Mark van der Laan, Jeremy Coyle, Nima Hejazi, Ivana Malenica, Rachael Phillips, Alan Hubbard

Targeted Learning in R

Causal Data Science with the tlverse Software Ecosystem

Contents

List of Tables

List of Figures

About this book

Targeted Learning in `R`*: Causal Data Science with the* `tlverse` *Software Ecosystem* is an open source, reproducible electronic handbook for applying the Targeted Learning methodology in practice using the `tlverse` software ecosystem. This work is currently in an early draft phase and is available to facilitate input from the community. To view or contribute to the available content, consider visiting the GitHub repository.

0.1 Outline

The contents of this handbook are meant to serve as a reference guide for applied research as well as materials that can be taught in a series of short courses focused on the applications of Targeted Learning. Each section introduces a set of distinct causal questions, motivated by a case study, alongside statistical methodology and software for assessing the causal claim of interest. The (evolving) set of materials includes

- Motivation: Why we need a statistical revolution
- The Roadmap and introductory case study: the WASH Beneifits data
- Introduction to the `tlverse` software ecosystem
- Cross-validation with the `origami` package
- Ensemble machine learning with the `sl3` package
- Targeted learning for causal inference with the `tmle3` package
- Optimal treatments regimes and the `tmle3mopttx` package
- Stochastic treatment regimes and the `tmle3shift` package
- Causal mediation analysis with the `tmle3mediate` package
- *Coda*: Why we need a statistical revolution

What this book is not

The focus of this work is **not** on providing in-depth technical descriptions of current statistical methodology or recent advancements. Instead, the goal is to convey key details of state-of-the-art techniques in an manner that is both clear and complete, without burdening the reader with extraneous information. We hope that the presentations herein will serve as references for researchers – methodologists and domain specialists alike – that empower them to deploy the central tools of Targeted Learning in an efficient manner. For technical details and in-depth descriptions of both classical theory and recent advances in the field of Targeted Learning, the interested reader is invited to consult van der Laan and Rose (2011) and/or van der Laan and Rose (2018) as appropriate. The primary literature in statistical causal inference, machine learning, and non/semiparametric theory include many of the most recent advances in Targeted Learning and related areas.

About the authors

Mark van der Laan

Mark van der Laan, PhD, is Professor of Biostatistics and Statistics at UC Berkeley. His research interests include statistical methods in computational biology, survival analysis, censored data, adaptive designs, targeted maximum likelihood estimation, causal inference, data-adaptive loss-based learning, and multiple testing. His research group developed loss-based super learning in semiparametric models, based on cross-validation, as a generic optimal tool for the estimation of infinite-dimensional parameters, such as nonparametric density estimation and prediction with both censored and uncensored data. Building on this work, his research group developed targeted maximum likelihood estimation for a target parameter of the data-generating distribution in arbitrary semiparametric and nonparametric models, as a generic optimal methodology for statistical and causal inference. Most recently, Mark's group has focused in part on the development of a centralized, principled set of software tools for targeted learning, the `tlverse`.

Jeremy Coyle

Jeremy Coyle, PhD, is a consulting data scientist and statistical programmer, currently leading the software development effort that has produced the `tlverse` ecosystem of R packages and related software tools. Jeremy earned his PhD in Biostatistics from UC Berkeley in 2016, primarily under the supervision of Alan Hubbard.

Nima Hejazi

Nima Hejazi is an NSF postdoctoral research fellow in the Division of Biostatistics at Weill Cornell Medicine and an affiliate postdoctoral researcher in the Vaccine and Infectious Disease Division at the Fred Hutchinson Cancer Research Center. In 2021, he completed his PhD in biostatistics under the supervision of Mark van der Laan and Alan Hubbard. Broadly, Nima's research interests combine causal inference, machine learning, and computational statistics, drawing on ideas from non/semi-parametric estimation to develop robust statistical procedures with efficiency guarantees. Particular methodological areas of current emphasis include causal mediation analysis, efficient inference under two-phase sampling designs, sieve estimation with machine learning, and targeted loss-based estimation. His work is usually inspired by applications in vaccine clinical trials, computational biology, and epidemiology. He is also passionate about statistical computing, open source software design for applied statistics, and research software engineering. Nima has made significant contributions to the `hal9001`, `sl3`, `tmle3`, and `origami` packages, is the primary author of the `tmle3shift` and `tmle3mediate` packages, and has been responsible for package maintenance and distribution across the `tlverse` ecosystem.

Ivana Malenica

Ivana Malenica is a PhD student in biostatistics advised by Mark van der Laan. Ivana is currently a fellow at the Berkeley Institute for Data Science, after serving as a NIH Biomedical Big Data and Freeport-McMoRan Genomic Engine fellow. She earned her Master's in Biostatistics and Bachelor's in Mathematics, and spent some time at the Translational Genomics Research Institute. Very broadly, her research interests span non/semi-parametric theory, probability theory, machine learning, causal inference and high-dimensional statistics. Most of her current work involves complex dependent settings (dependence through time and network) and adaptive sequential designs.

Rachael Phillips

Rachael Phillips is a PhD student in biostatistics, advised by Alan Hubbard and Mark van der Laan. She has an MA in Biostatistics, BS in Biology, and BA in Mathematics. A student of targeted learning and causal inference, Rachael's research focuses on statistical estimation and inference in realistic statistical models. Her current projects involve personalized online machine learning from EHR streaming data of vital signs, automated learning with highly adaptive lasso, and causal effect estimation for community-level interventions. She is also working on an FDA-funded project led Dr. Susan Gruber, *A Targeted Learning Framework for Causal Effect Estimation Using Real-World Data*. Rachael is an active contributor to the `hal9001` and `sl3` R packages in the `tlverse`.

Alan Hubbard

Alan Hubbard is Professor of Biostatistics, former head of the Division of Biostatistics at UC Berkeley, and head of data analytics core at UC Berkeley's SuperFund research program. His current research interests include causal inference, variable importance analysis, statistical machine learning, estimation of and inference for data-adaptive statistical target parameters, and targeted minimum loss-based estimation. Research in his group is generally motivated by applications to problems in computational biology, epidemiology, and precision medicine.

0.2 Learning resources

To effectively utilize this handbook, the reader need not be a fully trained statistician to begin understanding and applying these methods. However, it is highly recommended for the reader to have an understanding of basic statistical concepts such as confounding, probability distributions, confidence intervals, hypothesis tests, and regression. Advanced knowledge of mathematical statistics may be useful but is not necessary. Familiarity with the `R` programming language will be essential. We also recommend an understanding of introductory causal inference.

For learning the `R` programming language we recommend the following (free) introductory resources:

- Software Carpentry's *Programming with* `R`

- Software Carpentry's *`R` for Reproducible Scientific Analysis*
- Garret Grolemund and Hadley Wickham's *`R` for Data Science*

For a general introduction to causal inference, we recommend

- Miguel A. Hernán and James M. Robins' *Causal Inference: What If*, 2021
- Jason A. Roy's *A Crash Course in Causality: Inferring Causal Effects from Observational Data* on Coursera

0.3 Setup instructions

0.3.1 R and RStudio

R and **RStudio** are separate downloads and installations. R is the underlying statistical computing environment. RStudio is a graphical integrated development environment (IDE) that makes using R much easier and more interactive. You need to install R before you install RStudio.

0.3.1.1 Windows

0.3.1.1.1 If you already have R and RStudio installed

- Open RStudio, and click on "Help" > "Check for updates". If a new version is available, quit RStudio, and download the latest version for RStudio.
- To check which version of R you are using, start RStudio and the first thing that appears in the console indicates the version of R you are running. Alternatively, you can type `sessionInfo()`, which will also display which version of R you are running. Go on the CRAN website and check whether a more recent version is available. If so, please download and install it. You can check here for more information on how to remove old versions from your system if you wish to do so.

0.3.1.1.2 If you don't have R and RStudio installed

- Download R from the CRAN website.
- Run the `.exe` file that was just downloaded
- Go to the RStudio download page

- Under *Installers* select **RStudio x.yy.zzz - Windows XP/Vista/7/8** (where x, y, and z represent version numbers)
- Double click the file to install it
- Once it's installed, open RStudio to make sure it works and you don't get any error messages.

0.3.1.2 macOS / Mac OS X

0.3.1.2.1 If you already have R and RStudio installed

- Open RStudio, and click on "Help" > "Check for updates". If a new version is available, quit RStudio, and download the latest version for RStudio.
- To check the version of R you are using, start RStudio and the first thing that appears on the terminal indicates the version of R you are running. Alternatively, you can type `sessionInfo()`, which will also display which version of R you are running. Go on the CRAN website and check whether a more recent version is available. If so, please download and install it.

0.3.1.2.2 If you don't have R and RStudio installed

- Download R from the CRAN website.
- Select the `.pkg` file for the latest R version
- Double click on the downloaded file to install R
- It is also a good idea to install XQuartz (needed by some packages)
- Go to the RStudio download page
- Under *Installers* select **RStudio x.yy.zzz - Mac OS X 10.6+ (64-bit)** (where x, y, and z represent version numbers)
- Double click the file to install RStudio
- Once it's installed, open RStudio to make sure it works and you don't get any error messages.

0.3.1.3 Linux

- Follow the instructions for your distribution from CRAN, they provide information to get the most recent version of R for common distributions. For most distributions, you could use your package manager (e.g., for Debian/Ubuntu run `sudo apt-get install r-base`, and for Fedora `sudo yum install R`), but we don't recommend this approach as the versions provided by this are usually out of date. In any case, make sure you have at least R 3.3.1.

- Go to the RStudio download page
- Under *Installers* select the version that matches your distribution, and install it with your preferred method (e.g., with Debian/Ubuntu `sudo dpkg -i rstudio-x.yy.zzz-amd64.deb` at the terminal).
- Once it's installed, open RStudio to make sure it works and you don't get any error messages.

These setup instructions are adapted from those written for Data Carpentry: R for Data Analysis and Visualization of Ecological Data.

1

Robust Statistics and Reproducible Science

> "One enemy of robust science is our humanity – our appetite for being right, and our tendency to find patterns in noise, to see supporting evidence for what we already believe is true, and to ignore the facts that do not fit."
>
> — *Nature* Editorial (Anonymous) (2015b)

Scientific research is at a unique point in its history. The need to improve rigor and reproducibility in our field is greater than ever; corroboration moves science forward, yet there is growing alarm that results cannot be reproduced or validated, suggesting the possibility that many discoveries may be false (Baker, 2016). Consequences of not meeting this need will result in further decline in the rate of scientific progress, the reputation of the sciences, and the public's trust in scientific findings (Munafò et al., 2017; *Nature* Editorial (Anonymous), 2015a).

> "The key question we want to answer when seeing the results of any scientific study is whether we can trust the data analysis."
>
> — Peng (2015)

Unfortunately, in its current state, the culture of statistical data analysis enables, rather than precludes, the manner in which human bias may affect the results of (ideally objective) data analytic efforts. A significant degree of human bias enters statistical analysis efforts in the form improper model selection. All procedures for estimation and hypothesis testing are derived based on a choice of statistical model; thus, obtaining valid estimates and statistical inference relies critically on the chosen statistical model containing an accurate representation of the process that generated the data. Consider, for example, a hypothetical study in which a treatment was assigned to a group of patients: Was the treatment assigned randomly or were characteristics of the individuals (i.e., baseline covariates) used in making the treatment decision? Such knowledge can should be incorporated in the statistical model. Alternatively, the data could be from an observational study, in which there is no control over the treatment assignment mechanism. In such cases, available knowledge about the data-generating process (DGP) is more limited still. If this is the case, then the statistical model should contain *all* possible distributions of the data. In practice, however, models are not selected based on scientific knowledge available about the DGP; instead, models are often selected based on (1) the philosophical leanings of the analyst, (2) the relative convenience of implementation of statistical methods admissible within the choice of model, and (3) the results of significance testing (i.e., p-values) applied within the choice of model.

This practice of "cargo-cult statistics — the ritualistic miming of statistics rather than conscientious practice," (Stark and Saltelli, 2018) is characterized by arbitrary modeling choices, even though these choices often result in different answers to the same research question. That is, "increasingly often, [statistics] is used instead to aid and abet weak science, a role it can perform well when used mechanically or ritually," as opposed to its original purpose of safeguarding against weak science by providing formal techniques for evaluating the veracity of a claim using properly collected data (Stark and Saltelli, 2018). This presents a fundamental drive behind the epidemic of false findings from which scientific research is suffering (van der Laan and Starmans, 2014).

> "We suggest that the weak statistical understanding is probably due to inadequate"statistics lite" education. This approach does not build up appropriate mathematical fundamentals and does not provide scientifically rigorous introduction into statistics. Hence, students' knowledge may remain imprecise, patchy, and prone to serious misunderstandings. What this approach achieves, however, is providing students with false confidence of being able to use inferential tools whereas they usually only interpret the p-value provided by black box statistical software. While this educational problem remains unaddressed, poor statistical practices will prevail regardless of what procedures and measures may be favored and/or banned by editorials."
>
> — Szucs and Ioannidis (2017)

Our team at the University of California, Berkeley is uniquely positioned to provide such an education. Spearheaded by Professor Mark van der Laan, and spreading rapidly by many of his students and colleagues who have greatly enriched the field, the aptly named "Targeted Learning" methodology emphasizes a focus of (i.e., "targeting of") the scientific question at hand, running counter to the current culture problem of "convenience statistics," which opens the door to biased estimation, misleading analytic results, and erroneous discoveries. Targeted Learning embraces the fundamentals that formalized the field of statistics, notably including the notions that a statistical model must represent real knowledge about the experiment that generated the data and that a target parameter represents what we are seeking to learn from the data as a feature of the distribution that generated it (van der Laan and Starmans, 2014). In this way, Targeted Learning defines a truth and establishes a principled standard for estimation, thereby curtailing our all-too-human biases (e.g., hindsight bias, confirmation bias, and outcome bias) from infiltrating our objective analytic efforts.

> "The key for effective classical [statistical] inference is to have well-defined questions and an analysis plan that tests those questions."
>
> — Nosek et al. (2018)

This handbook aims to provide practical training to students, researchers, industry professionals, and academicians in the sciences (whether biological, physical, economic, or social), public health, statistics, and numerous other fields, to equip them with the necessary knowledge and skills to utilize the the methodological developments of Targeted Learning — a technique that provides tailored pre-specified machines for answering queries — taking advantage of estimators that are efficient, minimally biased, and that provide formal statistical inference — so that each and every data analysis incorporates state-of-the-art statistical methodology, all while ensuring compatibility with the guiding principles of computational reproducibility.

Just as the conscientious use of modern statistical methodology is necessary to ensure that scientific practice thrives, robust, well-tested software plays a critical role in allowing practitioners to direct access the published results of a given scientific investigation. In fact, "an article…in a scientific publication is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures," thus making the availability and adoption of robust statistical software key to enhancing the transparency that is an inherent (and assumed) aspect of the scientific process (Buckheit and Donoho, 1995).

For a statistical methodology to be readily accessible in practice, it is crucial that it is accompanied by user-friendly software (Pullenayegum et al., 2016; Stromberg et al., 2004). The `tlverse` software ecosystem, composed of a set of package for the `R` language and environment for statistical computing (R Core Team, 2021), was developed to fulfill this need for the Targeted Learning methodological framework. Not only does this suite of software tools facilitate computationally reproducible and efficient analyses, it is also a tool for Targeted Learning education, since its workflow mirrors the central aspects of the statistical methodology. In particular, the programming paradigm central to the `tlverse` ecosystem does not focus on implementing a specific estimator or a small set of related estimators. Instead, the focus is on exposing the statistical framework of Targeted Learning itself — all software packages in the `tlverse` ecosystem directly model the key objects defined in the mathematical and theoretical framework of Targeted Learning. What's more, the `tlverse` software packages share a core set of design principles centered on extensibility, allowing for them all to be used in conjunction with each other and even used cohesively as building blocks for formulating sophisticated statistical analyses. For an introduction to the Targeted Learning framework, we recommend a recent review paper from Coyle et al. (2021).

In this handbook, the reader will embark on a journey through the `tlverse` ecosystem. Guided by `R` programming exercises, case studies, and intuition-building explanations, readers will learn to use a toolbox for applying the Targeted Learning statistical methodology, which will translate to real-world causal inference analyses. Some preliminaries are required prior to this learning endeavor – we have made available a list of recommended learning resources.

2

The Roadmap for Targeted Learning

Nima Hejazi and *Rachael Phillips*

Updated: 2021-12-30

Learning Objectives

In this chapter, we provide guidance on how to

1. Translate scientific questions to statistical questions. 2. Define a statistical model based on the knowledge of the experiment that generated the data. 3. Identify a causal parameter as a function of the observed data distribution. 4. Explain the following statistical and causal assumptions and their implications: i.i.d., consistency, no unmeasured confounding, interference, positivity.

Introduction

The roadmap of statistical learning is concerned with the translation from real-world data applications to a mathematical and statistical formulation of the relevant estimation problem. This involves data as a random variable having a probability distribution, scientific knowledge represented by a statistical model, a statistical target parameter representing an answer to the question of interest, and the notion of an estimator and sampling distribution of the estimator.

2.1 The Roadmap

The roadmap is a five-stage process of defining the following.

1. Data as a random variable with a probability distribution, $O \sim P_0$.
2. The statistical model $\mathcal{M}$ such that $P_0 \in \mathcal{M}$.
3. The statistical target parameter Ψ and estimand $\Psi(P_0)$.
4. The estimator $\hat{\Psi}$ and estimate $\hat{\Psi}(P_n)$.
5. A measure of uncertainty for the estimate $\hat{\Psi}(P_n)$.

(1) Data: A random variable with a probability distribution, $O \sim P_0$

The data set we are confronted with is the collection of the results of an experiment, and we can view the data as a *random variable* – that is, if we were to repeat the experiment, we would have a different realization of the data generated by the experiment in question. In particular, if the experiment were repeated many times, the probability distribution generating the data, P_0, could be learned. So, the observed data on a single unit, O, may be thought of as being drawn from a probability distribution P_0. Most often, we observe n *independent identically distributed* (i.i.d.) observations of the random variable O, so the observed data is the collection $O_1, \ldots, O_n$, where the subscripts denote the individual observational units. While not all data are i.i.d., this is certainly the most common case in applied data analysis; moreover, there are a number of techniques for handling non-i.i.d. data, such as establishing conditional independence, stratifying data to create distinct sets of identically distributed data, and inferential corrections for repeated or clustered observations, to name but a few.

It is crucial that the domain scientist (i.e., researcher) have absolute clarity about what is actually known about the data-generating distribution for a given problem of interest. Just as critical is that this scientific information be communicated to the statistician, whose job it is to use such knowledge to guide any assumptions encoded in the choice of statistical model. Unfortunately, communication between statisticians and researchers is often fraught with misinterpretation. The roadmap provides a mechanism by which to ensure clear communication between the researcher and the statistician – it is an invaluable tool for such communication!

The empirical probability measure, P_n

With n i.i.d. observations in hand, we can define an empirical probability measure, P_n. The empirical probability measure is an approximation of the true probability measure, P_0, allowing us to learn from the observed data. For example, we can define the empirical probability measure of a set X to be the proportion of observations that belong in X. That is,

$$P_n(X) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}(O_i \in X)$$

In order to start learning from the data, we next need to ask *"What do we know about the probability distribution of the data?"* This brings us on to Step 2.

(2) Defining the statistical model $\mathcal{M}$ such that $P_0 \in \mathcal{M}$

The statistical model $\mathcal{M}$ is defined by the question we asked at the end of Step 1. It is the set of possible probability distributions that could describe our observed data, appropriately constrained by background scientific knowledge. Often $\mathcal{M}$ is very large (e.g., nonparametric), reflecting the fact that statistical knowledge about the data-generating process is limited.

Alternatively, if the probability distribution of the data at hand is described by a finite number of parameters, then the statistical model is referred to as *parametric*. Such an assumption is made, for example, by the proposition that the random variable of interest, O, has a normal distribution with mean μ and variance σ^2. More generally, a parametric model may be defined as

$$\mathcal{M} = \{P_\theta : \theta \in \mathbb{R}^d\},$$

which describes a statistical model consisting of all distributions P_θ, that is all distributions indexed only by the parameter θ.

The assumption that the data-generating distribution has a specific, parametric form is made quite commonly, even when such assumptions are not supported by existing knowledge. This practice of oversimplification in the current culture of data analysis typically complicates any attempt at trying to answer the scientific question at hand, owing to the fact that possible model misspecification introduces bias of unknown magnitude. The philosophy used to justify such parametric assumptions is captured by the quote of George Box that "All models are wrong but some are useful," which encourages the data analyst to make arbitrary modeling choices. The result is a practice of data science that often yields starkly different answers to the same scientific problem, due to the differing modeling decisions and assumptions made by different analysts. Even in the nascent days of data analysis, it was recognized that it is “far better [to develop] an approximate answer to the right question…than an exact answer to the wrong question, which can always be made precise” (Tukey, 1962), though traditional statistics failed to heed this advice for a number of decades (Donoho, 2017). The Targeted Learning paradigm avoids this bias by defining the statistical model through a representation of the true data-generating distribution corresponding to the observed data. The ultimate goal is to formulate the statistical estimation problem *exactly*, so that one can then set out to tailor to the problem the best possible estimation procedure.

