvA, Over_2.5 =
BbAv.2.5, Under_2.5 = BbAv.2.5.1)
season1 <- rename(season1, H_Odds = BbAvH, D_Odds = BbAvD, A_Odds = BbAvA, Over_2.5 =
BbAv.2.5, Under_2.5 = BbAv.2.5.1)
season2 <- rename(season2, H_Odds = BbAvH, D_Odds = BbAvD, A_Odds = BbAvA, Over_2.5 =
BbAv.2.5, Under_2.5 = BbAv.2.5.1)
season3 <- rename(season3, H_Odds = BbAvH, D_Odds = BbAvD, A_Odds = BbAvA, Over_2.5 =
BbAv.2.5, Under_2.5 = BbAv.2.5.1)
season4 <- rename(season4, H_Odds = BbAvH, D_Odds = BbAvD, A_Odds = BbAvA, Over_2.5 =
BbAv.2.5, Under_2.5 = BbAv.2.5.1)
```

```
season5 <- rename(season5, H_Odds = AvgH, D_Odds = AvgD, A_Odds = AvgA, Over_2.5 = Avg.2.5,
Under_2.5 = Avg.2.5.1)
season6 <- rename(season6, H_Odds = AvgH, D_Odds = AvgD, A_Odds = AvgA, Over_2.5 = Avg.2.5,
Under_2.5 = Avg.2.5.1)

df <- rbind(season0, season1, season2, season3, season4, season5, season6)
rm(season0, season1, season2, season3, season4, season5, season6)

sum(is.na(df))

#Build functions to compute new variables-------
#create a function that takes a teams last 6/total games and sums up the stats
stat_function <- function(df, team, date) {
  last_6 <- tail(df[(df$HomeTeam == team | df$AwayTeam == team) & (df$Date < date),])
  past <- df[(df$HomeTeam == team | df$AwayTeam == team) & (df$Date < date),]
  point_count <- 0
  scored_count <- 0
  conceded_count <- 0
  shots_count <- 0
  shots_on_target_count <- 0
  corners_count <- 0
  fouls_count <- 0
  yellow_card_count <- 0
  red_card_count <- 0
  point_count_6 <- 0
  scored_count_6 <- 0
```

```
conceded_count_6 <- 0
shots_count_6 <- 0
shots_on_target_count_6 <- 0
corners_count_6 <- 0
fouls_count_6 <- 0
yellow_card_count_6 <- 0
red_card_count_6 <- 0
if(nrow(last_6) < 3){


}
else {
  for (match in 1:nrow(last_6)){
    m <- last_6[match,]
    if (m$HomeTeam == team){
      scored_count_6 <- scored_count_6 + m$FTHG
      conceded_count_6 <- conceded_count_6 + m$FTAG
      shots_count_6 <- shots_count_6 + m$HS
      shots_on_target_count_6 <- shots_on_target_count_6 + m$HST
      corners_count_6 <- corners_count_6 + m$HC
      fouls_count_6 <- fouls_count_6 + m$HF
      yellow_card_count_6 <- yellow_card_count_6 + m$HY*10
      red_card_count_6 <- red_card_count_6 + m$HR*25
      if (m$FTR == "H"){
        point_count_6 <- point_count_6 + 3
      }
      else if (m$FTR == "D"){
        point_count_6 <- point_count_6 +1
      }
    }
    else {
      scored_count_6 <- scored_count_6 + m$FTAG
      conceded_count_6 <- conceded_count_6 + m$FTHG
      shots_count_6 <- shots_count_6 + m$AS
      shots_on_target_count_6 <- shots_on_target_count_6 + m$AST
      corners_count_6 <- corners_count_6 + m$AC
      fouls_count_6 <- fouls_count_6 + m$AF
      yellow_card_count_6 <- yellow_card_count_6 + m$AY*10
      red_card_count_6 <- red_card_count_6 + m$AR*25
      if (m$FTR == "A"){
        point_count_6 <- point_count_6 + 3
      }
      else if (m$FTR == "D"){
        point_count_6 <- point_count_6 +1
      }
    }
  }
}

