Predicting the Winner in Chess

Jojoha

05/01/2021

Introduction

Here is the kaggle link to obtain the dataset: https://www.kaggle.com/datasnaek/chess

Chess is arguably one of the most known games of all time in human history, still being played to this day by millions worldwide both in real life events and in online sites like lichess, chess and chess24, you can ask anyone if they knew about the game and they would probably say yes.

I selected this dataset because i thought this was a fun challenge to do, since i also love playing chess!

This document will try to predict which side (white or black) is more likely to win based on their various features that are available in the dataset (rating, opening, moves and much more).

The dataset is based on 20,000 different games collected on the online site lichess.org

Setup

This section is for importing the necessary packages and also the dataset and then preparing it for further work.

Importing the dataset

```
if(!require(readr)) install.packages("readr")
```

```
## Loading required package: readr
```

```
library(readr)


# Downloading the dataset from my github, since downloading it from kaggle requires an account or you w
myfile <- 'https://raw.githubusercontent.com/jojoha1337/EDX-Chess-Predictions/main/games.csv'

dat<-read_csv(url(myfile))
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   id = col_character(),
##   rated = col_logical(),
##   created_at = col_double(),
##   last_move_at = col_double(),
```

```
##   turns = col_double(),
##   victory_status = col_character(),
##   winner = col_character(),
##   increment_code = col_character(),
##   white_id = col_character(),
##   white_rating = col_double(),
##   black_id = col_character(),
##   black_rating = col_double(),
##   moves = col_character(),
##   opening_eco = col_character(),
##   opening_name = col_character(),
##   opening_ply = col_double()
## )
```

```
# ranger is what we will be using to generate our models
if (!require('ranger')) install.packages('ranger'); library('ranger')
```

```
## Loading required package: ranger
```

```
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tidyverse
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.3.2     v dplyr   1.0.2
## v tibble  3.0.4     v stringr 1.4.0
## v tidyr   1.1.2     v forcats 0.5.0
## v purrr   0.3.4
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: caret
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(tidyverse)
library(caret)
```

Dataset Summary

```
names(dat)
```

```
##  [1] "id"             "rated"          "created_at"     "last_move_at"
##  [5] "turns"          "victory_status" "winner"         "increment_code"
##  [9] "white_id"       "white_rating"   "black_id"       "black_rating"
## [13] "moves"          "opening_eco"    "opening_name"   "opening_ply"
```

Explanations for each individual column:

id: The id of the game played on lichess, you can view the entire game by typing lichess.org/id on your browser

rated: If the game was worth rating points or not, generally we can assume that rated games will be played more seriously by players when their rating is at stake.

created_at: Time that the game started

last_move_at: Time that the game ended

turns: How many turns the game took

victory_status: The method that the winning side won (checkmate, out of time or opponent resignation)

winner: The color of the side that won (White or Black)

increment_code: The timemode that the game was played (bullet, blitz, classical and more) we can expect that games with shorter timemodes will end much sooner than a classical mode since blunders are more likely.

white_id and black_id: Name of the player with white or black pieces

white_rating and black_rating: Rating of the player with white or black pieces

moves: The move order of the game in standard chess notation

opening_eco: Id of the opening based on the Encyclopedia of Chess Openings

opening_name: Name of the opening

opening_ply: Number of moves in the opening phase of the game

Analysis

```
# viewing the shape of the dataset
dim(dat)
```

```
## [1] 20058    16
```

The size of our data is relatively small, but it makes it so that its easier to train models on.

Since we will only be predicting if a side is going to either win or lose, we shall remove all games that were draws.

```
dat <- dat[dat$winner != 'draw',]
```

Lets view the overall winrates on our dataset

```
table(dat$winner) / nrow(dat)
```

```
##
##     black     white
## 0.4766067 0.5233933
```

How about winrates for each opening name?

```
 opening_name_winrates <- dat %>% group_by(opening_name) %>%
  summarise(white = mean(winner == 'white'),
            black = mean(winner == 'black'))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
opening_name_winrates
```

```
## # A tibble: 1,453 x 3
##    opening_name                                                white black
##    <chr>                                                       <dbl> <dbl>
##  1 Alekhine Defense                                            0.523 0.477
##  2 Alekhine Defense #2                                         0.5   0.5
##  3 Alekhine Defense #3                                         0     1
##  4 Alekhine Defense: Balogh Variation                          0     1
##  5 Alekhine Defense: Brooklyn Variation                        1     0
##  6 Alekhine Defense: Exchange Variation                        0.75  0.25
##  7 Alekhine Defense: Four Pawns Attack                         0.5   0.5
##  8 Alekhine Defense: Four Pawns Attack |  6...Nc6              1     0
##  9 Alekhine Defense: Four Pawns Attack |  Fianchetto Variation 1     0
## 10 Alekhine Defense: Four Pawns Attack |  Main Line            1     0
## # ... with 1,443 more rows
```

