Targeted Maximum Likelihood Estimation with Missing Data

2024-09-25

Contents

1 Introduction

The purpose of this vignette is to demonstrate how to use targeted maximum likelihood estimation (also targeted minimum loss-based estimation, or TMLE (van der Laan and Rubin 2006; van der Laan and Rose 2011)) to estimate and make inference (e.g., compute 95% confidence intervals) on binary treatment effects in an observational study setting when data on key confounders are subject to missingness. We consider a setting where some study participants have missing values of a key confounding variable. We consider four targets of inference: the marginal risk difference (mRD), marginal relative risk (mRR), marginal odds ratio (mOR), and conditional odds ratio (cOR). Below, we describe these estimands in more detail. Finally, we conclude with a demonstration using the `tmle` `R` package version 2.0.1 (Gruber and van der Laan 2012) to compute point estimates and 95% confidence intervals for each estimand. In this demonstration, we will compare two versions of the TMLE: a simple version that only uses machine learning to address missing data, that we call "TMLE-M" (and use for illustration); and a more complex version that uses machine learning in all aspects of estimation, that we call "TMLE-MTO" (which we recommend in most situations). This vignette was created using `R` version 4.4.0; for more information about the `R` session, see Section 6. As of 2024-09-05, there is a new `R` package `twoStageDesignTMLE` (Gruber and van der Laan 2024) that can also perform the functionality that we demonstrate here, without the explicit need for utility functions that we include here. We are currently testing this package to ensure that it gives identical results, and will update this vignette once the testing is complete.

2 Generating data

The code chunk below generates a simple dataset that we will use to illustrate TMLE. Variables include a binary exposure (or treatment) X, binary outcome Y, continuous confounders Z that are always observed, and continuous confounders W that are subject to missingness. The function below produces a single dataset called "data" with N observations of $(Y, X, Z, W, W_{\text{obs}}, \Delta)$, where W is the latent confounder variable with no missingness, Δ is an indicator of whether W is observed ($\Delta = 1$ implies observed), and W_{obs} is the observed version of this variable that is subject to missingness determined by a simple logistic model dependent on observed (X, Z) (is `NA` if missing). This is a missing-at-random (MAR) mechanism for the missing data, i.e., one that depends only on fully-observed covariates.

```r
# define the function to generate data
generating_data <- function(N) {

  # Producing Z, W, and latent X
  # (cutpoint used later to determine yes/no for binary indicator)
  lowcor <- .1
  midcor <- .4
  highcor <- .7
  gencor <- .2

  corMat <- matrix(gencor, nrow = 3, ncol = 3)
  corMat[2, 1] <- corMat[1, 2] <- lowcor  ###cor (X,Z)
  corMat[3, 1] <- corMat[1, 3] <- midcor  ###cor (X,W)

  varMat <- corMat
  diag(varMat) <- 1

  XZW <- mvtnorm::rmvnorm(N, rep(0, 3), varMat)

  latentX <- XZW[, 1]
  Z <- XZW[, 2]
  W <- XZW[, 3]

  pX <- 0.4
  X <- ifelse(latentX < quantile(latentX, p = pX), 1, 0)

  # Now, generating Y using generated X, Z, and W with logistic model.
  beta0 <- -2.4
  betaX <- log(1.5)
  betaZ <- -log(1.3)
  betaW <- -log(1.75)

  Xbeta <- beta0 + betaX * X + betaZ * Z + betaW * W

  Y <- ifelse(runif(length(X)) < stats::plogis(Xbeta), 1, 0)

  # Finally, supposing W is not observed for some individuals.
  alpha0 <- -.67
  alphaX <- log(1.5)
  alphaZ <- log(2.5)
  XbetaM <- alpha0 + alphaX * X + alphaZ * Z
  miss <- ifelse(runif(length(X)) < stats::plogis(XbetaM), 1, 0)
  Delta <- 1 - miss
```

```r
  W_obs <- ifelse(miss == 0, W, NA)

  data <- data.frame(Y, X, Z, W, W_obs, Delta)

  return(data)
}
# generate a sample of size 10,000
set.seed(20240227)
data <- generating_data(N = 10000)
```

3 Estimands

An important first step in any analysis is to define the target of estimation (i.e., the estimand). There are several possible estimands of interest in a setting with a binary exposure and binary outcome, as we have in the simple setting described above.

First, we use the potential (or counterfactual) outcomes framework (Rubin 1974) to define the average treatment effect, which is a function of several causal quantities. We use $Y(x)$ to refer to the potential outcome when exposure is equal to x; under this notation, $Y(1)$ refers to the potential outcome value under the exposure ($x = 1$) and $Y(0)$ is the potential outcome value under non-exposure ($x = 0$). Then we can write the average treatment effect as

$$E\{Y(1) - Y(0)\} = E\{Y(1)\} - E\{Y(0)\},$$

which is interpreted as the difference in the average outcome value under exposure versus non-exposure.

Since we do not observe both potential outcomes for each participant in the study, additional assumptions are necessary to estimate the average treatment effect using our data. These assumptions allow us to *identify* the causal estimand, relating the causal estimand to a statistical estimand (van der Laan and Rose 2011). The *consistency* assumption states that the observed outcome is the potential outcome that would be observed if we set the exposure to its observed level; in other words, that if $X = x$, then $Y(x) = Y$. The *randomization* or *no unmeasured confounders* assumption states that the potential outcome is independent of the exposure, given the measured covariates; in other words, that $Y(x) \perp X \mid (Z, W)$. The *positivity* assumption states that there is a positive probability of exposure level x within all possible strata of the covariates; in other words, $P(X = x \mid Z = z, W = w) > 0$ for all (z, w).

We define the treatment-specific mean outcome values

$$\mu_1 = E(Y \mid X = 1) = P(Y = 1 \mid X = 1) \text{ and}$$
$$\mu_0 = E(Y \mid X = 1) = P(Y = 1 \mid X = 0),$$

where E is shorthand for the expected value under the data-generating distribution and P is shorthand for the probability under this distribution. Under the consistency, randomization, and positivity assumptions, we can identify $E\{Y(1)\}$ with μ_1 and $E\{Y(0)\}$ with μ_0, in other words,

$$E\{Y(1)\} = \mu_1 \text{ and } E\{Y(0)\} = \mu_0.$$

Thus, under the assumptions, μ_1 is the expected outcome value when the exposure is set to 1; μ_0 is the expected outcome value when the exposure is set to 0. Importantly, these treatment-specific mean outcome values are still useful quantities to estimate even if the causal identification assumptions do not hold.

