Regression in Time-Fixed Settings

Ashley I Naimi

Spring 2023

Contents

1 Introduction to Regression

Regression is a cornerstone tool of any empirical analysis. It is arguably the most widely used tool in science. Regression models are often deployed in an attempt to understand cause-effect relations between exposures and outcomes of interest, or to obtain predictions for an outcome of interest. Consider a scenario in which we are interested in regressing an outcome Y against a set of covariates X. These covariates can be an exposure combined with a set of confounders needed for identification, or a set of predictors used to create a prediction algorithm via regression. In its most basic (?) formulation, a regression model can be written as[1]:

$$E(Y \mid X) = f(X)$$

In principle, this model is most flexible in that it states that the conditional mean of Y is simply a *arbitrary function* of the covariates X. We are not stating (or assuming) precisely **how** the conditional mean is related to these covariates. Using this model, we might get predictions from to facilitate a decision making process, or obtain a contrast of expected means between two groups.

Because of the flexibility of this model, we may be interested in fitting it to a given dataset. But we can't. There is simply not enough information in this equation for us to quantify $f(X)$, even if we had all the data we could use. In addition to data, we need some "traction" or "leverage" to be able to quantify the function of interest.

[1] Note that, in this formulation, the function of interest on the left hand side of the equation is the conditional mean function, $E(Y \mid X)$. However, there are other options, including hazards, failure times, distribution quantiles, and many more.

2 Ordinary Least Squares

The earliest attempt to find some "traction" to quantify $f(X)$ was proposed in the 1800s. The approach starts by accepting a few tenets[2]. First, we want the difference between the observed Y for any given individual and the fitted values $f(X)$ for that person to be "small." If we didn't care about the direction of these errors (we usually don't), we can square the error and take it's average[3]:

$$E[(Y - f(X))^2]$$

Thus, we can define the "optimal" $f(X)$ as the function of X that minimizes the mean squared error.

[2] Much of this section is based on the (excellent) forthcoming book by Cosma Shalizi (2019).

[3] Recall that *mean squared error* can be be re-written as the sum of the squared bias and the variance: $E[(Y - f(X))^2] = [E(Y) - f(X)]^2 + Var(Y)$. This formulation might help clarify why we like to minimize it.

You may recall that finding the $f(X)$ that minimizes mean squared error can be achieved by taking the derivative of the mean squared error with respect to $f(X)$, setting it to zero, then solving for $f(X)$.

We've made some progress, but without a better sense of what $f(X)$ looks like, we still can't move forward. For example, there are several functions where either the derivative simply does not exist (e.g., if $f(X)$ is discontinuous), or where the derivative is still complex enough that we can't make progress with finding a unique solution for $f(X)$ that minimizes mean squared error (see technical note on nonlinear models).

Early on, it was recognized that if we select $f(X)$ to be *linear* (more technically, *affine*) the problem of finding the optimal $f(X)$ becomes much easier. That is, if we can simplify $f(X) = b_0 + b_1 X$, then we can use calculus and simple algebra to find an optimal *linear* solution set $b_0 = \beta_0, b_1 = \beta_1$ that minimizes MSE.

In the case of this ordinary least squares regression estimator, taking the derivative

$$E[(Y - [\beta_0 + \beta_1 X])^2]$$

with respect to β_0 and β_1 and rearranging with (matrix) algebra, gives us the least squares "normal" equations, which ultimately leads to the ordinary least squares estimator for $\hat{\beta}$ (Rencher, 2000, Shalizi (2019)):

$$\hat{\beta} = (X^\intercal X^{-1}) X^\intercal y$$

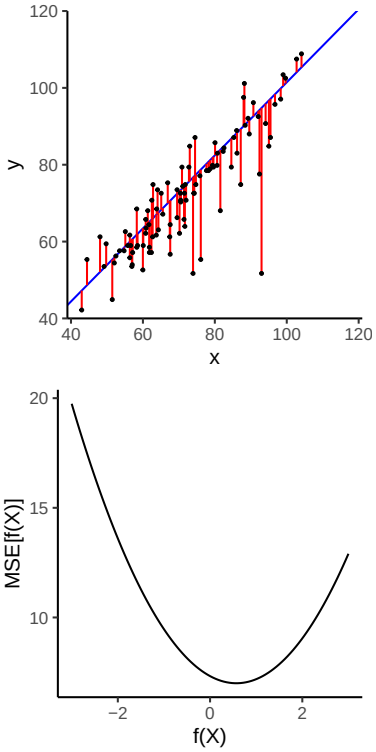


Figure 1: Line of 'best fit' (blue line) defined on the basis of minimizing the sum of squared residuals (red lines) displayed in the top panel; Partial representation of the mean squared error as a function of f(X) in the bottom panel.

Technical Note:

Technically (almost to the point of pedantry), a nonlinear model is a model where the first derivative of the expectation taken with respect to the parameters is itself a function of other parameters (Seber and Wild, 1989). For example,

$$E(Y \mid X) = \beta_0 + \frac{X}{\beta_1}$$

is a nonlinear model, because it's first derivative taken with respect to β_1 is still a function of β_1:

$$\frac{dE(Y \mid X)}{d\beta_1} = \frac{d\left(\beta_0 + \frac{X}{\beta_1}\right)}{d\beta_1} = -\frac{X}{\beta_1^2}$$

Why is this important? Solutions to these regression equations (which serve as our estimates), are obtained by finding where the slope of the tangent line of the parameters is zero. To do this, we need to set the first derivative of these regression equations to zero. But if there are still parameters in these first derivative equations, then there will not be a unique solution to the equation, and finding an estimate will require more complex approaches. This is the complication introduced by nonlinear models.

On the other hand, curvilinear models are easy to find solutions for, since their first derivatives are not functions of parameters. For instance, for a quadratic model such as:

$$E(Y \mid X) = \beta_0 + \beta_1 X + \beta_2 X^2$$

The first derivatives taken with respect to each parameters turn out to be:

$$\frac{dE(Y \mid X, C)}{d\beta_0} = 1$$

$$\frac{dE(Y \mid X, C)}{d\beta_1} = X$$

$$\frac{dE(Y \mid X, C)}{d\beta_2} = X^2$$

Thus, even though the regression function will not be a "straight line" on a plot, this model is still linear.

There are some important points to note in how we formulated the problem of estimating $E(Y \mid X) = f(X)$ with a linear model:

- What we needed to invoke to make this work is that a linear approximation to $f(X)$ is "good enough" for our interest. We need the linear approximation so we can take derivatives of the MSE function without running into problems. These assumptions are usually referred to as "regularity" condi-

tions (Longford, 2008).

- We didn't explicitly state it, but on the basis of Figure 1, this approach makes most sense if Y is continuous. If Y is binary, categorical, or time-to-event, the rationale motivating this approach starts to break down to a degree. That is, while it is possible to fit an OLS estimator to a binary outcome data, not everyone agrees that this is a good idea.
- We *did not* need to invoke any any assumptions about homoscedasticity, or independent and identically distributed (iid) observations. If we were able to make these assumptions, then we obtain an estimator that is the "best linear unbiased estimator" (Rencher, 2000).
- We *did not* need to invoke any distributional assumptions about Y (or, more specifically, the conditional mean of Y). If we can assume Gaussian with constant variance, then we can equate the OLS estimator with the maximum likelihood estimator with a Gaussian distribution and identity link function.

Unfortunately, in the way we formulated it above, the set of linear models we can use to find a solution $f(X)$ that minimizes $MSE(f(X))$ is limited. For instance, in the case where Y is binary, and $E(Y) = P(Y = 1)$, using a linear model such as $b_0 + b_1X$ can easily lead to problems, most notably that predicted probabilities lie outside of the bounds $[0, 1]$.

An additional problem is how we can judge whether the linear combination of parameters in the model is good enough (bullet point 1 above)? As we'll see, this is rarely an easy task. This issue, which we've referred to previously as "correct model specification" is one reason why machine learning methods are becoming so popular.

3 Maximum Likelihood Estimation

Instead of minimizing mean squared error, we could take another approach where we start with a distribution. To demonstrate the motivation here, consider that we, in fact, have a binary outcome ($Y \in [0, 1]$), and we're interested in regressing this outcome against some variable X.

We'll start with a model for the $P(Y = 1)$, which can be modeled using the binomial distribution, defined as:

$$P(Y = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

To understand this equation, say we're interested in understanding the probability of flipping a 50:50 coin "heads" exactly 5 times out of 10 flips.

Here, the total sample size n is 10, and the event number k is 5. With this, we can compute:

$$P(Y = 5) = \binom{10}{5} .5^5 (1-.5)^{10-5} = 0.246$$

In the context of an epidemiologic study, we're usually interested in the probability of a single event (such as a death). Let's say we're interested in understanding how the probability of death ($Y = 1$) is associated with smoking status ($X = 1$). With a single ($k = 1$) binary outcome, this probability function reduces to the Bernoulli distribution:

$$P(Y = 1) = p^y (1-p)^{1-y}$$

where p captures the probability of death. If we believe that p is a function of smoking status X, we can write $p = f(X)$ as we did above, which will give us something like:

$$P(Y = 1 \mid X) = [f(X)]^y (1 - [f(X)])^{1-y}$$

At this point, let's imagine that we have a dataset of three observations:

ID	Y	X
1	1	1
2	0	1
3	1	0

Let's also imagine that the true probability of death is $f(X = 1) = .3$ for smokers, and $f(X = 0) = .15$ for nonsmokers. With the data in the table above, we could compute (among many things) the probability of the observed Y using the equation above. For each row, we get:

$$P(Y = 1 \mid X) = [0.3]^1 (1 - [0.3])^{1-1} = 0.3$$

$$P(Y = 1 \mid X) = [0.3]^0(1 - [0.3])^{1-0} = 0.7$$

$$P(Y = 1 \mid X) = [0.15]^1(1 - [0.15])^{1-1} = 0.15$$

giving us:

ID	Y	X	$p(Y)$
1	1	1	0.3
2	0	1	0.7
3	1	0	0.15

More generally, for a given value of $f(X)$, we can use the data to determine the observed event probability (i.e., if we change the value of $f(X)$, how does $p(Y)$ change?). We can also calculate the overall observed probability of the outcome in the above dataset. If the events are independent, we just need to multiply the probabilities together: $0.3 \times 0.7 \times 0.15 = 0.0315$. Thus, the probability of the observed data given the value of $f(X)$ is 0.0315.

Unfortunately, this ability to compute the observed event probability is rarely useful, since we usually don't know the value of $f(X)$. Maximum likelihood estimation takes the logic of the above scenario and turns it around. Instead of asking: "for a given value of $f(X)$ and a given dataset, what's the observed event probability?" maximum likelihood estimation asks: "for a given dataset and observed event probability, what's the most likely value of $f(X)$?".

To distinguish the Bernoulli function above as a tool to quantify the most likely value of $f(X)$ (as opposed to using it to quantify the probability of Y), we often re-write it as:

$$\mathcal{L}(f(X) \mid Y, X) = \prod_{i=1}^{3} [f(X_i)]^{y_i}(1 - [f(X_i)])^{1-y_i}$$

Notice this is the same equation as above. The only difference is that we're using it to compute the value of $f(X)$ instead of the value of $P(Y = 1 \mid X)$. To understand exactly how we use this equation to compute the optimal value of $f(X)$, we can rewrite it to be more specific to our data:

$$\mathcal{L}(f(X) \mid Y, X) = f(X = 1) \times [1 - f(X = 1)] \times f(X = 0)$$

Written this way, we can now start to interpret this likelihood function as a measure of compatibility between the data and the estimated value of $f(X)$. The higher the likelihood, the more compatible the estimated value of $f(X)$ is with the observed data. This leads us to the idea of the likelihood function, which we can maximize. Similar to the situation with the least squares estimator above (minimizing MSE), we can take the derivative of this likelihood function, set this derivative to zero, and solve for the optimal value of $f(X)$.

The problem is that there are several details that make taking this derivative considerably more complex. When the outcome is binary, we may elect to constrain $f(X)$ so that the predictions are forced to lie within $[0, 1]$. For instance, we could use the inverse of the logistic function and define $f(X) = \frac{1}{1+\exp(-b_0-b_1X)}$.

However, in this case the derivatives would no longer work out as simply as we'd need them too (see Technical Note) because this is a nonlinear function, which means the derivatives of the model $f(X)$ cannot simply be set to zero, even if it is a linear (affine) function. As it turns out, in the early 1970's, Nelder and Wedderburn (Nelder and Wedderburn, 1972) made a seminal contribution that enabled fitting nonlinear models when the outcome belongs to the exponential family of distributions,[4] and the conditional mean of the outcome can be linked to the covariates through some smooth and invertible linearizing function (which includes the $\log$ and logit functions).

[4] The exponential family of distributions is not to be confused with the exponential distribution. It refers to a family of distributions that can be re-written such that they can be represented in a common form. These distributions include (but are not limited to) the Gaussian, exponential, bernoulli (binomial), Poisson, and negative binomial.

4 Generalized Linear Models

Generalized linear models consist of a family of regression models that are fully characterized by a selected distribution and a link function. That is, to fully specify a GLM, one must select a distribution (which determines the form of the conditional mean and variance of the outcome) and a link function (which determines how the conditional mean of the outcome relates to the covariates).

There are a wide variety of distributions and link functions available in standard statistical software programs that fit GLMs. Here, we'll consider a binary outcome Y with probability $P(Y = 1)$, and focus attention on three link functions:

1. Logit, or the log-odds: $\log P(Y = 1)/[1 - P(Y = 1)]$

2. Log: $\log[P(Y = 1)]$
3. Identity: $P(Y = 1)$.

A common misconception is that to use GLMs correctly, one must choose the distribution that best characterizes the data, as well as the canonical link function corresponding to this distribution. For example, if the outcome is binary, one "must" choose the binomial distribution with the logit link. While the binomial distribution and logit link work well together for binary outcomes, they do not easily provide contrasts like the risk difference or risk ratio, because of the selected link function. Alternative specification of the distribution and link function for GLMs can address this limitation.

4.1 Link Functions and Effect Measures

There is an important relation between the chosen link function, and the interpretation of the coefficients from a GLM. For models of a binary outcome and the logit or log link, this relation stems from the properties and rules governing the natural logarithm. The quotient rule states that $\log(X/Y) = \log(X) - \log(Y)$.

Because of this relation, the natural exponent of the coefficient in a logistic regression model yields an estimate of the odds ratio. However, by the same reasoning, exponentiating the coefficient from a GLM with a log link function and a binomial distribution (i.e., log-binomial regression) yields an estimate of the risk ratio. Alternately, for GLM models with a binomial distribution and identity link function, because logarithms are not used, the unexponentiated coefficient yields an estimate of the risk difference.

Unfortunately, using a binomial distribution can lead to convergence problems with the $\log()$ or identity link functions for reasons that have been explored (Zou, 2004). This will occur when, for example, the combined numerical value of all the independent variables in the model is very large. This can result in estimated probabilities that exceed 1, which violates the very definition of a probability (binomial) model (probabilities can only lie between zero and one) and hence, convergence problems. Let's see how these problems can be overcome.

4.2 A Data Example

We use data from the National Health and Nutrition Examination Survey (NHEFS), available as a companion dataset to the Hernán and Robins (Forthcoming) book. We are interested primarily in the covariate adjusted association (on the risk difference and risk ratio scales) between quitting smoking and a greater than median weight change between 1971 and 1982.

