R Lab 5 - Introduction to the `ltmle` Package

Advanced Topics in Causal Inference

Assigned: October 19, 2020
Lab due: October 27, 2020 on bCourses. Please answer all questions and include relevant `R` code. You are encouraged to discuss the assignment in groups, but should not copy code or interpretations verbatim. Upload your own completed lab as a pdf to bCourses.

Last lab:
IPTW estimation and performance evaluation for treatment specific means and MSMs.

Goals for this lab:
1. Gain familiarity with the `ltmle R` package and apply it to estimate causal parameters using point treatment data.
2. Evaluate the bias, variance, MSE, and confidence interval coverage of the g-computation, IPTW, and TMLE estimators in a point treatment setting.

Next lab:
Implement parametric g-computation estimator, ICE representation of longitudinal g-computation formula, and TMLE.

---

1 Introduction and Motivation

The `ltmle` package in R is very useful for estimating parameters from a longitudinal data structure. However, to get started, we will use the package to estimate parameters in the setting of one time point or point treatment. In the point treatment setting, we will also learn that `ltmle` is very useful for multi-level (more than 2) treatments for which the effects can be summarized via estimation of an MSM.

Figure 1: L-TaMaLE package.

2 This lab

For this lab you'll need to have the `ltmle` package installed. To do so, run `install.packages("ltmle")` in your console. Note that you only need to install a package once, so don't keep the `install.packages()` line in your `R` script. Then load the package: `library(ltmle)`.

2.1 To turn in:

In this lab, we won't be answering questions pertaining to each of the data structures, as in previous labs. Instead, go through this document, answering questions written in **bold**.

3 Formatting for the `ltmle` Package

The `ltmle` package takes in *wide-form* data. Wide-form data in this context means every row of the data represents the entire history of a subject – each row is unique to one subject. If we consider an individual row in the data, it should be ordered according to the structural equations, with the baseline variables being on the left. If we stack all of these rows on top of each other, the data is in wide-form. So far, in these labs we have been working with wide-form data. To see this, let us load some data we created in `R` Lab 1.

1. **Load `DataStructure2.RData` using the `load()` function.** Make sure you have specified the correct file path. You should see the dataframe `ObsData2` come up in your global environment.
2. **Look at the `head()` of `ObsData2`.** Based on the definition above, why is this considered to be wide-form data?

4 Arguments of the `ltmle()` function

The `ltmle()` function lives within the `ltmle` package. Type `?ltmle` in your console to look at the function's help file. The first argument of the function is `data`, a dataframe which we must specify. Specifying the `data` argument tells `ltmle()` to look within that dataframe to carry out its computations.

We must also set the nodes. For example, for the argument `Anodes`, we can set by name:

```
> Anodes = c("A1", "A2", "A3", "A4")
```

or by position:

```
> Anodes = c(2, 4, 6, 8)
```

1. **Make a new dataframe `ObsData`. Let it have only the columns `L1`, `A1`, and `Y` of the original data.**
   Hint: use the `subset()` function.
2. **Set the variables `Lnodes`, `Anodes`, and `Ynodes` to the character names they have in `ObsData2`.** For example, make `Lnodes = "L1"`.

5 Quick review of estimators for treatment specific mean

Recall that in Causal I we learned about three estimators: 1) g-computation, 2) Inverse Probability of Treatment Weighting (IPTW), and 3) Targeted maximum Likelihood Estimator (TMLE). These estimators estimate the treatment specific mean under a point treatment setting, or:

$$\Psi^F(P_{UX}) = E_{U,X}[Y_a]$$

In words, this is the counterfactual mean outcome of the population under treatment, $A = a$.

1. G-computation estimator

$$\hat{\psi}_{n,gcomp}(P_n) = \frac{1}{n}\sum_{i=1}^{n} \bar{Q}_n(A_i = a, W_i)$$

where $\bar{Q}_n(A, W)$ is an estimate of $\bar{Q}_0(A, W) = E_0[Y|A, W]$, the outcome regression.

To obtain $\bar{Q}_n$ we will perform a logistic regression, noting the usual limitations of this approach (i.e., if the model is misspecified, the estimator will be biased).

For statistical inference we will use the non-parametric bootstrap, forming the 95% confidence interval from the 2.5% and 97.5% quantiles.