Based on these quantities, we can define the following three marginal estimands:

$$mRD = \mu_1 - \mu_0$$
$$mRR = \frac{\mu_1}{\mu_0}$$
$$mOR = \frac{\mu_1/(1-\mu_1)}{\mu_0/(1-\mu_0)}.$$

The marginal risk difference (mRD) is simply the difference between the treatment-specific means. The marginal relative risk (mRR) is their ratio. The marginal odds ratio (mOR) is a more complex function (the odds) based on the treatment-specific means. These quantities are *marginal* because they are averages over the covariate distributions (here, the distribution of (W, Z)) (Cole and Frangakis 2009). Often, one of the mRD, mRR, or mOR are of interest in causal inference, because (under the key assumptions listed above) they measure a causal effect if the exposure were set to either of its two levels for everyone in the population.

A fourth estimand of interest is the odds ratio conditional on covariates (we will often use the shorthand *conditional odds ratio* or cOR). This is the natural parameter from a logistic regression model, and is the target of interest in many applied settings. Suppose that we fit the following logistic regression model to the ideal data (with no missingness): $\text{logit}\, P(Y = 1 \mid X = x, Z = z, W = w) = \beta_0 + \beta_1 x + \beta_2 z + \beta_3 w$. In this model, $\exp(\beta_1)$ is the cOR comparing $X = 1$ to $X = 0$; we will consider this to be the cOR of interest. It is *conditional* because the interpretation of β_1 is "adjusted" for Z and W.

We do not need to believe that the logistic regression model holds for the conditional odds ratio to be defined. Suppose instead that we specify the above model as a *working* model. Then the conditional odds ratio is the projection of the true underlying data-generating probability that the outcome equals 1, given the variables, onto this working logistic regression model. This idea is important for TMLE, as we will see below.

4 Targeted maximum likelihood estimation

4.1 Overall framework and algorithm

TMLE was developed to provide a doubly-robust approach to inference on general target parameters. The philosophy underlying TMLE is that in any analysis, the first step is to define a target parameter (e.g., the mRD or mRR); the second step is to estimate and make inference on this parameter, using flexible tools (e.g., machine learning) if necessary. Traditional applications of TMLE are in cases with no missing data, but with an exposure and outcome of interest and possibly some confounding factors. In this context, "doubly-robust" refers to the fact that the TML estimator of the treatment effect is consistent if either the outcome regression *or* the propensity score model for treatment is estimated consistently (i.e., the estimator of the true outcome regression or propensity score approaches the true value as the sample size increases); and the TML estimator is efficient (smallest variance in its class of estimators) if both the outcome regression *and* the propensity score are estimated consistently. The abbreviation *TMLE* can refer either to *targeted maximum likelihood estimation* (van der Laan and Rubin 2006) or *targeted minimum loss-based estimation* (van der Laan and Rose 2011). The latter term is more general than the former, which specifies a log-likelihood loss function for learning.

While TMLE can be used to estimate and make inference on a wide array of target parameters, the details of the algorithm depend on the target parameter of interest. This allows the algorithm to efficiently estimate the target parameter. Suppose that our target parameter is the marginal risk difference, $\mu_1 - \mu_0$, and that for now there is no missing data. First, we must obtain initial estimates of the *outcome regression* $Q(x, z, w) = E(Y \mid X, Z, W)$ and the *propensity score* $g(x \mid z, w) = P(X = x \mid Z, W)$. We can use any procedure for these two nuisance functions, including machine learning. As mentioned above, flexibility here is desirable because if the initial estimator Q_n of Q or g_n of g is consistent, then the TMLE will be consistent. Second, we must fluctuate the initial estimate of Q in such a way that the TMLE solves the *efficient influence function* (EIF) estimating equation for the target parameter. The EIF is a key object in semiparametric statistics: it plays a role in establishing efficiency bounds for regular and asymptotically linear estimators of the target parameter (Bickel et al. 1993). In particular, the TMLE is a regular and asymptotically linear estimator with influence function equal to the EIF.

Based on initial outcome regression estimator Q_n of Q and propensity score estimator g_n of g, the fluctuation for the mRD is determined by the EIFs for the treatment-specific means (since the EIF for the mRD is the

difference between these two EIFs). The EIFs for the treatment-specific means are

$$\phi_1(y,x,z,w) = \frac{I(x=1)}{g(1 \mid z,w)}\{y - Q(x,z,w)\} + Q(1,z,w) - \mu_1$$
$$\phi_0(y,x,z,w) = \frac{I(x=0)}{g(0 \mid z,w)}\{y - Q(x,z,w)\} + Q(0,z,w) - \mu_0,$$

so the fluctuations are:

$$H_0(x,z,w) = \frac{I(x=0)}{g(0 \mid z,w)}$$
$$H_1(x,z,w) = \frac{I(x=1)}{g(1 \mid z,w)}.$$

To ensure that the TMLE solves the EIF estimating equation, we must have that

$$0 = \frac{1}{n}\sum_{i=1}^{n} \begin{bmatrix} H_1(X_i,Z_i,W_i) \\ -H_0(X_i,Z_i,W_i) \end{bmatrix} \{Y_i - Q_n(X_i,Z_i,W_i)\}.$$

We can solve this estimating equation for the mRD target parameter using off-the-shelf logistic regression: the outcome is Y, covariates are $[H_1(X,Z,W), H_0(X,Z,W)]$, and we use logit Q_n as an offset (for this example, this logistic regression step – solved by maximum likelihood – is the "MLE" in "TMLE"). This yields regression coefficient estimates $(\hat{\epsilon}_1, \hat{\epsilon}_0)$. Then, we set

$$\text{logit}\{Q_n^*(x,z,w)\} = \text{logit}\{Q_n(x,z,w)\} + \hat{\epsilon}_1 H_1(x,z,w) + \hat{\epsilon}_0 H_0(x,z,w),$$

and the final TML estimator is $\frac{1}{n}\sum_{i=1}^{n} Q_n^*(X_i,Z_i,W_i)$.

4.2 Incorporating missing data

It is straightforward to incorporate missing data in covariates. The final key quantity we need to estimate is the *missing-data mechanism* $\pi = P(\Delta = 1 \mid X,Z,Y)$. Once we have an estimator π_n of π, we can use inverse weights $1/\pi_n$ within the TMLE procedure to obtain valid estimates of and inference on our target parameter. Estimation and inference proceed among the participants with full covariate information; all steps discussed above are weighted using the inverse probability weights. The final TMLE estimator in the missing-data setting is $\frac{1}{n}\sum_{i=1}^{n} \frac{\Delta_i}{\pi_n(X_i,Z_i,Y_i)} Q_n^*(X_i,Z_i,W_i)$. This is the two-phase TMLE introduced by Rose and van der Laan (2011).