In our analyses, we regress an indicator of greater than median weight change against an indicator of whether the person quit smoking. We adjust for exercise status, sex, age, race, income, marital status, education, and indicators of whether the person was asthmatic or had bronchitis. We start by loading the data:

```
#' Load relevant packages
packages <- c("broom", "here", "tidyverse",
    "skimr", "rlang", "sandwich", "boot",
    "kableExtra")

for (package in packages) {
    if (!require(package, character.only = T,
        quietly = T)) {
        install.packages(package, repos = "http://lib.stat.cmu.edu/R/CRAN")
    }
}

for (package in packages) {
    library(package, character.only = T)
}

#' Define where the data are
file_loc <- url("https://cdn1.sph.harvard.edu/wp-content/uploads/sites/1268/1268/20/nhefs.csv")

#' This begins the process of cleaning and formatting the data
nhefs <- read_csv(file_loc) %>%
    select(qsmk, wt82_71, wt82, wt71, exercise,
```

```
        sex, age, race, income, marital,
        school, asthma, bronch, starts_with("alcohol"),
        -alcoholpy, starts_with("price"),
        starts_with("tax"), starts_with("smoke"),
        smkintensity82_71) %>%
    mutate(income = as.numeric(income > 15),
        marital = as.numeric(marital > 2),
        alcoholfreq = as.numeric(alcoholfreq >
            1)) %>%
    na.omit(.)

factor_names <- c("exercise", "income", "marital",
    "sex", "race", "asthma", "bronch")
nhefs[, factor_names] <- lapply(nhefs[, factor_names],
    factor)

#' Define outcome
nhefs <- nhefs %>%
    mutate(id = row_number(), wt_delta = as.numeric(wt82_71 >
        median(wt82_71)), .before = qsmk)

#' Quick summary of data
nhefs %>%
    print(n = 5)
```

```
## # A tibble: 1,055 x 27
##      id wt_delta  qsmk wt82_71  wt82  wt71 exercise sex     age race  income
##   <int>    <dbl> <dbl>   <dbl> <dbl> <dbl> <fct>    <fct> <dbl> <fct> <fct>
## 1     1        0     0  -10.1   68.9  79.0 2        0        42 1     1
## 2     2        0     0    2.60  61.2  58.6 0        0        36 0     1
## 3     3        1     0    4.99  64.4  59.4 2        0        68 1     0
## 4     4        1     0    4.99  92.1  87.1 1        0        40 0     1
## 5     5        1     0    4.42 103.   99   1        1        43 1     0
## # ... with 1,050 more rows, and 16 more variables: marital <fct>, school <dbl>,
## #   asthma <fct>, bronch <fct>, alcoholfreq <dbl>, alcoholtype <dbl>,
```

```
## #   alcoholhowmuch <dbl>, price71 <dbl>, price82 <dbl>, price71_82 <dbl>,
## #   tax71 <dbl>, tax82 <dbl>, tax71_82 <dbl>, smokeintensity <dbl>,
## #   smokeyrs <dbl>, smkintensity82_71 <dbl>
```

4.3 GLMs for risk differences and ratios

For our analyses of the data described above using GLM with a binomial distributed outcome with a log link function to estimate the risk ratio and identity link function to estimate risk difference, an error is returned:

```
#' Here, we start fitting relevant regression models to the data.
#' modelForm is a regression argument that one can use to regress the
#' outcome (wt_delta) against the exposure (qsmk) and selected confounders.

formulaVars <- paste(names(nhefs)[c(3, 7:16)],
    collapse = "+")
modelForm <- as.formula(paste0("wt_delta ~",
    formulaVars))
modelForm
```

```
## wt_delta ~ qsmk + exercise + sex + age + race + income + marital +
##     school + asthma + bronch + alcoholfreq
```

```
#' This model can be used to quantify a conditionally adjusted
#' odds ratio with correct standard error
modelOR <- glm(modelForm, data = nhefs, family = binomial("logit"))
tidy(modelOR)[2, ]
```

```
## # A tibble: 1 x 5
##   term  estimate std.error statistic   p.value
##   <chr>    <dbl>     <dbl>     <dbl>     <dbl>
## 1 qsmk     0.623     0.153      4.07 0.0000471
```

```
#' This model can be used to quantify a conditionally adjusted risk
#' ratio with with correct standard error
```

```
#' However, error it returns an error and thus does not provide any results.
modelRR_binom <- glm(modelForm, data = nhefs,
    family = binomial("log"))
```

```
## Error: no valid set of coefficients has been found: please supply starting values
```

Why is this error returned? The most likely explanation in this context is as follows: We are modeling $P(Y = 1 \mid X) = \exp\{\beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p\}$. In this context, there may be *no set of values* for the parameters in the model that yield $P(Y = 1 \mid X) < 1$ for every observation in the sample. Because R's glm function (under a binomial distribution) correctly recognizes this as a problem, it returns an error.

Instead, one may resort to using different distributions that are more compatible with the link functions that return the association measures of interest. For the risk ratio, one may use a GLM with a Poisson distribution and log link function. Doing so will return an exposure coefficient whose natural exponent can be interpreted as a risk ratio.

```
#' This model can be used to quantify a conditionally risk ratio
#' using the Poisson distribuiton and log link function.
#' However, because the Poisson distribution is used, the model
#' provides incorrect standard error estimates.
modelRR <- glm(modelForm, data = nhefs, family = poisson("log"))
tidy(modelRR)[2, ]
```

```
## # A tibble: 1 x 5
##   term  estimate std.error statistic p.value
##   <chr>    <dbl>     <dbl>     <dbl>   <dbl>
## 1 qsmk     0.277    0.0982      2.82 0.00482
```

It's important to recognize what we're doing here. We are using this model as a tool to quantify the log mean ratio contrasting $P(Y = 1 \mid X_{qsmk} = 1)$ to $P(Y = 1 \mid X_{qsmk} = 0)$ (all other things being equal). However, we should not generally assume that ever aspect of this model is correct. In particular, note that the max predicted probability from this model is 1.087:

```
summary(modelRR$fitted.values)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.2214  0.3961  0.4865  0.4995  0.5857  1.0873
```

We can use the `augment` function in the `broom` package to evaluate the distribution of these probabilities (among other things):

```
fitted_dat <- augment(modelRR, type.predict = "response")

fitted_dat
```

```
## # A tibble: 1,055 x 18
##    wt_delta  qsmk exercise sex     age race  income marital school asthma bronch
##       <dbl> <dbl> <fct>    <fct> <dbl> <fct> <fct>  <fct>    <dbl> <fct>  <fct>
##  1        0     0 2        0        42 1     1      0            7 0      0
##  2        0     0 0        0        36 0     1      0            9 0      0
##  3        1     0 2        0        68 1     0      1            5 0      0
##  4        1     0 1        0        40 0     1      0           11 0      0
##  5        1     0 1        1        43 1     0      1            9 0      0
##  6        0     0 2        0        51 0     1      0           10 0      0
##  7        1     0 2        0        43 0     1      0           11 0      0
##  8        1     1 1        0        43 0     1      0           12 0      0
##  9        0     0 2        0        34 0     1      0           12 0      0
## 10        1     0 0        1        47 0     1      0           12 0      0
## # ... with 1,045 more rows, and 7 more variables: alcoholfreq <dbl>,
## #   .fitted <dbl>, .resid <dbl>, .std.resid <dbl>, .hat <dbl>, .sigma <dbl>,
## #   .cooksd <dbl>
```

```
plot_hist <- ggplot(fitted_dat) + geom_histogram(aes(.fitted)) +
    scale_y_continuous(expand = c(0, 0)) +
    scale_x_continuous(expand = c(0, 0))

ggsave(here("figures", "2022_02_21-rr_hist_plot.pdf"),
    plot = plot_hist)
```

This distribution is shown in margin Figure 2. We can also see that there are only two observations in the sample with predicted risks greater than 1.

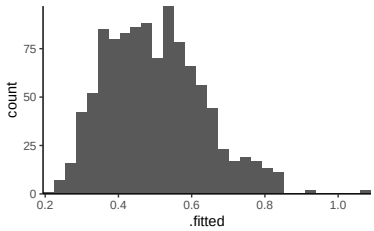


Figure 2: Distribution of fitted values from the Poisson GLM with log link function to obtain an estimate of the adjusted risk ratio for the association between quitting smoking and greater than median weight gain in the NHEFS.

```
fitted_dat %>%
    filter(.fitted >= 1) %>%
    select(wt_delta, qsmk, age, .fitted)
```

```
## # A tibble: 2 x 4
##   wt_delta  qsmk   age .fitted
##      <dbl> <dbl> <dbl>   <dbl>
## 1        1     1    32    1.06
## 2        1     1    25    1.09
```

For these reasons, we are not particularly concerned about the fact that the model predicts risks that are slightly large than 1. However, the model-based standard errors (i.e., the SEs that one typically obtains directly from the GLM output) are no longer valid. Instead, one should use the robust (or sandwich) variance estimator to obtain valid SEs (the bootstrap can also be used) (Zou, 2004).

```
#' To obtain the correct variance, we use the 'sandwich'
#' function to obtain correct sandwich (robust) standard
#' error estimates.
sqrt(sandwich(modelRR)[2, 2])
```

```
## [1] 0.06424196
```

For the risk difference, one may use a GLM with a Gaussian (i.e., normal) distribution and identity link function, or, equivalently, an ordinary least squares estimator. Doing so will return an exposure coefficient that can be interpreted as a risk difference. However, once again the robust variance estimator (or bootstrap) should be used to obtain valid SEs.

```
#' This model can be used to obtain a risk difference
#' with the gaussian distribiton or using ordinary least
#' squares (OLS, via the lm function). Again, the model
```

```
#' based standard error estimates are incorrect.
modelRD <- glm(modelForm, data = nhefs, family = gaussian("identity"))
modelRD <- lm(modelForm, data = nhefs)
tidy(modelRD)[2, ]
```

```
## # A tibble: 1 x 5
##   term  estimate std.error statistic   p.value
##   <chr>    <dbl>     <dbl>     <dbl>     <dbl>
## 1 qsmk     0.145    0.0353      4.10 0.0000438
```

```
#' To obtain the correct variance, we use the 'sandwich' function
#' to obtain correct sandwich (robust) standard error estimates.
sqrt(sandwich(modelRD)[2, 2])
```

```
## [1] 0.03474946
```

The risk ratio and difference, as well as the 95% sandwich variance confidence intervals, obtained for the relation between quitting smoking and greater than median weight change are provided Table 1.

```
knitr::kable(table1_data)
```

Method	Risk Difference	Risk Ratio
GLM	0.14 (0.09, 0.20)	1.32 (1.19, 1.46)
Marginal Standardization	0.14 (0.09, 0.21)	1.31 (1.18, 1.46)

Results in this table obtained using a conditionally adjusted regression model without interactions. Gaussian distribution and identity link was used to obtain the risk difference. A Poisson distribution and log link was used to obtain the risk ratio. 95% CIs obtained via the sandwich variance estimator. 95% CIs obtained using the bias-corrected and accelerated bootstrap CI estimator.

Unfortunately, use of a Poisson or Gaussian distribution for GLMs for a binomial outcome can introduce different problems. For one, while not entirely worrysome in our setting, a model that predicts probabilities greater than one should not instill confidence in the user. Second, performance of the robust variance estimator is notoriously poor with small sample sizes. Finally, the

interpretation of the risk differences and ratios becomes more complex when the exposure interacts with other variables in the model.

Table 3: Methods to use for quantifying conditionally adjusted odds ratios, risk ratios, and risk differences.

Odds Ratio	Risk Ratio	Risk Difference
GLM Family = Binomial	GLM Family = Binomial	GLM Family = Binomial
GLM Link = Logistic	GLM Link = Log	GLM Link = Identity
Standard Errors = Model Based	Standard Errors = Model Based	Standard Errors = Model Based
	GLM Family = Poisson	GLM Family = Gaussian
	GLM Link = Log	GLM Link = Identity
	Standard Errors = Sandwich	Standard Errors = Sandwich
		Least Squares Regression
		Standard Errors = Sandwich

For instance, let's assume that in the NHEFS data, the association between quitting smoking and weight gain interacts with baseline exercise status:

```
#' Potential evidence for interaction
#' between smoking and exercise on the risk difference scale?

table(nhefs$exercise)
```

```
##
##   0   1   2
## 222 465 368
```

```
names(nhefs)[c(3, 7:16)]
```

```
##  [1] "qsmk"        "exercise"    "sex"         "age"         "race"
##  [6] "income"      "marital"     "school"      "asthma"      "bronch"
## [11] "alcoholfreq"
```

```
formulaVars <- paste(names(nhefs)[c(3, 7:16)],
    collapse = "+")

modelForm <- as.formula(paste0("wt_delta ~",
    formulaVars))
modelForm
```

```
## wt_delta ~ qsmk + exercise + sex + age + race + income + marital +
##     school + asthma + bronch + alcoholfreq
```

```
modelForm_int <- as.formula(paste0("wt_delta ~",
    formulaVars, "+ qsmk*exercise"))
modelForm_int
```

```
## wt_delta ~ qsmk + exercise + sex + age + race + income + marital +
##     school + asthma + bronch + alcoholfreq + qsmk * exercise
```

```
summary(glm(modelForm, data = nhefs, family = binomial(link = "identity")))
```

```
##
## Call:
## glm(formula = modelForm, family = binomial(link = "identity"),
##     data = nhefs)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.8774  -1.1109  -0.4881   1.1171   1.7639
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.933190   0.106882   8.731  < 2e-16 ***
## qsmk         0.142133   0.034391   4.133 3.58e-05 ***
## exercise1   -0.051513   0.039628  -1.300   0.1936
## exercise2   -0.086949   0.042210  -2.060   0.0394 *
## sex1         0.013125   0.030968   0.424   0.6717
```

```
## age         -0.009192   0.001266  -7.258 3.92e-13 ***
## race1       -0.054870   0.044979  -1.220   0.2225
## income1     -0.035566   0.045700  -0.778   0.4364
## marital1     0.022557   0.038809   0.581   0.5611
## school      -0.002138   0.005681  -0.376   0.7067
## asthma1      0.116282   0.066745   1.742   0.0815 .
## bronch1     -0.033874   0.057587  -0.588   0.5564
## alcoholfreq  0.032848   0.030762   1.068   0.2856
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1462.5  on 1054  degrees of freedom
## Residual deviance: 1386.6  on 1042  degrees of freedom
## AIC: 1412.6
##
## Number of Fisher Scoring iterations: 5
```

```
summary(glm(modelForm_int, data = nhefs,
    family = binomial(link = "identity")))
```

```
##
## Call:
## glm(formula = modelForm_int, family = binomial(link = "identity"),
##     data = nhefs)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.9778  -1.1015  -0.4924   1.1180   1.7670
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)    0.951771   0.107108   8.886  < 2e-16 ***
## qsmk           0.035309   0.082402   0.428   0.6683
```

```
## exercise1      -0.088077   0.045304  -1.944   0.0519 .
## exercise2      -0.104491   0.047673  -2.192   0.0284 *
## sex1            0.015022   0.030954   0.485   0.6275
## age            -0.009107   0.001269  -7.177 7.11e-13 ***
## race1          -0.056297   0.045041  -1.250   0.2113
## income1        -0.034171   0.045592  -0.750   0.4535
## marital1        0.018684   0.038871   0.481   0.6308
## school         -0.002105   0.005676  -0.371   0.7108
## asthma1         0.123167   0.068232   1.805   0.0711 .
## bronch1        -0.042762   0.057493  -0.744   0.4570
## alcoholfreq     0.030062   0.030759   0.977   0.3284
## qsmk:exercise1  0.157140   0.095850   1.639   0.1011
## qsmk:exercise2  0.090399   0.100681   0.898   0.3693
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1462.5  on 1054  degrees of freedom
## Residual deviance: 1383.8  on 1040  degrees of freedom
## AIC: 1413.8
##
## Number of Fisher Scoring iterations: 5
```

If this were the case, to properly interpret the association, the interaction between exercise status and qsmk should be considered. But how could we do this? One approach would be to include the interaction term and interpret the association between quitting smoking and weight change separately for each level of exercise.

For example, in the model that includes the interaction term with exercise, we can no longer simply interpret the coefficient for `qsmk` as the treatment effet of interest. Instead, (under causal identifiability) we have three treatment effects: The effect of `qsmk` for those with `exercise = 0, 1,` and `2`. If we were, in fact, interested in the average treatment effect in the sample (and not a unique treatment effect for each level of exercise), we would have to take a weighted average of the coefficients for these effects, where the weights are

defined as a function of the proportion of individuals in each exercise level.

Clearly, this approach can quickly become too burdensome when there are several relevant interactions in the model, and is not worth the effort when we are interested in the marginal association. As an alternative, we can use marginal or model-based standardization, which can greatly simplify the process.