2. IPTW estimator

$$\hat{\psi}_{n,IPTW}(P_n) = \frac{1}{n}\sum_{i=1}^{n} \frac{\mathbb{I}(A_i = a)Y_i}{g_n(A_i = a|W_i)}$$

where $g_n(A = a|W)$ is an estimate of $g_0(A = a|W) = P_0(A = a|W)$. Again, to obtain g_n we will perform a logistic regression, again noting the usual limitations of this approach (i.e., if the model is misspecified, the estimator will be biased).

In the ltmle package the standard error for the IPTW estimator is obtained via the IPTW influence curve, which we approximate by:

$$\hat{IC}_{IPTW}(O_i) = \frac{\mathbb{I}(A_i = a)Y_i}{g_n(A_i = a|W_i)} - \hat{\psi}_{n,IPTW}$$

and the standard error is

$$\hat{SE}_{IPTW} = \frac{\hat{\sigma}\left(\hat{IC}_{IPTW})\right)}{\sqrt{n}}$$

Note: $\hat{\sigma}$ is the symbol for "take sample standard deviation of". This means we plug in the data to the $\hat{IC}$ formula, forming a vector of length, n, and then take the standard deviation of that vector. We then obtain the confidence interval:

$$\hat{\psi}_{n,IPTW} \pm 1.96 \times \hat{SE}_{IPTW}$$

For comparison, we will also implement the non-parametric bootstrap.

3. TMLE

$$\hat{\psi}_{n,TMLE}(P_n) = \frac{1}{n}\sum_{i=1}^{n} \bar{Q}^{\star}_n(A_i = a, W_i)$$

where $\bar{Q}^{\star}_n$ is an update of initial estimate, $\bar{Q}_n$.

As with the g-computation estimator, we will perform a logistic regression to obtain $\bar{Q}_n$, noting that in most (if not all) situations one would have a semi- or non- parametric model and thus employ SuperLearner to estimate $\bar{Q}_n$ before updating.

We first compute the clever covariate:

$$H(A_i, W_i) = \frac{\mathbb{I}(A_i = a)}{g_n(A_i = a|W_i)}$$

where $g_n(A = a|W)$ is an estimate of $g_0(A = a|W) = P_0(A = a|W)$, obtained by logistic regression here, but normally obtained by using SuperLearner if g_0 is unknown.

We then perform a logistic regression of Y regressed on this clever covariate, suppressing the intercept, and using $logit(\bar{Q}_n)$ as the intercept, and obtain the estimated coefficient on the clever covariate, ϵ_n (an approximation of ϵ). That is:

$$logit\left(E_0[Y|A, W]\right) = \epsilon H(A, W) + logit(\bar{Q}_n)$$

Note: the `ltmle` package actually moves the $\frac{1}{g_n}$ term in the clever covariate into the weights of the logistic regression, which reduces the variance of the estimator. Therefore we have:

$$\bar{Q}_n^\star(A_i = a, W_i) = expit[\epsilon_n + logit(\bar{Q}_n(A_i = a, W_i)]$$

using $H(A_i, W_i)$ as a weight. Note that this is an alternative TMLE to the one we implemented in Casual 1 – it uses a different parametric submodel and different loss function. Both are valid; however, this implementation can improve stability in the face of practical postivity violations and is thus the default in the `ltmle` package.

Again, the `ltmle()` package obtains the standard error for TMLE estimates via the TMLE influence curve, which we can approximate as:

$$\hat{IC}_{TMLE}(P_n^\star)(O_i) = \frac{\mathbb{I}(A_i = a)}{g_n(A_i = a|W)}(Y_i - \bar{Q}_n^\star(A_i, W_i)) + \bar{Q}_n^\star(A_i = a, W_i) - \hat{\psi}_{(P_n^\star)}$$

We estimate the standard error as the square root of the variance of the efficient influence curve over n, at our updated distribution, $P_n^\star$:

$$\hat{SE}_{TMLE} = \frac{\hat{\sigma}\left(\hat{IC}_{TMLE}(P_n^\star)\right)}{\sqrt{n}}$$

and form the 95% confidence interval:

$$\hat{\psi}_{n,TMLE}(P_n) \pm 1.96 \times \hat{SE}_{TMLE}$$

Again, for comparison we will also implement the non-parametric bootstrap.

6 Estimating treatment specific mean using `ltmle()` for g-computation, IPTW, and TMLE estimators

The `ltmle()` package implements all of the above estimators and their corresponding confidence intervals for the treatment specific mean, risk difference, risk ratio and odds ratio. By default, the `ltmle()` function will compute IPTW and TMLE estimates with inference as described above.