if(nrow(past) < 3){


}
else {
```

```
  for (match in 1:nrow(past)){
    m <- past[match,]
    if (m$HomeTeam == team){
      scored_count <- scored_count + m$FTHG
      conceded_count <- conceded_count + m$FTAG
      shots_count <- shots_count + m$HS
      shots_on_target_count <- shots_on_target_count + m$HST
      corners_count <- corners_count + m$HC
      fouls_count <- fouls_count + m$HF
      yellow_card_count <- yellow_card_count + m$HY*10
      red_card_count <- red_card_count + m$HR*25
      if (m$FTR == "H"){
        point_count <- point_count + 3
      }
      else if (m$FTR == "D"){
        point_count <- point_count +1
      }
    }
    else {
      scored_count <- scored_count + m$FTAG
      conceded_count <- conceded_count + m$FTHG
      shots_count <- shots_count + m$AS
      shots_on_target_count <- shots_on_target_count + m$AST
      corners_count <- corners_count + m$AC
      fouls_count <- fouls_count + m$AF
      yellow_card_count <- yellow_card_count + m$AY*10
      red_card_count <- red_card_count + m$AR*25
      if (m$FTR == "A"){
        point_count <- point_count + 3
      }
      else if (m$FTR == "D"){
        point_count <- point_count +1
      }
    }
  }
}
games_count <- nrow(past)
points <- point_count/games_count
mean_scored <- scored_count/games_count
mean_conceded <- conceded_count/games_count
mean_shots <- shots_count/games_count
mean_shots_on_target <- shots_on_target_count/games_count
mean_corners <- corners_count/games_count
mean_fouls <- fouls_count/games_count
card_points <- (yellow_card_count + red_card_count)/games_count
card_points_6 <- yellow_card_count_6 + red_card_count_6

result <<- data.frame("point_form" = point_count_6,
                      "scored_form" = scored_count_6,
                      "conceded_form" = conceded_count_6,
                      "shots_form" = shots_count_6,
                      "shots_on_target_form" = shots_on_target_count_6,
                      "corners_form" = corners_count_6,
                      "fouls_form" = fouls_count_6,
                      "card_points_form" = card_points_6,
                      "mean_points" <- points,
                      "mean_scored" = mean_scored,
                      "mean_conceded" = mean_conceded,
                      "mean_shots" = mean_shots,
                      "mean_shots_on_target" = mean_shots_on_target,
                      "mean_corners" = mean_corners,
                      "mean_fouls" = mean_fouls,
                      "card_points" = card_points
                      )

result
}



#create a function that takes a teams last 6/total games vs Opponent and sums up the stats
stat_function_opp <- function(df, HT, AT, date){
last_6 <- tail(df[((df$HomeTeam == HT & df$AwayTeam == AT) | (df$HomeTeam == AT &
df$AwayTeam == HT)) & (df$Date < date),])
past <-df[((df$HomeTeam == HT & df$AwayTeam == AT) | (df$HomeTeam == AT & df$AwayTeam ==
HT)) & (df$Date < date),]
point_count <- 0
scored_count <- 0
conceded_count <- 0
```

```
shots_count <- 0

shots_on_target_count <- 0

corners_count <- 0

fouls_count <- 0

yellow_card_count <- 0

red_card_count <- 0

point_count_6 <- 0

scored_count_6 <- 0

conceded_count_6 <- 0

shots_count_6 <- 0

shots_on_target_count_6 <- 0

corners_count_6 <- 0

fouls_count_6 <- 0

yellow_card_count_6 <- 0

red_card_count_6 <- 0

if(nrow(last_6) == 0){


}

else{

  for (match in 1:nrow(last_6)){

    m <- last_6[match,]

