Modern Statistical Learning Methods for Observational Biomedical Data

Lab 2: TMLE for single timepoint interventions

David Benkeser
Emory Univ.

Marco Carone
Univ. of Washington

Larry Kessler
Univ. of Washington

Module 14

5th Annual Summer Institute for Statistics in Clinical Research

07/22/2018

Installing R packages

To follow along with this demonstration, you will need several R packages.

- Packages are freely available from the Comprehensive R Archive Network (CRAN).
- Packages are downloaded to your local computer via the `install.packages` function.
- Packages are loaded into your current R session via `require` or `library`

```
# these packages are needed to execute the demo
pkgs <- c("drtmle","earth","SuperLearner","nloptr",
          "quadprog","plotmo","plotrix","TeachingDemos")
# see what packages are currently installed
installed_pacakges <- row.names(installed.packages())
# loop over the needed packages
for(p in pkgs){
  # check if package is installed
  already_installed <- p %in% installed_pacakges
  # if not already installed, install it
  if(!already_installed){
    install.packages(p)
  }
  # and load package
  library(p, character.only = TRUE)
}
```

Simulating data

We will use simulated data sets where we know the truth to demonstrate some key ideas.

- We generate a data set of $n = 300$ observations.
- There are two covariates, $W = (W_1, W_2)$.
- The treatment probability is logistic-linear in W_1, W_2.
- The outcome is generated according to a linear model.
- The true ATE is 0.20.

```
# set a seed for reproducibility
set.seed(212)
# sample size
n <- 300
# W1 has Normal distribution, W2 has Uniform distribution
W1 <- rnorm(n); W2 <- runif(n)
# make a data.frame of covariates
W <- data.frame(W1 = W1, W2 = W2)
# pr(A = 1 | W) is logistic linear in W
g1W <- plogis(-1 + W1 - W2 + W1*W2)
# generate binary treatment
A <- rbinom(n, 1, g1W)
# E[Y | A, W] is logistic linear in A, W
QAW <- plogis(W1 - W2 + A)
# generate outcome by adding random error
Y <- rbinom(n, 1, QAW)
```

The drtmle package

The drtmle package facilitates doubly-robust estimation and inference about average treatment effects. It is available on CRAN and GitHub.

- These slides are based off the package vignette.

Learning objectives for today:

1. understanding and executing basic calls to drtmle;
2. understanding interface between drtmle and SuperLearner;
3. implementing estimators of additional regressions for robust inference;
4. implementing drtmle with missing treatment assignments and missing outcomes.

Basic calls to `drtmle`

A rundown of the most important options for the `drtmle` function:

- `W` = covariates;
- `A` = treatment assignment (can have multiple discrete values);
- `Y` = outcome (i.e., Y for outcome regression, A for propensity score);
- `family` = `gaussian()` or `binomial()`, description of `Y`;
- `a_0` = `drtmle` will estimate counterfactual mean for all values of `a_0` supplied by user;
- `SL.Q` = super learner library for the outcome regression;
- `SL.g` = super learner library for the propensity score;
- `SL.Qr` = super learner library for residual outcome regression;
- `SL.gr` = super learner library for residual outcome regression;
- `stratify` = should outcome regression pool over levels of `A` (`stratify = FALSE`) or should a separate outcome regression be fit for each level of `A` (`stratify = TRUE`);
- `maxIter` = maximum number of TMLE iterations (sane default is 3, smaller number = faster run);
- `tolg` = truncation level for propensity score (default is 0.01, sanity is context dependent).

Basic calls to drtmle

Let's start by making a simple call to drtmle and parsing the output.

- Get counterfactual mean for both levels of treatment.
- The outcome is binary, so we use family = binomial() for SuperLearner wrappers.
- The super learner library for both regressions uses polynomial multivariate adaptive regression splines and logistic regression.
- The super learner library for the residual regression uses same library.
- We fit a single outcome regression using observations with A = 1 and A = 0.

```
set.seed(123)
fit1 <- drtmle(W = W, A = A, Y = Y, a_0 = c(0,1), family = binomial(),
               SL_g = c("SL.earth", "SL.glm"),
               SL_Q = c("SL.earth", "SL.glm"),
               SL_gr = c("SL.earth", "SL.glm"),
               SL_Qr = c("SL.earth", "SL.glm"),
               stratify = FALSE)
fit1
```

Basic calls to `drtmle`

Let's start by making a simple call to `drtmle` and parsing the output.