4.4 Marginal or Model-Based Standardization

Another approach to obtaining risk differences and ratios from GLMs that are not subject to the limitations noted above is to use marginal standardization, which is equivalent to g computation when the exposure is measured at a single time point (Naimi et al., 2017). This process can be implemented by fitting a single logistic model, regressing the binary outcome against all confounder variables, including all relevant interactions. But instead of reading the coefficients the model, one can obtain odds ratios, risk ratios, or risk differences by using this model to generate predicted risks for each individual under "exposed" and "unexposed" scenarios in the dataset. To obtain standard errors, the entire procedure must be bootstrapped (see supplemental material for code). These marginal risk differences and ratios, as well as their bootstrapped CIs are presented in the table above.

Here is some code to implement this marginal standardization in the NHEFS data:

```
#' Marginal Standardization: version 1
formulaVars <- paste(names(nhefs)[c(3, 7:16)],
    collapse = "+")
modelForm <- as.formula(paste0("wt_delta ~",
    formulaVars, "+ qsmk*exercise"))  ## include the interaction!
modelForm
```

```
## wt_delta ~ qsmk + exercise + sex + age + race + income + marital +
##     school + asthma + bronch + alcoholfreq + qsmk * exercise
```

```r
#' Regress the outcome against the confounders with interaction
ms_model <- glm(modelForm, data = nhefs,
    family = binomial("logit"))
##' Generate predictions for everyone in the sample to obtain
##' unexposed (mu0 predictions) and exposed (mu1 predictions) risks.
mu1 <- predict(ms_model, newdata = transform(nhefs,
    qsmk = 1), type = "response")
mu0 <- predict(ms_model, newdata = transform(nhefs,
    qsmk = 0), type = "response")

#' Marginally adjusted odds ratio
marg_stand_OR <- (mean(mu1)/mean(1 - mu1))/(mean(mu0)/mean(1 -
    mu0))
#' Marginally adjusted risk ratio
marg_stand_RR <- mean(mu1)/mean(mu0)
#' Marginally adjusted risk difference
marg_stand_RD <- mean(mu1) - mean(mu0)

#' Using the bootstrap to obtain confidence intervals for the marginally adjusted
#' risk ratio and risk difference.
bootfunc <- function(data, index) {
    boot_dat <- data[index, ]
    ms_model <- glm(modelForm, data = boot_dat,
        family = binomial("logit"))
    mu1 <- predict(ms_model, newdata = transform(boot_dat,
        qsmk = 1), type = "response")
    mu0 <- predict(ms_model, newdata = transform(boot_dat,
        qsmk = 0), type = "response")

    marg_stand_OR_ <- (mean(mu1)/mean(1 -
        mu1))/(mean(mu0)/mean(1 - mu0))
    marg_stand_RR_ <- mean(mu1)/mean(mu0)
    marg_stand_RD_ <- mean(mu1) - mean(mu0)
    res <- c(marg_stand_RD_, marg_stand_RR_,
```

```
        marg_stand_OR_)
    return(res)
}

#' Run the boot function. Set a seed to obtain reproducibility
set.seed(123)
boot_res <- boot(nhefs, bootfunc, R = 2000)

boot_RD <- boot.ci(boot_res, index = 1)
```

```
## Warning in boot.ci(boot_res, index = 1): bootstrap variances needed for
## studentized intervals
```

```
boot_RR <- boot.ci(boot_res, index = 2)
```

```
## Warning in boot.ci(boot_res, index = 2): bootstrap variances needed for
## studentized intervals
```

```
boot_OR <- boot.ci(boot_res, index = 3)
```

```
## Warning in boot.ci(boot_res, index = 3): bootstrap variances needed for
## studentized intervals
```

```
marg_stand_OR
```

```
## [1] 1.754113
```

```
marg_stand_RR
```

```
## [1] 1.299942
```

```
marg_stand_RD
```

```
## [1] 0.1389621
```

```
boot_RD
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 2000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = boot_res, index = 1)
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 0.0704,  0.2113 )   ( 0.0688,  0.2088 )
##
## Level     Percentile            BCa
## 95%   ( 0.0691,  0.2091 )   ( 0.0720,  0.2129 )
## Calculations and Intervals on Original Scale
```

```
boot_RR
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 2000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = boot_res, index = 2)
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 1.137,  1.467 )   ( 1.125,  1.456 )
##
## Level     Percentile            BCa
## 95%   ( 1.144,  1.475 )   ( 1.151,  1.488 )
## Calculations and Intervals on Original Scale
```

```
boot_OR
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
```

```
## Based on 2000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = boot_res, index = 3)
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 1.226,  2.265 )   ( 1.149,  2.188 )
##
## Level     Percentile            BCa
## 95%   ( 1.320,  2.360 )   ( 1.333,  2.399 )
## Calculations and Intervals on Original Scale
```

While this marginal standardization approach is more flexible in that it accounts for the interaction between quitting smoking and exercise, and still yields an estimate of the average treatment effect (again, under identifiability), it still assumes a constant effect of qsmk on weight change across levels of all of the other variables in the model. This constant effect assumption might be true, but if one wanted to account for potential interactions between the exposure and all of the confounders in the model, there is an easy way. We call this the "stratified modeling approach."

This stratified modeling approach avoids the exposure effect homogeneity assumption across levels of all the confounders. In effect, the approach fits a separate model for each exposure stratum. To obtain predictions under the "exposed" scenario, we use the model fit to the exposed individuals to generate predicted outcomes in the entire sample. To obtain predictions under the "unexposed" scenario, we repeat the same procedure, but with the model fit among the unexposed. One can then average the risks obtained under each exposure scenario, and take their difference and ratio to obtain the risk differences and ratios of interest.

```
#' Marginal Standardization
##' To avoid assuming no interaction between
##' smoking and any of the other variables
##' in the model, we subset modeling among
##' exposed/unexposed. This code removes smoking from the model,
```

```
##' which will allow us to regress the outcome
##' against the confounders among the exposed and
##' the unexposed searately. Doing so will allow us
##' to account for any potential exposure-covariate interactions
##' that may be present.
formulaVars <- paste(names(nhefs)[c(7:16)],
    collapse = "+")
modelForm <- as.formula(paste0("wt_delta ~",
    formulaVars))
modelForm
```

```
## wt_delta ~ exercise + sex + age + race + income + marital + school +
##     asthma + bronch + alcoholfreq
```

```
#' Regress the outcome against the confounders
#' among the unexposed (model0) and then among the exposed (model1)
model0 <- glm(modelForm, data = subset(nhefs,
    qsmk == 0), family = binomial("logit"))
model1 <- glm(modelForm, data = subset(nhefs,
    qsmk == 1), family = binomial("logit"))
##' Generate predictions for everyone in the sample using the model fit to only the
##' unexposed (mu0 predictions) and only the exposed (mu1 predictions).
mu1 <- predict(model1, newdata = nhefs, type = "response")
mu0 <- predict(model0, newdata = nhefs, type = "response")

#' Marginally adjusted odds ratio
marg_stand_OR <- (mean(mu1)/mean(1 - mu1))/(mean(mu0)/mean(1 -
    mu0))
#' Marginally adjusted risk ratio
marg_stand_RR <- mean(mu1)/mean(mu0)
#' Marginally adjusted risk difference
marg_stand_RD <- mean(mu1) - mean(mu0)

#' Using the bootstrap to obtain confidence intervals for the marginally adjusted
#' risk ratio and risk difference.
```

```
bootfunc <- function(data, index) {
    boot_dat <- data[index, ]
    model0 <- glm(modelForm, data = subset(boot_dat,
        qsmk == 0), family = binomial("logit"))
    model1 <- glm(modelForm, data = subset(boot_dat,
        qsmk == 1), family = binomial("logit"))
    mu1 <- predict(model1, newdata = boot_dat,
        type = "response")
    mu0 <- predict(model0, newdata = boot_dat,
        type = "response")

    marg_stand_OR_ <- (mean(mu1)/mean(1 -
        mu1))/(mean(mu0)/mean(1 - mu0))
    marg_stand_RR_ <- mean(mu1)/mean(mu0)
    marg_stand_RD_ <- mean(mu1) - mean(mu0)
    res <- c(marg_stand_RD_, marg_stand_RR_,
        marg_stand_OR_)
    return(res)
}

#' Run the boot function. Set a seed to obtain reproducibility
set.seed(123)
boot_res <- boot(nhefs, bootfunc, R = 2000)

boot_RD <- boot.ci(boot_res, index = 1)
```

```
## Warning in boot.ci(boot_res, index = 1): bootstrap variances needed for
## studentized intervals
```

```
boot_RR <- boot.ci(boot_res, index = 2)
```

```
## Warning in boot.ci(boot_res, index = 2): bootstrap variances needed for
## studentized intervals
```

```
boot_OR <- boot.ci(boot_res, index = 2)
```

```
## Warning in boot.ci(boot_res, index = 2): bootstrap variances needed for
## studentized intervals
```

```
marg_stand_OR
```

```
## [1] 1.820838
```

```
marg_stand_RR
```

```
## [1] 1.318331
```

```
marg_stand_RD
```

```
## [1] 0.1478221
```

```
boot_RD
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 2000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = boot_res, index = 1)
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 0.0750,  0.2236 )   ( 0.0737,  0.2205 )
##
## Level     Percentile            BCa
## 95%   ( 0.0752,  0.2220 )   ( 0.0770,  0.2245 )
## Calculations and Intervals on Original Scale
```

```
boot_RR
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 2000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = boot_res, index = 2)
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 1.147,  1.493 )   ( 1.136,  1.482 )
##
## Level     Percentile            BCa
## 95%   ( 1.154,  1.501 )   ( 1.164,  1.507 )
## Calculations and Intervals on Original Scale
```

```
boot_OR
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 2000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = boot_res, index = 2)
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 1.147,  1.493 )   ( 1.136,  1.482 )
##
## Level     Percentile            BCa
## 95%   ( 1.154,  1.501 )   ( 1.164,  1.507 )
## Calculations and Intervals on Original Scale
```

When predicted risks are estimated using a logistic model, relying on marginal standardization will not result in probability estimates outside the bounds [0, 1]. And because the robust variance estimator is not required,

model-based standardization will not be as affected by small sample sizes. However, the bootstrap is more computationally demanding than alternative variance estimators, which may pose problems in larger datasets.

5 Introduction to Regression for Time-to-Event Data

In the last set of notes, we looked at how to use regression to estimate risk differences, risk ratios, and odds ratios, via both conditional adjustment and marginal standardization. However, while these methods can still be used when time-to-event data are available, one may want to pursue a different course. In this set of notes, we are going to look at regression methods for time-to-event outcomes. In particular, we will focus on the Cox proportional hazards regression model and pooled logistic regression. We will also discuss options when competing risks are present, focusing on how to compute confounder adjusted cumulative subdistribution risks.

Throughout, we will use the time-to-event dataset we introduced in Section 1 of the course[5]:

[5] Note that I updated the simulated dataset to contain 2000 observations instead of the original 100.

```
a <- read_csv(here("data", "2022_03_09-Section1_cohort.csv"))


print(a, n = 5)
```

```
## # A tibble: 2,000 x 6
##      ID  stop exposure confounder outcome start
##   <dbl> <dbl>    <dbl>      <dbl>   <dbl> <dbl>
## 1     1  5           0          1       0     0
## 2     2  5           1          0       0     0
## 3     3  2.09        0          0       1     0
## 4     4  5           1          1       0     0
## 5     5  2.08        0          1       1     0
## # ... with 1,995 more rows
```

Recall, in these data, the outcome had three levels:

```
a %>%
    count(outcome)
```

```
## # A tibble: 3 x 2
##   outcome     n
##     <dbl> <int>
## 1       0   865
## 2       1   721
## 3       2   414
```

where `0` indicates no event, and event types `1` and `2` represent competing events. Note again that these data are considered "time-to-event" precisely because we have measured the event times measured, represented by the variable `stop`.

6 Cox Proportional Hazards Regression

Perhaps the first method that comes to mind when considering time-to-event analyses is the Cox proportional hazards (PH) model used to compute the hazard ratio. This is arguably one of the most common techniques used to evaluate time-to-event data.

The Cox model is a technique that regresses the *hazard* against a set of covariates. The hazard is a summary outcome measure like the odds, risk, or rate. It is a function of time that captures the instantaneous rate of events at a given time t, with a range of $[0, \infty]$. It is often denoted $h(t)$ or $\lambda(t)$, and is defined as:

$$\lambda(t) = \lim_{\Delta t \to 0+} \frac{P(t \leq T \leq t + \Delta t \mid t < T)}{\Delta t}$$

This function quantifies the probability that the observed event time T is between some index event time t and it's increment $t + \Delta t$, conditional on the set of individuals who are still at risk at the index event time t, all as the increment Δt for the index event time t approaches zero from the right.

Clearly, this is one of the first problems with the Cox proportional hazards regression model: **the hazard is a highly unintuitive measure of occurrence.**

More specifically, when interest lies in the effect of some exposure X, one

can define the hazard as a function of the exposure as well:

$$\lambda(t \mid X = x) = \lim_{\Delta t \to 0+} \frac{P(t \leq T \leq t + \Delta t \mid t < T, X = x)}{\Delta t}$$

Now, it's quite unconventional to condition the hazard on an exposure this way. Often, one is more likely to write $\lambda(t \mid X = x)$ as $\lambda_x(t)$. But the problem with this is that one may mis-interpret this hazard as a **causal** quantity, defined via potential outcomes.[6] However, if we wanted to target the causal hazard, we have to be able to make the necessary causal identifiability assumptions.[7] Then, one can formulate the hazard in terms of potential outcomes as:

[6] Recall, subscript notation is often used to denote potential outcomes.
[7] Counterfactual consistency, no interference, exchangeabilty, positivity.

$$\lambda^x(t) = \lim_{\Delta t \to 0+} \frac{P(t \leq T^x \leq t + \Delta t \mid t < T^x)}{\Delta t}$$

where T^x represents the outcome that would be observed if the exposure X was set to some value x. If X is a binary exposure $X \in [0, 1]$,[8] then we can then define the associational hazard ratio as $\lambda(t \mid X = 1)/\lambda(t \mid X = 0)$ or the causal hazard ratio as $\lambda^{x=1}(t)/\lambda^{x=0}(t)$.

[8] Note that we use a binary exposure for simplification. This can easily be generalized to multicategory X or continuous X.

Again, for a binary exposure $X \in [0, 1]$, the Cox PH model can be motivated as follows:

$$\begin{aligned} HR &= {}^{\lambda_1(t)}/_{\lambda_0(t)} \\ \exp(\beta) &= {}^{\lambda_1(t)}/_{\lambda_0(t)} \\ \implies \lambda_1(t) &= \lambda_0(t)\exp(\beta X) \qquad \text{(Cox Model)} \end{aligned}$$

Concept Question:

Should the additive portion of the Cox model contain an intercept? Why or why not?

In effect, the Cox model defines the hazard for those with $X = 1$ as a baseline hazard $\lambda_0(t)$ multiplied by some exposure contribution $\exp(\beta X)$. In other words, the hazard for those with $X = 1$ is a multiple of the hazard for those with $X = 0$.

Why does this model imply proportional hazards? Note that the second line of the motivating equations above implies that the ratio of two hazards is $\exp(\beta)$, which does not depend on time. In other words, the ratio of two hazards cannot change over time, thus requiring constant proportion across

time. An example of this with a fairly unrealistic baselien hazard is shown in Figure 3.

The proportional hazards aspect of the model is often noted as an important limitation of the approach, but nonproportional hazards can easily be accommodated. Proportional hazards would be violated if, for example, the multiplicative effect of a particular variable on the baseline hazard changed over time, leading the hazards to converge or diverge (nonproportionality).

Several tests exist for evaluating proportional hazards (e.g., Schoenfeld residual plot). One easy approach is to evaluate whether there are important interactions between time and the covariates. If so, leaving these interactions in the model can be one technique to account for the nonproportional nature of the hazards.[9]

One property of the Cox model that is often raised as a strength of the approach is that it is *semiparametric* in that it does not assume a specific form for the baseline hazard (see Technical Note).

