Updated RAPM Model Documentation

Basil Seif

8/14/2019

This version of regularized-adjusted plus-minus (RAPM) draws on a number of previously done RAPM calculations and open source studies on player evaulation, including work done by Joe Sill, Jeremias Engelman, and Justin Jacobs. The general goal of RAPM is to measure on-court player performance while minimizing the effect that different players and lineups have on that individual performance. The way RAPM does this is by using ridge regression to tease out a player’s on-court contribution while minimizing multicollinearity between different players.

Older versions of RAPM mainly focus on creating one holistic measurement based on the points margin of each 10-player stint (a stint is defined as a group of 10 players that are on the floor at any given time). In this version of RAPM, I will be calculating Offensive and Defensive RAPM and adding them together for each player to get a total RAPM value. This specific version of RAPM will be a 1 year RAPM model for each player from 1998-present including all Regular Season and Playoff games that each player played in a given season.

```r
####Read in Libraries####
library(RODBC)
library(data.table)
library(RColorBrewer)
library(weights)
library(dplyr)
library(Hmisc)
library(glmnet)
library(tidyr)
library(SafeBayes)
library(matlib)
library(plyr)
library(Matrix)

####Helper Functions####
remove_zero_var_cols <- function(df){
  colsToRemove <- c()
  for(i in 1:ncol(df)){
    if(length(unique(df[,i])) == 1){
      colsToRemove <- c(i, colsToRemove)
    }
  }
  if(length(colsToRemove) >= 1){
    df <- df[,-colsToRemove]
  }
  return(df)
}
outersect <- function(x, y) {
  sort(c(setdiff(x, y),
         setdiff(y, x)))
}
```

The first obvious step in building this model is to pull in the correct data from the database. As it currently stands, our database extracts NBA lineup data and calculates box score stats, starting scores, ending scores, and lineups for each stint in each NBA game (playoffs and regular season) starting as early as the 1997-18 NBA season. The below code pulls all of these stints from 1998-present and saves them in the $nba\_lineup\_game\_dim$ dataframe. I also decided to pull from the $nba\_all\_player\_stats$ table and apm_player_stats table for future use in this analysis.

```
####Read in Data####
mydb <- odbcDriverConnect(connection = "Driver={SQL Server Native Client 11.0};server=PHX-SQL07
\\BBOPS; database=BOps_DW;trusted_connection=yes;")
nba_lineup_game_dim <- sqlQuery(mydb, "select * from dbo.nba_lineup_game_dim
                                            where 1 = 1
                                            and league_id = 00
                                            and season_type in ('Regular Season', 'Playoffs')")
nba_all_player_stats <- sqlQuery(mydb, "select * from dbo.nba_all_player_stats
                                            where 1 = 1
                                            and league_id = 00
                                            and season_type = 'Regular Season'
                                                                and game_id = 'All'
                                                                and stat_type = 'Totals'
                                          and team_id <> 'TOT'")
apm_player_stats <- sqlQuery(mydb, "select * from dbo.apm_player_stats
                                          where 1 = 1
                                          and league_id = 00
                                          and game_id = 'All'
                                          and stat_type = 'Totals'")
odbcClose(mydb)


####Clean Data####
nba_lineup_game_dim$season_num <- as.numeric(gsub("NBA_", "", nba_lineup_game_dim$season_name))
nba_lineup_game_dim <- remove_zero_var_cols(nba_lineup_game_dim)
nba_all_player_stats <- remove_zero_var_cols(nba_all_player_stats)
nba_all_player_stats_totals <- setDT(nba_all_player_stats)[, lapply(.SD, sum), by = c("person_i
d", "season_name"), .SDcols = c(names(nba_all_player_stats)[which(grepl("poss_", names(nba_all_p
layer_stats)) == T)], "mp")]
apm_player_stats <- remove_zero_var_cols(apm_player_stats)
```

Building the stint matrix is one of the keys of developing a RAPM model. If there are p players in the NBA and s stints in a given timeframe, the basic stint matrix should be a s x $2p + 1$ matrix with an offensive and defensive column for each player populated with 1’s if a player played on offense in a specific stint, -1’s if a player played on defense in a specific stint, and 0’s otherwise. The extra column is a constant column populated with 1’s and $100 * pointsscoredperoffensivepossession$ is the target vector.

