Super Learner + `vimp`: Investigating HIV-1 correlates of protection in HVTN 505

Brian Williamson

2021-03-08

Contents

1 Introduction **1**

2 Dataset **2**

3 Correlates of Risk Analysis **3**
3.1 Candidate Variable Sets 3
3.2 Inverse Probability Weights 3
3.3 Super Learning Analysis 6
3.4 Variable Importance 40

4 Results **44**
4.1 Compiling Results and Creating Plots 44
4.2 Super Learning Analysis 49
4.3 Variable Importance Analysis 50

5 Conclusions **56**

6 Appendix **56**

References **63**

1 Introduction

We still do not have a broadly efficacious vaccine against HIV-1. HVTN 505 was a randomized, placebo-controlled trial designed to assess the efficacy of a candidate vaccine regimen in adults. Using data from this trial, we aim to:

1. Build an *estimated optimal surrogate*, a model that best predicts HIV-1 infection risk from immune response marker variables measured at month 7 and a few baseline variables; and
2. Identify the sets of immune response markers that best predict infection and how these sets work together.

We use the Super Learner (van der Laan, Polley, and Hubbard 2007) to address aim 1, and perform a variable importance analysis to address aim 2. Taken together, these aims constitute a correlates of risk (CoR) analysis using these data.

2 Dataset

The month 7 immune markers are cross-classified by the following factors into interpretable sets:

1. Assay type: CD4 T Cell, CD8 T Cell, IgG, IgA, functional (antibody-dependent cellular phagocytosis [ADCP; referred to as `phago`], Fcγ receptors [referred to as `fcrR2a` and `fcrR3a`], and neutralizing antibody [nAb] tier 1)
2. Antigen: any vaccine-matched HIV peptide pool ("Any HIV"), protein-specific vaccine-matched peptide pools ("Any VRC ENV", "Any VRC GAG", "Any VRC NEF", and "Any VRC POL"), Empty Ad5 VRC, CMV, gp120, gp140, V1V2, V3, p24, gp41, C1, C4

A few of the month 7 immune markers had missing values. Single imputation was used to fill in these values. We ignore uncertainty in the imputations in this analysis given the very small amount of missing data.

We performed some data cleaning and a dimension reduction step prior to the statistical analysis. Within each interpretable variable set:

- include quantitative variables and binary "high" vs. "low" variables, defined by above vs. below the median of vaccine recipients. The median is calculated for the cohort of vaccine recipients at risk of HIV-1 infection at month 7; thus, inverse sampling probability weights are used in the calculations.
- each BAMA variable (IgG, IgG3, IgA) is baseline subtracted. For IgG3 the variables are log fold-rise in immune response, whereas IgG and IgA are the log of the difference (month 7 - month 0) on the original scale. In addition, for each BAMA variable month 7 readouts without baseline subtraction are included.
- For ICS variables (CD4, CD8 T cell), magnitude measures are: percent of cells expressing IL2, TNFa, CD154, and/or IFNg; percent of cells expressing IL2 and/or IFNg; percent of cells expressing IL2 and/or TNFa; and percent of cells expressing each of the individual cytokines (TNFa, CD154, IFNg, IL2, IL4). All of the magnitudes are log10 net responses, i.e., background-subtracted and then log10-transformed. Polyfunctionality as measured by the COMPASS polyfunctionality score (proportion of antigen-specific cell subsets detected, weighted by their degree of functionality, where antigen-specific cell subsets are those with a statistically-significant difference in the percent of cytokine-expressing cells in unstimulated vs. stimulated samples). The following cytokines are considered in calculating the polyfunctionality score: TNFa, CD154, IFNg, IL2, IL4, and Granzyme B; however cells expressing Granzyme B only are not counted as "functional". Note functionality scores are not included because they are highly correlated and mostly redundant conceptually with polyfunctionality scores. Both magnitude and polyfunctional score variables are included for study, as the correlations are not uniformly high; they range from ~0.5–0.8 across antigens and T cell subsets and may capture different aspects of T cell response.
- For ICS responses to vaccine-mismatched antigens (Empty Ad5 VRC and CMV antigens), measure responses using the percent of cells expressing IL2, TNFa, CD154, and/or IFNg.
- For variables with a positive response call (IgG, IgG3, IgA, R2a, R3a, phago, Tier 1 nAb) require $>$ 20% vaccine group positive response. For all variables, require a significantly greater response in the vaccine group than the placebo group to include in the analysis (Wilcoxon rank sum test 2-sided p < 0.01). Note that no Tier 1 nAb variables are included because none met the $>$ 20% vaccine group response criterion.
- For all variables in an interpretable variable set (defined by assay and antigen) that are for vaccine-mismatched antigens, aggregate the variables into two scores to be kept for CoR analysis: PC1 and the maximal diversity weighted score (MDWS).

The data are included in the R package HVTN505. The data are available at the Statistical Center for HIV/AIDS Research and Prevention website, in the `correlates` subfolder. Install the package from `HVTN505_2019-4-25.tar.gz` (the version at the time of the final analysis; the current version online is dated 2020-8-4) using the following code:

```
# only run this line if the package isn't already installed; edit the file path
# to point to the directory where the .tar.gz file is located and make sure you have
```

```
# the correct name
# devtools::install_local("HVTN505_2019-4-25.tar.gz")
```

The object `dat.505` contains 189 rows, clinical covariates, and both continuous and dichotomized immune response biomarkers. The object `var.505` contains metadata on the individual immune response biomarkers. The object `score.505` contains metadata on the summary immune response biomarkers.

3 Correlates of Risk Analysis

All models adjust for the same baseline exposure covariates adjusted for by Janes et al. (2017), except race (white, black, Hispanic) is excluded to help avoid sparsity issues: age (years old at enrollment), BMI at enrollment (quantitative value in kg/m^2), and baseline behavioral risk (a weighted average of two binary risk factors identified by Hammer et al. (2013)). All analyses use the same empirical inverse probability sampling weights used by Janes et al. (2017) and Fong et al. (2018). These participant weights are included in the data file (`wt` column in `dat.505` in the `HVTN505` R package).

3.1 Candidate Variable Sets

We consider 11 candidate variable sets in this analysis, defined by assay combination:

1. All markers;
2. IgG + IgA;
3. IgG3;
4. T cells (CD4 and CD8);
5. Functional antibodies (Fx Ab);
6. IgG + IgA and IgG3;
7. IgG + IgA and T cells;
8. IgG + IgA and IgG3 and T cells;
9. IgG + IgA and IgG3 and Fx Ab;
10. T cells and Fx Ab; and
11. No markers.

In all analyses, as we mentioned above, we adjust for three baseline variables: age, BMI, and behavioral risk.

3.2 Inverse Probability Weights

All of the following analyses rely on vaccine recipients who were at risk of HIV-1 infection at Month 7. This is a case-control cohort within the full HVTN 505 analysis dataset. Thus, all of our analyses must account for this sampling. The HVTN 505 correlates dataset `dat.505` contains the weights for the vaccine recipients, which is useful for running the regressions of outcome on various sets of covariates. However, to fully account for the case-control sampling, we need the weights for all vaccine recipients along with an indicator of which vaccine recipients are in the case-control cohort. The following code uses the full HVTN 505 data to obtain weights for each trial participant, which we will use in the subsequent sections.

```
library("dplyr")
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library("tidyr")
library("tibble")
library("readr")
# ---------------------------------------------------------
# Read in the full dataset and make sure that
# strata are computed across ALL participants
# ---------------------------------------------------------
# these data come from the SCHARP ATLAS page for 505,
# https://atlas.scharp.org/cpas/project/HVTN%20Public%20Data/HVTN%20505/begin.view?
# (select primary505_for_sharing.csv within the Fong et al. (2018, JID) folder)
# I've saved this in a data/ subfolder, but change the path to where you've saved it
full_data <- readr::read_csv("data/primary505_for_sharing.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   pub_id = col_character(),
##   cc_strata = col_character(),
##   cc_fullstrata = col_character(),
##   hg_strata = col_character(),
##   site = col_character(),
##   gender = col_character(),
##   racefull = col_character(),
##   BMIcat = col_character(),
##   randdt = col_character(),
##   enrdt = col_character(),
##   DNA1dt = col_character(),
##   DNA2dt = col_character(),
##   DNA3dt = col_character(),
##   Ad5dt = col_character(),
##   lastvacdt = col_character(),
##   lastvstdt = col_character(),
##   lastnegdt = col_character(),
##   dxdt = col_character(),
##   epitdt = col_character(),
##   termdt = col_character()
##   # ... with 1 more columns
## )
```

```
## See spec(...) for full column specifications.
```

```
full_data_with_strata <- full_data %>%
  mutate(hg_strata = ifelse(is.na(hg_strata),
                            paste0(
                              ifelse(trt == 0, "Plac/", "Vacc/"),
                              ifelse(racecc == "White", "White", "Blk_Hisp"),
                              ifelse(trt == 1 & is.na(BMI), "/Missing", ""),
                              ifelse(trt == 1 & BMI < 25 & !is.na(BMI),
                                     "/[18.4, 25)", ""),
                              ifelse(trt == 1 &
                                       25 <= BMI &
                                       BMI < 29.8 &
```

```
                                        !is.na(BMI), "/[25, 29.8)", ""),
                                ifelse(trt == 1 & 29.8 <= BMI & !is.na(BMI),
                                       "/[29.8, 40)", "")
                              ),
                              hg_strata)) %>%
  filter(!is.na(BMI))

# read in table to match pub_id from this dataset with the
# ID in the HVTN505 R package; saved in the 'data' subfolder
pub_id_converter <- readr::read_csv("data/rx_v2.csv") %>%
  select(Ptid, pub_id) %>%
  rename(hvtn505_id = Ptid) %>%
  mutate(hvtn505_id = gsub("-", "", hvtn505_id))
```

```
## Parsed with column specification:
## cols(
##   Ptid = col_character(),
##   pub_id = col_character()
## )
```

```
# update pub id
full_data_with_matched_ids <- full_data_with_strata %>%
  left_join(pub_id_converter, by = "pub_id")
# ------------------------------------------------------
# Compute inverse probability weights
# ------------------------------------------------------
# compute strata
full_data_with_stratuminds <- full_data_with_matched_ids %>%
  mutate(
    stratuminds =
      case_when(
      hg_strata == "Plac/Blk_Hisp" ~ 1,
      hg_strata == "Plac/White" ~ 2,
      hg_strata == "Vacc/Blk_Hisp/[18.4, 25)" ~ 3,
      hg_strata == "Vacc/Blk_Hisp/[25, 29.8)" ~ 4,
      hg_strata == "Vacc/Blk_Hisp/[29.8, 40)" ~ 5,
      hg_strata == "Vacc/White/[18.4, 25)" ~ 6,
      hg_strata == "Vacc/White/[25, 29.8)" ~ 7,
      hg_strata == "Vacc/White/[29.8, 40)" ~ 8
    ),
    stratuminds_vaccs = stratuminds - 2
  )
cc_data <- full_data_with_stratuminds %>%
  filter(casecontrol == 1)
# numbers of cases and controls in each stratum
n0 <- full_data_with_stratuminds %>%
  filter(cc_cohort == 1, HIVwk28preunbl == 0) %>%
  group_by(cc_strata) %>%
  summarize(n0 = n(), .groups = "drop")
n1 <- full_data_with_stratuminds %>%
  filter(cc_cohort == 1, HIVwk28preunbl == 1) %>%
  group_by(cc_strata) %>%
  summarize(n1 = n(), .groups = "drop")
# compute the weights
```

```
weights <- cbind(n0$n0, n1$n1) / table(cc_data$stratuminds,
                                       cc_data$HIVwk28preunbl)
full_data_with_weights <- full_data_with_stratuminds %>%
  mutate(weight =
           ifelse(HIVwk28preunbl == 1,
                  1,
                  weights[, 1][full_data_with_stratuminds$stratuminds]))

# final tibble
Z_plus_weights <- full_data_with_weights %>%
  select(hvtn505_id, trt, HIVpreunbl, age, BMI, bhvrisk, weight) %>%
  rename(ptid = hvtn505_id, Y = HIVpreunbl) %>%
  mutate(ptid = as.numeric(ptid)) %>%
  filter(!is.na(Y), !is.na(age), !is.na(BMI), !is.na(bhvrisk))
```

3.3 Super Learning Analysis

We use the implementation of the Super Learner provided in the `SuperLearner` R package. In a Super Learning task, cross-validation is used to determine the optimal convex combination of a set of candidate learners chosen to minimize a cross-validated risk. This process results in an estimated risk for each of the candidate learners, along with an estimated risk for the traditional cross-validated selector and the optimal convex combination of the individual algorithms. This convex combination is called the Super Learner.

Specifying the set of candidate learners involves two steps: (1) specifying variable screens that reduce the number of variables passed to each algorithm, and (2) specifying a set of algorithms. The screens and algorithms are then combined, and each unique combination of screen and algorithm is applied to the regression problem.

Since we have only 25 events, we constructed aggressive variable screens that allowed a maximum of four marker variables into each candidate learner. In each case, the screen also forced age, BMI, and baseline behavioral risk to be adjusted for in the algorithm. The specific screens used are:

- dynamic range: exclude variables with 20th percentile equal to the 80th percentile
- dynamic range score: exclude all variables with standard deviation in the vaccinees divided by standard deviation in placebo recipients in the lower half of all participants
- lasso with fixed p-value threshold: first run a lasso (adjusting for age, BMI, and behavioral risk); then for each variable with nonzero lasso coefficient, run a logistic regression (adjusting for age, BMI, and behavioral risk) and exclude all variables for which the resulting p-value is above a threshold. We considered thresholds 0.01, 0.05, and 0.1
- high correlation: exclude any pair of variables that has Spearman correlation greater than 0.9.

The specific Super Learner algorithms we considered were:

- generalized linear models;
- generalized linear models with pairwise interactions;
- forward stepwise regression with pairwise interactions;
- boosted decision stumps: boosted trees with maximum depth of one node; and
- the `earth` algorithm of Doksum, Tang, and Tsui (2008).

In each case, we used a logit link function to relate the predictor variables to the risk of HIV-1 infection.