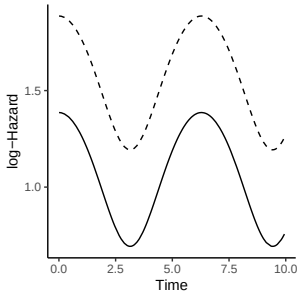


Figure 3: Illustration of the proportional hazards assumption, where the solid line represents the baseline hazard, and the dashed line represents the increase in the baseline due to some covariate.

[9] If there is an interaction between time and the exposure of interest, then one has to interpret the association between the exposure and the outcome as a function of this interaction.

Technical Note:

Technically, a parametric model assumes some finite set of parameters θ. Given the parameters, predictions from the model $\hat{y}$ are independent of the observed data $\mathcal{O}$:

$$P(\hat{y} \mid \theta, \mathcal{O}) = P(\hat{y} \mid \theta).$$

That is, θ captures everything there is to know about the data. With knowledge of the parameters, we no longer need the data to obtain information about the predictions.

Because the set of parameters are finite-dimensional, the complexity of the model is bounded, even in (asymptotic, theoretical) settings where we have infinite data.

In contrast, a nonparametric model assumes θ is *infinite dimensional*. In a nonparametric model, θ is often considered a function.

Because the set of parameters are infinite-dimensional, the complexity of the model depends on the complexity of the data. Thus, even in (asymptotic, theoretical) settings where we have infinite data, we can accommodate complexity.

Finally, a semiparametric model is one that contains both finite-dimensional and infinite-dimensional parameters. The classic example is, in fact, the Cox PH model, but many other semiparametric models exist. Consider a Cox model with two covariates X_1 and X_2:

$$\lambda(t) = \lambda_0(t) \times \exp(\beta_1 X_1 + \beta_2 X_2)$$

In this model, the baseline hazard $\lambda_0(t)$ is infinite-dimensional. As a result, it can accommodate any "shape" the data may require. On the other hand, the coefficient terms $\exp(\beta_1 X_1 + \beta_2 X_2)$ are parametric. As a result, the increase in the baseline hazard due to X_1 and X_2 are encoded in this model as multiplicative. In other words, this model rules out a potential additive effect of X_1 and X_2 on the baseline hazard, for instance:

$$\lambda(t) = \lambda_0(t) + \alpha_1 X_1 + \alpha_2 X_2$$

which would not be the case if the covariate effects were specified nonparametrically.

While its semiparametric feature is indeed a strength, it does not (IMO) overcome the serious limitations inherent in the Cox model, particularly when it comes to quantifying causal effects.

One important question with the Cox model is how one can quantify model parameters, particularly with a nonparametric baseline hazard? This was one of Sir David Cox's brilliant contributions: the *partial likelihood function* (Cox, 1972, Cox (1975)). He showed using the ordered event times that the full

likelihood could be re-written without incorporating the baseline hazard. For example, for K ordered event times, the partial likelihood can be written as:

$$\mathcal{L}(\beta) = \prod_{j=1}^{K} \frac{\lambda_0(T_j)\exp(\beta X_j)}{\sum_{i \in \mathcal{R}(T_j)} \lambda_0(T_j)\exp(\beta X_i)} = \prod_{j=1}^{K} \frac{\exp(\beta X_j)}{\sum_{i \in \mathcal{R}(T_j)} \exp(\beta X_i)}$$

where the index $i \in \mathcal{R}(T_j)$ refers to all individuals **at risk of the event at time** T_j. This is often referred to as the "risk-set". This has important implications for interpreting the hazard ratio, as we will see later. Moreover, because this expression requires *ordering* the event times, the presence of tied event times requires that the likelihood be modified. Many methods exist for handling ties, with Efron's method being the preferred choice in most settings (Hertz-Picciotto and Rockhill, 1997, Efron (1977)).

The Cox PH model can easily be fit in R. To do this, we'll evaluate the hazard that the outcome value is 1:

```
library(survival)

a <- a %>%
    mutate(outcome_one = as.numeric(outcome ==
        1))

cox_model <- coxph(Surv(time = start, time2 = stop,
    event = outcome_one) ~ exposure + confounder,
    data = a, ties = "efron")

summary(cox_model)
```

```
## Call:
## coxph(formula = Surv(time = start, time2 = stop, event = outcome_one) ~
##     exposure + confounder, data = a, ties = "efron")
##
##   n= 2000, number of events= 721
##
##                coef exp(coef) se(coef)      z Pr(>|z|)
## exposure   0.87140   2.39026  0.07692 11.329   <2e-16 ***
```

```
## confounder 0.11306   1.11970  0.07771  1.455    0.146
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##            exp(coef) exp(-coef) lower .95 upper .95
## exposure        2.39     0.4184    2.0558     2.779
## confounder      1.12     0.8931    0.9615     1.304
##
## Concordance= 0.609  (se = 0.01 )
## Likelihood ratio test= 140  on 2 df,   p=<2e-16
## Wald test            = 146.9  on 2 df,   p=<2e-16
## Score (logrank) test = 156.8  on 2 df,   p=<2e-16
```

```
a
```

```
## # A tibble: 2,000 x 7
##       ID  stop exposure confounder outcome start outcome_one
##    <dbl> <dbl>    <dbl>      <dbl>   <dbl> <dbl>       <dbl>
##  1     1  5           0          1       0     0           0
##  2     2  5           1          0       0     0           0
##  3     3  2.09        0          0       1     0           1
##  4     4  5           1          1       0     0           0
##  5     5  2.08        0          1       1     0           1
##  6     6  4.16        1          0       1     0           1
##  7     7  2.54        0          1       2     0           0
##  8     8  3.45        0          0       2     0           0
##  9     9  5           0          0       0     0           0
## 10    10  5           0          0       0     0           0
## # ... with 1,990 more rows
```

In this particular case, we estimate an adjusted hazard ratio for the association between the exposure and the time-to-event outcome of 2.39 with 95% CIs of 2.06, 2.78.

6.1 The Problem with Cox Regression

Over a decade ago, Hernán (2010) identified important problems with the hazard ratio as a causal estimand. These were:

1) The HR can change over time. That is, the HR early on during follow-up may be different from the HR at the end of follow up. However, researchers usually report a single HR, which represents an average over the entire follow-up period.

2) The HR has a built-in selection bias. This results from the fact that the hazard is a function that conditions on surviving past time t (see also denominator of partial likelihood). So in a study with a long follow-up period, the HR may be biased by the fact that it is dominated by the presence of survivors, suggesting no exposure effect, even when the exposure is harmful early on in follow-up.

Because of these two problems, it is important to consider other estimands for quantifying exposure effects with time-to-event outcomes.

7 Pooled Logistic Regression

Pooled logistic regression is a second approach that can be used to handle time-to-event outcomes (DAGOSTINO et al., 1990). This approach is an alternative technique that can be used to approximate the hazard ratio, particularly when the outcome is rare. But it can be used for other purposes that we will see (e.g., CDF estimation) that does not require rare outcomes.

Pooled logistic regression can be implemented by fitting a standard logistic regression model for the outcome, conditional on the exposure and relevant confounders. The key condition required to fit a pooled logistic model is the way the data are constructed. So far, our dataset `a` has been analysed in the "single row per observation" format. For the pooled logistic model, the data must be transformed so that each row represents a single time interval. The choice of time interval is important, particularly if one wants to use the pooled logistic model to approximate a hazard ratio. In this case, the interval should be chosen such that the proportion of events in each interval is no more than approximately 10%.

Let's look at how we can transform the section 1 data into the single row per time-interval.

```
# first, we create a 'stop_rounded'
# variable, which generates an integer
# count for the time on study.
a <- a %>%
    mutate(stop_rounded = ceiling(stop))  # why ceiling?


a
```

```
## # A tibble: 2,000 x 8
##       ID  stop exposure confounder outcome start outcome_one stop_rounded
##    <dbl> <dbl>    <dbl>      <dbl>   <dbl> <dbl>       <dbl>         <dbl>
##  1     1  5           0          1       0     0           0             5
##  2     2  5           1          0       0     0           0             5
##  3     3  2.09        0          0       1     0           1             3
##  4     4  5           1          1       0     0           0             5
##  5     5  2.08        0          1       1     0           1             3
##  6     6  4.16        1          0       1     0           1             5
##  7     7  2.54        0          1       2     0           0             3
##  8     8  3.45        0          0       2     0           0             4
##  9     9  5           0          0       0     0           0             5
## 10    10  5           0          0       0     0           0             5
## # ... with 1,990 more rows
```

```
# this shows how the unrounded
# time-to-event variable relates to the
# rounded variable
a %>%
    group_by(stop_rounded) %>%
    summarise(minStop = min(stop), medianStop = median(stop),
        maxStop = max(stop))
```

```
## # A tibble: 5 x 4
##   stop_rounded minStop medianStop maxStop
```

```
##          <dbl>   <dbl>      <dbl>   <dbl>
## 1            1  0.0118      0.446   0.989
## 2            2  1.00        1.44    2.00
## 3            3  2.00        2.44    3.00
## 4            4  3.00        3.45    3.99
## 5            5  4.00        5       5
```

Once the integer time interval variable is created, we can start creating the dataset that contains a single row per time interval:

```
# create the `b` dataset, which has multiple rows per person,
# using the "uncount" function

b <- a %>%
 uncount(stop_rounded) %>% # this row expands the dataset
                           # based on the count in stop_rounded
 group_by(ID) %>%
 mutate(counter = 1,     # create a counter that can be used to sum by ID
        time_var = cumsum(counter), # sum the counter by ID:
                                    # this becomes the new "time" variable
        last_id = !duplicated(ID, fromLast = T), # flag the last row for each person
        outcome_one = outcome_one*last_id, # set the new outcome
                                           # variable to 1 if last
                                           # AND old outcome is 1
        outcome_two = outcome*last_id) %>% # for competing risks
 ungroup(ID) %>%
 select(ID,time_var,stop,exposure,confounder,outcome,outcome_one,outcome_two)
b %>% filter(ID %in% c(1,3,7))
```

```
## # A tibble: 11 x 8
##       ID time_var  stop exposure confounder outcome outcome_one outcome_two
##    <dbl>    <dbl> <dbl>    <dbl>      <dbl>   <dbl>       <dbl>       <dbl>
##  1     1        1  5           0          1       0           0           0
##  2     1        2  5           0          1       0           0           0
##  3     1        3  5           0          1       0           0           0
##  4     1        4  5           0          1       0           0           0
```

```
##  5     1        5  5          0          1        0          0          0
##  6     3        1  2.09       0          0        1          0          0
##  7     3        2  2.09       0          0        1          0          0
##  8     3        3  2.09       0          0        1          1          1
##  9     7        1  2.54       0          1        2          0          0
## 10     7        2  2.54       0          1        2          0          0
## 11     7        3  2.54       0          1        2          0          2
```

With this dataset, we can now proceed with a pooled logistic regression model analysis. The difference between a "pooled" and standard logistic regression model is that the pooled model includes a flexible function of the time-to-event variable as a conditioning argument, in a dataset arranged in the person-time format. The pooled logistic model is often surreptitiously[10] written as:

[10] adverb; in a way that attempts to avoid notice or attention.

$$\text{logit}\{P(Y = 1 \mid T, X, C)\} = \beta_{0t} + \beta_1 X + \beta_2 C$$

where β_{0t} is a time-dependent intercept. In this model, and under the right circumstances (specifically, when the probability of the outcome in any given time interval is less than roughly 10%), one can interpret an estimate of β_1 as an approximation of the hazard ratio from the Cox PH regression model (DAGOSTINO et al., 1990). We can check this in our new dataset easily:

```
b %>%
    group_by(time_var) %>%
    summarise(meanOutcome = mean(outcome_one))
```

```
## # A tibble: 5 x 2
##   time_var meanOutcome
##      <dbl>       <dbl>
## 1        1      0.092
## 2        2      0.103
## 3        3      0.0893
## 4        4      0.0926
## 5        5      0.115
```

In practice, this time-dependent intercept can be obtained by including an

indicator term for each time-to-event in the model. For example, in R, this can be accomplished using the `factor` function:

```
plr_model <- glm(outcome_one ~ factor(time_var) +
    exposure + confounder, data = b, family = binomial(link = "logit"))

round(summary(plr_model)$coefficients["exposure",
    ], 2)
```

```
##   Estimate Std. Error    z value   Pr(>|z|)
##       0.96       0.08      11.70       0.00
```

Let's compare the estimated coefficient from this model to the hazard ratio above. Recall that the Cox model yielded a HR of 2.39 with 95% CIs of 2.06, 2.78. The pooled logistic model yields an approximated HR of 2.62 with 95% confidence intervals of 2.23 and 3.08.

Sometimes, the number of time-intervals created is too large to include in the model as a conditioning statement. For example, in a study with 60 weeks of follow-up, including weekly time-intervals with the factor function in the model would result in a model with at least 60 parameters. To deal with this, researchers will often smooth the time-intervals using polynomials or splines.

```
library(splines)

plr_model_linear <- glm(outcome_one ~ time_var +
    exposure + confounder, data = b, family = binomial(link = "logit"))

plr_model_squared <- glm(outcome_one ~ time_var +
    I(time_var^2) + exposure + confounder,
    data = b, family = binomial(link = "logit"))

plr_model_cubed <- glm(outcome_one ~ time_var +
    I(time_var^2) + I(time_var^3) + exposure +
    confounder, data = b, family = binomial(link = "logit"))

plr_model_spline <- glm(outcome_one ~ bs(time_var,
```

```
    df = 4) + exposure + confounder, data = b,
    family = binomial(link = "logit"))

round(summary(plr_model_linear)$coefficients["exposure",
    c("Estimate", "Std. Error")], 2)
```

```
##   Estimate Std. Error
##       0.96       0.08
```

```
round(summary(plr_model_squared)$coefficients["exposure",
    c("Estimate", "Std. Error")], 2)
```

```
##   Estimate Std. Error
##       0.96       0.08
```

```
round(summary(plr_model_cubed)$coefficients["exposure",
    c("Estimate", "Std. Error")], 2)
```

```
##   Estimate Std. Error
##       0.96       0.08
```

```
round(summary(plr_model_spline)$coefficients["exposure",
    c("Estimate", "Std. Error")], 2)
```

```
##   Estimate Std. Error
##       0.96       0.08
```

In these data, the functional form of the time variable does not matter.[11] This is a common usage of the pooled logistic regression model, particularly when (as we will see) the exposure and confounders are **time-varying.** One way to read the results of the pooled logistic regression output is to read the coefficient for the exposure in the model. As noted, when the outcome is rare within each unique time-interval specified in the model (or across the range of the continuous function for time in the model), this coefficient can be interpreted as an approximation to the hazard ratio.

[11] This is to be expected, because we simulated these time-to-events from an exponential distribution. This means that the simple linear term in `plr_model_linear` is actually compatible with the data generating mechanism. Additional flexibility is not required.

