Warning – this post discusses gambling odds and even describes me placing small $5 bets, which I can easily afford to lose. In no way should this be interpreted as advice to anyone else to do the same, and I accept absolutely no liability for anyone who treats this blog post as a basis for gambling advice. If you find yourself losing money at gambling or suspect you or someone close to you has a gambling problem, please seek help from <https://www.gamblinghelponline.org.au/> or other services.

**Appreciating home team advantage**

Last week [I blogged about using Elo ratings to predict the winners in the Australian Football League](http://freerangestats.info/blog/2019/03/23/afl-elo) (AFL) to help me blend in with the locals in my Melbourne workplace footy-tipping competition. Several people pointed out that my approach ignored the home team advantage, which is known to be a significant factor in the AFL. How significant? Well, here’s the proportion of games won by the home team in each season since the AFL’s beginning:

Overall, 59% of games in the AFL history have been won by the home team, although that proportion varies materially from season to season. Another way of putting this – if my AFL prediction system was as simple as “always pick the home team” I would expect (overall) to score a respectable 59% of successful picks.

My first inclination in response to this was just to add 0.09 to the chance of the home team winning from my model’s base probability, but first I thought I should check whether this adjustment varies by team. Well, it does, somewhat dramatically. This next chart, based on just modern era games (1997 and onwards), defines the home advantage as the proportion of home games won minus the proportion of away games won, divided by 2. The all-teams all-seasons average for this figure would be 0.09.

The teams that are conspicuously high on this measure are all non-Melbourne teams:

* Geelong
* Adelaide
* West Coast
* Fremantle
* Greater Western Sydney

Gold Coast (another non-Melbourne team, for any non-Australians reading) would show up as material if a success *ratio* measure were used instead of my simple (and crude) additive indicator; because their overall win rate is so low. Sydney and perhaps Brisbane stand out as good performers overall that don’t have as marked a home town advantage (or away-town disadvantage) as their non-Melbourne peers.

I presume this issue is well known to AFL afficianados. With the majority (or at least plurality – I haven’t counted) of games played in Melbourne, Melbourne-based clubs generally play many of their “away” matches still relatively close to players’ homes. Whereas (for example) the West Coast Eagles flying across the Nullarbor for a match is bound to cost their players, compared to the alternative situation of being at home and making the opponents fly west instead.

Geelong surprised me – Geelong is much closer to Melbourne than the inter-state teams, so no long flights are involved. But simply travelling even an hour up the M1, added to the enthusiastic partisanship of Geelong-Melbourne fan rivalry, perhaps explains the strong advantage.

Here’s the R code that performs these steps:

* load in functionality for the analysis
* download the AFL results from 1897 onwards from afltables using the fitzRoy package and store in the object r (for “results”)
* draw the chart of home win rates per season
* estimate and plot teams’ recent-decades home advantage
* create a new r2 object with home and away advantage and disadvantage adjustments for probabilities

*Post continues after code extract*

#================Setup and get data============

library(tidyverse)

library(scales)

library(fitzRoy) # for reading AFL scores

library(frs) # for Elo functions

library(Cairo)

library(ggrepel)

library(lubridate)

library(foreach)

library(doParallel)

library(knitr)

library(clipr)

the\_caption <- "Source: afltables via fitZroy; analysis by freerangestats.info"

update\_geom\_defaults("line", list(colour = "steelblue"))

results <- get\_match\_results()

# reshape and organise the data:

r <- results %>%

rename\_all(tolower) %>%

rename\_all(function(x){gsub("\\.", "\_", x)}) %>%

select(home\_team, away\_team, date, game, margin) %>%

gather(location, team, -date, -game, -margin) %>%

mutate(location = gsub("\_team", "", location),

margin = ifelse(location == "home", margin, -margin)) %>%

mutate(winner = margin > 0) %>%

arrange(date) %>%

mutate(starting\_elo = 1500,

new\_elo = 1500) %>%

group\_by(game) %>%

# sequence so the winner is the first listed each time for each game:

arrange(game, desc(winner)) %>%

ungroup() %>%

mutate(season = year(date)) %>%

group\_by(season, team) %>%

mutate(round = 1:n()) %>%

ungroup()