These screens and algorithms are implemented in R using the following code:

```
# create SL screens, algorithm + screen combinations

# -----------------------------------------------------------------------------
```

```r
# SL screens; all models adjust for baseline covariates age, BMI at enrollment,
# baseline behavioral risk
# ------------------------------------------------------------------------------
# screen based on logistic regression univariate p-value < level
rank_univariate_logistic_pval_plus_exposure <- function(Y, X, family,
                                                        obsWeights, id, ...) {
    # logistic regression of outcome on each variable
    listp <- apply(X, 2, function(x, Y, family) {
      summ <- coef(summary(glm(Y ~ x + X$age + X$BMI + X$bhvrisk,
                               family = family, weights = obsWeights)))
        ifelse(dim(summ)[1] > 1, summ[2, 4], 1)
    }, Y = Y, family = family)
    # rank the p-values; give age, BMI, bhvrisk the lowest rank
    # (will always set to TRUE anyways)
    ranked_vars <- rank(listp, ties = "average")
    ranked_vars[names(X) %in% c("age", "BMI", "bhvrisk")] <- 999
    return(ranked_vars)
}
# screen dynamic range: only keep variables with
# 20th percentile != 80th percentile
screen_dynamic_range_plus_exposure <- function(Y, X, family,
                                               obsWeights, id,
                                               nVar = 4, ...) {
  # set all vars to false
  vars <- rep(FALSE, ncol(X))

  # keep only those with dynamic range: 20th percentile != 80th percentile
  x_quantiles <- apply(X, 2, function(x) quantile(x, probs = c(0.2, 0.8)))
  vars <- apply(x_quantiles, 2, function(x) round(x[1], 4) != round(x[2], 4))
  # also keep the first three columns of X (correspond to age, BMI, bhvrisk)
  vars[names(X) %in% c("age", "BMI", "bhvrisk")] <- TRUE
  # keep only a max of nVar immune markers; rank by univariate p-value
  # in a model adjusting for age, BMI, bhvrisk
  X_initial_screen <- X %>%
    select(names(X)[vars], "age", "BMI", "bhvrisk")
  ranked_vars <- rank_univariate_logistic_pval_plus_exposure(Y,
                                                             X_initial_screen,
                                                             family,
                                                             obsWeights, id)
  vars[vars][ranked_vars > nVar] <- FALSE

  # also keep the first three columns of X (correspond to age, BMI, bhvrisk)
  vars[names(X) %in% c("age", "BMI", "bhvrisk")] <- TRUE
  return(vars)
}
# screen dynamic range score: only keep variables with
# sd(vacinees)/sd(placebo) > 75th percentile
# relies on having var.super loaded in the environment
screen_dynamic_range_score_plus_exposure <- function(Y, X, family,
                                                     obsWeights, id,
                                                     var_super = var.super,
                                                     nVar = 4, ...) {
  # set all to false
```

```
  vars <- rep(FALSE, ncol(X))
  # need to apply with the correct label in place of X
  vars_sd_ratio <- ifelse(is.na(var_super$sd.ratio), TRUE,
                          var_super$sd.ratio > quantile(var_super$sd.ratio,
                                                        probs = c(0.5),
                                                        na.rm = TRUE))
  vars <- names(X) %in% var_super$varname[vars_sd_ratio] |
    names(X) %in% paste0(var_super$varname[vars_sd_ratio], "_bin")
  # also keep the first three columns of X (correspond to age, BMI, bhvrisk)
  vars[names(X) %in% c("age", "BMI", "bhvrisk")] <- TRUE
  # keep only a max of nVar immune markers; rank by univariate p-value
  # in a model adjusting for age, BMI, bhvrisk
  X_initial_screen <- X %>%
    select(names(X)[vars], "age", "BMI", "bhvrisk")
  ranked_vars <- rank_univariate_logistic_pval_plus_exposure(Y,
                                                             X_initial_screen,
                                                             family,
                                                             obsWeights, id)
  vars[vars][ranked_vars > nVar] <- FALSE
  # also keep the first three columns of X (correspond to age, BMI, bhvrisk)
  vars[names(X) %in% c("age", "BMI", "bhvrisk")] <- TRUE
  return(vars)
}
# screen based on lasso
screen_glmnet_plus_exposure <- function(Y, X, family,
                                        obsWeights, id, alpha = 1,
                                        minscreen = 2, nfolds = 10,
                                        nlambda = 100, nVar = 4, ...) {
  vars <- screen.glmnet(Y, X, family, obsWeights, id, alpha = 1,
                        minscreen = 2, nfolds = 10, nlambda = 100, ...)
  # also keep the first three columns of X (correspond to age, BMI, bhvrisk)
  vars[names(X) %in% c("age", "BMI", "bhvrisk")] <- TRUE
  # keep only a max of nVar immune markers; rank by univariate p-value
  # in a model adjusting for age, BMI, bhvrisk
  X_initial_screen <- X %>%
    select(names(X)[vars], "age", "BMI", "bhvrisk")
  ranked_vars <- rank_univariate_logistic_pval_plus_exposure(Y,
                                                             X_initial_screen,
                                                             family,
                                                             obsWeights, id)
  vars[vars][ranked_vars > nVar] <- FALSE
  # also keep the first three columns of X (correspond to age, BMI, bhvrisk)
  vars[names(X) %in% c("age", "BMI", "bhvrisk")] <- TRUE
  return(vars)
}
# screen based on logistic regression univariate p-value < level
screen_univariate_logistic_pval_plus_exposure <- function(Y, X, family,
                                                          obsWeights, id,
                                                          minPvalue = 0.1,
                                                          minscreen = 2,
                                                          nVar = 4, ...) {
    # logistic regression of outcome on each variable
    listp <- apply(X, 2, function(x, Y, family) {
```

```
      summ <- coef(summary(glm(Y ~ x + X$age + X$BMI + X$bhvrisk,
                               family = family, weights = obsWeights)))
        ifelse(dim(summ)[1] > 1, summ[2, 4], 1)
    }, Y = Y, family = family)
    vars <- (listp <= minPvalue)
    # also keep the first three columns of X (correspond to age, BMI, bhvrisk)
    vars[names(X) %in% c("age", "BMI", "bhvrisk")] <- TRUE
    if (sum(vars) < minscreen) {
        warning("number of variables with p value less than minPvalue is
                less than minscreen")
        vars[rank(listp) <= minscreen] <- TRUE
    }
    # keep only a max of nVar immune markers; rank by univariate p-value
    # in a model adjusting for age, BMI, bhvrisk
    X_initial_screen <- X %>%
      select(names(X)[vars], "age", "BMI", "bhvrisk")
    ranked_vars <- rank_univariate_logistic_pval_plus_exposure(Y,
                                                               X_initial_screen,
                                                               family,
                                                               obsWeights, id)
    vars[vars][ranked_vars > nVar] <- FALSE
    vars[names(X) %in% c("age", "BMI", "bhvrisk")] <- TRUE
    return(vars)
}
screen_univariate_logistic_pval_plus_exposure_0.01 <- function(Y, X, family,
                                                               obsWeights, id,
                                                               minPvalue = 0.01,
                                                               minscreen = 2,
                                                               ...) {
  screen_univariate_logistic_pval_plus_exposure(Y, X, family, obsWeights,
                                                id, minPvalue, minscreen = 2,
                                                ...)
}
screen_univariate_logistic_pval_plus_exposure_0.05 <- function(Y, X, family,
                                                               obsWeights, id,
                                                               minPvalue = 0.05,
                                                               minscreen = 2,
                                                               ...) {
  screen_univariate_logistic_pval_plus_exposure(Y, X, family, obsWeights,
                                                id, minPvalue,
                                                minscreen = 2, ...)
}
screen_univariate_logistic_pval_plus_exposure_0.1 <- function(Y, X, family,
                                                              obsWeights, id,
                                                              minPvalue = 0.1,
                                                              minscreen = 2,
                                                              ...) {
  screen_univariate_logistic_pval_plus_exposure(Y, X, family, obsWeights,
                                                id, minPvalue,
                                                minscreen = 2, ...)
}
screen_highcor_plus_exposure <- function(Y, X, family, obsWeights, id,
                                         nVar = 4, ...) {
```

```
  # set all vars to FALSE
  vars <- rep(FALSE, ncol(X))
  # compute pairwise correlations between all marker vars
  cors <- cor(X, method = "spearman")
  diag(cors) <- NA
  cor_less_0.9 <- (cors <= 0.9)
  # screen out those with r > 0.9
  vars <- apply(cor_less_0.9, 1, function(x) all(x, na.rm = TRUE))
  # also keep the first three columns of X (correspond to age, BMI, bhvrisk)
  vars[names(X) %in% c("age", "BMI", "bhvrisk")] <- TRUE
  # keep only a max of nVar immune markers; rank by univariate p-value
  # in a model adjusting for age, BMI, bhvrisk
  X_initial_screen <- X %>%
    select(names(X)[vars], "age", "BMI", "bhvrisk")
  ranked_vars <- rank_univariate_logistic_pval_plus_exposure(Y,
                                                             X_initial_screen,
                                                             family,
                                                             obsWeights, id)
  vars[vars][ranked_vars > nVar] <- FALSE
  # make sure that age, BMI, bhvrisk are true
  vars[names(X) %in% c("age", "BMI", "bhvrisk")] <- TRUE
  return(vars)
}

# screen to always include the various variable sets
screen_assay_plus_exposure <- function(Y, X, family, obsWeights, id,
                                       assays, nVar = 4, ...) {
  # set all vars to be false
  vars <- rep(FALSE, ncol(X))
  # set vars with assay in name to be true
  # may be more than one
  for (i in 1:length(assays)) {
    vars[grepl(assays[i], names(X))] <- TRUE
  }
  # also keep the first three columns of X (correspond to age, BMI, bhvrisk)
  vars[names(X) %in% c("age", "BMI", "bhvrisk")] <- TRUE
  # keep only a max of nVar immune markers; rank by univariate p-value
  # in a model adjusting for age, BMI, bhvrisk
  X_initial_screen <- X %>%
    select(names(X)[vars], "age", "BMI", "bhvrisk")
  ranked_vars <- rank_univariate_logistic_pval_plus_exposure(Y,
                                                             X_initial_screen,
                                                             family,
                                                             obsWeights, id)
  vars[vars][ranked_vars > nVar] <- FALSE
  # set baseline exposure vars to true
  vars[names(X) %in% c("age", "BMI", "bhvrisk")] <- TRUE
  return(vars)
}

# actual screens for antigen combos
# baseline_exposure is the base model
screen_baseline_exposure <- function(Y, X, family, obsWeights, id, ...) {
```

```
  # set all vars to false
  vars <- rep(FALSE, ncol(X))
  # set baseline exposure vars to true
  vars[names(X) %in% c("age", "BMI", "bhvrisk")] <- TRUE
  return(vars)
}
screen_igg_iga_plus_exposure <- function(Y, X, family, obsWeights, id, ...) {
  screen_assay_plus_exposure(Y, X, family, obsWeights, id,
                             assays = c("IgG", "IgA"))
}
screen_tcells_plus_exposure <- function(Y, X, family, obsWeights, id, ...) {
  screen_assay_plus_exposure(Y, X, family, obsWeights, id,
                             assays = c("CD4", "CD8"))
}
screen_fxab_plus_exposure <- function(Y, X, family, obsWeights, id, ...) {
  screen_assay_plus_exposure(Y, X, family, obsWeights, id,
                             assays = c("IgG3", "phago", "fcrR2a", "fcrR3a"))
}
screen_igg_iga_tcells_plus_exposure <- function(Y, X, family, obsWeights, id, ...) {
  screen_assay_plus_exposure(Y, X,family,  obsWeights, id,
                             assays = c("IgG", "IgA", "CD4", "CD8"))
}
screen_igg_iga_fxab_plus_exposure <- function(Y, X, family, obsWeights, id, ...) {
  screen_assay_plus_exposure(Y, X, family, obsWeights, id,
                             assays = c("IgG", "IgA", "IgG3", "phago",
                                        "fcrR2a", "fcrR3a"))
}
screen_tcells_fxab_plus_exposure <- function(Y, X, family, obsWeights, id, ...) {
  screen_assay_plus_exposure(Y, X, family, obsWeights, id,
                             assays = c("CD4", "CD8", "IgG3", "phago",
                                        "fcrR2a", "fcrR3a"))
}

screens_with_assay_groups <- c(
  "screen_glmnet_plus_exposure",
  paste0("screen_univariate_logistic_pval_plus_exposure_",
         c(0.01, 0.05, 0.1)),
  "screen_dynamic_range_plus_exposure",
  "screen_dynamic_range_score_plus_exposure",
  "screen_baseline_exposure",
  paste0("screen_", c("igg_iga", "tcells", "fxab",
                      "igg_iga_tcells", "igg_iga_fxab", "tcells_fxab"),
         "_plus_exposure"),
  "screen_highcor_plus_exposure"
  )
screens <- c(
  "screen_glmnet_plus_exposure",
  paste0("screen_univariate_logistic_pval_plus_exposure_",
         c(0.01, 0.05, 0.1)),
  "screen_dynamic_range_plus_exposure",
  "screen_dynamic_range_score_plus_exposure",
  "screen_highcor_plus_exposure"
  )
```

```
# ------------------------------------------------------------------------------
# SL algorithms
# ------------------------------------------------------------------------------

# ------------------------------------------------------------------------
# define wrappers that are less memory-intensive than the usual SL functions
# ------------------------------------------------------------------------
# skinny glm
SL.glm.skinny <- function(Y, X, newX, family, obsWeights, ...){
  SL.glm.fit <- SL.glm(Y = Y, X = X, newX = newX, family = family,
                       obsWeights = obsWeights, ...)
  SL.glm.fit$fit$object$y <- NULL
  SL.glm.fit$fit$object$model <- NULL
  SL.glm.fit$fit$object$residuals <- NULL
  SL.glm.fit$fit$object$fitted.values <- NULL
  SL.glm.fit$fit$object$effects <- NULL
  SL.glm.fit$fit$object$qr$qr <- NULL
  SL.glm.fit$fit$object$linear.predictors <- NULL
  SL.glm.fit$fit$object$weights <- NULL
  SL.glm.fit$fit$object$prior.weights <- NULL
  SL.glm.fit$fit$object$data <- NULL
  SL.glm.fit$fit$object$family$variance <- NULL
  SL.glm.fit$fit$object$family$dev.resids <- NULL
  SL.glm.fit$fit$object$family$aic <- NULL
  SL.glm.fit$fit$object$family$validmu <- NULL
  SL.glm.fit$fit$object$family$simulate <- NULL
  attr(SL.glm.fit$fit$object$terms, ".Environment") <- NULL
  attr(SL.glm.fit$fit$object$formula, ".Environment") <- NULL
  return(SL.glm.fit)
}

# skinny glm with interactions
SL.glm.interaction.skinny <- function(Y, X, newX, family, obsWeights, ...){
  SL.glm.fit <- SL.glm.interaction(Y = Y, X = X, newX = newX,
                                   family = family, obsWeights = obsWeights,
                                   ...)
  SL.glm.fit$fit$object$y <- NULL
  SL.glm.fit$fit$object$model <- NULL
  SL.glm.fit$fit$object$residuals <- NULL
  SL.glm.fit$fit$object$fitted.values <- NULL
  SL.glm.fit$fit$object$effects <- NULL
  SL.glm.fit$fit$object$qr$qr <- NULL
  SL.glm.fit$fit$object$linear.predictors <- NULL
  SL.glm.fit$fit$object$weights <- NULL
  SL.glm.fit$fit$object$prior.weights <- NULL
  SL.glm.fit$fit$object$data <- NULL
  SL.glm.fit$fit$object$family$variance <- NULL
  SL.glm.fit$fit$object$family$dev.resids <- NULL
  SL.glm.fit$fit$object$family$aic <- NULL
  SL.glm.fit$fit$object$family$validmu <- NULL
  SL.glm.fit$fit$object$family$simulate <- NULL
  attr(SL.glm.fit$fit$object$terms, ".Environment") <- NULL
  attr(SL.glm.fit$fit$object$formula, ".Environment") <- NULL
```

```
  return(SL.glm.fit)
}

# skinny stepwise with interactions
SL.step.interaction.skinny <- function(Y, X, newX, family, obsWeights, ...){
  SL.step.interaction.fit <- SL.step.interaction(Y = Y, X = X, newX = newX,
                                                 family = family,
                                                 obsWeights = obsWeights,
                                                 direction = "forward", ...)
  SL.step.interaction.fit$fit$object$y <- NULL
  SL.step.interaction.fit$fit$object$model <- NULL
  SL.step.interaction.fit$fit$object$residuals <- NULL
  SL.step.interaction.fit$fit$object$fitted.values <- NULL
  SL.step.interaction.fit$fit$object$effects <- NULL
  SL.step.interaction.fit$fit$object$qr$qr <- NULL
  SL.step.interaction.fit$fit$object$linear.predictors <- NULL
  SL.step.interaction.fit$fit$object$weights <- NULL
  SL.step.interaction.fit$fit$object$prior.weights <- NULL
  SL.step.interaction.fit$fit$object$data <- NULL
  SL.step.interaction.fit$fit$object$family$variance <- NULL
  SL.step.interaction.fit$fit$object$family$dev.resids <- NULL
  SL.step.interaction.fit$fit$object$family$aic <- NULL
  SL.step.interaction.fit$fit$object$family$validmu <- NULL
  SL.step.interaction.fit$fit$object$family$simulate <- NULL
  attr(SL.step.interaction.fit$fit$object$terms, ".Environment") <- NULL
  attr(SL.step.interaction.fit$fit$object$formula, ".Environment") <- NULL
  return(SL.step.interaction.fit)
}

# skinny stepwise (forward)
SL.step.skinny <- function(Y, X, newX, family, obsWeights, ...){
  SL.step.fit <- SL.step(Y = Y, X = X, newX = newX, family = family,
                         obsWeights = obsWeights,
                         direction = "forward", ...)
  SL.step.fit$fit$object$y <- NULL
  SL.step.fit$fit$object$model <- NULL
  SL.step.fit$fit$object$residuals <- NULL
  SL.step.fit$fit$object$fitted.values <- NULL
  SL.step.fit$fit$object$effects <- NULL
  SL.step.fit$fit$object$qr$qr <- NULL
  SL.step.fit$fit$object$linear.predictors <- NULL
  SL.step.fit$fit$object$weights <- NULL
  SL.step.fit$fit$object$prior.weights <- NULL
  SL.step.fit$fit$object$data <- NULL
  SL.step.fit$fit$object$family$variance <- NULL
  SL.step.fit$fit$object$family$dev.resids <- NULL
  SL.step.fit$fit$object$family$aic <- NULL
  SL.step.fit$fit$object$family$validmu <- NULL
  SL.step.fit$fit$object$family$simulate <- NULL
  attr(SL.step.fit$fit$object$terms, ".Environment") <- NULL
  attr(SL.step.fit$fit$object$formula, ".Environment") <- NULL
  return(SL.step.fit)
}
```

```r
my_SL.xgboost <- function (Y, X, newX, family, obsWeights, id, ntrees = 1000,
                           max_depth = 4, shrinkage = 0.1, minobspernode = 10,
                           params = list(),
                           nthread = 1, verbose = 0,
                           save_period = NULL, ...) {
  if (packageVersion("xgboost") < 0.6)
    stop("SL.xgboost requires xgboost version >= 0.6,
         try help('SL.xgboost') for details")
  if (!is.matrix(X)) {
    X = model.matrix(~. - 1, X)
  }
  xgmat = xgboost::xgb.DMatrix(data = X, label = Y, weight = obsWeights)
  if (family$family == "gaussian") {
    if (packageVersion("xgboost") < "1.1.1.1") {
      objective <- "reg:linear"
    }
    else {
      objective <- "reg:squarederror"
    }
    model = xgboost::xgboost(data = xgmat, objective = objective,
                             nrounds = ntrees, max_depth = max_depth,
                             min_child_weight = minobspernode,
                             eta = shrinkage, verbose = verbose,
                             nthread = nthread,
                             params = params, save_period = save_period)
  }
  if (family$family == "binomial") {
    model = xgboost::xgboost(data = xgmat, objective = "binary:logistic",
                             nrounds = ntrees, max_depth = max_depth,
                             min_child_weight = minobspernode,
                             eta = shrinkage, verbose = verbose,
                             nthread = nthread,
                             params = params, save_period = save_period)
  }
  if (family$family == "multinomial") {
    model = xgboost::xgboost(data = xgmat, objective = "multi:softmax",
                             nrounds = ntrees, max_depth = max_depth,
                             min_child_weight = minobspernode,
                             eta = shrinkage, verbose = verbose,
                             num_class = length(unique(Y)),
                             nthread = nthread, params = params,
                             save_period = save_period)
  }
  if (!is.matrix(newX)) {
    newX = model.matrix(~. - 1, newX)
  }
  pred = predict(model, newdata = newX)
  fit = list(object = model)
  class(fit) = c("SL.xgboost")
  out = list(pred = pred, fit = fit)
  return(out)
}
# boosted decision stumps
```

```r
SL.stumpboost <- function(Y, X, newX, family, obsWeights, ...){
  fit <- my_SL.xgboost(Y = Y, X = X, newX = newX, family = family,
                       obsWeights = obsWeights,
                     max_depth = 1, # so it's only a stump
                     ...)
  return(fit)
}

# naive bayes wrapper
SL.naivebayes <- function(Y, X, newX, family, obsWeights, laplace = 0, ...){
  SuperLearner:::.SL.require("e1071")
  if(family$family == "gaussian"){
    stop("SL.naivebayes only works with binary outcomes")
  }else{
    nb <- naiveBayes(y = Y, x = X, laplace = laplace)
    pred <- predict(nb, newX, type = "raw")[,2]
    out <- list(fit = list(object = nb), pred = pred)
    class(out$fit) <- "SL.naivebayes"
    return(out)
  }
}

# predict method for naive bayes wrapper
predict.SL.naivebayes <- function(object, newdata, ...){
  pred <- predict(object$object, newdata = newdata, type = "raw")[,2]
  return(pred)
}

# methods <- c("SL.glm.skinny", "SL.glm.interaction.skinny",
#              "SL.step.interaction.skinny",
#              "SL.naivebayes", "SL.stumpboost", "SL.glmnet", "SL.earth")
methods <- c("SL.glm.skinny", "SL.glm.interaction.skinny",
             "SL.step.interaction.skinny",
             "SL.stumpboost", "SL.glmnet", "SL.earth")




# ------------------------------------------------------------------------------
# Add all alg/screen combinations to global environment, create SL library
# ------------------------------------------------------------------------------

#' This function takes a super learner method wrapper and a super learner
#' screen wrapper and combines them into a single wrapper and makes that
#' wrapper available in the specified environment. It also makes a predict
#' method available in the specified environment.
#' @param method A super learner method wrapper.
#' #             See ?SuperLearner::listWrappers(what = "method").
#' @param screen A super learner method wrapper.
#' #             See ?SuperLearner::listWrappers(what = "screen").
#' @param envir The environment to assign the functions to
#' #            (default is global environment)
#' @param verbose Print a message with the function names
```

```
#' #              confirming their assignment?
assign_combined_function <- function(method, screen, envir = .GlobalEnv,
                                     verbose = TRUE){
  fn <- eval(parse(text =
          paste0("function(Y, X, newX, obsWeights, family, ...){ \n",
                 "screen_call <- ",
                 screen,
                 "(Y = Y, X = X, newX = newX, obsWeights = obsWeights,
                 family = family, ...) \n",
                 "method_call <- ",
                 method,
                 "(Y = Y, X = X[,screen_call,drop=FALSE],
                 newX = newX[,screen_call,drop = FALSE],
                 obsWeights = obsWeights, family = family, ...) \n",
                 "pred <- method_call$pred \n",
                 "fit <- list(object = method_call$fit$object,
                 which_vars = screen_call) \n",
                 "class(fit) <- paste0('",
                 screen, "', '_', '", method, "') \n",
                 "out <- list(fit = fit, pred = pred) \n",
                    "return(out) \n",
                    "}")))
  fn_name <- paste0(screen,"_",method)
  assign(x = fn_name, value = fn, envir = envir)
  if(verbose){
    message(paste0("Function ", fn_name,
                   " now available in requested environment."))
  }
  if (method == "SL.glmnet") {
      pred_fn <- eval(parse(text =
              paste0("function(object, newdata, ...){ \n",
                     "screen_newdata <- newdata[,object$which_vars,
                     drop = FALSE] \n",
                     "pred <- predict(object$object, type = 'response',
                     newx = as.matrix(screen_newdata),
                     s = 'lambda.min', ...) \n",
                     "return(pred) \n",
                     "}")))
    } else if (method == "SL.stumpboost") {
      pred_fn <- eval(parse(text =
              paste0("function(object, newdata, ...){ \n",
              "screen_newdata <- newdata[,
                     object$which_vars,drop = FALSE] \n",
              "screen_newdata_2 <- matrix(unlist(lapply(screen_newdata,
              as.numeric)), nrow=nrow(screen_newdata),
              ncol=ncol(screen_newdata)) \n",
              "pred <- predict(object$object,
              newdata = screen_newdata_2, ...) \n",
              "return(pred) \n",
              "}")))
    } else if (method == "SL.naivebayes") {
      pred_fn <- eval(parse(text =
              paste0("function(object, newdata, ...){ \n",
```

```
                    "screen_newdata <- newdata[,
                    object$which_vars,drop = FALSE] \n",
                    'pred <- predict(object$object,
                    newdata = screen_newdata, type = "raw", ...)[,2] \n',
                       "return(pred) \n",
                    "}")))
    } else if (method == "SL.randomForest") {
      pred_fn <- eval(parse(text =
              paste0("function(object, newdata, ...){ \n",
              "screen_newdata <- newdata[,
              object$which_vars,drop = FALSE] \n",
              "if (object$object$type != 'classification') {
                  pred <- predict(object$object,
                  newdata = screen_newdata, type = 'response')
               } else {
                  pred <- predict(object$object,
                  newdata = screen_newdata, type = 'vote')[, 2]
               }
               pred",
              "}")))
    }else {
      pred_fn <- eval(parse(text =
              paste0("function(object, newdata, ...){ \n",
                     "screen_newdata <- newdata[,
                     object$which_vars,drop = FALSE] \n",
                     "pred <- predict(object$object, type = 'response',
                     newdata = screen_newdata, ...) \n",
                     "return(pred) \n",
                     "}")))
    }

  pred_fn_name <- paste0("predict.",screen,"_",method)
  assign(x = pred_fn_name, value = pred_fn, envir = envir)
  if(verbose){
    message(paste0("Function ", pred_fn_name,
                   " now available in requested environment."))
  }
}

# make a data frame of all method/screen combinations needed
screen_method_frame_with_assay_groups <- expand.grid(
  screen = screens_with_assay_groups, method = methods
  )
screen_method_frame <- expand.grid(screen = screens, method = methods)

# add to global environment
apply(screen_method_frame_with_assay_groups, 1,
      function(x) {
        assign_combined_function(screen = x[1], method = x[2], verbose = FALSE)
        })
```

```
## NULL
```

```r
apply(screen_method_frame, 1,
      function(x) {
        assign_combined_function(screen = x[1], method = x[2], verbose = FALSE)
        })
```

```
## NULL
```

```r
# create SL library; reference method is glm with baseline exposure vars
SL_library_with_assay_groups <- c(
  apply(screen_method_frame_with_assay_groups, 1, paste0, collapse = "_")
  )
SL_library <- c(apply(screen_method_frame, 1, paste0, collapse = "_"))
```