This is a second procedure that can be used to quantify a hazard ratio. However, one still has all the same problems with interpreting the hazard ratio. Fortunately, we can do a bit more with this model than we can with the Cox model. In particular, we can generate time-specific predicted probabilities for the outcome for each individual in the sample under exposed and unexposed states[12]:

[12] Technically, we *can* do this with the Cox model. We just need to use an estimator for the baseline hazard (which is not quantified in the Cox model), such as the Breslow estimator.

```
plr_model_spline <- glm(outcome_one ~ bs(time_var,
    df = 4) + exposure + confounder, data = b,
    family = binomial(link = "logit"))

summary(plr_model_spline)
```

```
##
## Call:
## glm(formula = outcome_one ~ bs(time_var, df = 4) + exposure +
##     confounder, family = binomial(link = "logit"), data = b)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -0.6893  -0.4417  -0.3865  -0.3611   2.3521
##
## Coefficients:
##                       Estimate Std. Error z value Pr(>|z|)
## (Intercept)           -2.69756    0.09061 -29.771   <2e-16 ***
## bs(time_var, df = 4)1  0.32591    0.38465   0.847   0.3968
## bs(time_var, df = 4)2 -0.10313    0.53658  -0.192   0.8476
## bs(time_var, df = 4)3 -0.06673    0.35062  -0.190   0.8490
## bs(time_var, df = 4)4  0.30522    0.12573   2.428   0.0152 *
## exposure               0.96225    0.08228  11.695   <2e-16 ***
## confounder             0.11403    0.08299   1.374   0.1694
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
```

```
##     Null deviance: 4726.6  on 7396  degrees of freedom
## Residual deviance: 4570.8  on 7390  degrees of freedom
## AIC: 4584.8
##
## Number of Fisher Scoring iterations: 5
```

```
# create three datasets, one that
# predicts under natural conditions and
# two that predict under exposure = [0,
# 1]
mu <- tibble(b, mu = predict(plr_model_spline,
    newdata = b, type = "response"))
mu1 <- tibble(b, mu1 = predict(plr_model_spline,
    newdata = transform(b, exposure = 1),
    type = "response"))
mu0 <- tibble(b, mu0 = predict(plr_model_spline,
    newdata = transform(b, exposure = 0),
    type = "response"))

# average the predictions for each
# individual stratified by time
mu <- mu %>%
    group_by(time_var) %>%
    summarise(mean_mu = mean(mu))
mu1 <- mu1 %>%
    group_by(time_var) %>%
    summarise(mean_mu1 = mean(mu1))
mu0 <- mu0 %>%
    group_by(time_var) %>%
    summarise(mean_mu0 = mean(mu0))

# cumulatively sum the predictions over
# time to estimate cumulative risk
mu <- mu %>%
    mutate(cum_risk = cumsum(mean_mu))
```

```
mu1 <- mu1 %>%
    mutate(cum_risk = cumsum(mean_mu1))
mu0 <- mu0 %>%
    mutate(cum_risk = cumsum(mean_mu0))


mu
```

```
## # A tibble: 5 x 3
##   time_var mean_mu cum_risk
##      <dbl>   <dbl>    <dbl>
## 1        1  0.0920   0.0920
## 2        2  0.103    0.195
## 3        3  0.0893   0.285
## 4        4  0.0926   0.377
## 5        5  0.115    0.493
```

```
mu1
```

```
## # A tibble: 5 x 3
##   time_var mean_mu1 cum_risk
##      <dbl>    <dbl>    <dbl>
## 1        1    0.155    0.155
## 2        2    0.175    0.330
## 3        3    0.155    0.485
## 4        4    0.161    0.646
## 5        5    0.199    0.845
```

```
mu0
```

```
## # A tibble: 5 x 3
##   time_var mean_mu0 cum_risk
##      <dbl>    <dbl>    <dbl>
## 1        1   0.0657   0.0657
## 2        2   0.0748   0.140
## 3        3   0.0653   0.206
```

```
## 4         4   0.0684   0.274
## 5         5   0.0869   0.361
```

In effect, this procedure is marginal standardization. Except, rather than only estimating the risk at the end of follow-up under exposed and unexposed states, we are quantifying the cumulative risk function over follow-up time. Note also, this cumulative risk function is marginally adjusted for the confounder. Thus, here we have a way to compute marginally standardized risk functions using a pooled logistic model. This approach, however, will only work with a single outcome of interest.

8 Multinomial Logistic Regression: Competing Risks

In the situation where competing events are present, the pooled logistic regression approach can be extended by replacing the binomial distribution with the multinomial distribution. The multinomial distribution is a generalization of the binomial distribution to scenarios where the event of interest can take on more than one value. One can define a simple multinomial logistic regression model as:

$$\text{logit}[P(Y = k \mid X, C)] = \beta_{0k} + \beta_{1k}X + \beta_{2k}C$$

for $k = 1, \ldots, K$. In this simple model for a binary X variable, $\exp(\beta_1)$ can be interpreted as the following odds ratio:

$$\frac{P(Y = k \mid X = 1, C)}{P(Y = k' \mid X = 1, C)} \Big/ \frac{P(Y = k \mid X = 0, C)}{P(Y = k' \mid X = 0, C)}$$

where $Y = k'$ refers to some referent outcome level.

For example, in our example data, the outcome takes on three levels: $Y \in [0, 1, 2]$. Thus, if we take $Y = 0$ as our referent level, we obtain two odds ratios from this model:

$$\frac{P(Y = 2 \mid X = 1, C)}{P(Y = 0 \mid X = 1, C)} \Big/ \frac{P(Y = 2 \mid X = 0, C)}{P(Y = 0 \mid X = 0, C)}$$

and

$$\frac{P(Y = 1 \mid X = 1, C)}{P(Y = 0 \mid X = 1, C)} \Big/ \frac{P(Y = 1 \mid X = 0, C)}{P(Y = 0 \mid X = 0, C)}$$

This multinomial model can also be adapted to the time-to-event setting in the same way we adapted the logistic regression model. In effect, we can fit a pooled multinomial model as:

$$\text{logit}[P(Y = k \mid X, C)] = \beta_{0tk} + \beta_{1k}X + \beta_{2k}C$$

which effectively makes each outcome-specific intercept a function of time.

In R, their are several packages that can be used to fit a multinomial model, including the `multinom` function in the `nnet`, and the `vglm` function in the `VGAM` package. We'll use the latter here.

Let's first look at the data we'll be fitting:

```
b %>%
    filter(ID %in% c(1, 3, 7))
```

```
## # A tibble: 11 x 8
##       ID time_var  stop exposure confounder outcome outcome_one outcome_two
##    <dbl>    <dbl> <dbl>    <dbl>      <dbl>   <dbl>       <dbl>       <dbl>
##  1     1        1  5           0          1       0           0           0
##  2     1        2  5           0          1       0           0           0
##  3     1        3  5           0          1       0           0           0
##  4     1        4  5           0          1       0           0           0
##  5     1        5  5           0          1       0           0           0
##  6     3        1  2.09        0          0       1           0           0
##  7     3        2  2.09        0          0       1           0           0
##  8     3        3  2.09        0          0       1           1           1
##  9     7        1  2.54        0          1       2           0           0
## 10     7        2  2.54        0          1       2           0           0
## 11     7        3  2.54        0          1       2           0           2
```

Note that the `outcome_two` variable is indeed a three-level variable:

```
b %>%
    count(outcome_two)
```

```
## # A tibble: 3 x 2
##   outcome_two     n
```

```
##         <dbl> <int>
## 1           0  6262
## 2           1   721
## 3           2   414
```

To implement the pooled multinomial model for competing risks, we can use the `vglm` function. To allow for flexibility along the time dimension, we'll use splines:

```
library(VGAM)
```

```
## Loading required package: stats4
```

```
##
## Attaching package: 'VGAM'
```

```
## The following objects are masked from 'package:boot':
##
##     logit, simplex
```

```
## The following object is masked from 'package:tidyr':
##
##     fill
```

```
pmr1 <- vglm(outcome_two ~ time_var + exposure +
    confounder, data = b, family = multinomial(refLevel = 1))  ## note the need for refLevel


summary(pmr1)@coef3[c("exposure:1", "exposure:2"),
    ]
```

```
##              Estimate Std. Error   z value     Pr(>|z|)
## exposure:1  0.8990578 0.08243999 10.905603 1.083725e-27
## exposure:2 -1.5438587 0.19359563 -7.974657 1.528045e-15
```

However, once again, the coefficients in this model will be challenging to interpret. We can implement the same procedure with the binomial model to obtain adjusted sub-distribution risk curves. To do this, it's useful to see what we obtain when we apply the predict function to an object resulting from the fit of the `vglm` function:

```
head(predict(pmr1, newdata = b, type = "response"))
```

```
##           0          1          2
## 1 0.8504551 0.07035747 0.07918746
## 2 0.8487853 0.07355806 0.07765667
## 3 0.8469676 0.07689052 0.07614188
## 4 0.8449980 0.08035913 0.07464287
## 5 0.8428724 0.08396821 0.07315943
## 6 0.8347971 0.15047995 0.01472296
```

The output from the predict function with a multinomial model (fitted with `vglm`) is a K column dataset, where K is the number of outcome levels. We thus have to adjust our code to ensure we select the correct probabilites when creating standardized risk curves:

```
pmr1 <- vglm(outcome_two ~ scale(time_var) +
    exposure + confounder, data = b, family = multinomial(refLevel = 1))

# create three datasets, one that
# predicts under natural conditions and
# two that predict under exposure = [0,
# 1] FOR OUTCOME = 1
mu_1 <- tibble(b, mu_1 = predict(pmr1, newdata = b,
    type = "response")[, 2])
mu_11 <- tibble(b, mu_11 = predict(pmr1,
    newdata = transform(b, exposure = 1),
    type = "response")[, 2])
mu_10 <- tibble(b, mu_10 = predict(pmr1,
    newdata = transform(b, exposure = 0),
    type = "response")[, 2])
## FOR OUTCOME = 2
mu_2 <- tibble(b, mu_2 = predict(pmr1, newdata = b,
    type = "response")[, 3])
mu_21 <- tibble(b, mu_21 = predict(pmr1,
    newdata = transform(b, exposure = 1),
```

```
    type = "response")[, 3])
mu_20 <- tibble(b, mu_20 = predict(pmr1,
    newdata = transform(b, exposure = 0),
    type = "response")[, 3])

# average the predictions for each
# individual stratified by time FOR
# OUTCOME = 1
mu_1 <- mu_1 %>%
    group_by(time_var) %>%
    summarise(mean_mu_1 = mean(mu_1))
mu_11 <- mu_11 %>%
    group_by(time_var) %>%
    summarise(mean_mu_11 = mean(mu_11))
mu_10 <- mu_10 %>%
    group_by(time_var) %>%
    summarise(mean_mu_10 = mean(mu_10))
## FOR OUTCOME = 2
mu_2 <- mu_2 %>%
    group_by(time_var) %>%
    summarise(mean_mu_2 = mean(mu_2))
mu_21 <- mu_21 %>%
    group_by(time_var) %>%
    summarise(mean_mu_21 = mean(mu_21))
mu_20 <- mu_20 %>%
    group_by(time_var) %>%
    summarise(mean_mu_20 = mean(mu_20))

# cumulatively sum the predictions over
# time to estimate cumulative risk FOR
# OUTCOME = 1
mu_1 <- mu_1 %>%
    mutate(cum_risk = cumsum(mean_mu_1))
mu_11 <- mu_11 %>%
```

```
    mutate(cum_risk = cumsum(mean_mu_11))
mu_10 <- mu_10 %>%
    mutate(cum_risk = cumsum(mean_mu_10))
## FOR OUTCOME = 2
mu_2 <- mu_2 %>%
    mutate(cum_risk = cumsum(mean_mu_2))
mu_21 <- mu_21 %>%
    mutate(cum_risk = cumsum(mean_mu_21))
mu_20 <- mu_20 %>%
    mutate(cum_risk = cumsum(mean_mu_20))


mu_1
```

```
## # A tibble: 5 x 3
##   time_var mean_mu_1 cum_risk
##      <dbl>     <dbl>    <dbl>
## 1        1    0.0925   0.0925
## 2        2    0.0957   0.188
## 3        3    0.0980   0.286
## 4        4    0.101    0.388
## 5        5    0.105    0.492
```

```
mu_11
```

```
## # A tibble: 5 x 3
##   time_var mean_mu_11 cum_risk
##      <dbl>      <dbl>    <dbl>
## 1        1      0.156    0.156
## 2        2      0.162    0.318
## 3        3      0.169    0.487
## 4        4      0.175    0.662
## 5        5      0.182    0.844
```

```
mu_10
```

```
## # A tibble: 5 x 3
##   time_var mean_mu_10 cum_risk
##      <dbl>      <dbl>    <dbl>
## 1        1     0.0660   0.0660
## 2        2     0.0689   0.135
## 3        3     0.0720   0.207
## 4        4     0.0752   0.282
## 5        5     0.0786   0.361
```

```
mu_2
```

```
## # A tibble: 5 x 3
##   time_var mean_mu_2 cum_risk
##      <dbl>     <dbl>    <dbl>
## 1        1    0.0569   0.0569
## 2        2    0.0562   0.113
## 3        3    0.0560   0.169
## 4        4    0.0553   0.224
## 5        5    0.0546   0.279
```

```
mu_21
```

```
## # A tibble: 5 x 3
##   time_var mean_mu_21 cum_risk
##      <dbl>      <dbl>    <dbl>
## 1        1     0.0153   0.0153
## 2        2     0.0149   0.0302
## 3        3     0.0145   0.0447
## 4        4     0.0142   0.0588
## 5        5     0.0138   0.0726
```

```
mu_20
```

```
## # A tibble: 5 x 3
##   time_var mean_mu_20 cum_risk
##      <dbl>      <dbl>    <dbl>
## 1        1     0.0742   0.0742
## 2        2     0.0728   0.147
## 3        3     0.0713   0.218
## 4        4     0.0699   0.288
## 5        5     0.0685   0.357
```

9 Introduction to Propensity Score Methods

So far in this course, we've focused on obtaining quantitative estimates of the average treatment effect using an outcome modeling approach. In this approach, one regresses the outcome against the exposure and confounders. This works for the conditionally adjusted estimator, where we read the coefficient for the exposure from the regression model, or the marginally adjusted estimator (e.g., using marginal standardization). The basic formulation of this outcome modeling approach can be depicted heuristically using the DAG in Figure 4:

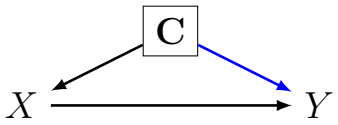


Figure 4: Directed acyclic graph depicting confounder adjustment using an outcome modelling approach. In this approach, information from the confounder to the outcome is 'blocked' (blue arrow). With this adjustment, the exposure X os d-separated from the outcome Y, rendering the estimate of the exposure-outcome association unconfounded.

In this Figure, the open back-door path from X to Y is blocked by conditioning on C. This leads to a conditionally adjusted estimate of the exposure effect. One can marginalize over the distribution on C to get a marginally adjusted estimate from a model for the outcome. For a binary outcome, such estimates may be obtained on the risk difference, risk ratio, or odds ratio scales, or one may obtain a marginally adjusted estimate of the cumulative risk

function that would be observed if everyone were exposed or unexposed.

On the other hand, we may want to obtain an adjusted estimate of the exposure effect by modeling the exposure. This can be accomplished using propensity score methods.

The propensity score was first defined by Rosenbaum and Rubin (Rosenbaum and Rubin, 1983) as the conditional probability of receiving the treatment (or, equivalently, of being exposed). The true propensity score is defined as

$$e(C) = P(X = 1 \mid C)$$

This propensity score is typically known in a randomized trial. It's important to distinguish this true PS from the estimated PS:

$$\hat{e}(C) = \hat{P}(X = 1 \mid C)$$

This estimated PS can be fit using a parametric model, in which case, we might write:

$$\hat{e}(C; \alpha) = \hat{P}(X = 1 \mid C) = \text{expit}(\alpha C)$$

For reasons that are not entirely obvious, it is usually better to use the estimated PS, even when the true PS is known (Robins et al., 1992, Henmi and Eguchi (2004)).

If the set of conditioning variables consists of the relevant confounding variables, the propensity score can be used to invoke conditional exchangeability. To see why, consider the case where the probability of being exposed is conditional on two binary confounders:

$$f(C) = P(X = 1 \mid C) = \text{expit}(\alpha_0 + \alpha_1 C_1 + \alpha_2 C_2 + \alpha_3 C_1 C_2)$$

Consider that, for two binary confounders, there are four possible joint confounder levels:

C1	C2
0	0
1	0
0	1

C1	C2
1	1

Notice also that there are four parameters in the above model, one parameter for each level. This implies that, for this simple example, there is a unique propensity score value for each unique confounder level. In other words, the one-dimensional propensity score in this example contains all the information available in the two-dimensional set of confounders. We can thus reduce the complexity of the set of confounders to a single variable, and adjust for this variable instead of the confounders. For example:

$$E[Y \mid X, p(X)] = \beta_0 + \beta_1 X + \beta_c f(C)$$

Heuristically (again), we can show why this approach works using the DAG in Figure 2

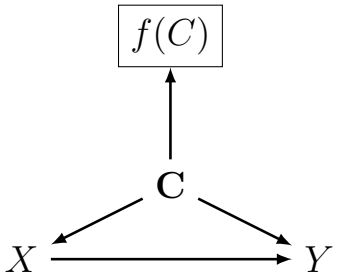


Figure 5: Directed acyclic graph depicting confounder adjustment using an outcome modelling approach. In this approach, information from the confounder to the outcome is 'blocked' (blue arrow). With this adjustment, the exposure X os d-separated from the outcome Y, rendering the estimate of the exposure-outcome association unconfounded.