4.3 Estimating marginal parameters

It is relatively straightforward to obtain estimates of marginal parameters (e.g., the mRD, mRR, or mOR) using the TMLE machinery, implemented in the `tmle` `R` package (Gruber and van der Laan 2012). The key is that procedures must be specified for estimating the outcome regression, the propensity score, and the missing-data mechanism. One can use a single procedure (e.g., a logistic regression model) for any of these three nuisance functions; however, it may be advantageous to consider instead an ensemble of several *candidate learners* (prediction algorithms, e.g., logistic regression or random forests), ranging from simple to complex, in an effort to better capture the underlying function. The Super Learner (van der Laan, Polley, and Hubbard 2007) is one such ensemble that has several desirable asymptotic guarantees, and uses cross-validation to select the ensemble of candidate learners that optimizes a chosen loss function (defaults to negative log likelihood loss for binary variables). In addition to specifying a library of candidate learners, then, the number of cross-validation folds must be specified. In general, we advocate for using a Super Learner for estimating the three nuisance functions required for TMLE in this setting.

4.4 Estimating conditional parameters

Finally, we describe our approach for estimating the conditional odds ratio using TMLE. Suppose that our working logistic regression model is $\text{logit}\, E(Y \mid X = x, Z = z, W = w) = \beta_0 + \beta_1 x + \beta_2 z + \beta_3 w$, as defined above. The cOR $\exp(\beta_1)$ can be expressed as the parameter in a working model known as a *marginal structural model* (Robins, Hernán, and Brumback 2000; Gruber and van der Laan 2012). If the marginal structural model is misspecified (i.e., the true conditional mean function is not linear in the variables on the logit scale), then the cOR parameter $\exp(\beta_1)$ can be viewed as the projection of the true causal effect parameter onto the working model (Gruber and van der Laan 2012). In this vignette, we specify as marginal structural model (MSM) the working logistic regression model described above.

5 Using the `tmle` package

5.1 Defining key data objects and possible candidate learners

First, we set up some data objects for the `tmle` package. The main object is a `data.frame` of the confounder(s) that are prone to missingness (here, just W) *with no missing data*. Importantly, in the syntax of the `tmle` package, `"A"` refers to the exposure (our X) and `"W"` refers to the complete set of confounders (our (W, Z)).

```
# get the values of W for participants who have no missing data
W <- data[, "W", drop = FALSE]
# the syntax below guarantees that obs_W is still a data.frame
obs_W <- W[complete.cases(data), , drop = FALSE]
```

Next, we set up some candidate learners that could be used in Super Learners within `tmle`. We could be more exhaustive than this with our candidate learners, but these represent several well-performing tree-based strategies.

```
# load required packages
library("SuperLearner")
```

```
## Loading required package: nnls
## Loading required package: gam
## Loading required package: splines
## Loading required package: foreach
## Loaded gam 1.22-3
## Super Learner
## Version: 2.0-29
## Package created on 2024-02-06
```

```
library("glmnet")
```

```
## Loading required package: Matrix
## Loaded glmnet 4.1-8
```

```
library("xgboost")
```

```
##
## Attaching package: 'xgboost'
## The following object is masked from 'package:dplyr':
##
##     slice
```

```r
library("ranger")
library("tmle")
```

```
## Welcome to the tmle package, version 2.0.1
##
## Use tmleNews() to see details on changes and bug fixes
```

```r
# gradient boosted trees, from package xgboost
# set up a list of tuning parameters with varying maximum tree depth and shrinkage rate
xgb_tune_params <- list(max_depth = c(1, 3), shrinkage = c(1e-2, 1e-1), ntrees = 500)

# random forests, from package ranger
# set up a list of tuning parameters with varying minimum node size
n <- nrow(data)
min_node_sizes <- round(seq.int(n / 100, n / 10, length.out = 4))
rf_tune_params <- list(num.trees = 500, min.node.size = min_node_sizes,
                       verbose = FALSE)
# create all possible boosted tree and RF learners based on the tuning parameters
# specified above
xgb_learners <- create.Learner("SL.xgboost", tune = xgb_tune_params,
                               detailed_names = TRUE, name_prefix = "xgb")
rf_learners <- create.Learner("SL.ranger", tune = rf_tune_params,
                              detailed_names = TRUE, name_prefix = "rf")
```

Prior to running the code in the next sections, you will need several utility functions. We have placed these in the Appendix at the end of this document. If you are accessing this vignette in R Markdown, you can load in the utility functions now. If you are accessing this vignette as a PDF, you will need to run the code in the Appendix before continuing. (this section is currently under development, because the `twoStageDesignTMLE` package removes the need to load these utility functions)

5.2 Obtaining estimates using TMLE-M

Recall that we have observational data and are interested in estimating a treatment effect for a binary exposure variable, and are missing data on a key confounding variable for some study participants. Before running the TMLE, we must decide which learners to use for the outcome regression, propensity score, and missing-data mechanism. The full TMLE procedure, which we recommend in most cases, involves specifying a flexible library of candidate learners (e.g., the tree-based learners defined above) for all three.

In the following two code chunks, we obtain TMLE estimates using a simplified version of a full TMLE procedure. We call this procedure "TMLE-M": we only use a Super Learner to estimate the missing-data model ("M"); we use logistic regression for the outcome regression and propensity score. In general, we do not advocate for this approach; however, it allows for a direct comparison with other missing-data methods (e.g., generalized raking, MICE) that similarly rely on a simple outcome regression model.