The first step in building the stint matrix for a given year is to filter and clean the data properly. In this fist section of code, I define the first and last years observed in the overall play by play data as well as an empty list $RAPM\_list$ to store the results for each year. I have also coded an option to run a 2 or 3 year RAPM model. If $RAPM\_type$ is changed from 1 to 2 or 3, the stints used to model RAPM will be stints played from the last 2 or 3 season of available NBA play by play data.

```
####Stint Evaluation####
RAPM_type = 1
RAPM_list <- list()
seasons_loop_min = min(nba_lineup_game_dim$season_num) - (RAPM_type - 1)
seasons_loop_max = max(nba_lineup_game_dim$season_num)
seasons_loop = c(seasons_loop_min:seasons_loop_max)
```

This is also where I start a for loop that cycles through each season or timeframe and calculates RAPM for the corresponding players. The main goal here is to set the appropriate date filters to the data and to get rid of any garbage time stints that the data may have.

When filtering the data for each season, I take either the stint data for the season indicated by ssn or the stint data for that season and several previous seasons as well (depending on if we are doing a 1-year, 2-year, or 3-year RAPM). Another important thing to note is that I take only the stints for the home team (each stint is listed in the data twice, once for each team. While we are filtering out half of the stints, in the end the final stint matrix will also have a duplicate of each stint, in order to account for each team playing both offense and defense in the same stint).

The next step is to arrange the data by $game\_id$, $period$, and $game\_clock$. Here, I also address the issue of stints with 0 possessions that have more than 0 points scored. Depending on how points are counted, this may be the case if a lineup is specifically subbed in for a free throw. For stints like this, I change the number of possessions to 1. I filter out any other stints where there are 0 possessions and zero points. Fixing these issues is important when calculating the target vector later in the modeling (Note: these stints with 0 possessions and more than 0 points can be entirely excluded from the dataset, it is really up to how you want to interpret the data).

