Partial pooling across seasons

Zane

2023-05-23

# Introduction

# Per-strain (across all seasons) analysis

So our original reason for testing the partial pooling models was to determine whether we could ameliorate the effect of different strain panels used across seasons – due to budget, time, and supply issues, the most recent seasons do not analyze many of the most distant strains so the panel is truncated (see **?@fig-indivLines**).

We can see from the test model that the adaptive pooling model is more similar to the no-pooling than the complete-pooling model, but across seasons there will be more outlying individuals who will be regularized towards the mean.

So we’ll test on the A(H1N1) Michigan 2015 strain again, this time using data from across all of the seasons. First we need to do a bit of data cleaning. I should probably do this in a dedicated script.

clean\_data <-  
 readr::read\_rds(  
 here::here("data", "processed", "distance\_data.rds")  
 )  
  
dat <-  
 clean\_data |>  
 dplyr::mutate(  
 vaccine\_type = stringr::str\_remove(vaccine\_type, "\_vaccine\_fullname$") |>  
 stringr::str\_to\_upper() |>  
 factor()  
 ) |>  
 dplyr::filter(  
 vaccine\_type == strain\_type,  
 # Get only the Ag distance methods that Amanda used.  
 method %in% c("cart\_2d\_post", "p\_epi", "year"),  
 season < 2019  
 ) |>  
 dplyr::mutate(  
 dplyr::across(tidyselect:::where(is.factor), forcats::fct\_drop)  
 )  
  
# Pivot the data so we can fit pre/post models at the same time  
dat\_models <-  
 dat |>  
 dplyr::select(-pretiter, -postiter) |>  
 tidyr::pivot\_longer(  
 cols = c(prevactiter, postvactiter, titerincrease),  
 names\_to = "outcome",  
 values\_to = "y"  
 ) |>  
 # Turn the categorical variables into integer indexes, this is required  
 # for index coding in stan  
 dplyr::mutate(  
 id = uniq\_id |>  
 factor() |>  
 forcats::fct\_inorder() |>  
 as.integer()  
 ) |>  
 # Normalize the antigenic distance measurements within vaccine group  
 dplyr::group\_by(vaccine\_fullname, method) |>  
 dplyr::mutate(  
 norm\_dist = distance / max(distance)  
 ) |>  
 dplyr::ungroup()  
  
# We want to fit a model for every unique combination of:  
# vaccine strain; outcome; distance method.  
# (Models will be fitted ACROSS seasons, not per-season.)  
# We'll do this by nesting the data into strata to get a per-stratum  
# data frame and then mapping the model fitting function over the  
# nested data frames.  
dat\_nested1 <-  
 dat\_models |>  
 dplyr::filter(dose == "SD") |>  
 # Select only the data we need RIGHT NOW to prevent Stan from throwing a fit,  
 # it will often get angry over unused data that is in the wrong format.  
 dplyr::select(  
 # Variables that should go into the Stan model. These DO need to be  
 # processed to be NUMERIC.  
 id = uniq\_id, y, x = norm\_dist,  
 # Stratum/nesting variables -- these do NOT need to be processed.  
 vaccine\_fullname, method, outcome  
 ) |>  
 # Cleaning up the factor variabels that have to become indices  
 # it has to be done in groups to ensure there are no missing levels!  
 dplyr::group\_by(dplyr::across(!c(id, x, y))) |>  
 dplyr::mutate(  
 id = id |> factor() |> forcats::fct\_inorder() |> as.integer()  
 ) |>  
 dplyr::ungroup() |>  
 # This line creates the stratified data frames. Analogous to base R split()  
 # but organized better.  
 tidyr::nest(dat = c(id, x, y))  
  
dat\_nested <-  
 dat\_nested1 |>  
 # rethinking will convert data to a list so we may as well do it ourselves.  
 # This turns each of the subsample data frames into a list of vectors.  
 # Note that some of the Ag distance methods are missing values so we also  
 # need to omit those missing values.  
 dplyr::mutate(dat = purrr::map(dat, \(x) x |> na.omit() |>as.list()))

# Creating the model objects

Fortunately I think it is basically safe to use the same model(s) across all the strata, including all three outcomes. We have a pretty large amount of data so ideally the data will dominate the prior. These models can be found in the Partial-Pooling-Test.qmd document, I’ll fit the complete pooling, no pooling, and partial pooling models for each of the outcome/strain/distance measure combinations.

models <-  
 list(  
 "complete pooling" =  
 alist(  
 y ~ dnorm(mu, s),  
 mu <- a + b \* x,  
 a ~ dnorm(5, 5),  
 b ~ dnorm(0, 5),  
 s ~ dexp(1)  
 ),  
 "no pooling" =  
 alist(  
 y ~ dnorm(mu, s),  
 mu <- a[id] + b[id] \* x,  
 vector[id]:a ~ dnorm(5, 5),  
 vector[id]:b ~ dnorm(0, 5),  
 s ~ dexp(1)  
 ),  
 "partial pooling" =  
 alist(  
 y ~ dnorm(mu, s),  
 mu <- a[id] + b[id] \* x,  
 a[id] ~ dnorm(abar, phi),  
 b[id] ~ dnorm(bbar, psi),  
 abar ~ dnorm(5, 5),  
 bbar ~ dnorm(0, 5),  
 phi ~ dexp(1),  
 psi ~ dexp(1),  
 s ~ dexp(1)  
 )  
 )  
  
models <- rev(models)

# Fitting the models

Here’s the code I ran to generate the models.

# Fitting details  
set.seed(370)  
N <- 10000  
CTRL <-  
 list(  
 seed = 370,  
 adapt\_delta = 0.95  
 )  
  
n\_models <- nrow(dat\_nested)  
  
fit\_a\_model <-  
 function(data, data\_index, model\_index, model\_flist, mm = 3) {  
 # Informational message  
 mn <- data\_index + n\_models \* (model\_index - 1)  
 paste0("Fitting model ", mn, " of ", n\_models \* mm, "!\n") |>  
 crayon::white() |>  
 crayon::bgRed() |>  
 cat()  
   
 # Fit the next model  
 mdl <- rethinking::ulam(  
 # Use the model formula that we're currently on in the  
 # outside loop  
 flist = model\_flist,  
 # Use the dataframe that we're currently on in the inside loop  
 data = data,  
 chains = 8,  
 cores = 8,  
 iter = 2500,  
 control = CTRL,  
 cmdstan = TRUE  
 )  
   
 # Invisibly call the garbage collector even though many people  
 # say this is pointless  
 zlib::quiet(gc())  
   
 # Return the model  
 return(mdl)  
 }  
  
possibly\_fit\_a\_model <-  
 purrr::possibly(fit\_a\_model, otherwise = "Uh oh!")