In this DAG, $f(C)$ is a proxy for all the variables C. Thus, we can use $f(C)$ to replace C in an outcome regression model.

As a univariate proxy for the multivariate set of confounders, we can use the propensity score to adjust for confounding using a number of different techniques. These techniques include regression adjustment, propensity score matching, stratification, and weighting. In this lecture, we will quickly cover the first three and focus on the creation and use of IP-weights. The reason for this emphasis is that i in epidemiology, the most commonly used PS technique is inverse probability weighting, and so it is important to know how to implement; and ii, there are theoretical justifications suggesting that weighting is the most optimal use of the PS.

10 Adjustment, Matching, Stratification

To demonstrate these techniques, we will again use the Section 1 cohort data:

```
a <- read_csv(here("data", "2022_03_09-Section1_cohort.csv")) %>%
    mutate(outcome_one = as.numeric(outcome ==
        1))

print(a, n = 5)
```

```
## # A tibble: 2,000 x 7
##      ID  stop exposure confounder outcome start outcome_one
##   <dbl> <dbl>    <dbl>      <dbl>   <dbl> <dbl>       <dbl>
## 1     1  5           0          1       0     0           0
## 2     2  5           1          0       0     0           0
## 3     3  2.09        0          0       1     0           1
## 4     4  5           1          1       0     0           0
## 5     5  2.08        0          1       1     0           1
## # ... with 1,995 more rows
```

The first step in any PS analysis is to obtain an estimate of the propensity score. This can be done using an array of techniques, including nonparametric techniques such as machine learning methods (e.g., Lee et al., 2010) or a more general nonparametric approach (Hirano et al., 2003) (see Technical Note), but is most often accomplished using logistic regression.

Technical Note:

Using the propensity score to adjust for confounding is a technique that falls into the class of single robust estimation. Singly robust estimators rely on a single model to adjust for confounding (or missing data or selection bias). In this case, the single model is the propensity score model.

Many researchers have proposed or implemented single robust estimation methods using machine learning methods. Machine learning methods consist of a wide range of analytic techniques that do not require hard to verify modeling assumptions. Because of this, they are often assumed to be less biased than their standard parametric counterparts. This perceived property has motivated many to either recommended or use machine learning methods to quantify exposure effects (Lee et al., 2010, Westreich et al. (2010),Snowden et al. (2011),Oulhote et al. (2019)). These "machine learning'' methods include techniques like kernel regression, splines, random forests, boosting, etc., which exploit smoothness across covariate patterns to estimate the regression function.

However, for any nonparametric approach there is an explicit bias-variance trade-off that arises in the choice of tuning parameters; less smoothing yields smaller bias but larger variance, while more smoothing yields smaller variance but larger bias (parametric models can be viewed as an extreme form of smoothing). This tradeoff has important consequences.

Convergence rates for nonparametric estimators become slower with more flexibility and more covariates. For example, a standard rate for estimating smooth regression functions is $N^{-\beta/(2\beta+d)}$, where β represents the number of derivatives of the true regression function, and d represents the dimension of, or number of covariates in, the true regression function. This issue is known as the **curse of dimensionality** (Györfi et al., 2002, Robins and Ritov (1997),Wasserman (2006)). Sometimes this is viewed as a disadvantage of nonparametric methods; however, it is just the cost of making weaker assumptions: if a parametric model is misspecified, it will converge very quickly to the wrong answer.

In addition to slower convergence rates, confidence intervals are harder to obtain. Specifically, even in the rare case where one can derive asymptotic distributions for nonparametric estimators, it is typically not possible to construct confidence intervals (even via the bootstrap) without impractically undersmoothing the regression function (i.e., overfitting the data) (Wasserman, 2006).

These complications (slow rates and lack of valid confidence intervals) are generally inherited by singly robust estimators such as methods that use only the propensity score for adjustment (this is apart from a few special cases which require simple estimators, such as kernel methods with strong smoothness assumptions and careful tuning parameter choices that are suboptimal for estimating f or g).

For general nonparametric estimators of the exposure or outcome model, convergence will be slow, and honest confidence intervals will not be computable (note that "honest" confidence intervals are defined as CIs with a minimum coverage probability no less than the nominal value over a rich class of nonparametric regression functions).

For these reasons, it is generally not advisable to use machine learning or nonparametric methods to estimate the propensity score and then proceed with PS adjustment in a regression model, matching, stratification, or weighting (Naimi et al., 2022). Instead, one should implement double robust estimation methods when machine learning or nonparametric methods are used.

In a previous lecture, we discussed using different link functions and distributions to estimate a conditionally adjusted risk difference, risk ratio, or odds ratio. However, when using parametric regression to estimate the propensity score, one would typically (always?)[13] use logistic regression:

[13] Generally, you will always want to use a method that bounds the predicted probabilities between 0 and 1.

```
# create the propensity score in the
# dataset
a$propensity_score <- glm(exposure ~ confounder,
    data = a, family = binomial("logit"))$fitted.values
```

10.1 Propensity Score Adjustment

For regression adjustment, we can then simply adjust for this propensity score in an outcome regression model to obtain an adjusted estimate of the parameter of interest. This is one of the easiest techniques available for using the propensity score to adjust for confounders:

```
# adjust for the propensity score to
# obtain conditionally adjusted risk
# difference
model1 <- glm(outcome_one ~ exposure + propensity_score,
    data = a, family = binomial("identity"))

summary(model1)$coefficients
```

```
##                   Estimate Std. Error   z value     Pr(>|z|)
## (Intercept)      0.2370501 0.02991547  7.923997 2.299954e-15
## exposure         0.3142727 0.02427909 12.944169 2.534920e-38
## propensity_score 0.1134088 0.10147668  1.117585 2.637442e-01
```

In the above output, the point estimate can be interpreted as the risk difference, with valid 95% CIs (binomial distribution, identity link). In principle, nothing here prevents us from using a logistic link function, and then marginalizing over the distribution the way we do with marginal standardization and the confounders (though we must use the bootstrap for standard error assessment).

10.2 Propensity Score Stratification

For propensity score stratification, we can create quantiles of the propensity score, and quantify the exposure effect within each stratum of this categorized propensity score. One can then combine the stratum specific point estimates and standard errors using a simple weighted average (Lunceford and Davidian, 2004). In a simple setting with a single binary confounder, there will be only two unique propensity score values:

```
table(a$propensity_score)
```

```
##
## 0.210485133020448 0.427977839335174
##              1278               722
```

In this situation, we'd create two strata to implement PS stratification. Because there are only two PS levels, we can use this code to create the strata:

```
a$ps_strata <- factor(a$propensity_score,
    labels = c("1", "2"))

a %>%
    group_by(ps_strata) %>%
    count()
```

```
## # A tibble: 2 x 2
## # Groups:   ps_strata [2]
##   ps_strata     n
##   <fct>     <int>
## 1 1          1278
## 2 2           722
```

In more general settings with a propensity score that takes on many unique values (i.e., continuous between 0 and 1), we can use this code to construct and evaluate quantiles:

```
## Identity the quintiles of the PS
quants <- quantile(a$propensity_score, prob = seq(0,
    1, by = 1), na.rm = T)

print(quants)

a$ps_strata <- cut(a$propensity_score, breaks = quants,
    include.lowest = T)

a %>%
    group_by(ps_strata) %>%
    count()
```

Either way, once the PS strata are created, the next step is to estimate the stratum specific associations of interest. Here, we quantify the risk differences using the binomial distribution and identity link:

```
# Perform logistic regression within
# each stratum

stratified_ps <- a %>%
    group_by(ps_strata) %>%
    do(model = glm(outcome_one ~ exposure,
        data = ., family = binomial("identity")))

stratified_ps$model[[1]]$coefficients
```

```
## (Intercept)    exposure
##   0.2646184   0.2930024
```

```
stratified_ps$model[[2]]$coefficients
```

```
## (Intercept)    exposure
##   0.2760291   0.3388577
```

The stratum specific coefficients and standard errors can then be combined using standard equations:

$$\hat{\gamma} = \frac{1}{K}\sum_{k=1}^{K}\left\{\hat{\gamma}^{(i)}\right\}, \qquad SE(\hat{\gamma}) = \sqrt{\frac{1}{K^2}\sum_{k=1}^{K}\left\{SE(\hat{\gamma}^{(i)})^2\right\}}$$

Propensity Score Matching

Finally, one can match on the basis of the propensity score, using a range of different techniques. These include 1:1 or 1:M matching, matching with or without replacement, greedy versus optimal matching, and nearest neighbor versus caliper matching (Austin, 2011).

```
library(MatchIt)
library(optmatch)
```

```
## The optmatch package has an academic license. Enter relaxinfo() for more information.
```

```
ps_matching <- matchit(formula = exposure ~ confounder, data     = a,
                       method = "full", distance = "logit", # Distance defined by usual propensity scor
                       ratio = 1, # gives us 1:1 matching, which is the default
                       estimand = "ATE"
                       )


ps_matching
```

```
## A matchit object
##  - method: Optimal full matching
##  - distance: Propensity score
##              - estimated with logistic regression
##  - number of obs.: 2000 (original), 2000 (matched)
##  - target estimand: ATE
##  - covariates: confounder
```

Once we've identified the matched pairs, we can create a dataset that contains the information we need to match. The `matchit` function in `R`, as with some other matching approaches, constructs a set of weights to operationalize

the matched sets (Yoshida et al., 2017). The dataset will include a weights variable and subclass variable that provide the information needed to conduct the matched analysis:

```
matched_data <- match.data(ps_matching)

matched_data %>%
    select(ID, exposure, confounder, propensity_score,
        distance, weights, subclass) %>%
    print(n = 6)
```

```
## # A tibble: 2,000 x 7
##      ID exposure confounder propensity_score distance weights subclass
##   <dbl>    <dbl>      <dbl>            <dbl>    <dbl>   <dbl> <fct>
## 1     1        0          1            0.428    0.428   0.718 532
## 2     2        1          0            0.210    0.210   0.578 244
## 3     3        0          0            0.210    0.210   0.728 531
## 4     4        1          1            0.428    0.428   0.578 309
## 5     5        0          1            0.428    0.428   0.718 532
## 6     6        1          0            0.210    0.210   0.578 377
## # ... with 1,994 more rows
```

One can then use this dataset with a standard `glm` model, weighted to implement the matching:

```
matched_model <- glm(outcome_one ~ exposure,
    data = matched_data, weights = weights,
    family = binomial(link = "logit"))
```

```
## Warning in eval(family$initialize): non-integer #successes in a binomial glm!
```

```
summary(matched_model)$coefficients
```

```
##                Estimate Std. Error    z value     Pr(>|z|)
## (Intercept) -0.9900412 0.05966912 -16.592188 7.937571e-62
## exposure     0.4190147 0.10516861   3.984219 6.770251e-05
```

This model creates a warning referring to "non-integer #successes in a binomial glm". This warning is referring to the fact that, while each individual contributes either 0 or 1 event to the likelihood function, with the weights added, these "contributions" become non-integer values. For example, if an individual's weight is 0.718, they will contribute a total of 0.718 events to the overall analysis. This is not actually a problem when we use weights to conduct an analysis. It's just the consequence of using weights. Thus, the only real problem is the warning that we get (not the content of the warning itself).

One "solution" to this artificial problem[14] is to use the `quasibinomial` distribution instead of the `binomial` option.

[14] This "artificial" problem becomes real in the context of, e.g., a simulation study where the warning will prematurely interrupt the simulation. That is, sometimes a warning will interrupt a function that needs to be run in full. In this case, it's helpful to use the quasibinomial approach, which is identical to the binomial except no warning is thrown.

The coefficient from this model can be interpreted as an average treatment effect obtained via matching. The next step is to obtain a standard error for this parameter. This is where things get a little uncertain. There are generally two approaches used to obtain standard errors for a matching estimator: the robust (sandwich) variance estimator, and the bootstrap.

Here is some code to implement the robust variance estimator. Note the need for the "subclass" argument and the ID argument:

```
library(lmtest)
library(sandwich)

coeftest(matched_model, vcov. = vcovCL, cluster = ~subclass +
    ID)
```

```
##
## z test of coefficients:
##
##              Estimate Std. Error  z value Pr(>|z|)
## (Intercept) -0.990041   0.058157 -17.0237   <2e-16 ***
## exposure     0.419015   0.547134   0.7658   0.4438
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The problem here is that there is comparatively little evidence supporting the use of the robust (sandwich) variance estimator or the bootstrap for matching estimators. There is even some theoretical work suggesting that the

bootstrap is biased for a propensity score matching estimator (Abadie and Imbens, 2008).

11 Inverse Probability Weighting: Intuition

The most commonly employed propensity score adjustment technique is inverse probability weighting. The simple heuristic often used to describe the way IP-weighting works is that, when applied to data, they yield a "pseudo population" where there is no longer an effect of the confounder on the exposure. The causal structure of the variables in this new pseudo-population can be depicted in Figure 6:

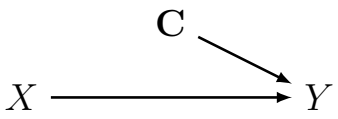


Figure 6: Directed acyclic graph depicting the causal relations between variables in a 'pseudo-population' obtained via inverse probability weighting. Again, with this adjustment, the exposure X os d-separated from the outcome Y, rendering the estimate of the exposure-outcome association unconfounded.

The weights for each individual needed to create this pseudo-population are defined as the inverse of the probability of receiving their observed exposure. Let's consider the following simple example to explain why this works. In this example, there are 20 observations, with one binary confounder (50:50 split) and one binary exposure. Let's suppose the probability that the exposure `x = 1` is 0.2 for those with `c = 0` and 0.9 for those with `c = 1`:

Under these scenarios, we'd have exposure and confounder data that looks like this:

```
c <- c(0, 0, 1, 1)
x <- c(1, 0, 1, 0)
n <- c(2, 8, 9, 1)

tibble(x, c, n)
```

```
## # A tibble: 4 x 3
##       x     c     n
```

```
##   <dbl> <dbl> <dbl>
## 1     1     0     2
## 2     0     0     8
## 3     1     1     9
## 4     0     1     1
```

Let's focus on the stratum of individuals with `c = 0`. In this stratum, there are 2 exposed individuals, and 8 unexposed individuals. Now let's ask what these data should look like if we were able to implement an ideal (marginally) randomized trial with the probability of treatment being 50% for all individuals. We would expect that within the stratum of individuals with `c = 0`, there would be 5 exposed and 5 unexposed individuals. Inverse probability weighting seeks to accomplish this balance.

Consider that the inverse probability of the **observed exposure** among those with `c = 0` is 0.2 for those who were actually exposed, and 0.8 for those who were unexposed.

The inverse probability weight is thus $\frac{1}{0.2} = 5$ for the exposed in this confounder stratum, and $\frac{1}{0.8} = 1.25$ for the unexposed in this confounder stratum.

This suggests that, in their contribution to the overall analysis, the two exposed individuals in the `c = 0` status would each receive a weight of 5, while the eight unexposed individuals in the `c = 0` stratum would each receive a weight of 1.25. Under these conditions, we would have a re-balanced set of observations in the `c = 0` stratum of:

$$\text{Exposed Observations:} \quad 5 \times 2 = 10$$

$$\text{Unexposed Observations:} \quad 8 \times 1.25 = 10$$

In effect, our inverse probability weighting strategy made it such that we now have an equal number of exposed and unexposed observations within the `c = 0` stratum. In these weighted data, we can now compute the difference in the outcome among the exposed and unexposed individuals (if we had it) to obtain an estimate of our average treatment effect.

12 Inverse Probability Weighting In Practice

Simply taking the inverse of the probability of the observed exposure, while valid, is not the usual strategy for implementing inverse probability weights. In practice, one will often use stabilized weights, stabilized normalized weights, potentially with some degree of "truncation" or, more accurately, trimming of the weights.[15]

Furthermore, it's important to note that "data" are often not weighted, but rather the contribution that each individual in the sample makes to the estimating function (e.g., likelihood, estimating equation, or other function used to find parameters). This is important in that one must choose a fitting algorithms that allows for this type of weighting.

To start, the simplest type of weight used in practice is the stabilzied inverse probability weight. These are often defined as:

$$sw = \begin{cases} \dfrac{P(X=1)}{P(X=1 \mid C)} & \text{if } X = 1 \\ \dfrac{P(X=0)}{P(X=0 \mid C)} & \text{if } X = 0 \end{cases}$$

but are sometimes written more succinctly as[16]:

$$sw = \frac{f(X)}{f(X \mid C)}$$

Let's use the cohort data again to construct the stabilized weights. We will re-fit the PS model to the data, and construct the weights:

```
# create the propensity score in the
# dataset
a$propensity_score <- glm(exposure ~ confounder,
    data = a, family = binomial("logit"))$fitted.values

# stabilized inverse probability
# weights
a$sw <- (mean(a$exposure)/a$propensity_score) *
    a$exposure + ((1 - mean(a$exposure))/(1 -
    a$propensity_score)) * (1 - a$exposure)
```

[15] In contrast to our emphasis of the usage of the word "truncation" which refers to the removal of observations from the dataset, researchers will often refer to "truncating" the weights, which sets the largest value to be equal to the 99th or 95th percentile values. This is more accurately referred to as "trimming" the weights, since no truncation is occurring.

[16] This formulation is unusual, since $f(.)$ represents the probability density function, which is usually taken for a specific realization of the random variable. However, in this case, because the weights are defined as a function of the observed exposure status, the argument in the operator is the observed data (denoted with capital letter), as opposed to some specific realization.