The next important step in this part of the code is to get rid of any garbage time stints that may bias the results. To define garbage time, I used Ben Falk's garbage time rules that he uses to calculate many of the metrics from his website CleaningTheGlass.com. To summarize them, there are 3 specific ways that garbage time can be defined: 1) if there are 9-12 minutes left in a game and the margin is greater than or equal to 25 points, 2) if there are 6-9 minutes left in a game and the margin is greater than or equal to 20 points, and 3) if there are <= 6 minutes left in a game and the margin is greater than or equal to 10 points. If the margin dips below any of these benchmarks during any of these specificed time markers, those stints are not considered garbage time.

```
####Start Stint Evaluation For Loop####
for(ssn in seasons_loop){
  #print(ssn)
  ##Filter Down Data for Each Year
  if(RAPM_type == 1){
    dataRAPM <- filter(nba_lineup_game_dim, season_num == ssn & home_away == "Home")
  }else{
    dataRAPM <- filter(nba_lineup_game_dim, season_num %in% c((ssn - (RAPM_type - 1)):ssn) & hom
e_away == "Home")
  }
  dataRAPM <- arrange(dataRAPM, game_id, period, -game_clock_int_start)
  dataRAPM$poss_tot[which(dataRAPM$pts > 0 & dataRAPM$poss_tot == 0)] = rep(1, length(which(data
RAPM$pts > 0 & dataRAPM$poss_tot == 0)))
  dataRAPM$poss_tot[which(dataRAPM$opp_pts > 0 & dataRAPM$poss_tot == 0)] = rep(1, length(which
(dataRAPM$opp_pts > 0 & dataRAPM$poss_tot == 0)))
  #dataRAPM <- filter(dataRAPM, poss_tot > 0 & poss_off > 0 & poss_def > 0)
  dataRAPM <- filter(dataRAPM, poss_tot > 0)

  ##Filter Out Garbage Time Stints
  #First Garbage Time Rule
  garbage_time_ind1_games <- dataRAPM %>% group_by(game_id) %>% filter(period == 4 & game_clock_
int_start <= 72000 & game_clock_int_start > 54000) %>% mutate(score_end = (opp_score_end - score
_end), score_start = (opp_score_start - score_start)) %>% select(game_id, score_end, score_star
t)
  garbage_time_ind1_games <- unique(garbage_time_ind1_games[c(which(garbage_time_ind1_games$scor
e_start >= 25 & garbage_time_ind1_games$score_end <= 25), which(garbage_time_ind1_games$score_st
art <= -25 & garbage_time_ind1_games$score_end >= -25)), "game_id"])
  garbage_time_ind1 <- which(dataRAPM$period == 4 & dataRAPM$game_clock_int_start <= 72000 & dat
aRAPM$game_clock_int_start > 54000 &
                               abs(dataRAPM$opp_score_end - dataRAPM$score_end) >= 25 & abs(data
RAPM$opp_score_start - dataRAPM$score_start) >= 25)

  #Second Garbage Time Rule
  garbage_time_ind2_games <- dataRAPM %>% group_by(game_id) %>% filter(period == 4 & game_clock_
int_start <= 54000 & game_clock_int_start > 36000) %>% mutate(score_end = (opp_score_end - score
_end), score_start = (opp_score_start - score_start)) %>% select(game_id, score_end, score_star
t)
  garbage_time_ind2_games <- unique(garbage_time_ind2_games[c(which(garbage_time_ind2_games$scor
e_start >= 20 & garbage_time_ind2_games$score_end <= 20), which(garbage_time_ind2_games$score_st
