Lab Notebook

Daniel Gardner, Oliver Baker, Grace Yan

2024-01-15

Contents


```
library(dplyr)
library(tidyr)
library(ggplot2)
library(readr)
library(BradleyTerry2)
library(TennisBT)
library(ggthemes)
library(cowplot)
library(tidyverse)
library(scales)
```

Code for Section 2: Data

Here we will be performing some basic analysis on the data to get an idea of any major problems with the dataset before we build the model. I.e. if we find any irrelevant data that we can remove this will greatly simplify things later down the line.

Missing Data

First we want to see how much data is actually available. From briefly skimming through the datasets we saw that not all years contain the same amount of variables. Therefore we quantify the amount of variables by measuring how many columns in the data frame actually contain data. Then we will also potentially use age and height as covariates in our model, although again by quickly looking at the .csv files we can see a fair amount of missing data. Hence we also measure this and see if it changes by year.

```
#Checking if a column is empty
containsdata<-function(column){
  if (all(is.na(column))){
    return(FALSE)
  }
  else{
    return(TRUE)
  }
}

#How many columns contain data
fullcolumns<-function(data){
  num<-0
  for (i in 1:dim(data)[2]){
    if (containsdata(data[,i])){
      num<-num+1
    }
  }
  return(num)
}
```

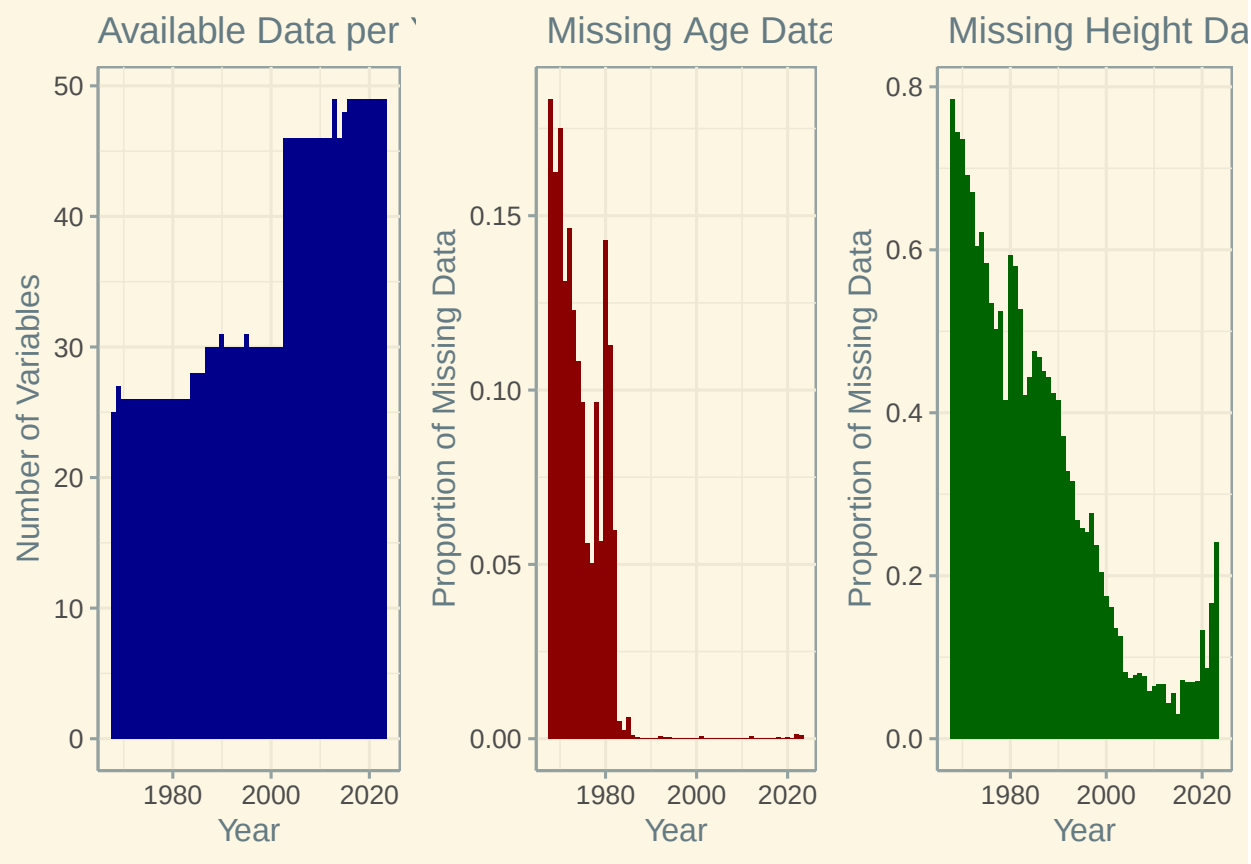
```
#Checking for missing data in age
ismissing_age<-function(data){
  num_missing<-0
  for (column in c('winner_age','loser_age')){
    for (j in 1:dim(data[,column])[1]){
      if (all(is.na(data[j,column]))){
        num_missing<-num_missing+1
      }
    }
  }
  return(num_missing)
}

#Checking for missing data in height
ismissing_height<-function(data){
  num_missing<-0
  for (column in c('winner_ht','loser_ht')){
    for (j in 1:dim(data[,column])[1]){
      if (all(is.na(data[j,column]))){
        num_missing<-num_missing+1
      }
    }
  }
  return(num_missing)
}
```

NOTE: If you wish to run this code yourself, it requires the full Tennis WTA Dataset from https://github.com/JeffSackmann/tennis_wta. Please download this file into the 'data' folder of our project to run.

```
#Finding amount of columns, missing height and age data for each year

#List of years from 1968 to 2023
years<-seq(1968,2023)

#Repeating for all years
M<-length(years)
fullcols<-rep(0,M)
mis_ages<-rep(0,M)
mis_hts<-rep(0,M)
for (i in 1:M){
  data<-read_csv(paste("data/tennis_wta/wta_matches_",as.character(years[i])
                       ,".csv", sep=""),show_col_types = FALSE)
  fullcols[i]<-fullcolumns(data)
  mis_ages[i]<-ismissing_age(data)/(2*dim(data)[1])
  mis_hts[i]<-ismissing_height(data)/(2*dim(data)[1])
}
```