# Outside loop: map over models  
  
to\_run <- models[3]  
  
start\_time <- Sys.time()  
res <-  
 purrr::map2(  
 to\_run, seq\_along(to\_run),  
 function(m, ind) {  
 # Informational message  
 paste0("Fitting ", names(models)[ind], " models now!\n") |>  
 crayon::white() |>  
 crayon::bgMagenta() |>  
 cat()  
   
 # Inside loop: map over strata  
 purrr::map2(  
 dat\_nested$dat, seq\_along(dat\_nested$dat),  
 \(d, idx) possibly\_fit\_a\_model(d, idx, ind, m)  
 )  
 }  
 )  
stop\_time <- Sys.time()  
  
readr::write\_rds(  
 res,  
 here::here("Results", "\_Out", "Per-Strain-Models.Rds"),  
 compress = "gz"  
)

When I ran this, it took about 24 hours total to run all the models, so I don’t currently plan to run it again. Instead I’ll load the results from disk just in case and then process the results.

The results file is over 40 gigabytes in size, so while I will temporarily save it in case the models need to be re-run, **only the processed model** **results will be committed to github**.

res <- readr::read\_rds(here::here("Results", "\_Out", "Per-Strain-Models.Rds"))

This part processes the 40gb file into multiple smaller files just in case. On a machine with less vRAM it will be impossible to read in the one large list.

# Breaking it down into smaller files  
purrr::walk(  
 names(res),  
 ~dir.create(here::here("Results", "\_Out", "Models", .x), showWarnings = FALSE)  
)  
  
for (i in 1) {  
 purrr::iwalk(  
 res[[i]],  
 \(x, ix) paste0(  
 here::here("Results", "\_Out", "Models", names(res)[[i]]), "/Model",  
 stringr::str\_pad(ix, width = 2, side = "left", pad = "0"), ".Rds") |>  
 readr::write\_rds(x = x, compress = "gz")  
 )  
}

# Counterfactual simulations

First we need to make the simulated data. Since the ID variable is created separately for each model, unfortunately we will need to make a list of simulated datasets where the IDs correspond to the correct ones for the current model. Then we’ll cross these IDs with a vector of *possible* values for the antigenic distance, which is how we get the counterfactual predictions for the unobserved antigenic distances.

# Make the simulated data  
ag\_dist\_vals <- seq(0, 1, 0.01)  
  
ids\_by\_model <-  
 purrr::map\_int(  
 dat\_nested$dat,  
 \(x) max(x[["id"]])  
 )  
  
# Repeat the data three times (once for each type of model)  
ids\_all\_models <- rep(ids\_by\_model, times = 3)

Now that we have the list of simulated data, we have to do a little bit of a complicated mapping operation – we’ll have to use a nested map like we did when we fit the models, in order to ensure that we repeat correctly for each of the models. So the outside map will be for the three model types, while the inside map2 will simultaneously map over the model results and the simulated data to get the posterior distributions.

Because of the vRAM limitations of a normal computer, we will have to load in each individual model, process the predictions, and then remove the model from the environment. First we need to make a list of the model files to iterate over.

# Listing all of the model files  
file\_names <-  
 purrr::map(  
 paste0(here::here("Results/\_Out/Models"), "/",  
 c("partial pooling", "no pooling", "complete pooling")),  
 \(x) list.files(x, full.names = TRUE)  
 ) |>  
 do.call(what = c)  
  
N <- length(file\_names)

As a first step, we’ll get the diagnostics (conveniently calculated for us by rethinking::precis()) and the posterior samples, and save those to files. This will allow us to avoid the overhead vRAM usage from loading the entire ulam model object that we don’t need.

start\_time <- Sys.time()  
  
# Save the posterior samples  
purrr::walk2(  
 file\_names, seq\_along(file\_names),  
 function(fn, idx) {  
 if (idx >= 145) {  
 # Get the formatted model number  
 num <- stringr::str\_pad(idx, width = 3, side = "left", pad = "0")  
   
 # Read in the next model  
 m <- readr::read\_rds(fn)  
   
 # Get the posterior samples  
 out <- rethinking::extract.samples(m, n = Inf)  
  
 # Save the posterior samples to file  
 readr::write\_rds(  
 out,  
 paste0(here::here("Results/\_Out/samples"), "/Model", num, ".Rds")  
 )  
   
 # Save the precis to file  
 m |>  
 rethinking::precis(depth = 3, warn = FALSE) |>  
 data.frame() |>  
 tibble::rownames\_to\_column("parameter") |>  
 readr::write\_rds(  
 paste0(here::here("Results/\_Out/precis"), "/Model", num, ".Rds")  
 )  
   
 # Cleanup  
 rm(m)  
 invisible(gc())  
 }  
 },  
 .progress = "Getting posterior samples and diagnostics!"  
)  
  
stop\_time <- Sys.time()  
rm(i)

Next we need to process the posterior samples and calculate the linear model. That is,

with all the subscripts that it needs that I’ll probably come back and add later but I am too lazy and busy right now. Since we’re doing Gaussian models, , i.e. . So this is not too difficult to do. The rethinking package outputs the posterior as a list of parameters, where each parameter is represented as a matrix of samples of and with one row per posterior sample and one column per individual id. So we take the column means of and to get the estimates of the mean for each individual.