We used two layers of nested cross-validation in this analysis. To obtain cross-validated estimates of risk, we used an outer layer of five-fold cross-validation stratified so that an equal proportion of events fell within each fold. To obtain estimates of the regression functions within each outer fold, we used an inner layer of leave-one-out cross-validation. We used the cross-validated negative log likelihood to determine our optimal convex combination of learners. Since the results may depend on the random number seed used to generate the folds for the outer layer of five-fold cross-validation, we used ten random seeds, and for each seed ran the full nested cross-validation procedure.

Our resulting proposed estimator for objective (1) above is the cross-validated Super Learner averaged over the ten random seeds, where each prediction is based on a leave-one-out cross-validated Super Learner when the particular observation was in the outer validation fold.

We next define some useful functions that we will use in subsequent analyses:

```r
# utility functions
# These functions help to compute AUC, R-squared, and their cross-fitted counterparts
# Also, helper functions to run CV.SuperLearner, print nice names for plots, etc.

# ------------------------------------------------------------------------
# Cross-validated predictiveness
# ------------------------------------------------------------------------
# get the CV-AUC for a single learner's predicted values
# @param preds the fitted values
# @param Y the outcome
# @param scale what scale should the IPCW correction be applied on?
#              Can help with numbers outside of (0, 1)
#             ("identity" denotes the identity scale;
#              "logit" means that AUC is transformed to the logit scale,
#              correction is applied, and then back-transformed)
# @param weights the inverse probability of censoring weights
# @param C the indicator of being observed in phase 2 (1) or not (0)
# @param Z a matrix of predictors observed on all participants in phase 1
#          (can include the outcome)
# @param ... other arguments to measure_auc, but should include at least:
#            a library of learners (using arg "SL.library")
#            and may include control parameters for the super learner
#            (e.g., cvControl = list(V = 5)
#             for 5-fold cross-validated super learner)
one_auc <- function(preds, Y, full_y = NULL, scale = "identity",
                    weights = rep(1, length(Y)), C = rep(1, length(Y)),
                    Z = NULL, weight_type = "aipw", ...) {
  auc_lst <- vimp::measure_auc(
    fitted_values = preds, y = Y, full_y = full_y, C = C, Z = Z,
```

```
    ipc_weights = weights,
    ipc_fit_type = "SL", ipc_est_type = weight_type, ...)
  list(auc = auc_lst$point_est, eif = auc_lst$eif)
}
# get the cross-fitted CV-AUC for a single learner's predicted values
# @param preds the fitted values
# @param Y the outcome
# @param folds the different cv folds that the learner was evaluated on
# @param scale what scale should the IPCW correction be applied on?
#              Can help with numbers outside of (0, 1)
#             ("identity" denotes the identity scale;
#              "logit" means that AUC is transformed to the logit scale,
#              correction is applied, and then back-transformed)
# @param weights the inverse probability of censoring weights
# @param C the indicator of being observed in phase 2 (1) or not (0)
# @param Z a matrix of predictors observed on all participants in phase 1
#          (can include the outcome)
# @param ... other arguments to measure_auc, but should include at least:
#            a library of learners (using arg "SL.library")
#            and may include control parameters for the super learner
#            (e.g., cvControl = list(V = 5)
#             for 5-fold cross-validated super learner)
cv_auc <- function(preds, Y, folds, scale = "identity",
                   weights = rep(1, length(Y)), C = rep(1, length(Y)),
                   Z = NULL, weight_type = "aipw", ...) {
  V <- length(folds)
  folds_numeric <- get_cv_sl_folds(folds)
  folds_z <- c(folds_numeric, sample(seq_len(V), nrow(Z) - length(folds_numeric),
                                     replace = TRUE))
  ests_eifs <- lapply(as.list(1:V), function(v) {
    one_auc(preds = preds[folds_numeric == v], Y[folds_numeric == v],
            full_y = Y, scale = scale,
            weights = weights[folds_z == v], C = C[folds_z == v],
            Z = Z[folds_z == v, , drop = FALSE], weight_type = weight_type, ...)
  })
  est <- mean(unlist(lapply(ests_eifs, function(l) l$auc)))
  all_eifs <- lapply(ests_eifs, function(l) l$eif)
  se <- vimp::vimp_se(list(est = est, all_eifs = all_eifs), n = length(Y))
  ci <- vimp::vimp_ci(est, se, scale = scale, level = 0.95)
  return(list(auc = est, se = se, ci = ci))
}
# get the folds from a CV.SL object, make them a vector
# @param cv_sl_folds the CV.SL folds (a named list of row numbers)
# @return a vector with the correct folds
get_cv_sl_folds <- function(cv_sl_folds) {
  folds_with_row_nums <- sapply(1:length(cv_sl_folds),
                                function(x)
                                  list(
                                    row_nums = cv_sl_folds[[x]],
                                    fold = rep(x, length(cv_sl_folds[[x]]))
                                    ),
                                simplify = FALSE
                                )
```

```
  folds_df <- data.table::rbindlist(folds_with_row_nums)
  folds_df$fold[order(folds_df$row_nums)]
}
# get the CV-AUC for all learners fit with SL
# @param sl_fit the super learner fit object
# @param scale what scale should the IPCW correction be applied on?
#              Can help with numbers outside of (0, 1)
#             ("identity" denotes the identity scale;
#              "logit" means that AUC is transformed to the logit scale,
#              correction is applied, and then back-transformed)
# @param weights the inverse probability of censoring weights
# @param C the indicator of being observed in phase 2 (1) or not (0)
# @param Z a matrix of predictors observed on all participants in phase 1
#          (can include the outcome)
# @param ... other arguments to measure_auc, but should include at least:
#            a library of learners (using arg "SL.library")
#            and may include control parameters for the super learner
#            (e.g., cvControl = list(V = 5)
#             for 5-fold cross-validated super learner)
get_all_aucs <- function(sl_fit, scale = "identity",
                         weights = rep(1, length(sl_fit$Y)),
                         C = rep(1, length(sl_fit$Y)),
                         Z = NULL, weight_type = "aipw", ...) {
  # get the CV-AUC of the SuperLearner predictions
  sl_auc <- cv_auc(preds = sl_fit$SL.predict, Y = sl_fit$Y,
                   folds = sl_fit$folds,
                   scale = scale, weights = weights, C = C, Z = Z,
                   weight_type = weight_type, ...)
  out <- data.frame(Learner="SL", Screen="All", AUC = sl_auc$auc,
                    se = sl_auc$se, ci_ll = sl_auc$ci[1], ci_ul=sl_auc$ci[2])

  # Get the CV-auc of the Discrete SuperLearner predictions
  discrete_sl_auc <- cv_auc(preds = sl_fit$discreteSL.predict, Y = sl_fit$Y,
                            folds = sl_fit$folds, scale = scale,
                            weights = weights, C = C,
                            Z = Z, weight_type = weight_type, ...)
  out <- rbind(out, data.frame(Learner="Discrete SL", Screen="All",
                               AUC = discrete_sl_auc$auc,
                               se = discrete_sl_auc$se,
                               ci_ll = discrete_sl_auc$ci[1],
                               ci_ul = discrete_sl_auc$ci[2]))

  # Get the cvauc of the individual learners in the library
  get_individual_auc <- function(sl_fit, col, scale = "identity",
                                 weights = rep(1, length(sl_fit$Y)),
                                 C = rep(1, length(sl_fit$Y)), Z = NULL,
                                 weight_type = "aipw", ...) {
    if(any(is.na(sl_fit$library.predict[, col]))) return(NULL)
    alg_auc <- cv_auc(preds = sl_fit$library.predict[, col], Y = sl_fit$Y,
                      scale = scale,
                      folds = sl_fit$folds, weights = weights,
                      C = C, Z = Z, weight_type = weight_type, ...)
    # get the regexp object
```

```
    alg_screen_string <- strsplit(colnames(sl_fit$library.predict)[col], "_",
                                  fixed = TRUE)[[1]]
    alg <- tail(alg_screen_string[grepl(".", alg_screen_string,
                                        fixed = TRUE)], n = 1)
    screen <- paste0(alg_screen_string[!grepl(alg, alg_screen_string,
                                              fixed = TRUE)],
                     collapse = "_")
    data.frame(Learner = alg, Screen = screen, AUC = alg_auc$auc,
               se = alg_auc$se,
               ci_ll = alg_auc$ci[1], ci_ul = alg_auc$ci[2])
  }
  other_aucs <- plyr::ldply(1:ncol(sl_fit$library.predict),
                            function(x)
                              get_individual_auc(
                                sl_fit = sl_fit,
                                col = x,
                                scale = scale,
                                weights = weights,
                                C = C, Z = Z, weight_type = weight_type, ...)
                            )
  rbind(out, other_aucs)
}
# get the CV-AUC for all super learner objects in a list
# @param sl_fit_lst a list of super learner fit objects
# @param scale what scale should the IPCW correction be applied on?
#              Can help with numbers outside of (0, 1)
#             ("identity" denotes the identity scale;
#              "logit" means that AUC is transformed to the logit scale,
#              correction is applied, and then back-transformed)
# @param weights the inverse probability of censoring weights
# @param C the indicator of being observed in phase 2 (1) or not (0)
# @param Z a matrix of predictors observed on all participants in phase 1
#          (can include the outcome)
# @param ... other arguments to measure_auc, but should include at least:
#            a library of learners (using arg "SL.library")
#            and may include control parameters for the super learner
#            (e.g., cvControl = list(V = 5)
#             for 5-fold cross-validated super learner)
get_all_aucs_lst <- function(sl_fit_lst, scale = "identity",
                             weights = rep(1, length(sl_fit_lst[[1]]$fit$Y)),
                             C = rep(1, length(sl_fit_lst[[1]]$fit$Y)),
                             Z = NULL, ...) {
  # get the CV-AUC of the SuperLearner predictions
  if (is.null(sl_fit_lst)) {
    return(NA)
  } else {
    sl_auc <- cv_auc(preds = sl_fit_lst$fit$SL.predict,
                     Y = sl_fit_lst$fit$Y,
                     scale = scale,
                     folds = sl_fit_lst$fit$folds, weights = weights,
                     C = C, Z = Z, ...)
    out <- data.frame(Learner="SL", Screen="All", AUC = sl_auc$auc,
                      se = sl_auc$se[1],
```

```
                      ci_ll = sl_auc$ci[1],
                      ci_ul=sl_auc$ci[2])

    # Get the CV-auc of the Discrete SuperLearner predictions
    discrete_sl_auc <- cv_auc(preds = sl_fit_lst$fit$discreteSL.predict,
                              Y = sl_fit_lst$fit$Y,
                              folds = sl_fit_lst$fit$folds,
                              scale = scale,
                              weights = weights, C = C, Z = Z, ...)
    out <- rbind(out, data.frame(Learner="Discrete SL", Screen="All",
                                 AUC = discrete_sl_auc$auc,
                                 se = discrete_sl_auc$se[1],
                                 ci_ll = discrete_sl_auc$ci[1],
                                 ci_ul = discrete_sl_auc$ci[2]))

    # Get the cvauc of the individual learners in the library
    get_individual_auc <- function(sl_fit, col, scale, weights, C, Z, ...) {
      if(any(is.na(sl_fit$library.predict[, col]))) return(NULL)
      alg_auc <- cv_auc(preds = sl_fit$library.predict[, col], Y = sl_fit$Y,
                        scale = scale,
                        folds = sl_fit$folds, weights = weights,
                        C = C, Z = Z, ...)
      # get the regexp object
      alg_screen_string <- strsplit(colnames(sl_fit$library.predict)[col], "_",
                                    fixed = TRUE)[[1]]
      alg <- tail(alg_screen_string[grepl(".", alg_screen_string,
                                          fixed = TRUE)], n = 1)
      screen <- paste0(alg_screen_string[!grepl(alg, alg_screen_string,
                                                fixed = TRUE)], collapse = "_")
      data.frame(Learner = alg, Screen = screen, AUC = alg_auc$auc,
                 se = alg_auc$se, ci_ll = alg_auc$ci[1], ci_ul = alg_auc$ci[2])
    }
    other_aucs <- plyr::ldply(1:ncol(sl_fit_lst$fit$library.predict),
                              function(x) get_individual_auc(
                                sl_fit = sl_fit_lst$fit,
                                col = x,
                                weights = weights, scale = scale,
                                C = C, Z = Z, ...)
                              )
    rbind(out, other_aucs)
  }
}
# get the CV-R^2 for a single learner's predicted values
# @param preds the fitted values
# @param Y the outcome
# @param scale what scale should the IPCW correction be applied on?
#              Can help with numbers outside of (0, 1)
#             ("identity" denotes the identity scale;
#              "logit" means that R^2 is transformed to the logit scale,
#             correction is applied, and then back-transformed)
# @param weights the inverse probability of censoring weights
# @param C the indicator of being observed in phase 2 (1) or not (0)
# @param Z a matrix of predictors observed on all participants in phase 1
```

```
#            (can include the outcome)
# @param ... other arguments to measure_auc, but should include at least:
#            a library of learners (using arg "SL.library")
#            and may include control parameters for the super learner
#            (e.g., cvControl = list(V = 5)
#             for 5-fold cross-validated super learner)
one_r2 <- function(preds, Y, full_y = NULL, scale = "identity",
                   weights = rep(1, length(Y)),
                   C = rep(1, length(Y)), Z = NULL, ...) {
  r2_lst <- vimp::measure_r_squared(fitted_values = preds, y = Y,
                                    full_y = full_y, C = C, Z = Z,
                                    ipc_weights = weights,
                                    ipc_fit_type = "SL", ...)
  list(r2 = r2_lst$point_est, eif = r2_lst$eif)
}
# get the cross-fitted CV-R^2 for a single learner's predicted values
# @param preds the fitted values
# @param Y the outcome
# @param folds the folds that the learner was evaluated on
# @param scale what scale should the IPCW correction be applied on?
#              Can help with numbers outside of (0, 1)
#             ("identity" denotes the identity scale;
#              "logit" means that R^2 is transformed to the logit scale,
#             correction is applied, and then back-transformed)
# @param weights the inverse probability of censoring weights
# @param C the indicator of being observed in phase 2 (1) or not (0)
# @param Z a matrix of predictors observed on all participants in phase 1
#          (can include the outcome)
# @param ... other arguments to measure_auc, but should include at least:
#            a library of learners (using arg "SL.library")
#            and may include control parameters for the super learner
#            (e.g., cvControl = list(V = 5)
#            for 5-fold cross-validated super learner)
cv_r2 <- function(preds, Y, folds, scale = "identity",
                  weights = rep(1, length(Y)),
                  C = rep(1, length(Y)), Z = NULL, ...) {
  V <- length(folds)
  folds_numeric <- get_cv_sl_folds(folds)
  ests_eifs <- lapply(as.list(1:V), function(v) {
    one_r2(preds = preds[folds_numeric == v], Y[folds_numeric == v],
           full_y = Y, scale = scale,
           weights = weights[folds_numeric == v], C = C[folds_numeric == v],
           Z = Z[folds_numeric == v, , drop = FALSE], ...)
  })
  est <- mean(unlist(lapply(ests_eifs, function(l) l$r2)))
  all_eifs <- lapply(ests_eifs, function(l) l$eif)
  se <- vimp::vimp_se(list(est = est, all_eifs = all_eifs), n = length(Y))
  ci <- vimp::vimp_ci(est = est, se = se, scale = scale, level = 0.95)
  return(list(r2 = est, se = se, ci = ci))
}
# get the CV-R^2 for a super learner fitted object
# @param sl_fit the super learner fit object
# @param scale what scale should the IPCW correction be applied on?
```

```
#              Can help with numbers outside of (0, 1)
#             ("identity" denotes the identity scale;
#              "logit" means that R^2 is transformed to the logit scale,
#              correction is applied, and then back-transformed)
# @param weights the inverse probability of censoring weights
# @param C the indicator of being observed in phase 2 (1) or not (0)
# @param Z a matrix of predictors observed on all participants in phase 1
#          (can include the outcome)
# @param ... other arguments to measure_auc, but should include at least:
#            a library of learners (using arg "SL.library")
#            and may include control parameters for the super learner
#            (e.g., cvControl = list(V = 5)
#             for 5-fold cross-validated super learner)
get_all_r2s <- function(sl_fit, scale = "identity",
                        weights = rep(1, length(sl_fit$Y)),
                        C = rep(1, length(sl_fit$Y)), Z = NULL, ...) {
  # get the CV-R^2 of the SuperLearner predictions
  sl_r2 <- cv_r2(preds = sl_fit$SL.predict, Y = sl_fit$Y,
                 folds = sl_fit$folds,
                 scale = scale, C = C, Z = Z, ...)
  out <- data.frame(Learner="SL", Screen="All", R2 = sl_r2$r2,
                    se = sl_r2$se,
                    ci_ll = sl_r2$ci[1], ci_ul=sl_r2$ci[2])

  # Get the CV-R2 of the Discrete SuperLearner predictions
  discrete_sl_r2 <- cv_r2(preds = sl_fit$discreteSL.predict, Y = sl_fit$Y,
                          folds = sl_fit$folds, scale = scale,
                          C = C, Z = Z, ...)
  out <- rbind(out, data.frame(Learner="Discrete SL", Screen="All",
                               R2 = discrete_sl_r2$r2, se = discrete_sl_r2$se,
                               ci_ll = discrete_sl_r2$ci[1],
                               ci_ul = discrete_sl_r2$ci[2]))

  # Get the cvr2 of the individual learners in the library
  get_individual_r2 <- function(sl_fit, col, scale = "identity",
                                weights = rep(1, length(sl_fit$Y)),
                                C = rep(1, length(sl_fit$Y)), Z = NULL, ...) {
    if(any(is.na(sl_fit$library.predict[, col]))) return(NULL)
    alg_r2 <- cv_r2(preds = sl_fit$library.predict[, col], Y = sl_fit$Y,
                    folds = sl_fit$folds, scale = scale, weights = weights,
                    C = C, Z = Z, ...)
    # get the regexp object
    alg_screen_string <- strsplit(colnames(sl_fit$library.predict)[col], "_",
                                  fixed = TRUE)[[1]]
    alg <- tail(alg_screen_string[grepl(".", alg_screen_string,
                                        fixed = TRUE)], n = 1)
    screen <- paste0(alg_screen_string[!grepl(alg, alg_screen_string,
                                              fixed = TRUE)], collapse = "_")
    data.frame(Learner = alg, Screen = screen, R2 = alg_r2$r2,
               se = alg_r2$se, ci_ll = alg_r2$ci[1], ci_ul = alg_r2$ci[2])
  }
  other_r2s <- plyr::ldply(1:ncol(sl_fit$library.predict),
                           function(x) get_individual_r2(
```

```
                              sl_fit = sl_fit, col = x,
                              scale = scale, weights = weights,
                              C = C, Z = Z, ...)
                            )
  rbind(out, other_r2s)
}
# get the CV-R^2 for a list of super learner objects
# @param sl_fit_lst the list of super learner fit objects
# @param scale what scale should the IPCW correction be applied on?
#              Can help with numbers outside of (0, 1)
#             ("identity" denotes the identity scale;
#              "logit" means that R^2 is transformed to the logit scale,
#               correction is applied, and then back-transformed)
# @param weights the inverse probability of censoring weights
# @param C the indicator of being observed in phase 2 (1) or not (0)
# @param Z a matrix of predictors observed on all participants in phase 1
#          (can include the outcome)
# @param ... other arguments to measure_auc, but should include at least:
#            a library of learners (using arg "SL.library")
#            and may include control parameters for the super learner
#            (e.g., cvControl = list(V = 5)
#             for 5-fold cross-validated super learner)
get_all_r2s_lst <- function(sl_fit_lst, scale = "identity",
                            weights = rep(1, length(sl_fit_lst[[1]]$fit$Y)),
                            C = rep(1, length(sl_fit_lst[[1]]$Y)),
                            Z = NULL, ...) {
  # get the CV-R^2 of the SuperLearner predictions
  if (is.null(sl_fit_lst)) {
    return(NA)
  } else {
    sl_r2 <- cv_r2(preds = sl_fit_lst$fit$SL.predict,
                   Y = sl_fit_lst$fit$Y,
                   folds = sl_fit_lst$fit$folds,
                   scale = scale, weights = weights,
                   C = C, Z = Z, ...)
    out <- data.frame(Learner="SL", Screen="All", R2 = sl_r2$r2,
                      se = sl_r2$se,
                      ci_ll = sl_r2$ci[1], ci_ul=sl_r2$ci[2])

    # Get the CV-R2 of the Discrete SuperLearner predictions
    discrete_sl_r2 <- cv_r2(preds = sl_fit_lst$fit$discreteSL.predict,
                            Y = sl_fit_lst$fit$Y, folds = sl_fit_lst$fit$folds,
                            scale = scale, weights = weights,
                            C = C, Z = Z, ...)
    out <- rbind(out, data.frame(Learner="Discrete SL", Screen="All",
                                 R2 = discrete_sl_r2$r2,
                                 se = discrete_sl_r2$se,
                                 ci_ll = discrete_sl_r2$ci[1],
                                 ci_ul = discrete_sl_r2$ci[2]))

    # Get the cvr2 of the individual learners in the library
    get_individual_r2 <- function(sl_fit, col, scale, weights, C, Z, ...) {
      if(any(is.na(sl_fit$library.predict[, col]))) return(NULL)
```

```
      alg_r2 <- cv_r2(preds = sl_fit$library.predict[, col], Y = sl_fit$Y,
                      scale = scale,
                      folds = sl_fit$folds, weights = weights,
                      C = C, Z = Z, ...)
      # get the regexp object
      alg_screen_string <- strsplit(colnames(sl_fit$library.predict)[col], "_",
                                    fixed = TRUE)[[1]]
      alg <- tail(alg_screen_string[grepl(".", alg_screen_string,
                                          fixed = TRUE)], n = 1)
      screen <- paste0(alg_screen_string[!grepl(alg, alg_screen_string,
                                                fixed = TRUE)], collapse = "_")
      data.frame(Learner = alg, Screen = screen, R2 = alg_r2$r2,
                 se = alg_r2$se, ci_ll = alg_r2$ci[1], ci_ul = alg_r2$ci[2])
    }
    other_r2s <- plyr::ldply(1:ncol(sl_fit_lst$fit$library.predict),
                             function(x) get_individual_r2(
                               sl_fit = sl_fit_lst$fit, col = x,
                               scale = scale, weights = weights,
                               C = C, Z = Z, ...)
                             )
    rbind(out, other_r2s)
  }
}

# ------------------------------------------------------------------------
# Run the CV Super Learner
# ------------------------------------------------------------------------
# run CV.SuperLearner for one given random seed
# @param seed the random number seed
# @param Y the outcome
# @param X_mat the covariates
# @param family the family, for super learner (e.g., "binomial")
# @param obsWeights the inverse probability of censoring weights
#                   (for super learner)
# @param all_weights the IPC weights for variable importance (full data)
# @param sl_lib the super learner library (e.g., "SL.ranger")
# @param method the method for determining the optimal combination
#               of base learners
# @param cvControl a list of control parameters to pass to the
#                  outer super learner
# @param innerCvControl a list of control parameters to pass to the
#                       inner super learners
# @param vimp determines whether or not we save the entire SL fit object
#        (for variable importance, we don't need it so can save some memory
#         by excluding large objects)
run_cv_sl_once <- function(seed = 1, Y = NULL, X_mat = NULL,
                           family = "binomial",