By plotting the amount of variables (non-zero columns) and percentage of missing data across time, we can see that the older data is much worse than the current, which is to be expected. For instance, the data around 1970 only has about half the variables recorded as in 2023, and over half the heights of players are not recorded. We can also see that for modern data, the proportion of missing age data is negligible. As expected also the amount of height data collected increases with time, although strangely falters past the 2010s.

```
#PLOTTING RESULTS

#Creating data frame of results
Missing_Data<-data_frame(Year=years,Mis_Age =mis_ages,Mis_hts=mis_hts
                         ,Fullcols=fullcols)

#Plotting
plot1<-ggplot(Missing_Data,aes(x=Year,y=Fullcols))+
  geom_bar(stat="identity",fill='darkblue') +
  theme_solarized() +
  labs(x='Year',y='Number of Variables',title='Available Data per Year')
plot2<-ggplot(Missing_Data,aes(x=Year,y=Mis_Age))+
  geom_bar(stat="identity",fill='darkred') +
  theme_solarized() +
  labs(x='Year',y='Proportion of Missing Data'
       ,title=' Missing Age Data per Year')
plot3<-ggplot(Missing_Data,aes(x=Year,y=Mis_hts))+
  geom_bar(stat="identity",fill='darkgreen') +
  theme_solarized() +
  labs(x='Year',y='Proportion of Missing Data'
       ,title=' Missing Height Data per Year')

plot_grid(plot1,plot2,plot3,nrow=1)
```



Games Played per Player

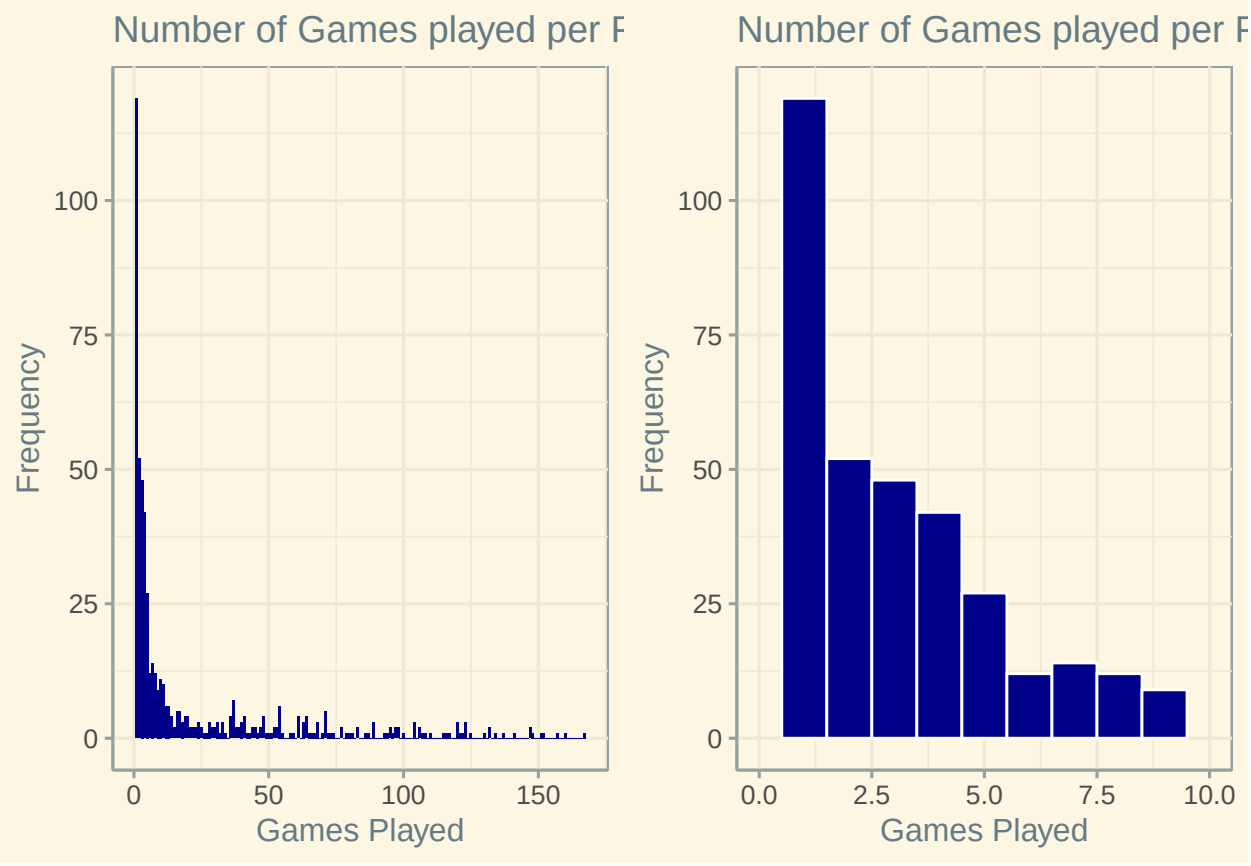
Now we have justification for sticking mostly to modern data, we can start to think about which players are relevant in our model. We suspect that not all players need to necessarily be included. For instance, a rogue player that has only played one game will input very little into our model's predictive power. We can quantify this suspicion by plotting each players total number of games played from 2020-2022.

```
#Creating array of all distinct competitors from 2020-2022
num_names<-c()
for (i in 53:(M-1)){
  data<-read_csv(paste("data/tennis_wta/wta_matches_",as.character(years[i])
                       ,".csv", sep=""),show_col_types = FALSE)
  num_names<-append(num_names,c(as.matrix(data['winner_name'])
                                ,as.matrix(data['loser_name'])))
}
names<-unique(num_names)
```