So, we will simultaneously map over the posterior samples and the simulated data. Then for each of the simulated data frames, we need to map over the rows to get the predictions using the linear model formula. Then we’ll save those predictions to a file and clean up the environment, hopefully preventing any vRAM issues by doing it in this painstaking way.

x\_mat <- matrix(ag\_dist\_vals, nrow = 1)  
i <- 1  
start\_time <- Sys.time()  
sample\_filenames <- list.files("Results/\_Out/samples", full.names = TRUE)  
  
purrr::walk2(  
 sample\_filenames[145:216], ids\_all\_models[145:216],  
 \(fn, d) {  
 # Get the formatted model number  
 num <- stringr::str\_pad(i, width = 3, side = "left", pad = "0")  
   
 out\_name <- paste0(here::here("Results/\_Out/mu"), "/Model", num, ".Rds")  
   
 # Don't do the steps if this model already has been processed  
 if ((!file.exists(out\_name)) | (isTRUE(OVERWRITE))) {  
 # Read in the posterior distribution  
 p <- readr::read\_rds(fn)  
   
 # IF WE ARE ON A COMPLETE POOLING MODEL, we have to transform the  
 # vector of samples to be an array of the correct size so it  
 # can be indexed by ID.  
 if (length(dim(p$a)) == 1) {  
 p$a <- array(p$a, dim = c(length(p$a), d))  
 }  
   
 if (length(dim(p$b)) == 1) {  
 p$b <- array(p$b, dim = c(length(p$b), d))  
 }  
   
 out\_list <- lapply(  
 1:d,  
 \(id) {  
 array(p$a[, id], dim = c(nrow(p$a), ncol(x\_mat))) +  
 p$b[, id] %\*% x\_mat  
 }  
 )  
   
 out <- do.call(cbind, out\_list)  
   
 means <- colMeans(out)  
 PIs <- apply(out, 2, rethinking::PI)  
   
 val <- data.frame(  
 est = means,  
 lwr = PIs[1, ],  
 upr = PIs[2, ],  
 x = rep(ag\_dist\_vals, times = d),  
 id = rep(1:d, each = length(ag\_dist\_vals))  
 )  
   
 # Write this matrix to file  
 readr::write\_rds(  
 val,  
 out\_name  
 )  
   
 # Clean up  
 rm(p, out)  
 invisible(gc())  
   
 }  
   
 # Increment model number  
 i <<- i + 1  
 },  
 .progress = "Calculating conditional mean samples!"  
)  
stop\_time <- Sys.time()

# Marginal predictions

I should have gotten the marginal predictions in the previous loop, but I forgot to. So now it seems more prudent to run a second loop to get the marginal predictions than to modify the previous loop. Reading in the files again adds significant unnecessary overhead, but it’s less than the computational cost of repeating the calculations I already did.

x\_mat <- matrix(ag\_dist\_vals, nrow = 1)  
i <- 1  
  
start\_time <- Sys.time()  
sample\_filenames <- list.files("Results/\_Out/samples", full.names = TRUE)  
  
purrr::walk2(  
 sample\_filenames[145:216], ids\_all\_models[145:216],  
 \(fn, d) {  
 # Get the formatted model number  
 num <- stringr::str\_pad(i, width = 3, side = "left", pad = "0")  
   
 out\_name <- paste0(here::here("Results/\_Out/marg"), "/Model", num, ".Rds")  
   
 # Don't do the steps if this model already has been processed  
 if ((!file.exists(out\_name)) | isTRUE(OVERWRITE)) {  
 # Read in the posterior distribution  
 p <- readr::read\_rds(fn)  
   
 # IF WE ARE ON A COMPLETE POOLING MODEL, we have to transform the  
 # vector of samples to be an array of the correct size so it  
 # can be indexed by ID.  
 if (length(dim(p$a)) == 1) {  
 p$a <- array(p$a, dim = c(length(p$a), d))  
 }  
   
 if (length(dim(p$b)) == 1) {  
 p$b <- array(p$b, dim = c(length(p$b), d))  
 }  
   
 m1 <- get\_marginals(ag\_dist\_vals, p, type = 1)  
 m3 <- get\_marginals(ag\_dist\_vals, p, type = 3)  
   
 out <-  
 dplyr::bind\_rows(  
 "type 1" = clean\_list(m1, ag\_dist\_vals),  
 "type 3" = clean\_list(m3, ag\_dist\_vals),  
 .id = "ci\_type"  
 )  
   
 # Write this df to file  
 readr::write\_rds(  
 out,  
 out\_name  
 )  
   
 # Clean up  
 rm(p, out)  
 invisible(gc())  
   
 }  
   
 # Increment model number  
 i <<- i + 1  
 },  
 .progress = "Calculating conditional mean samples!"  
)  
stop\_time <- Sys.time()

# Housekeeping

OK, now that we have all the means and PIs for each individual, we need to get all the data consolidated together. Since R can store data.frame objects of this size quite efficiently, the resulting object is actually less than a gigebyte in size.

models\_data <-  
 # Make a list of copies of the nested data containing the model identifying  
 # information -- one copy for each of the models.  
 replicate(length(models), list(dat\_nested)) |>  
 # Set the list names to the same order as the models list, so that we  
 # are labelling the data correctly.  
 setNames(names(models)) |>  
 # Combine all three of these replicated datasets into one rbind-ed dataset,  
 # which has a column called 'model' where the entries come from the  
 # name of the list element (which we just set to be the model names in the  
 # correct order).  
 dplyr::bind\_rows(.id = "model") |>  
 # Now read in all of the predictions dataframes and add this as a  
 # list-column to the tibble. Since we are doing everything in the correct  
 # order we have ensured that the predictions for the correct model are  
 # labelled correctly.  
 tibble::add\_column(  
 preds = purrr::map(  
 list.files("Results/\_Out/mu", full.names = TRUE),  
 readr::read\_rds  
 )  
 )  
  
# Now I'll repeat the same code for the marginals because I'm too lazy  
# to think of a better way. And it's easier if the marginals are in their  
# own data frame anyways.  
marginals\_data <-  
 # Make a list of copies of the nested data containing the model identifying  
 # information -- one copy for each of the models.  
 replicate(length(models), list(dat\_nested)) |>  
 # Set the list names to the same order as the models list, so that we  
 # are labelling the data correctly.  
 setNames(names(models)) |>  
 # Combine all three of these replicated datasets into one rbind-ed dataset,  
 # which has a column called 'model' where the entries come from the  
 # name of the list element (which we just set to be the model names in the  
 # correct order).  
 dplyr::bind\_rows(.id = "model") |>  
 # Now read in all of the predictions dataframes and add this as a  
 # list-column to the tibble. Since we are doing everything in the correct  
 # order we have ensured that the predictions for the correct model are  
 # labelled correctly.  
 tibble::add\_column(  
 preds = purrr::map(  
 list.files("Results/\_Out/marg", full.names = TRUE),  
 readr::read\_rds  
 )  
 )