```
summary(a$sw)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.6753  0.9006  0.9006  1.0000  1.2430  1.3730
```

```
a %>%
    select(ID, exposure, confounder, outcome_one,
        propensity_score, sw) %>%
    print(n = 5)
```

```
## # A tibble: 2,000 x 6
##      ID exposure confounder outcome_one propensity_score    sw
##   <dbl>    <dbl>      <dbl>       <dbl>            <dbl> <dbl>
## 1     1        0          1           0            0.428 1.24
## 2     2        1          0           0            0.210 1.37
## 3     3        0          0           1            0.210 0.901
## 4     4        1          1           0            0.428 0.675
## 5     5        0          1           1            0.428 1.24
## # ... with 1,995 more rows
```

As we can see from the output above, the stabilized weights are, in fact, well behaved, with a mean of one and a max value that is small.

```
model_RD_weighted <- glm(outcome_one ~ exposure,
    data = a, weights = sw, family = quasibinomial("identity"))

summary(model_RD_weighted)$coefficients
```

```
##              Estimate Std. Error  t value      Pr(>|t|)
## (Intercept) 0.2687377 0.01176165 22.84863 7.551318e-103
## exposure    0.3095561 0.02367867 13.07321  1.551220e-37
```

At times, the mean and max of the stabilized weights are sub-optimal, in that the mean may not be one, or the max may be too large for comfort. One strategy we can use here is to normalize the weights, by dividing the stabilized weights by the max stabilized weight:

```
a$sw_norm <- a$sw/max(a$sw)

summary(a$sw_norm)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.4918  0.6559  0.6559  0.7283  0.9053  1.0000
```

```
a %>%
    select(ID, exposure, confounder, outcome_one,
        propensity_score, sw, sw_norm) %>%
    print(n = 5)
```

```
## # A tibble: 2,000 x 7
##      ID exposure confounder outcome_one propensity_score    sw sw_norm
##   <dbl>    <dbl>      <dbl>       <dbl>            <dbl> <dbl>   <dbl>
## 1     1        0          1           0            0.428 1.24    0.905
## 2     2        1          0           0            0.210 1.37    1
## 3     3        0          0           1            0.210 0.901   0.656
## 4     4        1          1           0            0.428 0.675   0.492
## 5     5        0          1           1            0.428 1.24    0.905
## # ... with 1,995 more rows
```

In this case, the mean of the normalized weights is no longer expected to be one. However, the max weight will be one by definition. These weights can then be used in the same way:

```
model_RD_weighted_norm <- glm(outcome_one ~
    exposure, data = a, weights = sw_norm,
    family = quasibinomial("identity"))

summary(model_RD_weighted_norm)$coefficients
```

```
##              Estimate Std. Error  t value     Pr(>|t|)
## (Intercept) 0.2687377 0.01176165 22.84863 7.551318e-103
## exposure    0.3095561 0.02367867 13.07321  1.551220e-37
```

Finally, instead of normalizing, sometimes researchers will "trim" the weights to avoid problems induced by very large weights. The procedure for doing this is straightfoward, and requires simply replacing all weight values greater than a certain percentile with their percentile values:

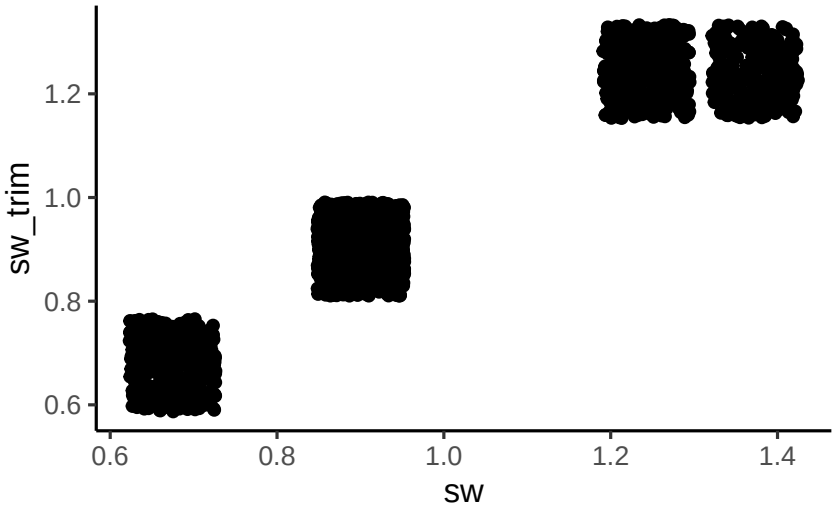
```
quantile(a$sw, 0.8)
```

```
##      80%
## 1.242959
```

```
a <- a %>%
    mutate(sw_trim = if_else(sw > quantile(sw,
        0.8), quantile(sw, 0.8), sw))
```

We can see how this changes the values in the following plot:

```
ggplot(a) + geom_jitter(aes(sw, sw_trim))
```



In this example analysis, we use the 80th percentile of the distribution of the stabilized weights to trim. In more typical settings, we would use the 99th, 95th, or 90th percentiles. This is importance, since trimming the weights like this induces a degree of potential bias in the estimator. In effect, it is a bias-variance tradeoff being made (Cole and Hernán, 2008).

A final note that is important with these weighted approaches is to consider how to estimate the standard errors. In fact, the model-based standard errors

are no longer valid when weighting is used. One must instead use the robust variance estimators, or the bootstrap. For example:

```
coeftest(model_RD_weighted_norm, vcov. = vcovHC)
```

```
##
## z test of coefficients:
##
##             Estimate Std. Error z value  Pr(>|z|)
## (Intercept) 0.268738   0.011918  22.549 < 2.2e-16 ***
## exposure    0.309556   0.024892  12.436 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

One can then construct CIs in the standard way using the estimated standard error in the output above.

13 Right Hand Side

In this set of notes, we will cover some considerations and techniques in specifying the right hand side of a regression model. This is sometimes referred to as "coding" the variables in the model, or "feature engineering" (in the machine learning literature). To illustrate some of the issues, we will use the NHEFS data where we seek to estimate the association between sex and smoke intensity adjusted for age and marital status:

```
nhefs <- read_csv(here("data", "nhefs_data.csv"))
```

Smoking intensity is distributed as follows:

The age variable is distributed as follows:

Additionally, marital status and sex are distributed as:

```
table(nhefs$marital)
```

```
##
##    2    3    4    5    6    8
## 1583  102  130  126   58    1
```

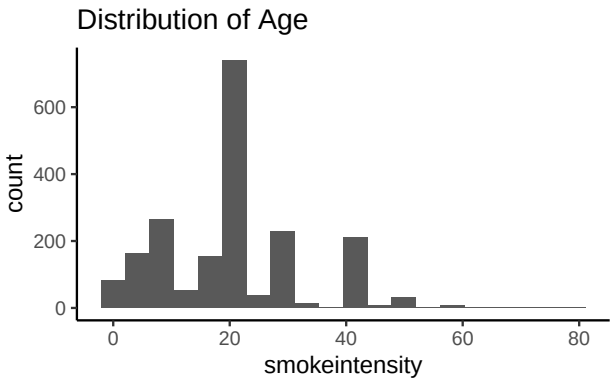


Figure 7: Distribution of smoking intensity in the NHEFS data.

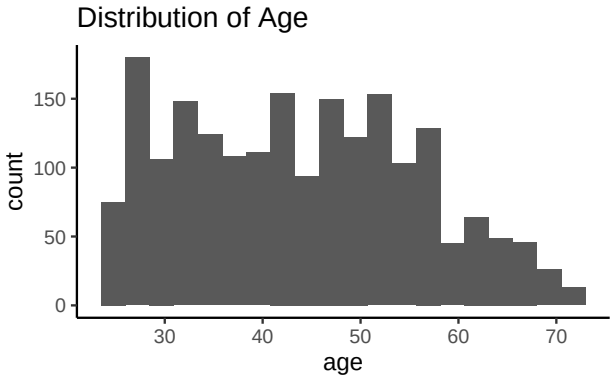


Figure 8: Distribution of age in the NHEFS data.

```
table(nhefs$sex)
```

```
##
##    0    1
##  994 1006
```

Using the NHEFS codebook, we can determine that the marital status categories are:

Category	Marital Status in 1971	N
2	Married	1583
3	Widowed	102
4	Never Married	130
5	Divorced	126
6	Separated	58
8	Unknown	1

First, we will deal with marital status. The first thing we want to do is reduce the number of categories as much as possible. This reduction will be based en-

tirely on substantive (background) knowledge. Let's assume, for our purposes, that we do not expect the average outcome between separated and divorced individuals to differ. We can thus combine their category:

```
nhefs$marital <- ifelse(nhefs$marital ==
    6, 5, nhefs$marital)

table(nhefs$marital)
```

```
##
##    2    3    4    5    8
## 1583  102  130  184    1
```

Next we have to deal with the last (unknown) category, which has a single observation. For our purposes, let's assume the person in this category is among those with the most common value: married

```
nhefs$marital <- ifelse(nhefs$marital ==
    8, 2, nhefs$marital)

table(nhefs$marital)
```

```
##
##    2    3    4    5
## 1584  102  130  184
```

Finally, we will examine the status of the marital variable in R:

```
class(nhefs$marital)
```

```
## [1] "numeric"
```

Because `marital` is a numeric variable, if we include it in a regression model as is, then we will be estimating the association between marital and smoking intensity assuming a linear relation between all the categories. This assumption is untenable for a variable like marital status.

To resolve this, **we must change the class of the marital variable.** We can do this in two ways: first by changing the class in the data object itself; second by changing the class in the model itself:

```
model_2 <- glm(smokeintensity ~ sex + age +
    factor(marital), data = nhefs, family = poisson(link = "log"))

summary(model_2)$coefficients[, 1:2]
```

```
##                      Estimate   Std. Error
## (Intercept)       3.30111415 0.0198670209
## sex              -0.24142834 0.0102123462
## age              -0.00377753 0.0004250036
## factor(marital)3  0.04046523 0.0244502187
## factor(marital)4 -0.10657459 0.0210181383
## factor(marital)5 -0.06804850 0.0181707534
```

We can tell from the output that the referent category for the marital variable is category 2: married. Finally, with the age variable, we must also account for the fact that the relation between age and smoking intensity is potentially nonlinear. We can do this with splines.

In R, splines are easily implemented using the `splines` package. For our particular example, we use b-splines:

```
library(splines)

model_2a <- glm(smokeintensity ~ sex + age +
    factor(marital), data = nhefs, family = poisson(link = "log"))

summary(model_2a)$coefficients[, 1:2]
```

```
##                      Estimate   Std. Error
## (Intercept)       3.30111415 0.0198670209
## sex              -0.24142834 0.0102123462
## age              -0.00377753 0.0004250036
## factor(marital)3  0.04046523 0.0244502187
```

```
## factor(marital)4 -0.10657459 0.0210181383
## factor(marital)5 -0.06804850 0.0181707534
```

```
model_2b <- glm(smokeintensity ~ sex + bs(age,
    df = 3, degree = 3) + factor(marital),
    data = nhefs, family = poisson(link = "log"))

summary(model_2b)$coefficients[, 1:2]
```

```
##                                 Estimate Std. Error
## (Intercept)                  3.108889493 0.01666093
## sex                         -0.243571695 0.01021521
## bs(age, df = 3, degree = 3)1  0.335596239 0.05164451
## bs(age, df = 3, degree = 3)2 -0.290153284 0.04323100
## bs(age, df = 3, degree = 3)3 -0.007047172 0.04103466
## factor(marital)3              0.049850244 0.02459812
## factor(marital)4             -0.094997166 0.02106360
## factor(marital)5             -0.062969924 0.01819567
```

But we can also use natural splines:

```
library(splines)

model_2a_ns <- glm(smokeintensity ~ sex +
    age + factor(marital), data = nhefs,
    family = poisson(link = "log"))

summary(model_2a)$coefficients[, 1:2]
```

```
##                     Estimate   Std. Error
## (Intercept)       3.30111415 0.0198670209
## sex              -0.24142834 0.0102123462
## age              -0.00377753 0.0004250036
## factor(marital)3  0.04046523 0.0244502187
## factor(marital)4 -0.10657459 0.0210181383
## factor(marital)5 -0.06804850 0.0181707534
```

```
model_2b_ns <- glm(smokeintensity ~ sex +
    ns(age, df = 3) + factor(marital), data = nhefs,
    family = poisson(link = "log"))

summary(model_2b)$coefficients[, 1:2]
```

```
##                                    Estimate Std. Error
## (Intercept)                     3.108889493 0.01666093
## sex                            -0.243571695 0.01021521
## bs(age, df = 3, degree = 3)1    0.335596239 0.05164451
## bs(age, df = 3, degree = 3)2 -0.290153284 0.04323100
## bs(age, df = 3, degree = 3)3 -0.007047172 0.04103466
## factor(marital)3                0.049850244 0.02459812
## factor(marital)4               -0.094997166 0.02106360
## factor(marital)5               -0.062969924 0.01819567
```

But what, exactly, are splines? The word spline is an engineering/architectural term. It refers to a flexible piece of material that individuals would use to draw up blue-prints that incorporated flexible curves:

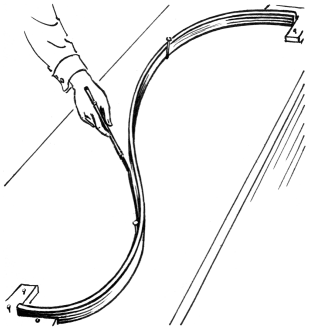


Figure 9: An illustration of a engineering/architectural spline use to draw flexible curves for blueprint diagrams. Source: Wikipedia

In the 1970s and 80s, statisticians began translating some of these engineering concepts to curve fitting. The basic idea was to create functions of a continuous variable that would yield the appropriate degree of flexibility between that value and the conditional outcome expectation.

When it comes to implementation, there are a great many number of different options one can use to fit splines. Among these include natural cubic splines (the `ns()` option in R), B-splines (the `bs()` option in R), generalized additive models (or GAMs, implemented in the `gam` package or the `mgcv` package

in R), penalized smoothing splines (implemented via the `smooth.spline()` function in R), or restricted quadratic splines (Howe et al., 2011).

Of all these, restricted quadratic splines are the easiest to understand. They do not share some of the ideal mathematical properties of the other implementations (properties that we will not discuss, but that relate to the derivatives of the spline functions). However, here we will walk through the steps to create restricted quadratic splines to demonstrate how splines work in principle.

