Soccer data set - Predict the winner

Bas Meulendijks

5/27/2019

Contents

1. Executive Summary 1
2. Analysis 2
3. Building models 7
4. Results 11
5. Conclusion 14

1. Executive Summary

As part of the final course in the "HardvardX Professional Certificate in Data Science" a model is built to predict the winner of a soccer match. The "ginf.csv" data set, owned by Alin Secareanu and downloaded from Kaggle is used. After creating a train and test set and data exploration, machine learning algorithms are built and trained, using the inputs in the train set to predict the winner of a soccer match in the test set, using two teams (the home and away team) as predictors. After predicting, the winner is compared with the actual winner, leading to an accuracy rating of the model. Several approaches were used, such as the educated guess, linear regression, K nearest neighbor algorithm and the random forest algorithm. After comparing the results of the approaches, the K nearest neighbor algorithm provided the highest accuracy: **55%**, when using $k = 12$.

2. Analysis

This chapter explains how the soccer data set is downloaded and loaded into R. Next, after mutating the data set, data analysis takes place and a train and test set are created. These sets are used to build models which can predict the winner of a soccer match, given two teams.

2.1 Create the data sets

The following R code installs and loads the required packages. After that, the soccer data set is downloaded and loaded into the variable 'ginf'. The "ginf.csv" data set is owned by Alin Secareanu and was originally downloaded from Kaggle (https://www.kaggle.com/secareanualin/football-events#ginf.csv) and later uploaded to GitHub to make the import in R easier.

```
# install required packages, if required.
if(!require(tidyverse))
  install.packages("tidyverse",   repos = "http://cran.us.r-project.org")
if(!require(caret))
  install.packages("caret",       repos = "http://cran.us.r-project.org")
if(!require(ggplot2))
  install.packages("ggplot2",     repos = "http://cran.us.r-project.org")
if(!require(randomForest))
  install.packages("randomForest",repos = "http://cran.us.r-project.org")
if(!require(rpart))
  install.packages("rpart",       repos = "http://cran.us.r-project.org")

# download the data set and load into a variable
ginf <- tempfile()
download.file("https://github.com/Henkeur10/edx-datascience/raw/master/ginf.csv", ginf)
ginf <- read.csv(ginf)
```

The data below shows the first short analysis on the soccer data set. It shows the number of rows, the column names and the unique values of some of the columns that will be used later.

```
nrow(ginf) # number of rows
```

```
## [1] 10112
```

```
names(ginf) # column names
```

```
##  [1] "id_odsp"   "link_odsp" "adv_stats" "date"      "league"
##  [6] "season"    "country"   "ht"        "at"        "fthg"
## [11] "ftag"      "odd_h"     "odd_d"     "odd_a"     "odd_over"
## [16] "odd_under" "odd_bts"   "odd_bts_n"
```

```
unique(ginf$season) # unique seasons in "ginf"
```

```
## [1] 2012 2013 2014 2015 2016 2017
```

```
unique(ginf$country)# unique countries in "ginf"
```

```
## [1] germany france  england spain   italy
## Levels: england france germany italy spain
```

```
unique(ginf$league) # unique leagues in "ginf"
```

```
## [1] D1  F1  E0  SP1 I1
## Levels: D1 E0 F1 I1 SP1
```

Each row represents a match in one of the several leagues. Per season, every team in a league plays two matches against each other: one at home (column "ht") and one away (column "at"). Note that the rows of the data set do not explicitly show the winner of the match. The columns "fthg" (full time home goals) and "ftag" (full time away goals) provide the final score, which can be used to determine the winner. Using the mutate function in R, columns will be added to list the winner of the match and the final goal difference:

```r
df <- ginf %>%
  mutate(result = factor(ifelse(.$fthg > .$ftag, 1, ifelse(.$fthg < .$ftag, 2, 3))),
         gd = .$fthg - .$ftag)
```

The new column "result" shows the result of a match: number 1 means the home team wins, number 2 means the away team wins and number 3 represents a draw. The new column "gd" shows the goal difference, calculated by subtracting the "ftag" from the "fthg" columns. If the number is a positive integer, the home team wins. If the number is a negative integer, the away team wins. A zero means it is a draw.