- `$est` is the estimates for each value of `a_0` (shown as row name).
- `$cov` is the estimated asymptotic covariance matrix divided by sample size.

```
## $est
##
## 0 0.4317429
## 1 0.7003781
##
## $cov
##                0             1
## 0 1.131273e-03 8.398455e-05
## 1 8.398455e-05 5.985252e-03
```

Basic calls to drtmle

drtmle also contains a ci method for computing doubly-robust confidence intervals.

- contrast specifies what quantity you would like a confidence interval for (default is for counterfactual means). Examples of other quantities on subsequent slides.
- est specifies which estimator to get a confidence interval for (drtmle = doubly-robust confidence intervals; tmle = TMLE confidence intervals; aiptw = AIPTW confidence intervals).
- level determines nominal coverage probability of the interval (default is 95%).

```
ci(fit1)
```

```
## $drtmle
##     est   cil   ciu
## 0 0.432 0.366 0.498
## 1 0.700 0.549 0.852
```

Basic calls to drtmle

If contrast is a vector, then ci computes confidence interval for the dot product of contrast and fit1$est.

- E.g., if contrast = c(1,-1) then ci computes confidence interval for the ATE.

```
ci(fit1, contrast = c(-1, 1))
```

```
## $drtmle
##                     est   cil   ciu
## E[Y(1)]-E[Y(0)] 0.269 0.105 0.432
```

Basic calls to `drtmle`

More generally, `contrast` may be input as a list, which allows the `ci` method to compute confidence intervals of the form

$$f^{-1}\left\{f(h(\psi_n)) \pm z_{1-\alpha/2}\nabla f(h(\psi_n))^T \Sigma_n \nabla f(h(\psi_n))\right\} \text{ , where}$$

- f (`contrast$f`) is the transformation of the confidence interval,
- f^{-1} (`contrast$f_inv`) is the back-transformation of the interval,
- h (`contrast$h`) is a contrast of counterfactual means, and
- $\nabla f(h(\psi_n))$(`contrast$fh_grad`) defines the gradient of the transformed contrast at the estimated counterfactual means.

Basic calls to `drtmle`

For example, we may be interested in the risk ratio $E[Y(0)]/E[Y(1)]$.

- The true risk ratio in the simulated data example is 0.66.

This confidence interval is often computed on the log scale and back-transformed,

$$\exp\left[\log\left\{\frac{\psi_n(1)}{\psi_n(0)}\right\} \pm z_{1-\alpha/2}\sigma_n^{\log}\right] ,$$

where $\sigma_n^{\log}$ is the estimated standard error on the log-scale.

By the delta method, an estimate of the standard error of the log-risk-ratio is

$$\sigma_n^{\log} = \nabla g(\psi_n)^T \Sigma_n \nabla g(\psi_n) ,$$

where Σ_n is the doubly-robust covariance matrix estimate output from `drtmle` and $\nabla g(\psi)$ is the gradient of $\log\{E[Y(1)]/E[Y(0)]\}$.

Basic calls to drtmle

This confidence interval can be computed using the following code.

```
riskRatio <- list(f = function(eff){ log(eff) },
                  f_inv = function(eff){ exp(eff) },
                  h = function(est){ est[1]/est[2] },
                  fh_grad =  function(est){ c(1/est[1],-1/est[2]) })
ci(fit1, contrast = riskRatio)
```

```
## $drtmle
##                  est   cil ciu
## user contrast 0.616 0.475 0.8
```

Basic calls to drtmle

This method of computing confidence intervals can also be useful for constructing confidence intervals about counterfactual means.

- Example: Y is binary so counterfactual mean is between 0 and 1. We would like our confidence interval to respect this.