                           obsWeights = rep(1, length(Y)),
                           all_weights = rep(1, nrow(Z)),
                           sl_lib = "SL.ranger", method = "method.CC_nloglik",
                           cvControl = list(V = 5),
                           innerCvControl = list(V = 5), vimp = FALSE,
                           C = rep(1, length(Y)), Z = NULL,
```

```
                            z_lib = "SL.ranger", scale = "identity",
                            weight_type = "aipw") {
  set.seed(seed)
  fit <- SuperLearner::CV.SuperLearner(Y = Y, X = X_mat, family = family,
                                       obsWeights = obsWeights,
                                       SL.library = sl_lib,
                                       method = method, cvControl = cvControl,
                                       innerCvControl = innerCvControl)
  aucs <- get_all_aucs(sl_fit = fit, scale = scale, weights = all_weights,
                       C = C, Z = Z, SL.library = z_lib,
                       weight_type = weight_type)
  ret_lst <- list(fit = fit, folds = fit$folds, aucs = aucs)
  if (vimp) {
    ret_lst <- list(fit = fit$SL.predict, folds = fit$folds, aucs = aucs)
  }
  return(ret_lst)
}
# run SuperLearner given the set of results from the CV.SL with all markers
# for one given random seed
# @param seed the random number seed
# @param Y the outcome -- this is actually the "fit" object from the
#          CV.SL results object
# @param X_mat the covariates
# @param family the family, for super learner (e.g., "binomial")
# @param obsWeights the inverse probability of censoring weights
#                   (for super learner)
# @param sl_lib the super learner library (e.g., "SL.ranger")
# @param method the method for determining the optimal combination of
#               base learners
# @param innerCvControl a list of control parameters to pass to the
#                         inner super learners
# @param vimp determines whether or not we save the entire SL fit object
#        (for variable importance, we don't need it so can save some memory
#         by excluding large objects)
run_reduced_cv_sl_once <- function(seed = 1, Y = NULL, X_mat = NULL,
                                   family = "binomial",
                                   obsWeights = rep(1, length(Y)),
                                   sl_lib = "SL.ranger",
                                   method = "method.CC_nloglik",
                                   innerCvControl = list(V = 5), vimp = TRUE) {
  # pull out the correct set of fitted values
  set.seed(4747)
  seeds <- round(runif(10, 1000, 10000))
  indx <- which(seed == seeds)
  full_fit <- Y[[indx]]$fit
  # use the same folds as the CV.SuperLearner
  fold_row_nums <- as.vector(do.call(cbind, full_fit$folds))
  folds_init <- rep(as.numeric(names(full_fit$folds)),
                    each = dim(X_mat)[1] /length(full_fit$folds))
  folds_mat <- cbind(fold_row_nums, folds_init)
  folds <- folds_mat[order(folds_mat[, 1]), 2]
  V <- length(unique(folds))
```

```r
  # set the seed, run the SL
  set.seed(seed)
  preds_lst <- vector("list", length = V)
  for (v in 1:V) {
    # run an SL of full fit on reduced set of predictors
    inner_sl <- SuperLearner::SuperLearner(Y = full_fit$SL.predict[folds != v],
                                           X = X_mat[folds != v, ,
                                                     drop = FALSE],
                                           newX = X_mat[folds == v, ,
                                                        drop = FALSE],
                                           family = family,
                                           obsWeights = obsWeights[folds != v],
                                           SL.library = sl_lib,
                                           method = method,
                                           cvControl = innerCvControl)
    # get the predicted values on the vth fold
    preds_lst[[v]] <- inner_sl$SL.predict
  }
  # make a vector out of the predictions
  preds_mat <- do.call(rbind.data.frame,
                       lapply(preds_lst,
                              function(x)
                                cbind.data.frame(x,
                                                 row_num =
                                                   as.numeric(rownames(x))
                                                 )
                              )
                       )
  preds_mat_ordered <- preds_mat[order(preds_mat$row_num), ]
  # return
  ret_lst <- list(fit = preds_mat_ordered[, 1], folds = full_fit$folds)
}
# ------------------------------------------------------------------------
# Cross-fitted estimates of variable importance based on the chosen measure
# ------------------------------------------------------------------------
# get the cv vim for each fold
# @param full_fit the fitted values from a regression with the
#                 variables of interest
# @param reduced_fit the fitted values from a regression without the
#                    variables of interest
# @param x the fold to get importance on
# @param type the type of importance (e.g., "auc")
# @param weights the IPC weights
# @param C the indicator of being observed in phase 2 (1) or not (0)
# @param Z the predictors/outcome measured in phase 1
# @param SL.library the super learner library for IPCW correction
# @param scale the scale to measure importance/CIs on
# @param vimp do we want to do variable importance, or just estimate
#             the CV-predictiveness?
# @param ... other arguments to inner super learner (e.g., cvControl)
get_fold_cv_vim <- function(full_fit = NULL, reduced_fit = NULL, x = NULL,
                            type = "auc",
                            weights = rep(1, length(full_fit)),
```

```
                          C = rep(1, length(full_fit)),
                          Z = NULL, SL.library = "ranger",
                          scale = "identity", vimp = FALSE,...) {
# get the outcome, folds
if (!vimp) {
  y <- full_fit[[x]]$fit$Y
} else {
  y <- reduced_fit[[x]]$fit$Y
}
if (is.null(full_fit[[x]]$folds)) {
  vim_est <- NA
} else {
  fold_row_nums <- as.vector(do.call(cbind, full_fit[[x]]$folds))
  folds_init <- rep(as.numeric(names(full_fit[[x]]$folds)),
                    each = length(y)/length(full_fit[[x]]$folds))
  folds_mat <- cbind(fold_row_nums, folds_init)
  folds <- folds_mat[order(folds_mat[, 1]), 2]

  # get the full, reduced predictions from the two CV objects
  get_reduced_fit <- function(i) {
    if (type == "r_squared" & is.null(reduced_fit[[x]]$fit$Y)) {
      tryCatch(reduced_fit[[x]]$fit[folds == i],
               error = function(e) rep(NA, sum(folds == i)))
    } else {
      if (is.list(reduced_fit[[x]]$fit)) {
        reduced_fit[[x]]$fit$SL.predict[folds == i]
      } else {
        reduced_fit[[x]]$fit[folds == i]
      }
    }
  }
  full_fits <- lapply(as.list(1:length(full_fit[[x]]$folds)), function(i) {
    if (is.list(full_fit[[x]]$fit)) {
      full_fit[[x]]$fit$SL.predict[folds == i]
    } else {
      full_fit[[x]]$fit[folds == i]
    }

  })
  redu_fits <- lapply(
    as.list(1:length(full_fit[[x]]$folds)),
    function(i) get_reduced_fit(i)
    )
  ys <- lapply(
    as.list(1:length(full_fit[[x]]$folds)),
    function(i) y[folds == i]
    )
  # create the list of folds; note that we didn't do sample splitting,
  # so p-values are invalid
  folds_lst <- list(outer_folds = c(rep(1, length(full_fit[[x]]$fit$Y)),
                                    rep(2, length(full_fit[[x]]$fit$Y))),
                    inner_folds = list(inner_folds_1 = folds,
                                       inner_folds_2 = folds))
```

```
    # thus, VIM is based on 2 copies of the same data
    y_vim <- rep(y, 2)
    X_vim <- dplyr::bind_rows(Z, Z)
    C_vim <- rep(C, 2)
    weights_vim <- rep(weights, 2)
    # variable importance
    vim_est <- tryCatch(vimp::cv_vim(Y = y_vim, X = X_vim,
                                     V = length(unique(folds)),
                                f1 = full_fits,
                                f2 = redu_fits,
                                folds = folds_lst,
                                type = type,
                                run_regression = FALSE,
                                SL.library = SL.library,
                                ipc_weights = weights_vim,
                                C = C_vim, Z = c("Y", paste0("X", 1:3)),
                                alpha = 0.05,
                                scale = scale, ...), error = function(e) NA)
  }
  return(vim_est)
}
# get the CV vim averaged over the 10 folds
# @param full_fit the fitted values from a regression with the
#                 variables of interest
# @param reduced_fit the fitted values from a regression without the
#                    variables of interest
# @param x the fold to get importance on
# @param type the type of importance (e.g., "auc")
# @param weights the IPC weights
# @param C the indicator of being observed in phase 2 (1) or not (0)
# @param Z the predictors/outcome measured in phase 1
# @param SL.library the super learner library for IPCW correction
# @param scale the scale to measure importance/CIs on
# @param vimp do we want to do variable importance, or just estimate
#             the CV-predictiveness?
# @param ... other arguments to inner super learner (e.g., cvControl)
get_cv_vim <- function(full_fit = NULL, reduced_fit = NULL, type = "auc",
                       weights = rep(1, length(full_fit)),
                       C = rep(1, length(full_fit)),
                       Z = NULL, SL.library = "SL.ranger",
                       scale = "identity", vimp = FALSE, ...) {
  # get the cv vim for each fold
  all_cv_vims <- lapply(as.list(1:length(full_fit)), get_fold_cv_vim,
                        full_fit = full_fit,
                        reduced_fit = reduced_fit, type = type,
                        weights = weights, scale = scale, vimp = vimp,
                        C = C, Z = Z, SL.library = SL.library, ...)
  return(all_cv_vims)
}
# get CV vim averaged over 10 starts, if I've pre-computed IPCW
# @param full_ests a tibble with estimate and CIs, all covariates
# @param reduced_ests a tibble with estimate and CIs, reduced set of covariates
get_cv_vim_precomputed <- function(full_ests, reduced_ests,
```

```
                                  risk_type = "auc",
                                  scale = "identity") {
if (risk_type == "auc") {
  full_est <- full_ests %>%
    mutate(est = AUC) %>%
    filter(Learner == "SL")
  reduced_est <- reduced_ests %>%
    mutate(est = AUC) %>%
    filter(Learner == "SL")
} else {
  full_est <- full_ests %>%
    mutate(est = R2) %>%
    filter(Learner == "SL")
  reduced_est <- reduced_ests %>%
    mutate(est = R2) %>%
    filter(Learner == "SL")
}
if (scale == "identity") {
  full_se_scaled <- (full_est$ci_ul - full_est$est)
  reduced_se_scaled <- (reduced_est$ci_ul - reduced_est$est)
} else if (scale == "logit") {
  full_se_scaled <- stats::qlogis(full_est$ci_ul) -
    stats::qlogis(full_est$est)
  reduced_se_scaled <- stats::qlogis(reduced_est$ci_ul) -
    stats::qlogis(reduced_est$est)
} else {
  full_se_scaled <- log(full_est$ci_ul) - log(full_est$est)
  reduced_se_scaled <- log(reduced_est$ci_ul) - log(reduced_est$est)
}
var_scaled <- (full_se_scaled / stats::qnorm(0.975)) ^ 2 +
  (reduced_se_scaled / stats::qnorm(0.975)) ^ 2
point_est <- full_est$est - reduced_est$est
if (scale == "identity") {
  out <- tibble(est = point_est,
                ci_ll = point_est - stats::qnorm(0.975) * sqrt(var_scaled),
                ci_ul = point_est + stats::qnorm(0.975) * sqrt(var_scaled))
} else if (scale == "logit") {
  tmp <- stats::qlogis(point_est)
  out <- tibble(est = ifelse(is.na(tmp), 0, point_est),
                ci_ll = ifelse(is.na(tmp),
                               0,
                               stats::plogis(stats::qlogis(point_est) -
                                             stats::qnorm(0.975) *
                                             sqrt(var_scaled))
                               ),
                ci_ul = ifelse(is.na(tmp),
                               0,
                               stats::plogis(stats::qlogis(point_est) +
                                             stats::qnorm(0.975) *
                                             sqrt(var_scaled))
                               )
                )
} else {
```

```
    tmp <- log(point_est)
    out <- tibble(est = ifelse(is.na(tmp), 0, point_est),
                  ci_ll = ifelse(is.na(tmp),
                                 0,
                                 exp(log(point_est) - stats::qnorm(0.975) *
                                       sqrt(var_scaled))),
                  ci_ul = ifelse(is.na(tmp),
                                 0,
                                 exp(log(point_est) - stats::qnorm(0.975) *
                                       sqrt(var_scaled))))
  }
  out
}
# get estimate, CI based on averaging over the 10 random starts
get_avg_est_ci <- function(vimp_lst) {
  ests <- unlist(lapply(vimp_lst, function(x)
    if (length(x) > 1) {
      x$est
    } else {
      NA
    }))
  cis <- do.call(rbind, lapply(vimp_lst,
                               function(x)
                                 if (length(x) > 1) {
                                   x$ci
                                 } else {
                                   NA
                                 }))
  pvals <- unlist(lapply(vimp_lst, function(x)
    if(length(x) > 1) {x$p_value} else {NA}))
  est <- mean(ests, na.rm = TRUE)
  ci <- colMeans(cis, na.rm = TRUE)
  pval <- mean(pvals, na.rm = TRUE)
  return(list(est = est, ci = ci, p = pval))
}
# ------------------------------------------------------------------------
# Average predictiveness over the 10 random starts
# ------------------------------------------------------------------------
# get the risk estimate, CI based on averaging over the 10 random starts
# @param vimp_lst the list of estimates
get_avg_risk_ci <- function(vimp_lst) {
  ests_full <- unlist(lapply(vimp_lst, function(x) x$risk_full))
  ests_redu <- unlist(lapply(vimp_lst, function(x) x$risk_reduced))
  cis_full <- do.call(rbind, lapply(vimp_lst, function(x) x$risk_ci_full))
  cis_redu <- do.call(rbind, lapply(vimp_lst, function(x) x$risk_ci_reduced))
  est_full <- mean(ests_full)
  est_redu <- mean(ests_redu)
  ci_full <- colMeans(cis_full)
  ci_redu <- colMeans(cis_redu)
  return(list(risk_full = est_full, risk_reduced = est_redu,
              risk_ci_full = ci_full, risk_ci_reduced = ci_redu))
}
# ------------------------------------------------------------------------
```

```r
# Functions for printing, plotting
# ------------------------------------------------------------------------
# get names for multiple assays, all antigens
# @param X the matrix of predictors
# @param assays a character vector with the assays of interest to include
#               (e.g., c("IgG", "IgA"))
# @param assays_to_exclude a character string with the assays to exclude
#                          (e.g., "IgG3")
get_nms_group_all_antigens <- function(X, assays, assays_to_exclude = "") {
  # set all vars to be false
  vars <- rep(FALSE, ncol(X))
  # set vars with assay in name to be true
  # may be more than one
  for (i in 1:length(assays)) {
    if (assays_to_exclude != "") {
      vars[grepl(assays[i], names(X)) &
             !grepl(assays_to_exclude, names(X))] <- TRUE
    } else {
      vars[grepl(assays[i], names(X))] <- TRUE
    }
    if (assays[i] == "phago") {
      vars[grepl("ADCP1", names(X))] <- TRUE
    }
  }
  return(vars)
}

# get names for multiple antigens, all assays
# @param X the matrix of predictors
# @param antigens a character vector with the antigens of interest
get_nms_group_all_assays <- function(X, antigens) {
  # set all vars to be false
  vars <- rep(FALSE, ncol(X))
  # set vars with assay in name to be true
  # may be more than one
  for (i in 1:length(antigens)) {
    vars[grepl(antigens[i], names(X))] <- TRUE
  }
  return(vars)
}

# get the names of a group, for importance
# @param X the matrix of predictors
# @param assay a character string with the name of the assay of interest
# @param antigens a character string with the antigen of interest
get_nms_group <- function(X, assay, antigen) {
  vars <- rep(FALSE, ncol(X))
  # toggle on the ones in assay/antigen group
  vars[grepl(assay, names(X)) & grepl(antigen, names(X))] <- TRUE
  return(vars)
}
# get the names of an individual, for importance
# @param X the matrix of predictors
```

```r
# @param nm_ind a string with the name of the variable of interest
get_nms_ind <- function(X, nm_ind) {
  vars <- rep(FALSE, ncol(X))
  # toggle on the one, for vimp compared to baseline only
  vars[grepl(nm_ind, names(X))] <- TRUE
  return(vars)
}

# make nice names for Learner/Screen combos
# @param x the original string
# @param str the part to remove
remove_str <- function(x, str) {
  if (length(x) > 1) {
    return(x[!grepl(str, x)])
  } else {
    return(x)
  }
}
# make learner names nice for printing
# @param learners the names of the learners
make_nice_learner_name <- function(learners) {
  no_dots <- strsplit(as.character(learners), ".", fixed = TRUE)
  no_sl <- lapply(no_dots, function(x) remove_str(x, "SL"))
  no_skinny <- lapply(no_sl, function(x) remove_str(x, "skinny"))
  learner_nms <- unlist(lapply(no_skinny, function(x)
    paste(x, collapse = "_")))
  return(learner_nms)
}
# make screen names nice for printing
# @param screens the names of the screens
make_nice_screen_name <- function(screens) {
  no_underscores <- strsplit(as.character(screens), "_", fixed = TRUE)
  no_screen_plus_exposure <- lapply(no_underscores,
                                    function(x)
                                      x[!grepl("screen", x) &
                                          !grepl("plus", x) &
                                          !grepl("exposure", x)])
  no_all <- lapply(no_screen_plus_exposure, function(x) remove_str(x, "All"))
  screen_nms <- unlist(lapply(no_all, function(x) paste(x, collapse = "_")))
  return(screen_nms)
}
# make nice variable names for printing
# @param varname the variable name
# @param antigen the name of the antigen
# @param assay the name of the assay
make_nice_variable_name <- function(varname, antigen, assay) {
  if (stringr::str_count(varname, "_") > 1) {
    gsub(paste0(antigen, "_"), "",
         gsub(paste0(assay, "_"),
              "", varname))
  } else {
    varname
  }
```

```
}
# get immunoassay set from character vector: T cells, Ab only, T cells and Ab
# @param vec the character vector of interest
get_immunoassay_set <- function(vec, variables = FALSE) {
  get_one_immunoassay <- function(x, variables = FALSE) {
    if (variables) {
      ret_vec <- c("T Cell variables", "Ab variables",
                   "T Cell and Ab variables", "No markers")
    } else {
      ret_vec <- c("T Cell variables", "Ab variables",
                   "T Cell and Ab", "No markers")
    }
    if (grepl("T Cells", x)) {
      ret_init <- ret_vec[c(1, 3)]
      if (grepl("IgG", x) | grepl("IgA", x) |
          grepl("IgG3", x) | grepl("Fx Ab", x)) {
        ret <- ret_init[2]
      } else {
        ret <- ret_init[1]
      }
    } else if (!grepl("No markers", x) & !grepl("All markers", x)) {
      ret <- ret_vec[2]
    } else if (grepl("All markers", x)) {
      ret <- ret_vec[3]
    } else {
      ret <- ret_vec[4]
    }
    return(ret)
  }
  ret <- apply(matrix(vec), 1, get_one_immunoassay, variables = variables)
  return(as.vector(ret))
}
# make a plot for a given assay and antigen
# @param vimp_tibble a tibble of variable importance objects
# @param assay the assay of interest
# @param antigen the antigen of interest
# @param risk_type the chosen predictiveness measure (e.g., "auc", "r_squared")
# @param main_font_size the size of main font in the plot
# @param point_size the size of points in the plot
# @param x_lim the x-axis limits
# @param cols colors for points
# @param cols2 colors for ?
assay_antigen_plot <- function(vimp_tibble = NULL, assay = "IgG",
                               antigen = "ANYHIV",
                               risk_type = "auc", main_font_size = 5,
                               point_size = 3,
                               x_lim = c(0, 1), cols = "black",
                               cols2 = NULL) {
  if (!is.null(cols2)) {
    current_tibble <- (vimp_tibble %>% filter(assay_group == assay,
                                              antigen_group == antigen))
    current_tibble$t_cell_type <- apply(
      matrix(grepl("CD8", current_tibble$var_name)),
```

```
    1, function(x) ifelse(x, "CD8", "CD4")
    )
  vimp_plot <- current_tibble %>%
    ggplot(aes(x = est,
               y = factor(var_name,
                          levels = var_name[order(est, decreasing = TRUE)],
                          labels = var_name[order(est, decreasing = TRUE)]),
               color = t_cell_type)) +
    geom_errorbarh(aes(xmin = cil, xmax = ciu, color = greater_zero),
                   size = point_size / 2,
                   show.legend = FALSE) +
    geom_point(size = point_size) +
    geom_vline(xintercept = 0, color = "red", linetype = "dotted") +
    scale_color_manual(breaks = c("CD4", "CD8"), values = c(cols2, cols)) +
    xlim(x_lim) +
    ylab("Variable name") +
    xlab(paste0("Variable importance estimate: difference in CV-",
                ifelse(risk_type == "r_squared", expression(R^2), "AUC"))) +
    labs(color = "T Cell type") +
    theme(legend.position = c(0.025, 0.15),
          axis.text.y = element_text(size = main_font_size),
          text = element_text(size = main_font_size),
          axis.title = element_text(size = main_font_size),
          axis.text.x = element_text(size = main_font_size),
          axis.title.x = element_text(margin = ggplot2::margin(t = 20,
                                                               r = 0,
                                                               b = 0,
                                                               l = 0),
                                      size = main_font_size),
          plot.margin=unit(c(1,0.5,0,0),"cm")) # top, right, bottom, left
} else {
  vimp_plot <- vimp_tibble %>%
    filter(assay_group == assay, antigen_group == antigen) %>%
    ggplot(aes(x = est, y = factor(var_name,
                                   levels = var_name[order(est,
                                                           decreasing =
                                                             TRUE)],
                                   labels = var_name[order(est,
                                                           decreasing =
                                                             TRUE)]))) +
    geom_errorbarh(aes(xmin = cil, xmax = ciu, color = greater_zero),
                   size = point_size / 2) +
    geom_point(size = point_size) +
    geom_vline(xintercept = 0, color = "red", linetype = "dotted") +
    scale_color_manual(values = cols) +
    xlim(x_lim) +
    ylab("Variable name") +
    xlab(paste0("Variable importance estimate: difference in CV-",
                ifelse(risk_type == "r_squared", expression(R^2), "AUC"))) +
    guides(color = FALSE) +
    theme(axis.text.y = element_text(size = main_font_size),
          text = element_text(size = main_font_size),
          axis.title = element_text(size = main_font_size),
```

```
            axis.text.x = element_text(size = main_font_size),
            axis.title.x = element_text(margin = ggplot2::margin(t = 20,
                                                                 r = 0,
                                                                 b = 0,
                                                                 l = 0),
                                        size = main_font_size),
            plot.margin=unit(c(1,0.5,0,0),"cm")) # top, right, bottom, left
  }
  vimp_plot
}

# list of plots for a given assay type, one for each antigen
# @param vimp_tibble a tibble of variable importance objects
# @param assay the assay of interest
# @param antigen the antigen of interest
# @param risk_type the chosen predictiveness measure (e.g., "auc", "r_squared")
# @param main_font_size the size of main font in the plot
# @param point_size the size of points in the plot
# @param x_lim the x-axis limits
# @param cols colors for points
# @param cols2 colors for ?
assay_antigen_plot_list <- function(vimp_tibble = NULL, assay = "IgG",
                                    antigen = "ANYHIV",
                                    risk_type = "auc",
                                    main_font_size = 5, point_size = 3,
                                    x_lim = c(0, 1), cols = "black",
                                    cols2 = NULL) {
  plot_lst <- lapply(as.list(antigens), assay_antigen_plot,
                     vimp_tibble = vimp_tibble, assay = assay,
                     risk_type = risk_type,
                     main_font_size = main_font_size, point_size = point_size,
                     x_lim = x_lim, cols = cols, cols2 = cols2)
  return(plot_lst)
}
```