# Save to disk  
fn <- here::here("Results", "\_Out", "combined-preds.Rds")  
if (!file.exists(fn) | isTRUE(OVERWRITE)) {readr::write\_rds(models\_data, fn)}  
models\_data <- readr::read\_rds(fn)  
  
# Save to disk  
fn <- here::here("Results", "\_Out", "combined-marginals.Rds")  
if (!file.exists(fn)| isTRUE(OVERWRITE)) {readr::write\_rds(marginals\_data, fn)}  
marginals\_data <- readr::read\_rds(fn)

OK so now we have a data frame that has all of our means and PIs and whatnot stored together. Since there’s really no reason for us to get back to the original ID scale, I won’t worry about joining back with the original data set. If we wanted to look for emerging patterns in covariates (without formally stratifying by them) we could do that also, but I am not going to do that right now either. Right now I just want to get the plots like Amanda has in figure 2 of her manuscript.

All of these plots will be saved in the file Results/Figures/Per-Strain-Plots in the following subdirectory structure: model-type/distance-measure/VaccineStrain.png or something similar to that.

# Making plots

First I’ll set the ggplot theme so I don’t have to update it individually for every single plot.

ggplot2::theme\_set(  
 ggplot2::theme\_bw() +  
 ggplot2::theme(  
 plot.background = ggplot2::element\_rect(fill = "white", color = "white"),  
 axis.text = ggplot2::element\_text(size = 16, color = "black"),  
 axis.title = ggplot2::element\_text(size = 18),  
 plot.subtitle = ggplot2::element\_text(  
 size = 16, hjust = 0, margin = ggplot2::margin(b = 2)  
 ),  
 plot.title = ggplot2::element\_text(  
 size = 24, hjust = 0, margin = ggplot2::margin(b = 4)  
 ),  
 plot.caption = ggplot2::element\_text(size = 14),  
 strip.text = ggplot2::element\_text(  
 size = 16, hjust = 0.5, margin = ggplot2::margin(b = 2, t = 2)  
 ),  
 panel.spacing = ggplot2::unit(2, "lines"),  
 legend.position = "bottom",  
 legend.text = ggplot2::element\_text(size = 16, color = "black"),  
 legend.title = ggplot2::element\_text(size = 18, color = "black")  
 )  
)

# Unnest the data so we can plot with it. Although I guess we could  
# technically plot with the nested data this is easier and there are  
# ~10M rows so it is small enough that we can do this.  
plt\_data <-  
 models\_data |>  
 tidyr::unnest(preds) |>  
 dplyr::select(-dat)  
  
plt\_marg <-  
 marginals\_data |>  
 tidyr::unnest(preds) |>  
 dplyr::select(-dat)

First I’ll grab the first set of model results that I want and mess with those until the plot looks how I want it to. Then we can worry about faceting and whatnot.

# Poster plots (move this eventually)

plt\_test <-  
 plt\_marg |>  
 dplyr::filter(  
 # TODO FILTER PLOTS THAT USE THIS IN THIS SCRIPT CAUSE I CHANTGED IT  
 vaccine\_fullname %in% c("H1N1-California-2009", "H3N2-Hong Kong-2014"),  
 model == "complete pooling",  
 ci\_type == "type 3"  
 ) |>  
 dplyr::mutate(  
 o = factor(outcome,  
 levels = c("prevactiter", "postvactiter", "titerincrease"),  
 labels = c("log2(Pre-vaccination titer/5)",  
 "log2(Post-vaccination titer/5)",  
 "log2(Titer ratio)")),  
 m = factor(  
 method,  
 levels = c("year", "p\_epi", "cart\_2d\_post"),  
 labels = c(  
 "Year difference",  
 "p-Epitope sequence distance",  
 "Antigenic cartography distance"  
 )  
 )  
 )  
  
plt\_test |>  
 ggplot() +  
 aes(x = x, y = est, ymin = lwr, ymax = upr) +  
 geom\_ribbon(alpha = 0.5, aes(fill = outcome)) +  
 geom\_line(aes(color = outcome)) +  
 facet\_wrap(~method) +  
 scale\_color\_manual(  
 values = c("blue", "orange", "black")  
 ) +  
 scale\_fill\_manual(  
 values = c("blue", "orange", "black")  
 )

ns <-  
 dat\_models |>  
 dplyr::filter(dose == "SD") |>  
 dplyr::mutate(  
 id = gsub("\\d{4}\_id", "", uniq\_id),  
 season,  
 vaccine\_fullname,  
 .keep = "none"  
 ) |>  
 dplyr::distinct() |>  
 dplyr::group\_by(vaccine\_fullname) |>  
 dplyr::count()

# Dose models

This is a quick model that is also stratified by dose.

m\_dose <-  
 alist(  
 y ~ dnorm(mu, s),  
 mu <- a[dose] + b[dose] \* x,  
 a[dose] ~ dnorm(5, 5),  
 b[dose] ~ dnorm(0, 5),  
 s ~ dexp(1)  
 )  
  
dat\_dose <-  
 dat\_models |>  
 dplyr::filter(age >= 65) |>  
 # Select only the data we need RIGHT NOW to prevent Stan from throwing a fit,  
 # it will often get angry over unused data that is in the wrong format.  
 dplyr::select(  
 # Variables that should go into the Stan model. These DO need to be  
 # processed to be NUMERIC.  
 id = uniq\_id, y, x = norm\_dist, dose,  
 # Stratum/nesting variables -- these do NOT need to be processed.  
 vaccine\_fullname, method, outcome  
 ) |>  
 # Clean up the dose factor -- no need to do by groups since the levels  
 # are always the same!  
 dplyr::mutate(  
 dose = dose |> as.integer()  
 ) |>  
 # Cleaning up the factor variables that have to become indices  
 # it has to be done in groups to ensure there are no missing levels!  
 dplyr::group\_by(dplyr::across(!c(id, x, y))) |>  
 dplyr::mutate(  
 id = id |> factor() |> forcats::fct\_inorder() |> as.integer()  
 ) |>  
 dplyr::ungroup() |>  
 # This line creates the stratified data frames. Analogous to base R split()  
 # but organized better.  
 tidyr::nest(dat = c(id, x, y)) |>  
 # rethinking will convert data to a list so we may as well do it ourselves.  
 # This turns each of the subsample data frames into a list of vectors.  
 # Note that some of the Ag distance methods are missing values so we also  
 # need to omit those missing values.  
 dplyr::mutate(dat = purrr::map(dat, \(x) x |> na.omit() |>as.list()))