    if (m$HomeTeam == HT){

      scored_count_6 <- scored_count_6 + m$FTHG

      conceded_count_6 <- conceded_count_6 + m$FTAG

      shots_count_6 <- shots_count_6 + m$HS

      shots_on_target_count_6 <- shots_on_target_count_6 + m$HST
      corners_count_6 <- corners_count_6 + m$HC
      fouls_count_6 <- fouls_count_6 + m$HF
      yellow_card_count_6 <- yellow_card_count_6 + m$HY*10
      red_card_count_6 <- red_card_count_6 + m$HR*25
      if (m$FTR == "H"){
        point_count_6 <- point_count_6 + 3
      }
      else if (m$FTR == "D"){
        point_count_6 <- point_count_6 +1
      }
    }
    else {
      scored_count_6 <- scored_count_6 + m$FTAG
      conceded_count_6 <- conceded_count_6 + m$FTHG
      shots_count_6 <- shots_count_6 + m$AS
      shots_on_target_count_6 <- shots_on_target_count_6 + m$AST
      corners_count_6 <- corners_count_6 + m$AC
      fouls_count_6 <- fouls_count_6 + m$AF
      yellow_card_count_6 <- yellow_card_count_6 + m$AY*10
      red_card_count_6 <- red_card_count_6 + m$AR*25
      if (m$FTR == "A"){
        point_count_6 <- point_count_6 + 3
      }
      else if (m$FTR == "D"){
        point_count_6 <- point_count_6 +1
      }
```

```
      }
    }
  }
  if(nrow(past) == 0){


  }
  else{
    for (match in 1:nrow(past)){
      m <- past[match,]
      if (m$HomeTeam == HT){
        scored_count <- scored_count + m$FTHG
        conceded_count <- conceded_count + m$FTAG
        shots_count <- shots_count + m$HS
        shots_on_target_count <- shots_on_target_count + m$HST
        corners_count <- corners_count + m$HC
        fouls_count <- fouls_count + m$HF
        yellow_card_count <- yellow_card_count + m$HY*10
        red_card_count <- red_card_count + m$HR*25
        if (m$FTR == "H"){
          point_count <- point_count + 3
        }
        else if (m$FTR == "D"){
          point_count <- point_count +1
        }
      }
      else {
        scored_count <- scored_count + m$FTAG
        conceded_count <- conceded_count + m$FTHG
        shots_count <- shots_count + m$AS
        shots_on_target_count <- shots_on_target_count + m$AST
        corners_count <- corners_count + m$AC
        fouls_count <- fouls_count + m$AF
        yellow_card_count <- yellow_card_count + m$AY*10
        red_card_count <- red_card_count + m$AR*25
        if (m$FTR == "A"){
          point_count <- point_count + 3
        }
        else if (m$FTR == "D"){
          point_count <- point_count +1
        }
      }
    }
  }
  games_count <- nrow(past)
  points <- point_count/games_count
  mean_scored <- scored_count/games_count
  mean_conceded <- conceded_count/games_count
  mean_shots <- shots_count/games_count
  mean_shots_on_target <- shots_on_target_count/games_count
  mean_corners <- corners_count/games_count
  mean_fouls <- fouls_count/games_count
  card_points <- (yellow_card_count + red_card_count)/games_count
  card_points_6 <- yellow_card_count_6 + red_card_count_6



  result <<- data.frame("point_form_opp" = point_count_6,
                        "scored_form_opp" = scored_count_6,
                        "conceded_form_opp" = conceded_count_6,
                        "shots_form_opp" = shots_count_6,
                        "shots_on_target_form_opp" = shots_on_target_count_6,
                        "corners_form_opp" = corners_count_6,
                        "fouls_form_opp" = fouls_count_6,
                        "card_points_form_opp" = card_points_6,
                        "mean_poins_opp" <- points,
                        "mean_scored_opp" = mean_scored,
                        "mean_conceded_opp" = mean_conceded,
                        "mean_shots_opp" = mean_shots,
                        "mean_shots_on_target_opp" = mean_shots_on_target,
                        "mean_corners_opp" = mean_corners,
                        "mean_fouls_opp" = mean_fouls,
                        "card_points_opp" = card_points
  )

  result
}

df$Date <- as.Date(df$Date, "%d/%m/%y")
df$goals <- df$FTHG + df$FTAG
```

```
df$Is_over_2.5 <- ifelse(df$goals > 2.5,1,0)