The following code runs the full Super Learner with a reduced library of candidate learners. This is for illustration only — we recommend using a large library of candidate learners, as we did in the original analysis. Please note, however, that the following code chunk may take several minutes to run. If the code chunk does not run, a useful first debugging step is to make sure that you have installed all of the required packages.

```
# load required libraries and functions
library("methods")
library("SuperLearner")
library("e1071")
library("glmnet")
library("xgboost")
library("earth")
library("dplyr")
# only run this if something has changed
# devtools::install_local("HVTN505_2019-4-25.tar.gz")
library("HVTN505")
# only run this if something has changed
# devtools::install_github("bdwilliamson/vimp", upgrade = "never")
library("vimp")
```

```r
library("kyotil")
library("argparse")

num_cores <- parallel::detectCores()
# print(num_cores)
# read in screens/algos and utility functions (shown above)
source(paste0("code/sl_screens.R"))
source(paste0("code/utils.R"))

# ------------------------------------------------------------------------------
# pre-process the data
# ------------------------------------------------------------------------------
# read in the full dataset
data("dat.505", package = "HVTN505")
# read in the super learner variables
# even if there is a warning message, it still exists
suppressWarnings(data("var.super", package = "HVTN505"))
# note that "var.super" contains individual vars for vaccine-matched antigens,
# and for vaccine-mismatched antigens, has either individual var (if only one)
# or PC1 and/or MDW (only PC1 if cor(PC1, MDW) > 0.9)

# scale vaccine recipients to have mean 0, sd 1 for all vars
# scale vaccine recipients to have mean 0, sd 1 for all vars
for (a in var.super$varname) {
  dat.505[[a]] <- as.vector(
    scale(dat.505[[a]],
          center = mean(dat.505[[a]][dat.505$trt == 1]),
          scale = sd(dat.505[[a]][dat.505$trt == 1])))
  dat.505[[a%.%"_bin"]] <- as.vector(
    scale(dat.505[[a%.%"_bin"]],
          center = mean(dat.505[[a%.%"_bin"]][dat.505$trt == 1]),
          scale = sd(dat.505[[a%.%"_bin"]][dat.505$trt == 1])))
}
for (a in c("age", "BMI", "bhvrisk")) {
  dat.505[[a]] <- as.vector(
    scale(dat.505[[a]],
          center = mean(dat.505[[a]][dat.505$trt == 1]),
          scale = sd(dat.505[[a]][dat.505$trt == 1])))
}

# set up X, Y for super learning
X_markers <- dat.505 %>%
  select(var.super$varname, paste0(var.super$varname, "_bin"))
X_exposure <- dat.505 %>%
  as_tibble() %>%
  select(age, BMI, bhvrisk)
X <- tibble::tibble(ptid = dat.505$ptid, trt = dat.505$trt,
                    weight = dat.505$wt) %>%
    bind_cols(X_exposure, X_markers)
X_none_all <- tibble::tibble(ptid = dat.505$ptid, trt = dat.505$trt,
                    weight = dat.505$wt) %>%
    bind_cols(X_exposure)
Y <- tibble(Y = dat.505$case)
```

```r
vaccinees <- dplyr::bind_cols(Y, X) %>%
  filter(trt == 1) %>%
  select(-trt)
Y_vaccine <- vaccinees$Y
weights_vaccine <- vaccinees$weight
X_vaccine <- vaccinees %>%
  select(-Y, -weight, -ptid)
X_none <- X_none_all %>%
  filter(trt == 1) %>%
  select(-trt, -ptid, -weight)
# match the rows in vaccinees to get Z, C
all_cc_vaccine <- Z_plus_weights %>%
  filter(ptid %in% vaccinees$ptid, trt == 1)
# pull out the participants who are NOT in the cc cohort and received the vaccine
all_non_cc_vaccine <- Z_plus_weights %>%
  filter(!(ptid %in% vaccinees$ptid), trt == 1)
# put them back together
phase_1_data_vaccine <- dplyr::bind_rows(all_cc_vaccine, all_non_cc_vaccine) %>%
  select(-trt)
Z_vaccine <- phase_1_data_vaccine %>%
  select(-ptid, -weight)
all_ipw_weights_vaccine <- phase_1_data_vaccine %>%
  pull(weight)
C <- (phase_1_data_vaccine$ptid %in% vaccinees$ptid)

V_outer <- 5
V_inner <- length(Y_vaccine) - 1

# ------------------------------------------------------------------------------
# run super learner, with leave-one-out cross-validation and all screens
# do 10 random starts, average over these
# use assay groups as screens
# ------------------------------------------------------------------------------
# ensure reproducibility
set.seed(4747)
seeds <- round(runif(10, 1000, 10000)) # average over 10 random starts
fits <- parallel::mclapply(seeds, FUN = run_cv_sl_once,
                           Y = Y_vaccine,
                           X_mat = X_vaccine,
                           family = "binomial",
                           Z = Z_vaccine,
                           C = C, z_lib = "SL.glm",
                           obsWeights = weights_vaccine,
                           all_weights = all_ipw_weights_vaccine,
                           scale = "logit",
                           sl_lib = SL_library[1],
                           method = "method.CC_nloglik",
                           cvControl =
                             list(V = V_outer, stratifyCV = TRUE),
                           innerCvControl =
                             list(list(V = V_inner)),
                           vimp = FALSE,
                           mc.cores = num_cores
```

```
)
sl_fits_varset_11_all <- fits
```

The code to reproduce the results of the original analysis is provided in the Appendix, and was designed to run on a high-performance cluster (HPC) computer (it takes multiple hours to run).