# TODO REDO WORKFLOW USING PRE-COMPILED CMDSTAN MODELS  
# THIS WOULD SAVE QUITE A BIT OF TIME! BUT NOT TODAY (2022-02-17)  
dose\_model <-  
 cmdstanr::cmdstan\_model(  
 stan\_file = here::here("Stan", "dose-complete-pooling.Stan"),  
 compile = TRUE  
 )  
  
fit\_a\_cmdstan\_model <-  
 function(data, data\_index, model\_index, model\_flist, mm = 3) {  
 # Informational message  
 mn <- data\_index + n\_models \* (model\_index - 1)  
 paste0("Fitting model ", mn, " of ", n\_models \* mm, "!\n") |>  
 crayon::white() |>  
 crayon::bgRed() |>  
 cat()  
   
 mdl <-  
 dose\_model$sample(  
 data = data,   
 seed = 370,   
 chains = 4,   
 parallel\_chains = 4,  
 iter\_warmup = N %//% 2,  
 iter\_sampling = N %//% 2,  
 adapt\_delta = 0.95,  
 refresh = 1000 # print update every 500 iters  
 )  
   
 # Invisibly call the garbage collector even though many people  
 # say this is pointless  
 zlib::quiet(gc())  
   
 # Return the model  
 return(mdl)  
 }  
  
possibly\_fit\_a\_cmdstan\_model <-  
 purrr::possibly(fit\_a\_model, otherwise = "Uh oh!")

# Fit the dose models  
set.seed(370)  
n\_models <- nrow(dat\_dose)  
start\_time <- Sys.time()  
  
dose\_res <-  
 purrr::walk2(  
 dat\_dose$dat, seq\_along(dat\_dose$dat),  
 \(d, idx) {  
 # Create the file path to save to  
 num <- stringr::str\_pad(idx, width = 3, side = "left", pad = "0")  
 out\_name <- paste0(here::here("Results/\_Out/Dose-Models/Fit"),  
 "/Model", num, ".Rds")  
   
 if (!file.exists(out\_name) | OVERWRITE) {  
   
 # Fit the model  
 fit <- possibly\_fit\_a\_model(d, idx, 1, m\_dose, mm = 1)  
   
 # Save the file  
 readr::write\_rds(fit, out\_name, compress = "gz")  
   
 # Clean up  
 invisible(gc())  
 rm(fit)  
 }  
 }  
 )  
  
stop\_time <- Sys.time()

Get the posterior predictions from the dose models

x\_mat <- matrix(ag\_dist\_vals, nrow = 1)  
# TODO see if we can do with link/sim if we clean up each time?  
start\_time <- Sys.time()  
sample\_filenames <-  
 list.files(  
 "Results/\_Out/Dose-Models/Fit",  
 full.names = TRUE  
 )  
  
 # Setup file name  
 get\_fn <- function(dir) {  
 paste0(  
 here::here("Results/\_Out/Dose-Models"),  
 "/", dir, "/Model", num, ".Rds"  
 )  
 }  
  
purrr::pwalk(  
 list(  
 fn = sample\_filenames,  
 idx = seq\_along(sample\_filenames)  
 ),  
 \(fn, idx) {  
 # Get the formatted model number  
 num <- stringr::str\_pad(idx, width = 3, side = "left", pad = "0")  
   
 # Read in the current model  
 m <- readr::read\_rds(fn)  
   
 # Get the diagnostics  
 pc <-  
 m |>  
 rethinking::precis(depth = 3, warn = FALSE) |>  
 data.frame() |>  
 tibble::rownames\_to\_column("parameter")  
 readr::write\_rds(pc, get\_fn("Precis"))  
   
 # Get the prior samples  
 # prior <- rethinking::extract.prior(m, n = 1250, chains = 8, cores = 8)  
 # readr::write\_rds(prior, get\_fn("Prior"))  
   
 # Get the posterior samples  
 post <- rethinking::extract.samples(m, n = Inf)  
 readr::write\_rds(post, get\_fn("Post"))  
   
 # PREDS ARE FOR COMPLETE POOLING WILL NEED TO UPDATE FOR MULTILEVEL!!  
   
 # Get the individual predictions  
 out\_list <- lapply(  
 1:2,  
 \(dose) {  
 array(post$a[, dose], dim = c(nrow(post$a), ncol(x\_mat))) +  
 post$b[, dose] %\*% x\_mat  
 }  
 )  
   
 out <- do.call(cbind, out\_list)  
   
 means <- colMeans(out)  
 PIs <- apply(out, 2, rethinking::PI)  
   
 val <- data.frame(  
 est = means,  
 lwr = PIs[1, ],  
 upr = PIs[2, ],  
 x = rep(ag\_dist\_vals, times = 2),  
 dose = rep(c("SD", "HD"), each = length(ag\_dist\_vals))  
 )  
   
 readr::write\_rds(val, get\_fn("Preds"))  
   
 # For this model there are no marginal predictions, the individual  
 # predictions are all exactly the same. But when I implement the  
 # other models I'll have to fix that I guess.  
   