Now, on to Step 3: *“What are we trying to learn from the data?”*

(3) The statistical target parameter Ψ and estimand $\Psi(P_0)$

The statistical target parameter, Ψ, is defined as a mapping from the statistical model, $\mathcal{M}$, to the parameter space (i.e., a real number) $\mathbb{R}$ – that is, the target parameter is the mapping $\Psi : \mathcal{M} \rightarrow \mathbb{R}$. The estimand may be seen as a representation of the quantity that we wish to learn from the data, the answer to a well-specified (often causal) question of interest. In contrast to purely statistical estimands, causal estimands require *identification from the observed data*, based on causal models that include several untestable assumptions, described in greater detail in the section on causal target parameters.

For a simple example, consider a data set which contains observations of a survival time on every subject, for which our question of interest is “What’s the probability that someone lives longer than five years?” We have,

$$\Psi(P_0) = \mathbb{P}_O(O > 5) = \int_5^\infty dP_O(o)$$

This answer to this question is the **estimand,** $\Psi(P_0)$, which is the quantity we wish to learn from the data. Once we have defined O, $\mathcal{M}$ and $\Psi(P_0)$ we have formally defined the statistical estimation problem.

(4) The estimator $\hat{\Psi}$ and estimate $\hat{\Psi}(P_n)$

Typically, we will focus on estimation in realistic, nonparametric models. To obtain a good approximation of the estimand, we need an estimator, an *a priori*-specified algorithm defined as a mapping from the set of possible empirical distributions, P_n, which live in a non-parametric statistical model, $\mathcal{M}_{NP}$ ($P_n \in \mathcal{M}_{NP}$), to the parameter space of the parameter of interest. That is, $\hat{\Psi} : \mathcal{M}_{NP} \to \mathbb{R}^d$. The estimator is a function that takes as input the observed data, a realization of P_n, and gives as output a value in the parameter space, which is the **estimate,** $\hat{\Psi}(P_n)$.

Where the estimator may be seen as an operator that maps the observed data and corresponding empirical distribution to a value in the parameter space, the numerical output that produced such a function is the estimate. Thus, it is an element of the parameter space based on the empirical probability distribution of the observed data. If we plug in a realization of P_n (based on a sample size n of the random variable O), we get back an estimate $\hat{\Psi}(P_n)$ of the true parameter value $\Psi(P_0)$.

In order to quantify the uncertainty in our estimate of the target parameter (i.e., to construct statistical inference), an understanding of the sampling distribution of our estimator will be necessary. This brings us to Step 5.

(5) A measure of uncertainty for the estimate $\hat{\Psi}(P_n)$

Since the estimator $\hat{\Psi}$ is a function of the empirical distribution P_n, the estimator itself is a random variable with a sampling distribution. So, if we repeat the experiment of drawing n observations we would every time end up with a different realization of our estimate and our estimator has a sampling distribution.

A primary goal in the construction of estimators is to be able to derive their asymptotic sampling distributions through a theoretical analysis of a given estimator. In this regard, an important property of the estimators on which we focus is their asymptotic linearity, which states that the difference between the estimator and the target estimand (i.e., the truth) can be represented, asymptotically, as an average of i.i.d. random variables:

$$\hat{\Psi}(P_n) - \Psi(P_0) = \frac{1}{n}\sum_{i=1}^{n} IC(O_i;\nu) + o_p(n^{-1/2}),$$

where ν represents possible nuisance parameters on which the influence curve (IC) depends. Based on the validity of the asymptotic approximation, one can then invoke the central limit theorem (CLT) to show

$$\sqrt{n}\left(\hat{\Psi}(P_n) - \Psi(P_0)\right) \sim N(0, \sigma^2_{IC}),$$

where σ^2_{IC} is the variance of $IC(O_i;\nu)$. Given an estimate of σ^2_{IC}, it is then possible to construct classic, *asymptotically accurate* Wald-type confidence intervals (CIs) and hypothesis tests. For example, a standard $(1-\alpha)$ CI of the form

$$\Psi(P_n) \pm Z_{1-\frac{\alpha}{2}} \hat{\sigma_{IC}} / \sqrt{n},$$

can be constructed, where $Z_{1-\frac{\alpha}{2}}$ is the $(1-\frac{\alpha}{2})^{\text{th}}$ quantile of the standard normal distribution. Often, we will be interested in constructing 95% confidence intervals, corresponding to mass $\alpha = 0.05$ in either tail of the limit distribution; thus, we will typically take $Z_{1-\frac{\alpha}{2}} \approx 1.96$.

2.2 Summary of the Roadmap

Data collected across n i.i.d. units, $O_1, \ldots, O_n$, can be viewed as a collection of random variables, O, all arising from the same probability distribution $\mathbb{P}_0$. This collection of data may be expressed $O_1, \ldots, O_n \sim P_0$, where we leverage statistical knowledge available about the experiment that generated the data. to support the statement that the true data distribution P_0 falls in a statistical model, $\mathcal{M}$, which is itself a collection of candidate probability distributions reflecting the data-generating experiment. Often these sets – that is, the statistical model $\mathcal{M}$ – must be very large, to appropriately reflect the fact that statistical knowledge is very limited. Hence, these *realistic* statistical models are often termed *semi-* or *non-parametric*, since they are too large to be indexed by a finite-dimensional set of parameters. Necessarily, our statistical query must begin with, "What are we trying to learn from the data?", a question whose answer is captured by the statistical target parameter, Ψ, which maps the true data-generating distribution P_0 into the statistical estimand, $\Psi(P_0)$. At this point the statistical estimation problem is formally defined, allowing for the use of statistical theory to guide the construction of optimal estimators.

2.3 Causal Target Parameters

In many cases, we are interested in problems that ask questions regarding the *causal* effect of an intervention on a future outcome of interest. These causal effects may be defined as summaries of the population of interest (e.g., the population mean of a particular outcome) under different conditions (e.g., treated versus untreated). For example, a causal effect could be defined as the difference in the means of a disease outcome between *causal contrasts* in which the population were to experience low pollution levels (for some pollutant) and the mean in the same population in the case that high pollution levels were experienced. There are different ways of operationalizing the theoretical experiments that generate the counterfactual data necessary for describing our causal contrasts of interest, including simply assuming that the counterfactual outcomes exist in theory for all treatment contrasts of interest (Neyman, 1938; Rubin, 2005; Imbens and Rubin, 2015) or through considering interventions on directed acyclic graphs (DAGs) or nonparametric structural equation models (NPSEMs) (Pearl, 1995, 2009), both of which encode the known or hypothesized set of relationships between variables in the system under study.

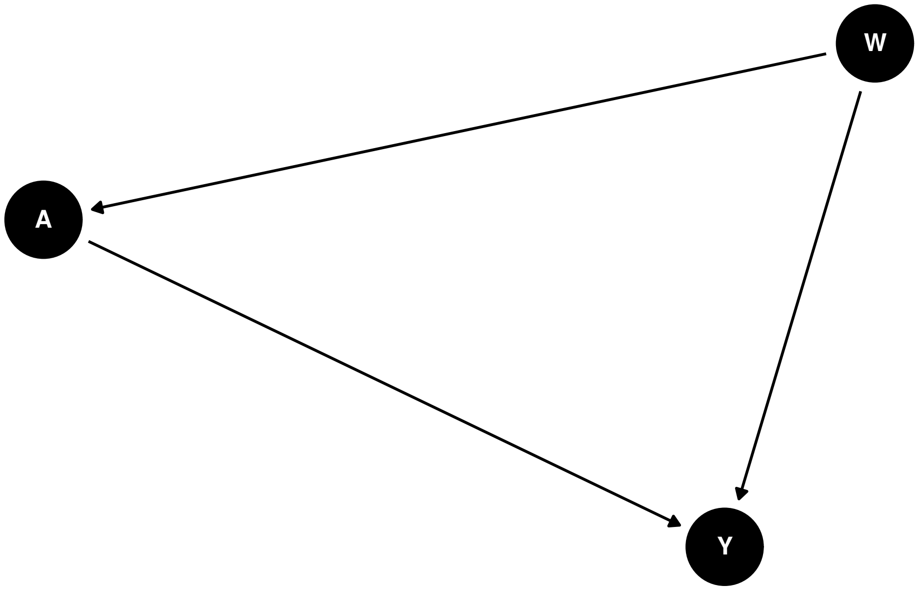
The Causal Model

We focus on the use of DAGs and NPSEMs for the description of causal parameters. Estimators of statistical parameters that correspond, under standard but untestable *identifiability* assumptions, to these causal parameters are introduced below. DAGs are a particularly useful tool for expressing what we know about the causal relations among variables in the system under study. Ignoring exogenous U terms (explained below), we assume the following ordering of the variables in the observed data O. We demonstrate the construction of a DAG below using `DAGitty` (Textor et al., 2011):

```r
library(dagitty)
library(ggdag)

# make DAG by specifying dependence structure
dag <- dagitty(
  "dag {
    W -> A
    W -> Y
    A -> Y
    W -> A -> Y
  }"
)
exposures(dag) <- c("A")
outcomes(dag) <- c("Y")
tidy_dag <- tidy_dagitty(dag)

# visualize DAG
ggdag(tidy_dag) +
  theme_dag()
```



While DAGs like the above provide a convenient means by which to visualize causal relations between variables, the same causal relations among variables can be equivalently represented by an NPSEM:

$$\begin{aligned} W &= f_W(U_W) \\ A &= f_A(W, U_A) \\ Y &= f_Y(W, A, U_Y), \end{aligned}$$

where the f's are unspecified (non-parametric) functions that generate the corresponding random variable as a function of the variable's parents (i.e., nodes with arrows into the variable) in the DAG and the unobserved, exogenous error terms (i.e., the U's). An NPSEM may be thought of as a representation of the algorithm that produces the data, O, in the population of interest. Much of statistics and data science is devoted to discovering properties of this system of equations (e.g., estimation of the prediction function f_Y).

The first hypothetical experiment we will consider is assigning exposure to the entire population and observing the outcome, and then withholding exposure to the same population and observing the outcome. This corresponds to a comparison of the outcome distribution in the population under two interventions:

1. A is set to 1 for all individuals, and
2. A is set to 0 for all individuals.

These interventions imply two new sets of nonparametric structural equations For the case $A = 1$, we have

$$\begin{aligned} W &= f_W(U_W) \\ A &= 1 \\ Y(1) &= f_Y(W, 1, U_Y), \end{aligned}$$

while, for the case $A = 0$,

$$\begin{aligned} W &= f_W(U_W) \\ A &= 0 \\ Y(0) &= f_Y(W, 0, U_Y). \end{aligned}$$

In these equations, A is no longer a function of W because of the intervention on the system that set A deterministically to either of the values 1 or 0. The new symbols $Y(1)$ and $Y(0)$ indicate the outcome variable in the population of interest when it is generated by the respective NPSEMs above; these are often called *counterfactuals*. The difference between the means of the outcome under these two interventions defines a parameter that is often called the "average treatment effect" (ATE), denoted

$$ATE = \mathbb{E}_X(Y(1) - Y(0)), \tag{2.1}$$

where $\mathbb{E}_X$ is the mean under the theoretical (unobserved) full data $X = (W, Y(1), Y(0))$.

Note, we can define much more complicated interventions on NPSEM's, such as interventions based upon rules (themselves based upon covariates), stochastic rules, etc. and each results in a different targeted parameter and entails different identifiability assumptions discussed below.

Identifiability

Because we can never observe both $Y(0)$ (the counterfactual outcome when $A = 0$) and $Y(1)$ (similarly, the counterfactual outcome when $A = 1$), we cannot estimate the quantity in Equation (2.1) directly. Thus, the primary task of causal inference methods in our context is to *identify* the assumptions necessary to express causal quantities of interest as functions of the data-generating distribution. We have to make assumptions under which this quantity may be estimated from the observed data $O \sim P_0$ under the data-generating distribution P_0. Fortunately, given the causal model specified in the NPSEM above, we can, with a handful of untestable assumptions, estimate the ATE from observational data. These assumptions may be summarized as follows.

1. *No unmeasured confounding*: $A \perp Y(a) \mid W$ for all $a \in \mathcal{A}$, which states that the potential outcomes $(Y(a) : a \in \mathcal{A})$ arise independently from exposure status A, conditional on the observed covariates W. This is the analog of the *randomization* assumption in data arising from natural experiments, ensuring that the effect of A on Y can be disentangled from that of W on Y, even though W affects both.
2. *No interference* between units: the outcome for unit i, Y_i, cannot be affected by the exposure of unit j, A_j, for all $i \neq j$.
3. *Consistency* of the treatment mechanism is also required, i.e., the outcome for unit i is $Y_i(a)$ whenever $A_i = a$, an assumption also known as "no other versions of treatment".
4. *Positivity* or *overlap*: All observed units, across strata defined by W, must have a bounded (non-deterministic) probability of receiving treatment – that is, $0 < \mathbb{P}(A = a \mid W) < 1$ for all a and W).

Given these assumptions, the ATE may be re-written as a function of P_0, specifically

$$ATE = \mathbb{E}_0(Y(1) - Y(0)) = \mathbb{E}_0\left(\mathbb{E}_0[Y \mid A = 1, W] - \mathbb{E}_0[Y \mid A = 0, W]\right). \tag{2.2}$$

In words, the ATE is the difference in the predicted outcome values for each subject, under the contrast of treatment conditions ($A = 0$ versus $A = 1$), in the population, averaged over all observations. Thus, a parameter of a theoretical "full" data distribution can be represented as an estimand of the observed data distribution. Significantly, there is nothing about the representation in Equation (2.2) that requires parameteric assumptions; thus, the regressions on the right hand side may be estimated. With different parameters, there will be potentially different identifiability assumptions and the resulting estimands can be functions of different components of P_0. We discuss several more complex estimands in later sections.

3

Welcome to the `tlverse`

Updated: 2021-12-30

Learning Objectives

This chapter introduces the ‘tlverse‘ software ecosystem, including

1. Understanding the ‘tlverse‘ ecosystem conceptually. 2. Identifying the core components of the ‘tlverse‘. 3. Installing ‘tlverse‘ ‘R‘ packages. 4. Understanding the Targeted Learning roadmap. 5. Learning about the WASH Benefits example data.

What is the `tlverse`?

The `tlverse` is a new framework for doing Targeted Learning in R, inspired by the `tidyverse` ecosystem of R packages.

By analogy to the `tidyverse`:

> The `tidyverse` is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

So, the `tlverse` is

- an opinionated collection of R packages for Targeted Learning
- sharing an underlying philosophy, grammar, and set of data structures

Anatomy of the `tlverse`

All Targeted Learning methods are targeted maximum likelihood (or minimum loss-based) estimators (TMLEs). The construction of any Targeted Learning estimator proceeds through a two-stage process:

1. Flexibly learning particular components of the data-generating distribution through macchine learning (e.g., Super Learning), resulting in *initial estimates* of nuisance parameters.
2. Use of a parametric model-based update via maximum likelihood estimation (i.e., MLE), incorporating the initial estimates produced by the prior step.

The packages making up the core components of the `tlverse` software ecosystem, `sl3` and `tmle3`, address the above two goals, respectively. Together, the very general functionality exposed by both allows one to build specific TMLEs tailored exactly to a particular estimation problem.

The software packages that make up the **core** of the `tlverse` are

- `sl3`: Modern Super Machine Learning
  - *What?* A modern object-oriented re-implementation of the Super Learner algorithm, employing recently developed paradigms in `R` programming.
  - *Why?* A design that leverages modern ideas for faster computation, is easily extensible and forward-looking, and forms one of the cornerstones of the `tlverse`.
- `tmle3`: An Engine for Targeted Learning
  - *What?* A generalized framework that simplifies Targeted Learning by identifying and implementing a series of common statistical estimation procedures.
  - *Why?* A common interface and engine that accommodates current algorithmic approaches to Targeted Learning and yet remains a flexible enough engine to power the implementation of emerging statistical techniques as they are developed.

Beyond these engines that provide the driving force behind the `tlverse`, there are a few supporting packages that play important roles in the background:

- `origami`: A Generalized Framework for Cross-Validation (Coyle and Hejazi, 2018)
    - *What?* A generalized framework for flexible cross-validation.
    - *Why?* Cross-validation is a key part of ensuring error estimates are honest and in preventing overfitting. It is an essential part of the both the Super Learner ensemble modeling algorithm and in the construction of Targeted Learning estimators.
- `delayed`: Parallelization Framework for Dependent Tasks
    - *What?* A framework for delayed computations (i.e., futures) based on task dependencies.
    - *Why?* Efficient allocation of compute resources is essential when deploying computationally intensive algorithms at large scale.

A key principle of the `tlverse` is extensibility. That is, the software ecosystem aims to support the development of new Targeted Learning estimators as they reaching maturity. To achieve this degree of flexibility, we follow the model of implementing new classes of estimators, for distinct causal inference problems, in separate packages, all of which use the core machinery provided by the `sl3` and `tmle3` packages There are currently three examples:

- `tmle3mopttx`: Optimal Treatments in the `tlverse`
    - *What?* Learn an optimal rule and estimate the mean outcome under the rule.
    - *Why?* Optimal treatments are a powerful tool in precision healthcare and other settings where a one-size-fits-all treatment approach is not appropriate.
- `tmle3shift`: Stochastic Shift Interventions in the `tlverse`
    - *What?* Stochastic shift interventions for continuous-valued treatments.
    - *Why?* Not all treatment variables are binary or categorical. Estimating the total effects of intervening on continuous-valued treatments provides a way to probe how an effect changes with shifts in the treatment variable.
- `tmle3mediate`: Causal Mediation Analysis in the `tlverse`
    - *What?* Techniques for evaluating the direct and indirect effects of treatments through mediating variables.
    - *Why?* Evaluating the total effect of a treatment does not provide information about the pathways through which it may operate. When mediating variables have been collected, one can instead evaluate direct and indirect effect parameters that speak to the *action mechanism* of the treatment.

3.1 Installation

The tlverse ecosystem of packages are currently hosted at https://github.com/tlverse, not yet on CRAN. You can use the usethis package to install them:

```
install.packages("devtools")
devtools::install_github("tlverse/tlverse")
```

The tlverse depends on a large number of other packages that are also hosted on GitHub. Because of this, you may see the following error:

```
Error: HTTP error 403.
  API rate limit exceeded for 71.204.135.82. (But here's the good news:
  Authenticated requests get a higher rate limit. Check out the documentation
  for more details.)

  Rate limit remaining: 0/60
  Rate limit reset at: 2019-03-04 19:39:05 UTC

  To increase your GitHub API rate limit
  - Use `usethis::browse_github_pat()` to create a Personal Access Token.
  - Use `usethis::edit_r_environ()` and add the token as `GITHUB_PAT`.
```

This just means that R tried to install too many packages from GitHub in too short of a window. To fix this, you need to tell R how to use GitHub as your user (you'll need a GitHub user account). Follow these two steps:

1. Type `usethis::browse_github_pat()` in your R console, which will direct you to GitHub's page to create a New Personal Access Token (PAT).
2. Create a PAT simply by clicking "Generate token" at the bottom of the page.
3. Copy your PAT, a long string of lowercase letters and numbers.
4. Type `usethis::edit_r_environ()` in your R console, which will open your `.Renviron` file in the source window of RStudio.

    a. If your `.Renviron` file does not pop-up after calling `usethis::edit_r_environ()`; then try inputting `Sys.setenv(GITHUB_PAT = "yourPAT")`, replacing your PAT with inside the quotes. If this does not error, then skip to step 8.

5. In your `.Renviron` file, type `GITHUB_PAT=` and then paste your PAT after the equals symbol with no space.

6. In your `.Renviron` file, press the enter key to ensure that your `.Renviron` ends with a new line.

7. Save your `.Renviron` file. The example below shows how this syntax should look.

```
GITHUB_PAT <- yourPAT
```

8. Restart R. You can restart R via the drop-down menu on RStudio's "Session" tab, which is located at the top of the RStudio interface. You have to restart R for the changes to take effect!

After following these steps, you should be able to successfully install the package which threw the error above.

4

Meet the Data

4.1 WASH Benefits Example Dataset

The data come from a study of the effect of water quality, sanitation, hand washing, and nutritional interventions on child development in rural Bangladesh (WASH Benefits Bangladesh): a cluster randomized controlled trial (Tofail et al., 2018). The study enrolled pregnant women in their first or second trimester from the rural villages of Gazipur, Kishoreganj, Mymensingh, and Tangail districts of central Bangladesh, with an average of eight women per cluster. Groups of eight geographically adjacent clusters were block randomized, using a random number generator, into six intervention groups (all of which received weekly visits from a community health promoter for the first 6 months and every 2 weeks for the next 18 months) and a double-sized control group (no intervention or health promoter visit). The six intervention groups were:

1. chlorinated drinking water;
2. improved sanitation;
3. hand-washing with soap;
4. combined water, sanitation, and hand washing;
5. improved nutrition through counseling and provision of lipid-based nutrient supplements; and
6. combined water, sanitation, handwashing, and nutrition.

In the handbook, we concentrate on child growth (size for age) as the outcome of interest. For reference, this trial was registered with ClinicalTrials.gov as NCT01590095.

```
library(readr)
# read in data via readr::read_csv
dat <- read_csv(
  paste0(
    "https://raw.githubusercontent.com/tlverse/tlverse-data/master/",
    "wash-benefits/washb_data.csv"
  )
)
```

For the purposes of this handbook, we start by treating the data as independent and identically distributed (i.i.d.) random draws from a very large target population. We could, with available options, account for the clustering of the data (within sampled geographic units), but, for simplification, we avoid these details in the handbook, although modifications of our methodology for biased samples, repeated measures, and related complications, are available.