By plotting a histogram of frequencies (Left), it is immediately obvious that our worries were correct, and a large proportion of players have played an insignificant amount of games over three years. In fact by zooming in on this histogram (right), a lot of players have indeed only played one game and then never played again.

```
#Finding how many games each player has played from 2020-2022
N<-length(names)
num_games<-rep(0,N)
```

```r
for (i in 1:N){
  num_games[i]<-length(which(num_names==names[i]))
}

#Creating data frame of results
Num_Game_Data<-data_frame(Num_Games=num_games)

#Plotting
plot4<-ggplot(Num_Game_Data,aes(x=Num_Games))+
  theme_solarized() +
  geom_histogram(binwidth = 1,fill='darkblue') +
  labs(x='Games Played',y='Frequency',title='Number of Games played per Player (2020-2022)')

plot5<-ggplot(Num_Game_Data,aes(x=Num_Games))+
  theme_solarized() +
  geom_histogram(binwidth = 1,fill='darkblue',col='white') +
  xlim(c(0,10)) +
  labs(x='Games Played',y='Frequency',title='Number of Games played per Player (2020-2022) [Zoomed]')

plot_grid(plot4,plot5,nrow=1)
```



Code for Section 3: Model and Evaluation Techniques

Motivating the Use of a Surface Parameter

This section provides the code to score the BT model on each of the different surfaces, and demonstrate why we need to consider the surface the match was played on.

In the TennisBT package, we have three files, grass_data, clay_data, and hard_data containing binomial win data for each player pairing in the training set on each surface.

```
nrow(grass_data)
```

```
## [1] 366
```

```
nrow(clay_data)
```

```
## [1] 1062
```

```
nrow(hard_data)
```

```
## [1] 2384
```

We can see that the vast majority of matches are played on a hard court, few matches are played on grass. In fact, only 7 of the 56 WTA tournaments are plated on grass in each calendar year. With this implying low data availability, we expect a larger degree of error in the grass data.

```
grass_btm <- BTm(cbind(win1,win2), factor(player1), factor(player2)
                 , data=grass_data)
hard_btm <- BTm(cbind(win1,win2), factor(player1), factor(player2)
                , data=hard_data)
clay_btm <- BTm(cbind(win1,win2), factor(player1), factor(player2)
                , data=clay_data)
```

Now we will rank the top 10 players on each surface, and show the differences between them. First, in the grass court table we see some very high standard errors, due to the low number of matches played on grass.

```
grass_df <- as.data.frame(BTabilities(grass_btm))
grass_df <- grass_df[order(-grass_df$ability),]
head(grass_df,10)
```

```
##                        ability         s.e.
## Ashleigh Barty      18.05488851 2558.3938610
## Johanna Konta       16.53285405 2563.6558748
## Beatriz Haddad Maia  0.99487136    1.1181628
## Ons Jabeur           0.75174641    0.9807728
## Simona Halep         0.60924123    1.0207950
## Jelena Ostapenko     0.39005199    0.8388992
## Elena Rybakina       0.17614059    0.8809571
## Angelique Kerber     0.14891353    1.0132818
## Ajla Tomljanovic     0.00000000    0.0000000
## Tatjana Maria       -0.04024822    1.0752890
```

```r
hard_df <- as.data.frame(BTabilities(hard_btm))
hard_df <- hard_df[order(-hard_df$ability),]
head(hard_df,10)
```

```
##                   ability      s.e.
## Ashleigh Barty   2.783247 0.6192415
## Iga Swiatek      1.950132 0.4036822
## Serena Williams  1.443618 0.6670241
## Anett Kontaveit  1.436598 0.3928183
## Garbine Muguruza 1.377632 0.4008520
## Danielle Collins 1.343972 0.4283982
## Simona Halep     1.322935 0.4277812
## Naomi Osaka      1.303551 0.4459612
## Maria Sakkari    1.279318 0.3847084
## Belinda Bencic   1.143043 0.3915679
```

```r
clay_df <- as.data.frame(BTabilities(clay_btm))
clay_df <- clay_df[order(-clay_df$ability),]
head(clay_df,10)
```

```
##                           ability      s.e.
## Iga Swiatek              4.099144 0.8430453
## Ashleigh Barty           3.194548 0.8576130
## Aryna Sabalenka          3.066806 0.7426426
## Ons Jabeur               2.811975 0.6935539
## Cori Gauff               2.795784 0.7176646
## Paula Badosa             2.685459 0.7063825
## Barbora Krejcikova       2.480695 0.7695411
## Jessica Pegula           2.259305 0.7279882
## Anastasia Pavlyuchenkova 2.221483 0.8064336
## Veronika Kudermetova     2.208846 0.7179232
```

We see that many of the players are very different on each surface, which motivates including surface. One solution is to train the three separate models, one on each surface, and predict each match with the model corresponding to the surface that it is on. However, as seen in the grass model, this results in very high errors as few matches were played on grass. Also, it is not true that matches on different surfaces are completely independent. If a player comfortably beats another on a hard court, it is a reasonable assumption that the same outcome will happen on either grass or clay, however, the probability of this occurring may be different.

Code for Section 4: Results

Baseline Model

This model will base player abilities on their WTA ranking points. Using points as opposed to rank allows for a finer grading of each player. For example, the difference in abilities of a player ranked n and a player ranked $n+1$ may be different to the difference in abilities of a player ranked $n+1$ and $n+2$, despite having the same difference in rank. This difference will be reflected by the ranking points.

```
data <- TennisTidyr(2022,2022,2023,20)
train <- data$train
test <- data$test
players <- data$main_names
```

```
ranking_points <- data.frame(row.names = players)
rp <- list()

for (p in 1:length(players)) {
  player <- players[p]

  # find the last occurrence of the name in winner_name or loser_name in 2022
  last_occurrence_win <- tail(which(train$winner_name == player), 1)
  last_occurrence_loss <- tail(which(train$loser_name == player), 1)

  win <- last_occurrence_loss < last_occurrence_win
  if (win) {
    rp[[player]] <- train$winner_rank_points[last_occurrence_win]
  } else {
    rp[[player]] <- train$loser_rank_points[last_occurrence_loss]
  }
}
```