Also viewing the winrates for the opening_ecos

```
opening_eco_winrates <- dat %>% group_by(opening_eco) %>%
  summarise(white = mean(winner == 'white'),
            black = mean(winner == 'black'))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
opening_eco_winrates
```

```
## # A tibble: 359 x 3
##    opening_eco white black
##    <chr>       <dbl> <dbl>
##  1 A00         0.411 0.589
##  2 A01         0.487 0.513
##  3 A02         0.416 0.584
##  4 A03         0.441 0.559
##  5 A04         0.607 0.393
##  6 A05         0.692 0.308
```

```
##  7 A06         0.504 0.496
##  8 A07         0.630 0.370
##  9 A08         0.545 0.455
## 10 A09         0.529 0.471
## # ... with 349 more rows
```

When comparing opening_eco and opening_names, it is clear that opening_eco offers more detailed winrates, while opening_names has some openings where one side has a 100% winrate, this tells us that there are few played games of that opening. opening_eco offers us a more "smoothed" out view of our winrates, while opening_names is more detailed.

Lets try predicting the Winner just by guessing the side that has the highest winrate for both opening_names and opening_eco and see their accuracies.

```
# Splitting the data into training and testing sets
ind <- createDataPartition(dat$winner, times = 1, p = 0.2, list = FALSE)

train_set <- dat[-ind,]
test_set <- dat[ind,]
```

```
## Warning: The ‘i‘ argument of ‘‘[‘()‘ can’t be a matrix as of tibble 3.0.0.
## Convert to a vector.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_warnings()‘ to see where this warning was generated.
```

```
# Explanation: If white's winrate is higher than black's, then predict white, else predict black
winrates_names <- ifelse(opening_name_winrates['white'] > opening_name_winrates['black'], 'white', 'bla

winrates_eco <- ifelse(opening_eco_winrates['white'] > opening_eco_winrates['black'], 'white', 'black')
```

Lets assign those values to 2 dataframes and see their results

```
# Creating the dataframes
wr_name_table <- data.frame(opening_name = opening_name_winrates$opening_name, winrate = winrates_names)

wr_eco_table <- data.frame(opening_eco = opening_eco_winrates$opening_eco, winrate = winrates_eco)

# renaming the second column, for some reason it is named "white", renaming it to something else.
names(wr_name_table)[names(wr_name_table) == "white"] <- "Side with Highest Winrate"
names(wr_eco_table)[names(wr_eco_table) == "white"] <- "Side with Highest Winrate"
```

Examples for interpretation of the above table:

White has a higher winrate than black in the Alekhine Defense opening Black has a higher winrate than white in the Alekhine Defense #2 opening

Time to create the predictions based on those dataframes

```
# Explanation: For each row in test set, look at it's opening_name/eco column, if the opening_name/eco
# then return the element of wr_name/eco_table$‘Side with Highest Winrate‘ else return an error (to war

predictions <- ifelse(test_set$opening_name %in% wr_name_table$opening_name, wr_name_table$‘Side with Hi
predictions_eco <- ifelse(test_set$opening_eco %in% wr_eco_table$opening_eco, wr_eco_table$‘Side with Hi
```

Creating a table to store the reported accuracies

```
results <- tibble()
results <- bind_rows(results,
                     tibble(Method = 'Guessing by opening_name winrates',
                            Accuracy = mean(predictions == test_set$winner)))
results <- bind_rows(results,
                     tibble(Method = 'Guessing by opening_eco winrates',
                            Accuracy = mean(predictions_eco == test_set$winner)))

results
```

```
## # A tibble: 2 x 2
##   Method                              Accuracy
##   <chr>                                  <dbl>
## 1 Guessing by opening_name winrates      0.495
## 2 Guessing by opening_eco winrates       0.501
```

Both predictions return a similar accuracy of ~50% using the opening win rates method. Lets see if we can do better with another method

Creating a new feature, The Rating Difference between players, we will try guessing the winner by predicting the side with the highest rating, if the rating is a tie, we will just do a random 50/50 guess

```
# creating the difference between white and black rating
test_rating_diffs <- test_set$white_rating - test_set$black_rating

# creating the predictions
# If the rating difference is 0, then do a 50/50 guess, if the rating difference is positive, then pred
preds <- ifelse(test_rating_diffs == 0, sample(c('white', 'black'),1, replace = TRUE, prob = c(0.5,0.5))
       ifelse(test_rating_diffs > 0, 'white', 'black'))

# appending the results to the accuracy table
results <- bind_rows(results,
                     tibble(Method = 'Guessing by rating difference',
                            Accuracy = mean(preds == test_set$winner)))
```

Now this is better, we observe an accuracy of ~65% just by using the rating difference

```
results
```

```
## # A tibble: 3 x 2
##   Method                              Accuracy
##   <chr>                                  <dbl>
## 1 Guessing by opening_name winrates      0.495
## 2 Guessing by opening_eco winrates       0.501
## 3 Guessing by rating difference          0.656
```

Random Forest Model

Now lets try using a more advanced model, a Random Forest model.