We have 28 variables measured, of which a single variable is set to be the outcome of interest. This outcome, Y, is the weight-for-height Z-score (`whz` in `dat`); the treatment of interest, A, is the randomized treatment group (`tr` in `dat`); and the adjustment set, W, consists simply of *everything else*. This results in our observed data structure being n i.i.d. copies of $O_i = (W_i, A_i, Y_i)$, for $i = 1, \ldots, n$.

Using the `skimr` package, we can quickly summarize the variables measured in the WASH Benefits data set:

skim_type	skim_variable	n_missing	complete_rate	character.min	character.max	character.empty	chara
character	tr	0	1.00000	3	15	0	
character	fracode	0	1.00000	2	6	0	
character	sex	0	1.00000	4	6	0	
character	momedu	0	1.00000	12	15	0	
character	hfiacat	0	1.00000	11	24	0	
numeric	whz	0	1.00000	NA	NA	NA	
numeric	month	0	1.00000	NA	NA	NA	
numeric	aged	0	1.00000	NA	NA	NA	
numeric	momage	18	0.99617	NA	NA	NA	
numeric	momheight	31	0.99340	NA	NA	NA	
numeric	Nlt18	0	1.00000	NA	NA	NA	
numeric	Ncomp	0	1.00000	NA	NA	NA	
numeric	watmin	0	1.00000	NA	NA	NA	
numeric	elec	0	1.00000	NA	NA	NA	
numeric	floor	0	1.00000	NA	NA	NA	
numeric	walls	0	1.00000	NA	NA	NA	
numeric	roof	0	1.00000	NA	NA	NA	
numeric	asset_wardrobe	0	1.00000	NA	NA	NA	
numeric	asset_table	0	1.00000	NA	NA	NA	
numeric	asset_chair	0	1.00000	NA	NA	NA	
numeric	asset_khat	0	1.00000	NA	NA	NA	
numeric	asset_chouki	0	1.00000	NA	NA	NA	
numeric	asset_tv	0	1.00000	NA	NA	NA	
numeric	asset_refrig	0	1.00000	NA	NA	NA	
numeric	asset_bike	0	1.00000	NA	NA	NA	
numeric	asset_moto	0	1.00000	NA	NA	NA	
numeric	asset_sewmach	0	1.00000	NA	NA	NA	
numeric	asset_mobile	0	1.00000	NA	NA	NA	

A convenient summary of the relevant variables is given just above, complete with a small visualization describing the marginal characteristics of each covariate. Note that the *asset* variables reflect socio-economic status of the study participants. Notice also the uniform distribution of the treatment groups (with twice as many controls); this is, of course, by design.

5

Cross-validation

Ivana Malenica

Based on the `origami` `R` package by *Jeremy Coyle, Nima Hejazi, Ivana Malenica and Rachael Phillips*.

Updated: 2021-12-30

Learning Objectives

By the end of this chapter you will be able to:

1. Differentiate between training, validation and test sets.

2. Understand the concept of a loss function, risk and cross-validation.

3. Select a loss function that is appropriate for the functional parameter to be estimated.

4. Understand and contrast different cross-validation schemes for i.i.d. data.

5. Understand and contrast different cross-validation schemes for time dependent data.

6. Setup the proper fold structure, build custom fold-based function, and cross-validate the proposed function using the ‘origami‘ ‘R‘ package.

7. Setup the proper cross-validation structure for the use by the Super Learner using the the ‘origami‘ ‘R‘ package.

5.1 Introduction

In this chapter, we start elaborating on the estimation step outlined in the introductory chapter, which discussed the *Roadmap for Targeted Learning*. In order to generate an initial estimate of our target parameter – which is the focus of the following chapter on Super Learning, we first need to translate, and incorporate, our knowledge about the data generating process into the estimation procedure, and decide how to evaluate our estimation performance.

The performance, or error, of any algorithm used in the estimation procedure directly relates to its generalizability on the independent data. The proper assessment of the performance of proposed algorithms is extremely important; it guides the choice of the final learning method, and it gives us a quantitative assessment of how good the chosen algorithm is doing. In order to assess the performance of an algorithm, we introduce the concept of a **loss** function, which helps us define the **risk**, also referred to as the **expected prediction error**.

Constructing a library that is consistent with the data-generating distribution

Our goal, as further specified in the next chapter, will be to estimate the true risk of the proposed statistical learning method. Our goal(s) consist of:

1. Estimating the performance of different algorithms in order to choose the best one. 2. Having chosen a winner, estimate the true risk of the proposed statistical learning method.

In the following, we propose a method to do so using the observed data and **cross-validation** procedure using the `origami` package (Coyle and Hejazi, 2018).

5.2 Background

Ideally, in a data-rich scenario, we would split our dataset into three parts:

1. training set,
2. validation set,
3. test set.

The training set is used to fit algorithm(s) of interest; we evaluate the performance of the fit(s) on a validation set, which can be used to estimate prediction error (e.g., for tuning and model selection). The final error of the chosen algorithm(s) is obtained by using the test set, which is kept separately, and doesn't see the data before the final evaluation. One might wonder, with training data readily available, why not use the training error to evaluate the proposed algorithm's performance? Unfortunately, the training error is not a good estimate of the true risk; it consistently decreases with model complexity, resulting in a possible overfit to the training data and low generalizability.

Since data are often scarce, separating it into training, validation and test set is usually not possible. In the absence of a large data set and a designated test set, we must resort to methods that estimate the true risk by efficient sample re-use. Re-sampling methods, in great generality, involve repeatedly sampling from the training set and fitting proposed algorithms on the new samples. While often computationally intensive, re-sampling methods are particularly useful for model selection and estimation of the true risk. In addition, they might provide more insight on variability and robustness of the algorithm fit then fitting an algorithm only once on all the training data.

5.2.1 Introducing: cross-validation

In this chapter, we focus on **cross-validation** – an essential tool for evaluating how any given algorithm extends from a sample to the target population from which the sample is derived. It has seen widespread application in all facets of statistics, perhaps most notably statistical machine learning. The cross-validation procedure can be used for model selection, as well as for estimation of the true risk associated with any statistical learning method in order to evaluate its performance. It particular, cross-validation directly estimates the true risk when the estimate is applied to an independent sample from the joint distribution of the predictors and outcome. When used for model selection, cross-validation has powerful optimality properties. The asymptotic optimality results state that the cross-validated selector performs (in terms of risk) asymptotically as well as an optimal oracle selector based on the true, unknown data generating distribution. For further details on the theoretical results, we suggest van der Laan and Dudoit (2003), van der Laan et al. (2004), Dudoit and van der Laan (2005) and Van der Vaart et al. (2006).

In great generality, cross-validation works by partitioning a sample into complementary subsets, applying a particular algorithm(s) on a subset (the training set), and evaluating the method of choice on the complementary subset (the validation/test set). This procedure is repeated across multiple partitions of the data. A variety of different partitioning schemes exist, depending on the problem of interest, data size, prevalence of the outcome, and dependence structure. The `origami` package provides a suite of tools that generalize the application of cross-validation to arbitrary data analytic procedures. In the the following, we describe different types of cross-validation schemes readily available in `origami`, introduce the general structure of the `origami` package, and show their use in applied settings.

5.3 Estimation Roadmap: how does it all fit together?

Similarly to how we defined the *Roadmap for Targeted Learning*, we can define the **Estimation Roadmap** to guide the estimation process. In particular, we have developed a unified loss-based cross-validation methodology for estimator construction, selection, and performance assessment in a series of articles (e.g., see van der Laan and Dudoit (2003), van der Laan et al. (2004), Dudoit and van der Laan (2005), Van der Vaart et al. (2006), and van der Laan et al. (2007)) that follow three main steps:

1. **The loss funtion**: Define the target parameter as the minimizer of the expected loss (risk) for a full data loss function chosen to represent the desired performance measure. Map the full data loss function into an observed data loss function, having the same expected value and leading to an efficient estimator of risk.
2. **The algorithms**: Construct a finite collection of candidate estimators for the parameter of interest.
3. **The cross-validation scheme**: Apply appropriate cross-validation to select an optimal estimator among the candidates, and assess the overall performance of the resulting estimator.

Step 1 of the Estimation Roadmap allows us to unify a broad range of problems that are traditionally treated separately in the statistical literature, including density estimation, prediction of polychotomous and continuous outcomes. For example, if we are interested in estimating the full joint conditional density, we could use the negative log-likelihood loss. If instead we are interested in the conditional mean with continuous outcome, one could use the squared error loss; had the outcome been binary, one could resort to the indicator (0-1) loss. The unified loss-based framework also reconciles censored and full data estimation methods, by generalizing any loss based learning for full data into loss based learning for general censored data.

5.4 Example: cross-validation and prediction

Now that we introduced the Estimation Roadmap, we can define our objective with more mathematical notation, using prediction as an example. Let the observed data be defined as $X = (W, Y)$, where a unit specific data can be written as $X_i = (W_i, Y_i)$, for $i = 1, \ldots, n$. For each of the n samples, we denote Y_i as the outcome of interest (polychotomous or continuous), and W_i as a p-dimensional set of covariates. Let $\psi_0(W)$ denote the target parameter of interest we want to estimate; for this example, we are interested in estimating the conditional expectation of the outcome given the covariates, $\psi_0(W) = E(Y \mid W)$. Following the Estimation Roadmap, we chose the appropriate loss function, L, such that $\psi_0(W) = \text{argmin}_{\psi} E[L(X, \psi(W))]$. But how do we know how each ψ is doing? In order to pick the optimal estimator among the candidates, and assess the overall performance of the resulting estimator, we use cross-validation – dividing the available data into the training set and validation set. Observations in the training set are used to fit (or train) the estimator, while the validation set is used to assess the risk of (or validate) it.

To derive a general representation for cross-validation, we define a **split vector**, $B_n = (B_n(i) : i = 1, \ldots, n) \in \{0, 1\}^n$. Note that split vector is independent of the empirical distribution, P_n. A realization of B_n defines a random split of the data into a training and validation set such that if

$$B_n(i) = 0, \ \ \text{i sample is in the training set}$$
$$B_n(i) = 1, \ \ \text{i sample is in the validation set.}$$

We can further define P^0_{n,B_n} and P^1_{n,B_n} as the empirical distributions of the training and validation sets, respectively. Then $n_0 = \sum_i (1 - B_n(i))$ and $n_1 = \sum_i B_n(i)$ denote the number of samples in each set. The particular distribution of the split vector B_n defines the type of cross-validation scheme, tailored to the problem and data set in hand.

5.5 Cross-validation schemes in `origami`

As we specified earlier, the particular distribution of the split vector B_n defines the type of the cross-validation method. In the following, we describe different types of cross-validation schemes available in `origami` package, and show their use in the sequel.

WASH Benefits Study Example

In order to illustrate different cross-validation schemes, we will be using the WASH data. Detailed information on the WASH Benefits Example Dataset can be found in Chapter 3. In particular, we are interested in predicting weight-for-height z-score `whz` using the available covariate data. For this illustration, we will start by treating the data as independent and identically distributed (i.i.d.) random draws. To see what each cross-validation scheme is doing, we will subset the data to only $n = 30$. Note that each row represents an i.i.d. sample, indexed by the row number.

```
library(data.table)
library(origami)
library(knitr)
library(kableExtra)

# load data set and take a peek
washb_data <- fread(
  paste0(
    "https://raw.githubusercontent.com/tlverse/tlverse-data/master/",
    "wash-benefits/washb_data.csv"
  ),
  stringsAsFactors = TRUE
)
```

whz	tr	fracode	month	aged	sex	momage	momedu	momheight	hfiacat
0.00	Control	N05265	9	268	male	30	Primary (1-5y)	146.40	Food Secu
-1.16	Control	N05265	9	286	male	25	Primary (1-5y)	148.75	Moderatel
-1.05	Control	N08002	9	264	male	25	Primary (1-5y)	152.15	Food Secu
-1.26	Control	N08002	9	252	female	28	Primary (1-5y)	140.25	Food Secu
-0.59	Control	N06531	9	336	female	19	Secondary (¿5y)	150.95	Food Secu
-0.51	Control	N06531	9	304	male	20	Secondary (¿5y)	154.20	Severely F

Above is a look at the first 30 of the data.

5.5.1 Cross-validation for i.i.d. data

5.5.1.1 Re-substitution

The re-substitution method is the simplest strategy for estimating the risk associated with fitting a proposed algorithm on a set of observations. Here, all observed data is used for both training and validation set.

We illustrate the usage of the re-substitution method with `origami` package below; we will use the function `folds_resubstitution(n)`. In order to setup `folds_resubstitution(n)`, we just need the total number of samples we want to allocate to training and validation sets; remember that each row of data is a unique i.i.d. sample. Notice the structure of the `origami` output:

1. **v:** the cross-validation fold
2. **training_set:** the indexes of the samples in the training set
3. **validation_set:** the indexes of the samples in the training set.

This structure of the `origami` output (aka, fold(s)) will persist for each of the cross-validation schemes we present in this chapter. Below, we show the fold generated by the re-substitution method:

```
folds_resubstitution(nrow(washb_data))
[[1]]
$v
[1] 1

$training_set
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26 27 28 29 30

$validation_set
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26 27 28 29 30

attr(,"class")
[1] "fold"
```

5.5.1.2 Holdout method

The holdout method, or the validation set approach, consists of randomly dividing the available data into the training set and validation set (holdout set). The model is then fitted on the training set, and further evaluated on the observations in the validation set. Typically, the data is split into $60/40$, $70/30$, $80/20$ or $90/10$ splits.

The holdout method is intuitive, conceptually easy, and computationally not too demanding. However, if we repeat the process of randomly splitting the data into the training and validation set, we might get a very different cross-validated emprical risk. In particular, the emprical mean of the loss over the validation sets might be highly variable, depending on which samples were included in the training/validation split. Overall, the cross-validated emprical risk for the holdout method is more variabiable, since in includes variability of the random split as well - which is not what we want. For classification problems, there is a possibility of an uneven distribution of different classes in the training and validation set unless data is stratified. Finally, note that we are not using all of the data to train and evaluate the performance of the proposed algorithm, which might result in bias.

5.5.1.3 Leave-one-out

The leave-one-out cross-validation scheme is closely related to the holdout method. In particular, it also involves splitting the data into the training and validation set; however, instead of partitioning the observed data into sets of similar size, a single observation is used as a validation set. With that, majority of the units are employed for training (fitting) the proposed algorithm. Since only one unit (for example $x_1 = (w_1, y_1)$) is not used in the fitting process, leave-one-out cross-validation results in a possibly less biased estimate of the true risk; typically, leave-one-out approach will not overestimate the risk as much as the holdout method. On the other hand, since the estimate of risk is based on a single sample, it is typically a highly variable estimate.

We can repeat the process of spiting the data into training and validation set until all samples are part of the validation set at some point. For example, next iteration of the cross-validation might have $x_2 = (w_2, y_2)$ as the validation set and all the rest of $n - 1$ samples as the training set. Repeating this approach n times results in, for example, n squared errors $MSE_1, MSE_2, \dots, MSE_n$. The estimate of the true risk is the average over the n squared errors. While the leave-one-out cross-validation results in a less biased (albeit, more variable) estimate of risk than the holdout method, it could be expensive to implement if n is large.

We illustrate the usage of the leave-one-out cross-validation with `origami` package below; we will use the function `folds_loo(n)`. In order to setup `folds_loo(n)`, similarly to the re-substitution method, we just need the total number of samples we want to cross-validate. We show the first two folds generated by the leave-one-out cross-validation below.

```
folds <- folds_loo(nrow(washb_data))
folds[[1]]
$v
[1] 1

$training_set
 [1]  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
[26] 27 28 29 30
```

```
$validation_set
[1] 1

attr(,"class")
[1] "fold"
folds[[2]]
$v
[1] 2

$training_set
 [1]  1  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
[26] 27 28 29 30

$validation_set
[1] 2

attr(,"class")
[1] "fold"
```

5.5.1.4 V-fold

An alternative to leave-one-out is V-fold cross-validation. This cross-validation scheme randomly divides the data into v sets (folds) of equal size; for each fold, the number of samples in the validation set are the same. For V-fold cross-validation, one of the folds is treated as a validation set, whereas the proposed algorithm is fit on the remaining $v - 1$ folds in the training set. The loss, for example MSE, is computed on the samples in the validation set. With the proposed algorithm trained and its performance evaluated on the first fold, we repeat this process v times; each time, a different group of samples is treated as a validation set. Note that with V-fold cross-validation we effectively use all of the data to train and evaluate the proposed algorithm without overfitting to the training data. In the end, the V-fold cross-validation results in v estimates of validation error. The final V-fold CV estimate is computed as an average over all the validation losses.

For a dataset with n samples, V-fold cross-validation with $v = n$ is just leave-one-out; similarly, if we set $n = 1$, we can get the holdout method's estimate of algorithm's performance. Despite the obvious computational advantages, V-fold cross-validation often gives more accurate estimates of the true risk. The reason for this comes from the bias-variance trade-off that comes from employing both methods; while leave-one-out might be less biased, it has higher variance. This difference becomes more obvious as $v << n$ (but not too small, as then we increase bias). With V-fold cross-validation, we end up averaging output from v fits that are typically less correlated than the outputs from leave-one-out fits. Since the mean of many highly correlated quantities has higher variance, leave-one-out estimate of the risk will have higher variance than the estimate based on V-fold cross-validation.

Let's see V-fold cross-validation with `origami` in action! In the next chapter we will study the Super Learner — an actual algorithm that we fit and evaluate its performance. The Super Learner relies on V-fold cross-validation as default cross-validation scheme. In order to set up V-fold CV, we need to call function `folds_vfold(n, V)`. Arguments for `folds_vfold(n, V)` require the total number of samples to be cross-validated, and the number of folds we want to get.

At $V = 2$, we get 2 folds with $n/2$ number of samples in both training and validation set.

```
folds <- folds_vfold(nrow(washb_data), V = 2)
folds[[1]]
$v
[1] 1

$training_set
 [1]  2  3  4  6  7  8 11 12 14 15 19 22 23 24 28

$validation_set
 [1]  1  5  9 10 13 16 17 18 20 21 25 26 27 29 30

attr(,"class")
[1] "fold"
folds[[2]]
$v
[1] 2

$training_set
 [1]  1  5  9 10 13 16 17 18 20 21 25 26 27 29 30

$validation_set
 [1]  2  3  4  6  7  8 11 12 14 15 19 22 23 24 28

attr(,"class")
[1] "fold"
```

5.5.1.5 Monte Carlo

With Monte Carlo cross-validation, we randomly select some fraction of the data (without replacement) to form the training set; we assign the rest of the samples to the validation set. With that, the data is repeatedly and randomly divided into two sets, a training set of $n_0 = n \cdot (1 - p)$ observations and a validation set of $n_1 = n \cdot p$ observations. This process is then repeated multiple times, generating (at random) new training and validation partitions each time.

Since the partitions are independent across folds, the same sample can appear in the validation set multiple times – note that this is a stark difference between Monte Carlo and V-fold cross-validation. For a given p, Monte Carlo cross-validation would be optimal if done infinite times, but this is not computationally feasible. With Monte Carlo cross-validation, one is able to explore many more available partitions than with V-fold cross-validation – resulting in a possibly less variable estimate of the risk, at a cost of an increase in bias. By having many overlapping splits, we often also need more splits (and thus more computational time) to achieve V-fold performance with only V splits.

We illustrate the usage of the Monte Carlo cross-validation with `origami` package below using the function `folds_montecarlo(n, V, pvalidation)`. In order to setup `folds_montecarlo(n, V, pvalidation)`, we need:

1. the total number of samples we want to cross-validate;
2. the number of folds;
3. the proportion of observations to be placed in the validation set.

At $V = 2$ and $pvalidation = 0.2$, we obtain 2 folds with approximately 6 samples in validation set per fold.

```
folds <- folds_montecarlo(nrow(washb_data), V = 2, pvalidation = 0.2)
folds[[1]]
$v
[1] 1

$training_set
 [1] 19 27 16 29 23 12  1  3 18 11  5  7  8  6  9 22 10 25 20 28 15  2 24 26

$validation_set
[1]  4 13 14 17 21 30

attr(,"class")
[1] "fold"
folds[[2]]
$v
[1] 2

$training_set
 [1] 19 15 28 25 29 11 20 17 14  4  9 12 30  8 27 18 16 10 13  6 24  3 26  1
```

```
$validation_set
[1]  2  5  7 21 22 23

attr(,"class")
[1] "fold"
```

5.5.1.6 Bootstrap

The bootstrap cross-validation also consists of randomly selecting samples, with replacement, for the training set. The rest of the samples not picked for the training set are allocated to the validation set. This process is then repeated multiple times, generating (at random) new training and validation partitions each time. In contract to the Monte Carlo cross-validation, the total number of samples in a training and validation size across folds is not constant. We also sample with replacement, hence the same samples can be in multiple training sets. The proportion of observations in the validation sets is a random variable, with expectation ~ 0.368.

We illustrate the usage of the bootstrap cross-validation with `origami` package below using the function `folds_bootstrap(n, V)`. In order to setup `folds_bootstrap(n, V)`, we need:

1. the total number of samples we want to cross-validate;
2. the number of folds.