3.4 Variable Importance

We use the `vimp` package to compute variable importance estimates and confidence intervals. In all cases, we consider variable importance as a summary of the true data-generating distribution. Since our outcome is binary, we estimate variable importance using the population difference in area under the receiver operating characteristic curve (AUC) (Williamson et al. 2020). We interpret the AUC-based importance as the difference in our ability to discriminate between cases and controls based on using a group of covariates. Load the `vimp` package using the following code:

```
# only run the next line if you haven't already installed 'vimp' version 2.1.9 or later
# devtools::install_github("bdwilliamson/vimp", upgrade = "never")
library("vimp")
```

We next ran multiple Super Learners with reduced sets of covariates. Again, we originally ran this code on an HPC computer, and recommend that any final analyses use the full library of candidate learners. However, in the code below, we use a smaller library for illustration. Please note that this code chunk may take a couple of minutes to run.

```
# only include the following variable sets:
assays <- unique(var.super$assay)
antigens <- unique(var.super$antigen)
# 1. None (baseline variables only)
var_set_none <- rep(FALSE, ncol(X_markers))
# 2. IgG + IgA (all antigens)
var_set_igg_iga <- get_nms_group_all_antigens(X_markers,
                                              assays = c("IgG", "IgA"),
                                              assays_to_exclude = "IgG3")
# 3. IgG3
var_set_igg3 <- get_nms_group_all_antigens(X_markers, assays = "IgG3")
# 4. T cells (all antigens)
var_set_tcells <- get_nms_group_all_antigens(X_markers,
                                             assays = c("CD4", "CD8"))
# 5. Fx Ab (all antigens)
var_set_fxab <- get_nms_group_all_antigens(X_markers,
                                           assays = c("phago", "R2a", "R3a"))
# 6. 1+2+3
var_set_igg_iga_igg3 <- get_nms_group_all_antigens(X_markers,
                                                   assays =
                                                     c("IgG", "IgA", "IgG3"))
# 7. 1+2+4
var_set_igg_iga_tcells <- get_nms_group_all_antigens(X_markers,
                                                     assays =
                                                       c("IgG", "IgA",
                                                         "CD4", "CD8"),
                                                     assays_to_exclude =
                                                       "IgG3")
# 8. 1+2+3+4
var_set_igg_iga_igg3_tcells <- get_nms_group_all_antigens(X_markers,
```

```
                                                    assays = c("IgG",
                                                               "IgA",
                                                               "IgG3",
                                                               "CD4",
                                                               "CD8"))
# 9. 1+2+3+5
var_set_igg_iga_igg3_fxab <- get_nms_group_all_antigens(X_markers,
                                                        assays = c("IgG",
                                                                   "IgA",
                                                                   "IgG3",
                                                                   "phago",
                                                                   "R2a",
                                                                   "R3a"))
# 10. 1+4+5
var_set_tcells_fxab <- get_nms_group_all_antigens(X_markers,
                                                  assays = c("CD4", "CD8",
                                                             "phago", "R2a",
                                                             "R3a"))
# 11. All
var_set_all <- rep(TRUE, ncol(X_markers))
# 12--14: extra runs to get variable importance
var_set_igg3_fxab <- get_nms_group_all_antigens(X_markers,
                                                assays = c("IgG3", "phago",
                                                           "R2a", "R3a"))
var_set_igg_iga_tcells_fxab <- get_nms_group_all_antigens(X_markers,
                                                          assays = c("IgG",
                                                                     "IgA",
                                                                     "CD4",
                                                                     "CD8",
                                                                     "phago",
                                                                     "R2a",
                                                                     "R3a"),
                                                          assays_to_exclude =
                                                            "IgG3")
var_set_igg3_tcells_fxab <- get_nms_group_all_antigens(X_markers,
                                                       assays = c("IgG3",
                                                                  "CD4", "CD8",
                                                                  "phago",
                                                                  "R2a",
                                                                  "R3a"))

var_set_names <- c("1_baseline_exposure", "2_igg_iga", "3_igg3",
                   "4_tcells", "5_fxab",
                   "6_igg_iga_igg3", "7_igg_iga_tcells",
                   "8_igg_iga_igg3_tcells",
                   "9_igg_iga_igg3_fxab", "10_tcells_fxab",
                   "11_all",
                   "12_igg3_fxab", "13_igg_iga_tcells_fxab",
                   "14_igg3_tcells_fxab")

# set up a matrix of all
var_set_matrix <- rbind(var_set_none, var_set_igg_iga,
                        var_set_igg3,
```

```
                        var_set_tcells, var_set_fxab,
                        var_set_igg_iga_igg3,
                        var_set_igg_iga_tcells,
                        var_set_igg_iga_igg3_tcells,
                        var_set_igg_iga_igg3_fxab,
                        var_set_tcells_fxab,
                        var_set_all, var_set_igg3_fxab,
                        var_set_igg_iga_tcells_fxab,
                        var_set_igg3_tcells_fxab)
for (i in (1:14)[-11]) {
  job_id <- i
  this_var_set <- var_set_matrix[job_id, ]
  cat("\n Running ", var_set_names[job_id], "\n")

  X_markers_varset <- X_markers %>%
    select(names(X_markers)[this_var_set])
  X_exposure <- dat.505 %>%
    as_tibble() %>%
    select(age, BMI, bhvrisk)
  X <- tibble::tibble(ptid = dat.505$ptid, trt = dat.505$trt,
                      weight = dat.505$wt) %>%
    bind_cols(X_exposure, X_markers_varset)
  weights <- dat.505$wt
  Y <- tibble(Y = dat.505$case)
  vaccinees <- dplyr::bind_cols(Y, X) %>%
    filter(trt == 1) %>%
    select(-trt)
  Y_vaccine <- vaccinees$Y
  weights_vaccine <- vaccinees$weight
  X_vaccine <- vaccinees %>%
    select(-Y, -weight, -ptid)
  # match the rows in vaccinees to get Z, C
  all_cc_vaccine <- Z_plus_weights %>%
    filter(ptid %in% vaccinees$ptid, trt == 1)
  # pull out the participants who are NOT in the cc cohort
  # and received the vaccine
  all_non_cc_vaccine <- Z_plus_weights %>%
    filter(!(ptid %in% vaccinees$ptid), trt == 1)
  # put them back together
  phase_1_data_vaccine <- dplyr::bind_rows(all_cc_vaccine,
                                           all_non_cc_vaccine) %>%
    select(-trt)
  Z_vaccine <- phase_1_data_vaccine %>%
    select(-ptid, -weight)
  all_ipw_weights_vaccine <- phase_1_data_vaccine %>%
    pull(weight)
  C <- (phase_1_data_vaccine$ptid %in% vaccinees$ptid)

  V_outer <- 5
  V_inner <- length(Y_vaccine) - 1

  # get the SL library
  # if var_set_none, then don't need screens; otherwise do
```

```r
  if (job_id == 1) {
    sl_lib <- methods[1]
  } else {
    sl_lib <- SL_library[1]
  }
  # ------------------------------------------------------------------------------
  # run super learner, with leave-one-out cross-validation and all screens
  # do 10 random starts, average over these
  # ------------------------------------------------------------------------------
  # ensure reproducibility
  set.seed(4747)
  seeds <- round(runif(10, 1000, 10000)) # average over 10 random starts
  fits <- parallel::mclapply(seeds, FUN = run_cv_sl_once, Y = Y_vaccine,
                             X_mat = X_vaccine,
                             family = "binomial",
                             C = C, Z = Z_vaccine, z_lib = "SL.glm",
                             obsWeights = weights_vaccine,
                             all_weights = all_ipw_weights_vaccine,
                             scale = "logit",
                             sl_lib = sl_lib,
                             method = "method.CC_nloglik",
                             cvControl = list(V = V_outer, stratifyCV = TRUE),
                             innerCvControl = list(list(V = V_inner)),
                             vimp = FALSE,
                             mc.cores = num_cores
  )
  # name the object so that we can refer to it later
  eval(parse(text = paste0("sl_fits_varset_",
                           var_set_names[job_id],
                           " <- fits")))
}
```

```
##
##  Running  1_baseline_exposure
##
##  Running  2_igg_iga
##
##  Running  3_igg3
##
##  Running  4_tcells
##
##  Running  5_fxab
##
##  Running  6_igg_iga_igg3
##
##  Running  7_igg_iga_tcells
##
##  Running  8_igg_iga_igg3_tcells
##
##  Running  9_igg_iga_igg3_fxab
##
##  Running  10_tcells_fxab
##
##  Running  12_igg3_fxab
```

```
##
##  Running  13_igg_iga_tcells_fxab
##
##  Running  14_igg3_tcells_fxab
```

4 Results

4.1 Compiling Results and Creating Plots

The following code compiles results and creates plots, and must be run after all Super Learner fits have been obtained.

```
# results from full SL analysis
# includes CV-AUC plots

# ------------------------------------------------------------------------------------------------------
# set up directories, load required packages
# ------------------------------------------------------------------------------------------------------
library("SuperLearner")
library("cvAUC")
```

```
## Loading required package: ROCR
## Loading required package: data.table
##
## Attaching package: 'data.table'
## The following object is masked from 'package:kyotil':
##
##     last
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
##
## cvAUC version: 1.1.0
## Notice to cvAUC users: Major speed improvements in version 1.1.0
##
```

```
library("tidyr")
library("dplyr")
library("ggplot2")
library("cowplot")
```

```
##
## ******************************************************
## Note: As of version 1.0.0, cowplot does not change the
##   default ggplot2 theme anymore. To recover the previous
##   behavior, execute:
##   theme_set(theme_cowplot())
## ******************************************************
```

```r
theme_set(theme_cowplot())
library("vimp")
library("kyotil")
method <- "method.CC_nloglik"
library("xgboost")
# The palette with black:
cbbPalette <- c("#000000", "#E69F00", "#56B4E9", "#009E73", "#F0E442",
                "#0072B2", "#D55E00", "#CC79A7")

# load in the data to get weights
# read in the full dataset
data("dat.505", package = "HVTN505")
# read in the super learner variables
suppressWarnings(data("var.super", package = "HVTN505"))

weights_vaccine <- dat.505$wt[dat.505$trt == 1]
# ------------------------------------------------------------------------------
# load results objects:
# ------------------------------------------------------------------------------
# each is a length 10 list (one for each random start)
var_set_names <- c("1_baseline_exposure", "2_igg_iga", "3_igg3","4_tcells",
                   "5_fxab",
                   "6_igg_iga_igg3", "7_igg_iga_tcells",
                   "8_igg_iga_igg3_tcells",
                   "9_igg_iga_igg3_fxab", "10_tcells_fxab",
                   "11_all")
# objects are created above

# average the AUCs over the 10 folds, for each
var_set_labels <- c("No markers", "IgG + IgA", "IgG3", "T Cells",
                    "Fx Ab", "IgG + IgA + IgG3",
                    "IgG + IgA + T Cells", "IgG + IgA + IgG3 + T Cells",
                    "IgG + IgA + IgG3 + Fx Ab", "T Cells + Fx Ab",
                    "All markers")
for (i in 1:(length(var_set_names))) {
  eval(parse(
    text = paste0("all_aucs_i <- as_tibble(rbindlist(lapply(sl_fits_varset_",
                  var_set_names[i],
                  ", function(x) x$aucs)))")))
    all_aucs_i <- all_aucs_i %>%
      filter(!is.na(Learner))
    this_name <- paste(unlist(strsplit(var_set_names[i], "_",
                                       fixed = TRUE))[-1], collapse = "_")
    eval(parse(text =
    paste0("avg_aucs_", var_set_names[i], " <- all_aucs_i %>%
    group_by(Learner, Screen) %>%
    summarize(AUC = mean(AUC), ci_ll = mean(ci_ll), ci_ul = mean(ci_ul), .groups = 'drop') %>%
    mutate(assay = this_name, varset_label = var_set_labels[i])")))
}

# combine into a full tibble; add a column to each that is the assay
avg_aucs <- bind_rows(avg_aucs_1_baseline_exposure, avg_aucs_2_igg_iga,
                      avg_aucs_3_igg3,
```

```
                      avg_aucs_4_tcells, avg_aucs_5_fxab,
                      avg_aucs_6_igg_iga_igg3,
                      avg_aucs_7_igg_iga_tcells,
                      avg_aucs_8_igg_iga_igg3_tcells,
                      avg_aucs_9_igg_iga_igg3_fxab,
                      avg_aucs_10_tcells_fxab,
                      avg_aucs_11_all)
```

The following code chunk creates a function that makes plots:

```
# create the forest plot of the specified measure for each of the
# top learner and SL from each assay combination
# @param avgs the average predictiveness measures
# @param type the predictiveness measure ("auc" or "r_squared")
# @param main_font_size_forest the main font size for text in the forest plot
# @param main_font_size_lab the font size for axis labels
# @param sl_only should we only plot the SL estimates, or all learners?
# @param immunoassay should we color by immunoassay?
# @param colors the colors to use
# @param point_size what size should the points be?
plot_assays <- function(avgs = NULL, type = "auc", main_font_size_forest = 5,
                        main_font_size_lab = 5, sl_only = TRUE,
                        immunoassay = TRUE,
                        colors = NULL, point_size = 3, x_lim = c(0.4, 1),
                        lgnd_pos = c(0.56, 0.2)) {
  # if type is AUC, make correct labels
  if (type == "auc") {
    x_lab <- "CV-AUC"
    avgs <- avgs %>%
      mutate(measure = AUC)
  } else {
    x_lab <- expression(paste("CV-", R^2))
    # x_lim <- c(-1, 1)
    avgs <- avgs %>%
      mutate(measure = R2)
    # lgnd_pos <- c(0.65, 0.15)
  }
  # get SL and the top individual learner/screen combo from each assay
  top_learners_init <- avgs %>%
    group_by(assay) %>%
    arrange(desc(measure), .by_group = TRUE) %>%
    filter((Learner == "SL" & Screen == "All") | row_number() == 1) %>%
    mutate(learner_nm = make_nice_learner_name(Learner),
           screen_nm = make_nice_screen_name(Screen)) %>%
    ungroup() %>%
    arrange(desc(measure)) %>%
    mutate(ci_ll_truncated = pmax(ci_ll, x_lim[1]),
           ci_ul_truncated = pmin(ci_ul, x_lim[2]))

  # if sl_only, only use the SL algorithm for each assay
  if (sl_only) {
    top_learners <- top_learners_init %>%
      filter(Learner == "SL")
  } else {
```

```
  top_learners <- top_learners_init
}

# forest plot with the groups/learners from the analysis plan
top_learner_plot <- top_learners %>%
  ggplot(aes(y = measure,
      x = factor(paste0(Screen, "_", Learner, "_", assay),
      levels = paste0(Screen, "_", Learner, "_", assay)[order(measure)],
      labels = paste0(varset_label, " ", learner_nm, " ",
                      screen_nm)[order(measure)]),
      xend = factor(paste0(Screen, "_", Learner, "_", assay),
      levels = paste0(Screen, "_", Learner, "_", assay)[order(measure)],
      labels = paste0(varset_label, " ", learner_nm, " ",
                      screen_nm)[order(measure)]))) +
  geom_errorbar(aes(ymin = ci_ll, ymax = ci_ul), size = point_size) +
  geom_pointrange(aes(ymin = ci_ll_truncated, ymax = ci_ul_truncated),
  size = point_size) +
  ylab(x_lab) +
  xlab("") +
  scale_y_continuous(breaks = round(seq(x_lim[1], x_lim[2], 0.1), 1),
                     labels = as.character(round(seq(x_lim[1],
                                                     x_lim[2], 0.1), 1)),
                     limits = x_lim) +
  coord_flip() +
  theme(legend.position = "",
        axis.text.y = element_blank(),
        text = element_text(size = main_font_size_forest),
        axis.title = element_text(size = main_font_size_forest),
        axis.text.x = element_text(size = main_font_size_forest),
        axis.ticks = element_line(size = 1),
        axis.ticks.length = unit(0.5, "cm"),
        axis.title.x = element_text(margin = ggplot2::margin(t = 5,
            r = 0, b = 0, l = 0), size = main_font_size_forest),
        plot.margin=unit(c(1.25,0.5,0.5,0),"cm")) # top, right, bottom, left
if (immunoassay) {
  # add on a legend to top_learner_plot, color based on immunoassay set
  top_learner_plot <- top_learner_plot +
    geom_errorbar(aes(ymin = ci_ll,
                      ymax = ci_ul,
                      color = immunoassay_set),
                  size = point_size) +
    geom_pointrange(aes(ymin = ci_ll_truncated,
                        ymax = ci_ul_truncated,
                        color = immunoassay_set),
                    size = point_size) +
    scale_color_manual(values = colors) +
    labs(color = "Assay set") +
    theme(legend.position = lgnd_pos,
          axis.text.y = element_blank(),
          text = element_text(size = main_font_size_forest),
          axis.title = element_text(size = main_font_size_forest),
          axis.text.x = element_text(size = main_font_size_forest),
          axis.title.x = element_text(margin = ggplot2::margin(t = 5,
```

```
            r = 0, b = 0, l = 0), size = main_font_size_forest),
          legend.text = element_text(size = main_font_size_forest),
          plot.margin=unit(c(1.25,0.5,0.5,0),"cm"))

  # separate plot with nice names,
  # printed values of measures, based on immunoassays only
  top_learners_labels <- top_learners %>%
    ungroup() %>%
    mutate(lab_measure = paste0(format(round(measure, 3), nsmall = 3), " [",
                           format(round(ci_ll, 3), nsmall = 3), ", ",
                           format(round(ci_ul, 3), nsmall = 3), "]")) %>%
    select(varset_label, lab_measure)
  # melt to make a single "value" column
  top_learners_labels$var <- 1
  top_learners_labs <- reshape2::melt(top_learners_labels, id.var = "var")
  # tack on x, y coordinates
  top_learners_labs$x_coord <- apply(matrix(top_learners_labs$variable), 1,
  function(x) which(grepl(x, c("varset_label", "lab_measure"))) -
  1 + c(0.2, 0.5)[which(grepl(x, c("varset_label", "lab_measure")))])
  top_learners_labs$y_coord <- rep(rev(as.numeric(rownames(top_learners))),
   dim(top_learners_labs)[1]/length(as.numeric(rownames(top_learners))))
} else {
  # separate plot with nice names, printed values of the measures
  top_learners_labels <- top_learners %>%
    ungroup() %>%
    mutate(lab_measure = paste0(format(round(measure, 3), nsmall = 3), " [",
                           format(round(ci_ll, 3), nsmall = 3), ", ",
                           format(round(ci_ul, 3), nsmall = 3), "]")) %>%
    select(varset_label, learner_nm, screen_nm, lab_measure)
  # melt to make a single "value" column
  top_learners_labels$var <- 1
  top_learners_labs <- reshape2::melt(top_learners_labels, id.var = "var")
  # tack on x, y coordinates
  top_learners_labs$x_coord <- apply(matrix(top_learners_labs$variable), 1,
  function(x) which(grepl(x,
      c("varset_label", "learner_nm", "screen_nm", "lab_measure"))) -
      1 + c(0, 0.3, -0.3, 0.2)[which(grepl(x,
          c("varset_label", "learner_nm", "screen_nm", "lab_measure")))])
  top_learners_labs$y_coord <- rep(rev(as.numeric(rownames(top_learners))),
   dim(top_learners_labs)[1]/length(as.numeric(rownames(top_learners))))
}
# make the plot
top_learner_nms_plot <- top_learners_labs %>%
  ggplot(aes(x = x_coord, y = y_coord, label = value)) +
  geom_text(size = main_font_size_lab, hjust = 0, vjust = 0.5) +
  xlim(c(min(top_learners_labs$x_coord), max(top_learners_labs$x_coord) + 0.75)) +
  ylim(c(1, max(top_learners_labs$y_coord))) +
  theme(legend.position="",
        axis.line=element_blank(),
        axis.text=element_blank(),
        text = element_text(size = main_font_size_lab),
        axis.title = element_blank(),
        axis.ticks = element_blank(),
```

```
          plot.margin=unit(c(0,0,0,0),"cm"),
          panel.background=element_blank(),
          panel.border=element_blank(),
          panel.grid.major=element_blank(),
          panel.grid.minor=element_blank(),
          plot.background=element_blank())


  return(list(top_learner_plot = top_learner_plot,
              top_learner_nms_plot = top_learner_nms_plot))
}
```

4.2 Super Learning Analysis

We ran the Super Learner algorithm for each of the 11 variable sets defined in Section 1 using the procedure defined in Section 2. We present the cross-validated AUC for the overall Super Learner from each of the variable sets. These are provided in Figure 1. In the figure, the error bars are colored by immunoassay set: antibody (Ab) variables (black), no markers (yellow), T cell variables (green), and T cell and Ab variables (blue).