art <= -20 & garbage_time_ind2_games$score_end >= -20)), "game_id"])
  garbage_time_ind2 <- which(dataRAPM$period == 4 & dataRAPM$game_clock_int_start <= 54000 & dat
aRAPM$game_clock_int_start > 36000 &
                               abs(dataRAPM$opp_score_end - dataRAPM$score_end) >= 20 & abs(data
RAPM$opp_score_start - dataRAPM$score_start) >= 20)

  #Third Garbage Time Rule
  garbage_time_ind3_games <- dataRAPM %>% group_by(game_id) %>% filter(period == 4 & game_clock_
int_start <= 36000) %>% mutate(score_end = (opp_score_end - score_end), score_start = (opp_score
_start - score_start)) %>% select(game_id, score_end, score_start)
  garbage_time_ind3_games <- unique(garbage_time_ind3_games[c(which(garbage_time_ind3_games$scor
e_start >= 10 & garbage_time_ind3_games$score_end <= 10), which(garbage_time_ind3_games$score_st
art <= -10 & garbage_time_ind3_games$score_end >= -10)), "game_id"])
  garbage_time_ind3 <- which(dataRAPM$period == 4 & dataRAPM$game_clock_int_start <= 36000 &
                              abs(dataRAPM$opp_score_end - dataRAPM$score_end) >= 10 & abs(data
RAPM$opp_score_start - dataRAPM$score_start) >= 10)

  #Combine and Filter
  garbage_time_games_exclude <- unique(c(garbage_time_ind1_games$game_id, garbage_time_ind2_game
s$game_id, garbage_time_ind3_games$game_id))
  garbage_time_games_exclude_ind <- which(dataRAPM$game_id %in% garbage_time_games_exclude)
  garbage_time_ind <- unique(c(garbage_time_ind1, garbage_time_ind2, garbage_time_ind3))
  garbage_time_ind <- garbage_time_ind[-which(garbage_time_ind %in% garbage_time_games_exclude_i
nd)]
  dataRAPM <- dataRAPM[-garbage_time_ind, ]
  dataRAPM <- filter(dataRAPM, is.na(score_start) == F)
  dataRAPMPossOffInd <- which(dataRAPM$poss_off == 0)
  dataRAPMPossDefInd <- which(dataRAPM$poss_def == 0)

  ##Find All Unique Player IDs
  player_id_refs <- data.frame(person_id = as.vector(t(dataRAPM[,names(dataRAPM)[which(grepl("_p
erson_id",names(dataRAPM)))]])),
                              full_name = as.vector(t(dataRAPM[,names(dataRAPM)[which(grepl("te
am_player.*_full_name", names(dataRAPM)))]])))
  player_id_refs <- unique(player_id_refs)
  row.names(player_id_refs) = NULL
  unique_ID = unique(player_id_refs$person_id)
  keep_id_unique = match(unique_ID, player_id_refs$person_id)
  player_id_refs = player_id_refs[keep_id_unique,]
  player_id_refs = arrange(player_id_refs, person_id)

  ##Find Low Min Players
  playerTotals <- filter(nba_all_player_stats_totals, season_name == paste0("NBA_", ssn))
  # minCutoff <- 250
  # HighMinPlayerTotals <- filter(playerTotals, mp >= minCutoff)
  # LowMinPlayerTotals <- filter(playerTotals, mp < minCutoff)

  ##Filter Out All Star Stuff
  team_ref_ids <- unique(dataRAPM[,c("team_abbr", "home_team_id")])
  team_ref_ids <- na.omit(team_ref_ids[1:30,])
  row.names(team_ref_ids) <- NULL
  team_ref_ids <- arrange(team_ref_ids, home_team_id)
  dataRAPM <- filter(dataRAPM, home_team_id %in% team_ref_ids$home_team_id)
```

