Supplemental Appendix A

Lucy D'Agostino McGowan

2018-09-07

Here we are going to demonstrate how to calculate the Observed Covariate E-Value and create the accompanying observed bias plot. We are going to use the same example highlighted in the main text, using the Right Heart Catheterization (RHC) dataset, originally used in Connors et al. (1). The first section, *Data processing / fitting the full model* simply demonstrates how to conduct one type of analysis on these data. The second section ***tipr** Demonstration* shows how to use our package, **tipr**, to calculate Observed Covariate E-values as well as an observed bias plot using the analysis set up in the first section.

Data processing / fitting the full model

We need four packages to run this analysis. The **tipr** packages can be installed from GitHub using the following code.

Packages

- **tipr**: to calculate the Observed Covariate E-value and set up the data for the observed bias plot.
- **tidyverse**: to process the data. *Note: This could also be accomplished without this package, we are simply demonstrating one way to to this.*
- **survey**: to implement propensity score weighting.
- **survival**: to complete the outcome analysis.

```
# install.packages("remotes")
remotes::install_github("LucyMcGowan/tipr",
                        ref = "observed-bias-plot")


library(tipr)
library(tidyverse)
library(survey)
library(survival)
```

The RHC data can be obtained from the Vanderbilt Datasets Wiki page.

```
rhc <- read_csv("http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/rhc.csv")
```

Data Cleaning

Here our exposure is `swang1`, our 30 day survival time is `t3d30` and our event variable is `dth30`. We are going to update my exposure variable to take the values `0` and `1` rather than `RHC` and `No RHC` and rename this variable `exposure`. We will will also rename `t3d30` `time` and update `dth30` to take values `0` and `1`, calling this new variable `event`.

```
rhc <- rhc %>%
  mutate(
    exposure = case_when(
      swang1 == "RHC" ~ 1,
      TRUE ~ 0
    ),
    time = t3d30,
    event = case_when(
      dth30 == "Yes" ~ 1,
      TRUE ~ 0
    )
  )
```

We have chosen the following 20 variables to include in this analysis.

```
vars <- c("renalhx", "gibledhx", "transhx", "aps1", "wblc1",
          "hrt1", "pafi1", "alb1", "hema1", "bili1",
          "meanbp1", "paco21", "dnr1", "ph1", "resp1",
          "neuro", "hema", "sex", "age", "surv2md1")
```

Propensity score model

We fit the full propensity score model using a logistic regression with all 20 variables included. We then add the propensity score `p` to the `rhc` data frame as well as the `weight` (2). For this analysis, we are using a weighted outcome model to incorporate the propensity score for demonstration purposes. Other propensity score methods could be implemented (such as matching, covariate adjustment, etc.) with the methods put forth in this paper.

```
ps_frm <- as.formula(
  paste("exposure ~ ", paste(vars, collapse = "+"))
)

ps_mod <- glm(ps_frm,
              family = binomial(link = "logit"),
              data = rhc)

rhc <- rhc %>%
  mutate(
    p = predict(ps_mod, type = "response"),
    weight = case_when(
      exposure == 1 ~ 1 - p,
      TRUE ~ p
```

Note

For this analysis, we use overlap weights (2). The methods put forth here could work with any appropriate weighting scheme.

```
    )
  )
```

Using the **survey** package, we create a survey design object that incorporates the propensity score weights we have just created. We will use this in the outcome model.

```
weight_des <- svydesign(ids = ~ 1,
                        data = rhc,
                        weights = ~ weight)
```

Outcome model

We fit the full outcome model with all 20 covariates included. We are fitting a weighted survival model, using the `svycoxph()` function from the **survey** package.

```
outcome_frm <- as.formula(
  paste("Surv(time, event) ~ exposure + ",
        paste(vars, collapse = "+"))
)

outcome_mod <- svycoxph(formula = outcome_frm,
                        design = weight_des)
```