1. **Obtain estimates of the treatment specific mean under the intervention A(1) = 1 using TMLE and IPTW.** That is, use the `ltmle()` function to obtain IPTW and TMLE estimates for $E[Y_{a(1)=1}]$ (i.e., the counterfactual mean test score forcing all students to get 8 or more hours of sleep on the first night of the study). Specify the following arguments:

- `data = ObsData`
- the relevant nodes (created above)
- `abar = 1` to specify the intervention of interest
- `variance.method = "ic"` to estimate variance from the influence curve.
  Note: the default `variance.method = "tmle"` gives a more robust variance estimate in the case of practical positivity violations, but it's slower and more complex (feel free to try it out! :)).

Store this result as the object `results.ltmle1`.

2. **Obtain estimates of the treatment specific mean under the intervention A(1) = 1 using g-computation and IPTW.** Again, use the `ltmle()` function to generate IPTW and g-computation estimates for $E[Y_{a(1)=1}]$. Use the same function specifications as in the previous question, and also specify `gcomp = TRUE`. Store this result as the object `results.ltmle2`.

3. **Print the summary of results for the TMLE, IPTW and g-computation estimators**. Within the `summary()` function, specify `"tmle"` or `"iptw"` as an argument for `results.ltmle1`; specify `"gcomp"` or `"iptw"` as an argument for `results.ltmle2`. To see what the `ltmle()` function returns, look at the help page to the section titled "Value". This will tell you what kind of objects `results.ltmle1` and `results.ltmle2` are and what they contain.

4. **Store the g-computation, IPTW, and TMLE estimates of the treatment specific means as objects.** For example, for the g-computation estimate:

   ```
   > PsiHat.gcomp = results.ltmle2$estimates["gcomp"]
   ```

   The true value of $\Psi(P_{U,X}) = E_{U,X}[Y_{a(1)=1}] = 68.2715$. How do the three estimators compare to the true causal parameter value? How would you go about evaluating how well the estimators did at estimating the true parameter value?

7 Non-parametric bootstrapping for inference

As you may have noticed in the previous section, a warning message came up for the g-computation estimator letting us know inference is based on the TMLE influence curve, and therefore it is not accurate. In this section we are going to use the non-parametric bootstrap to generate variance estimates of the sampling distribution to get inference on the three estimators we calculated above.

Recall that the non-parametric bootstrap approximates resampling from P_0 by resampling from the empirical distribution P_n. The steps are:

1. Generate a single bootstrap sample by sampling *with replacement* n times from the original sample. This puts a weight of $1/n$ on each resampled observation.
2. Apply our estimator to the bootstrap sample to obtain a point estimate.
3. Repeat this process B times. This gives us an estimate of the distribution of our estimator.
4. Estimate the variance of the estimator across the B bootstrap samples:

$$\hat{\sigma}^2_{Boot} = \frac{1}{B}\sum_{b=1}^{B}\left(\hat{\Psi}(P_n^b) - \bar{\hat{\Psi}}(P_n^b)\right)^2$$

   where P_n^b is the b^{th} bootstrap sample from the empirical distribution P_n and $\bar{\hat{\Psi}}(P_n^b)$ is the average of the point estimates across the bootstrapped samples.
5. Assuming a normal distribution, a 95% confidence interval is

$$\hat{\Psi}(P_n) \pm 1.96\ \hat{\sigma}_{Boot}$$

   Alternatively, we can use the 2.5% and 97.5% quantiles of the bootstrap distribution.

Note: Theory supporting the use of the non-parametric bootstrap relies on 1) the estimator being asymptotically linear at P_0 and 2) the estimator not changing behavior drastically if sample from a distribution P_n near P_0.