5.2.1 Estimating the mRD, mRR, and mOR using TMLE-M

In this code chunk, we set up the library of candidate learners for the TMLE-M and obtain estimates of the mRD, mRR, and mOR.

```r
# set up Super Learner library for outcome regression model;
# in this case, only a glm, so it's advantageous to pass in a formula
outcome_regression_lib <- "SL.glm"
# tmle package uses "A" to refer to the exposure variable
outcome_regression_formula <- "Y ~ A + Z + W"
# set up Super Learner library for treatment-assignment model (propensity score);
# in this case, only a glm, so it's advantageous to pass in a formula
```

```r
propensity_lib <- "SL.glm"
# tmle package uses "A" to refer to the exposure variable
propensity_formula <- "A ~ W + Z"
# set up Super Learner library for missing-data model;
# ranges from simple (glm) to more complex (RFs and boosted trees)
miss_lib <- c("SL.glm", xgb_learners$names, rf_learners$names)

# obtain marginal estimates using TMLE
set.seed(20240227)
marginal_tmlem_ests <- subcalTMLE(
  Y = data$Y, A = data$X,
  # here, W refers to the completely-observed confounders, our "Z"
  W = data[, "Z", drop = FALSE], # need W to be a data.frame
  # here, W.sub refers to the complete-case sample of missing-prone confounders
  # (our W)
  W.sub = obs_W, Delta.W = data$Delta,
  condSetNames = c("W", "A", "Y"),
  pi.SL.library = miss_lib, V.pi = 10,
  Q.family = "binomial",
  Q.SL.library = outcome_regression_lib, V.Q = 10, Qform = outcome_regression_formula,
  g.SL.library = propensity_lib, V.g = 10, gform = propensity_formula,
  augmentW = FALSE,
  verbose = FALSE
)
marginal_ests <- data.frame(
  "procedure" = "TMLE-M",
  "estimand" = c("mRD", "mOR (log)", "mRR (log)"),
  "est" = c(
    marginal_tmlem_ests$tmle$estimates$ATE$psi,
    log(marginal_tmlem_ests$tmle$estimates$OR$psi),
    log(marginal_tmlem_ests$tmle$estimates$RR$psi)
  ),
  "SE" = c(
    sqrt(as.numeric(marginal_tmlem_ests$tmle$estimates$ATE$var.psi)),
    sqrt(as.numeric(marginal_tmlem_ests$tmle$estimates$OR$var.log.psi)),
    sqrt(as.numeric(marginal_tmlem_ests$tmle$estimates$RR$var.log.psi))
  )
)
```

5.2.2 Estimating the cOR using TMLE-M

In this code chunk, we use the library of candidate learners defined in the previous code chunk, and use TMLE-M to estimate the cOR based on the assumed marginal structural model.

```r
# obtain conditional OR estimate from MSM using TMLE
# again, "A" corresponds to the exposure of interest
msm_formula <- "A + Z + W_obs"
set.seed(20240227)
conditional_tmlem_ests <- subcalTMLEMSM(
  Y = data$Y, A = data$X,
  # here, W refers to the completely-observed confounders, our "Z"
  W = data[, "Z", drop = FALSE], # need W to be a data.frame
  # here, W.sub refers to the complete-case sample of missing-prone confounders
  # (our W)
```

```
  W.sub = obs_W,
  # V refers to the variables to condition on in the MSM
  V = data[, c("X", "Z", "W_obs")],
  Delta.W = data$Delta,
  MSM = msm_formula, condSetNames = c("W", "A", "Y"),
  pi.SL.library = miss_lib, V.pi = 10,
  Q.family = "binomial",
  Q.SL.library = outcome_regression_lib, V.Q = 10, Qform = outcome_regression_formula,
  g.SL.library = propensity_lib, V.g = 10, gform = propensity_formula,
  verbose = FALSE
)
conditional_tmlem_summary <- summary(conditional_tmlem_ests$tmleMSM)$estimates[, 1:3]
conditional_est <- data.frame(
  "procedure" = "TMLE-M",
  "estimand" = "cOR (log)",
  "est" = conditional_tmlem_summary[2, 1],
  "SE" = conditional_tmlem_summary[2, 2]
)
```

5.2.3 Estimates of the mRD, mRR, mOR, and cOR using TMLE-M

We provide the point estimates and standard errors in Table 1.

```
all_tmlem_ests <- rbind.data.frame(marginal_ests, conditional_est)
# the function `%>%` comes from the dplyr package
all_tmlem_ests %>%
  select(-procedure) %>%
  rename(Estimand = estimand, `Point Estimate` = est, `Standard Error` = SE) %>%
  knitr::kable(format = "latex",
               caption = paste0("Point estimates and standard errors for the ",
                                "marginal risk difference (mRD), ",
                                "marginal relative risk (mRR), ",
                                "marginal odds ratio (mOR), and ",
                                "conditional odds ratio (cOR), ",
                                "obtained using TMLE-M.")) %>%
  kableExtra::kable_styling(latex_options = "hold_position")
```

Table 1: Point estimates and standard errors for the marginal risk difference (mRD), marginal relative risk (mRR), marginal odds ratio (mOR), and conditional odds ratio (cOR), obtained using TMLE-M.

Estimand	Point Estimate	Standard Error
mRD	0.0493687	0.0092275
mOR (log)	0.4834499	0.0883405
mRR (log)	0.4275104	0.0780497
cOR (log)	0.5020000	0.2360000

5.3 Obtaining estimates using TMLE-MTO

The full TMLE procedure, which we refer to here as "TMLE-MTO", uses a Super Learner for the missing-data mechanism ("M"), the propensity score ("T", for "treatment assignment"), and the outcome regression ("O"). This is the most flexible version of the TMLE procedure, which we recommend in most cases, because it guards against misspecification of any of the three nuisance functions. This can decrease bias in many cases, but may lead to increased variance in cases (such as the one in this vignette) where the true data-generating

functions are simple and sample sizes are modest. The following code chunk is nearly identical to the code chunk above, but with different learner libraries for the outcome regression and propensity score.

5.3.1 Estimating the mRD, mRR, and mOR using TMLE-MTO

In this code chunk, we set up the library of candidate learners for the TMLE-MTO and obtain estimates of the mRD, mRR, and mOR.

```
# set up Super Learner library for outcome regression model
outcome_regression_lib <- c("SL.glm", xgb_learners$names, rf_learners$names)
# set up Super Learner library for treatment-assignment model (propensity score)
propensity_lib <- c("SL.glm", xgb_learners$names, rf_learners$names)
# set up Super Learner library for missing-data model
miss_lib <- c("SL.glm", xgb_learners$names, rf_learners$names)

# obtain marginal estimates using TMLE
set.seed(20240227)
marginal_tmlemto_ests <- subcalTMLE(
  Y = data$Y, A = data$X,
  # here, W refers to the completely-observed confounders, our "Z"
  W = data[, "Z", drop = FALSE],
  # here, W.sub refers to the complete-case sample of missing-prone confounders
  # (our W)
  W.sub = obs_W,
  Delta.W = data$Delta,
  condSetNames = c("W", "A", "Y"),
  pi.SL.library = miss_lib, V.pi = 10,
  Q.family = "binomial",
  Q.SL.library = outcome_regression_lib, V.Q = 10,
  g.SL.library = propensity_lib, V.g = 10,
  verbose = FALSE
)
marginal_ests_tmlemto <- data.frame(
  "procedure" = "TMLE-MTO",
  "estimand" = c("mRD", "mOR (log)", "mRR (log)"),
  "est" = c(
    marginal_tmlemto_ests$tmle$estimates$ATE$psi,
    log(marginal_tmlemto_ests$tmle$estimates$OR$psi),
    log(marginal_tmlemto_ests$tmle$estimates$RR$psi)
  ),
  "SE" = c(
    sqrt(as.numeric(marginal_tmlemto_ests$tmle$estimates$ATE$var.psi)),
    sqrt(as.numeric(marginal_tmlemto_ests$tmle$estimates$OR$var.log.psi)),
    sqrt(as.numeric(marginal_tmlemto_ests$tmle$estimates$RR$var.log.psi))
  )
)
```