#Apply functions to dataframe----
#Create cols for the new variables that will be computed by the functions
#Distinguish between: HT and AT, Last 6 games and total past games, all games and against specific opponent


df$HT_Form_Points <- NA
df$HT_Form_Scored <- NA
df$HT_Form_Conceded <- NA
df$HT_shots_form <- NA
df$HT_shots_on_target_form <- NA
df$HT_corners_form <- NA
df$HT_fouls_form <- NA
df$HT_card_points_form <- NA


df$HT_Mean_Points <- NA
df$HT_Mean_Scored <- NA
df$HT_Mean_Conceded <- NA
df$HT_Mean_shots <- NA
df$HT_Mean_shots_on_target <- NA
df$HT_Mean_corners <- NA
df$HT_Mean_fouls<- NA
df$HT_Mean_card_points <- NA


df$AT_Form_Points <- NA
```

```
df$AT_Form_Scored <- NA

df$AT_Form_Conceded <- NA

df$AT_shots_form <- NA

df$AT_shots_on_target_form <- NA

df$AT_corners_form <- NA

df$AT_fouls_form <- NA

df$AT_card_points_form <- NA


df$AT_Mean_Points <- NA

df$AT_Mean_Scored <- NA

df$AT_Mean_Conceded <- NA

df$AT_Mean_shots <- NA

df$AT_Mean_shots_on_target <- NA

df$AT_Mean_corners <- NA

df$AT_Mean_fouls <- NA

df$AT_Mean_card_points <- NA


df$HT_Form_Points_opp <- NA

df$HT_Form_Scored_opp <- NA

df$HT_Form_Conceded_opp <- NA

df$HT_shots_form_opp <- NA

df$HT_shots_on_target_form_opp <- NA

df$HT_corners_form_opp <- NA

df$HT_fouls_form_opp <- NA

df$HT_card_points_form_opp <- NA
```

```
df$HT_Mean_Points_opp <- NA

df$HT_Mean_Scored_opp <- NA

df$HT_Mean_Conceded_opp <- NA

df$HT_Mean_shots_opp <- NA

df$HT_Mean_shots_on_target_opp <- NA

df$HT_Mean_corners_opp <- NA

df$HT_Mean_fouls_opp <- NA

df$HT_Mean_card_points_opp <- NA


df$AT_Form_Points_opp <- NA

df$AT_Form_Scored_opp <- NA

df$AT_Form_Conceded_opp <- NA

df$AT_shots_form_opp <- NA

df$AT_shots_on_target_form_opp <- NA

df$AT_corners_form_opp <- NA

df$AT_fouls_form_opp <- NA

df$AT_card_points_form_opp <- NA


df$AT_Mean_Points_opp <- NA

df$AT_Mean_Scored_opp <- NA

df$AT_Mean_Conceded_opp <- NA

df$AT_Mean_shots_opp <- NA

df$AT_Mean_shots_on_target_opp <- NA

df$AT_Mean_corners_opp <- NA

df$AT_Mean_fouls_opp <- NA

df$AT_Mean_card_points_opp <- NA
```

```
#apply the form functions on every match of the df
for (i in 1:nrow(df)){
  stat_home <- stat_function(df, df[i,]$HomeTeam, df[i,]$Date)
  df[i,]$HT_Form_Points <- stat_home[1]
  df[i,]$HT_Form_Scored <- stat_home[2]
  df[i,]$HT_Form_Conceded <- stat_home[3]
  df[i,]$HT_shots_form <- stat_home[4]
  df[i,]$HT_shots_on_target_form <- stat_home[5]
  df[i,]$HT_corners_form <- stat_home[6]
  df[i,]$HT_fouls_form <- stat_home[7]
  df[i,]$HT_card_points_form <- stat_home[8]