We can build an interval on the logit scale and back-transform,

$$\text{expit}\left[\log\left\{\frac{\psi_n(1)}{1-\psi_n(1)}\right\} \pm z_{1-\alpha/2}\sigma_n^{\text{logit}}\right].$$

```
logitMean <- list(f = function(eff){ qlogis(eff) },
                  f_inv = function(eff){ plogis(eff) },
                  h = function(est){ est[1] },
                  fh_grad = function(est){ c(1/(est[1] - est[1]^2), 0) })
ci(fit1, contrast = logitMean)
```

```
## $drtmle
##                  est   cil   ciu
## user contrast 0.432 0.367 0.498
```

Basic calls to drtmle

Hypothesis tests can be implemented in much the same way using the `wald_test` method.

- `null` specifies what value to test against.

Here, we perform two two-sided hypothesis tests:

```
wald_test(fit1, null = c(0.5, 0.6))
```

```
## $drtmle
##                    zstat  pval
## H0: E[Y(0)]=0.5 -2.029 0.042
## H0: E[Y(1)]=0.6  1.297 0.194
```

Basic calls to drtmle

As with ci, wald_test allows for testing linear combinations of counterfactual means via the contrast option.

- We can use this to test the null hypothesis that ATE $= 0$.
- We can test that the ATE equals a particular value via null option.

```
# test ATE = 0
wald_test(fit1, contrast = c(1, -1))
```

```
## $drtmle
##                          zstat  pval
## H0:E[Y(0)]-E[Y(1)]=0 -3.223 0.001
```

```
# test ATE = 0.1
wald_test(fit1, contrast = c(1, -1), null = 0.1)
```

```
## $drtmle
##                            zstat pval
## H0:E[Y(0)]-E[Y(1)]=0.1 -4.422    0
```

Basic calls to `drtmle`

`wald_test` also allows testing of arbitrary smooth contrasts of counterfactual means.

We can generally test the null hypothesis that $f(h(\psi_0))$ equals f_0 (the function f applied to the value passed to `null`) using the Wald statistic

$$Z_n := \frac{f(h(\psi_n)) - f_0}{\nabla f(h(\psi_n))^T \Sigma_n \nabla f(h(\psi_n))} .$$

We can use the `riskRatio` contrast defined previously to test $H_0 : E[Y(0)]/E[Y(1)] = 1$.

```
wald_test(fit1, contrast = riskRatio, null = 1)
```

```
## $drtmle
##                          zstat pval
## H0: user contrast = 1 -3.635    0
```

Note on common warning messages

You may see the following warning messages:

```
## Warning:  glm.fit:  fitted probabilities numerically 0 or 1 occurred
```

- Indicates that some logistic regression has lead to predicted values of 0 or 1.
- May indicate over-fitting of a regression, but generally for our purposes safe to ignore.

```
## Warning in method$computeCoef(Z = Z, Y = Y, libraryNames =
## libraryNames, : Algorithm 2 is duplicated. Setting weight to
## 0.
```

- Indicates that two regressions had an identical fit in `SuperLearner`.
- E.g., `SL.step` chooses the intercept-only model, which is the same as `SL.mean`.
- The duplicated algorithm is dropped from the super learner.

Note on the `SL.Qr` and `SL.gr`

Additional residual regressions are needed for doubly-robust inference.

- How these extra regressions are fit is specified via `SL.Qr` and `SL.gr`.

It is difficult to know a-priori how these regressions should be fit. Thus, we recommend being as flexible as possible in their estimation.

- A sensible "default" library might include `SL.glm`, `SL.gam`, `SL.mean`, `SL.earth`, and `SL.npreg`.
- `SL.npreg` is kernel regression with cross-validated bandwidth selection (discussed in Chapter 3).

Note on custom regression estimators

It is not necessary to use super learning to estimate regressions in `drtmle`.

- `glm_` options may be used to fit these regressions (see documentation).

For example, say that to estimate the outcome regression, we would like to use the screening wrapper that we wrote in Lab 1 and

- screen for variables that change the coefficient of `A` more than 10%;
- fit logistic regression with the resulting variables.

Users can write their own wrapper function that is passed to `SL_` option.