Now that we have completed the analysis, let's create a small data frame of the results to use later.

```
full_model <- data.frame(
  point_estimate = exp(coef(outcome_mod)["exposure"]),
  lb = exp(confint(outcome_mod, "exposure")[, 1]),
  ub = exp(confint(outcome_mod, "exposure")[, 2])
)
full_model
```

```
##          point_estimate      lb       ub
## exposure       1.235202 1.11277 1.371105
```

The result from our full model is a HR for the exposure of 1.24 (1.11, 1.37).

tipr *Demonstration*

Now that we have set up a basic demonstration of an analysis of the RHC data, we will demonstrate how to calculate the Observed Covariate E-values as well as create an observed bias plot.

Note

Since we are going to be re-running this outcome model multiple times with different formulas and propensity score weights, we can create a small function that will run this code each time. We will use this later:

```
run_outcome_mod <-
  function(formula, wt) {
    des <- svydesign(ids = ~ 1,
                     data = rhc,
                     weights = wt)
    svycoxph(formula = formula,
             design = des)
    }
```

As mentioned in the main text, one advantage of these methods is they can be performed to examine how removing a *single* observed covariate would change the final result, as well as how removing a *group* of observed covariates would change the result. We are going to examine how removing each variable, one at a time, will affect the final result as well as how removing two groups of variables, "Labs" and "Physiological Measurements", will affect the result.

Choose variable groups to drop simultaneously

We will first create a named list of the two groups of variables.

```
groups <- list(
  "No Labs" =
    c("wblc1", "pafi1", "alb1", "hema1", "bili1", "paco21", "ph1"),
  "No Physiological Measurements" =
    c("aps1", "hrt1", "wblc1", "pafi1", "alb1", "hema1", "bili1",
      "paco21", "meanbp1", "resp1", "ph1"))
```

Create an observed bias data frame

Using the `observed_bias_tbl()` function from **tipr**, we will create a data frame with six columns:

- `dropped`: The variable or group of variables that was dropped from the analysis
- `type`: Explanation of `dropped`, whether it refers to a single covariate (`covariate`) or a group of covariates (`group`)
- `ps_formula`: An updated propensity score formula, based on the dropped variables
- `outcome_formula`: An updated outcome formula, based on the dropped variables
- `ps_model`: A model object for the updated propensity score, refit using the full model provided
- `p`: The updated propensity score

```
(o <- observed_bias_tbl(ps_mod, outcome_mod, groups = groups))
```

```
## # A tibble: 22 x 6
##    dropped type  ps_formula outcome_formula
##    <chr>   <chr> <list>     <list>
##  1 age     cova~ <S3: form~ <S3: formula>
##  2 alb1    cova~ <S3: form~ <S3: formula>
##  3 aps1    cova~ <S3: form~ <S3: formula>
##  4 bili1   cova~ <S3: form~ <S3: formula>
##  5 dnr1    cova~ <S3: form~ <S3: formula>
```

***tipr** function*

The `observed_bias_tbl()` function takes three parameters:

- `ps_mod`: The full propensity score model we fit above
- `outcome_mod`: The full outcome model we fit above
- `groups`: A named list of groups we would like to simultaneously drop from the analysis

```
##  6 gibled~ cova~ <S3: form~ <S3: formula>
##  7 hema    cova~ <S3: form~ <S3: formula>
##  8 hema1   cova~ <S3: form~ <S3: formula>
##  9 hrt1    cova~ <S3: form~ <S3: formula>
## 10 meanbp1 cova~ <S3: form~ <S3: formula>
## # ... with 12 more rows, and 2 more
## #   variables: ps_model <list>, p <list>
```

Because we are doing a weighted analysis, we can use the propensity score, `p`, from this data frame to create our desired weights. If we were doing a different type of analysis, we could use the propensity score, `p`, for that purpose. We are going to add a column to this new data frame called `weight` using the same overlap weights used above.