First, lets examine the data's columns

We can observe that something is wrong when we try to create a Total Game Time variable, some games have a Total Game Time of 0.

```
# creating the rating diff variable and viewing the selected columns
dat %>% mutate(total_game_time = dat$last_move_at - dat$created_at) %>%
  select(total_game_time, winner, increment_code, turns, moves)
```

```
## # A tibble: 19,108 x 5
##    total_game_time winner increment_code turns moves
##              <dbl> <chr>  <chr>          <dbl> <chr>
##  1               0 white  15+2              13 d4 d5 c4 c6 cxd5 e6 dxe6 fxe6 Nf~
##  2               0 black  5+10              16 d4 Nc6 e4 e5 f4 f6 dxe5 fxe5 fxe~
##  3               0 white  5+10              61 e4 e5 d3 d6 Be3 c6 Be2 b5 Nd2 a5~
##  4               0 white  20+0              61 d4 d5 Nf3 Bf5 Nc3 Nf6 Bf4 Ng4 e3~
##  5               0 white  30+3              95 e4 e5 Nf3 d6 d4 Nc6 d5 Nb4 a3 Na~
##  6               0 white  10+0              33 d4 d5 e4 dxe4 Nc3 Nf6 f3 exf3 Nx~
##  7               0 black  15+30              9 e4 Nc6 d4 e5 d5 Nce7 c3 Ng6 b4
##  8               0 black  15+0              66 e4 e5 Bc4 Nc6 Nf3 Nd4 d3 Nxf3+ Q~
##  9               0 white  10+0             119 e4 d5 exd5 Qxd5 Nc3 Qe5+ Be2 Na6~
## 10               0 white  20+60             39 e3 e6 d4 d6 Bd3 c6 Nf3 Be7 Nc3 N~
## # ... with 19,098 more rows
```

But we can also observe that the matches still have their turns and moves, unfortunately this means that we cannot trust these 'time' based columns, we will have to remove them.

Also removing the id of the game and both of the player id's, those are clearly not useful for prediction and removing the 'moves' column, because viewing how the game ended by looking at the moves is not really how we want our model to predict. And removing the increment_code, opening_eco and opening name columns, for some reason, when attempting a random forest with these variables present, the model takes way too long to train.

```
dat <- subset(dat, select = -c(created_at, last_move_at, id, white_id, black_id, moves,increment_code, o
```

```
# adding the rating_difference variable, because it showed that it was a powerful predictor on the Anal
dat <- dat %>% mutate(rating_difference = dat$white_rating - dat$black_rating)
```

Splitting the data into three different parts, the train_set, test_set and the validation set

```
# Splitting the data into training and testing sets
ind <- createDataPartition(dat$winner, times = 1, p = 0.3, list = FALSE)

train_set <- dat[-ind,]
temp <- dat[ind,]

ind_2 <- createDataPartition(temp$winner, times = 1, p = 0.5, list = FALSE)

test_set <- temp[-ind_2,]
validation <- temp[ind_2,]
```

We will use the train_set and test_set to evaluate and tune our model and then use our final model on the validation set. Lets try a decision tree first.

```
#  Creating the Decision Tree model and binding it's score to the results table
dtreet <- train(winner ~ ., data = train_set, method = 'rpart')

preds <- predict(dtreet, test_set)

results <- bind_rows(results,
                     tibble(Method = 'Decision Tree',
                            Accuracy = mean(preds == test_set$winner)))
```

Now lets do a Random Forest with the ranger library

```
#  Creating the Decision Tree model and binding it's score to the results table
rf <- train(winner ~ ., data = train_set, method = 'ranger')

preds <- predict(rf, test_set)

results <- bind_rows(results,
                     tibble(Method = 'Random Forest',
                            Accuracy = mean(preds == test_set$winner)))
```

Results and Conclusion

```
results
```

```
## # A tibble: 5 x 2
##   Method                              Accuracy
##   <chr>                                  <dbl>
## 1 Guessing by opening_name winrates      0.495
## 2 Guessing by opening_eco winrates       0.501
## 3 Guessing by rating difference          0.656
## 4 Decision Tree                          0.680
## 5 Random Forest                          0.770
```

An accuracy of 76% was obtained by using a simple Random Forest and the perfomance of the model (execution timewise) was relatively quick.

Performing some parameter tuning may improve the accuracy even further and figuring out a way to include some of the removed columns may also prove helpful in that endeavor, however due to time constraints, i personally am unable to test these solutions out.

Also, discussing some of the limitations of this model: This model predicts which side may win, however it does have access to 2 clues: the amount of turns a game took and the way that victory was obtained (mate, resignation etc)

Thanks for reading!