At $V = 2$, we obtain 2 folds with different number of samples in the validation set across folds.

```
folds <- folds_bootstrap(nrow(washb_data), V = 2)
folds[[1]]
$v
[1] 1

$training_set
 [1]  2  5 30  1 29 16 10 11  8 25 28  2 11  2 16 28 15 28  1 27  9 19 20 30 18
[26] 11 13  2 18 12

$validation_set
 [1]  3  4  6  7 14 17 21 22 23 24 26

attr(,"class")
[1] "fold"
folds[[2]]
```

```
$v
[1] 2

$training_set
 [1] 12 16 10 29 22 15 27  9 27 16 12 28 10 28 26  1 14  6 23 14 21 16  5 20  8
[26] 23 25  8 27  5

$validation_set
 [1]  2  3  4  7 11 13 17 18 19 24 30

attr(,"class")
[1] "fold"
```

5.5.2 Cross-validation for dependent data

The origami package also supports numerous cross-validation schemes for time-series data, for both single and multiple time-series with arbitrary time and network dependence.

AirPassenger Example

In order to illustrate different cross-validation schemes for time-series, we will be using the AirPassenger data; this is a widely used, freely available dataset. The AirPassenger dataset in R provides monthly totals of international airline passengers from 1949 to 1960. This dataset is already of a time series class therefore no further class or date manipulation is required.

Constructing a library that is consistent with the data-generating distribution

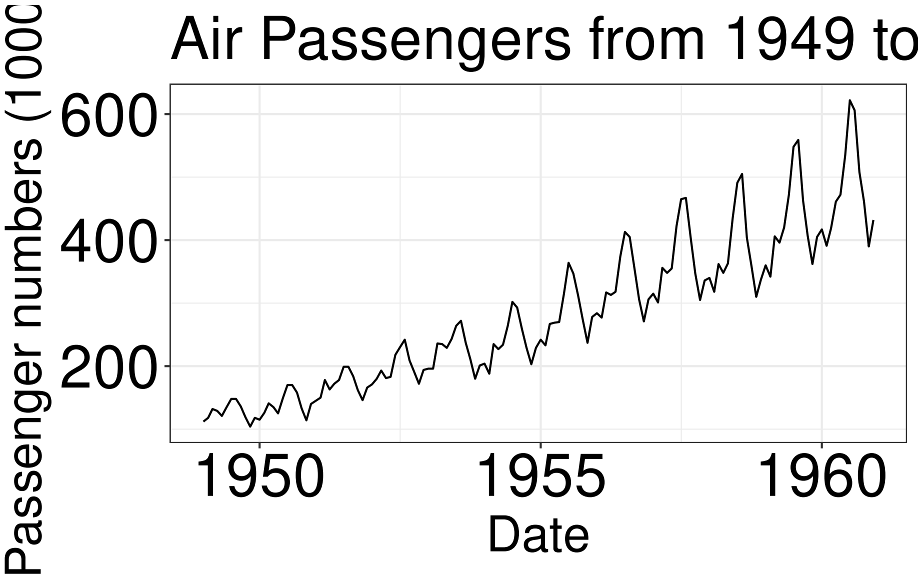
Goal: we want to forecast the number of airline passengers at time h horizon using the historical data from 1949 to 1960.

```
library(ggfortify)

data(AirPassengers)
AP <- AirPassengers

autoplot(AP) +
  labs(
    x = "Date",
    y = "Passenger numbers (1000's)",
    title = "Air Passengers from 1949 to 1961"
  )

t <- length(AP)
```



5.5.2.1 Rolling origin

Rolling origin cross-validation scheme lends itself to "online" algorithms, where large streams of data have to be fit continually, and the final fit is constantly updated with more data acquired. In general, the rolling origin scheme defines an initial training set, and with each iteration the size of the training set grows by m observations until we reach time t for a particular fold. The time points included in the training set are always behind the validation set time points; in addition, there might be a gap between training and validation times of size h.

To further illustrate rolling origin cross-validation, we show below an example with 3 folds. Here, the first window size is 15 time points, on which we first train the proposed algorithm. We then evaluate its performance on 10 time points, with a gap of size 5 between the training and validation time points. For the following fold, we train the algorithm on a longer stream of data, 25 time points, including the original 15 we started with. We then evaluate its performance on 10 time points in the future.

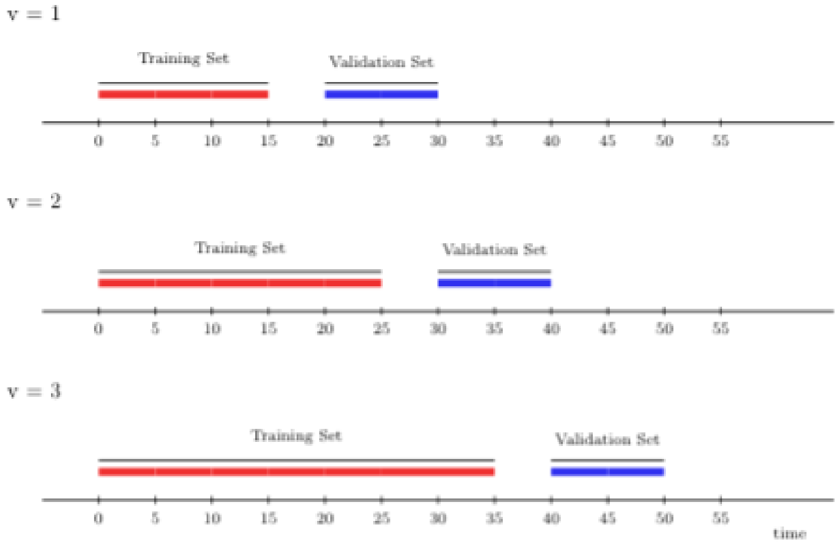


Figure 5.1: Rolling origin CV

We illustrate the usage of the rolling origin cross-validation with `origami` package below using the function `folds_rolling_origin(n, first_window, validation_size, gap, batch)`. In order to setup `folds_rolling_origin(n, first_window, validation_size, gap, batch)`, we need:

1. the total number of time points we want to cross-validate;
2. the size of the first training set;
3. the size of the validation set;
4. the gap between training and validation set;
5. the size of the update on the training set per each iteration of CV.

Our time-series has $t = 144$ time points. Setting the `first_window` to 50, `validation_size` to 10, `gap` to 5 and `batch` to 20, we get 4 time-series folds; we show the first two below.

```
folds <- folds_rolling_origin(
  t,
  first_window = 50, validation_size = 10, gap = 5, batch = 20
)
folds[[1]]
$v
```

```
[1] 1

$training_set
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50

$validation_set
 [1] 56 57 58 59 60 61 62 63 64 65

attr(,"class")
[1] "fold"
folds[[2]]
$v
[1] 2

$training_set
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
[51] 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70

$validation_set
 [1] 76 77 78 79 80 81 82 83 84 85

attr(,"class")
[1] "fold"
```

5.5.2.2 Rolling window

Instead of adding more time points to the training set per each iteration, the rolling window cross-validation scheme “rolls” the training sample forward by m time units. The rolling window scheme might be considered in parametric settings when one wishes to guard against moment or parameter drift that is difficult to model explicitly; it is also more efficient for computationally demanding settings such as streaming data, in which large amounts of training data cannot be stored. In contrast to rolling origin CV, the training sample for each iteration of the rolling window scheme is always the same.

To illustrate the rolling window cross-validation with 3 time-series folds below. The first window size is 15 time points, on which we first train the proposed algorithm. As in the previous illustration, we evaluate its performance on 10 time points, with a gap of size 5 between the training and validation time points. However, for the next fold, we train the algorithm on time points further away from the origin (here, 10 time points). Note that the size of the training set in the new fold is the same as in the first fold (15 time points). This setup keeps the training sets comparable over time (and fold) as compared to the rolling origin CV. We then evaluate the performance of the proposed algorithm on 10 time points in the future.

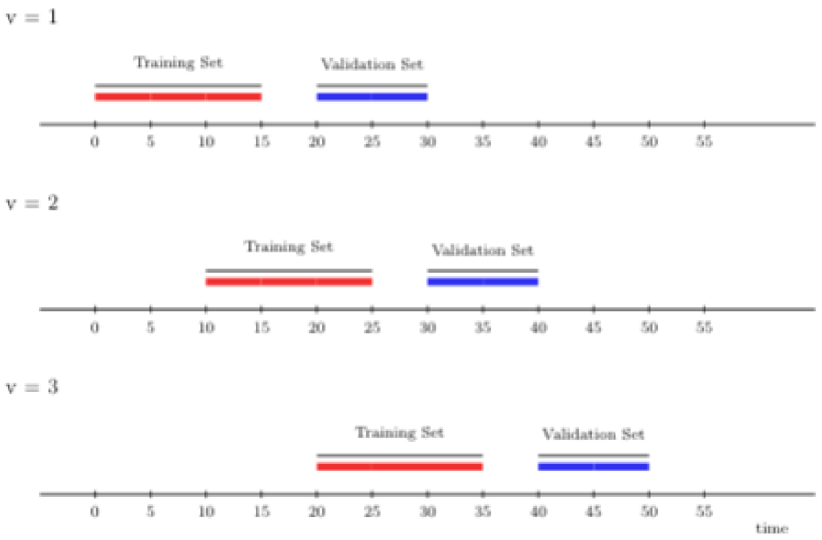


Figure 5.2: Rolling window CV

We illustrate the usage of the rolling window cross-validation with `origami` package below using the function `folds_rolling_window(n, window_size, validation_size, gap, batch)`. In order to setup `folds_rolling_window(n, window_size, validation_size, gap, batch)`, we need:

1. the total number of time points we want to cross-validate;
2. the size of the training sets;
3. the size of the validation set;
4. the gap between training and validation set;
5. the size of the update on the training set per each iteration of CV.

Setting the `window_size` to 50, `validation_size` to 10, `gap` to 5 and `batch` to 20, we also get 4 time-series folds; we show the first two below.

```
folds <- folds_rolling_window(
  t,
  window_size = 50, validation_size = 10, gap = 5, batch = 20
)
folds[[1]]
```

```
$v
[1] 1

$training_set
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50

$validation_set
 [1] 56 57 58 59 60 61 62 63 64 65

attr(,"class")
[1] "fold"
folds[[2]]
$v
[1] 2

$training_set
 [1] 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45
[26] 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70

$validation_set
 [1] 76 77 78 79 80 81 82 83 84 85

attr(,"class")
[1] "fold"
```

5.5.2.3 Rolling origin with V-fold

A variant of rolling origin scheme which accounts for sample dependence is the rolling-origin-V-fold cross-validation. In contrast to the canonical rolling origin CV, samples in the training and validation set are not the same, as the variant encompasses V-fold CV in addition to the time-series setup. The predictions are evaluated on the future times of time-series units not seen during the training step, allowing for dependence in both samples and time. One can use the rolling-origin-v-fold cross-validation with `origami` package using the function `folds_vfold_rolling_origin_pooled(n, t, id, time, V, first_window, validation_size, g`
In the figure below, we show $V = 2$ V-folds, and 2 time-series CV folds.

5.5.2.4 Rolling window with v-fold

Analogous to the previous section, we can extend rolling window CV to support multiple time-series with arbitrary sample dependence. One can use the

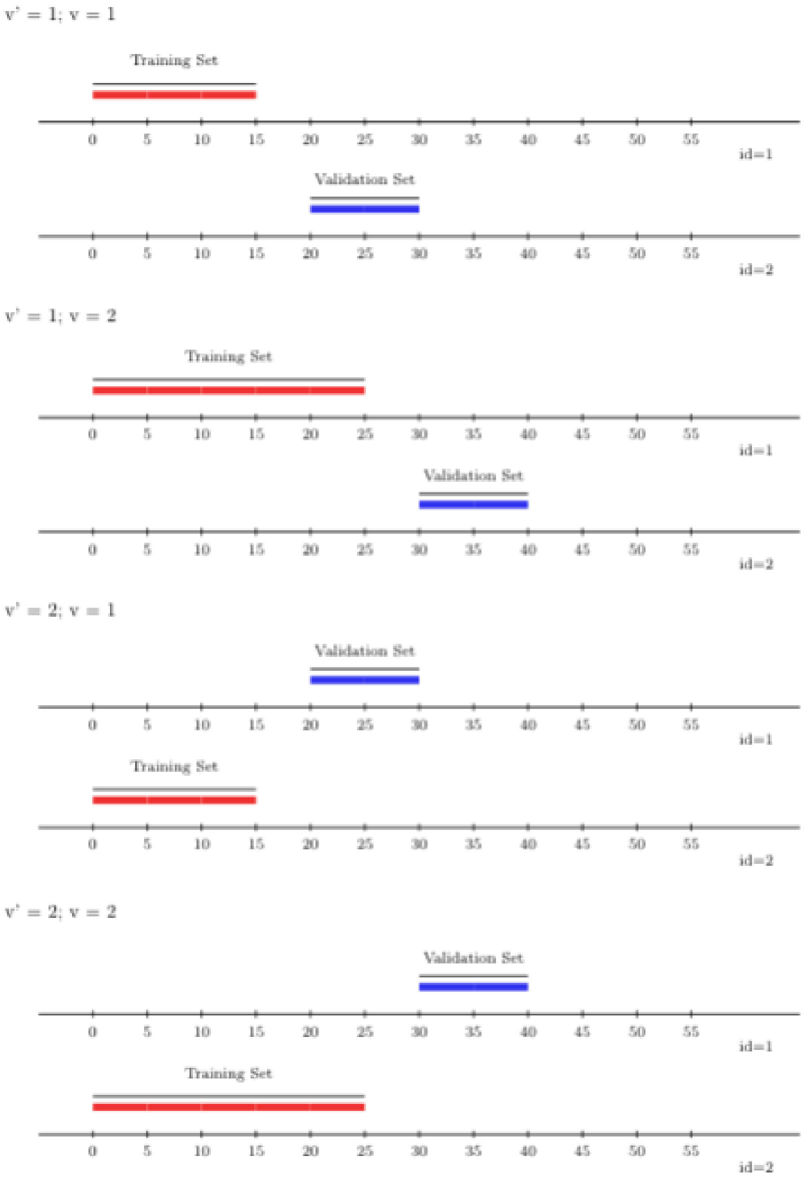


Figure 5.3: Rolling origin V-fold CV

rolling-window-V-fold cross-validation with `origami` package using the function `folds_vfold_rolling_window_pooled(n, t, id, time, V, window_size, validation_size, ga`
In the figure below, we show $V = 2$ V-folds, and 2 time-series CV folds.

5.6 General workflow of `origami`

Before we dive into more details, let's take a moment to review some of the basic functionality in `origami` R package. The main function in the `origami` is `cross_validate`. To start off, the user must define the fold structure and a function that operates on each fold. Once these are passed to `cross_validate`, `cross_validate` will apply the same specified function to each fold, and combine the fold-specific results in a meaningful way. We will see this in action in later sections; for now, we provide specific details on each each step of this process below.

5.6.1 (1) Define folds

The `folds` object passed to `cross_validate` is a list of folds; such lists can be generated using the `make_folds` function. Each fold consists of a list with a `training` index vector, a `validation` index vector, and a `fold_index` (its order in the list of folds). This function supports a variety of cross-validation schemes we describe in the following section. The `make_folds` can balance across levels of a variable (`strata_ids`), and it can also keep all observations from the same independent unit together (`cluster`).

5.6.2 (2) Define fold function

The `cv_fun` argument to `cross_validate` is a function that will perform some operation on each fold. The first argument to this function must be `fold`, which will receive an individual fold object to operate on. Additional arguments can be passed to `cv_fun` using the `...` argument to `cross_validate`. Within this function, the convenience functions `training`, `validation` and `fold_index` can return the various components of a fold object. If `training` or `validation` is passed an object, it will index it in a sensible way. For instance, if it is a vector, it will index the vector directly; if it is a `data.frame` or `matrix`, it will index rows. This allows the user to easily partition data into training and validation sets. The fold function must return a named list of results containing whatever fold-specific outputs are generated.

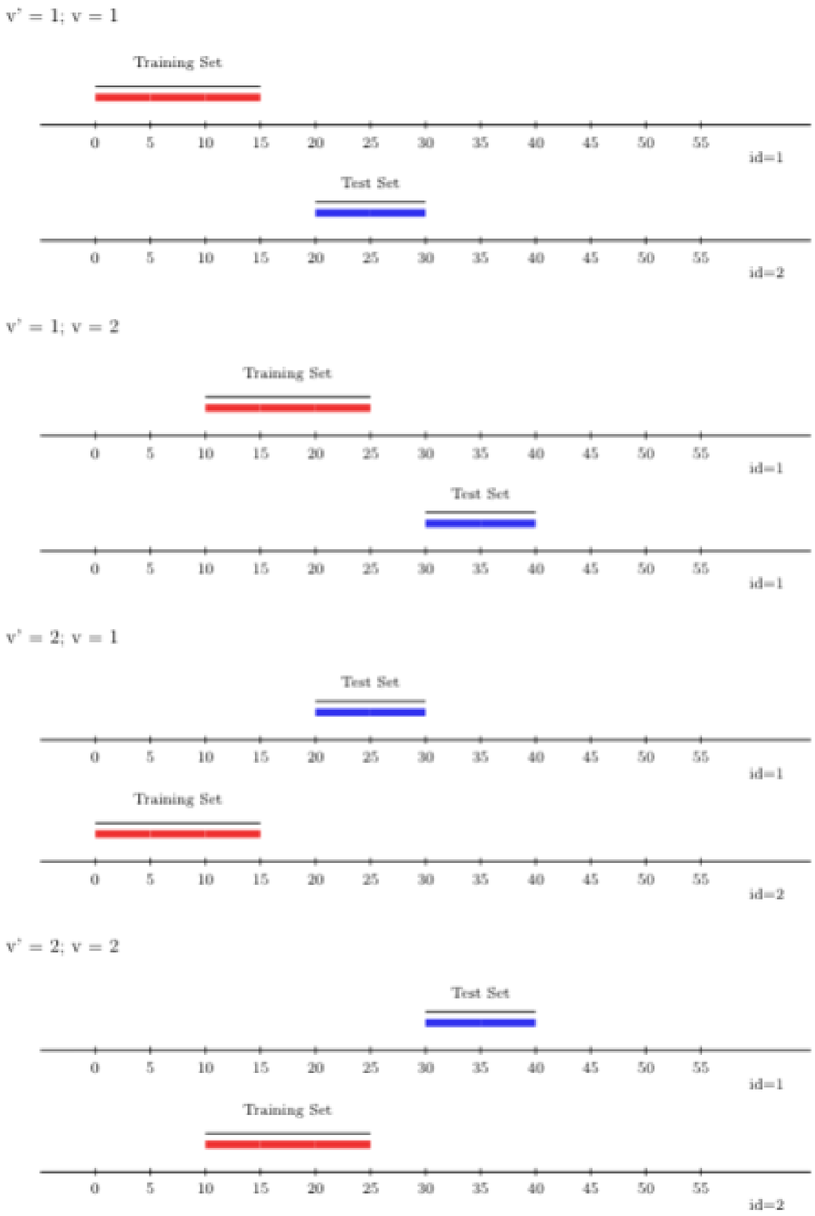


Figure 5.4: Rolling window V-fold CV

5.6.3 (3) Apply cross_validate

After defining folds, cross_validate can be used to map the cv_fun across the folds using future_lapply. This means that it can be easily parallelized by specifying a parallelization scheme (i.e., a plan from the future parallelization framework for R (Bengtsson, 2021)). The application of cross_validate generates a list of results. As described above, each call to cv_fun itself returns a list of results, with different elements for each type of result we care about. The main loop generates a list of these individual lists of results (a sort of "meta-list"). This "meta-list" is then inverted such that there is one element per result type (this too is a list of the results for each fold). By default, combine_results is used to combine these results type lists in a sensible manner. How results are combined is determined automatically by examining the data types of the results from the first fold. This can be modified by specifying a list of arguments to .combine_control.

5.7 Cross-validation in action

Let's see origami in action! In the following chapter we will learn how to use cross-validation with the Super Learner, and how we can utilize the power of cross-validation to build optimal ensembles of algorithms, not just its use on a single statistical learning method.