Note

Notice some of these columns are `list` columns. This allows us to include full model objects, formulas, or all of the propensity scores in a single row. In order to deal with these list columns, we will often use the `map` functions from the **purrr** package (loaded automatically with the **tidyverse** package).

```
o <- o %>%
  mutate(
    weight =
      map(p, ~ case_when(rhc$exposure == 1 ~ 1 - .x, TRUE ~ .x))
  )
```

Now using this `weight`, we can re-run the outcome model, using the function we created above called `run_outcome_mod()` along with the `outcome_formula` column here. We are going to create a new column called `outcome_model`.

```
o <- o %>%
  mutate(
    outcome_model =
      map2(outcome_formula, weight, run_outcome_mod)
  )
```

Explore the output

Let's take a look at what this data frame looks like now for a single row. In the first row, we dropped `age`. First, we can examine the propensity score formula that includes all of the variables except `age`.

```
o[["ps_formula"]][[1]]
```

```
## exposure ~ renalhx + gibledhx + transhx + aps1 + wblc1 + hrt1 +
##     pafi1 + alb1 + hema1 + bili1 + meanbp1 + paco21 + dnr1 +
##     ph1 + resp1 + neuro + hema + sex + surv2md1
## <environment: 0x7fe9d7350de8>
```

Similarly, the outcome formula includes all of the variables except `age`.

```
o[["outcome_formula"]][[1]]
```

```
## Surv(time, event) ~ exposure + renalhx + gibledhx + transhx +
##     aps1 + wblc1 + hrt1 + pafi1 + alb1 + hema1 + bili1 + meanbp1 +
##     paco21 + dnr1 + ph1 + resp1 + neuro + hema + sex + surv2md1
## <environment: 0x7fe9d63496d8>
```

The column `p` includes the propensity scores for this updated propensity score model. We can glance at the first six.

```
head(o[["p"]][[1]])
```

```
##         1         2         3         4
## 0.6378421 0.5404579 0.6158883 0.4703818
##         5         6
## 0.3113701 0.1461648
```

Finally, the columns `ps_model` and `outcome_model` include the updated model calls, using the information in the previous columns. For example, here is our updated outcome model. We can pull the updated coefficients out and exponentiate them.

```
o[["outcome_model"]][[1]] %>%
  coef() %>%
  exp()
```

```
##   exposure    renalhx   gibledhx    transhx
## 1.23386286 1.02624451 1.55072401 1.30577549
##       aps1      wblc1       hrt1      pafi1
## 1.00470994 1.00075025 1.00154206 0.99969610
##       alb1      hema1      bili1    meanbp1
## 0.98095956 1.00260853 1.02975147 1.00072254
##     paco21    dnr1Yes        ph1      resp1
## 0.99373245 2.67626459 0.65170268 0.99606186
##   neuroYes    hemaYes    sexMale   surv2md1
## 1.36363835 1.32492809 1.07073904 0.07236558
```

Note

Similarly, the column `weight` includes the weights for this updated propensity score model.

Note

Notice here that our result, the exposure effect, has slightly changed, from 1.24 in the full analysis to 1.23 in this analysis that dropped `age`.

Add observed bias effects

We can now extract the exposure effect for each of the updated models (using the `outcome_model` column). This step will differ depending on the form of your outcome model. For ours, we can use the `coef()` and `confint()` functions to extract the coefficients and confidence intervals.

```
o <- o %>%
  mutate(
    point_estimate =
      map_dbl(outcome_model, ~ exp(coef(.x)["exposure"])),
```

```
    lb =
      map_dbl(outcome_model, ~ exp(confint(.x, "exposure")[, 1])),
    ub =
      map_dbl(outcome_model, ~ exp(confint(.x, "exposure")[, 2]))
  )
```