Next, the soccer data set is split into two subsets: the train and test set. The train set will contain the data from seasons 2012 through 2016. The test set will contain the data from season 2017. The goal of this paper is to develop an algorithm that will predict the winner of a match. Using the train set, this algorithm will be built and tested against the test set. The following R code splits the soccer data set into the train and test set:

```r
nrow(df) # number of rows
train <- df[df$season != 2017,] # create train set: season 2012-2016
test <- df[df$season == 2017,]  # create test set: season 2017
nrow(train) + nrow(test)        # check if no rows got lost in the process
```

To be able to predict the winner of a match in the 2017 season, the teams in that season should also exist in the train set. This can verified with the following R code:

```r
diff <- setdiff(test$ht, train$ht) # determine which teams are in the test set,
                                   # but not in the train.
length(diff)                       # show the length of the array
```

```
## [1] 5
```

Note that length of the array "diff" is 5, meaning there are 5 teams in the 2017 season (test set) which are not playing in the 2012-2016 seasons (train set). This means those teams were promoted recently and haven't played on the highest level earlier. These teams need to be removed from the test set, since there is no data available to predict the winner:

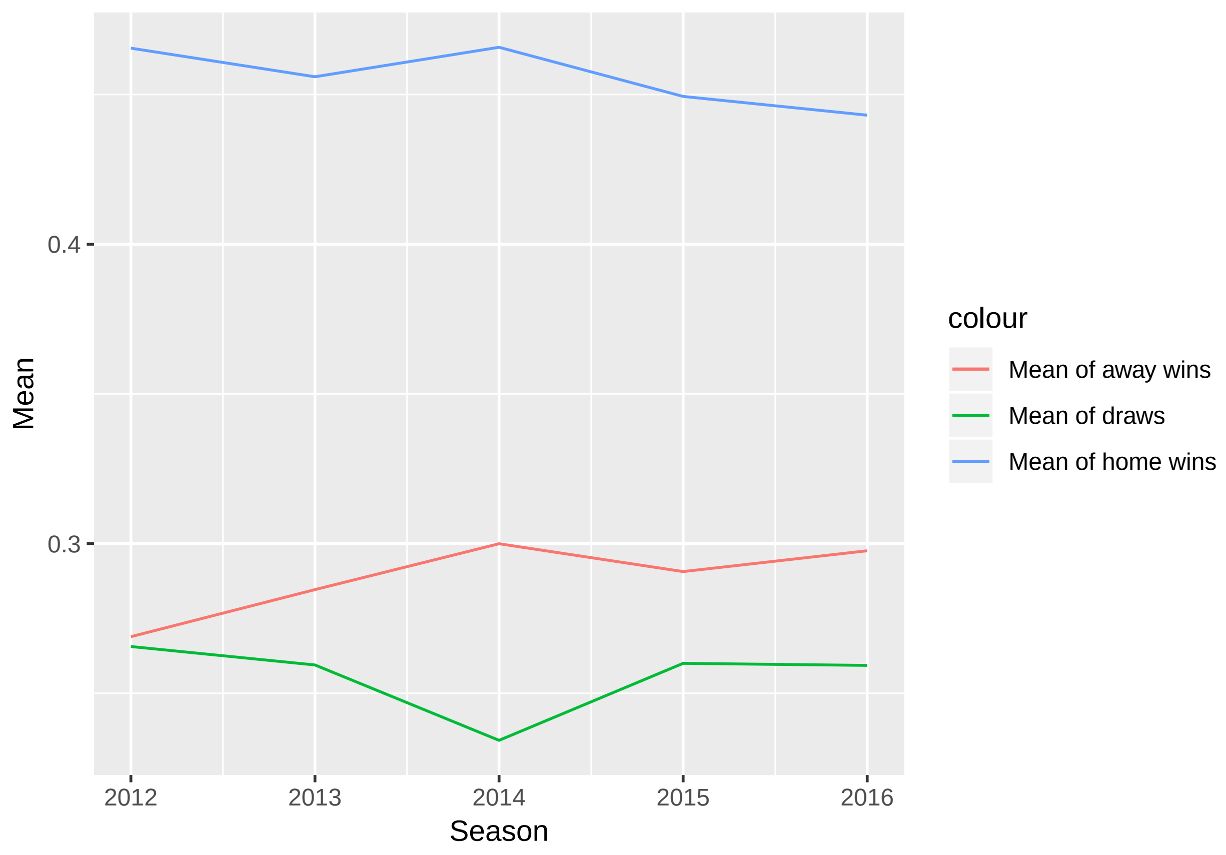
```r
# remove the teams from the test set.
test <- test[!(test$ht %in% diff | test$at %in% diff),]

setdiff(test$ht, train$ht) # double check
setdiff(test$at, train$at) # double check
```