5.7.1 Cross-validation with linear regression

First, we will load the relevant R packages, set a seed, and load the full WASH data once again. In order to illustrate cross-validation with origami and linear regression, we will focus on predicting the weight-for-height Z-score whz using all of the available covariate data. As stated previously, we will assume the data is independent and identically distributed, ignoring the cluster structure imposed by the clinical trial design. For the sake of illustration, we will work with a subset of data, and remove all samples with missing data from the dataset; we will learn in the next chapter how to deal with missingness.

```
library(stringr)
library(dplyr)
library(tidyr)

# load data set and take a peek
```

```
washb_data <- fread(
  paste0(
    "https://raw.githubusercontent.com/tlverse/tlverse-data/master/",
    "wash-benefits/washb_data.csv"
  ),
  stringsAsFactors = TRUE
)

# Remove missing data, then pick just the first 500 rows
washb_data <- washb_data %>%
  drop_na() %>%
  slice(1:500)

outcome <- "whz"
covars <- colnames(washb_data)[-which(names(washb_data) == outcome)]
```

Here’s a look at the data:

whz	tr	fracode	month	aged	sex	momage	momedu	momheight	hfiacat
0.00	Control	N05265	9	268	male	30	Primary (1-5y)	146.40	Food Secure
-1.16	Control	N05265	9	286	male	25	Primary (1-5y)	148.75	Moderately Food I
-1.05	Control	N08002	9	264	male	25	Primary (1-5y)	152.15	Food Secure
-1.26	Control	N08002	9	252	female	28	Primary (1-5y)	140.25	Food Secure
-0.59	Control	N06531	9	336	female	19	Secondary (¿5y)	150.95	Food Secure
-0.51	Control	N06531	9	304	male	20	Secondary (¿5y)	154.20	Severely Food Ins

We can see the covariates used in the prediction:

```
outcome
[1] "whz"
covars
 [1] "tr"             "fracode"        "month"          "aged"
 [5] "sex"            "momage"         "momedu"         "momheight"
 [9] "hfiacat"        "Nlt18"          "Ncomp"          "watmin"
[13] "elec"           "floor"          "walls"          "roof"
[17] "asset_wardrobe" "asset_table"    "asset_chair"    "asset_khat"
[21] "asset_chouki"   "asset_tv"       "asset_refrig"   "asset_bike"
[25] "asset_moto"     "asset_sewmach"  "asset_mobile"
```

Next, we fit a linear model on the full data, with the goal of predicting the weight-for-height Z-score `whz` using all of the available covariate data. Let’s try it out:

```
lm_mod <- lm(whz ~ ., data = washb_data)
summary(lm_mod)

Call:
```

```
lm(formula = whz ~ ., data = washb_data)

Residuals:
    Min      1Q  Median      3Q     Max 
-2.8890 -0.6799 -0.0169  0.6595  3.1005 

Coefficients:
                                Estimate Std. Error t value Pr(>|t|)   
(Intercept)                     -1.89006    1.72022   -1.10   0.2725   
trHandwashing                   -0.25276    0.17032   -1.48   0.1385   
trNutrition                     -0.09695    0.15696   -0.62   0.5371   
trNutrition + WSH               -0.09587    0.16528   -0.58   0.5622   
trSanitation                    -0.27702    0.15846   -1.75   0.0811 . 
trWSH                           -0.02846    0.15967   -0.18   0.8586   
trWater                         -0.07148    0.15813   -0.45   0.6515   
fracodeN05160                    0.62355    0.30719    2.03   0.0430 * 
fracodeN05265                    0.38762    0.31011    1.25   0.2120   
fracodeN05359                    0.10187    0.31329    0.33   0.7452   
fracodeN06229                    0.30933    0.29766    1.04   0.2993   
fracodeN06453                    0.08066    0.30006    0.27   0.7882   
fracodeN06458                    0.43707    0.29970    1.46   0.1454   
fracodeN06473                    0.45406    0.30912    1.47   0.1426   
fracodeN06479                    0.60994    0.31463    1.94   0.0532 . 
fracodeN06489                    0.25923    0.31901    0.81   0.4169   
fracodeN06500                    0.07539    0.35794    0.21   0.8333   
fracodeN06502                    0.36748    0.30504    1.20   0.2290   
fracodeN06505                    0.20038    0.31560    0.63   0.5258   
fracodeN06516                    0.55455    0.29807    1.86   0.0635 . 
fracodeN06524                    0.49429    0.31423    1.57   0.1164   
fracodeN06528                    0.75966    0.31060    2.45   0.0148 * 
fracodeN06531                    0.36856    0.30155    1.22   0.2223   
fracodeN06862                    0.56932    0.29293    1.94   0.0526 . 
fracodeN08002                    0.36779    0.26846    1.37   0.1714   
month                            0.17161    0.10865    1.58   0.1149   
aged                            -0.00336    0.00112   -3.00   0.0029 **
sexmale                          0.12352    0.09203    1.34   0.1802   
momage                          -0.01379    0.00973   -1.42   0.1570   
momeduPrimary (1-5y)            -0.13214    0.15225   -0.87   0.3859   
momeduSecondary (>5y)            0.12632    0.16041    0.79   0.4314   
momheight                        0.00512    0.00919    0.56   0.5776   
hfiacatMildly Food Insecure      0.05804    0.19341    0.30   0.7643   
hfiacatModerately Food Insecure -0.01362    0.12887   -0.11   0.9159   
hfiacatSeverely Food Insecure   -0.13447    0.25418   -0.53   0.5970   
Nlt18                           -0.02557    0.04060   -0.63   0.5291   
Ncomp                            0.00179    0.00762    0.23   0.8145   
watmin                           0.01347    0.00861    1.57   0.1182   
```

```
elec                             0.08906    0.10700    0.83   0.4057
floor                           -0.17763    0.17734   -1.00   0.3171
walls                           -0.03001    0.21445   -0.14   0.8888
roof                            -0.03716    0.49214   -0.08   0.9399
asset_wardrobe                  -0.05754    0.13736   -0.42   0.6755
asset_table                     -0.22079    0.12276   -1.80   0.0728 .
asset_chair                      0.28012    0.13750    2.04   0.0422 *
asset_khat                       0.02306    0.11766    0.20   0.8447
asset_chouki                    -0.13943    0.14084   -0.99   0.3227
asset_tv                         0.17723    0.12972    1.37   0.1726
asset_refrig                     0.12613    0.23162    0.54   0.5863
asset_bike                      -0.02568    0.10083   -0.25   0.7990
asset_moto                      -0.32094    0.19944   -1.61   0.1083
asset_sewmach                    0.05090    0.17795    0.29   0.7750
asset_mobile                     0.01420    0.14972    0.09   0.9245
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.984 on 447 degrees of freedom
Multiple R-squared:  0.129, Adjusted R-squared:  0.0277
F-statistic: 1.27 on 52 and 447 DF,  p-value: 0.104
```

We can assess how well the model fits the data by comparing the predictions of the linear model to the true outcomes observed in the data set. This is the well known (and standard) mean squared error. We can extract that from the `lm` model object as follows:

```
(err <- mean(resid(lm_mod)^2))
[1] 0.86568
```

The mean squared error is 0.86568. There is an important problem that arises when we assess the model in this way - that is, we have trained our linear regression model on the full data set and assessed the error on the full data set, using up all of our data. We, of course, are generally not interested in how well the model explains variation in the observed data; rather, we are interested in how the explanation provided by the model generalizes to a target population from which the sample is presumably derived. Having used all of our available data, we cannot honestly evaluate how well the model fits (and thus explains) variation at the population level.

To resolve this issue, cross-validation allows for a particular procedure (e.g., linear regression) to be implemented over subsets of the data, evaluating how well the procedure fits on a testing (“validation”) set, thereby providing an honest evaluation of the error.

We can easily add cross-validation to our linear regression procedure using `origami`. First, let us define a new function to perform linear regression on a specific partition of the data (called a “fold”):

```
cv_lm <- function(fold, data, reg_form) {
  # get name and index of outcome variable from regression formula
  out_var <- as.character(unlist(str_split(reg_form, " "))[1])
  out_var_ind <- as.numeric(which(colnames(data) == out_var))

  # split up data into training and validation sets
  train_data <- training(data)
  valid_data <- validation(data)

  # fit linear model on training set and predict on validation set
  mod <- lm(as.formula(reg_form), data = train_data)
  preds <- predict(mod, newdata = valid_data)
  valid_data <- as.data.frame(valid_data)

  # capture results to be returned as output
  out <- list(
    coef = data.frame(t(coef(mod))),
    SE = (preds - valid_data[, out_var_ind])^2
  )
  return(out)
}
```

Our cv_lm function is rather simple: we merely split the available data into a training and validation sets (using the eponymous functions provided in origami) fit the linear model on the training set, and evaluate the model on the validation set. This is a simple example of what origami considers to be cv_fun — functions for using cross-validation to perform a particular routine over an input data set. Having defined such a function, we can simply generate a set of partitions using origami's make_folds function, and apply our cv_lm function over the resultant folds object. Below, we replicate the re-substitution estimate of the error – we did this "by hand" above – using the functions make_folds and cv_lm.

```
# re-substitution estimate
resub <- make_folds(washb_data, fold_fun = folds_resubstitution)[[1]]
resub_results <- cv_lm(fold = resub, data = washb_data, reg_form = "whz ~ .")
mean(resub_results$SE, na.rm = TRUE)
[1] 0.86568
```

This (nearly) matches the estimate of the error that we obtained above.

We can more honestly evaluate the error by V-fold cross-validation, which partitions the data into v subsets, fitting the model on $v-1$ of the subsets and evaluating on the subset that was held out for testing. This is repeated such that each subset is used for validation. We can easily apply our cv_lm function using origami's cross_validate (n.b., by default this performs 10-fold cross-validation):

```
# cross-validated estimate
folds <- make_folds(washb_data)
cvlm_results <- cross_validate(
  cv_fun = cv_lm, folds = folds, data = washb_data, reg_form = "whz ~ .",
  use_future = FALSE
)
mean(cvlm_results$SE, na.rm = TRUE)
[1] 1.35
```

Having performed 10-fold cross-validation, we quickly notice that our previous estimate of the model error (by resubstitution) was a bit optimistic. The honest estimate of the error is larger!

5.7.2 Cross-validation with random forests

To examine `origami` further, let us return to our example analysis using the WASH data set. Here, we will write a new `cv_fun` type object. As an example, we will use Breiman's `randomForest` (Breiman, 2001):

```
# make sure to load the package!
library(randomForest)

cv_rf <- function(fold, data, reg_form) {
  # get name and index of outcome variable from regression formula
  out_var <- as.character(unlist(str_split(reg_form, " "))[1])
  out_var_ind <- as.numeric(which(colnames(data) == out_var))

  # define training and validation sets based on input object of class "folds"
  train_data <- training(data)
  valid_data <- validation(data)

  # fit Random Forest regression on training set and predict on holdout set
  mod <- randomForest(formula = as.formula(reg_form), data = train_data)
  preds <- predict(mod, newdata = valid_data)
  valid_data <- as.data.frame(valid_data)

  # define output object to be returned as list (for flexibility)
  out <- list(
    coef = data.frame(mod$coefs),
    SE = ((preds - valid_data[, out_var_ind])^2)
  )
  return(out)
}
```

Above, in writing our cv_rf function to cross-validate randomForest, we used our previous function cv_lm as an example. For now, individual cv_fun must be written by hand; however, in future releases, a wrapper may be available to support auto-generating cv_funs to be used with origami.

Below, we use cross_validate to apply our new cv_rf function over the folds object generated by make_folds.

```
# now, let's cross-validate...
folds <- make_folds(washb_data)
cvrf_results <- cross_validate(
  cv_fun = cv_rf, folds = folds,
  data = washb_data, reg_form = "whz ~ .",
  use_future = FALSE
)
mean(cvrf_results$SE)
[1] 1.0271
```

Using 10-fold cross-validation (the default), we obtain an honest estimate of the prediction error of random forests. From this, we gather that the use of origami's cross_validate procedure can be generalized to arbitrary estimation techniques, given availability of an appropriate cv_fun function.

5.7.3 Cross-validation with arima

Cross-validation can also be used for forecast model selection in a time series setting. Here, the partitioning scheme mirrors the application of the forecasting model: we'll train the data on past observations (either all available or a recent subset), and then use the model fit to predict the next few observations. We consider the AirPassengers dataset again, a monthly time series of passenger air traffic in thousands of people.

```
data(AirPassengers)
print(AirPassengers)
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1949 112 118 132 129 121 135 148 148 136 119 104 118
1950 115 126 141 135 125 149 170 170 158 133 114 140
1951 145 150 178 163 172 178 199 199 184 162 146 166
1952 171 180 193 181 183 218 230 242 209 191 172 194
1953 196 196 236 235 229 243 264 272 237 211 180 201
1954 204 188 235 227 234 264 302 293 259 229 203 229
1955 242 233 267 269 270 315 364 347 312 274 237 278
1956 284 277 317 313 318 374 413 405 355 306 271 306
1957 315 301 356 348 355 422 465 467 404 347 305 336
1958 340 318 362 348 363 435 491 505 404 359 310 337
```

```
1959 360 342 406 396 420 472 548 559 463 407 362 405
1960 417 391 419 461 472 535 622 606 508 461 390 432
```

Suppose we want to pick between two forecasting models with different `arima` configurations. We can do that by evaluating their forecasting performance. First, we set up the appropriate cross-validation scheme for time-series.

```
folds <- make_folds(AirPassengers,
  fold_fun = folds_rolling_origin,
  first_window = 36, validation_size = 24, batch = 10
)

# How many folds where generated?
length(folds)
[1] 9

# Examine the first 2 folds.
folds[[1]]
$v
[1] 1

$training_set
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26 27 28 29 30 31 32 33 34 35 36

$validation_set
 [1] 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60

attr(,"class")
[1] "fold"
folds[[2]]
$v
[1] 2

$training_set
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46

$validation_set
 [1] 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70

attr(,"class")
[1] "fold"
```

By default, `folds_rolling_origin` will increase the size of the training set by one time point each fold. Had we followed the default option, we would have 85 folds to train! Luckily, we can pass the `batch` as option to `folds_rolling_origin` that tells it to increase the size of the training set by 10 points each iteration. Since we want to forecast the immediate next point, `gap` argument remains the default (0).

```
# make sure to load the package!
library(forecast)

# function to calculate cross-validated squared error
cv_forecasts <- function(fold, data) {
  # Get training and validation data
  train_data <- training(data)
  valid_data <- validation(data)
  valid_size <- length(valid_data)

  train_ts <- ts(log10(train_data), frequency = 12)

  # First arima model
  arima_fit <- arima(train_ts, c(0, 1, 1),
    seasonal = list(
      order = c(0, 1, 1),
      period = 12
    )
  )
  raw_arima_pred <- predict(arima_fit, n.ahead = valid_size)
  arima_pred <- 10^raw_arima_pred$pred
  arima_MSE <- mean((arima_pred - valid_data)^2)

  # Second arima model
  arima_fit2 <- arima(train_ts, c(5, 1, 1),
    seasonal = list(
      order = c(0, 1, 1),
      period = 12
    )
  )
  raw_arima_pred2 <- predict(arima_fit2, n.ahead = valid_size)
  arima_pred2 <- 10^raw_arima_pred2$pred
  arima_MSE2 <- mean((arima_pred2 - valid_data)^2)

  out <- list(mse = data.frame(
    fold = fold_index(),
    arima = arima_MSE, arima2 = arima_MSE2
  ))
  return(out)
}

mses <- cross_validate(
  cv_fun = cv_forecasts, folds = folds, data = AirPassengers,
  use_future = FALSE
)
mses$mse
  fold   arima  arima2
1    1   68.21  137.28
2    2  319.68  313.15
3    3  578.35  713.36
4    4  428.69  505.31
5    5  407.33  371.27
6    6  281.82  250.99
7    7  827.56  910.12
8    8 2099.59 2213.15
9    9  398.37  293.38
colMeans(mses$mse[, c("arima", "arima2")])
 arima arima2
601.07 634.22
```

The arima model with no AR component seems to be a better fit for this data.

5.8 Exercises

5.8.1 Review of Key Concepts

1. Compare and contrast V-fold cross-validation with resubstitution cross-validation. What are some of the differences between the two methods? How are they similar? Describe a scenario when you would use one over the other.

2. What are the advantages and disadvantages of v-fold CV relative to:

   a. holdout CV?
   b. leave-one-out CV?

3. Why can't we use V-fold cross-validation for time-series data?

4. Would you use rolling window or origin for non-stationary time-series? Why?

5.8.2 The Ideas in Action

1. Let Y be a binary variable with $P(Y = 1 \mid W) = 0.01$. What kind of cross-validation scheme should be used for a rare outcome? How can we do this with the `origami` package?

2. Consider the WASH benefits dataset presented in this chapter. How can we include cluster information into cross-validation? How can we do this with the `origami` package?

5.8.3 Advanced Topics

1. Think about a dataset with arbitrary spatial dependence, where we know the extent of dependence, and groups formed by such dependence are clear with no spillover effects. What kind of cross-validation can we use?

2. Continuing on the last problem, what kind of procedure, and cross-validation method, can we use if the spatial dependence is not clearly defined as in the previous problem?

3. Consider a classification problem with a large number of predictors. A statistician proposes the following analysis:

   a. First screen the predictors, leaving only covariates with a strong correlation with the class labels.
   b. Fit some algorithm using only the subset of highly correlated covariates.
   c. Use cross-validation to estimate the tuning parameters and the performance of the proposed algorithm.

   Is this a correct application of cross-validation? Why?

6

Super (Machine) Learning

Rachael Phillips

Based on the `sl3` `R` package by *Jeremy Coyle, Nima Hejazi, Ivana Malenica, Rachael Phillips, and Oleg Sofrygin*.

Updated: 2021-12-30

Learning Objectives

By the end of this chapter you will be able to:

1. Select an objective function that (i) aligns with the intention of the analysis and (ii) is optimized by the target parameter.

2. Assemble a diverse library of learners to be considered in the Super Learner ensemble. In particular, you should be able to:

    a. Customize a learner by modifying it's tuning parameters.
    b. Create several different versions of the same learner at once by specifying a grid of tuning parameters.
    c. Curate covariate screening pipelines in order to pass a screener's output, a subset of covariates, as input for another learner that will use the subset of covariates selected by the screener to model the data.

3. Specify the learner for ensembling (the metalearner) such that it corresponds to your objective function.

4. Fit the Super Learner ensemble with nested cross-validation to obtain an estimate of the performance of the ensemble itself on out-of-sample data.

5. Obtain `sl3` variable importance metrics.
6. Interpret the fit for discrete and continuous Super Learners' from the cross-validated risk table and the coefficients.
7. Justify the base library of machine learning algorithms and the ensembling learner in terms of the prediction problem, statistical model $\mathcal{M}$, data sparsity, and the dimensionality of the covariates.

Motivation

- A common task in data analysis is prediction – using the observed data (input variables and outcomes) to learn a function that can map new input variables into a predicted outcome.
- For some data, algorithms that learn complex relationships between variables are necessary to adequately model the data. For other data, main terms regression models might fit the data quite well.
- It is generally impossible to know a priori which algorithm will be the best for a given data set and prediction problem. It's like picking the winner of *The Great British Bake Off* at the start of the first week!
- The Super Learner solves this issue of algorithm selection by creating an ensemble of many algorithms, from the simplest (intercept-only) to most complex (neural nets, tree-based methods, support vector machines, etc.).
- Super Learner works by using cross-validation in a manner that theoretically (in large samples) guarantees the resulting fit will be as good as possible, given the algorithms provided.

6.1 Introduction

In Chapter 1, we introduced the *Roadmap for Targeted Learning* as a general template to translate real-world data applications into formal statistical estimation problems. The first steps of this roadmap define the *statistical estimation problem*, which establish

1. **The data $O$ as a random variable, or equivalently, a realization of a particular**

experiment/study, which has probability distribution P_0. This is written $O \sim P_0$, and P_0 is also commonly referred to as the data-generating process (DGP) and also the data-generating distribution (DGD). The data structure O is comprised of variables, such as a vector of covariates W, a treatment or exposure A, and an outcome Y, $O = (W, A, Y) \sim P_0$. We often observe the random variable O n times, by repeating the common experiment n times. For example, $O_1, \ldots, O_n$ random variables could be the result of a random sample of n subjects from a population, collecting baseline characteristics W, randomly assigning treatment A, and then later measuring an outcome Y.

2. **A statistical model $\mathcal{M}$ as a set of possible probability distributions that could have given rise to the data.** It's essential for $\mathcal{M}$ to only constrained by factual subject-matter knowledge in order to guarantee P_0 resides in the statistical model, written $P_0 \in \mathcal{M}$. Continuing the example from step 1, the following restrictions could be placed on the statistical model: the $O_1, \ldots, O_n$ observations in the data are independent and identically distributed (i.i.d.), the assignment of treatment A was random and not based on covariates W.
3. **A translation of the scientific question of interest into a function of P_0, the target statistical estimand $\Psi(P_0)$.** For example, we might be interested in the average difference in mean outcomes under treatment $A = 1$ versus placebo $A = 0$: $\Psi(P_0) = E_{P_0}\Big[E_{P_0}(Y|A = 1, W) - E_{P_0}(Y|A = 0, W)\Big]$. Note that, if the scientific question is causal, then it's translation will produce a target *causal* estimand; another layer of translation, identifiability, is required to express the target causal estimand as a function of the observed data distribution P_0. See causal target parameters for more information on causal quantities, causal models and identifiability.

Once the statistical estimation problem has been established, then the estimator can be constructed. Estimators (also referred to as algorithms and learners) are functions that take as input the observed data, and return as output an estimate of P_0 or some feature of P_0 (such as a component of the target estimand). We will use the Super Learner (SL) algorithm to estimate a prediction function, and then we will use the SL estimator of the prediction function to predict outcomes from new input (e.g., covariate/predictor) data. Occasionally, the prediction function itself is the target estimand; more commonly, the prediction function is a component of a target estimand.

Consider an example where we need to estimate the prediction function for $E_{P_0}(Y|A, W)$ (the conditional mean outcome, given treatment A and baseline covariates W), so we can predict what the outcomes would have been under a hypothetical scenario where all subjects received treatment $A = 1$. In order for this estimator to output predictions that correspond to outcomes in a world where all subjects received treatment $A = 1$, we would need to supply the estimator with input data that reflects it; specifically, the baseline covariate information W would remain the same treatment as it is in the observed data, but the treatment A would be set to 1 for all individuals, regardless of whether or not they actually received it. This learning paradigm corresponds to estimation of a component of the target statistical estimand mentioned in step 3 above, the $E_{P_0}(Y|A = 1, W)$ component of $\Psi(P_0)$.