- Format of function identical to what is used by `SuperLearner` (see Lab 1).

Note on custom regression estimators

Here is the same screening function from Lab 1.

```
screen_confounders <- function(Y, X, family, trt_name = "A",
                               change = 0.1, ...){
    # fit treatment only model & get coefficient for treatment
    fit_init <- glm(as.formula(paste0("Y ~ ", trt_name)),
                    data = X, family = family)
    trt_coef <- fit_init$coef[2]
    # identify which column of X is the trt variable
    trt_col <- which(colnames(X) == trt_name)
    # set all variables except trt to not be included initially
    include <- rep(FALSE, ncol(X)); include[trt_col] <- TRUE
    # loop over variables in X other than trt
    for(j in seq_len(ncol(X))[-trt_col]){
        # find variable name
        var_name <- colnames(X)[j]
        # fit trt + variable model, get trt coefficient
        fit <- glm(as.formula(paste0("Y ~ ", trt_name, "+", var_name)),
                   data = X, family = family)
        new_trt_coef <- fit$coef[2]
        # check if changed more than specified amount
        include[j] <- abs((new_trt_coef - trt_coef)/trt_coef) > change
    }
    return(include)
}
```

Note on custom regression estimators

Now we write a SuperLearner-style wrapper that does what we want.

```
SL.screened_regression <- function(Y, X, newX, family, ...){
    # screen columns of X using screen_confounders
    include_cols <- screen_confounders(Y = Y, X = X, family = family)
    # fit main terms glm with only those columns
    fitted_glm <- glm(Y ~ ., data = X[ , include_cols], family = family)
    # get predictions
    pred <- predict(fitted_glm, newdata = newX[ , include_cols],
                    type = "response")
    # format output
    out <- list(fit = list(fitted_model = fitted_glm,
                           include_cols = include_cols),
                pred = pred)
    # assign class
    class(out$fit) <- "SL.screened_regression"
    return(out)
}
```

Note on custom regression estimators

We also need to define a `predict` method.

```
predict.SL.screened_regression <- function(object, newdata, ...){
    pred <- predict(object$fitted_model,
                    newdata = newdata[ , object$include_cols],
                    type = "response")
    return(pred)
}
```

Note on custom regression estimators

We can now use `drtmle` with a main-terms logistic regression for the propensity and our custom outcome regression wrapper.

```
set.seed(123)
fit2 <- drtmle(W = W, A = A, Y = Y, a_0 = c(0,1), family = gaussian(),
               # specify main terms logistic regression via glm_g
               glm_g = "W1 + W2",
               # specify custom outcome regression via SL_Qs
               SL_Q = "SL.screened_regression",
               # the residual regression stay the same
               SL_gr = c("SL.earth", "SL.glm", "SL.mean"),
               SL_Qr = c("SL.earth", "SL.glm", "SL.mean"),
               stratify = FALSE)
fit2
```

```
## $est
##
## 0 0.4312075
## 1 0.6927319
##
## $cov
##                0            1
## 0 0.0010901910 0.0001009967
## 1 0.0001009967 0.0053828564
```

Missing data

`drtmle` supports missing data in `A` and `Y`. The estimators we have discussed can be modified to allow for missingness.

- Let Δ_A and Δ_Y be indicators that A and Y are observed, respectively.

The outcome regression is $E(\Delta_Y Y \mid \Delta_A A = a, \Delta_A = 1, \Delta_Y = 1, W)$.

The propensity score is

$$\begin{aligned}&\text{pr}(\Delta_A = 1, \Delta_A A = a, \Delta_Y = 1 \mid W) \\ &\quad = \text{pr}_0(\Delta_A = 1 \mid W) \times \text{pr}_0(\Delta_A A = a \mid \Delta_A = 1, W) \times \text{pr}_0(\Delta_Y = 1 \mid \Delta_A = 1, \Delta_A A = a, W) \ .\end{aligned}$$

We can introduce missing values to `A` and `Y` in our running example.

```
set.seed(123)
DeltaA <- rbinom(n, 1, plogis(2 + W$W1))
DeltaY <- rbinom(n, 1, plogis(2 + W$W2 + A))
A[DeltaA == 0] <- NA
Y[DeltaY == 0] <- NA
```

Missing data

The syntax is the same except for `SL.g`, which now must specify a regression for each of the three components of the PS.