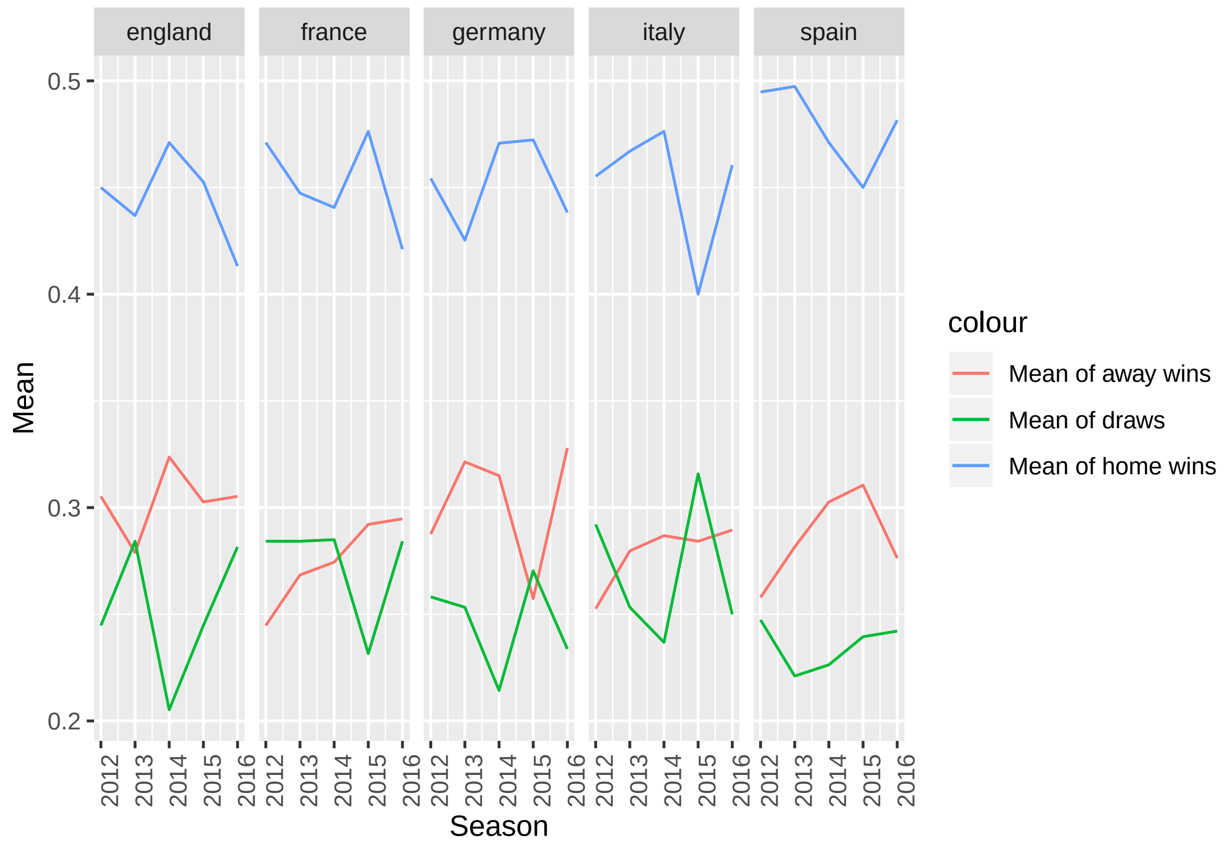
The train and test set are now ready to be further analyzed.

2.2 Explore the data sets

In the previous chapter the soccer data set was mutated by adding the "result" and "gd" columns. Looking more closely to these columns helps understanding the distribution of the data in the train set. The following graph shows the means of the contents of the "result" column in the train set:

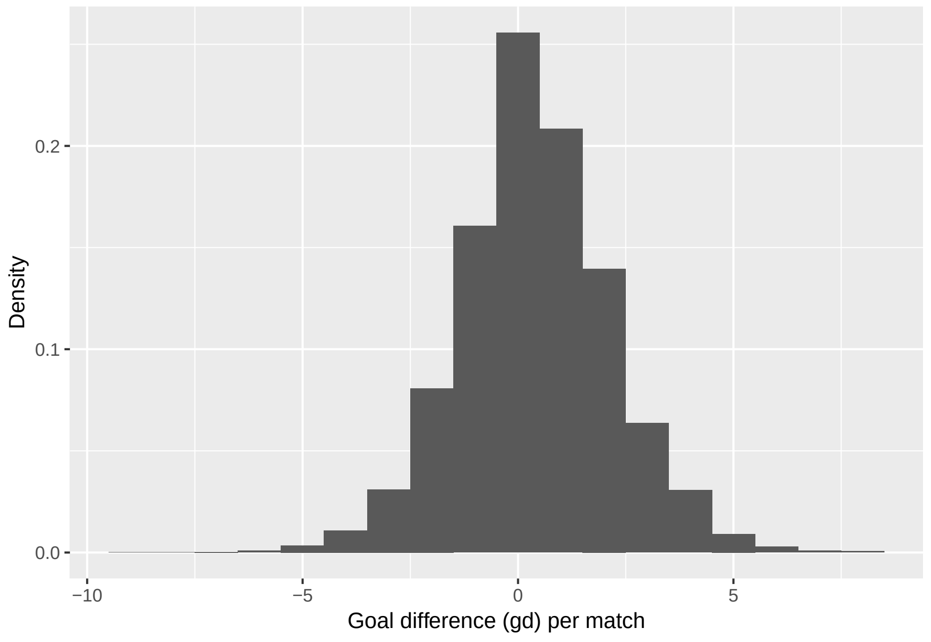


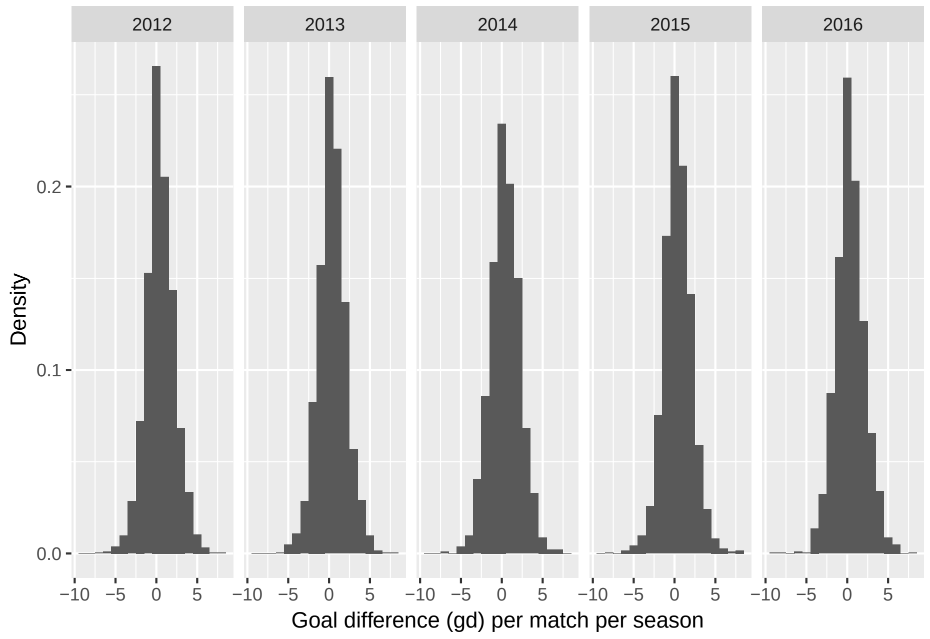
Note there is big difference between the mean of the home wins and the other two means. In approximately 45% of the matches, during the seasons 2012 through 2016, the home team wins. If the home team does not win, it is more likely to lose than to draw. Plotting the means per season, per country, results in the following graph:

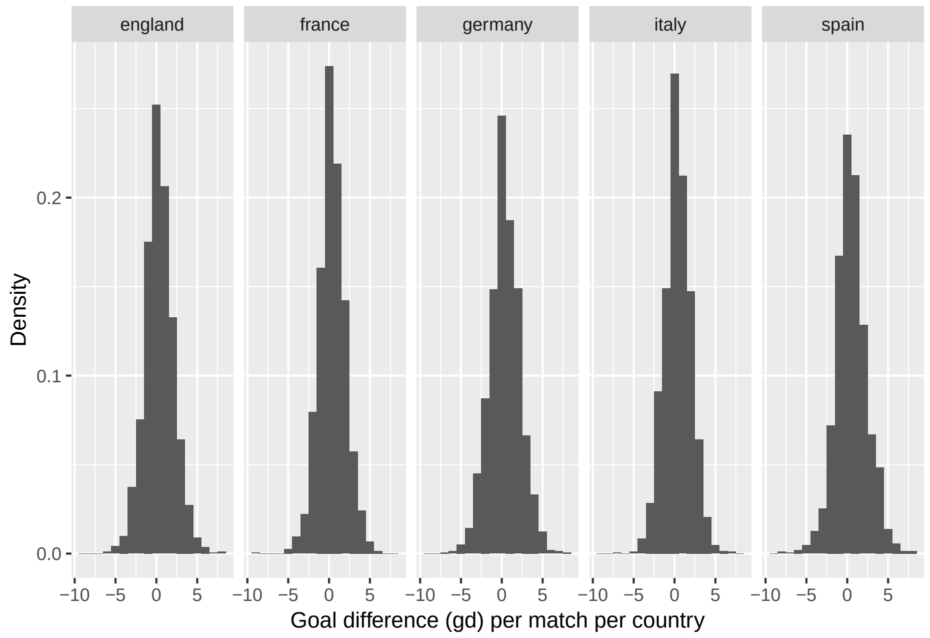


The same trend can be seen in all countries: the home team is more far likely to win than to lose or draw.