5.3.2 Estimating the cOR using TMLE-MTO

In this code chunk, we use the library of candidate learners defined in the previous code chunk, and use TMLE-MTO to estimate the cOR based on the assumed marginal structural model.

```
# obtain conditional OR estimate from MSM using TMLE
# again, "A" corresponds to the exposure of interest
msm_formula <- "A + Z + W_obs"
set.seed(20240227)
```

```r
conditional_tmlemto_ests <- subcalTMLEMSM(
  Y = data$Y, A = data$X,
  # here, W refers to the completely-observed confounders, our "Z"
  W = data[, "Z", drop = FALSE], # need W to be a data.frame
  # here, W.sub refers to the complete-case sample of missing-prone confounders
  # (our W)
  W.sub = obs_W, V = data[, c("X", "Z", "W_obs")], Delta.W = data$Delta,
  MSM = msm_formula, condSetNames = c("W", "A", "Y"),
  pi.SL.library = miss_lib, V.pi = 10,
  Q.family = "binomial",
  Q.SL.library = outcome_regression_lib, V.Q = 10,
  g.SL.library = propensity_lib, V.g = 10,
  verbose = FALSE
)
conditional_tmlemto_summary <- summary(conditional_tmlemto_ests$tmleMSM)$estimates[, 1:3]
conditional_est_tmlemto <- data.frame(
  "procedure" = "TMLE-M",
  "estimand" = "cOR (log)",
  "est" = conditional_tmlemto_summary[2, 1],
  "SE" = conditional_tmlemto_summary[2, 2]
)
```

5.3.3 Estimates of the mRD, mRR, mOR, and cOR using TMLE-MTO

We provide the point estimates and standard errors in Table 2.

```r
all_tmlemto_ests <- rbind.data.frame(marginal_ests_tmlemto, conditional_est_tmlemto)
# the function `%>%` comes from the dplyr package
all_tmlemto_ests %>%
  select(-procedure) %>%
  rename(Estimand = estimand, `Point Estimate` = est, `Standard Error` = SE) %>%
  knitr::kable(format = "latex",
               caption = paste0("Point estimates and standard errors for the ",
                                "marginal risk difference (mRD), ",
                                "marginal relative risk (mRR), ",
                                "marginal odds ratio (mOR), and ",
                                "conditional odds ratio (cOR), ",
                                "obtained using TMLE-MTO.")) %>%
  kableExtra::kable_styling(latex_options = "hold_position")
```

Table 2: Point estimates and standard errors for the marginal risk difference (mRD), marginal relative risk (mRR), marginal odds ratio (mOR), and conditional odds ratio (cOR), obtained using TMLE-MTO.

Estimand	Point Estimate	Standard Error
mRD	0.0493888	0.0088682
mOR (log)	0.4815725	0.0846543
mRR (log)	0.4255743	0.0747549
cOR (log)	0.4920000	0.0890000

5.4 Comparing TMLE-M and TMLE-MTO

In this example, where our true data-generating model is simple, the point estimates for all target parameters from TMLE-M do not differ much from TMLE-MTO; this is unsurprising, because the true data-generating

mechanism is simple, so using the Super Learner to estimate the propensity score and outcome regression does not remove any estimation bias. Subtracting the TMLE-M point estimates and standard errors from their TMLE-MTO counterparts yields the result in Table 3. The standard errors for TMLE-MTO are larger than their TMLE-M counterparts for the mRD, mRR, and mOR; this is unsurprising, because TMLE-MTO allows more flexibility in estimating the propensity score and outcome regression (which is unneccessary in this case, because the true data-generating functions are simple). The standard error for the cOR from TMLE-MTO is much smaller than that for TMLE-M.

```
est_se_diff <- all_tmlemto_ests %>%
  rename(est_mto = est, SE_mto = SE) %>%
  left_join(all_tmlem_ests %>%
              rename(est_m = est, SE_m = SE) , by = "estimand") %>%
  mutate(est_diff = est_mto - est_m,
         se_diff = SE_mto - SE_m) %>%
  select(-ends_with("mto"), -ends_with("m"), -contains("procedure"))
est_se_diff %>%
  rename(Estimand = estimand,
         `Difference between point estimates (TMLE-MTO - TMLE-M)` = est_diff,
         `Difference between SEs (TMLE-MTO - TMLE-M)` = se_diff) %>%
  knitr::kable(format = "latex",
               caption = paste0("Difference between the point estimates ",
                                "and standard errors ",
                                "for the marginal risk difference (mRD), ",
                                "marginal relative risk (mRR), ",
                                "marginal odds ratio (mOR), and ",
                                "conditional odds ratio (cOR), ",
                                "comparing TMLE-MTO to TMLE-M.")) %>%
  kableExtra::kable_styling(latex_options = "hold_position") %>%
  kableExtra::column_spec(column = 2:3, width = "2in")
```

Table 3: Difference between the point estimates and standard errors for the marginal risk difference (mRD), marginal relative risk (mRR), marginal odds ratio (mOR), and conditional odds ratio (cOR), comparing TMLE-MTO to TMLE-M.