  df[i,]$HT_Mean_Points <- stat_home[9]
  df[i,]$HT_Mean_Scored <- stat_home[10]
  df[i,]$HT_Mean_Conceded <- stat_home[11]
  df[i,]$HT_Mean_shots <- stat_home[12]
  df[i,]$HT_Mean_shots_on_target <- stat_home[13]
  df[i,]$HT_Mean_corners <- stat_home[14]
  df[i,]$HT_Mean_fouls<- stat_home[15]
  df[i,]$HT_Mean_card_points <- stat_home[16]

  stat_away <- stat_function(df, df[i,]$AwayTeam, df[i,]$Date)
  df[i,]$AT_Form_Points <- stat_away[1]
  df[i,]$AT_Form_Scored <- stat_away[2]
```

```
df[i,]$AT_Form_Conceded <- stat_away[3]

df[i,]$AT_shots_form <- stat_away[4]

df[i,]$AT_shots_on_target_form <- stat_away[5]

df[i,]$AT_corners_form <- stat_away[6]

df[i,]$AT_fouls_form <- stat_away[7]

df[i,]$AT_card_points_form <- stat_away[8]


df[i,]$AT_Mean_Points <- stat_away[9]

df[i,]$AT_Mean_Scored <- stat_away[10]

df[i,]$AT_Mean_Conceded <- stat_away[11]

df[i,]$AT_Mean_shots <- stat_away[12]

df[i,]$AT_Mean_shots_on_target <- stat_away[13]

df[i,]$AT_Mean_corners <- stat_away[14]

df[i,]$AT_Mean_fouls<- stat_away[15]

df[i,]$AT_Mean_card_points <- stat_away[16]


stat_home_opp <- stat_function_opp(df, df[i,]$HomeTeam, df[i,]$AwayTeam, df[i,]$Date)

df[i,]$HT_Form_Points_opp <- stat_home_opp[1]

df[i,]$HT_Form_Scored_opp <- stat_home_opp[2]

df[i,]$HT_Form_Conceded_opp <- stat_home_opp[3]

df[i,]$HT_shots_form_opp <- stat_home_opp[4]

df[i,]$HT_shots_on_target_form_opp <- stat_home_opp[5]

df[i,]$HT_corners_form_opp <- stat_home_opp[6]

df[i,]$HT_fouls_form_opp <- stat_home_opp[7]

df[i,]$HT_card_points_form_opp <- stat_home_opp[8]
```

```
df[i,]$HT_Mean_Points_opp <- stat_home_opp[9]

df[i,]$HT_Mean_Scored_opp <- stat_home_opp[10]

df[i,]$HT_Mean_Conceded_opp <- stat_home_opp[11]

df[i,]$HT_Mean_shots_opp <- stat_home_opp[12]

df[i,]$HT_Mean_shots_on_target_opp <- stat_home_opp[13]

df[i,]$HT_Mean_corners_opp <- stat_home_opp[14]

df[i,]$HT_Mean_fouls_opp<- stat_home_opp[15]

df[i,]$HT_Mean_card_points_opp <- stat_home_opp[16]


stat_away_opp <- stat_function_opp(df, df[i,]$AwayTeam, df[i,]$HomeTeam, df[i,]$Date)

df[i,]$AT_Form_Points_opp <- stat_away_opp[1]

df[i,]$AT_Form_Scored_opp <- stat_away_opp[2]

df[i,]$AT_Form_Conceded_opp <- stat_away_opp[3]

df[i,]$AT_shots_form_opp <- stat_away_opp[4]