In Figure 1, we see improved performance of the Super Learner when including marker variables compared to baseline risk variables only. Indeed, the Super Learner using only baseline variables and no markers has a cross-validated AUC of 0.755. Other variable sets have a smaller estimated cross-validated AUC, but this is likely due to the small candidate library that we used for this analysis. Our proposed estimator of HIV-1 infection risk among vaccinees, the cross-validated Super Learner, has an estimated AUC of 0.866 based on all markers, with a 95% confidence interval of [0.796, 0.914]. This is an improvement over the Super Learner based on only the baseline risk variables, with an estimated AUC of 0.755 [0.701, 0.802]. These results must be interpreted with the large caveat that we have used a restricted set of learners in our Super Learner library to reduce computation time. A fuller set of learners is recommended for any publication-ready analysis (indeed, when we used the full library of learners in the full published analysis, the Super Learner based on all markers had a much improved AUC over the Super Learner with baseline risk variables only).

```
# The palette with black:
cbbPalette <- c("#000000", "#E69F00", "#56B4E9", "#009E73",
                "#F0E442", "#0072B2", "#D55E00", "#CC79A7")
# load in code to plot CV performance (described above)
source("code/plot_assays.R")
# ------------------------------------------------------------------------------
# FIGURE 1: forest plot of CV-AUC for the top learner
#           and SL for each assay combination
# ------------------------------------------------------------------------------
title_font_size <- 18
main_font_size <- 5
fig_width <- fig_height <- 2590
y_title <- 0.96
auc_forest_plot_init <- plot_assays(avgs = avg_aucs, type = "auc",
                                    main_font_size_forest = main_font_size * 3,
                                    main_font_size_lab = main_font_size,
                                    sl_only = FALSE, immunoassay = FALSE,
                                    point_size = 1.5)
sl_plus_top_learner_auc_plot <- plot_grid(auc_forest_plot_init$top_learner_nms_plot,
                                          auc_forest_plot_init$top_learner_plot,
                                          nrow = 1, align = "h") +
```

```
  draw_label("Assay combination", size = title_font_size, x = 0.075,
             y = y_title) +
  draw_label("Algorithm", size = title_font_size, x = 0.175,
             y = y_title) +
  draw_label("Screen", size = title_font_size, x = 0.25,
             y = y_title) +
  draw_label("CV-AUC [95% CI]", size = title_font_size, x = 0.43,
             y = y_title)

# add on immunoassay set
avg_aucs <- avg_aucs %>%
  mutate(immunoassay_set = get_immunoassay_set(varset_label))

title_font_size <- 26
main_font_size_forest <- 31
main_font_size_lab <- 9.3
fig_width <- fig_height <- 2590
y_title <- 0.945
point_size <- 2
auc_forest_plot_plus_assay <- plot_assays(avgs = avg_aucs, type = "auc",
                                main_font_size_forest = main_font_size_forest,
                                main_font_size_lab = main_font_size_lab,
                                sl_only = TRUE, immunoassay = TRUE,
                                colors = cbbPalette,
                                point_size = point_size,
                                x_lim = c(0.4, 1.2),
                                lgnd_pos = c(0.6, 0.2))
auc_immunoassay_plot <- plot_grid(auc_forest_plot_plus_assay$top_learner_nms_plot,
                                            auc_forest_plot_plus_assay$top_learner_plot,
                                            nrow = 1, align = "h") +
  draw_label("Month 7", size = title_font_size, x = 0.07,
             y = y_title + 0.04,
             fontface = "bold") +
  draw_label("Marker Set", size = title_font_size, x = 0.08,
             y = y_title,
             fontface = "bold") +
  draw_label("CV-AUC", size = title_font_size, x = 0.35,
             y = y_title + 0.04,
             fontface = "bold") +
  draw_label("[95% CI]", size = title_font_size, x = 0.35,
             y = y_title,
             fontface = "bold")
auc_immunoassay_plot
```

4.3 Variable Importance Analysis

We estimated the variable importance for 10 groups: all marker variables, IgG + IgA, IgG3, T cells, functional antibodies, IgG + IgA and IgG3, IgG + IgA and T cells, IgG + IgA and IgG3 and T cells, IgG + IgA and IgG3 and functional antibodies, and T cells and functional antibodies. In each case, we used the cross-validated Super Learner predictions from the analysis based on using the appropriate set of marker variables and baseline risk variables as our estimate of the full regression function; we then used the cross-validated Super Learner based on the baseline risk variables only to define the reduced regression function. This defines

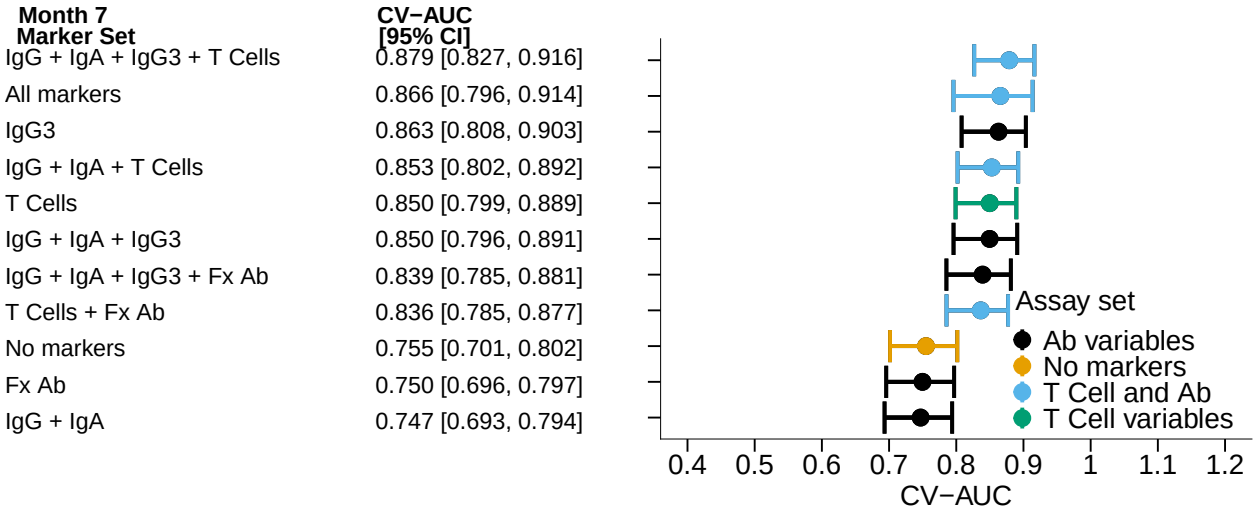

Figure 1: Forest plot of cross-validated AUC for the Super Learner and the top individual learner for each assay combination.

variable importance as the improvement in prediction performance from including a group of marker variables relative to including baseline variables only. The following code compiles the variable importance results for easy plotting:

```
# ----------------------------------------------------------------------------
# Variable importance plot for the different assay combinations:
# All markers yields the full regression fit
# Variable set 1 gives reduced for importance of all markers (group 8)
# Variable set 2 gives reduced for importance of Fx Ab + Tcells (group 7)
# Variable set 3 gives reduced for importance of IgG + IgA + Fx Ab (group 6)
# Variable set 4 gives reduced for importance of IgG + IgA + Tcells (group 5)
# Variable set 5 gives reduced for importance of Fx Ab (group 4)
# Variable set 6 gives reduced for importance of T cells (group 3)
# Variable set 7 gives reduced for importance of IgG + IgA (group 2)
# ----------------------------------------------------------------------------
# load the data
# read in the full dataset
data("dat.505", package = "HVTN505")
# read in the super learner variables
suppressWarnings(data("var.super", package = "HVTN505"))
# note that "var.super" contains individual vars for vaccine-matched antigens,
# and for vaccine-mismatched antigens, has either individual var (if only one)
# or PC1 and/or MDW (only PC1 if cor(PC1, MDW) > 0.9)

# scale vaccine recipients to have mean 0, sd 1 for all vars
# scale vaccine recipients to have mean 0, sd 1 for all vars
for (a in var.super$varname) {
  dat.505[[a]] <- as.vector(
    scale(dat.505[[a]], center = mean(dat.505[[a]][dat.505$trt == 1]),
          scale = sd(dat.505[[a]][dat.505$trt == 1]))
    )
  dat.505[[a%.%"_bin"]] <- as.vector(
    scale(dat.505[[a%.%"_bin"]],
          center = mean(dat.505[[a%.%"_bin"]][dat.505$trt == 1]),
```

```
          scale = sd(dat.505[[a%.%"_bin"]][dat.505$trt == 1]))
    )
}
for (a in c("age", "BMI", "bhvrisk")) {
  dat.505[[a]] <- as.vector(
    scale(dat.505[[a]],
          center = mean(dat.505[[a]][dat.505$trt == 1]),
          scale = sd(dat.505[[a]][dat.505$trt == 1]))
    )
}

# set up X, Y for super learning
X_markers <- dat.505 %>%
  select(var.super$varname, paste0(var.super$varname, "_bin"))
X_exposure <- dat.505 %>%
  as_tibble() %>%
  select(age, BMI, bhvrisk)
X <- tibble::tibble(ptid = dat.505$ptid, trt = dat.505$trt,
                    weight = dat.505$wt) %>%
     bind_cols(X_exposure, X_markers)
X_none_all <- tibble::tibble(ptid = dat.505$ptid, trt = dat.505$trt,
                    weight = dat.505$wt) %>%
     bind_cols(X_exposure)
Y <- tibble(Y = dat.505$case)
vaccinees <- dplyr::bind_cols(Y, X) %>%
  filter(trt == 1) %>%
  select(-trt)
Y_vaccine <- vaccinees$Y
weights_vaccine <- vaccinees$weight
X_vaccine <- vaccinees %>%
  select(-Y, -weight, -ptid)

# ------------------------------------------------------------------------------
# FIGURE 3, 4: do variable importance relative to baseline risk vars only
# ------------------------------------------------------------------------------
risk_type <- "auc"
scale <- "logit"

# set up the full fits
full_fit_11_all <- sl_fits_varset_11_all
full_fit_10_tcells_fxab <- sl_fits_varset_10_tcells_fxab
full_fit_9_igg_iga_igg3_fxab <- sl_fits_varset_9_igg_iga_igg3_fxab
full_fit_8_igg_iga_igg3_tcells <- sl_fits_varset_8_igg_iga_igg3_tcells
full_fit_7_igg_iga_tcells <- sl_fits_varset_7_igg_iga_tcells
full_fit_6_igg_iga_igg3 <- sl_fits_varset_6_igg_iga_igg3
full_fit_5_fxab <- sl_fits_varset_5_fxab
full_fit_4_tcells <- sl_fits_varset_4_tcells
full_fit_3_igg3 <- sl_fits_varset_3_igg3
full_fit_2_igg_iga <- sl_fits_varset_2_igg_iga

# reduced fits
reduced_fit_none <- sl_fits_varset_1_baseline_exposure
```

```r
# compute variable importance
vimp_all_markers <- get_cv_vim_precomputed(
  full_ests = avg_aucs_11_all,
  reduced_ests = avg_aucs_1_baseline_exposure,
  risk_type = risk_type,
  scale = scale)
vimp_tcells_fxab <- get_cv_vim_precomputed(
  full_ests = avg_aucs_10_tcells_fxab,
  reduced_ests = avg_aucs_1_baseline_exposure,
  risk_type = risk_type,
  scale = scale)
vimp_igg_iga_igg3_fxab <- get_cv_vim_precomputed(
  full_ests = avg_aucs_9_igg_iga_igg3_fxab,
  reduced_ests = avg_aucs_1_baseline_exposure,
  risk_type = risk_type,
  scale = scale)
vimp_igg_iga_igg3_tcells <- get_cv_vim_precomputed(
  full_ests = avg_aucs_8_igg_iga_igg3_tcells,
  reduced_ests = avg_aucs_1_baseline_exposure,
  risk_type = risk_type,
  scale = scale)
vimp_igg_iga_tcells <- get_cv_vim_precomputed(
  full_ests = avg_aucs_7_igg_iga_tcells,
  reduced_ests = avg_aucs_1_baseline_exposure,
  risk_type = risk_type,
  scale = scale)
vimp_igg_iga_igg3 <- get_cv_vim_precomputed(
  full_ests = avg_aucs_6_igg_iga_igg3,
  reduced_ests = avg_aucs_1_baseline_exposure,
  risk_type = risk_type,
  scale = scale)
vimp_fxab <- get_cv_vim_precomputed(
  full_ests = avg_aucs_5_fxab,
  reduced_ests = avg_aucs_1_baseline_exposure,
  risk_type = risk_type,
  scale = scale)
```

```
## Warning in stats::qlogis(point_est): NaNs produced
```

```r
vimp_tcells <- get_cv_vim_precomputed(
  full_ests = avg_aucs_4_tcells,
  reduced_ests = avg_aucs_1_baseline_exposure,
  risk_type = risk_type,
  scale = scale)
vimp_igg3 <- get_cv_vim_precomputed(
  full_ests = avg_aucs_3_igg3,
  reduced_ests = avg_aucs_1_baseline_exposure,
  risk_type = risk_type,
  scale = scale)
vimp_igg_iga <- get_cv_vim_precomputed(
  full_ests = avg_aucs_2_igg_iga,
  reduced_ests = avg_aucs_1_baseline_exposure,
  risk_type = risk_type,
  scale = scale)
```

```
## Warning in stats::qlogis(point_est): NaNs produced
```

```r
# combine together
vimp_tibble <- tibble(assay_grp = c("All markers",
                                    "T Cells + Fx Ab",
                                    "IgG + IgA + IgG3 + Fx Ab",
                                    "IgG + IgA + IgG3 + T Cells",
                                    "IgG + IgA + T Cells",
                                    "IgG + IgA + IgG3",
                                    "Fx Ab",
                                    "T Cells",
                                    "IgG3",
                                    "IgG + IgA"),
                      est = c(vimp_all_markers$est,
                              vimp_tcells_fxab$est,
                              vimp_igg_iga_igg3_fxab$est,
                              vimp_igg_iga_igg3_tcells$est,
                              vimp_igg_iga_tcells$est,
                              vimp_igg_iga_igg3$est,
                              vimp_fxab$est,
                              vimp_tcells$est,
                              vimp_igg3$est,
                              vimp_igg_iga$est),
                      cil = c(vimp_all_markers$ci_ll,
                              vimp_tcells_fxab$ci_ll,
                              vimp_igg_iga_igg3_fxab$ci_ll,
                              vimp_igg_iga_igg3_tcells$ci_ll,
                              vimp_igg_iga_tcells$ci_ll,
                              vimp_igg_iga_igg3$ci_ll,
                              vimp_fxab$ci_ll,
                              vimp_tcells$ci_ll,
                              vimp_igg3$ci_ll,
                              vimp_igg_iga$ci_ll),
                      ciu = c(vimp_all_markers$ci_ul,
                              vimp_tcells_fxab$ci_ul,
                              vimp_igg_iga_igg3_fxab$ci_ul,
                              vimp_igg_iga_igg3_tcells$ci_ul,
                              vimp_igg_iga_tcells$ci_ul,
                              vimp_igg_iga_igg3$ci_ul,
                              vimp_fxab$ci_ul,
                              vimp_tcells$ci_ul,
                              vimp_igg3$ci_ul,
                              vimp_igg_iga$ci_ul))
vimp_tibble <- tibble::add_column(vimp_tibble,
                                  immunoassay_set =
                                    get_immunoassay_set(vimp_tibble$assay_grp))
```

The results based on cross-validated AUC are presented in Figure 2. Here, we see that using the group of all marker variables results in an estimated increase in cross-validated AUC of only 0.111 over using the baseline risk variables only. Interestingly, the groups of IgG, IgA, IgG3, and T cell markers; IgG, IgA, and T cell markers; and T cell markers and those markers measured in functional antibody assays have estimated equivalent prediction performance. The most important group appears to be IgG3 markers, with an estimated increase in cross-validated AUC of approximately 0.108 compared to using baseline risk variables only. Again, these results must be interpreted with the large caveat that we have used a restricted set of learners in our Super Learner library to reduce computation time. A fuller set of learners is recommended for any

publication-ready analysis.

```
title_font_size <- 26
main_font_size_forest <- 31
main_font_size_lab <- 9.3
y_title <- 0.945
point_size <- 5
lgnd_pos <- c(0.75, 0.8)
# forest plot of vimp, with labels for the groups
vimp_forest_plot <- vimp_tibble %>%
  ggplot(aes(x = est,
             y = factor(assay_grp,
                        levels = assay_grp[order(est, decreasing = TRUE)],
                        labels = assay_grp[order(est, decreasing = TRUE)]))) +
  geom_errorbarh(aes(xmin = cil, xmax = ciu, color = immunoassay_set), size = point_size/2) +
  geom_point(size = point_size) +
  scale_color_manual(values = cbbPalette[-2]) +
  ylab("Month 7 Marker Set") +
  labs(color = "Assay set") +
  xlab(paste0("Variable importance estimate: difference in CV-",
              ifelse(risk_type == "r_squared", expression(R^2), "AUC"))) +
  theme(legend.position = lgnd_pos,
        axis.text.y = element_text(size = main_font_size_forest),
        text = element_text(size = main_font_size_forest),
        axis.title = element_text(size = main_font_size_forest),
        axis.text.x = element_text(size = main_font_size_forest),
        axis.title.x = element_text(margin = ggplot2::margin(t = 20, r = 0,
                                                             b = 0, l = 0),
                                    size = main_font_size_forest),
        plot.margin=unit(c(1,0.5,0,0),"cm")) # top, right, bottom, left
vimp_forest_plot
```

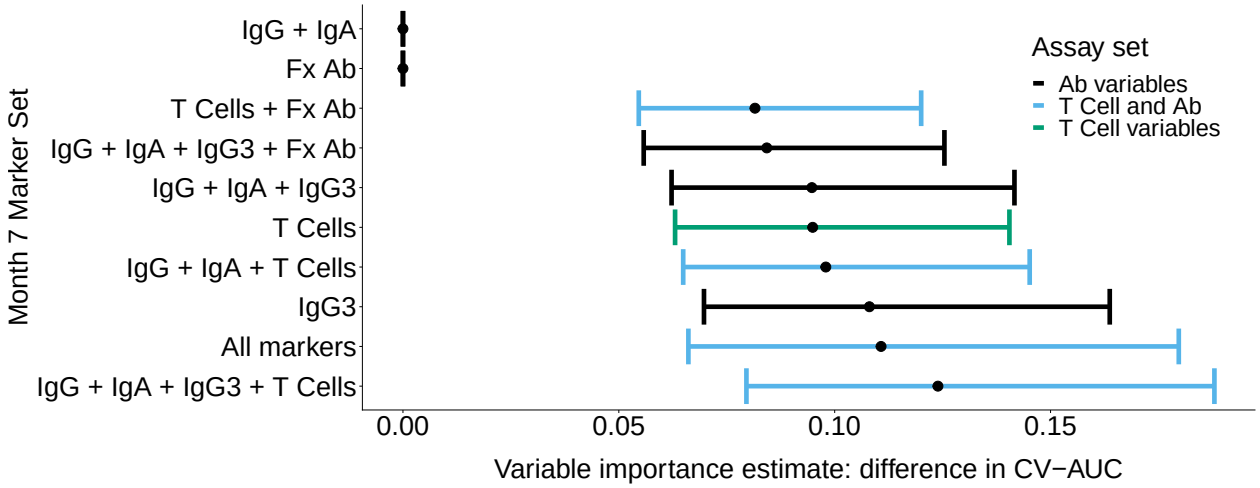

Figure 2: Estimated variable importance (based on the difference in AUCs relative to a model with baseline exposure variables only) for each assay combination.