We have now stored each player with their year-end ranking points for 2022, and will use these to predict the 2023 season.

```
draws <- test[c('winner_name', "loser_name")]
draws$winner_rp <- sapply(draws$winner_name, function(x) rp[[x]])
draws$loser_rp <- sapply(draws$loser_name, function(x) rp[[x]])
draws$pred <- draws$winner_rp/(draws$winner_rp+draws$loser_rp)
```

```
score_predictions(draws)
```

```
## $accuracy
## [1] 0.6349892
##
## $avg_probability
## [1] 0.6595516
##
## $avg_prob_se
## [1] 0.004399248
##
## $avg_log_probability
## [1] -0.428807
##
## $abg_log_prob_se
## [1] 0.006507552
```

Basic Model

Here we reproduce the results for the standard BT model.

First we will get the training data.

```
data <- TennisTidyr(2013,2022,2023,20)
train <- data$train
test <- data$test
players <- data$main_names
```

Now we can fit the BT model.

```
btm <- BTm(outcome=1, factor(winner_name, levels=players), factor(loser_name, levels=players),
                    formula=~player, id='player', data=train)
```

```
#summary(btm)
```

Finally we can predict the 2023 season and find the model scores.

```
predictions <- predict_matches(test, players, btm)
head(predictions)
```

```
##        winner_name                loser_name      pred
## 1   Jessica Pegula           Martina Trevisan 0.8188168
## 2     Madison Keys            Lucia Bronzetti 0.7677103
## 3   Jessica Pegula                Iga Swiatek 0.3233470
## 4     Madison Keys              Magda Linette 0.7245977
## 5 Martina Trevisan              Maria Sakkari 0.2039218
## 6  Lucia Bronzetti Valentini Grammatikopoulou 0.7715677
```

```
score_predictions(predictions)
```

```
## $accuracy
## [1] 0.6340694
##
## $avg_probability
## [1] 0.6746671
##
## $avg_prob_se
## [1] 0.003485468
##
## $avg_log_probability
## [1] -0.4067051
##
## $abg_log_prob_se
## [1] 0.005106492
```

Time Weighting

Here we perform the search of w parameters for the time weighting.

```
data <- TennisTidyr(2013,2022,2023,20)
train <- data$train
test <- data$test
players <- data$main_names
```

Test a model using each of the $w \in \{0.1, 0.2, \ldots 1\}$.

```
scores <- data.frame(matrix(0, ncol=5, nrow=10), row.names = seq(0.1,1,0.1))
colnames(scores) <- c("Accuracy", "Avg Prob", "Avg Prob SE", "Avg Log Prob"
                      , "Avg Log Prob SE")

row_count = 1
for (recency_weighting in seq(0.1,1,0.1)) {
  #cat("w=",recency_weighting,"\n")
  weights <- get_recency_weights(train, recency_weighting)

  btm <- BTm(outcome=1, factor(winner_name, levels=players), factor(loser_name, levels=players),
                    weights = weights, id='player', data=train)

  predictions <- predict_matches(test, players, btm)
  score <- score_predictions(predictions)

  scores[row_count, ]["Accuracy"] <- score$accuracy
  scores[row_count, ]["Avg Prob"] <- score$avg_probability
  scores[row_count, ]["Avg Prob SE"] <- score$avg_prob_se
  scores[row_count, ]["Avg Log Prob"] <- score$avg_log_probability
  scores[row_count, ]["Avg Log Prob SE"] <- score$abg_log_prob_se

  row_count <- row_count + 1
}
scores
```

```
##      Accuracy  Avg Prob Avg Prob SE Avg Log Prob Avg Log Prob SE
## 0.1 0.6422713 0.7079612 0.004174705   -0.3629620     0.005882172
## 0.2 0.6429022 0.6995623 0.004017393   -0.3739729     0.005720565
## 0.3 0.6397476 0.6945068 0.003883539   -0.3802527     0.005562281
## 0.4 0.6340694 0.6909292 0.003761958   -0.3844255     0.005403451
## 0.5 0.6391167 0.6846734 0.003672416   -0.3931775     0.005316774
## 0.6 0.6397476 0.6802998 0.003591983   -0.3991243     0.005225210
## 0.7 0.6378549 0.6773396 0.003532186   -0.4030771     0.005153158
## 0.8 0.6340694 0.6758882 0.003495092   -0.4049045     0.005106035
## 0.9 0.6403785 0.6730521 0.003483995   -0.4092734     0.005112978
## 1   0.6340694 0.6746671 0.003485468   -0.4067051     0.005106492
```