The next three graphs show the distribution of the "gd" column. The first shows the density of the goal difference per match for the whole train set. The second shows the density of the goal difference per match per season and the third per country.







The data looks normally distributed. The number zero (meaning the match is a draw) occurs the most. The bars on the right side of the zero are higher than on the left side. This is explained by the fact that a team is more likely to win (meaning the goal difference is positive) than to lose (meaning the goal difference is negative), as seen earlier.

3. Building models

Now that the train and test are created and the data explored, models to predict the winner of a soccer match, using the home and away team as predictors, can be built. This chapter shows several approaches. The results of the approaches will be discussed in the next chapter.

3.1 Educated guess

As shown in the previous chapter, the home team is far more likely to win a soccer match than to lose or draw, regardless of the country. If there were no machine learning capabilities available and one had to 'guess' the results, the educated guess would be to predict the home team as the winning team. In R, predicting a win, loss or a draw for the home team, can be done as follows, with the number 881 representing the number of rows in the test set:

```
y_hat_ht <- factor(replicate(881, 1)) # guessing on a win for every home team

y_hat_at <- factor(replicate(881, 2)) # guessing on a win for every away team

y_hat_dr <- factor(replicate(881, 3)) # guessing on a draw all the time
```

3.2 Linear regression

The next approach is to use linear regression. The following R code uses the linear model to create a fit using the train set and predicts the results using the test set. Because the result column only uses the integers 1, 2 and 3, the round function is used.

```
# set seed to reproduce the same results
set.seed(1)

# create a fit which can predict the result, using the ht (home team) and at
# (away team) as predictors
fit <- lm(result ~ ht + at, train)

# predict the result, using the fit on the test set
y_hat_lm_1 <- predict(fit, test)

# round to nearest integer, since we only use 1,2,3. Convert to factor
y_hat_lm_1 <- factor(round(y_hat_lm_1))
```

```
mean(y_hat_lm_1 == "2")
```

```
## [1] 0.9012486
```

This approach, when using the round function, results in a lot of wins for the away teams (number 2) as shown with the R code above (more than 90%). Based on the data analysis, the home team is more likely to win than the away team. Instead of the round function, in the following approach, the integer value of the predicted number will be used (e.g. 1.6 becomes 1). Therefore, any integer number 0 cannot be used and need to be changed to integer 1.

```
# set seed to reproduce the same results
set.seed(1)

# create a fit which can predict the result, using the ht (home team) and at
# (away team) as predictors
y_hat_lm_2 <- predict(fit, test)

# Insteand of rounding, just grab the integer value.
```

```r
y_hat_lm_2 <- factor(as.integer(y_hat_lm_2))

# change the 0 to 1
y_hat_lm_2[y_hat_lm_2 == "0"] <- "1"

# drop levels that are not used
y_hat_lm_2 <- droplevels(y_hat_lm_2)
```

```r
mean(y_hat_lm_2 == "1")
```

```
## [1] 0.7627696
```

This approach looks more realistic, since the integer number 1 is more present than the number 2. The actual results will be discussed later.