Now we have the observed bias effects, defined as the updated hazard ratio, along with a lower bound and upper bound for each the analyses dropping each of the indicated variables.

```
o %>%
  select(dropped, point_estimate, lb, ub)
```

```
## # A tibble: 22 x 4
##    dropped  point_estimate    lb    ub
##    <chr>             <dbl> <dbl> <dbl>
##  1 age                1.23  1.11  1.37
##  2 alb1               1.24  1.11  1.37
##  3 aps1               1.25  1.12  1.38
##  4 bili1              1.24  1.11  1.37
##  5 dnr1               1.11  1.00  1.23
##  6 gibledhx           1.23  1.10  1.36
##  7 hema               1.22  1.10  1.35
##  8 hema1              1.23  1.11  1.37
##  9 hrt1               1.25  1.13  1.39
## 10 meanbp1            1.23  1.11  1.36
## # ... with 12 more rows
```

Add tipping point

In addition to observing the affect of dropping each variable and groups of variables, we can also add rows that demonstrate what the effect would look like if it were shifted to be null, in this case crossing 1. To do this we use the `observed_bias_tip()` function from the **tipr** package.

For this analysis, we are going to do this twice, once to see how the effect would look if the lower bound crossed 1 (plugging in `full_model$lb` for the `tip` argument), and once to see how the effect would look if the point estimate crossed 1 (plugging in `full_model$point_estimate` for the `tip` argument).

```
o <- bind_rows(
  o,
  observed_bias_tip(
    tip = full_model$lb,
    full_model$point_estimate,
```

***tipr** function*

The `observed_bias_tip()` function takes five arguments:

- `tip`: The value you would like to shift
- `point_estimate`: The resulting effect from the full model (in our case the hazard ratio we've saved in the `full_model` object)
- `lb`: The resulting lower bound from the full model
- `ub`: The resulting upper bound from the full model
- `tip_desc`: A description of the tipping point you've chosen

```
    full_model$lb,
    full_model$ub,
    "Hypothetical unmeasured confounder (Tip LB)"),
  observed_bias_tip(
    tip = full_model$point_estimate,
    full_model$point_estimate,
    full_model$lb,
    full_model$ub,
    "Hypothetical unmeasured confounder (Tip Point Est)")
)
```

Add Observed Covariate E-value

Let's add a column that creates the Observed Covariate E-value. To do this, we will use the `observed_covariate_e_value()` function from the **tipr** package. Since we have hazard ratios, we can use the transformation suggested by VanderWeele and Ding (3).

```
o <- o %>%
  mutate(
    e_value = map2_dbl(
      lb,
      ub,
      ~ observed_covariate_e_value(
        lb = full_model$lb,
        ub = full_model$ub,
        lb_adj = .x,
        ub_adj = .y,
        transform = "HR")
    )
  )
```

Observed bias plot

We have calculated our observed bias effects and Observed Covariate E-values, now we are going to create a publication-ready plot that displays these. There are a few steps we can take to improve the aesthetics of the plot.

- Update variable names with the appropriate variable labels.
- Re-order the data frame so it will display the plot in a logical manner. For example, we could order the data frame, `o`, by the lower bound of the observed bias effect, in our case `lb`.

We have created a small function, `observed_bias_order()` in **tipr** to assist with ordering your observed bias data frame for the observed bias plot. This will output the same data frame in the correct order.

***tipr** function*

The `observed_covariate_e_value()` function takes five arguments:

- `lb`: The lower bound from the full model (in our case `full_model$lb`)
- `ub`: The upper bound from the full model
- `lb_adj`: The lower bound from the adjusted model (in our case `o$lb`)
- `ub_adj`: The upper bound from the adjusted model
- `transform`: The transformation (if any) you would like to perform if the input is not a risk ratio, based on VanderWeele and Ding's suggestions. In our case, we will input "HR".