Covariates and Model Selection

For model selection, we used a stepwise approach. The code used to produce the results in Table 2 is given below, along with the standard Bradley-Terry model with no covariates.

```
dat <- TennisTidyr(2013, 2022, 2023, 20)
dat_csv <- dat$train
data_test <- dat$test
players <- dat$main_names

# Extracting necessary columns and removing NAs
data <- data.frame(dat_csv[,c("winner_id", "winner_name", "winner_hand"
                              , "winner_ht", "w_ace", "w_df", "w_svpt"
                              , "w_1stIn", "w_1stWon",
```

```
                              "w_2ndWon", "w_SvGms", "w_bpSaved", "w_bpFaced", "loser_id", "loser_name"
                              "l_ace", "l_df", "l_svpt", "l_1stIn", "l_1stWon", "l_2ndWon", "l_SvGms",
                              "l_bpFaced", "surface")])
data <- na.omit(data)
attach(data)
```

```
# Extract all elements from winner_name and loser_name columns
all_players <- data.frame(Player=c(winner_name,loser_name))
# Count number of rows each unique player has in 'all_players'
#(i.e. each player's number of total wins & losses)
count <- count(all_players, Player, name="Total_matches")
# Only keep players who have played 20+ matches
players <- filter(count, Total_matches >= 20)

# Check that all the players in 'players' have won at least one match
which( !(players$Player%in%winner_name) )
```

```
## integer(0)
```

```
# Make a dataframe showing the hand of each player
hand <- data.frame(Player=c(winner_name,loser_name)
                   , Hand=c(winner_hand,loser_hand))
hand <- distinct(hand) # eliminates duplicate rows
# Eliminate players whose hand is unknown
hand <- filter(hand, Hand != "U")
# Similarly for height
height <- data.frame(Player=c(winner_name,loser_name)
                     , Height=c(winner_ht,loser_ht))
height <- distinct(height)

# Make 'predictors' dataframe
predictors <- players %>% inner_join(hand) %>% inner_join(height)
```

```
## Joining with `by = join_by(Player)`
## Joining with `by = join_by(Player)`
```

```
predictors = predictors[,-2]
# Convert characters to factors
predictors$Player <- factor(predictors$Player)
predictors$Hand <- factor(predictors$Hand)
```

```
# Create indices that allow us to extract matches where both the winner and
# loser must be in 'predictors$Player' and their hand is known
indices_w <- winner_name %in% predictors$Player
indices_l <- loser_name %in% predictors$Player
indices_wh <- (winner_hand != "U")
indices_lh <- (loser_hand != "U")
indices <- indices_w == TRUE & indices_l == TRUE & indices_wh == TRUE & indices_lh == TRUE

# Extract 'winner_name' and 'loser_name', and keeping only the matches marked
# TRUE by indices
matches <- data.frame( Winner=factor(winner_name[indices]), Loser=factor(loser_name[indices]) )
players <- unique( c(matches[,1], matches[,2]) )
```

```
##### Make new variables
# Extract Surface variable for the matches we're considering
Surface <- surface[indices]

# Creating 3 dataframes for matches played on clay, grass, hard court
ind_clay <- Surface == "Clay"
ind_grass <- Surface == "Grass"
ind_hard <- Surface =="Hard"
matches_clay <- matches[ind_clay,]
matches_grass <- matches[ind_grass,]
matches_hard <- matches[ind_hard,]
# Initial values for the variables we're aiming to find
prev_wins_clay <- matrix(0, nrow=dim(matches_clay)[1], 2)
prev_wins_grass <-matrix(0, nrow=dim(matches_grass)[1], 2)
prev_wins_hard <- matrix(0, nrow=dim(matches_hard)[1], 2)

func_clay <- func_surface(matches_clay, prev_wins_clay)
func_grass <- func_surface(matches_grass, prev_wins_grass)
func_hard <- func_surface(matches_hard, prev_wins_hard)
```

```
# Split 'matches' dataframe
winners0 <- data.frame(Player=matches[,1])
losers0 <- data.frame(Player=matches[,2])
# Add these columns to the 'winners0' and 'losers0' dataframes
# Hard court
winners0$prev.wins.hard = rep(0,dim(winners0)[1])
winners0$prev.wins.hard[which(ind_hard==TRUE)] <- func_hard[,1]
losers0$prev.wins.hard = rep(0,dim(losers0)[1])
losers0$prev.wins.hard[which(ind_hard==TRUE)] <- func_hard[,2]
# Clay
winners0$prev.wins.clay = rep(0,dim(winners0)[1])
winners0$prev.wins.clay[which(ind_clay==TRUE)] <- func_clay[,1]
losers0$prev.wins.clay = rep(0,dim(losers0)[1])
losers0$prev.wins.clay[which(ind_clay==TRUE)] <- func_clay[,2]
# Grass
winners0$prev.wins.grass = rep(0,dim(winners0)[1])
winners0$prev.wins.grass[which(ind_grass==TRUE)] <- func_grass[,1]
losers0$prev.wins.grass = rep(0,dim(losers0)[1])
losers0$prev.wins.grass[which(ind_grass==TRUE)] <- func_grass[,2]

# Surface variable:
winners0$Surface <- winners0$prev.wins.clay + winners0$prev.wins.grass + winners0$prev.wins.hard
losers0$Surface <- losers0$prev.wins.clay + losers0$prev.wins.grass + losers0$prev.wins.hard
winners0 <- winners0[,-(2:4)]
losers0 <- losers0[,-(2:4)]

variables <- func(matches, data, indices)
```

```
# Add in columns for new variables
winners <- data.frame(winners0, variables$w_output)
losers <- data.frame(losers0, variables$l_output)
# Convert the 'Player' column to rows
library(tibble)
```

```
predictors = column_to_rownames(predictors, var="Player")

# Combine everything into a single list
data_for_model <- list(winners=winners, losers=losers, predictors=predictors)

attach(data_test)
```

```
indices_test <- winner_name %in% players == TRUE & loser_name %in% players
matches_test <- data.frame( Winner=factor(winner_name[indices_test]), Loser=factor(loser_name[indices_t
players_test <- unique( c(matches_test[,1], matches_test[,2]) )
# Make sure Winner and Loser factors have same number of levels as # players
levels(matches_test$Winner) <- levels(players_test)
levels(matches_test$Loser) <- levels(players_test)

##### Make new variables
# Extract Surface variable for the matches we're considering
Surface <- surface[indices_test]

# Creating 3 dataframes for matches played on clay, grass, hard court
ind_clay <- Surface == "Clay"
ind_grass <- Surface == "Grass"
ind_hard <- Surface =="Hard"
matches_clay <- matches_test[ind_clay,]
matches_grass <- matches_test[ind_grass,]
matches_hard <- matches_test[ind_hard,]

# Initial values for the variables we're aiming to find
prev_wins_clay <- matrix(0, nrow=dim(matches_clay)[1], 2)
prev_wins_grass <-matrix(0, nrow=dim(matches_grass)[1], 2)
prev_wins_hard <- matrix(0, nrow=dim(matches_hard)[1], 2)

# Apply func_surface
func_clay <- func_surface(matches_clay, prev_wins_clay)
func_grass <- func_surface(matches_grass, prev_wins_grass)
func_hard <- func_surface(matches_hard, prev_wins_hard)

# Split 'matches' dataframe
winners0 <- data.frame(Player=matches_test[,1])
losers0 <- data.frame(Player=matches_test[,2])
```

```
# Add these columns to the 'winners0' and 'losers0' dataframes
# Hard court
winners0$prev.wins.hard = rep(0,dim(winners0)[1])
winners0$prev.wins.hard[which(ind_hard==TRUE)] <- func_hard[,1]
losers0$prev.wins.hard = rep(0,dim(losers0)[1])
losers0$prev.wins.hard[which(ind_hard==TRUE)] <- func_hard[,2]
# Clay
winners0$prev.wins.clay = rep(0,dim(winners0)[1])
winners0$prev.wins.clay[which(ind_clay==TRUE)] <- func_clay[,1]
losers0$prev.wins.clay = rep(0,dim(losers0)[1])
losers0$prev.wins.clay[which(ind_clay==TRUE)] <- func_clay[,2]
# Grass
winners0$prev.wins.grass = rep(0,dim(winners0)[1])
```

```
winners0$prev.wins.grass[which(ind_grass==TRUE)] <- func_grass[,1]
losers0$prev.wins.grass = rep(0,dim(losers0)[1])
losers0$prev.wins.grass[which(ind_grass==TRUE)] <- func_grass[,2]

# Surface variable:
winners0$Surface <- winners0$prev.wins.clay + winners0$prev.wins.grass + winners0$prev.wins.hard
losers0$Surface <- losers0$prev.wins.clay + losers0$prev.wins.grass + losers0$prev.wins.hard
winners0 <- winners0[,-(2:4)]
losers0 <- losers0[,-(2:4)]
```

```
# Apply func
variables <- func(matches_test, data_test, indices_test)

# Add in columns
winners_test <- data.frame(winners0, variables$w_output)
losers_test <- data.frame(losers0, variables$l_output)
# Combine everything into a single list
data_for_model_test <- list(winners=winners_test, losers=losers_test, predictors=predictors[players_tes
```