- `SL.g` is now a `list` with named entries `"DeltaA"`, `"A"`, and `"DeltaY"`.
- Each entry in the list specifies the super learner library for that regression.
- If only a single library is passed to `SL.g`, it is used for each of the three regressions.

```
set.seed(123)
fit3 <- drtmle(W = W, A = A, Y = Y, a_0 = c(0,1), family = binomial(),
               SL_g = list(DeltaA = c("SL.earth", "SL.glm", "SL.mean"),
                           A = c("SL.earth", "SL.glm", "SL.mean"),
                           DeltaY = c("SL.glm", "SL.mean")),
               SL_Q = c("SL.earth", "SL.glm", "SL.mean"),
               SL_gr = c("SL.earth", "SL.glm", "SL.mean"),
               SL_Qr = c("SL.earth", "SL.glm", "SL.mean"),
               stratify = FALSE)
```

Missing data

```
fit3
```

```
## $est
##
## 0 0.4428648
## 1 0.8408505
##
## $cov
##                0             1
## 0  1.430528e-03 -1.817527e-05
## 1 -1.817527e-05  1.949183e-03
```

```
# calls to ci and wald_test are same as before
ci(fit3)
```

```
## $drtmle
##     est   cil   ciu
## 0 0.443 0.369 0.517
## 1 0.841 0.754 0.927
```

Multi-level treatments

So far we have only considered binary treatments. However, `drtmle` can handle treatments with an arbitrary number of discrete values.

Suppose A assumes values in $\mathcal{A}$. The propensity score estimation is modified to ensure that

$$\sum_{a \in \mathcal{A}} \hat{\text{pr}}(A = a \mid W = w) = 1 \text{ for all } w \ .$$

The number of multi-level regression methodologies is somewhat limited, so `drtmle` uses a sequence of binary regressions to ensure compatible propensity score estimates.

For example, if $\mathcal{A} = \{0, 1, 2\}$, then obtain estimates

$$\hat{\text{pr}}(A = 0 \mid W) \ , \text{ and } \hat{\text{pr}}(A = 1 \mid A > 0, W) \ ,$$

and we set

$$\begin{aligned}
\hat{\text{pr}}(A = 1 \mid W) &= \hat{\text{pr}}(A = 1 \mid A > 0, W)[1 - \hat{\text{pr}}(A = 0 \mid W)] \\
\hat{\text{pr}}(A = 2 \mid W) &= 1 - \hat{\text{pr}}(A = 0 \mid W) - \hat{\text{pr}}(A = 1 \mid W) \ .
\end{aligned}$$

Multi-level treatments

Here we generate data that has three treatment levels, `A = 0,1,2`.

```
set.seed(1234)
n <- 300
W <- data.frame(W1 = runif(n), W2 = rbinom(n, 1, 0.5))
A <- rbinom(n, 2, plogis(W$W1 + W$W2))
Y <- rbinom(n, 1, plogis(W$W1 + W$W2*A))
```

The call to `drtmle` is the same as before:

```
fit4 <- drtmle(W = W, A = A, Y = Y, stratify = FALSE,
               SL_Q = c("SL.earth", "SL.glm"),
               SL_g = c("SL.earth", "SL.glm"),
               SL_Qr = c("SL.earth", "SL.glm"),
               SL_gr = c("SL.earth", "SL.glm"),
               family = binomial(), a_0 = c(0,1,2))
```

Multi-level treatments

The output now includes an estimated counterfactual mean for each level of treatment.

```
fit4
```

```
## $est
##
## 0 0.7181684
## 1 0.7746543
## 2 0.7720335
##
## $cov
##                0             1             2
## 0  2.279984e-02 -9.057861e-05 -1.146287e-04
## 1 -9.057861e-05  1.481616e-03  9.782748e-05
## 2 -1.146287e-04  9.782748e-05  1.129728e-03
```