3.3 K nearest neighbor algorithm

The third approach is using the K nearest neighbor (knn) algorithm. The R code below contains two functions. The first function grabs the unique countries in the train set and creates a train subset per unique country. The second function, used within the first function, applies the knn algorithm to the train subsets with different values of k (2 through 20, with steps of 2). After the algorithm has determined a fit and has predicted the results of the matches in the test set (2017 season), the predicted values are compared with the actual values using the confusion matrix. The accuracies are logged within a list and discussed in the next chapter.

```r
set.seed(1)

countries <- unique(train$country) # grab all the unique countries

# use the knn alogrithm per country
accuracy_knn <- map_df(countries, function(c){
  train_knn <- train %>% # create a train set per country
    filter(country == c)

  test_knn <- test %>% # create a test set per country
    filter(country == c)

  # drop all the unused levels
  train_knn$ht <- droplevels(train_knn$ht)
  train_knn$at <- droplevels(train_knn$at)
  test_knn$ht <- droplevels(test_knn$ht)
  test_knn$at <- droplevels(test_knn$at)

  # trick to make sure all levels in train are also in the test set.
  # Otherwise the predict function will give an error
  test_knn <- rbind(train_knn[1, ] , test_knn)
  test_knn <- test_knn[-1,]

  # define an array with the different number of K's that will be used
  ks <- seq(2, 20, 2)

  map_df(ks, function(k){ # per country, use the knn3 algorithm
    # create a fit which can predict the result, using the ht (home team)
    # and at (away team) as predictors
    fit_knn <- knn3(result ~ ht + at, train, k = k)
    # predic the result, using the fit on the test set
    y_hat_knn <- predict(fit_knn, newdata = test, type = "class")
    # determine the accuracy by comparing the predicted value with the actual value
    cM <- confusionMatrix(data = y_hat_knn,
                          reference = test$result)$overall["Accuracy"]

    # list the results
    list(country=c, accuracy=cM, k = k)
  })
})
```

3.4 Random forest algorithm

The final approach makes use of the random forest algorithm. As with the previous approach, two functions are used. The first grabs the unique countries in the train set and creates a separate subset for each of them. The second function, used within the first, applies the algorithm to the train subset. Different values of the number of trees are used. After the fits and predictions are created, the predicted result is compared with the actual result, leading to an accuracy value. These values are logged in a list and discussed later.

```r
set.seed(1)

countries <- unique(train$country) # grab all the unique countries

# use the knn alogrithm per country
accuracy_rf <- map_df(countries, function(c){
  train_rf <- train %>% # create a train set per country
  filter(country == c)

  test_rf <- test %>% # create a test set per country
  filter(country == c)

  # drop all the unused levels
  train_rf$ht <- droplevels(train_rf$ht)
  train_rf$at <- droplevels(train_rf$at)
  test_rf$ht <- droplevels(test_rf$ht)
  test_rf$at <- droplevels(test_rf$at)

  # trick to make sure all levels in train are also in the test set.
  # Otherwise the predict function will give an error
  test_rf <- rbind(train_rf[1, ] , test_rf)
  test_rf <- test_rf[-1,]

  # define an array with the different number of trees that will be used
  number_trees <- seq(10, 100, 10)

  map_df(number_trees, function(n){
    # create a fit which can predict the result, using the ht (home team)
    # and at (away team) as predictors
    fit_rf <- randomForest(result ~ ht + at, train_rf, ntree = n)
    # predic the result, using the fit on the test set
    y_hat_rf <- predict(fit_rf, test_rf)
    # determine the accuracy by comparing the predicted value with the actual value
    cM <- confusionMatrix(data = y_hat_rf,
                          reference = test_rf$result)$overall["Accuracy"]
    # list the results
    list(country=c, accuracy=cM, n = n)
  })
})
```

4. Results

In the previous chapter several approaches were built to predict the winner of a soccer for the 2017 season (test set) based on the 2012 through 2016 season results (train set). This chapter shows and discusses the results of the approaches.

4.1 Educated guess

The following R code compares the actual results with the predicted ones, using the confusion matrix:

```r
y <- test$result # the actual results of the match in the 2017 season

# Educated guess
confusionMatrix(y_hat_ht, y)$overall["Accuracy"] # all home teams win
```

```
##  Accuracy
## 0.5051078
```

```r
confusionMatrix(y_hat_at, y)$overall["Accuracy"] # all away teams win
```

```
##  Accuracy
## 0.2576617
```

```r
confusionMatrix(y_hat_dr, y)$overall["Accuracy"] # only draws
```

```
##  Accuracy
## 0.2372304
```

Just ‘guessing’ on a win for the home team on every match results in an accuracy of little over 50%. This is not surprising, since the data analysis showed the chance of a win for the home was was between 40% and 50% during the 2012-2016 seasons. As expected, ‘guessing’ on a loss or draw for the home team does not lead to a high accuracy.