1. **Set `n` to the number of rows in ObsData**.
2. **Set `B` to the number of bootstrap samples**. When writing the code, set `B` to 5. Then after we are sure the code is working properly, increase `B` to 500.
3. **Create data frame `estimates` as an empty matrix with `B` rows (number of bootstrap estimates) by 3 columns (for g-computation, IPTW, and TMLE).**
4. **Within a `for` loop from `b in 1:B`**:
   (a) Create bootstrap sample `bootData` by sampling with replacement from the observed data. First, use the `sample()` function to sample the indices $1, \ldots, n$ with replacement. Then, assign the observed data from the re-sampled subjects to `bootData`.
   ```
   > bootIndices = sample(1:n, replace=T)
   > bootData = ObsData[bootIndices,]
   ```
   (b) Estimate the treatment specific means using the g-computation estimator, IPTW and TMLE.
   Hint: Copy the relevant code from the previous section, but be sure to change the `ObsData` to `bootData` where appropriate.
   (c) Save the estimates `PsiHat.gcomp.b`, `PsiHat.IPTW.b` and `PsiHat.TMLE.b` as row `b` in matrix `estimates`. We are appending `.b` in order to distinguish the point estimates from the observed data and the point estimates from the bootstrapped samples.
5. **When you are confident that your code is working, increase the number of bootstrapped samples `B` and rerun your code**. Warning: creating `B=500` bootstrapped samples and running the estimators can take a long time.
6. **Assuming a normal distribution, compute the 95% confidence interval for the point estimates**.
7. **Use the `quantiles()` function to obtain the 2.5% and 97.5% quantiles of the bootstrap distribution and to compute the 95% confidence interval for the point estimates**.

8 Incorporating SuperLearner for `ltmle()`

So far we have not estimated g_0 and Q_0 using data adaptive methods. The `ltmle()` function has an `SL.library` argument, where one can specify the SuperLearner library used to estimate both g_0 and Q_0 just as we did in Causal Inference I. This is very important when our model is semi- or non-parametric, which is often the case.

If SL.library is not specified, then `ltmle()` just uses `glm`. In other words, the default is to fit a logistic regression with each past variable as a linear main term. Be careful, as this will not work well if you have an extensive longitudinal data structure (you can quickly get into an overfit of both g and Q).

1. **Set the object `SL.library` to the character vector `c("SL.glm", "SL.bayesglm", "SL.mean")`**
2. **Rerun the estimation of $E[Y_{a=1}]$ from Section 6, specifying the SuperLearner library.**

9 Estimating additive treatment effect using `ltmle()` for g-computation, IPTW, and TMLE estimators

The `ltmle()` package can estimate the additive treatment effect (if you have a continuous outcome) or the risk difference, risk ratio, and odds ratio (if you have a binary outcome) in just one line of code. Here, we will

estimate the additive treatment effect. In order to estimate this quantity, we can implement the same code as the above section; we just need to declare the comparison of two regimens, in this case, $A(1) = 1$ versus $A(1) = 0$. This must be entered as a list in the `abar` argument. Foreshadowing to the last lab: if we had a multiple time point treatment, say three, and we wanted to compare the means under treatment $\bar{A}(3) = (1, 1, 1)$ and control $\bar{A}(3) = (0, 0, 0)$, we would enter `abar = list(c(1, 1, 1), c(0, 0, 0))` as the argument. `ltmle()` calls the first item in the list the treatment and the second the control.

1. **Build off of the previous section and use the `ltmle()` function to estimate the additive treatment effect using IPTW, TMLE, and the g-computation estimators.** Remember to specify the correct regime for the argument `abar`.
2. **Print the summary of each of the three results.**

10 MSMs in the point treatment setting using `ltmleMSM()`

Though the "l" in ltmle stands for "longitudinal", we can use the package as we have seen, for simple single time point problems (a special case of causal effects of multiple intervention nodes). We can also use it to construct estimates where there is multi-level treatment in the point treatment setting. When there are more than two levels of treatment, often we can summarize the relationship between levels of treatment and the outcome using an MSM.

Consider a patient who comes in for a visit where two baseline variables are measured, $W1$ and $W2$. The patient then receives one of three "dose" levels of exposure, denoted by A. Then, the presence of disease is ascertained, indicated by the binary variable Y, where $Y = 1$ if the disease is present.

We will use `ltmleMSM()` to estimate the parameters of the following MSM used to summarize the relationship between exposure category (dose level) and the outcome (disease):

$$m(a|\beta) = logit\big(E_{U,X}[Y_a]\big) = logit\big(P_{U,X}(Y_a = 1)\big) = \beta_0 + \beta_1 a$$

1. **Load `ObsDataMSM.RData`.** You should see a dataframe `ObsDataMSM` come up in your global environment.
2. **Look at the `head()` and `summary()` of `ObsDataMSM` to check out its variables**.
3. **Create binary variables `A1` and `A2` within `ObsDataMSM` from all combinations of `A`.** That is:
   - If $A = 1$, then `A1` $= 1$ and `A2` $= 0$
   - If $A = 2$, then `A1` $= 0$ and `A2` $= 1$
   - If $A = 3$, then `A1` $= 0$ and `A2` $= 0$