#==================analysis of the home team advantage===================

# Proportion of home winners over full range of history:

r %>%

filter(location == "home") %>%

group\_by(season) %>%

summarise(prop\_home = mean(winner)) %>%

ggplot(aes(x = season, y = prop\_home)) +

geom\_line() +

scale\_y\_continuous("Percentage of games won by the home team", label = percent\_format(accuracy = 1)) +

labs(caption = the\_caption,

x = "") +

ggtitle("Home teams win around 59 percent of the time in the AFL")

# Home team advantage from 1997 onwards - the modern era:

d <- r %>%

group\_by(team) %>%

summarise(start = min(date),

end = max(date),

home\_wins = mean(winner[location == "home" & date > "1997-01-01"]),

away\_wins = mean(winner[location == "away" & date > "1997-01-01"]),

home\_adv = (home\_wins - away\_wins) / 2) %>%

arrange(desc(home\_adv)) %>%

filter(end > "2000-01-01")

d %>%

ggplot(aes(x = away\_wins, y = home\_wins, label = team, colour = home\_adv)) +

geom\_abline(intercept = 0, slope = 1) +

geom\_segment(aes(xend = away\_wins, yend = away\_wins)) +

geom\_text(hjust = 0) +

scale\_x\_continuous(label = percent\_format(accuracy = 1)) +

scale\_y\_continuous(label = percent\_format(accuracy = 1)) +

scale\_color\_viridis\_c("Home\nadvantage",

option = "D", direction = -1, label = percent\_format(accuracy = 1),

breaks = c(6,8,10,12,14) / 100) +

ggtitle("Wins at home compared to away in the AFL, 1997 to present",

"All teams win more at home than away.") +

labs(x = "Proportion of away games won",

y = "Proportion of home games won",

caption = the\_caption) +

theme(legend.position = "right") +

coord\_flip() +

expand\_limits(y = c(0.2, 0.85),

x= c(0.15, 0.55))

adjustments <- d %>%

mutate(home = home\_adv / 2,

away = -home\_adv / 2) %>%

select(team, home, away) %>%

gather(location, location\_adjustment, -team)

r2 <- r %>%

left\_join(adjustments, by = c("team", "location")) %>%

mutate(location\_adjustment = replace\_na(location\_adjustment, 0),

predicted\_winner = NA)

**Choosing a better set of parameters**

Seeing as I had to revisit my predictive method to adjust for home and away advantage/disadvantage, I decided to also take a more systematic approach to some parameters that I had set either arbitrarily or without noticing they were parameters in last week’s post. These fit into two areas:

* The FIBS-based Elo rating method I use requires a “match length” parameter. Longer matches mean the better player is more likely to win, and also lead to larger adjustments in Elo ratings once the result is known. I had used the winning margin as a proxy/equivalent for match length, reasoning that a team that won by a large amount had shown their superiority in a similar way to a backgammon player winning a longer match. But should a 30 point margin count as match length of 30 (ie equivalent to points) or 5 (equivalent to goals) or something in-between? And what margin or “match length” should I use for predicting future games, where even a margin of 1 is enough to win? And even more philosophically, is margin really related to the backgammon concept of match length in a linear way, or should there be discounting of larger margins? Today, I created three parameters to scale the margin, to choose a margin for predicting future matches, and put the scaled margin to the power of a number from zero to one for non-linearity.
* Elo ratings are by the nature sticky and based on a player/team’s whole career. I had noted that I got better performance in predicting the 2018 season by restarting all teams’ ratings at 1500 at the beginning of the 2017 season. But this is fairly crude. The FIBS Elo rating method has a parameter for teams’ past experience which in effect lets you control how responsive the rating is to new information (the idea being to help new players in a competition quickly move from 1500 up or down to their natural level). I have now added to this with a “new round factor” which shrinks ratings for the first game of the season towards 1500, effectively discounting past experience

Here’s the code that describes and defines those parameters a bit better, in an enhanced version of my afl\_elos() function I introduced last week.