 # Clean up  
 rm(m, pc, post, out\_list, out, means, PIs, val)  
 invisible(gc())  
 },  
 .progress = "Processing model fits!"  
)  
stop\_time <- Sys.time()

now we need to process the dose predictions data

dm\_dose <-  
 dat\_dose |>  
 # Now read in all of the predictions dataframes and add this as a  
 # list-column to the tibble. Since we are doing everything in the correct  
 # order we have ensured that the predictions for the correct model are  
 # labelled correctly.  
 tibble::add\_column(  
 preds = purrr::map(  
 list.files("Results/\_Out/Dose-Models/Preds", full.names = TRUE),  
 readr::read\_rds  
 )  
 )  
  
dm\_data <-  
 dm\_dose |>  
 tidyr::unnest(preds)  
  
readr::write\_rds(dm\_dose, here::here("Andreas-Poster-Plots/Dm-Dose.Rds"))

dm\_plot <-  
 dm\_data |>  
 dplyr::filter(  
 #age >= 65,  
 vaccine\_fullname == "H1N1-California-2009",  
 outcome == "titerincrease"  
 ) |>  
 dplyr::mutate(  
 o = factor(outcome,  
 levels = c("prevactiter", "postvactiter", "titerincrease"),  
 labels = c("log2(Pre-vaccination titer/5)",  
 "log2(Post-vaccination titer/5)",  
 "log2(Titer ratio)")),  
 m = factor(  
 method,  
 levels = c("year", "p\_epi", "cart\_2d\_post"),  
 labels = c(  
 "Year difference",  
 "p-Epitope sequence distance",  
 "Antigenic cartography distance"  
 )  
 ),  
 d = factor(  
 dose,  
 levels = c("SD", "HD"),  
 labels = c("Standard dose", "High dose")  
 )  
 )  
  
doseh1 <-  
 dat\_models |>  
 dplyr::filter(  
 age >= 65,  
 vaccine\_fullname == "H1N1-California-2009",  
 outcome == "titerincrease"  
 ) |>  
 dplyr::mutate(  
 o = factor(outcome,  
 levels = c("prevactiter", "postvactiter", "titerincrease"),  
 labels = c("log2(Pre-vaccination titer/5)",  
 "log2(Post-vaccination titer/5)",  
 "log2(Titer ratio)")),  
 m = factor(  
 method,  
 levels = c("year", "p\_epi", "cart\_2d\_post"),  
 labels = c(  
 "Year difference",  
 "p-Epitope sequence distance",  
 "Antigenic cartography distance"  
 )  
 ),  
 d = factor(  
 dose,  
 levels = c("SD", "HD"),  
 labels = c("Standard dose", "High dose")  
 )  
 ) |>  
 ggplot2::ggplot() +  
 ggplot2::aes(x = norm\_dist, y = y, color = d) +  
 ggplot2::geom\_point(  
 alpha = 0.5,  
 size = 1,  
 position = ggplot2::position\_jitter(width = 0.01, height = 0.25, seed = 370)  
 ) +  
 ggplot2::geom\_ribbon(  
 data = dm\_plot,  
 mapping = ggplot2::aes(  
 x = x, y = est, ymin = est - 0.05, ymax = est + 0.05, group = d  
 ),  
 alpha = 0.25,  
 color = "white"  
 ) +  
 ggplot2::geom\_line(  
 data = dm\_plot,  
 mapping = ggplot2::aes(  
 x = x, y = est, color = d  
 ),  
 linewidth = 1.5,  
 alpha = 0.85  
 ) +  
 ggplot2::facet\_grid(cols = ggplot2::vars(m)) +  
 ggplot2::scale\_color\_manual(values = c("#592C88", "#009E73")) +  
 ggplot2::scale\_x\_continuous(  
 limits = c(-0.05, 1.05),  
 breaks = seq(0, 1, 0.25),  
 expand = ggplot2::expansion(0, 0)  
 ) +  
 ggplot2::scale\_y\_continuous(  
 limits = c(-6, 8.25),  
 breaks = c(-5, 0, 5, 10),  
 expand = ggplot2::expansion(0, 0)  
 ) +  
 ggplot2::labs(  
 x = NULL,  
 y = "Titer increase",  
 color = NULL,  
 title = "H1N1-California-2009"  
 ) +  
 ggplot2::guides(  
 color = ggplot2::guide\_legend(override.aes = list(alpha = 1, size = 3))  
 )  
  
doseh3 <-  
 dat\_models |>  
 dplyr::filter(  
 age >= 65,  
 vaccine\_fullname == "H3N2-Hong Kong-2014",  
 outcome == "titerincrease"  
 ) |>  
 dplyr::mutate(  
 o = factor(outcome,  
 levels = c("prevactiter", "postvactiter", "titerincrease"),  
 labels = c("log2(Pre-vaccination titer/5)",  
 "log2(Post-vaccination titer/5)",  
 "log2(Titer ratio)")),  
 m = factor(  
 method,  
 levels = c("year", "p\_epi", "cart\_2d\_post"),  
 labels = c(  
 "Year difference",  
 "p-Epitope sequence distance",  
 "Antigenic cartography distance"  
 )  
 ),  
 d = factor(  
 dose,  
 levels = c("SD", "HD"),  
 labels = c("Standard dose", "High dose")  
 )  
 ) |>  
 ggplot2::ggplot() +  
 ggplot2::aes(x = norm\_dist, y = y, color = d) +  
 ggplot2::geom\_point(  
 alpha = 0.5,  
 size = 1,  
 position = ggplot2::position\_jitter(width = 0.01, height = 0.25, seed = 370)  
 ) +  
 ggplot2::geom\_ribbon(  
 data = dm\_plot,  
 mapping = ggplot2::aes(  
 x = x, y = est, ymin = est - 0.05, ymax = est + 0.05, group = d  
 ),  
 alpha = 0.25,  
 color = "white"  
 ) +  
 ggplot2::geom\_line(  
 data = dm\_plot,  
 mapping = ggplot2::aes(  
 x = x, y = est, color = d  
 ),  
 linewidth = 1.5,  
 alpha = 0.85  
 ) +  
 ggplot2::facet\_grid(cols = ggplot2::vars(m)) +  
 ggplot2::scale\_color\_manual(values = c("#592C88", "#009E73")) +  
 ggplot2::scale\_x\_continuous(  
 limits = c(-0.05, 1.05),  
 breaks = seq(0, 1, 0.25),  
 expand = ggplot2::expansion(0, 0)  
 ) +  
 ggplot2::scale\_y\_continuous(  
 limits = c(-6, 8.25),  
 breaks = c(-5, 0, 5, 10),  
 expand = ggplot2::expansion(0, 0)  
 ) +  
 ggplot2::labs(  
 x = "Normalized antigenic distance",  
 y = NULL,  
 color = NULL,  
 title = "H3N2-Hong Kong-2014"  
 ) +  
 ggplot2::guides(  
 color = ggplot2::guide\_legend(override.aes = list(alpha = 1, size = 3))  
 )  
  
ylab <- doseh1$labels$y  
doseh1$labels$y <- doseh3$labels$y <- " "  
  
png(  
 here::here("Andreas-Poster-Plots/p2.png"),  
 width = 11, height = 8.5, res = 300,  
 units = "in"  
)  
 doseh1 + doseh3 +  
 patchwork::plot\_layout(guides = "collect", ncol = 1) &  
 ggplot2::theme(  
 plot.margin = ggplot2::margin(l = 12, r = 5.5, t = 5.5, b = 5.5)  
 )  
grid::grid.draw(grid::textGrob(ylab, x = 0.02, rot = 90, gp = grid::gpar(fontsize = 18)))  
  
dev.off()