Estimand	Difference between point estimates (TMLE-MTO - TMLE-M)	Difference between SEs (TMLE-MTO - TMLE-M)
mRD	0.0000200	-0.0003593
mOR (log)	-0.0018775	-0.0036862
mRR (log)	-0.0019361	-0.0032948
cOR (log)	-0.0100000	-0.1470000

6 Appendix: R session info

```
sessionInfo()
```

```
## R version 4.4.0 (2024-04-24 ucrt)
## Platform: x86_64-w64-mingw32/x64
## Running under: Windows 10 x64 (build 19045)
##
## Matrix products: default
##
##
```

```
## locale:
## [1] LC_COLLATE=English_United States.utf8
## [2] LC_CTYPE=English_United States.utf8
## [3] LC_MONETARY=English_United States.utf8
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.utf8
##
## time zone: America/Los_Angeles
## tzcode source: internal
##
## attached base packages:
## [1] splines   stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] tmle_2.0.1          ranger_0.16.0       xgboost_1.7.7.1
##  [4] glmnet_4.1-8        Matrix_1.7-0        SuperLearner_2.0-29
##  [7] gam_1.22-3          foreach_1.5.2       nnls_1.5
## [10] dplyr_1.1.4
##
## loaded via a namespace (and not attached):
##  [1] utf8_1.2.4        generics_0.1.3    xml2_1.3.6        shape_1.4.6.1
##  [5] stringi_1.8.4     lattice_0.22-6    digest_0.6.35     magrittr_2.0.3
##  [9] evaluate_0.23     grid_4.4.0        bookdown_0.39     iterators_1.0.14
## [13] mvtnorm_1.2-4     fastmap_1.1.1     jsonlite_1.8.8    survival_3.6-4
## [17] tinytex_0.51      fansi_1.0.6       viridisLite_0.4.2 scales_1.3.0
## [21] codetools_0.2-20  cli_3.6.2         rlang_1.1.3       munsell_0.5.1
## [25] withr_3.0.0       yaml_2.3.8        tools_4.4.0       colorspace_2.1-0
## [29] kableExtra_1.4.0  vctrs_0.6.5       R6_2.5.1          lifecycle_1.0.4
## [33] stringr_1.5.1     pkgconfig_2.0.3   pillar_1.9.0      glue_1.7.0
## [37] data.table_1.15.4 Rcpp_1.0.12       systemfonts_1.0.6 xfun_0.43
## [41] tibble_3.2.1      tidyselect_1.2.1  rstudioapi_0.16.0 knitr_1.46
## [45] htmltools_0.5.8.1 svglite_2.1.3     rmarkdown_2.26    compiler_4.4.0
```