*Post continues after code extract*

#---------------Function to determine ratings for one particular set of parameters-----------------

#' Calculate Elo ratings for a historical set of AFL results

#'

#' @param r a data frame with columns including date, team, game, starting\_elo, new\_elo

#' @param sc scaling factor for dividing the winning margin by to convert it to "match length"

#' for Elo calculation purposes (FIBS method requires a match length)

#' @param pred\_margin the prediction margin in points to use as match length for prediction purposes.

#' Note that this gets divided by sc so it is on the same scale as the historical margins.

#' @param margin\_power parameter that scaled prediction margin is put to the power of in the match length

#' calculation. 0 will mean all match lengths are 1 (so sc and pred\_margin make no difference)

#' @param experience how much experience (sum of previous match length) teams are presumed to have, for the

#' FIBS-style Elo rating. Numbers below 400 make the rating adjustments more responsive to recent results.

#' @param new\_round\_factor how much to shrink Elo ratings towards 1500 in the first match of each season.

#' 1 means no shrinkage, 0 means every team starts the season fresh with a rating of 1500 regardless of

#' past performance.

afl\_elos <- function(r, sc = 1, pred\_margin = 30, margin\_power = 1, experience = 100, new\_round\_factor = 1){

r <- r %>%

group\_by(game) %>%

arrange(date, game, desc(winner))

all\_games <- unique(r$game)

# This seems inherently iterative so perhaps a loop is the logical way to do it:

for(g in all\_games){

this\_game <- r[r$game == g, ] %>%

mutate(starting\_elo = ifelse(round == 1,

(starting\_elo - 1500) \* new\_round\_factor + 1500,

starting\_elo))

# er: calculate elo rating arising from this game to use for changes in ratings

er <- elo\_rating(a = this\_game[1, "starting\_elo"],

b = this\_game[2, "starting\_elo"],

winner = "a",

ml = (round(this\_game[1, "margin"] / sc) ^ margin\_power),

axp = experience,

bxp = experience,

a\_adv = this\_game[1, "location\_adjustment"],

b\_adv = this\_game[2, "location\_adjustment"])

# er: calculate elo rating arising from this game to use for the prediction, for use in

# measureing prediction success

er2 <- elo\_rating(a = this\_game[1, "starting\_elo"],

b = this\_game[2, "starting\_elo"],

winner = "a",

ml = (round(pred\_margin / sc) ^ margin\_power),

axp = experience,

bxp = experience,

a\_adv = this\_game[1, "location\_adjustment"],

b\_adv = this\_game[2, "location\_adjustment"])

r[r$game == g, "new\_elo"] <- unlist(c(er$a, er$b))

r[r$game == g, "predicted\_winner"] <- c(er2$winproba >= 0.5, er2$winproba < 0.5)

r <- r %>%

group\_by(team) %>%

mutate(starting\_elo = lag(new\_elo, default = 1500)) %>%

ungroup()

}

r <- r %>%

mutate(season = lubridate::year(date)) %>%

group\_by(game) %>%

mutate(successful\_prediction = predicted\_winner == winner) %>%

ungroup()

return(r)

}

This function is now pretty slow when run on the whole AFL history form 1897. Unlike last week, it calls the underlying frs::elo\_rating() function for each game twice – once (the object er above) to determine the ratings after the match outcome is known, and once to determine the prediction of the match’s result, for benchmarking purposes (the object er2). Last week I didn’t need to use elo\_rating() twice, because the prediction of the winner was as simple as choosing the team with the highest Elo rating going in to the match. Now, we have to calculate the actual probability of winning, adjusted for home and away advantage and disadvantage. This calculation depends on the parameter choices that impact on converting margin to “match length” and what winning margin we base our predictions on, so the calculation is an additional one to the change in rating that came about from the actual result of the game.