In most RAPM models, the stint matrix and the target are the only things needed. However, because we have the data easily accessible, I have added features to reflect the effects of home court advantage, days rest, the score difference at a certain point of the game, and time remaining in a game.

For the home court features, I created a binary matrix with the home team in each stint indicated as a 1 and the away team in each stint indicated with a -1.

For the days rest feature, I created a similar binary matrix with columns for the number of days of rest for a team and the number of days of rest for an opposing team. All of the columns corresponding to the days rest of the opposing team are filled with -1's instead of 1's. Days rest in a given NBA season can range from 0 to 3 days for the bulk of the schedule, up to 7 to 10 days for unique parts of the schedule (All Star Break).

For the score difference feature, I calculate every 5th percentile of scoring differences throughout the stint data and create a binary matrix to indicate whether or not a score margin for a certain stint falls into one of those ranges.

Finally, the time remaining feature is simply a vector of the average of time remaining at the beginning of a stint and at the end of a stint. This is then multiplied by the score difference matrix to create a combined score difference/time remaining effect.

```
  ##Add Home Court Features (Home Court, Days Rest, Score Diff, Time Remaining)
  homeCourtMat <- model.matrix(~ as.character(home_team_id) + 0, dataRAPM)
  colnames(homeCourtMat) <- paste0("HomeCourt_", team_ref_ids$team_abbr)
  homeCourtMat2 <- homeCourtMat * -1
  homeCourtMatFull <- rbind(homeCourtMat, homeCourtMat2)

  ##Add Days Rest Features
  dataRAPM$team_days_rest[which(dataRAPM$team_days_rest < 0)] = rep(0, length(which(dataRAPM$tea
m_days_rest < 0)))
  dataRAPM$opp_team_days_rest[which(dataRAPM$opp_team_days_rest < 0)] = rep(0, length(which(data
RAPM$opp_team_days_rest < 0)))
  daysRestMat <- model.matrix(~ as.character(team_days_rest) + 0, dataRAPM)
  oppDaysRestMat <- model.matrix(~ as.character(opp_team_days_rest) + 0, dataRAPM)
  oppDaysRestMat <- oppDaysRestMat * -1
  colnames(daysRestMat) <- gsub("as.character[(]team_days_rest[)]", "DaysRest", colnames(daysRes
tMat))
  colnames(oppDaysRestMat) <- gsub("as.character[(]opp_team_days_rest[)]", "OppDaysRest", colnam
es(oppDaysRestMat))
  daysRestMatFull <- cbind(daysRestMat, oppDaysRestMat)

  daysRestMat2 <- oppDaysRestMat * -1
  oppDaysRestMat2 <- daysRestMat * -1
  colnames(daysRestMat2) <- gsub("OppDaysRest", "DaysRest", colnames(daysRestMat2))
  colnames(oppDaysRestMat2) <- gsub("DaysRest", "OppDaysRest", colnames(oppDaysRestMat2))
  daysRestMatFull2 <- cbind(daysRestMat2, oppDaysRestMat2)

  daysRestMatFull <- rbind.fill.matrix(daysRestMatFull, daysRestMatFull2)
  daysRestMatFull <- daysRestMatFull[,sort(colnames(daysRestMatFull))]
  daysRestMatFull[is.na(daysRestMatFull)] = 0

  ##Score Difference Features
  scoreDiff <- (dataRAPM$score_start - dataRAPM$opp_score_start + dataRAPM$score_end - dataRAPM
$opp_score_end)/2
  scoreDiff2 <- scoreDiff * -1
  scoreDiffBins <- quantile(c(scoreDiff, scoreDiff2), na.rm = T, seq(0, 1, 0.05))
  scoreDiffBins[length(scoreDiffBins)] <- scoreDiffBins[length(scoreDiffBins)] + 1
  scoreDiffMat <- matrix(0, ncol = length(scoreDiffBins) - 1, nrow = nrow(dataRAPM))
  scoreDiffMat2 <- matrix(0, ncol = length(scoreDiffBins) - 1, nrow = nrow(dataRAPM))
  for(i in 1:ncol(scoreDiffMat)){
    scoreDiffMat[,i] <- as.numeric(scoreDiff >= scoreDiffBins[i] & scoreDiff < scoreDiffBins[i +
1])
    scoreDiffMat2[,i] <- as.numeric(scoreDiff2 >= scoreDiffBins[i] & scoreDiff2 < scoreDiffBins
[i + 1])
  }
  scoreDiffMatFull <- rbind(scoreDiffMat, scoreDiffMat2)
  colnames(scoreDiffMatFull) <- paste0("ScoreDiff_", sapply(1:(length(scoreDiffBins) - 1), funct
ion(x) paste0(gsub("%","",names(scoreDiffBins)[x]), "-", names(scoreDiffBins)[x+1])))

  ##Time Remaining Features
  timeRemaining <- (dataRAPM$game_clock_int_end + dataRAPM$game_clock_int_start)/2/6000 + ifelse
(4 - dataRAPM$period < 0, 0, 4 - dataRAPM$period) * 12
  timeRemainingFull <- rep(timeRemaining, 2)
```

```
##Absorb Extra Terms into Intercept
homeCourtMatFull <- homeCourtMatFull[,-1]
daysRestMatFull <- daysRestMatFull[, -c(1, (ncol(daysRestMatFull)/2 + 1))]
scoreDiffMatFull <- scoreDiffMatFull[,-1]
scoreTimeInt <- scoreDiffMatFull * timeRemainingFull
```

For most of the features and effects matrices that were built above, it is noticeable that most of them required making a duplicate matrix of sorts and binding the duplicate matrix to the bottom of the original matrix. As discussed previously, this is becasue we are only looking at the stint data for the home team and will therefore have to read those stints for the away team.

In building the stint matrix, as one can see, the first two lines of code are building matrices of identical size, $stintMat$ and $stintMat2$. The second matrix will reflect the stint data for the away teams in each game. Within the year for loop, I create another for loop that cycles through all of the unique players in the stint data and fills these two stint matrices. As previously described, each of the two stint matrices indicates an offensive player with a 1 and a defensive player with a -1. This for loop also updates the minutes played and possessions played for each player after the garbage time stints have been removed from the data. This is important because players in the lowest quartile of possessions played in the stint data are removed from the analysis and final prediction matrix entirely (meaning that every stint they are involved in is also removed). This is to make sure that low possession players are not over inflated by the model (this may however filter out players who would typically be high possession players but are injured).