   Remember that the variables of the data used in `ltmle` must be ordered correctly, so **reorder the columns of `ObsDataMSM`**:

   ```
   > ObsDataMSM = ObsDataMSM[, c("W1", "W2", "A1", "A2", "Y")]
   ```

4. **Create the object `regimes`, which we will set as the `regimes` argument in the `ltmleMSM()` function.** In this example, a regime is a point treatment represented as a multinomial binary vector, but when dealing with longitudinal data, regimes will represent a series of binary treatments. Regimes can be declared in two different ways in `ltmleMSM()` (implement one of the two below):

   (a) *Using a three dimensional binary array, setting all n subjects to fixed regimes.* Make a new binary array called `regimes` using the `array()` function. Let it have dimensions: $n\times$ number of A nodes $\times$ number of regimes. That is, `regimes[a, b, c]` is the binary treatment of interest for participant `a`, at the $\texttt{b}^{th}$ treatment node, for regime `c`. The first 4 rows should look like this:

```
> regimes[1:4, , ]
, , 1

     [,1] [,2]
[1,]    1    0
[2,]    1    0
[3,]    1    0
[4,]    1    0

, , 2

     [,1] [,2]
[1,]    0    1
[2,]    0    1
[3,]    0    1
[4,]    0    1

, , 3

     [,1] [,2]
[1,]    0    0
[2,]    0    0
[3,]    0    0
[4,]    0    0
```

(b) *Using "rule" functions,* which uses the following code:

```
> regimes = list(function (row) c(1, 0),
+                function (row) c(0, 1),
+                function (row) c(0, 0))
```

What `ltmleMSM()` does here is apply the function to each ID's row for each of the 3 regimes. It is short way of doing what we did in the previous approach. The first function merely sets all ID's regimes to $(1, 0)$, giving the equivalent `regimes[, ,1]` as in the first approach – and so forth.

5. **Create the object `summary.measures`, which we will set as the `summary.measures` argument in the `ltmleMSM()` function.**
   - The `summary.measures` argument is a three dimensional array with the following dimensions: number of regimes $\times$ number of summary measures $\times$ number of time points.
   - The column names of `summary.measures` must be named. Make the column name `"level"` using the `colnames()` function.
   - Note: if we specify a working MSM (i.e., insert a formula into the `working.msm` argument, which we will do next), all of the terms in `working.msm` must be columns of `summary.measures` (or baseline covariates).
6. **Create the object `working.msm`, which we will set as the `working.msm` argument in the `ltmleMSM()` function.** Set it equal to the character `"Y` $\sim$ `level"`.
7. **Run `ltmleMSM()` to estimate the parameters of the above MSM**, specifying the following arguments: `data, Anodes, Ynodes, working.msm, regimes, summary.measures, variance.method`. Give results for IPTW, TMLE, and g-computation estimators. As in the previous examples, to implement the g-computation estimator, you will have to call the function again specifying `gcomp = TRUE`.