df[i,]$AT_shots_on_target_form_opp <- stat_away_opp[5]

df[i,]$AT_corners_form_opp <- stat_away_opp[6]

df[i,]$AT_fouls_form_opp <- stat_away_opp[7]

df[i,]$AT_card_points_form_opp <- stat_away_opp[8]


df[i,]$AT_Mean_Points_opp <- stat_away_opp[9]

df[i,]$AT_Mean_Scored_opp <- stat_away_opp[10]

df[i,]$AT_Mean_Conceded_opp <- stat_away_opp[11]

df[i,]$AT_Mean_shots_opp <- stat_away_opp[12]

df[i,]$AT_Mean_shots_on_target_opp <- stat_away_opp[13]

df[i,]$AT_Mean_corners_opp <- stat_away_opp[14]

df[i,]$AT_Mean_fouls_opp<- stat_away_opp[15]
```

```
  df[i,]$AT_Mean_card_points_opp <- stat_away_opp[16]

}




#Convert the dataypes of the columns from list to numeric

d <- df


d[,26:89] <- lapply(df[,26:89], unlist)

d[,26:89] <- lapply(df[,26:89], as.numeric)




#Remove NaN-rows in the data

sum(is.na(d))

d <- na.omit(d)

str(d)


#Round the values

d <- d %>% mutate_if(is.numeric, ~round(.,2))


#Remove unneeded df's

rm(result,stat_away, stat_away_opp, stat_home, stat_home_opp)


###Model Building-------

d_odds <- d


#Remove columns that we don't need for the model
```

```
drop <- c("Date", "HomeTeam", "AwayTeam",   "FTHG",             "FTAG","FTR"    ,"HS"     ,"AS"
           ,"HST","AST","HF","AF","HC","AC","HY","AY","HR","AR",    "H_Odds",    "A_Odds",    "D_Odds",
"Over_2.5", "Under_2.5", "goals" ,"home_team_win", "away_team_win", "draw", "result")
d <- d[,!(names(d) %in% drop)]


d$Is_over_2.5 <- as.factor(d$Is_over_2.5)
levels(d$Is_over_2.5) <- c("Yes", "No")



plot(d$Is_over_2.5)
mean(df$goals)


#Feature Selection----


#identify correlated predictors
cor_mat <- cor(subset(d, select = -Is_over_2.5))
highlyCorrelated <- findCorrelation(cor_mat, cutoff=0.75)


highlyCorrelated <- highlyCorrelated[!highlyCorrelated %in% 1]
names(d[,highlyCorrelated])


d <- d[,-highlyCorrelated]



#Recursive Feature Elimination
set.seed(101)
```

```
rfeC <- rfeControl(functions = rfFuncs, method="cv", number=10)



results <- rfe(d[,2:37], d[,1], sizes=(1:18), rfeControl=rfeC)


print(results)
predictors(results)
plot(results, type=c("g", "o"))


p <- head(results$optVariables, 7)
p <- c("Is_over_2.5", p)


d <- d[,p]



#Train models----


#Setup control function to make rolling forecasts for every matchday (9 games) based on the whole
data before the respective matchday


ctrl <- trainControl(method = "timeslice",
            initialWindow = 306,
            horizon = 9,
            fixedWindow = FALSE,
            skip = 9,
            classProbs = TRUE,
                savePredictions = TRUE,
                summaryFunction = twoClassSummary,
                allowParallel = TRUE)



#Logistic Regression
d.log <-  train(Is_over_2.5 ~ ., data=d, method="glm", family="binomial", metric ="ROC",
trControl=ctrl)
confusionMatrix(d.log$pred$pred, d.log$pred$obs)

#Linear Discriminant Analysis
d.lda <-  train(Is_over_2.5 ~ ., data=d, method="lda", metric ="ROC", trControl=ctrl)
confusionMatrix(d.lda$pred$pred, d.lda$pred$obs)