Notice here we are using the `map2_dbl()` function so that we can iterate over every column in our observed bias data frame. If we wanted to calculate the Observed Covariate E-value for a single model, we could just use the `observed_covariate_e_value()` function alone. For example, using the variable `dnr1`, we would plug in the following.

```
observed_covariate_e_value(
  lb = 1.11,
  ub = 1.37,
  lb_adj = 1.00,
  ub_adj = 1.23,
  transform = "HR")
## [1] 1.358969
```

Note

Before creating the plot, we like to merge in variable labels. To do this, we create a file, `var_labels.csv`, with two columns: the variable name and the label, for example

```
var,var_label
adld3p,ADL
age,Age
...
```

We then read this into R and left join it with our data frame, `o`.

```
o <- o %>%
  left_join(
    read_csv("var_labels.csv"),
    by = c("dropped" = "var")
  ) %>%
  mutate(
    dropped = case_when(
      type == "covariate" ~ var_label,
      TRUE ~ dropped
    )
  )
```

```
o <- observed_bias_order(o, "lb")
```

Now we're ready to make the plot.

We are going to demonstrate how this is creating by building the plot up in layers. Let's begin with the `ggplot()` function, giving it our observed bias data frame, `o`.

```
g <- ggplot(data = o)
```

In order to demonstrate what our original result was with the full model, we can add a line at the full model's point estimate along with a rectangle spanning the confidence bounds.

```
g <- g +
  geom_hline(yintercept = full_model$point_estimate,
             lwd = 2,
             color = "light blue") +
  geom_rect(ymin = full_model$lb,
            ymax = full_model$ub,
            xmin = 0,
            xmax = 25,
            alpha = 0.01,
            fill = "light blue")
```

Now we are going to add the observed bias effects (the point estimate, lower bound, and upper bound of each of the models fit after dropping the variable(s)). We do this by feeding the data frame `o` to the `ggplot()` function and using the function `geom_pointrange()` to create a point at the point estimate along with a line for the width of the confidence interval. We are going to also use the `cord_flip()` function to flip the coordinates so that the plot is oriented vertically rather than horizontally.

```
g <- g +
  geom_pointrange(aes(x = dropped,
                      y = point_estimate,
                      ymin = lb,
                      ymax = ub)) +
  coord_flip()
```

Next, we can add the Observed Covariate E-values as well as the traditional E-values to the plot.

```
g <- g +
  geom_point(aes(x = dropped, y = e_value, color = type),
             pch = 8) +
  scale_color_manual(name = "",
```

tipr function

The `observed_bias_order()` function takes two arguments:

- `d`: Your observed bias data frame (in our case, `o`)
- `by`: The name of the variable to order by (in our case we are going to order by the lower bound of the adjusted effect, `lb`)

Note

Here we are demonstrating how to build the observed bias plots using the `ggplot()` function from the **ggplot2** package (loaded automatically with the **tidyverse** package). The tools demonstrated here from the **tipr** package can work with any plotting functions.