There are various strategies that estimators can employ to model relationships from the observed data, and there is no "one fits all" algorithm in the realm of real-world data science. However, the statistical performance of algorithms' (e.g., mean squared error) can be used to compare them. Therefore, algorithm selection should be driven criteria that have been (i) proven to optimize relevant statistical properties (e.g., provide theoretical guarantees) and (ii) shown to be reliable in practice (e.g., with complex real-world data). The SL is an algorithm that is equipped with such a standard, the cross-validation criterion, which ensures in large samples that the SL will perform atleast as well as the unknown best-performing candidate algorithm (van der Laan and Dudoit, 2003; Van der Vaart et al., 2006; van der Laan et al., 2007). Also, as an ensemble machine learning algorithm, SL leverages information learned from a variety of candidate algorithms by creating a weighted combination of them (i.e., metalearning). In summary, SL represents a practical approach for principled machine learning. It has been shown to be adaptive and robust, even in small samples (Polley and van der Laan, 2010).

6.1.1 Candidate Learners and Ensembling

The set of algorithms considering by the SL (also referred to as "library") should consist of those that align with what's known about the DGP and what is not known about the DGP. In other words, the learners in the library should be tailored to respect the statistical model $\mathcal{M}$, both in terms of

1. the restrictions placed on $\mathcal{M}$, so the candidate algorithms represent functions that align with the knowledge about the DGP, and
2. the vastness of $\mathcal{M}$, so the library is able to adapt to a diversity of possible forms for the DGP, which can be acheived by including a variety of learning strategies (e.g., that range from parametric regression models to multi-step algorithms involving screening covariates, penalizations, optimizing tuning parameters, etc.)

6.1.1.1 Example: Respecting known bounds on the outcome

Suppose it is known that the outcome cannot take certain values, (e.g., the outcome is always a positive real number). The statistical model should be constrained to reflect these outcome bounds, and in order for the learners in the library to respect the statistical model, the learners should be constructed such that their predictions do not fall outside of the outcome bounds. For learners that allow link functions (e.g., generalized elastic-net regression models), different link functions can be chosen to match known outcome bounds. If a learner does not support link functions, or some other bounding criteria during model fitting, then there is a possibility that the learner will yield predictions that fall outside of known outcome bounds. In this scenario, the predictions that are not within known bounds could be truncated, which might be fine for learners that seldomly produce predictions that violate known outcome bounds.

In general, the use of link function(s) is more sensible, since it formulates a model for the function that respects the statistical model, and then optimizes the fit in that model. The truncation option optimizes in a model that's too big, losing information, and then corrects ad hoc. However, it might be limiting to only include learners that support some desired link function (e.g., a library of several logistic regression models), since a diversity of possible DGPs might not be captured by this library.

Recall that SL is a weighted combination of this library of candidate algorithms, and not all weighted combinations are created equally. In order to ensure that the SL has the same bounds on the predictions as the candidates in the library, SL's weighted combination should be a convex combination (i.e., weights are non-negative and sum to one). The most simple example of a convex combination would be the so-called "discrete SL" or "cross-validated selector", which uses a metalearner that assigns a weight of one to the *best* candidate algorithm in the library, and a weight of zero to all others (where *best* is described in step 4(a) in the step-by-step overview below). More flexible metalearners, like the default metalearner in `sl3`, are those that allow multiple algorithms to have nonzero weights and still enforce a convex combination.

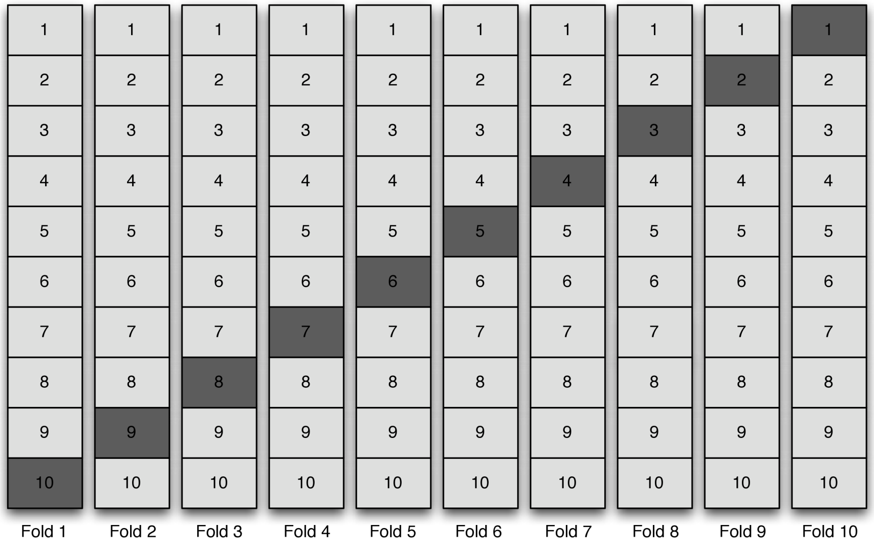
Keep in mind

The learners in the SL library and should be justified in terms of the prediction problem, statistical model $\mathcal{M}$, data sparsity (e.g., number of independent samples, number of events for rare binary outcomes), and the number of covariates. The metalearning strategy should be similarly justified.

6.1.2 Fitting the Super Learner

6.1.2.1 Cross-validation

- There are many different cross-validation schemes, which are designed to accommodate different study designs, data structures, and prediction problems. See cross-validation for more detail.

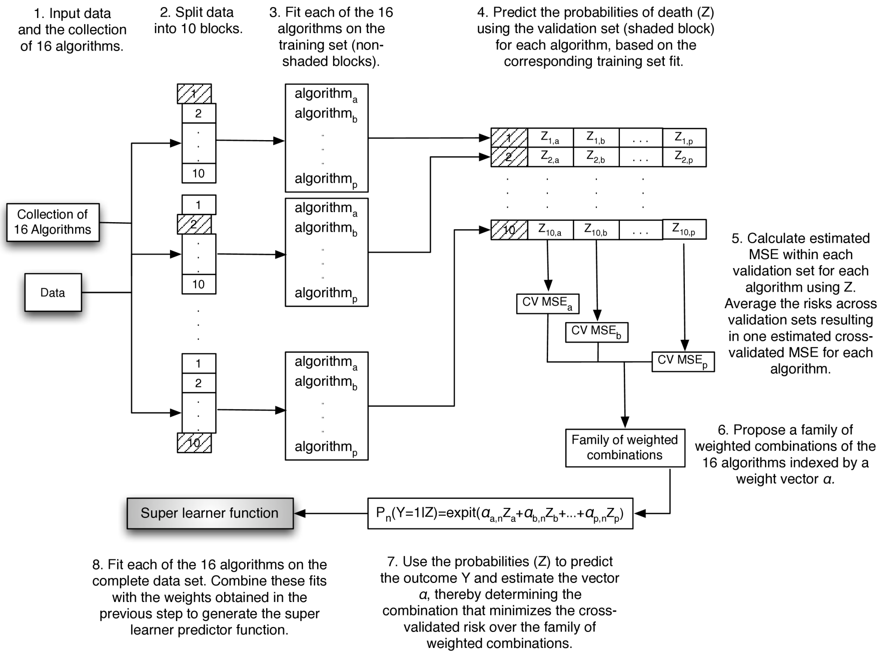


The figure above shows an example of V-fold cross-validation with $V = 10$ folds, and this is the default cross-validation structure in the `sl3` `R` package. The darker boxes represent the so-called "validation data" and the lighter boxes represent the so-called "training data". The following details are important to notice:

- Across all folds, there are V (10) copies of the dataset. The only difference between each copy is the coloring, which distinguishes the subset of the data that's considered as the training data from the subset that's considered as the validation data.
- Within each fold 1/V (1/10) of the data is the validation data.
- Across all folds, all of the data will be considered as validation data and no observation will be included twice as validation data. Therefore, the total number of validation data observations across all of the folds is equal to the total number of observations in the data.

6.1.2.2 Step-by-step procedure with V-fold Cross-validation

1. Fit each learner (say there are K learners) on the whole dataset. We refer to these learners that are trained on the whole dataset as "full-fit" learners.
2. Break up the data evenly into V disjoint subsets. Separately, create V copies of the data. For each copy v, where $v = 1, \ldots, V$, create the V folds by labelling the portion of the data that was included in subset v as the validation sample, and the labelling what's remaining of the data as the training sample.
3. For each fold v, $v = 1, \ldots, V$, fit each learner (say there are K learners) on the training sample and predict the validation sample outcomes by providing each fitted learner with the validation sample covariates as input. Notice that each learner will be fit V times. We refer to these learners that are trained across the V cross-validation folds as "cross-validated fit" learners.
4. Combine the validation sample predictions from all folds and all learners to create the so-called K column matrix of "cross-validated predictions". This matrix is also commonly referred to as the Z matrix. Notice that it contains, for each learner, out-of-sample predictions for all of the observations in the data.
5. Train the metalearner (e.g., a non-negative least squares regression) on data with predictors and outcomes being the Z matrix and the observed data outcomes, respectively. The metalearner — just like any ordinary ML algorithm — estimates the parameters of it's model using the training data and afterwards, the fitted model can be used to obtain predicted outcomes from new input data. What's special about the metalearner is that it's estimated model parameters (e.g., regression coefficients) correspond to it's predictors, which are the variables in the Z matrix, the K learners' predictions. Once the metalearner is fit, it can be used to obtain predicted outcomes from new input data; that is, new K learners predictions' can be supplied to the fitted metalearner in order to obtain predicted outcomes.
6. The fitted metalearner and the full-fit learners define the weighted combination of the K learners, finalizing the Super Learner (SL) fit. To obtain SL predictions the full-fit learners' predictions are first obtained and then fed as input to the fitted metalearner; the metalearner's output is the SL predictions.


1. Input data and the collection of 16 algorithms.
2. Split data into 10 blocks.
3. Fit each of the 16 algorithms on the training set (non-shaded blocks).
4. Predict the probabilities of death (Z) using the validation set (shaded block) for each algorithm, based on the corresponding training set fit.
Collection of 16 Algorithms
Data
1
2
10
algorithm$_a$
algorithm$_b$
algorithm$_p$
$Z_{1,a}$ $Z_{1,b}$ . . . $Z_{1,p}$
$Z_{2,a}$ $Z_{2,b}$ . . . $Z_{2,p}$
$Z_{10,a}$ $Z_{10,b}$ . . . $Z_{10,p}$
CV MSE$_a$
CV MSE$_b$
CV MSE$_p$
5. Calculate estimated MSE within each validation set for each algorithm using Z. Average the risks across validation sets resulting in one estimated cross-validated MSE for each algorithm.
Family of weighted combinations
6. Propose a family of weighted combinations of the 16 algorithms indexed by a weight vector α.
Super learner function
$P_n(Y=1|Z)=\text{expit}(\alpha_{a,n}Z_a+\alpha_{b,n}Z_b+...+\alpha_{p,n}Z_p)$
8. Fit each of the 16 algorithms on the complete data set. Combine these fits with the weights obtained in the previous step to generate the super learner predictor function.
7. Use the probabilities (Z) to predict the outcome Y and estimate the vector α, thereby determining the combination that minimizes the cross-validated risk over the family of weighted combinations.

6.1.3 Theoretical foundations

This section is under construction.

For more detail on Super Learner algorithm we refer the reader to Polley and van der Laan (2010) and van der Laan et al. (2007). The optimality results for the cross-validation selector among a family of algorithms were established in van der Laan and Dudoit (2003) and extended in Van der Vaart et al. (2006).

`sl3` "Microwave Dinner" Implementation

We begin by illustrating the core functionality of the SL algorithm as implemented in `sl3`.

The `sl3` implementation consists of the following steps:

0. Load the necessary libraries and data
1. Define the machine learning task

2. Make an SL by creating library of base learners and a metalearner
3. Train the SL on the machine learning task
4. Obtain predicted values

WASH Benefits Study Example

Using the WASH Benefits Bangladesh data, we are interested in predicting weight-for-height z-score `whz` using the available covariate data. More information on this dataset, and all other data that we will work with, are described in this chapter of the `tlverse` handbook. Let's begin!

0. Load the necessary libraries and data

First, we will load the relevant `R` packages, set a seed, and load the data.

```
library(data.table)
library(dplyr)
library(readr)
library(ggplot2)
library(SuperLearner)
library(origami)
library(sl3)
library(knitr)
library(kableExtra)

# load data set and take a peek
washb_data <- fread(
  paste0(
    "https://raw.githubusercontent.com/tlverse/tlverse-data/master/",
    "wash-benefits/washb_data.csv"
  ),
  stringsAsFactors = TRUE
)
head(washb_data) %>%
  kable() %>%
  kableExtra::kable_styling(fixed_thead = T) %>%
  scroll_box(width = "100%", height = "300px")
```

1. Define the machine learning task

To define the machine learning `task` (predict weight-for-height Z-score `whz` using the available covariate data), we need to create an `sl3_Task` object.

whz	tr	fracode	month	aged	sex	momage	momedu	momheight	hfiacat
0.00	Control	N05265	9	268	male	30	Primary (1-5y)	146.40	Food Secu
-1.16	Control	N05265	9	286	male	25	Primary (1-5y)	148.75	Moderatel
-1.05	Control	N08002	9	264	male	25	Primary (1-5y)	152.15	Food Secu
-1.26	Control	N08002	9	252	female	28	Primary (1-5y)	140.25	Food Secu
-0.59	Control	N06531	9	336	female	19	Secondary (¿5y)	150.95	Food Secu
-0.51	Control	N06531	9	304	male	20	Secondary (¿5y)	154.20	Severely F

The `sl3_Task` keeps track of the roles the variables play in the machine learning problem, the data, and any metadata (e.g., observational-level weights, IDs, offset).

Also, if we had missing outcomes, we would need to set `drop_missing_outcome = TRUE` when we create the task. In the next analysis, with the IST stroke trial data, we do have a missing outcome. In the following chapter, we need to estimate this missingness mechanism; which is the conditional probably that the outcome is observed, given the history (i.e., variables that were measured before the missingness). Estimating the missingness mechanism requires learning a prediction function that outputs the predicted probability that a unit is missing, given their history.

```
# specify the outcome and covariates
outcome <- "whz"
covars <- colnames(washb_data)[-which(names(washb_data) == outcome)]

# create the sl3 task
washb_task <- make_sl3_Task(
  data = washb_data,
  covariates = covars,
  outcome = outcome
)
Warning in process_data(data, nodes, column_names = column_names, flag = flag, :
Missing covariate data detected: imputing covariates.
```

This warning is important. The task just imputed missing covariates for us. Specifically, for each covariate column with missing values, `sl3` uses the median to impute missing continuous covariates, and the mode to impute binary and categorical covariates.

Also, for each covariate column with missing values, `sl3` adds an additional column indicating whether or not the value was imputed, which is particularly handy when the missingness in the data might be informative.

Also, notice that we did not specify the number of folds, or the loss function in the task. The default cross-validation scheme is V-fold, with $V = 10$ number of folds.

Let's visualize our `washb_task`:

```
washb_task
A sl3 Task with 4695 obs and these nodes:
$covariates
 [1] "tr"              "fracode"         "month"           "aged"
 [5] "sex"             "momage"          "momedu"          "momheight"
 [9] "hfiacat"         "Nlt18"           "Ncomp"           "watmin"
[13] "elec"            "floor"           "walls"           "roof"
[17] "asset_wardrobe"  "asset_table"     "asset_chair"     "asset_khat"
[21] "asset_chouki"    "asset_tv"        "asset_refrig"    "asset_bike"
[25] "asset_moto"      "asset_sewmach"   "asset_mobile"    "delta_momage"
[29] "delta_momheight"

$outcome
[1] "whz"

$id
NULL

$weights
NULL

$offset
NULL

$time
NULL
```

We can't see when we print the task, but the default cross-validation fold structure (V-fold cross-validation with V=10 folds) was created when we defined the task.

```
length(washb_task$folds) # how many folds?
[1] 10

head(washb_task$folds[[1]]$training_set) # row indexes for fold 1 training
[1] 1 2 3 4 5 6
head(washb_task$folds[[1]]$validation_set) # row indexes for fold 1 validation
[1] 12 21 29 41 43 53

any(
  washb_task$folds[[1]]$training_set %in% washb_task$folds[[1]]$validation_set
)
[1] FALSE
```

Tip: If you type `washb_task$` and then press the tab button (you will need to press tab twice if you're not in RStudio), you can view all of the active and public fields and methods that can be accessed from the `washb_task` object.

2. Make a Super Learner

Now that we have defined our machine learning problem with the `sl3_Task`, we are ready to make the Super Learner (SL). This requires specification of

- A set of candidate machine learning algorithms, also commonly referred to as a library of learners. The set should include a diversity of algorithms that are believed to be consistent with the true data-generating distribution.
- A metalearner, to ensemble the base learners.

We might also incorporate

- Feature selection, to pass only a subset of the predictors to the algorithm.
- Hyperparameter specification, to tune base learners.

Learners have properties that indicate what features they support. We may use `sl3_list_properties()` to get a list of all properties supported by at least one learner.

```
sl3_list_properties()
 [1] "binomial"      "categorical"   "continuous"    "cv"
 [5] "density"       "h2o"           "ids"           "importance"
 [9] "offset"        "preprocessing" "sampling"      "screener"
[13] "timeseries"    "weights"       "wrapper"
```

Since we have a continuous outcome, we may identify the learners that support this outcome type with `sl3_list_learners()`.

```
sl3_list_learners("continuous")
 [1] "Lrnr_arima"                 "Lrnr_bartMachine"
 [3] "Lrnr_bayesglm"              "Lrnr_bilstm"
 [5] "Lrnr_bound"                 "Lrnr_caret"
 [7] "Lrnr_cv_selector"           "Lrnr_dbarts"
 [9] "Lrnr_earth"                 "Lrnr_expSmooth"
[11] "Lrnr_gam"                   "Lrnr_gbm"
[13] "Lrnr_glm"                   "Lrnr_glm_fast"
[15] "Lrnr_glmnet"                "Lrnr_grf"
[17] "Lrnr_gru_keras"             "Lrnr_gts"
[19] "Lrnr_h2o_glm"               "Lrnr_h2o_grid"
```

```
[21] "Lrnr_hal9001"                   "Lrnr_HarmonicReg"
[23] "Lrnr_hts"                       "Lrnr_lightgbm"
[25] "Lrnr_lstm_keras"                "Lrnr_mean"
[27] "Lrnr_multiple_ts"               "Lrnr_nnet"
[29] "Lrnr_nnls"                      "Lrnr_optim"
[31] "Lrnr_pkg_SuperLearner"          "Lrnr_pkg_SuperLearner_method"
[33] "Lrnr_pkg_SuperLearner_screener" "Lrnr_polspline"
[35] "Lrnr_randomForest"              "Lrnr_ranger"
[37] "Lrnr_rpart"                     "Lrnr_rugarch"
[39] "Lrnr_screener_correlation"      "Lrnr_solnp"
[41] "Lrnr_stratified"                "Lrnr_svm"
[43] "Lrnr_tsDyn"                     "Lrnr_xgboost"
```

Now that we have an idea of some learners, we can construct them using the `make_learner` function or the `new` method.

```
# choose base learners
lrn_glm <- make_learner(Lrnr_glm)
lrn_mean <- Lrnr_mean$new()
```

We can customize learner hyperparameters to incorporate a diversity of different settings. Documentation for the learners and their hyperparameters can be found in the `sl3` Learners Reference.

```
lrn_lasso <- make_learner(Lrnr_glmnet) # alpha default is 1
lrn_ridge <- Lrnr_glmnet$new(alpha = 0)
lrn_enet.5 <- make_learner(Lrnr_glmnet, alpha = 0.5)

lrn_polspline <- Lrnr_polspline$new()

lrn_ranger100 <- make_learner(Lrnr_ranger, num.trees = 100)

lrn_hal_faster <- Lrnr_hal9001$new(max_degree = 2, reduce_basis = 0.05)

xgb_fast <- Lrnr_xgboost$new() # default with nrounds = 20 is pretty fast
xgb_50 <- Lrnr_xgboost$new(nrounds = 50)
```

We can use `Lrnr_define_interactions` to define interaction terms among covariates. The interactions should be supplied as list of character vectors, where each vector specifies an interaction. For example, we specify interactions below between (1) `tr` (whether or not the subject received the WASH intervention) and `elec` (whether or not the subject had electricity); and between (2) `tr` and `hfiacat` (the subject's level of food security).

```
interactions <- list(c("elec", "tr"), c("tr", "hfiacat"))
# main terms as well as the interactions above will be included
lrn_interaction <- make_learner(Lrnr_define_interactions, interactions)
```

What we just defined above is incomplete. In order to fit learners with these interactions, we need to create a `Pipeline`. A `Pipeline` is a set of learners to be fit sequentially, where the fit from one learner is used to define the task for the next learner. We need to create a `Pipeline` with the interaction defining learner and another learner that incorporate these terms when fitting a model. Let's create a learner pipeline that will fit a linear model with the combination of main terms and interactions terms, as specified in `lrn_interaction`.

```
# we already instantiated a linear model learner, no need to do that again
lrn_glm_interaction <- make_learner(Pipeline, lrn_interaction, lrn_glm)
lrn_glm_interaction
[1] "Lrnr_define_interactions_TRUE"
[1] "Lrnr_glm_TRUE"
```

We can also include learners from the `SuperLearner R` package.

```
lrn_bayesglm <- Lrnr_pkg_SuperLearner$new("SL.bayesglm")
```

Here is a fun trick to create customized learners over a grid of parameters.

```
# I like to crock pot my SLs
grid_params <- list(
  cost = c(0.01, 0.1, 1, 10, 100, 1000),
  gamma = c(0.001, 0.01, 0.1, 1),
  kernel = c("polynomial", "radial", "sigmoid"),
  degree = c(1, 2, 3)
)
grid <- expand.grid(grid_params, KEEP.OUT.ATTRS = FALSE)
svm_learners <- apply(grid, MARGIN = 1, function(tuning_params) {
  do.call(Lrnr_svm$new, as.list(tuning_params))
})

grid_params <- list(
  max_depth = c(2, 4, 6),
  eta = c(0.001, 0.1, 0.3),
  nrounds = 100
)
grid <- expand.grid(grid_params, KEEP.OUT.ATTRS = FALSE)
grid
  max_depth   eta nrounds
1         2 0.001     100
2         4 0.001     100
3         6 0.001     100
4         2 0.100     100
5         4 0.100     100
6         6 0.100     100
7         2 0.300     100
```

```
8          4 0.300     100
9          6 0.300     100

xgb_learners <- apply(grid, MARGIN = 1, function(tuning_params) {
  do.call(Lrnr_xgboost$new, as.list(tuning_params))
})
xgb_learners
[[1]]
[1] "Lrnr_xgboost_100_1_2_0.001"

[[2]]
[1] "Lrnr_xgboost_100_1_4_0.001"

[[3]]
[1] "Lrnr_xgboost_100_1_6_0.001"

[[4]]
[1] "Lrnr_xgboost_100_1_2_0.1"

[[5]]
[1] "Lrnr_xgboost_100_1_4_0.1"

[[6]]
[1] "Lrnr_xgboost_100_1_6_0.1"

[[7]]
[1] "Lrnr_xgboost_100_1_2_0.3"

[[8]]
[1] "Lrnr_xgboost_100_1_4_0.3"

[[9]]
[1] "Lrnr_xgboost_100_1_6_0.3"
```

Did you see `Lrnr_caret` when we called `sl3_list_learners(c("binomial"))`? All we need to specify to use this popular algorithm as a candidate in our SL is the `algorithm` we want to tune, which is passed as `method` to `caret::train()`.

```
# Unlike xgboost, I have no idea how to tune a neural net or BART machine, so
# I let caret take the reins
lrnr_caret_nnet <- make_learner(Lrnr_caret, algorithm = "nnet")
lrnr_caret_bartMachine <- make_learner(Lrnr_caret,
  algorithm = "bartMachine",
  method = "boot", metric = "Accuracy",
  tuneLength = 10
)
```

In order to assemble the library of learners, we need to `Stack` them together.