7 Appendix: utility functions for TMLE

```
# utility functions for subcalTMLE ----------------------------------------
#-------------------------------------------------------------------------
# Susan Gruber
# sgruber@TLrevolution.com
# August 10, 2023

# functions for implementing a TMLE for subset calibration
# (based  on two-stage TMLE Rose&vanderLaan, 2011)
# that evaluates a point treatment effect or the marginal mean outcome for a
# single arm study using a 2-stage sampling design
#
# For a two-arm study the full data consists of baseline covariates X, and
# potential outcomes Y0, Y1. Observed data consists of
# baseline covariates (W, Delta.W, Delta.W W'), treatment A,
# outcome (Delta, Delta Y), where covariates X can be divided into two sets:
# W measured on everyone, and W' measured on only a subset of individuals.
# Delta.W is an indicator of being in the selected subset.
```

```r
# A is a unary (single-arm study) or binary treatment indicator, Delta indicates
# whether the outcome is measured and DeltaY is the outcome value
# (when Delta = 1), otherwise missing.

# The functions defined below take an initial observed dataset and transform it
# into a suitable representation for applying targeted maximum likelihood
# estimation (TMLE) using the tmle package (>= 2.0.0).
# Marginal parameter estimates (ATE/RD, RR, OR) can be obtained by calling
# the tmle function.
# Conditional parameter estimates are obtained by calling the tmleMSM function.
# Note this requires specifying the desired MSM. Data on observations in the
# subset are passed in to these functions, along with sampling weights that
# account for the possibly biased selection into the sample.
#
# Step 1. Simple imputation of miscellaneous missing values in W + augment W
#       with binary indicators of which values were present/absent
# Step 2. Construct inverse probability of censoring weights to pass into the
#         tmle or tmleMSM functions
#   2a. Model conditional probability of being included in the subset using
#       SuperLearner or logistic regression.
#   2b. Construct IP weights
# Step 3. Call the tmle or tmleMSM function to evaluate the parameter, estimate
#         variance and CI bounds using influence curve-based inference.
#         Optionally, the tmle function can also provide targeted
#         bootstrap-based variance estimates (recommended when there are near
#         positivity violations).
#-------------------------------------------------------------------------

# For NOW - no missing data allowed in W, W.sub
#------------------------------
# Utilities
summary.subcal <- function(object, ...) {
  if (!is.null(object$coef)) {
    picoef <- object$coef
    if (inherits(picoef, "matrix")) {
      piterms <- colnames(picoef)
    } else {
      piterms <- names(picoef)
    }
    pimodel <- paste("Delta.W ~ 1")
    if (length(piterms) > 1) {
      pimodel <- paste("Delta.W ~ ", paste(piterms, collapse = " + "))
    }
  } else {
    pimodel <- piterms <- picoef <- NULL
  }
  return(list(
    pimodel = pimodel, piterms = piterms, picoef = picoef, pitype = object$type,
    pidiscreteSL = object$discreteSL
  ))
}

print.summary.subcalTMLE <- function(x, ...) {
```

```
  if (inherits(x, "summary.subcalTMLE")) {
    cat("\n Estimation of Pi (subset sampling mechanism)\n")
    cat("\t Procedure:", x$subcal$pitype)
    if (!(is.null(x$subcal$pidiscreteSL))) {
      if (x$subcal$pidiscreteSL) {
        cat(", discrete")
      } else {
        cat(", ensemble")
      }
    }
    if (!(is.null(x$subcal$piAUC))) {
      cat("\t Empirical AUC =", round(x$subcal$piAUC, 4), "\n")
    }
    cat("\n")
    if (!(is.na(x$subcal$picoef[1]))) {
      cat("\t Model:\n\t\t", x$subcal$pimodel, "\n")
      cat("\n\t Coefficients: \n")
      terms <- sprintf("%15s", x$subcal$piterms)
      extra <- ifelse(x$subcal$picoef >= 0, "  ", " ")
      for (i in 1:length(x$subcal$picoef)) {
        cat("\t", terms[i], extra[i], x$subcal$picoef[i], "\n")
      }
    }
    cat("\n")
    print(x$tmle)
  }
}

print.subcalTMLE <- function(x, ...) {
  cat("Subset calibration TMLE\n")
  if (inherits(x, "subcalTMLE")) {
    print(x$tmle)
  }
}

summary.subcalTMLE <- function(x, ...) {
  # summary for estimating Pi here
  if (inherits(x, "subcalTMLE")) {
    sum.subcalTMLE <- list()
    sum.subcalTMLE$subcal <- summary.subcal(x$subcal)
    sum.subcalTMLE$tmle <- summary(x$tmle)
    class(sum.subcalTMLE) <- "summary.subcalTMLE"
    return(sum.subcalTMLE)
  }
}

#----------------------------------------
# Set the number of cross-validation
# folds as a function of n.effective
# See Phillips 2023 doi.org/10.1093/ije/dyad023
#----------------------------------------
setV <- function(n.effective) {
  if (n.effective <= 30) {
```

```
    V <- n.effective
  } else if (n.effective <= 500) {
    V <- 20
  } else if (n.effective <= 1000) {
    V <- 10
  } else if (n.effective <= 10000) {
    V <- 5
  } else {
    V <- 2
  }
  return(V)
}


#-----------------
# construct column names
#-----------------
.getColNames <- function(condSetNames, Wnames, Vnames = NULL) {
  orig.colnames <- NULL
  for (i in 1:length(condSetNames)) {
    # if (condSetNames[i] %in% c("A", "Y")) {
    if (condSetNames[i] == "A") {
      orig.colnames <- c(orig.colnames, condSetNames[i])
    } else if (condSetNames[i] == "Y") {
      orig.colnames <- c(orig.colnames, "real_Y")
    } else if (condSetNames[i] == "W") {
      orig.colnames <- c(orig.colnames, Wnames)
    } else if (condSetNames[i] == "V") {
      orig.colnames <- c(orig.colnames, Vnames)
    }
  }
  return(orig.colnames)
}

#-------------------------
# Function subcalTMLE
# purpose: subset calibration TMLE
# Arguments:
#   Y : outcome of interest (missingness allowed)
#   A : binary treatment indicator
#   W : matrix or data.frame of covariates measured on entire population
#   Delta.W : Indicator of inclusion in subset with additional information
#   W.sub : matrix or data.frame of covariates measured in subset population
#             (in same order as the corresponding rows in W. DO NOT include
#              rows for subjects not included in the subset)
#   Z : Only required when the goal is estimating a controlled direct effect
#       mediated by a binary variable,Z,  between treatment and outcome
#   Delta : binary indicator that outcome Y is observed
#   pi : optional vector of sampling probabilities
#   piform : optional parametric regression model for estimating pi
#   pi.SL.library : optional SL library specification for estimating pi
#     (ignored when piform or pi is provided)
#   V.pi : optional number of cross-validation folds for super learning
```

```
#     (ignored when piform or pi is provided)
#   pi.discreteSL: flag to indicate whether to use ensemble or discrete super
#     learning (ignored when piform or pi is provided)
#   condSet : variables to condition on when estimating pi. Covariates in W are
#     always included. Can optionally also condition on A and/or Y.
#   id : optional indicator of independent units of observation
#   Q.family : "gaussian" or "binomial", used to evaluate the TMLE
#   verbose : set to TRUE to print out informative messages
#   ... other arugments passed to the tmle function (see tmle help files)
# Return value
#   object of class "subcalTMLE" (a list):
#    subcal - a list of details on estimating the conditional sampling
#             probabilities
#    pi - predicted probabilities
#    coef - coefficients in the parametric model or SL ensemble weights
#    type - "user-supplied regression formula" or "Super Learner"
#    "discreteSL" - FALSE when ensemble SL used, TRUE when best single algorithm
#                   selected
#    tmle - results returned from the call to the tmle function with
#             observation weights equal to 1/pi. (class "tmle", see tmle package
#           documentation)
#--------------------------
subcalTMLE <- function(Y, A, W, Delta.W, W.sub, Z = NULL,
                       Delta = rep(1, length(Y)), pi = NULL, piform = NULL,
                       pi.SL.library = c("SL.glm", "SL.gam", "SL.glmnet",
                                         "tmle.SL.dbarts.k.5"), V.pi = 10,
                       pi.discreteSL = TRUE, condSetNames = c("A", "W"),
                       id = NULL, Q.family = "gaussian", verbose = FALSE, ...) {
  if (!requireNamespace("tmle", versionCheck = ">=2.0", quietly = FALSE)) {
    stop("Loading required tmle package (>=2.0) failed", call. = FALSE)
  }

  if (is.null(id)) {
    id <- 1:length(Y)
  }

  if (is.vector(W.sub)) {
    W.sub <- as.matrix(W.sub)
    colnames(W.sub) <- "W.sub"
  }

  # Evaluate conditional sampling probabilities
  if (is.null(pi)) {
    validCondSetNames <- c("A", "W", "Y")
    if (!(all(condSetNames %in% validCondSetNames))) {
      stop("condSetNames must be any combination of 'A', 'W', 'Y'")
    }
    if (any(condSetNames == "Y")) {
      if (any(is.na(Y))) {
        stop(paste0("Cannot condition on the outcome to evaluate sampling ",
                    "probabilities when some outcome values are missing"))
      }
    }
```

```
    d.pi <- data.frame(Delta.W = Delta.W, mget(condSetNames))
    colnames(d.pi)[-1] <- .getColNames(condSetNames, colnames(W))
    res.subcal <- tmle:::estimateG(
      d = d.pi, g1W = NULL, gform = piform, SL.library = pi.SL.library,
      id = id, V = V.pi, message = "sampling weights", outcome = "Delta.W",
      discreteSL = pi.discreteSL,
      obsWeights = rep(1, nrow(W)), verbose = verbose
    )
    names(res.subcal)[1] <- "pi"
  } else {
    res.subcal <- list()
    res.subcal$pi <- pi
    res.subcal$type <- "User supplied values"
    res.subcal$coef <- NA
    res.subcal$discreteSL <- NULL
  }

  obsWeights <- Delta.W / res.subcal$pi
  result <- list()
  result$tmle <- try(tmle(Y[Delta.W == 1],
                          A = A[Delta.W == 1],
                          W = cbind(W[Delta.W == 1, , drop = FALSE], W.sub),
                          Z = Z[Delta.W == 1],
                          Delta = Delta[Delta.W == 1],
                          obsWeights = obsWeights[Delta.W == 1],
                          id = id[Delta.W == 1],
                          family = Q.family, verbose = verbose, ...
  ))

  if (inherits(result$tmle, "try-error")) {
    cat(paste0("Error calling tmle. ",
               "Estimated sampling probabilites are being returned"))
    result$tmle <- NULL
  }
  result$subcal <- res.subcal
  class(result) <- "subcalTMLE" # for tmleMSM can use same class
  return(result)
}


#-------------------------
# Function subcalTMLEMSM
# purpose: subset calibration TMLE for conditional effects
# Arguments:
#   Y : outcome of interest (missingness allowed)
#   A : binary treatment indicator
#   W : matrix or data.frame of covariates measured on entire population
#   Delta.W : Indicator of inclusion in subset with additional information
#   W.sub : matrix or data.frame of covariates measured in subset population
#             (in same order as the corresponding rows in W. DO NOT include
#              rows for subjects not included in the subset)
#   Z : Only required when the goal is estimating a controlled direct effect
#       mediated by a binary variable,Z,  between treatment and outcome
```

```
#   Delta : binary indicator that outcome Y is observed
#   pi : optional vector of sampling probabilities
#   piform : optional parametric regression model for estimating pi
#   pi.SL.library : optional SL library specification for estimating pi
#     (ignored when piform or pi is provided)
#   V.pi : optional number of cross-validation folds for super learning
#     (ignored when piform or pi is provided)
#   pi.discreteSL: flag to indicate whether to use ensemble or discrete super
#     learning (ignored when piform or pi is provided)
#   condSet : variables to condition on when estimating pi.
#      Covariates in W are always included. Can optionally also condition on
#      A and/or Y.
#   id : optional indicator of independent units of observation
#   Q.family : "gaussian" or "binomial", used to evaluate the TMLE
#   verbose : set to TRUE to print out informative messages
#   ... other arugments passed to the tmleMSM function
#       (see tmleMSM documentation in the tmle package)
# Return value
#   object of class "subcalTMLE" (a list):
#    subcal - a list of details on estimating the conditional sampling
#             probabilities
#    pi - predicted probabilities
#    coef - coefficients in the parametric model or SL ensemble weights
#    type - "user-supplied regression formula" or "Super Learner"
#    "discreteSL" - FALSE when ensemble SL used, TRUE when best single algorithm
#                   selected
#    tmle - results returned from the call to the tmleMSM function with
#             observation weights equal to 1/pi. (class "tmleMSM", see tmle package
#           documentation)
#-------------------------

subcalTMLEMSM <- function(Y, A, W, V, Delta.W, W.sub, Delta = rep(1, length(Y)),
                          pi = NULL, piform = NULL,
                          pi.SL.library = c("SL.glm", "SL.gam", "SL.glmnet",
                                            "tmle.SL.dbarts.k.5"),
                          V.pi = 10,
                          pi.discreteSL = TRUE, condSetNames = c("A", "W"),
                          id = NULL, Q.family = "gaussian", verbose = FALSE,
                          ...) {
  if (!requireNamespace("tmle", versionCheck = ">=2.0", quietly = FALSE)) {
    stop("Loading required tmle package (>-2.0) failed", call. = FALSE)
  }

  if (is.null(id)) {
    id <- 1:length(Y)
  }

  if (is.vector(W.sub)) {
    W.sub <- as.matrix(W.sub)
    colnames(W.sub) <- "W.sub"
  }
  V <- as.matrix(V)
```

```
  # Evaluate conditional sampling probabilities
  if (is.null(pi)) {
    validCondSetNames <- c("A", "V", "W", "Y")
    if (!(all(condSetNames %in% validCondSetNames))) {
      stop("condSetNames must be any combination of 'A', 'V', 'W', 'Y'")
    }
    if (any(condSetNames == "Y")) {
      if (any(is.na(Y))) {
        stop(paste0("Cannot condition on the outcome to evaluate sampling ",
                    "probabilities when some outcome values are missing"))
      }
    }

    d.pi <- data.frame(Delta.W = Delta.W, mget(condSetNames))
    colnames(d.pi)[-1] <- .getColNames(condSetNames, colnames(W), colnames(V))
    res.subcal <- tmle:::estimateG(
      d = d.pi, g1W = NULL, gform = piform, SL.library = pi.SL.library,
      id = id, V = V.pi, message = "sampling weights", outcome = "A",
      discreteSL = pi.discreteSL,
      obsWeights = rep(1, nrow(W)), verbose = verbose
    )
    names(res.subcal)[1] <- "pi"
  } else {
    res.subcal <- list()
    res.subcal$pi <- pi
    res.subcal$type <- "User supplied values"
    res.subcal$coef <- NA
    res.subcal$discreteSL <- NULL
  }

  obsWeights <- Delta.W / res.subcal$pi

  result <- list()
  result$tmleMSM <- try(tmleMSM(
    Y = Y[Delta.W == 1], A = A[Delta.W == 1],
    W = cbind(W[Delta.W == 1, , drop = FALSE], W.sub),
    V = V[Delta.W == 1, , drop = FALSE], Delta = Delta[Delta.W == 1],
    family = Q.family, obsWeights = obsWeights[Delta.W == 1],
    id = id[Delta.W == 1], verbose = verbose, ...
  ))

  if (inherits(result$tmleMSM, "try-error")) {
    cat(paste0("Error calling tmleMSM. ",
               "Estimated sampling probabilites are being returned"))
    result$tmleMSM <- NULL
  }
  result$subcal <- res.subcal
  class(result) <- "subcalTMLE" # for tmleMSM can use same class
  return(result)
}
```