# Slopes, intercepts, and AUCs

dat\_parms <-  
 marginals\_data |>  
 dplyr::filter(model == "complete pooling") |>  
 dplyr::select(-model) |>  
 dplyr::mutate(  
 # Get only the Type 1 CI estimates (it doesn't really matter which for  
 # only the complete pooling models)  
 preds = purrr::map(  
 preds,  
 \(d) d |>  
 dplyr::filter(ci\_type == "type 1") |>  
 dplyr::select(-ci\_type)  
 ),  
 # Iterate over the precis files (that we already saved) to get the linear  
 # model mean parameter estimates for each model  
 parameters = purrr::map(  
 list.files(path = "Results/\_Out/precis", full.names = TRUE)[145:216],  
 \(f) {  
 p <- readr::read\_rds(f)  
 parms <-  
 p |>  
 dplyr::filter(parameter %in% c("a", "b")) |>  
 dplyr::select(parameter, est = mean, lwr = `X5.5.`, upr = `X94.5.`) |>  
 tibble::tibble()  
 }  
 ),  
 # Compute the mean outcome for comparison to AUCS  
 ymean = purrr::map(  
 dat,  
 \(d) ggplot2::mean\_cl\_normal(d$y, conf.int = .89) |>  
 rlang::set\_names(c("est", "lwr", "upr")) |>  
 tibble::tibble()  
 )  
 ) |>  
 dplyr::select(-dat, -preds)

# Compute the AUCs  
  
# This data frame contains the antigenic distance unit cutoffs for weighting  
cutoff\_df <-  
 readr::read\_rds(file = here::here("Results/tables/AU\_cutoff.rds")) |>  
 dplyr::filter(distance\_type == "Strain") |>  
 # Remove the subtype from the beginning of the short\_name  
 dplyr::mutate(  
 short\_name = gsub("^.{4}:", "", vac\_short)  
 ) |>  
 dplyr::select(short\_name, cutoff) |>  
 dplyr::left\_join(  
 # This file contains the conversion between the short name and the long name  
 readr::read\_rds(here::here("data", "processed", "virus\_info.rds")),  
 by = "short\_name"  
 ) |>  
 dplyr::select(vaccine\_fullname = analysis\_name, cutoff)  
  
# First we need to join this DF to the outcomes  
AUC\_comps <-  
 marginals\_data |>  
 dplyr::left\_join(cutoff\_df, by = "vaccine\_fullname") |>  
 # Calculate the AUC for each model -- uses trapezoidal approximation  
 # Consider switching to a better method? Not like it matters for a  
 # straight line though  
 # We use the three different weighting schemes as well.  
 # Finally, the SEs were computed by taking the AUC of the low and high  
 # marginal predictions band. Unclear where this is correct or we would  
 # need to bootstrap on the individual samples or something.  
 dplyr::filter(model == "complete pooling") |>  
 dplyr::select(-model) |>  
 dplyr::mutate(  
 AUC\_unweighted = purrr::map(  
 preds,  
 \(d) tibble::tibble(  
 est = pracma::trapz(d$x, d$est),  
 lwr = pracma::trapz(d$x, d$lwr),  
 upr = pracma::trapz(d$x, d$upr)  
 )  
 ),  
 AUC\_linear = purrr::map(  
 preds,  
 \(d) tibble::tibble(  
 est = pracma::trapz(d$x, d$est \* (1 - d$x)),  
 lwr = pracma::trapz(d$x, d$lwr \* (1 - d$x)),  
 upr = pracma::trapz(d$x, d$upr \* (1 - d$x))  
 )  
 ),  
 AUC\_step = purrr::map2(  
 preds, cutoff,  
 \(d, c) tibble::tibble(  
 est = pracma::trapz(d$x, d$est \* (d$x <= c)),  
 lwr = pracma::trapz(d$x, d$lwr \* (d$x <= c)),  
 upr = pracma::trapz(d$x, d$upr \* (d$x <= c))  
 )  
 ),  
 ) |>  
 dplyr::mutate(  
 AUC = purrr::pmap(  
 list(AUC\_unweighted, AUC\_linear, AUC\_step),  
 \(x, y, z) dplyr::bind\_rows(  
 "unweighted" = x,  
 "linear" = y,  
 "2 AU" = z,  
 .id = "weighting"  
 )  
 ),  
 .keep = "unused"  
 ) |>  
 dplyr::select(-dat, -preds, -cutoff) |>  
 tidyr::unnest(AUC) |>  
 dplyr::mutate(name = paste0("AUC (", weighting, ")"), .keep = "unused")

dat\_table <-  
 dat\_parms |>  
 tidyr::unnest(parameters) |>  
 tidyr::pivot\_wider(  
 names\_from = parameter,  
 values\_from = c(est, lwr, upr),  
 names\_glue = "{parameter}\_{.value}"  
 ) |>  
 tidyr::unnest(ymean, names\_sep = "\_") |>  
 tidyr::pivot\_longer(cols = !(vaccine\_fullname:outcome)) |>  
 tidyr::separate(name, into = c("name", "stat")) |>  
 tidyr::pivot\_wider(names\_from = stat, values\_from = value) |>  
 dplyr::bind\_rows(AUC\_comps) |>  
 dplyr::left\_join(  
 # Get the short names back  
 readr::read\_rds(here::here("data", "processed", "virus\_info.rds")),  
 by = c("vaccine\_fullname" = "analysis\_name")  
 ) |>  
 dplyr::mutate(  
 subtype = substr(vaccine\_fullname, 1, 4),  
 method = factor(  
 method,  
 levels = c("year", "p\_epi", "cart\_2d\_post"),  
 labels = c("Year difference", "p-Epitope", "Antigenic cartography")  
 ),  
 outcome = factor(  
 outcome,  
 levels = c("prevactiter", "postvactiter", "titerincrease"),  
 labels = c("log2 pre-titer",  
 "log2 post-titer",  
 "log2 titer ratio")  
 ),  
 name = factor(  
 name,  
 levels = c("ymean", "AUC (unweighted)", "AUC (linear)", "AUC (2 AU)",  
 "a", "b"),  
 labels = c("Mean outcome",  
 "AUC (unweighted)",  
 "AUC (linear)",  
 "AUC (2 AU)",  
 "Intercept",  
 "Slope")  
 )  
 ) |>  
 dplyr::select(  
 subtype, strain = short\_name, outcome, method, est, lwr, upr, name  
 ) |>  
 dplyr::arrange(subtype, strain, outcome, method, name)  
  