4.2 Linear regression

Comparing the actual results of the matches with the ones predicted using the linear models, leads to the following accuracies:

```r
# Lineair regression
confusionMatrix(y_hat_lm_1, y)$overall["Accuracy"] # with rounding
```

```
## Accuracy
## 0.322361
```

```r
confusionMatrix(y_hat_lm_2, y)$overall["Accuracy"] # with integer value
```

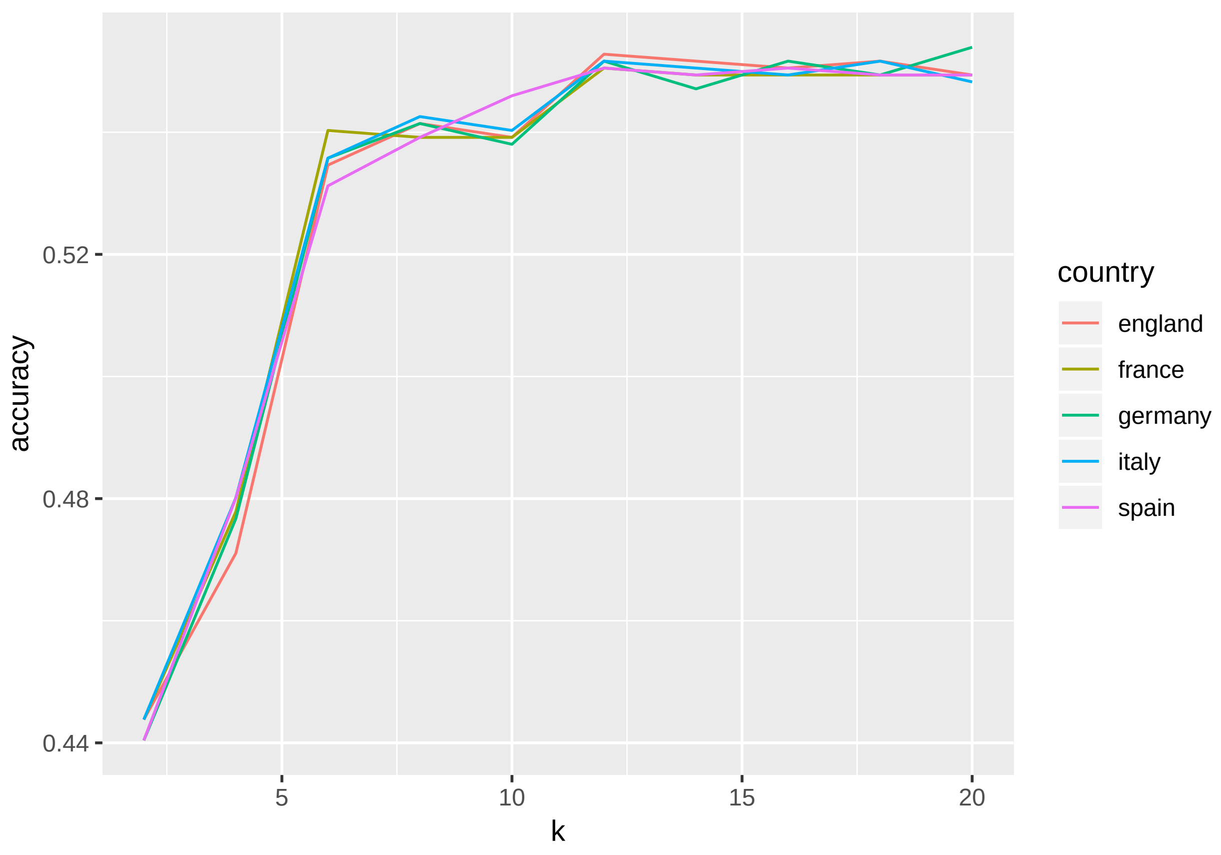
```
##  Accuracy
## 0.5255392
```

The first approach, which makes use of the round function, has a very low accuracy. This was expected, as explained earlier, due to the fact that rounding leads to a lot of integer 2 results, while integer 1 are more likely to occur.

The second approach, which takes the integer value of the predicted number, results in a higher accuracy than the first approach and an even higher accuracy than the educated guess method.

4.3 K nearest neighbor algorithm

The plot below visualizes the accuracy per different value of k per country. Starting with a low accuracy, the graphs rise quickly and reach an accuracy over 50%. This approach has the same effect on all countries, since all the graphs follow the same pattern.



Calculating the mean of the accuracy per k with the R code below and filtering out the maximum value, leads to the following accuracy:

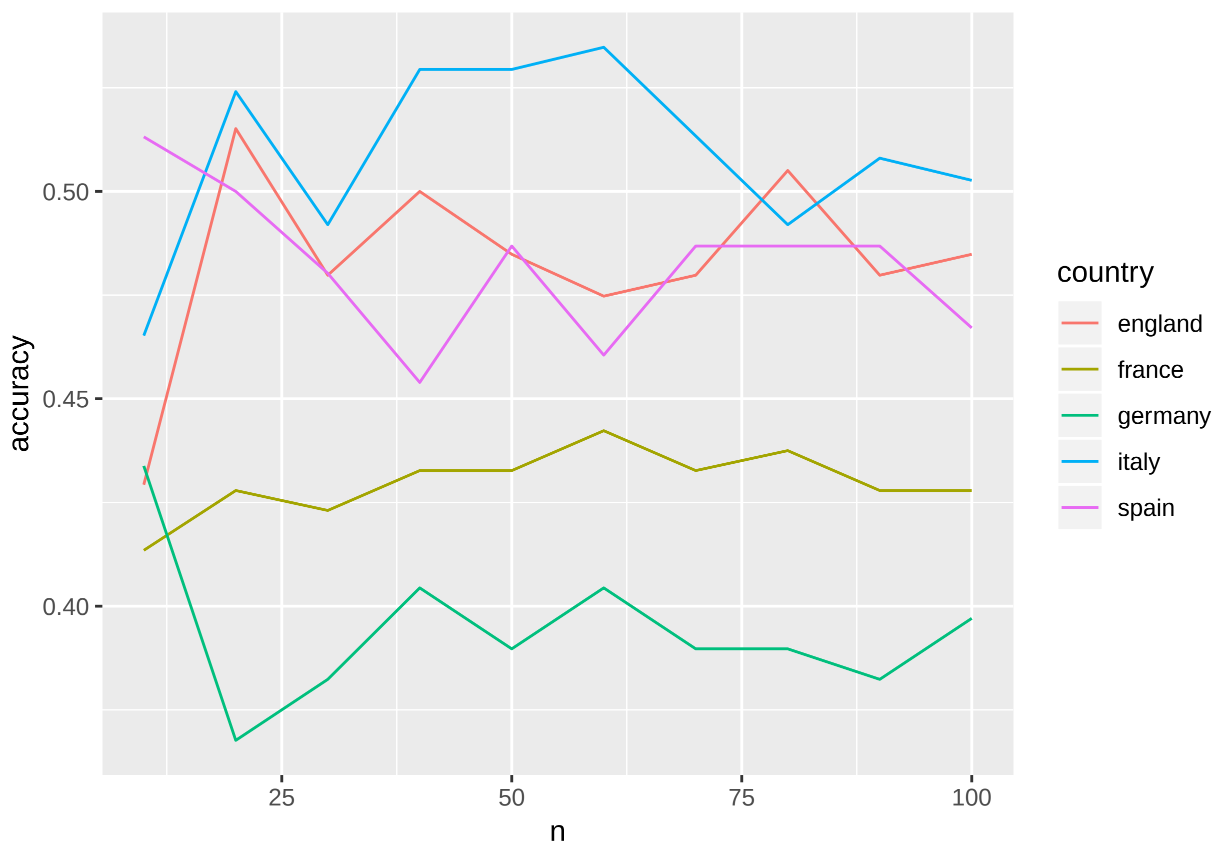
```r
accuracy_knn %>%
  group_by(k) %>%
  summarise(mean = mean(accuracy)) %>%
  filter(mean == max(mean))
```

```
## # A tibble: 1 x 2
##       k  mean
##   <dbl> <dbl>
## 1    12 0.551
```

With $k = 12$, an accuracy of 55% is obtained, making the knn approach the best so far.

4.4 Random forest algorithm

The plot below visualizes the accuracy per different value of n per country. In contrast to the k nearest neighbor algorithm, the random forest algorithm is not performing well on all countries. Where the different graphs per country in the knn plot were grouped together, the graphs in the random forest plot are spread out. For example, the graph for Italy stays mostly above the 50% accuracy line, while the graph for Germany mostly stays under the 40% accuracy line.



Calculating the mean of the accuracy per n with the R code below and filtering out the maximum value, leads to the following accuracy:

```
accuracy_rf %>%
  group_by(n) %>%
  summarise(mean = mean(accuracy)) %>%
  filter(mean == max(mean))
```

```
## # A tibble: 1 x 2
##       n  mean
##   <dbl> <dbl>
## 1    20 0.467
```

With $n = 20$, an accuracy of 47% is obtained, which makes this approach less suitable for predicting results of soccer matches.

5. Conclusion

The goal of this paper was to build a model which can predict results of soccer matches by predicting a win, loss or draw for the home team, using the home and away team as predictors. Several approaches were used: the educated guess, linear regression, the k nearest neighbor algorithm and the random forest algorithm. Comparing the outcomes of the several approaches leads to the conclusion that the K nearest neighbor algorithm is the best approach. Using $k = 12$ leads to an accuracy of **55%**.

Although the outcome is not as high as hoped, still more than half of the matches were predicted successfully. The recommendation is to further investigate the K nearest neighbor algorithm approach for predicting soccer matches and look more into other approaches.