14 Restricted Quadratic Splines

When using splines, the basic question is about how to code the relation between the conditional expectation of the outcome and a continuous covariate. For example, suppose we had the following exposure (x) and outcome (y) data:

```
# load package needed to generate
# laplace distribution
install.packages("rmutil", repos = "http://lib.stat.cmu.edu/R/CRAN/")
```

```
##
## The downloaded binary packages are in
##  /var/folders/z_/cty0tpg97wz_x1d1zgdhwllr0000gs/T//RtmpEbfrkz/downloaded_packages
```

```
library(rmutil)
```

```
## Registered S3 method overwritten by 'rmutil':
##   method         from
##   print.response httr
##
## Attaching package: 'rmutil'
## The following objects are masked from 'package:VGAM':
##
##     dbetabinom, dlaplace, dlevy, dpareto, dsimplex, nobs, pbetabinom,
##     plaplace, plevy, ppareto, qlaplace, qlevy, qpareto, rbetabinom,
##     rlaplace, rlevy, rpareto, rsimplex
```

```
## The following object is masked from 'package:stats4':
##
##     nobs

## The following object is masked from 'package:rlang':
##
##     int

## The following object is masked from 'package:Hmisc':
##
##     units

## The following object is masked from 'package:tidyr':
##
##     nesting

## The following object is masked from 'package:stats':
##
##     nobs

## The following objects are masked from 'package:base':
##
##     as.data.frame, units
```

```
# set the seed for reproducibility
set.seed(12345)


## generate the observed data
n = 1000
# uniform random variable bounded by 0
# and 8
x = runif(n, 0, 8)
# continuous outcome as a complex
# function of x
y = 5 + 4 * sqrt(9 * x) * as.numeric(x <
    2) + as.numeric(x >= 2) * (abs(x - 6)^(2)) +
    rlaplace(n)
```

```
a <- data.frame(x = x, y = y)
head(a)
```

```
##          x          y
## 1 5.767231  3.1931580
## 2 7.006186  7.1753275
## 3 6.087859  0.7120441
## 4 7.088997  5.8326162
## 5 3.651848 10.9792288
## 6 1.330974 18.1672097
```

We can create a scatter plot of these data to see how they relate:

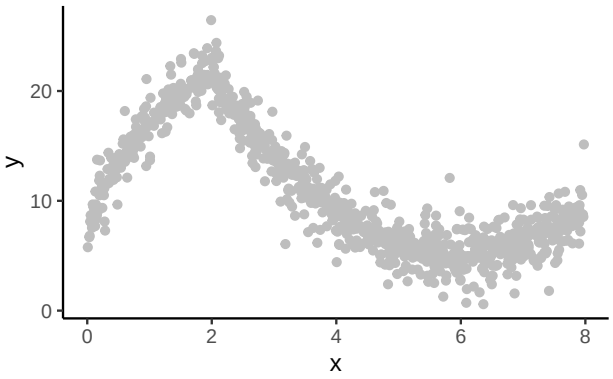


Figure 10: Scatterplot of the relation between a simulated exposure and simulated outcome with a complex curvilinear relation

Obviously, the relation between X and Y is not a straight line. But suppose we assume linearity, fitting the following regression model to these data:

$$E(Y \mid X) = \beta_0 + \beta_1 X$$

```
model1 <- lm(y ~ x)
a$y_pred1 <- predict(model1)
```

If X were a confounder and we assumed such a linear fit, there would be an important degree of residual confounding left over in our estimate. If X were our exposure of interest, such a linear fit would seriously mis-represent the true functional relation between the exposure and the outcome. Splines are meant to solve these problems.

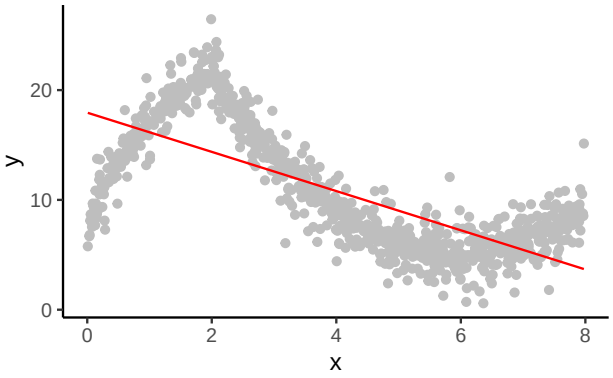


Figure 11: Scatterplot of the relation between a simulated exposure and simulated outcome with a complex curvilinear relation and a linear fit

Splines are essentially a function that take the exposure as an argument, and return a set of **basis functions** that account for the curvilinear relation. Any spline starts by selecting knots, which are the points along the variable's distribution where we will create categories. In our example, we will use three knots chosen at $x = 1$, $x = 4$, and $x = 6$. We will denote these χ_1, χ_2, and χ_3, respectively.[17]

Restricted quadratic spline basis functions for a three knot spline can then be defined as follows:

$$f(x) = \left[(x - \chi_1)_+^2 - (x - \chi_3)_+^2\right] \left[(x - \chi_2)_+^2 - (x - \chi_3)_+^2\right]$$

The parentheses with a subscripted plus sign refers to the *positive part function* returns the value of the difference if it is positive, and zero otherwise[18]

With these equations, we can create the spline basis functions we need to fit restricted quadratic splines with out simulated data:

```
basis_1 <- as.numeric((x - 1) > 0) * (x -
    1)^2 - as.numeric((x - 6) > 0) * (x -
    6)^2
basis_2 <- as.numeric((x - 3) > 0) * (x -
    3)^2 - as.numeric((x - 6) > 0) * (x -
    6)^2
```

We can now fit our regression model using these spline basis functions:

[17] Knots can be chosen as *a priori* cutpoints, or by selecting percentile's of the distribution (e.g., the 25th, 50th, and 75th percentile values).

[18] formally, $(x-\chi_1)_+ = x-\chi_1$ if $(x-\chi_1) > 0; 0$ otherwise

```
model2 <- lm(y ~ x + basis_1 + basis_2)
a$y_pred2 <- predict(model2)
```

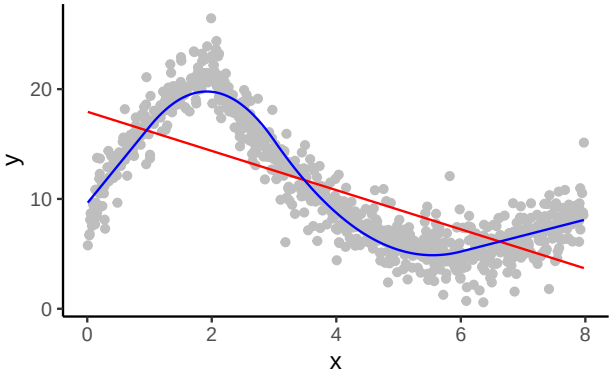


Figure 12: Scatterplot of the relation between a simulated exposure and simulated outcome with a complex curvilinear relation and a linear fit

Clearly, using splines gives us a much better fit.

A natural question that arises from this illustration is, how do we interpret the spline results? For example, if we look at a summary of the estimates from `model2`, we get:

```
summary(model2)$coefficients[, -3]
```

```
##              Estimate Std. Error      Pr(>|t|)
## (Intercept)  9.604306 0.20346470 2.641625e-256
## x            6.974522 0.15343656 3.773854e-245
## basis_1     -3.796962 0.05921589  0.000000e+00
## basis_2      5.408630 0.08260659  0.000000e+00
```

Can we interpret the 7, -3.8, and 5.4 that we estimated for the linear and spline terms? The answer is **no.**

An analyst will encounter using splines in two settings: 1) adjusting for a continuous confounder; and 2) accounting for a curvilinear relation between an exposure and an outcome. When adjusting for confounding, interest often lies primarily in the exposure-outcome relation.

The confounder-outcome relation is usually not of particular interest. Indeed, this part of the relation is often referred to as the "nuisance function" in the literature on semiparametric methods (Tsiatis, 2006). As a result, even though spline estimates do not have an interpretation, it does not matter as long as they are appropriately adjusting for the confounder-outcome relation.

If splines are being used to model a continuous expsoure-outcome relation, then the interpretation of the estimates will not matter as long as there is a curvilinear relation. Consider, for example, the use of quadratic and cubic terms:

$$E(Y \mid X) = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3$$

The objective of fitting these quadratic and cubic terms is to account for any curvilinear relation. One would not typically interpret the coefficients of the squared and cubic terms. One could, however, predict the outcome under different values of X from this model, and compare these predicted outcomes.

The same principles apply to splines. There is no interesting way to interpret the coefficients for the spline terms. However we could obtain estimates of the effect of changing X from one level to another. Suppose we were interested in comparing the average outcome under $X = 1$ versus $X = 2$ and $X = 2$ versus $X = 4$. We could easily do this using the splines we fit above:

```
x = 1
basis_1 <- as.numeric((x - 1) > 0) * (x -
    1)^2 - as.numeric((x - 6) > 0) * (x -
    6)^2
basis_2 <- as.numeric((x - 3) > 0) * (x -
    3)^2 - as.numeric((x - 6) > 0) * (x -
    6)^2

nd1 <- data.frame(x, basis_1, basis_2)
mu1 <- predict(model2, newdata = nd1)

x = 2
basis_1 <- as.numeric((x - 1) > 0) * (x -
    1)^2 - as.numeric((x - 6) > 0) * (x -
    6)^2
basis_2 <- as.numeric((x - 3) > 0) * (x -
    3)^2 - as.numeric((x - 6) > 0) * (x -
    6)^2
```

```
nd2 <- data.frame(x, basis_1, basis_2)
mu2 <- predict(model2, newdata = nd2)

x = 6
basis_1 <- as.numeric((x - 1) > 0) * (x -
    1)^2 - as.numeric((x - 6) > 0) * (x -
    6)^2
basis_2 <- as.numeric((x - 3) > 0) * (x -
    3)^2 - as.numeric((x - 6) > 0) * (x -
    6)^2

nd6 <- data.frame(x, basis_1, basis_2)
mu6 <- predict(model2, newdata = nd6)

mu2 - mu1
```

```
##       1
## 3.17756
```

```
mu6 - mu2
```

```
##         1
## -14.55133
```

Here, we see that the effect of going from $X = 1$ to $X = 2$ is 3.18 while the effect of going from $X = 6$ to $X = 2$ is -14.55. Notably, these estimates account for the curvilinear relation between X and Y. Confidence intervals can be obtained using the bootstrap.

References

A Abadie and Guido W. Imbens. On the failure of the bootstrap for matching estimators. *Econometrica*, 76(6):1537–1557, 2008.

Peter C. Austin. Optimal caliper widths for propensity-score matching when estimating differences in means and differences in proportions in observational studies. *Pharmaceutical Statistics*, 10(2):150–161, 2022/03/22 2011. DOI: https://doi.org/10.1002/pst.433. URL https://doi.org/10.1002/pst.433.

S. R. Cole and M. A. Hernán. Constructing inverse probability weights for marginal structural models. *Am J Epidemiol*, 168(6):656–64, 2008.

D. R. Cox. Regression models and life-tables. *J R Stat Soc B*, 34(2):187–220, 1972.

D. R. Cox. Partial likelihood. *Biometrika*, 62(2):269–276, 1975.

R. B. DAGOSTINO, M. L. LEE, A. J. BELANGER, L. A. CUPPLES, K. ANDERSON, and W. B. KANNEL. Relation of pooled logistic-regression to time-dependent cox regression-analysis - the framingham heart-study. *STATISTICS IN MEDICINE*, 9(12):1501–1515, Dec 1990. ISSN 0277-6715. DOI: 10.1002/sim.4780091214.

Bradley Efron. The efficiency of cox's likelihood function for censored data. *J Am Stat Assoc*, 72(359):557—565, 1977.

L. Györfi, M. Kohler, A. Krzyzak, and H. Walk. *A Distribution-Free Theory of Nonparametric Regression*. Springer, New York, NY, 2002.

Masayuki Henmi and Shinto Eguchi. A paradox concerning nuisance parameters and projected estimating functions. *Biometrika*, 91(4):929–941, 12 2004.

M. A. Hernán. The hazards of hazard ratios. *Epidemiol*, 21:13–5, 2010.

M. A. Hernán and JM Robins. *Causal Inference*. Chapman/Hall, Boca Raton, FL, Forthcoming.

Irva Hertz-Picciotto and Beverly Rockhill. Validity and efficiency of approximation methods for tied survival times in cox regression. *Biometrics*, 53(3): 1151–1156, 1997.

Keisuke Hirano, Guido W. Imbens, and Geert Ridder. Efficient estimation of average treatment effects using the estimated propensity score. *Econometrica*, 71(4):1161–1189, 2003.

CJ Howe, SR Cole, DJ Westreich, S Greenland, S Napravnik, and JJ Eron. Splines for trend analysis and continuous confounder control. *Epidemiol*, 22(6):874–5, 2011.

Brian K. Lee, Justin Lessler, and Elizabeth A. Stuart. Improving propensity score weighting using machine learning. *Stat Med*, 29(3):337–346, 2010.

NT Longford. *Studying Human Populations: An Advanced Course in Statistics*. Springer, New York, 2008.

Jared K. Lunceford and Marie Davidian. Stratification and weighting via the propensity score in estimation of causal treatment effects: a comparative study. *Statistics in Medicine*, 23(19):2937–2960, 2022/03/22 2004. DOI: https://doi.org/10.1002/sim.1903. URL https://doi.org/10.1002/sim.1903.

AI Naimi, A Mishler, and Edward H. Kennedy. Challenges in obtaining valid causal effect estimates with machine learning algorithms. *Am J Epidemiol*, kwab201, 2022.

Ashley I Naimi, Stephen R Cole, and Edward H Kennedy. An Introduction to G Methods. *Int J Epidemiol*, 46(2):756–62, 2017.

J. A. Nelder and R. W. M. Wedderburn. Generalized linear models. *JRSS-A*, 135 (3):370–384, 1972.

Youssef Oulhote, Brent Coull, Marie-Abele Bind, Frodi Debes, Flemming Nielsen, Ibon Tamayo, Pal Weihe, and Philippe Grandjean. Joint and independent neurotoxic effects of early life exposures to a chemical mixture: A multipollutant approach combining ensemble learning and g-computation. *Environmental Epidemiology*, 3(5):e063, 2019.

Alvin C. Rencher. *Linear Models in Statistics*. Wiley, New York, 2000.

J M Robins and Y Ritov. Toward a curse of dimensionality appropriate (coda) asymptotic theory for semi-parametric models. *Stat Med*, 16(1-3):285–319, Jan 1997.

J. M. Robins, S. D. Mark, and W. K. Newey. Estimating exposure effects by modelling the expectation of exposure conditional on confounders. *Biometrics*, 48(2):479–95, 1992.

Paul R. Rosenbaum and Donald B. Rubin. The central role of the propensity score in observational studies for causal effects. *Biometrika*, 70(1):41–55, 1983.

G. A. F. Seber and C. J. Wild. *Nonlinear regression*. Wiley, New York, 1989.

Cosma Rohilla Shalizi. *The Truth About Linear Regression*. https://www.stat.cmu.edu/ cshalizi/TALR/TALR.pdf, 2019.

Jonathan M. Snowden, Sherri Rose, and Kathleen M. Mortimer. Implementation of g-computation on a simulated data set: Demonstration of a causal inference technique. *Am J Epidemiol*, 173(7):731–738, 2011.

Anastasios A. Tsiatis. *Semiparametric theory and missing data*. Springer, New York, 2006.

Larry Wasserman. *All of nonparametric statistics*. Springer, New York; London, 2006.

Daniel Westreich, Justin Lessler, and Michele Jonsson Funk. Propensity score estimation: neural networks, support vector machines, decision trees (cart), and meta-classifiers as alternatives to logistic regression. *J Clin Epidemiol*, 63(8):826 – 833, 2010.

Kazuki Yoshida, Sonia Hernández-Díaz, Daniel H Solomon, John W Jackson, Joshua J Gagne, Robert J Glynn, and Jessica M Franklin. Matching weights to simultaneously compare three treatment groups: Comparison to three-way matching. *Epidemiology (Cambridge, Mass.)*, 28(3):387–395, 2017.

Guangyong Zou. A modified poisson regression approach to prospective studies with binary data. *Am J Epidemiol*, 159(7):702–706, Apr 2004.