There are doubtless efficiencies that could be made, but I’m not enthused to spend too much time refactoring at this point…

I have no theories and hardly any hunches on what parameter combinations will give the best performance, so the only way to choose a set is to try many and pick the one that would have worked best in predicting AFL games to date. I defined about 2,500 combinations of parameters, removed some that were effective duplicates (because if margin\_power is 0, then the value of sc and pred\_margin are immaterial) and for the purposes of this blog ran just 100 random prediction competitions, based on the games from 1950 onwards. With each individual run taking five minutes or more, I used parallel processing to do 7 runs simultaneously and get the total time for those 100 runs down to about 90 minutes, all I was prepared to do today for the purposes of this blog. I might run it for a longer period of time overnight later.

The top twenty parameter sets from this competition are in the table below. The best combination of factors led to an overall prediction success of about 69%, which is better than last week’s 65%, the crude “always pick the home team” success of 59% and a coin flip of 50%; but not as much better as I hoped. Clearly picking these winners is hard – AFL is more like backgammon or poker in terms of predicting outcomes than it is like chess.

| **sc** | **pred\_margin** | **margin\_power** | **experience** | **new\_round\_factor** | **success\_rate** |
| --- | --- | --- | --- | --- | --- |
| 1 | 30 | 0.8333333 | 100 | 0.4 | 0.6853485 |
| 1 | 20 | 0.8333333 | 200 | 0.8 | 0.6839260 |
| 1 | 20 | 1.0000000 | 100 | 0.6 | 0.6828829 |
| 1 | 10 | 0.8333333 | 0 | 0.4 | 0.6822191 |
| 1 | 20 | 0.6666667 | 100 | 0.8 | 0.6812707 |
| 3 | 20 | 0.8333333 | 0 | 0.8 | 0.6806069 |
| 1 | 20 | 1.0000000 | 400 | 0.8 | 0.6785206 |
| 3 | 10 | 1.0000000 | 0 | 0.8 | 0.6777620 |
| 1 | 20 | 0.5000000 | 0 | 0.8 | 0.6776671 |
| 1 | 10 | 1.0000000 | 300 | 0.8 | 0.6774775 |
| 3 | 30 | 0.8333333 | 200 | 0.6 | 0.6762447 |
| 1 | 10 | 1.0000000 | 300 | 0.6 | 0.6747274 |
| 1 | 20 | 0.8333333 | 400 | 0.6 | 0.6676150 |
| 3 | 10 | 0.6666667 | 100 | 0.8 | 0.6668563 |
| 6 | 20 | 1.0000000 | 100 | 1.0 | 0.6661925 |
| 1 | 30 | 0.5000000 | 100 | 1.0 | 0.6634424 |
| 1 | 10 | 0.5000000 | 0 | 0.4 | 0.6629682 |
| 3 | 20 | 0.6666667 | 100 | 1.0 | 0.6617354 |
| 6 | 20 | 0.8333333 | 100 | 0.6 | 0.6592698 |
| 6 | 30 | 0.6666667 | 200 | 0.8 | 0.6570887 |

The best models had a modest shrinkage of ratings towards 1500 (new\_round\_factor of 0.4 to 0.8, compared to 0 which would mean everyone starting at 1500 in each round 1); and modest if any non-linearity in the conversion of winning margin to a notional “match length”. They had relatively low levels of “experience”, effectively increasing the importance of recent results and downplaying long term momentum; while treating match results in points (sc = 1) and predicting based on a relatively large margin.

I only had time to try a random sample of parameter combinations, and would be very lucky indeed if I have ended up with the best set. How confident can I be that I’ve got something close enough? Here’s the distribution of success rates for that post 1950 series:

Without over-thinking it, it’s reasonable to infer a few more extreme values on the right are possible if we looked at the full set of parameters; but that they wouldn’t be *that much* more successful. It’s certainly good enough for a workplace footy tipping competition.