Finally, once the stint matrices are filled and the feature matrices are binded to the overall stint matrix, the result vector is calculated as the pts per 100 possessions for each stint.

```
  ##Build Stint Matrix
  stintMat <- matrix(0, ncol = nrow(player_id_refs) * 2, nrow = nrow(dataRAPM))
  stintMat2 <- matrix(0, ncol = nrow(player_id_refs) * 2, nrow = nrow(dataRAPM))
  updatedPlayerMinutes <- data.frame(person_id = player_id_refs$person_id, full_name = player_id
_refs$full_name, mp = NA, poss_off = NA, poss_def = NA)
  colnames(stintMat) <- c(paste0("Off_", paste0("ID", player_id_refs$person_id)), paste0("Def_",
paste0("ID", player_id_refs$person_id)))
  colnames(stintMat2) <- colnames(stintMat)
  for(i in 1:nrow(player_id_refs)){
    stintMat[,i] <- stintMat[,i] + as.numeric(rowSums(dataRAPM[,c(paste0("team_player", c(1:5),
"_person_id"))] == player_id_refs$person_id[i]))
    stintMat[,i + nrow(player_id_refs)] <- stintMat[,i + nrow(player_id_refs)] - as.numeric(rowS
ums(dataRAPM[,c(paste0("opp_team_player", c(1:5), "_person_id"))] == player_id_refs$person_id
[i]))

    stintMat2[,i] <- stintMat2[,i] + as.numeric(rowSums(dataRAPM[,c(paste0("opp_team_player", c(
1:5), "_person_id"))] == player_id_refs$person_id[i]))
    stintMat2[,i + nrow(player_id_refs)] <- stintMat2[,i + nrow(player_id_refs)] - as.numeric(ro
wSums(dataRAPM[,c(paste0("team_player", c(1:5), "_person_id"))] == player_id_refs$person_id[i]))

    mp_update <- sum(dataRAPM$mp[which(as.numeric(rowSums(dataRAPM[,c(paste0("team_player", c(1:
5), "_person_id"), paste0("opp_team_player", c(1:5), "_person_id"))] == player_id_refs$person_id
[i])) == 1)])
    poss_off_update <-  sum(dataRAPM$poss_off[which(as.numeric(rowSums(dataRAPM[,c(paste0("team_
player", c(1:5), "_person_id"), paste0("opp_team_player", c(1:5), "_person_id"))] == player_id_r
efs$person_id[i])) == 1)])
    poss_def_update <- sum(dataRAPM$poss_def[which(as.numeric(rowSums(dataRAPM[,c(paste0("team_p
layer", c(1:5), "_person_id"), paste0("opp_team_player", c(1:5), "_person_id"))] == player_id_re
fs$person_id[i])) == 1)])
    updatedPlayerMinutes$mp[which(updatedPlayerMinutes$person_id == player_id_refs$person_id
[i])] = mp_update
    updatedPlayerMinutes$poss_off[which(updatedPlayerMinutes$person_id == player_id_refs$person_
id[i])] = poss_off_update
    updatedPlayerMinutes$poss_def[which(updatedPlayerMinutes$person_id == player_id_refs$person_
id[i])] = poss_def_update
  }
  updatedPlayerMinutes$poss_tot <- updatedPlayerMinutes$poss_off + updatedPlayerMinutes$poss_def
  stintMatFull <- rbind(stintMat, stintMat2)
  stintMatFull <- cbind(1, stintMatFull)
  colnames(stintMatFull)[1] <- "Constant"
  margin <- 100 * (dataRAPM$pts)/dataRAPM$poss_off
  margin2 <- 100 * (dataRAPM$opp_pts)/dataRAPM$poss_def
  marginFull <- c(margin, margin2)
  marginFull[is.na(marginFull)] = 0
  marginFull[is.infinite(marginFull)] = 0
  playerTotals <- filter(playerTotals, person_id %in% as.numeric(gsub("Def_ID|Off_ID", "",colnam
es(stintMatFull)[-1])))
  playerTotals <- arrange(playerTotals, person_id)
  possVec <- c(dataRAPM$poss_off, dataRAPM$poss_def)
  stintMatFull <- cbind(stintMatFull, homeCourtMatFull, daysRestMatFull, scoreDiffMatFull, score
TimeInt)
  rm(homeCourtMatFull, homeCourtMat, homeCourtMat2, daysRestMatFull, daysRestMatFull2, scoreDiff
Mat, scoreDiffMat2, scoreDiffMatFull, scoreTimeInt, stintMat, stintMat2)
```

```
  ##Get Rid of Low Possession Player Stints
  possCutoff <- summary(updatedPlayerMinutes$poss_tot)[2]
  LowPossPlayerTotals <- filter(updatedPlayerMinutes, poss_tot < possCutoff)
  lowPossCols <- c(paste0("Off_ID", LowPossPlayerTotals$person_id), paste0("Def_ID", LowPossPlay
erTotals$person_id))
  stintMatFull <- stintMatFull[, -which(colnames(stintMatFull) %in% lowPossCols)]
  playerIDVars <- colnames(stintMatFull)[which(grepl("Def_ID|Off_ID", colnames(stintMatFull)) ==
T)]
  stintsToRemove <- which(rowSums(abs(stintMatFull[,which(colnames(stintMatFull) %in% playerIDVa
rs)])) != 10)
  stintMatFull <- stintMatFull[-stintsToRemove,]
  marginFull <- marginFull[-stintsToRemove]
  possFull <- possVec[-stintsToRemove]
```