readr::write\_rds(  
 list(dat\_models, plt\_test, dat\_table),  
 file = here::here("Andreas-Poster-Plots/p1data.Rds")  
)

## Make the table for titerincrease

dat\_table |>  
 dplyr::mutate(  
 value = paste0(  
 sprintf("%.2f", est)  
 # " (", sprintf("%.2f", lwr), ", ",  
 # sprintf("%.2f", upr), ")"  
 ),  
 .keep = "unused"  
 ) |>  
 dplyr::filter(  
 outcome == "log2 titer ratio",  
 name %in% c("AUC (unweighted)", "Intercept", "Slope")  
 ) |>  
 dplyr::mutate(  
 name = factor(  
 name,  
 levels = c("Slope", "Intercept", "AUC (unweighted)"),  
 labels = c("Strength", "Breadth", "Total")  
 ),  
 ) |>  
 dplyr::select(  
 -outcome,  
 -c(lwr, upr)  
 ) |>  
 tidyr::pivot\_wider(  
 names\_from = method,  
 values\_from = value,  
 names\_glue = "Distance measure\_{method}"  
 ) |>  
 flextable::flextable() |>  
 flextable::merge\_v(j = ~subtype + strain) |>  
 flextable::valign(j = ~ subtype + strain, valign = "top") |>  
 flextable::align(j = 4:6, align = "center") |>  
 flextable::separate\_header(split = "\_") |>  
 flextable::autofit()

# Weight figure theme

wt\_plot\_df <-  
 dat\_models |>  
 dplyr::filter(  
 vaccine\_fullname == "H1N1-California-2009",  
 outcome == "titerincrease"  
 ) |>  
 list() |>  
 replicate(n = 3) |>  
 setNames(c("unweighted", "linear", "2 AU")) |>  
 dplyr::bind\_rows(.id = "weighting") |>  
 dplyr::mutate(  
 weighting = forcats::fct\_inorder(weighting)  
 ) |>  
 dplyr::mutate(  
 method = factor(  
 method,  
 levels = c("year", "p\_epi", "cart\_2d\_post"),  
 labels = c("Year difference", "p-Epitope", "Antigenic cartography")  
 )  
 )  
  
wt\_plot\_auc <-  
 AUC\_comps |>  
 dplyr::filter(  
 vaccine\_fullname == "H1N1-California-2009",  
 outcome == "titerincrease"  
 ) |>  
 dplyr::mutate(  
 weighting = stringr::str\_extract(name, "AUC \\((.\*)\\)", group = 1),  
 est = paste0("AUC: ", sprintf("%.2f", est)),  
 x = 0.9,  
 y = 4  
 ) |>  
 dplyr::select(method, est, weighting, x, y) |>  
 dplyr::mutate(  
 method = factor(  
 method,  
 levels = c("year", "p\_epi", "cart\_2d\_post"),  
 labels = c("Year difference", "p-Epitope", "Antigenic cartography")  
 ),  
 weighting = forcats::fct\_inorder(weighting)  
 )  
  
wt\_plot\_lines <-  
 dat\_parms |>  
 tidyr::unnest(parameters) |>  
 tidyr::pivot\_wider(  
 names\_from = parameter,  
 values\_from = c(est, lwr, upr),  
 names\_glue = "{parameter}\_{.value}"  
 ) |>  
 tidyr::unnest(ymean, names\_sep = "\_") |>  
 tidyr::pivot\_longer(cols = !(vaccine\_fullname:outcome)) |>  
 tidyr::separate(name, into = c("name", "stat")) |>  
 tidyr::pivot\_wider(names\_from = stat, values\_from = value) |>  
 dplyr::filter(  
 vaccine\_fullname == "H1N1-California-2009",  
 outcome == "titerincrease",  
 name != "ymean"  
 ) |>  
 tidyr::pivot\_wider(  
 names\_from = name,  
 values\_from = c(est, lwr, upr)  
 ) |>  
 list() |>  
 replicate(n = 3) |>  
 setNames(c("unweighted", "linear", "2 AU")) |>  
 dplyr::bind\_rows(.id = "weighting") |>  
 dplyr::mutate(  
 weighting = forcats::fct\_inorder(weighting)  
 ) |>  
 dplyr::mutate(  
 method = factor(  
 method,  
 levels = c("year", "p\_epi", "cart\_2d\_post"),  
 labels = c("Year difference", "p-Epitope", "Antigenic cartography")  
 )  
 ) |>  
 dplyr::select(-vaccine\_fullname, -outcome)  
  
wt\_plot\_gradient <-  
 tibble::tibble(  
 weighting = rep(c("unweighted", "linear", "2 AU"), each = 1001),  
 x = rep(seq(0, 1, 0.001), times = 3),  
 weight = c(  
 rep(1, times = 1001),  
 1 - seq(0, 1, 0.001),  
 as.numeric(seq(0, 1, 0.001) <= 0.4672067)  
 )  
 ) |>  
 dplyr::right\_join(  
 wt\_plot\_lines, by = "weighting", multiple = "all"  
 )  
  
readr::write\_rds(  
 list(  
 "df" = wt\_plot\_df,   
 "auc" = wt\_plot\_auc,  
 "lines" = wt\_plot\_lines,  
 "grad" = wt\_plot\_gradient  
 ),  
 here::here("Andreas-Poster-Plots/p3data.Rds")  
)