Here’s the predictive success of the best model over time, now applied to the full range of data not just the post 1950 period for which it was optimised:

… and the code that did the above “parameter competition”, using the foreach and doParallel R packages for parallel processing to bring the elapsed time down to reasonable levels:

*Post continues after code extract*

#---------------Define all possible combinations of parameters---------------------

set.seed(123)

params <- expand.grid(

sc = c(1, 3, 6),

pred\_margin = c(1, 10, 20, 30),

margin\_power = 0:6 / 6,

experience = 0:4 \* 100,

new\_round\_factor = 0:5 / 5

) %>%

# if margin\_power is 0 then sc and pred\_margin make no difference, so we can drop a few

# paramter combinations:

mutate(sc = ifelse(margin\_power == 0, 1, sc),

pred\_margin = ifelse(margin\_power == 0, 1, margin\_power)) %>%

distinct() %>%

# sort in random order, as we're not going to have time to run all combinations

mutate(rnd = runif(n())) %>%

arrange(rnd) %>%

mutate(row\_num = 1:n()) %>%

select(-rnd)

#-------------------------Run the model with a subset of the combinations of parameters------------------

cluster <- makeCluster(7) # only any good if you have at least 7 processors :)

registerDoParallel(cluster)

clusterEvalQ(cluster, {

library(foreach)

library(tidyverse)

library(frs)

})

clusterExport(cluster, c("r2", "afl\_elos", "params"))

# 519 seconds for 10 sets of parameters. So can do about 1.1 per minute (with 7 processors); 180 will take 3 hours.

system.time({

suppressWarnings(rm(res))

res60 <- foreach(i = 1:100, .combine = rbind) %dopar% {

x <- params[i, ]

success\_rate <- r2 %>%

filter(season >= 1950) %>%

afl\_elos(sc = x$sc,

pred\_margin = x$pred\_margin,

margin\_power = x$margin\_power,

experience = x$experience,

new\_round\_factor = x$new\_round\_factor) %>%

filter(location == "home") %>%

summarise(sp = mean(successful\_prediction)) %>%

pull(sp)

return(data.frame = data.frame(row\_num = i, success\_rate = success\_rate))

}

})

stopCluster(cluster)

#------------------Examine results------------------------------

params\_with\_res <- params %>%

left\_join(res, by = "row\_num") %>%

filter(!is.na(success\_rate)) %>%

arrange(desc(success\_rate)) %>%

select(-row\_num)

params\_with\_res%>%

slice(1:20) %>%

kable() %>%

write\_clip()

ggplot(params\_with\_res, aes(x = success\_rate)) +

geom\_density(fill = "steelblue", alpha = 0.5) +

geom\_rug() +

ggtitle("Distribution of success rates in parameter contest")

best\_params <- params\_with\_res[1, ]

elos\_best <- r2 %>%

afl\_elos(sc = best\_params$sc,

pred\_margin = best\_params$pred\_margin,

margin\_power = best\_params$margin\_power,

experience = best\_params$experience,

new\_round\_factor = best\_params$new\_round\_factor)

elos\_best %>%

filter(location == "away" & date < "2019-01-01") %>%

group\_by(season) %>%

summarise(successful\_prediction = mean(successful\_prediction)) %>%

ggplot(aes(x = season, y = successful\_prediction)) +

geom\_line() +

scale\_y\_continuous("Percentage of successful predictions", label = percent\_format(accuracy = 1)) +

ggtitle("Predictions of past AFL games",

"Using the best combination found of parameters for Elo ratings and shrinkage during the off-season.