References

Bickel, Peter J, Chris AJ Klaassen, Peter J Bickel, Ya'acov Ritov, J Klaassen, and Jon A Wellner. 1993. *Efficient and Adaptive Estimation for Semiparametric Models*. Vol. 4. Springer.

Cole, Stephen R, and Constantine E Frangakis. 2009. "The Consistency Statement in Causal Inference: A Definition or an Assumption?" *Epidemiology* 20 (1): 3–5.

Gruber, Susan, and Mark van der Laan. 2012. "Tmle: An R Package for Targeted Maximum Likelihood Estimation." *Journal of Statistical Software* 51: 1–35.

———. 2024. "twoStageDesignTMLE: An R Package for Analyzing Two-Stage Design Study Data with TMLE." https://CRAN.R-project.org/package=twoStageDesignTMLE.

Robins, James M, Miguel Angel Hernán, and Babette Brumback. 2000. "Marginal Structural Models and Causal Inference in Epidemiology." *Epidemiology*, 550–60.

Rose, Sherri, and Mark J van der Laan. 2011. "A Targeted Maximum Likelihood Estimator for Two-Stage Designs." *The International Journal of Biostatistics* 7 (1): Article 17.

Rubin, Donald B. 1974. "Estimating Causal Effects of Treatments in Randomized and Nonrandomized Studies." *Journal of Educational Psychology* 66 (5): 688.

van der Laan, Mark J, Eric C Polley, and Alan E Hubbard. 2007. "Super Learner." *Statistical Applications in Genetics and Molecular Biology* 6 (1).

van der Laan, Mark J, and Sherri Rose. 2011. *Targeted Learning: Causal Inference for Observational and Experimental Data*. Vol. 4. Springer.

van der Laan, Mark J, and Daniel Rubin. 2006. "Targeted Maximum Likelihood Learning." *The International Journal of Biostatistics* 2 (1).