Here in each iteration we compute the model using n covariates, use 'summary' to see which covariate is the least significant, and then remove it to compute the next model with $n-1$ covariates. However, we have not included the printed summaries here as this takes up too many pages.

```
##### Without covariates
data_for_model0 <- list(matches=matches, predictors=predictors)
data_for_model_test0 <- list(matches=matches_test
                             , predictors=predictors[players_test,] )
model0 <- BTm(rep(1,dim(matches)[1]), player1=Winner, player2=Loser, data=data_for_model0)

##### With covariates
model_full <- BTm(rep(1,dim(matches)[1]), player1=winners, player2=losers,
              formula = ~  Hand[Player] + Height[Player] + Surface + Ace +
                Df + Svpt + FirstIn + FirstWon + SecondWon + SvGms + BpSaved +
                BpFaced + (1|Player), id="Player", data=data_for_model)
# Removing FirstIn
model1 <- BTm(rep(1,dim(matches)[1]), player1=winners, player2=losers,
                  formula = ~  Hand[Player] + Height[Player] + Surface +
                  Ace + Df + Svpt + SvGms + FirstWon + SecondWon +
                  BpSaved + BpFaced + (1|Player), id="Player",
                  data=data_for_model)
# Removing BpSaved
model2 <- BTm(rep(1,dim(matches)[1]), player1=winners, player2=losers,
              formula = ~  Hand[Player] + Height[Player] + Surface + Ace + Df +
                Svpt + SvGms + FirstWon + SecondWon + BpFaced + (1|Player),
              id="Player", data=data_for_model)
# Removing SvGms
model3 <- BTm(rep(1,dim(matches)[1]), player1=winners, player2=losers,
              formula = ~  Hand[Player] + Height[Player] + Surface + Ace + Df +
                Svpt +  FirstWon + SecondWon + BpFaced + (1|Player),
              id="Player", data=data_for_model)
# Removing Hand
model4 <- BTm(rep(1,dim(matches)[1]), player1=winners, player2=losers,
              formula = ~  BpFaced + Height[Player] + Surface + Ace + Df +
```

```
                Svpt +  FirstWon + SecondWon + (1|Player),
              id="Player", data=data_for_model)
# Removing BpFaced (although all variables are already signif at <0.1)
model5 <- BTm(rep(1,dim(matches)[1]), player1=winners, player2=losers,
              formula = ~  Height[Player] + Surface + Ace + Df + Svpt +
                FirstWon + SecondWon + (1|Player),
              id="Player", data=data_for_model)
# Removing Height
model6 <- BTm(rep(1,dim(matches)[1]), player1=winners, player2=losers,
              formula = ~  SecondWon + Surface + Ace + Df + Svpt +
                FirstWon + (1|Player),
              id="Player", data=data_for_model)
# Removing Df
model7 <- BTm(rep(1,dim(matches)[1]), player1=winners, player2=losers,
              formula = ~  Surface + Ace + SecondWon + Svpt +
                FirstWon + (1|Player),
              id="Player", data=data_for_model)
# Removing Ace
model8 <- BTm(rep(1,dim(matches)[1]), player1=winners, player2=losers,
              formula = ~  Surface + SecondWon + Svpt +
                FirstWon + (1|Player),
              id="Player", data=data_for_model)
# Removing Surface
model9 <- BTm(rep(1,dim(matches)[1]), player1=winners, player2=losers,
              formula = ~  SecondWon + Svpt +
                FirstWon + (1|Player),
              id="Player", data=data_for_model)

# Estimating accuracy and mean probability for each model
Models<-list(model1,model2,model3,model4,model5,model6,model7,model8,model9)
Cov_Results<-matrix(NA,nrow=11,ncol=5)

#Full and Model0
pred_full <- predict(model_full, newdata=data_for_model_test, type="response", se.fit=TRUE)
pred0 <- predict(model0, newdata=data_for_model_test0, type="response", se.fit=TRUE)
pred0$fit <- na.omit(pred0$fit)
pred_full$fit <- na.omit(pred_full$fit)

Cov_Results[1,1]<-mean(pred0$fit)
Cov_Results[1,2]<-mean(log(pred0$fit))
Cov_Results[1,3]<-standard_error(log(pred0$fit))
Cov_Results[1,4]<-length(which(pred0$fit>0.5)) / length(pred0$fit)
Cov_Results[1,5]<-standard_error(pred0$fit)

Cov_Results[2,1]<-mean(pred_full$fit)
Cov_Results[2,2]<-mean(log(pred_full$fit))
Cov_Results[2,3]<-standard_error(log(pred_full$fit))
Cov_Results[2,4]<-length(which(pred_full$fit>0.5)) / length(pred_full$fit)
Cov_Results[2,5]<-standard_error(pred_full$fit)

# All other models
for (i in 1:9){
  pred <- predict(Models[[i]], newdata=data_for_model_test, type="response", se.fit=TRUE)
```

```
  pred$fit <- na.omit(pred$fit)
  Cov_Results[i+2,1]<-mean(pred$fit)
  Cov_Results[i+2,2]<-mean(log(pred$fit))
  Cov_Results[i+2,3]<-standard_error(log(pred$fit))
  Cov_Results[i+2,4]<-length(which(pred$fit>0.5)) / length(pred$fit)
  Cov_Results[i+2,5]<-standard_error(pred$fit)
}

Cov_Results
```

```
##            [,1]       [,2]        [,3]      [,4]        [,5]
##  [1,] 0.5857033 -0.6052909 0.011818306 0.6581967 0.005452596
##  [2,] 0.5688977 -0.6066504 0.008995763 0.6855713 0.004344572
##  [3,] 0.5688876 -0.6066233 0.008991113 0.6847373 0.004342289
##  [4,] 0.5688721 -0.6066964 0.008997260 0.6855713 0.004343515
##  [5,] 0.5688185 -0.6067699 0.008993759 0.6847373 0.004342978
##  [6,] 0.5682496 -0.6074284 0.008953848 0.6872394 0.004324672
##  [7,] 0.5681042 -0.6077244 0.008955948 0.6864053 0.004326043
##  [8,] 0.5727572 -0.5990773 0.008867549 0.6889074 0.004359501
##  [9,] 0.5708447 -0.6011371 0.008716136 0.6822352 0.004287246
## [10,] 0.5724520 -0.5981106 0.008689508 0.6972477 0.004291474
## [11,] 0.5735120 -0.6022123 0.009373797 0.6880734 0.004561605
```