Prediction success of 70% is difficult to achieve in the modern era.") +

labs(caption = the\_caption, x = "")

**This weeks’ predictions are…**

To turn this model into my tips for this week, I need to extract the final Elo ratings from the best model, join them with the actual fixture and then use the model to predict actual probabilities of winning. Here’s what I get:

| **home** | **away** | **home\_elo** | **away\_elo** | **home\_adjustment** | **away\_adjustment** | **final\_prob** | **winner** | **fair\_returns\_home** | **fair\_returns\_away** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Richmond | Collingwood | 1623.362 | 1552.782 | 0.0392339 | -0.0305072 | 0.6527701 | Richmond | 1.531933 | 2.879936 |
| Sydney | Adelaide | 1506.489 | 1476.328 | 0.0467485 | -0.0632875 | 0.6457876 | Sydney | 1.548497 | 2.823164 |
| Essendon | St Kilda | 1507.538 | 1404.657 | 0.0505882 | -0.0428714 | 0.7132448 | Essendon | 1.402043 | 3.487295 |
| Port Adelaide | Carlton | 1544.746 | 1340.797 | 0.0571146 | -0.0257644 | 0.8077331 | Port Adelaide | 1.238033 | 5.201104 |
| Geelong | Melbourne | 1552.818 | 1521.317 | 0.0747884 | -0.0402286 | 0.6523510 | Geelong | 1.532917 | 2.876464 |
| West Coast | GWS | 1543.355 | 1565.494 | 0.0699069 | -0.0648148 | 0.6084577 | West Coast | 1.643499 | 2.554003 |
| North Melbourne | Brisbane Lions | 1461.086 | 1483.270 | 0.0406327 | -0.0558655 | 0.5701814 | North Melbourne | 1.753828 | 2.326563 |
| Hawthorn | Footscray | 1567.933 | 1482.579 | 0.0490421 | -0.0383632 | 0.6873887 | Hawthorn | 1.454781 | 3.198861 |
| Gold Coast | Fremantle | 1382.276 | 1483.175 | 0.0402515 | -0.0729052 | 0.4955912 | Fremantle | 2.017792 | 1.982519 |

That final\_prob column is the estimated probability of the home team winning.

As you can see, I translate my probabilities into a “fair return”, which I’m using to scan for opportunities with poorly chosen odds from the bookies. These opportunities don’t arrive very often as the bookies are professionals, but when they are paying 50% more than the model predicts to be “fair” I’m going to punt $5 and we’ll see how we go at the end of the season. So far I’m $26 up from this strategy but it’s early days and I’m far from assured the luck will continue.

Judging from the tips and odds by the public, the only controversial picks in the above are for North Melbourne to beat Brisbane and Gold Coast to be nearly a coin flip in contest with Fremantle. In both cases my algorithm is tipping a home advantage to equalise the comparative relative strength of the away team. For the North Melbourne match, the bookies agree with me, whereas the tippers on tipping.afl.com.au are going for a Brisbane win, so I think we can say that reasonable people disagree about the outcome there and it is uncertain. For the other match, I have grave doubts about Gold Coast’s chances against Fremantle (who had a stellar victory last weekend), but am inclined to think the $3.50 return bookies are offering to pay for a Gold Coast win is over-generous and underestimating how much Fremantle struggle when playing away from home. So that’s my recommended match to watch for a potential surprise outcome.

At the time of writing, the first two of these predictions in my table above have already gone astray (for me, the average punters and the average tippers) in the Thursday and Friday night matchs, as 2019 continues its run of surprise results. Collingwood and Adelaide both pulled off against-the-odds wins against teams that were both stronger on paper and playing at home. I won’t say my predictions were “wrong”, because when you say something has a 0.6 chance of happening and it doesn’t, there’s a good chance you were just unlucky, not wrong.

But as they say, prediction is hard, particularly about the future.