#K nearest neighbors classification
d.knn <-  train(Is_over_2.5 ~ ., data=d, method="knn", metric ="ROC", trControl=ctrl, tuneLength = 10)
confusionMatrix(d.knn$pred$pred, d.knn$pred$obs)

#SVM
d.svm <- train(Is_over_2.5~., data=d, method="svmRadial", metric="ROC", trControl=ctrl)
confusionMatrix(d.svm$pred$pred, d.svm$pred$obs)

#SVM with grid
svm.grid <- expand.grid(sigma = c(.01, .03, .05), C=c(.25, .75, 1))
d.svmg <- train(Is_over_2.5~., data=d, method="svmRadial", metric="ROC", trControl=ctrl,
tuneGrid=svm.grid)
```

```
confusionMatrix(d.svmg$pred$pred, d.svmg$pred$obs)


#Naive Bayes

d.nb <- train(Is_over_2.5~., data=d, method="nb", metric="ROC", trControl=ctrl)

confusionMatrix(d.nb$pred$pred, d.nb$pred$obs)



#lets compare all resampling approaches

d.models <- list(logit=d.log, lda=d.lda, knn=d.knn, svm = d.svm, svmg = d.svmg, nb = d.nb)

d.resamples = resamples(d.models)


#plot performance comparisons

bwplot(d.resamples, metric="ROC")

bwplot(d.resamples, metric="Sens")

bwplot(d.resamples, metric="Spec")



#calculate ROC curves on resampled data

d.log.roc<- roc(response= d.log$pred$obs, predictor=d.log$pred$Yes)

d.lda.roc<- roc(response= d.lda$pred$obs, predictor=d.lda$pred$Yes)

d.knn.roc<- roc(response= d.knn$pred[d.knn$pred$k==23,]$obs,
predictor=d.knn$pred[d.knn$pred$k==23,]$Yes)

d.svm.roc<- roc(response= d.svm$pred$obs, predictor=d.svm$pred$Yes)

d.svmg.roc<- roc(response= d.svmg$pred$obs, predictor=d.svmg$pred$Yes)

d.nb.roc<- roc(response= d.nb$pred$obs, predictor=d.nb$pred$Yes)
```

```
#build combined ROC plot with resampled ROC curves

plot(d.log.roc, legacy.axes=T, col="Blue", lty=2)

plot(d.lda.roc, add=T, col="Green", lty=2)

plot(d.knn.roc, add=T, col="Red", lty=2)

plot(d.svm.roc, add=T, col="Yellow")

plot(d.svmg.roc, add=T, col="Purple")

plot(d.nb.roc, add=T, col="Pink")

legend(x=1, y=1, legend=c("Logit", "LDA", "KNN", "SVM", "SVMg", "NB"), col=c("blue","green","red",
"yellow", "purple", "pink"),lty=1, cex = 1)


#extract threshold from roc curve  get threshold at coordinates top left most corner

d.log.Thresh<- coords(d.log.roc, x="best", best.method="closest.topleft")

d.log.Thresh

d.lda.Thresh<- coords(d.lda.roc, x="best", best.method="closest.topleft")

d.lda.Thresh

d.knn.Thresh<- coords(d.knn.roc, x="best", best.method="closest.topleft")

d.knn.Thresh

d.svm.Thresh<- coords(d.svm.roc, x="best", best.method="closest.topleft")

d.svm.Thresh

d.svmg.Thresh<- coords(d.svmg.roc, x="best", best.method="closest.topleft")

d.svmg.Thresh

d.nb.Thresh<- coords(d.nb.roc, x="best", best.method="closest.topleft")

d.nb.Thresh


#lets make new predictions with this cut-off and recalculate confusion matrix
```