Solution:

```
> # 1. load ObsDataMSM function
> load("../ObsDataMSM.RData")

> # 2. look at head and summary
> head(ObsDataMSM)

          W1         W2 A Y
1  0.1200021  0.2924051 2 1
2  0.1406192  0.1284775 3 0
3 -0.6337848 -0.3677661 1 0
4 -1.1497352 -0.1755746 2 0
5 -0.9552687  0.3633784 1 0
6 -0.1565968 -0.3831401 2 0

> summary(ObsDataMSM)

       W1                  W2                 A               Y
 Min.   :-3.51257   Min.   :-3.33216   Min.   :1.000   Min.   :0.000
 1st Qu.:-0.74654   1st Qu.:-0.60948   1st Qu.:1.000   1st Qu.:0.000
 Median :-0.03766   Median : 0.06390   Median :2.000   Median :0.000
 Mean   :-0.03880   Mean   : 0.05156   Mean   :1.807   Mean   :0.224
 3rd Qu.: 0.66595   3rd Qu.: 0.70014   3rd Qu.:2.000   3rd Qu.:0.000
 Max.   : 3.02020   Max.   : 3.07162   Max.   :3.000   Max.   :1.000

> # 3. create A1 and A2 within ObsDataMSM
> ObsDataMSM$A1 = as.numeric(ObsDataMSM$A == 1)
> ObsDataMSM$A2 = as.numeric(ObsDataMSM$A == 2)

> # reorder data
> ObsDataMSM = ObsDataMSM[, c("W1", "W2", "A1", "A2", "Y")]

> # check data
> head(ObsDataMSM)

          W1         W2 A1 A2 Y
1  0.1200021  0.2924051  0  1 1
2  0.1406192  0.1284775  0  0 0
3 -0.6337848 -0.3677661  1  0 0
4 -1.1497352 -0.1755746  0  1 0
5 -0.9552687  0.3633784  1  0 0
6 -0.1565968 -0.3831401  0  1 0

> summary(ObsDataMSM)

       W1                  W2                 A1              A2
 Min.   :-3.51257   Min.   :-3.33216   Min.   :0.000   Min.   :0.000
 1st Qu.:-0.74654   1st Qu.:-0.60948   1st Qu.:0.000   1st Qu.:0.000
 Median :-0.03766   Median : 0.06390   Median :0.000   Median :1.000
 Mean   :-0.03880   Mean   : 0.05156   Mean   :0.277   Mean   :0.639
```

```
 3rd Qu.: 0.66595   3rd Qu.: 0.70014   3rd Qu.:1.000   3rd Qu.:1.000
 Max.   : 3.02020   Max.   : 3.07162   Max.   :1.000   Max.   :1.000
       Y
 Min.   :0.000
 1st Qu.:0.000
 Median :0.000
 Mean   :0.224
 3rd Qu.:0.000
 Max.   :1.000

> # 4. create regimes
> # method 1: 3D binary array
> # dimensions: n X number of Anodes X number of regimes
> regimes = array(dim = c(1000, 2, 3))
> for (a in 1:3) {
+   regimes[, 1, a] <- as.numeric(a == 1)
+   regimes[, 2, a] <- as.numeric(a == 2)
+ }

> # method 2: rule functions
> regimes = list(function (row) c(1, 0),
+                function (row) c(0, 1),
+                function (row) c(0, 0))

> # 5. create summary.measures
> # dimensions = 3 regimes X 1 summary measure X 1 time point
> summary.measures = array(1:3, dim = c(3, 1, 1))
> # name column
> colnames(summary.measures) = "level"

> # 6. create working.msm
> working.msm = "Y ~ level"

> # 7. run ltmleMSM() with above parameters
> resultsMSM1 = ltmleMSM(data = ObsDataMSM,
+                        Anodes = c("A1", "A2"), Ynodes = "Y",
+                        regimes = regimes,
+                        summary.measures = summary.measures,
+                        variance.method = "ic",
+                        working.msm = "Y ~ level")
> resultsMSM2 = ltmleMSM(data = ObsDataMSM,
+                              Anodes = c("A1", "A2"), Ynodes = "Y",
+                              regimes = regimes,
+                              summary.measures = summary.measures,
+                              variance.method = "ic",
+                              working.msm = "Y ~ level",
+                              gcomp = TRUE)

> summary(resultsMSM1, "iptw")

Estimator:  iptw
            Estimate Std. Error CI 2.5% CI 97.5% p-value
```

```
(Intercept)  -0.6430     0.2868 -1.2051   -0.081  0.0250 *
level        -0.3635     0.1588 -0.6749   -0.052  0.0221 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
> summary(resultsMSM1, "tmle")
```

```
Estimator:  tmle
            Estimate Std. Error CI 2.5% CI 97.5% p-value
(Intercept)  -0.5997     0.2639 -1.1169   -0.083 0.02304 *
level        -0.3885     0.1453 -0.6732   -0.104 0.00748 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
> summary(resultsMSM2, "gcomp")
```

```
Estimator:  gcomp
Warning: inference for gcomp is not accurate! It is based on TMLE influence curves.
            Estimate Std. Error CI 2.5% CI 97.5% p-value
(Intercept)  -0.5287     0.2616 -1.0414   -0.016 0.04326 *
level        -0.4210     0.1447 -0.7047   -0.137 0.00362 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

11 Optional Feedback

You may attach responses to these questions to your lab. Thank you in advance!

1. Did you catch any errors in this lab? If so, where?
2. What did you learn in this lab?
3. Do you think that this lab met the goals listed at the beginning?
4. What else would you have liked to review? What would have helped your understanding?
5. Any other feedback?