Once the prediction matrix and target vector are created, the next step is building the ridge regression model. In the code below, I tried two different models.

The first method I tried was using $cv.glmnet$ function from the $glmnet$ package. This function picks the optimal lambda for you by iterating through a vector of potential lambdas that I provided in the $lambdas$ vector. Typically in ridge regression, most functions standardize the data. It is important to NOT standardize the data here because it will dampen the effects of the model. Also, because of the way that the $glmnet$ package does its ridge calculations, the actual resulting coefficients from the model will never be exactly the same as the manual ridge calculation done below, even if the lambdas are the same. Similarly, the optimal lambda will typically be as small as possible (with the current vector of lambdas, the optimal lambda tends to be 50).

The second model I try is manually calculating ridge regression using matrix mulitplication. One of the important things to note here is that is set to be 2222. This is based on research done by Joe Sill and Jeremias Engelman on the optimal value for for RAPM (again, because cv.glmnet calculates ridge a bit differently, the optimal will never be close to 2222 in the first model). Another thing to note here is the W matrix, which is a diagonal matrix with the total number of possessions per stint as the diagonal. One of the frustrating things about RAPM is that low possession stints are weighted the same as high possessions, meaning that those stints get disproportionate weight in player rankings. While RAPM is an efficiency statistic and values players who produce positively per stint, I find it important to include a possession-weighting matrix W into the ridge regression (in cv.glmnet, I also weighted the ridge regression by number of possession per stint, a can be seen below). The last step in this analysis is to report the standard deviations for each RAPM player coefficient.

```
  ##CV Ridge Regression
  x <- stintMatFull
  y <- marginFull
  lambdas <- seq(50, 4000, 50)
  fit <- glmnet(x, y, alpha = 0, lambda = lambdas, standardize = F, standardize.response = F)
  cv_fit <- cv.glmnet(x, y, alpha = 0, lambda = lambdas, weights = possFull, nfolds = 10, standa
rdize = F, standardize.response = F)
  # cv_fit <- cv.glmnet(x, y, alpha = 0, lambda = lambdas, standardize = F, standardize.response
= F)
  opt_lambda <- cv_fit$lambda.min
  #opt_lambda <- 50
  fit <- cv_fit$glmnet.fit
  summary(fit)
  allBetas <- cv_fit$glmnet.fit$beta[,which(cv_fit$glmnet.fit$lambda == opt_lambda)]
  playerBetas <- allBetas[names(allBetas)[which(grepl("Off_ID|Def_ID", names(allBetas)) == T)]]
  results <- as.data.frame(do.call(rbind, strsplit(names(playerBetas), "_ID")))
  names(results) <- c("rapm_type", "person_id")
  results$RAPM <- playerBetas
  results <- merge(results, player_id_refs, by = "person_id", all.x = T)
  results <- spread(results, rapm_type, RAPM)
  # results <- subset(results, select = -c(Constant))
  names(results)[which(names(results) == "Def")] = "D_RAPM"
  names(results)[which(names(results) == "Off")] = "O_RAPM"
  results$RAPM <- results$D_RAPM + results$O_RAPM