```
d.log.newpreds <- as.factor(ifelse(d.log$pred$Yes > 0.4242194, "Yes", "No"))

d.lda.newpreds <- as.factor(ifelse(d.lda$pred$Yes > 0.4242657, "Yes", "No"))

d.knn.newpreds <- as.factor(ifelse(d.knn$pred$Yes > 0.4130435, "Yes", "No"))

d.svm.newpreds <- as.factor(ifelse(d.svm$pred$Yes > 0.4309935, "Yes", "No"))

d.svmg.newpreds <- as.factor(ifelse(d.svmg$pred$Yes > 0.4379796, "Yes", "No"))

d.nb.newpreds <- as.factor(ifelse(d.nb$pred$Yes > 0.4557247, "Yes", "No"))


#recalculate confusion matrix with new cut off predictions

confusionMatrix(d.log.newpreds, d.log$pred$obs)

confusionMatrix(d.lda.newpreds, d.lda$pred$obs)

confusionMatrix(d.knn.newpreds, d.knn$pred$obs)

confusionMatrix(d.svm.newpreds, d.svm$pred$obs)

confusionMatrix(d.svmg.newpreds, d.svmg$pred$obs)

confusionMatrix(d.nb.newpreds, d.nb$pred$obs)



#NB is the best

#The overall accuracy dropped from 57% to 55%

#But: Sensitivity increased from 38% to 55%

#Specificity decreased from 70% to 55%

#More balanced predictions


nb.c <- confusionMatrix(d.nb.newpreds, d.nb$pred$obs)


nb.acc <- nb.c$overall["Accuracy"]

nb.sens <- nb.c$byClass["Sensitivity"]
nb.spec <- nb.c$byClass["Specificity"]


#Calculate accuracy of the sports betting provider


d_odds$pred <- as.factor(ifelse(d_odds$Over_2.5 < d_odds$Under_2.5,"1", "0"))

d_odds$Is_over_2.5 <- as.factor(d_odds$Is_over_2.5)

confusionMatrix(d_odds$pred, d_odds$Is_over_2.5)


odds.c <- confusionMatrix(d_odds$pred, d_odds$Is_over_2.5)


odds.acc <- odds.c$overall["Accuracy"]

odds.sens <- odds.c$byClass["Sensitivity"]

odds.spec <- odds.c$byClass["Specificity"]


par(mfrow=c(1,3))

barplot(cbind(nb.acc, odds.acc), main="Accuracy")

barplot(cbind(nb.sens, odds.sens), main="Sensitivity")

barplot(cbind(nb.spec, odds.spec), main="Specificity")



#Our accuracy is lower, but we have a higher TPR. That means we are better in predicting games

#with more than 2.5 goals per game, while the bookmakers are much better in predicting

#low scoring games


#Now we want to test the economic efficiency of our model with the bookmakers odds

#Betting strategy: Always bet on the prediction
```

```
#We bet 1$ hypothetically on every predicted event (Over or under 2.5 goals)
#If the prediction is right, we multiply the specific odd for the outcome with 1$ and add it to our "bank account"
#At the end we see, how much money we lost/won


bet_test <-tail(d_odds, n=306)


preds <- predict(d.nb, bet_test)


bet <- cbind(bet_test[,c("Is_over_2.5", "Over_2.5", "Under_2.5")], preds)
bet$acc <- NA


for (i in 1:nrow(bet)){
 b <- bet[i,]
 if (b$Is_over_2.5 == 1 & b$preds == "Yes"){
  bet[i,]$acc <- b$Over_2.5 -1
  }
 else {
  if (b$Is_over_2.5 == 0 & b$preds == "No"){
   bet[i,]$acc <- b$Under_2.5 -1
  }
  else {
   bet[i,]$acc <- -1
  }
 }
}
```

```
sum(bet$acc)


#We lose $49.23 with our betting strategy over the course of 1 season

#As our TPR is better than the bookmakers, we may have better chances by only betting on

#over 2.5 predictions


sum(bet[bet$preds == "Yes",]$acc)

#We indeed lose way less money than with our first strategy
```