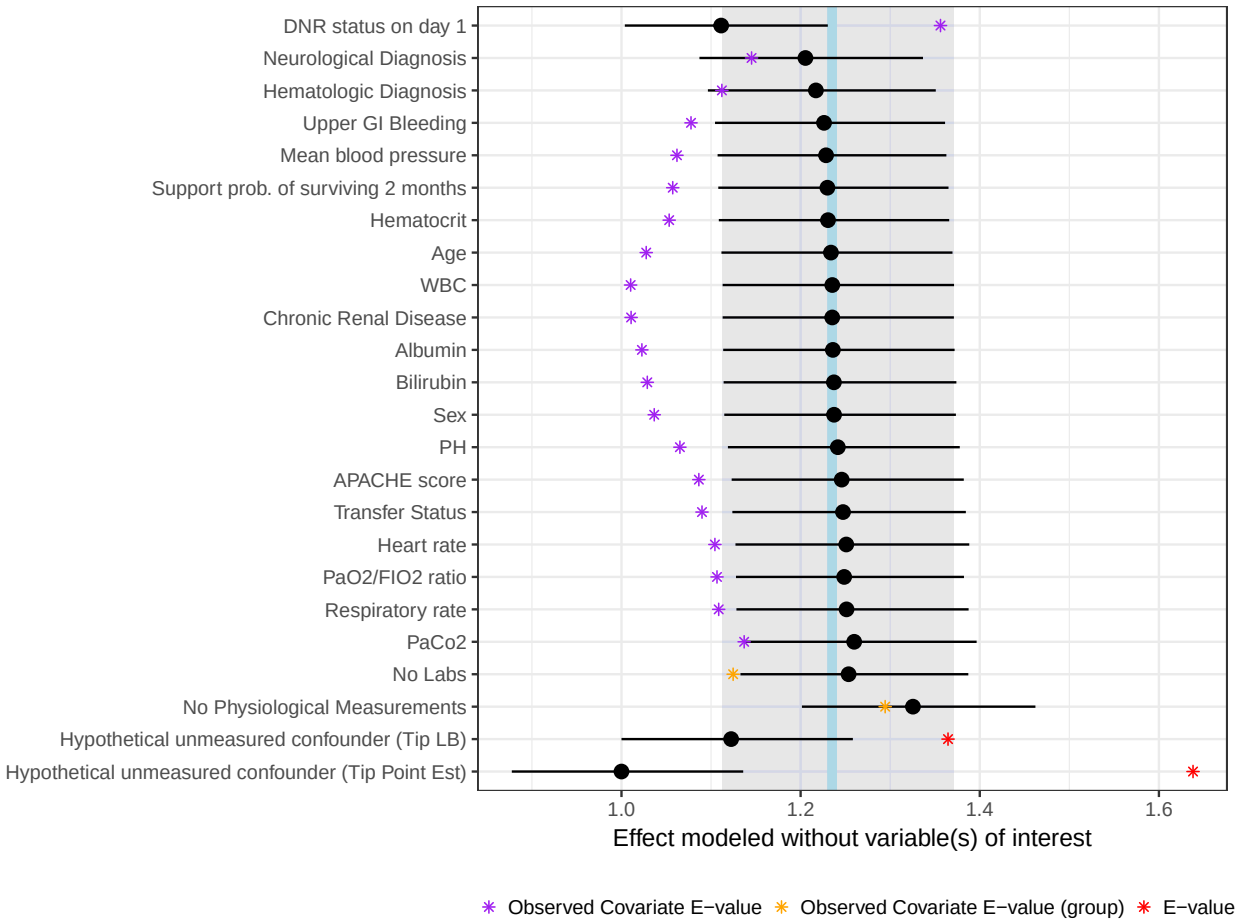
```
                    values = c("covariate" = "purple",
                               "group" = "orange",
                               "tip" = "red"),
                    labels = c("Observed Covariate E-value",
                               "Observed Covariate E-value (group)",
                               "E-value")
  )
```

Finally, we can add some aesthetic updates, such as x- and y-axis labels, a theme, etc.

```
g <- g +
  ylab("Effect modeled without variable(s) of interest") +
  xlab("") +
  theme_bw() +
  theme(legend.title = element_blank(),
        legend.position = "bottom")
```

Let's see how that looks!

```
g
```



1. Connors AF, Speroff T, Dawson NV, Thomas C, Harrell FE, Wagner D, et al. The Effectiveness of Right Heart Catheterization in the Initial Care of Critically Ill Patients. Jama. 1996 Sep;276(11):889–97.

2. Li F, Morgan KL, Zaslavsky AM. Balancing Covariates via Propensity Score Weighting. Journal of the American Statistical Association. 2018;113(521):390–400.

3. VanderWeele TJ, Ding P. Sensitivity Analysis in Observational Research: Introducing the E-Value. Annals of Internal Medicine. 2017 Jul;167(4):268–74.