  results <- merge(results, updatedPlayerMinutes[,c("person_id", "mp")], by = "person_id", all.x
= T)
  results <- arrange(results, -RAPM)

  ##Bayesian Ridge Calculation
  lambda <- 2222
  # lambda <- 4500
  # lambda <- 1500
  W <- Matrix(0, nrow = nrow(x), ncol = nrow(x))
  diag(W) <- possFull
  xTx <- t(x) %*% W %*% x
  xTy <- t(x) %*% W %*% y
  xTx <- t(x) %*% x
  xTy <- t(x) %*% y
  I <- diag(nrow(xTy))
  xTx_lambda <- xTx + lambda * I
  mean <- solve(xTx_lambda) %*% xTy
  playerMeanBetas <- mean[which(grepl("Off_ID|Def_ID", colnames(x)) == T)]
  meanDF <- data.frame(varID = colnames(x)[which(grepl("Off_ID|Def_ID", colnames(x)) == T)], RAP
M = playerMeanBetas)
  row.names(meanDF) <- NULL
  meanDF$varID <- as.character(meanDF$varID)
  meanDF <- data.frame(rapm_type = do.call(rbind, strsplit(meanDF$varID, "_ID"))[,1],
                       person_id = as.numeric(do.call(rbind, strsplit(meanDF$varID, "_ID"))[,2
]),
                       RAPM = meanDF$RAPM)
  meanDF <- merge(meanDF, player_id_refs, by = "person_id", all.x = T)
  meanDF <- spread(meanDF, rapm_type, RAPM)
  names(meanDF)[which(names(meanDF) == "Def")] = "D_RAPM"
  names(meanDF)[which(names(meanDF) == "Off")] = "O_RAPM"
  meanDF <- merge(meanDF, updatedPlayerMinutes[,c("person_id", "mp", "poss_off", "poss_def", "po
ss_tot")], by = "person_id", all.x = T)
  meanDF$RAPM <- meanDF$O_RAPM + meanDF$D_RAPM
  meanDF <- arrange(meanDF, -RAPM)

  sigma <- 3
  var <- lambda * sigma * solve(xTx_lambda)
  varDF <- data.frame(rapm_type = do.call(rbind, strsplit(names(diag(var)), "_ID"))[,1],
                      person_id = as.numeric(do.call(rbind, strsplit(names(diag(var)), "_ID"))[,
2]),
                      VAR = diag(var))
  row.names(varDF) <- NULL
  varDF <- filter(varDF, rapm_type != "Constant")
  varDF <- spread(varDF, rapm_type, VAR)
  names(varDF)[which(names(varDF) == "Def")] = "D_Var"
  names(varDF)[which(names(varDF) == "Off")] = "O_Var"
  RAPM_DF <- merge(meanDF, varDF, by = "person_id", all.x = T)
  RAPM_DF <- arrange(RAPM_DF, -RAPM)
  RAPM_DF$season_name <- rep(paste0("NBA_", ssn), nrow(RAPM_DF))
  RAPM_list[[ssn]] <- RAPM_DF
}
RAPM_full <- do.call(rbind, RAPM_list)
```

The biggest improvement to be made is to implement a model that adds an informative Bayesian prior with the previous year's O-RAPM and D-RAPM for each player. While RAPM is a very valuable statistic and can be very positive when given enough stints to learn over time, some players in certain years can get very undervalued due to certain inconsistencies during the year. For instance, in 2018-19 Rudy Gobert's plus-minus production dipped relative to his defensive dominance in previous years. Instead of being a top D-RAPM candidate like he usually is, he was not very close to the top in production and rating. One way to correct for that is to run a 3-year RAPM model to regularize the results a bit. Another answer could be to use his 2017-18 O-RAPM and D-RAPM results as priors.

Another important thing to fix is how variance and standard deviation results are calculated in the manual ridge regression. I had trouble calculating what sigma actually should be to yield the correct standard deviation results. Some more thought and work should be put into this.