Multi-level treatments

The confidence interval and testing procedures extend to multi-level treatments.

```
ci(fit4)
```

```
## $drtmle
##     est   cil   ciu
## 0 0.718 0.422 1.014
## 1 0.775 0.699 0.850
## 2 0.772 0.706 0.838
```

```
wald_test(fit4, null = c(0.4, 0.5, 0.6))
```

```
## $drtmle
##                   zstat  pval
## H0: E[Y(0)]=0.4 2.107 0.035
## H0: E[Y(1)]=0.5 7.135 0.000
## H0: E[Y(2)]=0.6 5.118 0.000
```

Multi-level treatments

The `contrast` option works as well.

```
ci(fit4, contrast = c(-1, 1, 0))
```

```
## $drtmle
##                     est   cil   ciu
## E[Y(1)]-E[Y(0)] 0.056 -0.25 0.363
```

```
ci(fit4, contrast = c(-1, 0, 1))
```

```
## $drtmle
##                     est    cil   ciu
## E[Y(2)]-E[Y(0)] 0.054 -0.251 0.359
```

Multi-level treatments

The `contrast` option works as well. We can modify the previous `riskRatio` object to compute the risk ratio comparing $A = 1$ to $A = 0$:

```
riskRatio_1v0 <- list(f = function(eff){ log(eff) },
                      f_inv = function(eff){ exp(eff) },
                      h = function(est){ est[2]/est[1] },
                      fh_grad =  function(est){ c(1/est[2], -1/est[1], 0) })
ci(fit4, contrast = riskRatio_1v0)
```

```
## $drtmle
##                  est   cil   ciu
## user contrast 1.079 0.725 1.606
```

Multi-level treatments

The `contrast` option works as well. We can modify the previous `riskRatio` object to compute the risk ratio comparing $A = 2$ to $A = 0$:

```
riskRatio_2v0 <- list(f = function(eff){ log(eff) },
                      f_inv = function(eff){ exp(eff) },
                      # note now comparing 3rd to 1st estimate
                      h = function(est){ est[3]/est[1] },
                      fh_grad =  function(est){ c(0, -1/est[1], 1/est[3]) })
ci(fit4, contrast = riskRatio_2v0)
```

```
## $drtmle
##                  est   cil   ciu
## user contrast 1.075 0.944 1.225
```

Example write-up of TMLE analysis

Methods

We estimated the average counterfactual outcome if patients received treatment versus if patients received control using super learning and targeted minimum loss-based estimation with robust inference (Benkeser et al. 2017). This procedure requires regression of the outcome on treatment and confounders and of the treatment on confounders. The set of putative confounders included in these regressions included [. . .]. For the outcome regressions, we estimated the linear combination of candidate regression estimators that minimizes ten-fold cross-validated mean squared-error. We included three candidate regression estimators in the super learner: polynomial multivariate regression splines, main terms logistic regression, and intercept-only regression. The same set of candidate estimators was used for estimating the probability of treatment. However, in this case we estimated the logistic-linear combination of regression estimators that minimizes ten-fold cross-validated negative log-likelihood loss. To produce robust inference, this procedure additionally requires residual smoothing, which was also achieved via super learning (details in Appendix A). We tested the null hypothesis that the average outcomes were the same under treatment versus control using a two-sided, level 0.05 Wald test with robust influence function-based standard errors estimates (Benkeser et al. 2017). Analyses were performed using the SuperLearner and drtmle R packages (Polley et al, 2018; Lendle et al 2017).

Example write-up of TMLE analysis

Results

The super learners for the outcome regression gave weight to several algorithms with the majority of the weight placed on polynomial multivariate adaptive regression splines, while for the treatment probability the main-terms logistic regression received the most weight the most weight (Table 1, Appendix A).

The estimated average counterfactual outcome if patients received treatment at all three time points was . . . (95% CI: . . . , . . .). The estimated average counterfactual outcome if patients received control at all three time points was . . . (95% CI: . . . , . . .). Our test of the null hypothesis of equality of these quantities rejected the null hypothesis (p-value = . . .).