Final chunk of R code for today – converting the model into predictions for this round:

#==================Predictions for this round!=====================

elos\_latest <- elos\_best %>%

group\_by(team) %>%

arrange(desc(date)) %>%

slice(1) %>%

ungroup() %>%

arrange(desc(new\_elo)) %>%

select(team, elo = new\_elo)

fixture <- tibble(

home = c("Richmond", "Sydney", "Essendon", "Port Adelaide", "Geelong", "West Coast",

"North Melbourne", "Hawthorn", "Gold Coast"),

away = c("Collingwood", "Adelaide", "St Kilda", "Carlton", "Melbourne", "GWS", "Brisbane Lions",

"Footscray", "Fremantle")

)

fixture %>%

left\_join(elos\_latest, by = c("home" = "team")) %>%

rename(home\_elo = elo) %>%

left\_join(elos\_latest, by = c("away" = "team")) %>%

rename(away\_elo = elo) %>%

left\_join(filter(adjustments, location == "home"), by = c("home" = "team")) %>%

rename(home\_adjustment = location\_adjustment) %>%

select(-location) %>%

left\_join(filter(adjustments, location == "away"), by = c("away" = "team")) %>%

rename(away\_adjustment = location\_adjustment) %>%

select(-location) %>%

mutate(final\_prob = elo\_prob(home\_elo, away\_elo,

ml = (best\_params$pred\_margin / best\_params$sc) ^ best\_params$margin\_power,

a\_adv = home\_adjustment,

b\_adv = away\_adjustment),

winner = ifelse(final\_prob > 0.5, home, away),

fair\_returns\_home = 1/final\_prob,

fair\_returns\_away = 1/ (1- final\_prob)) %>%

kable() %>%

write\_clip()

That’s all.

Here’s the R packages used in producing this post:

thankr::shoulders() %>%

mutate(maintainer = str\_squish(gsub("<.+>", "", maintainer))) %>%

group\_by(maintainer) %>%

summarise(`Number packages` = sum(no\_packages),

packages = paste(packages, collapse = ", ")) %>%

knitr::kable() %>%

clipr::write\_clip()

| **maintainer** | **Number packages** | **packages** |
| --- | --- | --- |
| Hadley Wickham | 15 | assertthat, dplyr, forcats, ggplot2, gtable, haven, httr, lazyeval, modelr, plyr, rvest, scales, stringr, tidyr, tidyverse |
| R Core Team | 12 | base, compiler, datasets, graphics, grDevices, grid, methods, parallel, stats, tools, utils, nlme |
| Yihui Xie | 5 | evaluate, highr, knitr, rmarkdown, xfun |
| Kirill Müller | 4 | DBI, hms, pillar, tibble |
| Winston Chang | 4 | extrafont, extrafontdb, R6, Rttf2pt1 |
| Gábor Csárdi | 3 | cli, crayon, pkgconfig |
| Jim Hester | 3 | glue, withr, readr |
| Lionel Henry | 3 | purrr, rlang, tidyselect |
| Rich Calaway | 3 | doParallel, foreach, iterators |
| Yixuan Qiu | 3 | showtext, showtextdb, sysfonts |
| Dirk Eddelbuettel | 2 | digest, Rcpp |
| Jennifer Bryan | 2 | cellranger, readxl |
| Jeroen Ooms | 2 | curl, jsonlite |
| Simon Urbanek | 2 | audio, Cairo |
| Achim Zeileis | 1 | colorspace |
| Alex Hayes | 1 | broom |
| Brodie Gaslam | 1 | fansi |
| Charlotte Wickham | 1 | munsell |
| Deepayan Sarkar | 1 | lattice |
| James Day | 1 | fitzRoy |
| James Hester | 1 | xml2 |
| Jeremy Stephens | 1 | yaml |
| Joe Cheng | 1 | htmltools |
| Justin Talbot | 1 | labeling |
| Kamil Slowikowski | 1 | ggrepel |
| Kevin Ushey | 1 | rstudioapi |
| Luke Tierney | 1 | codetools |
| Marek Gagolewski | 1 | stringi |
| Matthew Lincoln | 1 | clipr |
| Max Kuhn | 1 | generics |
| Michel Lang | 1 | backports |
| Patrick O. Perry | 1 | utf8 |
| Peter Ellis | 1 | frs |
| Rasmus Bååth | 1 | beepr |
| Simon Garnier | 1 | viridisLite |
| Stefan Milton Bache | 1 | magrittr |
| Vitalie Spinu | 1 | lubridate |