5 Conclusions

In this analysis, we developed a model that predicts HIV-1 infection in vaccinees from month 7 marker variables and baseline risk variables. This model employs nested cross-validation to combine a flexible library of individual learners into a prediction algorithm that admits desirable asymptotic properties: namely, that the resulting cross-validated estimator is asymptotically guaranteed to have the same risk as the oracle estimator.

Based on a variable importance analysis using the difference in cross-validated AUCs, we see that: (i) adding IgG and IgA variables only to baseline variables results in improved prediction performance; (ii) adding functional antibody variables and IgG3 variables to IgG and IgA variables and baseline variables results in improved prediction performance over adding IgG and IgA variables alone to baseline variables; (iii) adding T cell variables or functional antibody variables only results in the largest increase in prediction performance among all individual groups; and (iv) that combining T cell and antibody variables together results in a further incremental improvement over T cells alone, but only if all antibody classes are present.

The main limitation of this analysis is that, to save computation time in this vignette, we used a small library of candidate learning algorithms. The Appendix contains code that may be used to fully reproduce the analysis in Neidich et al. (2019). Additionally, we did not show here results based on cross-validated R^2: due to the small number of events, cross-validated R^2 appears to be a poor estimator both of prediction performance and of variable importance. This is in line with previous work suggesting that R^2 may not always be appropriate for rare binary endpoints. We suspect that with a larger number of events the behavior of the R^2 estimators, and the conclusions drawn from using cross-validated R^2 to define both prediction performance and variable importance, would be more in line with the conclusions drawn from using cross-validated AUC.

6 Appendix

Code to reproduce the full Super Learning analysis is provided here. The original analysis of the HVTN 505 data (Neidich et al. 2019) used a previous version of the `vimp` package (specifically, `vimp` version 1.3.0); in this version of the package, hypothesis testing was not available, and the regressions based on the group of covariates of interest (e.g., IgG + IgA variables) and the group of remaining covariates (e.g., all variables besides IgG + IgA) were estimated on the same data. In the updated `vimp` version used here (version 2.1.4), it is possible to do hypothesis testing and the inverse weighting is properly handled. However, the hypothesis testing relies on the regressions being estimated on separate splits of the data; we do not do that here to harmonize this analysis with the original analysis.

```
#!/usr/local/bin/Rscript

# run the super learner
# make sure that it is CV.SL, averaged over 10 random starts

# load required libraries and functions
library("methods")
library("SuperLearner")
# library("future")
# library("future.apply")
library("e1071")
library("glmnet")
library("xgboost")
library("earth")
library("dplyr")
# only run this if something has changed
# install.packages("HVTN505_2019-4-9.tar.gz", type = "source", repos = NULL)
```

```r
library("HVTN505")
# only run this if something has changed
# devtools::install_github("bdwilliamson/vimp", upgrade = "never")
library("vimp", lib.loc = .libPaths()[2])
library("kyotil")
library("argparse")
# only run this if something has changed
# devtools::install_github("bdwilliamson/vimp", upgrade = "never")

# set up code directory
if (!is.na(Sys.getenv("RSTUDIO", unset = NA))) { # if running locally
  code_dir <- "code/"
  data_dir <- "data/"
  # load vimp from user library (to make sure it has correct version)
  library("vimp")
} else {
  code_dir <- ""
  # load vimp from user library (to make sure it has correct version)
  library("vimp", lib.loc = .libPaths()[2])
}
num_cores <- parallel::detectCores()
print(num_cores)
source(paste0(code_dir, "sl_screens.R")) # set up the screen/algorithm combinations
source(paste0(code_dir, "utils.R")) # get CV-AUC for all algs

# ------------------------------------------------------------------------------
# pre-process the data
# ------------------------------------------------------------------------------
# read in the full dataset
data("dat.505", package = "HVTN505")
# read in the super learner variables
data("var.super", package = "HVTN505") # even if there is a warning message, it still exists
# note that "var.super" contains individual vars for vaccine-matched antigens,
# and for vaccine-mismatched antigens, has either individual var (if only one)
# or PC1 and/or MDW (only PC1 if cor(PC1, MDW) > 0.9)

# scale vaccine recipients to have mean 0, sd 1 for all vars
for (a in var.super$varname) {
  dat.505[[a]] <- as.vector(
    scale(dat.505[[a]],
          center = mean(dat.505[[a]][dat.505$trt == 1]),
          scale = sd(dat.505[[a]][dat.505$trt == 1]))
    )
  dat.505[[a%.%"_bin"]] <- as.vector(
    scale(dat.505[[a%.%"_bin"]],
          center = mean(dat.505[[a%.%"_bin"]][dat.505$trt == 1]),
          scale = sd(dat.505[[a%.%"_bin"]][dat.505$trt == 1]))
    )
}
for (a in c("age", "BMI", "bhvrisk")) {
  dat.505[[a]] <- as.vector(
    scale(dat.505[[a]],
          center = mean(dat.505[[a]][dat.505$trt == 1]),
```

```
          scale = sd(dat.505[[a]][dat.505$trt == 1]))
    )
}

# set up X, Y for super learning
X_markers <- dat.505 %>%
  select(var.super$varname, paste0(var.super$varname, "_bin"))
X_exposure <- dat.505 %>%
  select(age, BMI, bhvrisk)
X <- tibble::tibble(ptid = dat.505$ptid, trt = dat.505$trt,
                    weight = dat.505$wt) %>%
     bind_cols(X_exposure, X_markers_varset)
weights <- dat.505$wt
Y <- tibble(Y = dat.505$case)
vaccinees <- dplyr::bind_cols(Y, X) %>%
  filter(trt == 1) %>%
  select(-trt)
Y_vaccine <- vaccinees$Y
weights_vaccine <- vaccinees$weight
X_vaccine <- vaccinees %>%
  select(-Y, -weight, -ptid)
# read in the full phase 1 dataset and weights,
# and reorder so that rows match rows of X_vaccine with the remaining rows after
# note that in this case, Z_plus_weights has 2494 rows
# (the number of participants with complete data on age, BMI, bhvrisk)
# and that the number of vaccinees is 1250
Z_plus_weights <- readRDS(file = paste0(data_dir,
                                        "z_and_weights_for_505_analysis.rds"))
# pull out the participants in the cc cohort who also received the vaccine;
# this matches the rows in vaccinees
all_cc_vaccine <- Z_plus_weights %>%
  filter(ptid %in% vaccinees$ptid, trt == 1)
# pull out the participants who are NOT in the cc cohort and received the vaccine
all_non_cc_vaccine <- Z_plus_weights %>%
  filter(!(ptid %in% vaccinees$ptid), trt == 1)
# put them back together
phase_1_data_vaccine <- dplyr::bind_rows(all_cc_vaccine, all_non_cc_vaccine) %>%
  select(-trt)
Z_vaccine <- phase_1_data_vaccine %>%
  select(-ptid, -weight)
all_ipw_weights_vaccine <- phase_1_data_vaccine %>%
  pull(weight)
C <- (phase_1_data_vaccine$ptid %in% vaccinees$ptid)

V_outer <- 5
V_inner <- length(Y_vaccine) - 1
# ------------------------------------------------------------------------------
# do 10 random starts, average over these
# use assay groups as screens
# ------------------------------------------------------------------------------
# ensure reproducibility
set.seed(4747)
seeds <- round(runif(10, 1000, 10000)) # average over 10 random starts
```

```
fits <- parallel::mclapply(seeds, FUN = run_cv_sl_once, Y = Y_vaccine,
                           X_mat = X_vaccine, family = "binomial",
              Z = Z, C = C, z_lib = methods[!grepl("earth", methods)],
              obsWeights = weights_vaccine,
              sl_lib = SL_library_with_assay_groups,
              scale = "logit",
              method = "method.CC_nloglik",
              cvControl = list(V = V_outer, stratifyCV = TRUE),
              innerCvControl = list(list(V = V_inner)),
              vimp = FALSE,
              mc.cores = num_cores
)
saveRDS(fits, "sl_fits.rds")
```

Code to reproduce the full variable importance analysis:

```
#!/usr/local/bin/Rscript

# run the super learner
# make sure that it is CV.SL, averaged over 10 random starts

# load required libraries and functions
library("methods")
library("SuperLearner")
# library("future")
# library("future.apply")
library("e1071")
library("glmnet")
library("xgboost")
library("earth")
library("dplyr")
# only run this if something has changed
# install.packages("HVTN505_2019-4-25.tar.gz", type = "source", repos = NULL)
library("HVTN505")
library("kyotil")
library("argparse")
library("nloptr")
library("quadprog")
# only run this if something has changed
# devtools::install_github("bdwilliamson/vimp", upgrade = "never")

# set up code directory
if (!is.na(Sys.getenv("RSTUDIO", unset = NA))) { # if running locally
  code_dir <- "code/"
  data_dir <- "data/"
  # load vimp from user library (to make sure it has correct version)
  library("vimp")
} else {
  code_dir <- ""
  data_dir <- ""
  # load vimp from user library (to make sure it has correct version)
  library("vimp", lib.loc = .libPaths()[2])
}
num_cores <- parallel::detectCores()
```

```r
print(num_cores)
source(paste0(code_dir, "sl_screens.R")) # set up the screen/algorithm combinations
source(paste0(code_dir, "utils.R")) # get CV-AUC for all algs

parser <- ArgumentParser()
parser$add_argument("--weight-type", default = "aipw",
                    help = "type of weighting to use")
args <- parser$parse_args()

# ------------------------------------------------------------------------------
# pre-process the data
# ------------------------------------------------------------------------------
# read in the full dataset
data("dat.505", package = "HVTN505")
# read in the super learner variables
data("var.super", package = "HVTN505") # even if there is a warning message, it still exists
# note that "var.super" contains individual vars for vaccine-matched antigens,
# and for vaccine-mismatched antigens, has either individual var (if only one)
# or PC1 and/or MDW (only PC1 if cor(PC1, MDW) > 0.9)

# scale vaccine recipients to have mean 0, sd 1 for all vars
for (a in var.super$varname) {
  dat.505[[a]] <- as.vector(
    scale(dat.505[[a]],
          center = mean(dat.505[[a]][dat.505$trt == 1]),
          scale = sd(dat.505[[a]][dat.505$trt == 1]))
    )
  dat.505[[a%.%"_bin"]] <- as.vector(
    scale(dat.505[[a%.%"_bin"]],
          center = mean(dat.505[[a%.%"_bin"]][dat.505$trt == 1]),
          scale = sd(dat.505[[a%.%"_bin"]][dat.505$trt == 1]))
    )
}
for (a in c("age", "BMI", "bhvrisk")) {
  dat.505[[a]] <- as.vector(
    scale(dat.505[[a]],
          center = mean(dat.505[[a]][dat.505$trt == 1]),
          scale = sd(dat.505[[a]][dat.505$trt == 1]))
    )
}

# set up X, Y for super learning
X_markers <- dat.505 %>%
  select(var.super$varname, paste0(var.super$varname, "_bin"))

# only include the following variable sets:
assays <- unique(var.super$assay)
antigens <- unique(var.super$antigen)
# 1. None (baseline variables only)
var_set_none <- rep(FALSE, ncol(X_markers))
# 2. IgG + IgA (all antigens)
var_set_igg_iga <- get_nms_group_all_antigens(
  X_markers, assays = c("IgG", "IgA"), assays_to_exclude = "IgG3"
```

```
  )
# 3. IgG3
var_set_igg3 <- get_nms_group_all_antigens(
  X_markers, assays = "IgG3"
  )
# 4. T cells (all antigens)
var_set_tcells <- get_nms_group_all_antigens(
  X_markers, assays = c("CD4", "CD8")
  )
# 5. Fx Ab (all antigens)
var_set_fxab <- get_nms_group_all_antigens(
  X_markers, assays = c("phago", "R2a", "R3a")
  )
# 6. 1+2+3
var_set_igg_iga_igg3 <- get_nms_group_all_antigens(
  X_markers, assays = c("IgG", "IgA", "IgG3")
  )
# 7. 1+2+4
var_set_igg_iga_tcells <- get_nms_group_all_antigens(
  X_markers, assays = c("IgG", "IgA", "CD4", "CD8"), assays_to_exclude = "IgG3"
  )
# 8. 1+2+3+4
var_set_igg_iga_igg3_tcells <- get_nms_group_all_antigens(
  X_markers, assays = c("IgG", "IgA", "IgG3", "CD4", "CD8")
  )
# 9. 1+2+3+5
var_set_igg_iga_igg3_fxab <- get_nms_group_all_antigens(
  X_markers, assays = c("IgG", "IgA", "IgG3", "phago", "R2a", "R3a")
  )
# 10. 1+4+5
var_set_tcells_fxab <- get_nms_group_all_antigens(
  X_markers, assays = c("CD4", "CD8", "phago", "R2a", "R3a")
  )
# 11. All
var_set_all <- rep(TRUE, ncol(X_markers))
# 12--14: extra runs to get variable importance
var_set_igg3_fxab <- get_nms_group_all_antigens(
  X_markers, assays = c("IgG3", "phago", "R2a", "R3a")
  )
var_set_igg_iga_tcells_fxab <- get_nms_group_all_antigens(
  X_markers,
  assays = c("IgG", "IgA", "CD4", "CD8", "phago", "R2a", "R3a"),
  assays_to_exclude = "IgG3"
  )
var_set_igg3_tcells_fxab <- get_nms_group_all_antigens(
  X_markers, assays = c("IgG3", "CD4", "CD8", "phago", "R2a", "R3a")
  )

var_set_names <- c("1_baseline_exposure", "2_igg_iga",
                   "3_igg3","4_tcells", "5_fxab",
                   "6_igg_iga_igg3", "7_igg_iga_tcells",
                   "8_igg_iga_igg3_tcells",
                   "9_igg_iga_igg3_fxab", "10_tcells_fxab",
```

```
                  "11_all",
                  "12_igg3_fxab", "13_igg_iga_tcells_fxab",
                  "14_igg3_tcells_fxab")

# set up a matrix of all
var_set_matrix <- rbind(var_set_none, var_set_igg_iga, var_set_igg3,
                        var_set_tcells, var_set_fxab,
                        var_set_igg_iga_igg3, var_set_igg_iga_tcells,
                        var_set_igg_iga_igg3_tcells,
                        var_set_igg_iga_igg3_fxab, var_set_tcells_fxab,
                        var_set_all,
                        var_set_igg3_fxab, var_set_igg_iga_tcells_fxab,
                        var_set_igg3_tcells_fxab)
job_id <- as.numeric(Sys.getenv("SLURM_ARRAY_TASK_ID"))
this_var_set <- var_set_matrix[job_id, ]
cat("\n Running", var_set_names[job_id], "\n")
cat("\n using", toupper(args$weight_type), "\n")

# get the current phase 2 dataset based on the markers of interest
X_markers_varset <- X_markers %>%
  select(names(X_markers)[this_var_set])
X_exposure <- dat.505 %>%
  as_tibble() %>%
  select(age, BMI, bhvrisk)
X <- tibble::tibble(ptid = dat.505$ptid, trt = dat.505$trt,
                    weight = dat.505$wt) %>%
     bind_cols(X_exposure, X_markers_varset)
Y <- tibble(Y = dat.505$case)
vaccinees <- dplyr::bind_cols(Y, X) %>%
  filter(trt == 1) %>%
  select(-trt)
Y_vaccine <- vaccinees$Y
weights_vaccine <- vaccinees$weight
X_vaccine <- vaccinees %>%
  select(-Y, -weight, -ptid)
# read in the full phase 1 dataset and weights,
# and reorder so that rows match rows of X_vaccine with the remaining rows after
# note that in this case, Z_plus_weights has 2494 rows
# (the number of participants with complete data on age, BMI, bhvrisk)
# and that the number of vaccinees is 1250
Z_plus_weights <- readRDS(file = paste0(data_dir,
                                        "z_and_weights_for_505_analysis.rds"))
# pull out the participants in the cc cohort who also received the vaccine;
# this matches the rows in vaccinees
all_cc_vaccine <- Z_plus_weights %>%
  filter(ptid %in% vaccinees$ptid, trt == 1)
# pull out the participants who are NOT in the cc cohort and received the vaccine
all_non_cc_vaccine <- Z_plus_weights %>%
  filter(!(ptid %in% vaccinees$ptid), trt == 1)
# put them back together
phase_1_data_vaccine <- dplyr::bind_rows(all_cc_vaccine, all_non_cc_vaccine) %>%
  select(-trt)
Z_vaccine <- phase_1_data_vaccine %>%
```

```
  select(-ptid, -weight)
all_ipw_weights_vaccine <- phase_1_data_vaccine %>%
  pull(weight)
C <- (phase_1_data_vaccine$ptid %in% vaccinees$ptid)

V_outer <- 5
V_inner <- length(Y_vaccine) - 1

# get the SL library
# if var_set_none, then don't need screens; otherwise do
if (job_id == 1) {
  sl_lib <- methods[!grepl("earth", methods)]
} else {
  sl_lib <- SL_library[!grepl("earth", SL_library)]
}
# ------------------------------------------------------------------------------
# run super learner, with leave-one-out cross-validation and all screens
# do 10 random starts, average over these
# ------------------------------------------------------------------------------
# ensure reproducibility
set.seed(4747)
seeds <- round(runif(10, 1000, 10000)) # average over 10 random starts
fits <- parallel::mclapply(seeds, FUN = run_cv_sl_once, Y = Y_vaccine,
                           X_mat = X_vaccine, family = "binomial",
                           Z = Z_vaccine, C = C,
                           z_lib = methods[!grepl("earth", methods)],
                           obsWeights = weights_vaccine,
                           all_weights = all_ipw_weights_vaccine,
                           scale = "logit",
                           sl_lib = sl_lib, # this comes from sl_screens.R
                           method = "method.CC_nloglik",
                           cvControl = list(V = V_outer, stratifyCV = TRUE),
                           innerCvControl = list(list(V = V_inner)),
                           vimp = FALSE, weight_type = args$weight_type,
                           mc.cores = num_cores)
saveRDS(fits, paste0("sl_fits_varset_", var_set_names[job_id],
                     "_", args$weight_type, ".rds"))
warnings()
```

References

Doksum, K, S Tang, and KW Tsui. 2008. https://doi.org/10.1198/016214508000000878.

Fong, Youyi, X Shen, VC Ashley, A Deal, and et al. 2018. https://doi.org/10.1093/infdis/jiy008.

Hammer, SM, ME Sobieszczyk, HE Janes, ST Karuna, MJ Mulligan, and et al. 2013. https://doi.org/10.1056/NEJMoa1310566.

Janes, HE, KW Cohen, N Frahm, SC De Rosa, G Sanchez, and et al. 2017. https://doi.org/10.1093/infdis/jix086.

Neidich, SD, Y Fong, SS Li, DE Geraghty, BD Williamson, and et al. 2019. https://doi.org/10.1172/JCI126391.

van der Laan, Mark J, Eric C Polley, and Alan E Hubbard. 2007. https://doi.org/10.2202/1544-6115.1309.

Williamson, Brian D, Peter B Gilbert, Noah Simon, and Marco Carone. 2020. "A Unified Approach for Inference on Algorithm-Agnostic Variable Importance." https://arxiv.org/abs/2004.03683.