Estimating Performance over Time

Now we will plot the change in predicted rankings over time, and see how this correlates with the actual WTA rankings. First we will use a weighting of 0.9 to find the 'best five players' according to our model over the entire training set.

```
# PICKING 5 MOST IMPORTANT PLAYERS

#Loading data
Big_Data<-TennisTidyr(2013,2022,2023,5)
Big_Train<-Big_Data$train
players<-Big_Data$main_names

#Number of top players
M<-5

#Training model
weights <- get_recency_weights(Big_Train, 0.9)
Big_Model <- BTm(outcome=1, factor(winner_name, levels=players)
                 , factor(loser_name, levels=players),
                    weights = weights, id='player', data=Big_Train)

#Finding top players
Big_Rankings<-data_frame(Players=players,Ranks=BTabilities(Big_Model)[,1])
Big_Rankings<-Big_Rankings[order(Big_Rankings$Ranks,decreasing=TRUE),]
top_players<-Big_Rankings$Players[1:M]

top_players
```

```
## [1] "Iga Swiatek"       "Aryna Sabalenka"   "Petra Kvitova"
## [4] "Elina Svitolina"   "Karolina Pliskova"
```

Now we define a function for estimating these players ranks per year, now using a weighting of 0.1 and a timescale of 3 years.

```
#Function for finding rankings
Ranks<-function(year){
  #Loading data
  Data<-TennisTidyr(year-3,year,2023,1)
  Train<-Data$train
  players<-Data$main_names


#Training model
weights <- get_recency_weights(Train, 0.1)
Model <- BTm(outcome=1, factor(winner_name, levels=players), factor(loser_name, levels=players),
                    weights = weights, id='player', data=Train)

#Finding top players
Rankings<-data_frame(Players=players,Ranks=BTabilities(Model)[,1])
Rankings<-Rankings[order(Rankings$Ranks,decreasing=TRUE),]

#Finding the ranks of each top player. If they have none, we output NA
ranks<-rep(0,M)
for (i in 1:M){
  if(top_players[i] %in% Rankings$Players){
  ranks[i]<-which(Rankings$Players == top_players[i])
  }
  else{
    ranks[i]<-NA
  }
}
return(ranks)
}
```

Using this function, we call it for each year from 2016 to 2022, and store the data in a matrix.

```
#Matrix for rankings
Rankings_Mat<-matrix(NA,nrow=M,ncol=7)

for (i in 2016:2022){
  Rankings_Mat[,i-2015]<-Ranks(i)
}
```

We then transform this matrix to be formatted correctly for ggplot2, then plot the results.

```
n=7

#Transforming data to fit ggplot2
Rankings_Data<-as.data.frame(t(Rankings_Mat),row.names = seq(2016,2022))
colnames(Rankings_Data)<-top_players
longDat <- Rankings_Data %>% pivot_longer(cols = everything()
```