A `Stack` is a special learner and it has the same interface as all other learners. What makes a stack special is that it combines multiple learners by training them simultaneously, so that their predictions can be either combined or compared.

```
stack <- make_learner(
  Stack, lrn_glm, lrn_polspline, lrn_enet.5, lrn_ridge, lrn_lasso, xgb_50
)
stack
[1] "Lrnr_glm_TRUE"
[2] "Lrnr_polspline_5"
[3] "Lrnr_glmnet_NULL_deviance_10_0.5_100_TRUE_FALSE"
[4] "Lrnr_glmnet_NULL_deviance_10_0_100_TRUE_FALSE"
[5] "Lrnr_glmnet_NULL_deviance_10_1_100_TRUE_FALSE"
[6] "Lrnr_xgboost_50_1"
```

We can also stack the learners by first creating a vector, and then instantiating the stack. I prefer this method, since it easily allows us to modify the names of the learners.

```
# named vector of learners first
learners <- c(
  lrn_glm, lrn_polspline, lrn_enet.5, lrn_ridge, lrn_lasso, xgb_50
)
names(learners) <- c(
  "glm", "polspline", "enet.5", "ridge", "lasso", "xgboost50"
)
# next make the stack
stack <- make_learner(Stack, learners)
# now the names are pretty
stack
[1] "glm"       "polspline" "enet.5"    "ridge"     "lasso"     "xgboost50"
```

We're jumping ahead a bit, but let's check something out quickly. It's straightforward, and just one more step, to set up this stack such that all of the learners will train in a cross-validated manner.

```
cv_stack <- Lrnr_cv$new(stack)
cv_stack
[1] "Lrnr_cv"
[1] "glm"       "polspline" "enet.5"    "ridge"     "lasso"     "xgboost50"
```

Screening Algorithms for Feature Selection

We can optionally select a subset of available covariates and pass only those variables to the modeling algorithm. The current set of learners that can be used for prescreening covariates is included below.

- Lrnr_screener_importance selects num_screen (default = 5) covariates based on the variable importance ranking provided by the learner. Any learner with an importance method can be used in Lrnr_screener_importance; and this currently includes Lrnr_ranger, Lrnr_randomForest, and Lrnr_xgboost.
- Lrnr_screener_coefs, which provides screening of covariates based on the magnitude of their estimated coefficients in a (possibly regularized) GLM. The threshold (default = 1e-3) defines the minimum absolute size of the coefficients, and thus covariates, to be kept. Also, a max_retain argument can be optionally provided to restrict the number of selected covariates to be no more than max_retain.
- Lrnr_screener_correlation provides covariate screening procedures by running a test of correlation (Pearson default), and then selecting the (1) top ranked variables (default), or (2) the variables with a pvalue lower than some pre-specified threshold.
- Lrnr_screener_augment augments a set of screened covariates with additional covariates that should be included by default, even if the screener did not select them. An example of how to use this screener is included below.

Let's consider screening covariates based on their randomForest variable importance ranking (ordered by mean decrease in accuracy). To select the top 5 most important covariates according to this ranking, we can combine Lrnr_screener_importance with Lrnr_ranger (limiting the number of trees by setting ntree = 20).

Hang on! Before you think it – we will confess: Bob Ross and us both know that 20 trees makes for a lonely forest, and we shouldn't consider it, but these are the sacrifices we make for this chapter to be built in time!

```
miniforest <- Lrnr_ranger$new(
  num.trees = 20, write.forest = FALSE,
  importance = "impurity_corrected"
)

# learner must already be instantiated, we did this when we created miniforest
screen_rf <- Lrnr_screener_importance$new(learner = miniforest, num_screen = 5)
screen_rf
[1] "Lrnr_screener_importance_5"
```

```
# which covariates are selected on the full data?
screen_rf$train(washb_task)
[1] "Lrnr_screener_importance_5"
$selected
[1] "aged"      "month"     "tr"        "momheight" "momedu"
```

An example of how to format `Lrnr_screener_augment` is included below for clarity.

```
keepme <- c("aged", "momage")
# screener must already be instantiated, we did this when we created screen_rf
screen_augment_rf <- Lrnr_screener_augment$new(
  screener = screen_rf, default_covariates = keepme
)
screen_augment_rf
[1] "Lrnr_screener_augment_c(\"aged\", \"momage\")"
```

Selecting covariates with non-zero lasso coefficients is quite common. Let's construct `Lrnr_screener_coefs` screener that does just that, and test it out.

```
# we already instantiated a lasso learner above, no need to do it again
screen_lasso <- Lrnr_screener_coefs$new(learner = lrn_lasso, threshold = 0)
screen_lasso
[1] "Lrnr_screener_coefs_0_NULL_2"
```

To pipe only the selected covariates to the modeling algorithm, we need to make a `Pipeline`, similar to the one we built for the regression model with interaction terms.

```
screen_rf_pipe <- make_learner(Pipeline, screen_rf, stack)
screen_lasso_pipe <- make_learner(Pipeline, screen_lasso, stack)
```

Now, these learners will be preceded by a screening step.

We also consider the original `stack`, to compare how the feature selection methods perform in comparison to the methods without feature selection.

Analogous to what we have seen before, we have to stack the pipeline and original `stack` together, so we may use them as base learners in our super learner.

```
# pretty names again
learners2 <- c(learners, screen_rf_pipe, screen_lasso_pipe)
names(learners2) <- c(names(learners), "randomforest_screen", "lasso_screen")

fancy_stack <- make_learner(Stack, learners2)
fancy_stack
[1] "glm"                 "polspline"           "enet.5"
[4] "ridge"               "lasso"               "xgboost50"
[7] "randomforest_screen" "lasso_screen"
```

We will use the default metalearner, which uses `Lrnr_solnp` to provide fitting procedures for a pairing of loss function and metalearner function. This default metalearner selects a loss and metalearner pairing based on the outcome type. Note that any learner can be used as a metalearner.

Now that we have made a diverse stack of base learners, we are ready to make the SL. The SL algorithm fits a metalearner on the validation set predictions/losses across all folds.

```
sl <- make_learner(Lrnr_sl, learners = fancy_stack)
```

We can also use `Lrnr_cv` to build a SL, cross-validate a stack of learners to compare performance of the learners in the stack, or cross-validate any single learner (see "Cross-validation" section of this `sl3` introductory tutorial).

Furthermore, we can Define New `sl3` Learners which can be used in all the places you could otherwise use any other `sl3` learners, including `Pipelines`, `Stacks`, and the SL.

Recall that the discrete SL, or cross-validated selector, is a metalearner that assigns a weight of 1 to the learner with the lowest cross-validated empirical risk, and weight of 0 to all other learners. This metalearner specification can be invoked with `Lrnr_cv_selector`.

```
discrete_sl_metalrn <- Lrnr_cv_selector$new()
discrete_sl <- Lrnr_sl$new(
  learners = fancy_stack,
  metalearner = discrete_sl_metalrn
)
```

3. Train the Super Learner on the machine learning task

The SL algorithm fits a metalearner on the validation-set predictions in a cross-validated manner, thereby avoiding overfitting.

Now we are ready to `train` our SL on our `sl3_task` object, `washb_task`.

```
set.seed(4197)
sl_fit <- sl$train(washb_task)
```

4. Obtain predicted values

Now that we have fit the SL, we are ready to calculate the predicted outcome for each subject.

```
# we did it! now we have SL predictions
sl_preds <- sl_fit$predict()
head(sl_preds)
[1] -0.65442 -0.77055 -0.67359 -0.65109 -0.65577 -0.65673
```

We can also obtain a summary of the results.

```
sl_fit$cv_risk(loss_fun = loss_squared_error)
                          learner coefficients   risk        se   fold_sd
 1:                           glm     0.055571 1.0202 0.023955 0.067500
 2:                     polspline     0.055556 1.0208 0.023577 0.067921
 3:                        enet.5     0.055564 1.0131 0.023598 0.065732
 4:                         ridge     0.055570 1.0153 0.023739 0.065299
 5:                         lasso     0.055564 1.0130 0.023592 0.065840
 6:                     xgboost50     0.055591 1.1136 0.025262 0.077580
 7:       randomforest_screen_glm     0.055546 1.0271 0.024119 0.069913
 8: randomforest_screen_polspline     0.055561 1.0236 0.024174 0.068710
 9:    randomforest_screen_enet.5     0.055546 1.0266 0.024101 0.070117
10:     randomforest_screen_ridge     0.055546 1.0268 0.024120 0.069784
11:     randomforest_screen_lasso     0.055546 1.0266 0.024101 0.070135
12: randomforest_screen_xgboost50     0.055523 1.1399 0.026341 0.100112
13:              lasso_screen_glm     0.055559 1.0164 0.023542 0.065018
14:        lasso_screen_polspline     0.055559 1.0177 0.023520 0.065566
15:           lasso_screen_enet.5     0.055559 1.0163 0.023544 0.065017
16:            lasso_screen_ridge     0.055559 1.0166 0.023553 0.064869
17:            lasso_screen_lasso     0.055559 1.0163 0.023544 0.065020
18:        lasso_screen_xgboost50     0.055521 1.1256 0.025939 0.084270
19:                  SuperLearner           NA 1.0135 0.023615 0.067434
    fold_min_risk fold_max_risk
 1:       0.89442        1.1200
 2:       0.89892        1.1255
 3:       0.88839        1.1058
 4:       0.88559        1.1063
 5:       0.88842        1.1060
 6:       0.96019        1.2337
 7:       0.90251        1.1326
 8:       0.90167        1.1412
 9:       0.90030        1.1319
10:       0.90068        1.1311
11:       0.90043        1.1321
12:       0.92377        1.2549
13:       0.90204        1.1156
14:       0.89742        1.1162
15:       0.90184        1.1154
16:       0.90120        1.1146
17:       0.90183        1.1154
18:       0.96251        1.2327
```

```
19:        0.88685        1.1102
```

Cross-validated Super Learner

We can cross-validate the SL to see how well the SL performs on unseen data, and obtain an estimate of the cross-validated risk of the SL.

This estimation procedure requires an outer/external layer of cross-validation, also called nested cross-validation, which involves setting aside a separate holdout sample that we don't use to fit the SL. This external cross-validation procedure may also incorporate 10 folds, which is the default in `sl3`. However, we will incorporate 2 outer/external folds of cross-validation for computational efficiency.

We also need to specify a loss function to evaluate SL. Documentation for the available loss functions can be found in the `sl3` Loss Function Reference.

```
washb_task_new <- make_sl3_Task(
  data = washb_data,
  covariates = covars,
  outcome = outcome,
  folds = origami::make_folds(washb_data, fold_fun = folds_vfold, V = 2)
)
CVsl <- CV_lrnr_sl(
  lrnr_sl = sl_fit, task = washb_task_new, loss_fun = loss_squared_error
)
CVsl %>%
  kable(digits = 4) %>%
  kableExtra::kable_styling(fixed_thead = T) %>%
  scroll_box(width = "100%", height = "300px")
```

Variable Importance Measures with `sl3`

Variable importance can be interesting and informative. It can also be contradictory and confusing. Nevertheless, we like it, and so do our collaborators, so we created a variable importance function in `sl3`! The `sl3 importance` function returns a table with variables listed in decreasing order of importance (i.e., most important on the first row).

learner	coefficients	risk	se	fold_sd	fold_min_risk	fold_max_risk
glm	0.0556	1.0340	0.0255	0.0195	1.0203	1.0478
polspline	0.0556	1.0231	0.0236	0.0005	1.0227	1.0235
enet.5	0.0556	1.0140	0.0236	0.0081	1.0083	1.0197
ridge	0.0556	1.0192	0.0239	0.0118	1.0108	1.0275
lasso	0.0556	1.0141	0.0236	0.0081	1.0084	1.0198
xgboost50	0.0556	1.1647	0.0259	0.0025	1.1629	1.1665
randomforest_screen_glm	0.0556	1.0245	0.0239	0.0222	1.0088	1.0403
randomforest_screen_polspline	0.0555	1.0315	0.0239	0.0231	1.0151	1.0478
randomforest_screen_enet.5	0.0556	1.0240	0.0238	0.0214	1.0089	1.0392
randomforest_screen_ridge	0.0556	1.0243	0.0239	0.0218	1.0089	1.0397
randomforest_screen_lasso	0.0556	1.0240	0.0238	0.0214	1.0089	1.0392
randomforest_screen_xgboost50	0.0555	1.1798	0.0270	0.0522	1.1430	1.2167
lasso_screen_glm	0.0556	1.0233	0.0238	0.0001	1.0232	1.0234
lasso_screen_polspline	0.0556	1.0247	0.0238	0.0018	1.0235	1.0260
lasso_screen_enet.5	0.0556	1.0233	0.0238	0.0001	1.0232	1.0233
lasso_screen_ridge	0.0556	1.0233	0.0238	0.0002	1.0231	1.0235
lasso_screen_lasso	0.0556	1.0233	0.0238	0.0001	1.0232	1.0233
lasso_screen_xgboost50	0.0556	1.1265	0.0253	0.0263	1.1079	1.1451
SuperLearner	NA	1.0144	0.0236	0.0086	1.0083	1.0204

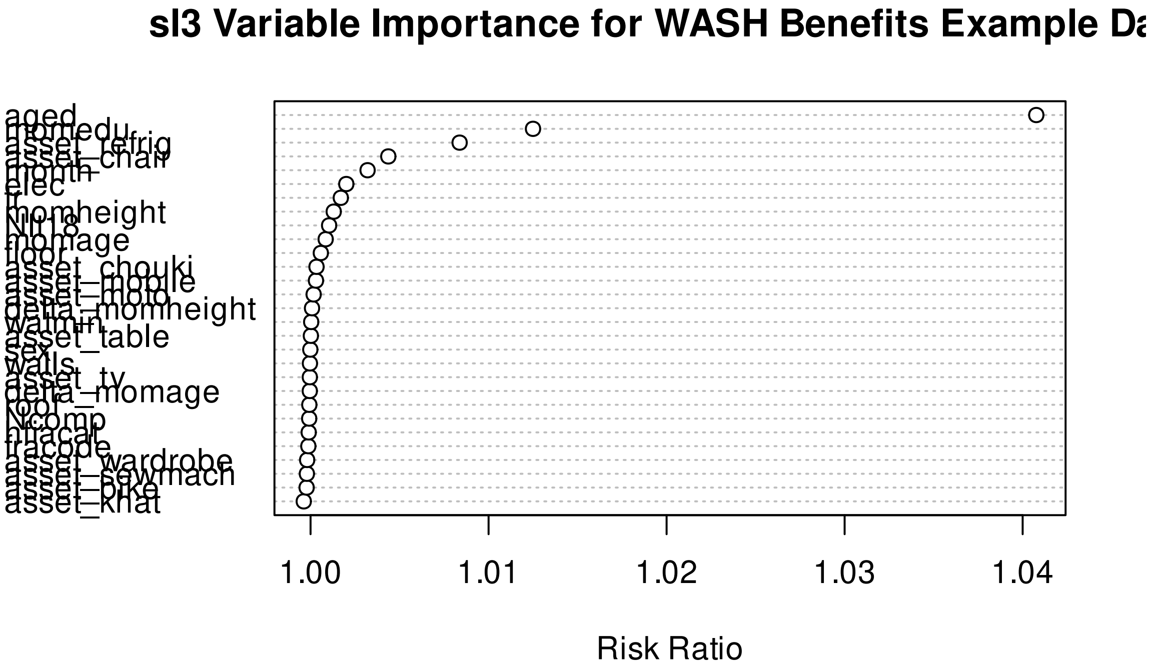
The measure of importance in `sl3` is based on a risk ratio, or risk difference, between the learner fit with a removed, or permuted, covariate and the learner fit with the true covariate, across all covariates. In this manner, the larger the risk difference, the more important the variable is in the prediction.

The intuition of this measure is that it calculates the risk (in terms of the average loss in predictive accuracy) of losing one covariate, while keeping everything else fixed, and compares it to the risk if the covariate was not lost. If this risk ratio is one, or risk difference is zero, then losing that covariate had no impact, and is thus not important by this measure. We do this across all of the covariates. As stated above, we can remove the covariate and refit the SL without it, or we just permute the covariate (faster) and hope for the shuffling to distort any meaningful information that was present in the covariate. This idea of permuting instead of removing saves a lot of time, and is also incorporated in the `randomForest` variable importance measures. However, the permutation approach is risky, so the importance function default is to remove and refit.

Let's explore the `sl3` variable importance measurements for the `washb` data.

```r
washb_varimp <- importance(sl_fit, loss = loss_squared_error, type = "permute")
washb_varimp %>%
  kable(digits = 4) %>%
  kableExtra:::kable_styling(fixed_thead = TRUE) %>%
  scroll_box(width = "100%", height = "300px")

# plot variable importance
importance_plot(
  washb_varimp,
  main = "sl3 Variable Importance for WASH Benefits Example Data"
)
```



X	risk_ratio
aged	1.0408
momedu	1.0125
asset_refrig	1.0084
asset_chair	1.0044
month	1.0032
elec	1.0020
tr	1.0017
momheight	1.0013
Nlt18	1.0010
momage	1.0008
floor	1.0006
asset_chouki	1.0003
asset_mobile	1.0003
asset_moto	1.0002
delta_momheight	1.0001
watmin	1.0000
asset_table	1.0000
sex	1.0000
walls	1.0000
asset_tv	1.0000
delta_momage	1.0000
roof	0.9999
Ncomp	0.9999
hfiacat	0.9999
fracode	0.9999
asset_wardrobe	0.9998
asset_sewmach	0.9998
asset_bike	0.9998
asset_khat	0.9996

6.2 Exercises

6.2.1 Predicting Myocardial Infarction with s13

Follow the steps below to predict myocardial infarction (mi) using the available covariate data. We thank Prof. David Benkeser at Emory University for making the this Cardiovascular Health Study (CHS) data accessible.

```
# load the data set
db_data <- url(
  paste0(
    "https://raw.githubusercontent.com/benkeser/sllecture/master/",
    "chspred.csv"
  )
)
chspred <- read_csv(file = db_data, col_names = TRUE)
```

Let's take a quick peek at the data:

waist	alcoh	hdl	beta	smoke	ace	ldl	bmi	aspirin	gend	age	estrgn	glu
110.164	0.0000	66.497	0	0	1	114.216	27.997	0	0	73.518	0	159.931
89.976	0.0000	50.065	0	0	0	103.777	20.893	0	0	61.772	0	153.389
106.194	8.4174	40.506	0	0	0	165.716	28.455	1	1	72.931	0	121.715
90.057	0.0000	36.175	0	0	0	45.203	23.961	0	0	79.119	0	53.969
78.614	2.9790	71.064	0	1	0	131.312	10.966	0	1	69.018	0	94.315
91.659	0.0000	59.496	0	0	0	171.187	29.132	0	1	81.835	0	212.907

1. Create an s13 task, setting myocardial infarction mi as the outcome and using all available covariate data.
2. Make a library of seven relatively fast base learning algorithms (i.e., do not consider BART or HAL). Customize hyperparameters for one of your learners. Feel free to use learners from s13 or SuperLearner. You may use the same base learning library that is presented above.
3. Incorporate at least one pipeline with feature selection. Any screener and learner(s) can be used.
4. Fit the metalearning step with the default metalearner.
5. With the metalearner and base learners, make the Super Learner (SL) and train it on the task.
6. Print the SL fit results by adding $cv_risk(loss_squared_error) to your fit object.