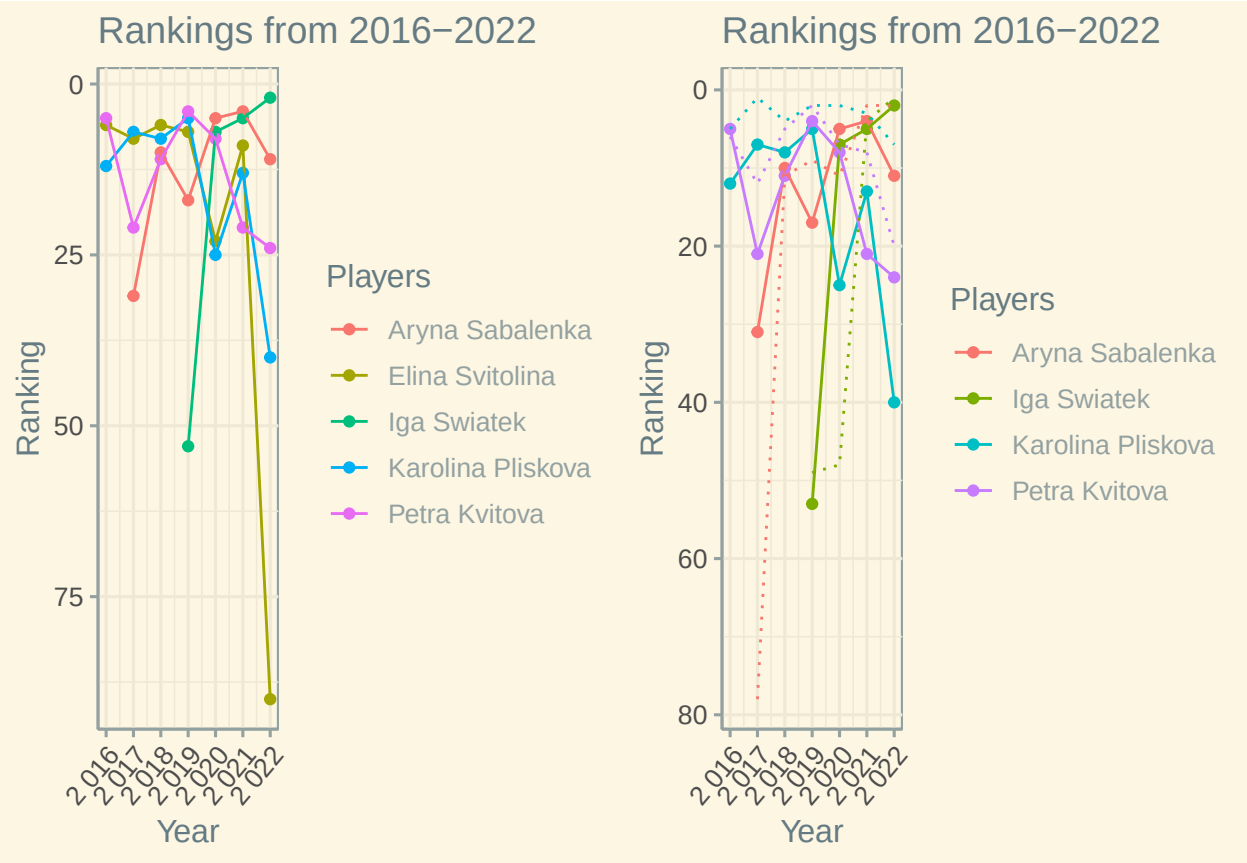
```
                                          , names_to = "Players"
                                          , values_to = "Ranks") %>% dplyr::arrange(Players)
longerDat<-longDat %>% group_by(Players) %>%
            slice(1:n) %>%
            mutate(Year = as.numeric(row_number()+2015))
breaks = seq(min(longerDat$Year),max(longerDat$Year), length.out = n)

#Plotting
timeplot1<-ggplot(longerDat,aes(x = Year, y = Ranks, col = Players)) +
            geom_point() +
            geom_line() +
  theme_solarized() +
  scale_y_reverse() +
   theme(axis.text.x = element_text(angle = 50, vjust = 1, hjust = 1)) +
  scale_x_continuous(labels = label_number(accuracy = 1),breaks=breaks) +
  labs(x='Year',y='Ranking',title='Rankings from 2016-2022')
```

We then repeat this process, except now looking for the WTA ranking instead of our own. We can then plot the WTA ranking over ours.

```
#PLOTTING WTA RANK AGAINST OURS

#Function for finding WTA rankings
WTA_Ranks<-function(year){
  data<-read.csv(paste('data/wta_matches_',year,'.csv',sep=''))
  ranks<-rep(NA,M)
  for (i in 1:M){
    possible_ranks<-data$winner_rank[data$winner_name==top_players[i]]
    if(length(possible_ranks>0)){
      ranks[i]<-min(possible_ranks)
    }
  }
  return(ranks)
}

#Matrix for rankings
WTA_Rankings_Mat<-matrix(NA,nrow=M,ncol=7)
for (i in 2016:2022){
  WTA_Rankings_Mat[,i-2015]<-WTA_Ranks(i)
}

#Transforming Data for ggplot2
WTA_Rankings_Data<-as.data.frame(t(WTA_Rankings_Mat),row.names = seq(2016,2022))
colnames(WTA_Rankings_Data)<-top_players
longDat <- WTA_Rankings_Data %>% pivot_longer(cols = everything()
                                              , names_to = "Players"
                                              , values_to = "Ranks") %>% dplyr::arrange(Players)
WTA_longerDat<-longDat %>% group_by(Players) %>%
            slice(1:7) %>%
            mutate(Year = as.numeric(row_number()+2015))

#Removing anomaly
WTA_longerDat$Ranks[WTA_longerDat$Players=='Iga Swiatek' & WTA_longerDat$Year==2018]<-NA
```

```
#Plotting with Elina Svitolina removed
timeplot2<-ggplot(longerDat[!longerDat$Players=='Elina Svitolina',]
                  ,aes(x = Year, y = Ranks, col = Players)) +
             geom_point() +
             geom_line() +
    geom_line(data=WTA_longerDat[!WTA_longerDat$Players=='Elina Svitolina',]
              ,aes(x = Year, y=Ranks, col=Players),linetype=3) +
  theme_solarized() +
  scale_y_reverse() +
   theme(axis.text.x = element_text(angle = 50, vjust = 1, hjust = 1)) +
  scale_x_continuous(labels = label_number(accuracy = 1),breaks=breaks) +
  labs(x='Year',y='Ranking',title='Rankings from 2016-2022')
```

```
#FINAL OUTPUT
plot_grid(timeplot1,timeplot2,nrow=1)
```



Github

https://github.com/oijbaker/Tennis.git