The squared error loss is specified here, since that's what is used by the default metalearner.

7. Cross-validate your SL fit to see how well it performs on unseen data. Specify the `loss_squared_error` loss function to evaluate the SL as it's the same loss that was used by the default metalearner. Print the result.
8. Use the `importance()` function to identify the "most important" predictor of myocardial infarction, according to `sl3` importance metrics. Print the result.

6.3 Concluding Remarks

- Super Learner (SL) is a general approach that can be applied to a diversity of estimation and prediction problems which can be defined by a loss function.

- It would be straightforward to plug in the estimator returned by SL into the target parameter mapping.

    - For example, suppose we are after the average treatment effect (ATE) of a binary treatment intervention: $\Psi_0 = E_{0,W}[E_0(Y|A=1,W) - E_0(Y|A=0,W)]$.
    - We could use the SL that was trained on the original data (let's call this `sl_fit`) to predict the outcome for all subjects under each intervention. All we would need to do is take the average difference between the counterfactual outcomes under each intervention of interest.
    - Considering Ψ_0 above, we would first need two n-length vectors of predicted outcomes under each intervention. One vector would represent the predicted outcomes under an intervention that sets all subjects to receive $A = 1$, $Y_i|A_i = 1, W_i$ for all $i = 1, \ldots, n$. The other vector would represent the predicted outcomes under an intervention that sets all subjects to receive $A = 0$, $Y_i|A_i = 0, W_i$ for all $i = 1, \ldots, n$.
    - After obtaining these vectors of counterfactual predicted outcomes, all we would need to do is average and then take the difference in order to "plug-in" the SL estimator into the target parameter mapping.
    - In `sl3` and with our current ATE example, this could be achieved with `mean(sl_fit$predict(A1_task))- mean(sl_fit$predict(A0_task))`; where `A1_task$data` would contain all 1's (or the level that pertains to receiving the treatment) for the treatment column in the data (keeping all else the same),

and `A0_task$data` would contain all 0's (or the level that pertains to not receiving the treatment) for the treatment column in the data.

- It's a worthwhile exercise to obtain the predicted counterfactual outcomes and create these counterfactual `sl3` tasks. It's too biased; however, to plug the SL fit into the target parameter mapping, (e.g., calling the result of `mean(sl_fit$predict(A1_task))- mean(sl_fit$predict(A0_task))` the estimated ATE. We would end up with an estimator for the ATE that was optimized for estimation of the prediction function, and not the ATE!
- At the end of the "analysis day", we want an estimator that is optimized for our target estimand of interest. We ultimately care about doing a good job estimating ψ_0. The SL is an essential step to help us get there. In fact, we will use the counterfactual predicted outcomes that were explained at length above. However, SL is not the end of the estimation procedure. Specifically, the Super Learner would not be an asymptotically linear estimator of the target estimand; and it is not an efficient substitution estimator. This begs the question, why is it so important for an estimator to possess these properties?
  - An asymptotically linear estimator converges to the estimand a $\frac{1}{\sqrt{n}}$ rate, thereby permitting formal statistical inference (i.e., confidence intervals and p-values) [ADD REF].
  - Substitution, or plug-in, estimators of the estimand are desirable because they respect both the local and global constraints of the statistical model (e.g., bounds), and have they have better finite-sample properties[ADD REF].
  - An efficient estimator is optimal in the sense that it has the lowest possible variance, and is thus the most precise. An estimator is efficient if and only if is asymptotically linear with influence curve equal to the canonical gradient [ADD REF].
    - The canonical gradient is a mathematical object that is specific to the target estimand, and it provides information on the level of difficulty of the estimation problem [ADD REF]. Various canonical gradient are shown in the chapters that follow.
    - Practitioner's do not need to know how to calculate a canonical gradient in order to understand efficiency and use Targeted Maximum Likelihood Estimation (TMLE). Metaphorically, you do not need to be Yoda in order to be a Jedi.

- TMLE is a general strategy that succeeds in constructing efficient and asymptotically linear plug-in estimators.

- SL is fantastic for pure prediction, and for obtaining an initial estimate in the first step of TMLE, but we need the second step of TMLE to have the desirable statistical properties mentioned above.

- In the chapters that follow, we focus on the targeted maximum likelihood estimator and the targeted minimum loss-based estimator, both referred to as TMLE.

6.4 Appendix

6.4.1 Exercise 1 Solution

Here is a potential solution to the sl3 Exercise 1 – Predicting Myocardial Infarction with sl3.

```
db_data <- url(
  "https://raw.githubusercontent.com/benkeser/sllecture/master/chspred.csv"
)
chspred <- read_csv(file = db_data, col_names = TRUE)

# make task
chspred_task <- make_sl3_Task(
  data = chspred,
  covariates = head(colnames(chspred), -1),
  outcome = "mi"
)

# make learners
glm_learner <- Lrnr_glm$new()
lasso_learner <- Lrnr_glmnet$new(alpha = 1)
ridge_learner <- Lrnr_glmnet$new(alpha = 0)
enet_learner <- Lrnr_glmnet$new(alpha = 0.5)
# curated_glm_learner uses formula = "mi ~ smoke + beta + waist"
curated_glm_learner <- Lrnr_glm_fast$new(covariates = c("smoke, beta, waist"))
mean_learner <- Lrnr_mean$new() # That is one mean learner!
glm_fast_learner <- Lrnr_glm_fast$new()
ranger_learner <- Lrnr_ranger$new()
svm_learner <- Lrnr_svm$new()
xgb_learner <- Lrnr_xgboost$new()
```

```
# screening
screen_cor <- make_learner(Lrnr_screener_correlation)
glm_pipeline <- make_learner(Pipeline, screen_cor, glm_learner)

# stack learners together
stack <- make_learner(
  Stack,
  glm_pipeline, glm_learner,
  lasso_learner, ridge_learner, enet_learner,
  curated_glm_learner, mean_learner, glm_fast_learner,
  ranger_learner, svm_learner, xgb_learner
)

# make and train SL
sl <- Lrnr_sl$new(
  learners = stack
)
sl_fit <- sl$train(chspred_task)
sl_fit$cv_risk(loss_squared_error)

CVsl <- CV_lrnr_sl(sl_fit, chspred_task, loss_squared_error)
CVsl

varimp <- importance(sl_fit)
importance_plot(varimp)
```

6.4.2 Exercise 2 Solution

Here is a potential solution to `sl3` Exercise 2 – Predicting Recurrent Ischemic Stroke in an RCT with `sl3`.

```
library(ROCR) # for AUC calculation

ist_data <- paste0(
  "https://raw.githubusercontent.com/tlverse/",
  "tlverse-handbook/master/data/ist_sample.csv"
) %>% fread()

# stack
ist_task <- make_sl3_Task(
  data = ist_data,
  outcome = "DRSISC",
  covariates = colnames(ist_data)[-which(names(ist_data) == "DRSISC")],
  drop_missing_outcome = TRUE
```

```
)

# learner library
lrn_glm <- Lrnr_glm$new()
lrn_lasso <- Lrnr_glmnet$new(alpha = 1)
lrn_ridge <- Lrnr_glmnet$new(alpha = 0)
lrn_enet <- Lrnr_glmnet$new(alpha = 0.5)
lrn_mean <- Lrnr_mean$new()
lrn_ranger <- Lrnr_ranger$new()
lrn_svm <- Lrnr_svm$new()
# xgboost grid
grid_params <- list(
  max_depth = c(2, 5, 8),
  eta = c(0.01, 0.15, 0.3)
)
grid <- expand.grid(grid_params, KEEP.OUT.ATTRS = FALSE)
params_default <- list(nthread = getOption("sl.cores.learners", 1))
xgb_learners <- apply(grid, MARGIN = 1, function(params_tune) {
  do.call(Lrnr_xgboost$new, c(params_default, as.list(params_tune)))
})
learners <- unlist(list(
  xgb_learners, lrn_ridge, lrn_mean, lrn_lasso,
  lrn_glm, lrn_enet, lrn_ranger, lrn_svm
),
recursive = TRUE
)

# SL
sl <- Lrnr_sl$new(learners)
sl_fit <- sl$train(ist_task)

# AUC
preds <- sl_fit$predict()
obs <- c(na.omit(ist_data$DRSISC))
AUC <- performance(prediction(sl_preds, obs), measure = "auc")@y.values[[1]]
plot(performance(prediction(sl_preds, obs), "tpr", "fpr"))

# CVsl
ist_task_CVsl <- make_sl3_Task(
  data = ist_data,
  outcome = "DRSISC",
  covariates = colnames(ist_data)[-which(names(ist_data) == "DRSISC")],
  drop_missing_outcome = TRUE,
  folds = origami::make_folds(
    n = sum(!is.na(ist_data$DRSISC)),
    fold_fun = folds_vfold,
```

```r
    V = 5
  )
)
CVsl <- CV_lrnr_sl(sl_fit, ist_task_CVsl, loss_loglik_binomial)
CVsl

# sl3 variable importance plot
ist_varimp <- importance(sl_fit, type = "permute")
ist_varimp %>%
  importance_plot(
    main = "Variable Importance for Predicting Recurrent Ischemic Stroke"
  )
```

7

The TMLE Framework

Jeremy Coyle

Based on the `tmle3` `R` package.

7.1 Learning Objectives

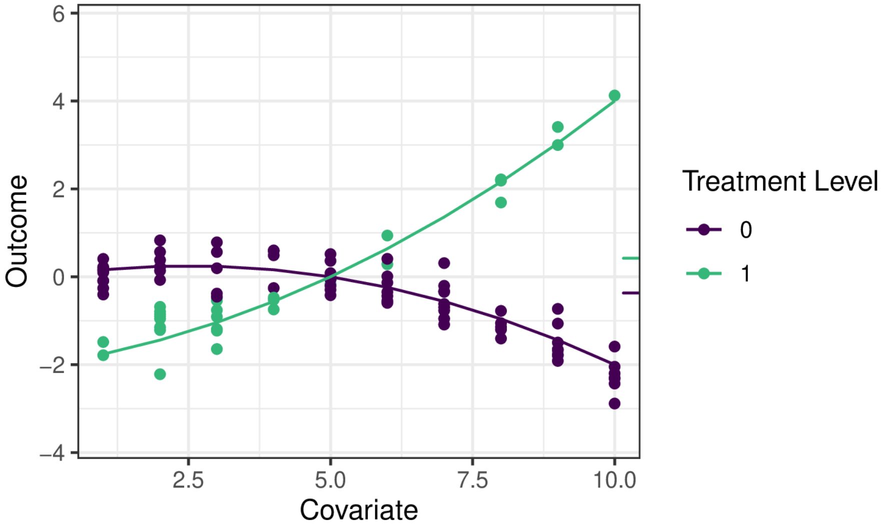
By the end of this chapter, you will be able to

1. Understand why we use TMLE for effect estimation.
2. Use `tmle3` to estimate an Average Treatment Effect (ATE).
3. Understand how to use `tmle3` "Specs" objects.
4. Fit `tmle3` for a custom set of target parameters.
5. Use the delta method to estimate transformations of target parameters.

7.2 Introduction

In the previous chapter on `sl3` we learned how to estimate a regression function like $\mathbb{E}[Y \mid X]$ from data. That's an important first step in learning from data, but how can we use this predictive model to estimate statistical and causal effects?

Going back to the roadmap for targeted learning, suppose we'd like to estimate the effect of a treatment variable A on an outcome Y. As discussed, one potential parameter that characterizes that effect is the Average Treatment Effect (ATE), defined as $\psi_0 = \mathbb{E}_W[\mathbb{E}[Y \mid A = 1, W] - \mathbb{E}[Y \mid A = 0, W]]$ and interpreted as the difference in mean outcome under when treatment $A = 1$ and $A = 0$, averaging over the distribution of covariates W. We'll illustrate several potential estimators for this parameter, and motivate the use of the TMLE (targeted maximum likelihood estimation; targeted minimum loss-based estimation) framework, using the following example data:



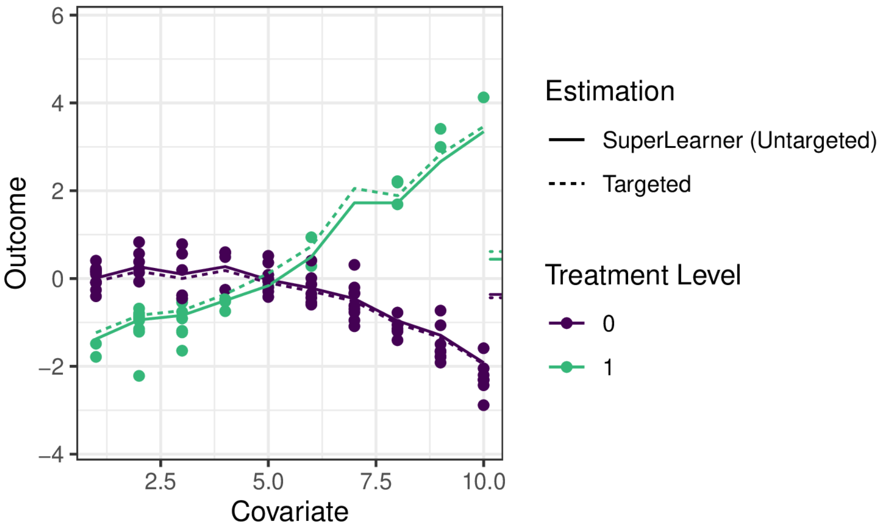
The small ticks on the right indicate the mean outcomes (averaging over W) under $A = 1$ and $A = 0$ respectively, so their difference is the quantity we'd like to estimate.

While we hope to motivate the application of TMLE in this chapter, we refer the interested reader to the two Targeted Learning books and associated works for full technical details.

7.3 Substitution Estimators

We can use `sl3` to fit a Super Learner or other regression model to estimate the outcome regression function $\mathbb{E}_0[Y \mid A, W]$, which we often refer to as $\overline{Q}_0(A,W)$ and whose estimate we denote $\overline{Q}_n(A,W)$. To construct an estimate of the ATE ψ_n, we need only "plug-in" the estimates of $\overline{Q}_n(A,W)$, evaluated at the two intervention contrasts, to the corresponding ATE "plug-in" formula: $\psi_n = \frac{1}{n}\sum(\overline{Q}_n(1,W) - \overline{Q}_n(0,W))$. This kind of estimator is called a ***plug-in*** or ***substitution*** estimator, since accurate estimates ψ_n of the parameter ψ_0 may be obtained by substituting estimates $\overline{Q}_n(A,W)$ for the relevant regression functions $\overline{Q}_0(A,W)$ themselves.

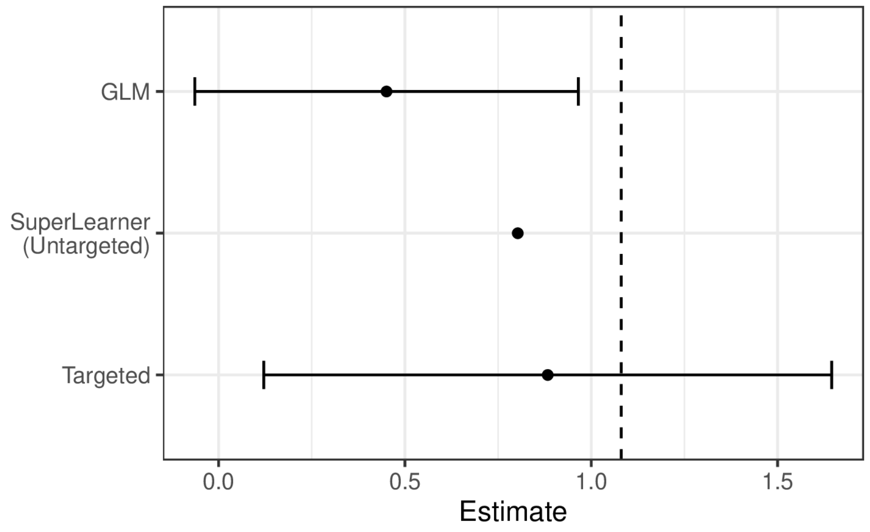
Applying `sl3` to estimate the outcome regression in our example, we can see that the ensemble machine learning predictions fit the data quite well:



The solid lines indicate the `sl3` estimate of the regression function, with the dotted lines indicating the `tmle3` updates (described below).

While substitution estimators are intuitive, naively using this approach with a Super Learner estimate of $\overline{Q}_0(A,W)$ has several limitations. First, Super Learner is selecting learner weights to minimize risk across the entire regression function, instead of "targeting" the ATE parameter we hope to estimate, leading to biased estimation. That is, `sl3` is trying to do well on the full regression curve on the left, instead of focusing on the small ticks on the right. What's more, the sampling distribution of this approach is not asymptotically linear, and therefore inference is not possible.

We can see these limitations illustrated in the estimates generated for the example data:



We see that Super Learner, estimates the true parameter value (indicated by the dashed vertical line) more accurately than GLM. However, it is still less accurate than TMLE, and valid inference is not possible. In contrast, TMLE achieves a less biased estimator and valid inference.

7.4 Targeted Maximum Likelihood Estimation

TMLE takes an initial estimate $\overline{Q}_n(A, W)$ as well as an estimate of the propensity score $g_n(A \mid W) = \mathbb{P}(A = 1 \mid W)$ and produces an updated estimate $\overline{Q}^{\star}_n(A, W)$ that is "targeted" to the parameter of interest. TMLE keeps the benefits of substitution estimators (it is one), but augments the original, potentially erratic estimates to *correct for bias* while also resulting in an *asymptotically linear* (and thus normally distributed) estimator that accommodates inference via asymptotically consistent Wald-style confidence intervals.

7.4.1 TMLE Updates

There are different types of TMLEs (and, sometimes, multiple for the same set of target parameters) – below, we give an example of the algorithm for TML estimation of the ATE. $\overline{Q}^{\star}_n(A, W)$ is the TMLE-augmented estimate $f(\overline{Q}^{\star}_n(A, W)) = f(\overline{Q}_n(A, W)) + \epsilon \cdot$

$H_n(A, W)$, where $f(\cdot)$ is the appropriate link function (e.g., $\text{logit}(x) = \log(x/(1 - x))$), and an estimate ϵ_n of the coefficient ϵ of the "clever covariate" $H_n(A, W)$ is computed. The form of the covariate $H_n(A, W)$ differs across target parameters; in this case of the ATE, it is $H_n(A, W) = \frac{A}{g_n(A|W)} - \frac{1-A}{1-g_n(A,W)}$, with $g_n(A, W) = \mathbb{P}(A = 1 \mid W)$ being the estimated propensity score, so the estimator depends both on the initial fit (by sl3) of the outcome regression ($\overline{Q}_n$) and of the propensity score (g_n).

There are several robust augmentations that are used across the tlverse, including the use of an additional layer of cross-validation to avoid over-fitting bias (i.e., CV-TMLE) as well as approaches for more consistently estimating several parameters simultaneously (e.g., the points on a survival curve).

7.4.2 Statistical Inference

Since TMLE yields an **asymptotically linear** estimator, obtaining statistical inference is very convenient. Each TML estimator has a corresponding **(efficient) influence function** (often, "EIF", for short) that describes the asymptotic distribution of the estimator. By using the estimated EIF, Wald-style inference (asymptotically correct confidence intervals) can be constructed simply by plugging into the form of the EIF our initial estimates $\overline{Q}_n$ and g_n, then computing the sample standard error.

The following sections describe both a simple and more detailed way of specifying and estimating a TMLE in the tlverse. In designing tmle3, we sought to replicate as closely as possible the very general estimation framework of TMLE, and so each theoretical object relevant to TMLE is encoded in a corresponding software object/method. First, we will present the simple application of tmle3 to the WASH Benefits example, and then go on to describe the underlying objects in greater detail.

7.5 Easy-Bake Example: tmle3 for ATE

We'll illustrate the most basic use of TMLE using the WASH Benefits data introduced earlier and estimating an average treatment effect.

7.5.1 Load the Data

We'll use the same WASH Benefits data as the earlier chapters:

```
library(data.table)
library(dplyr)
library(tmle3)
library(sl3)
washb_data <- fread(
  paste0(
    "https://raw.githubusercontent.com/tlverse/tlverse-data/master/",
    "wash-benefits/washb_data.csv"
  ),
  stringsAsFactors = TRUE
)
```

7.5.2 Define the variable roles

We'll use the common W (covariates), A (treatment/intervention), Y (outcome) data structure. `tmle3` needs to know what variables in the dataset correspond to each of these roles. We use a list of character vectors to tell it. We call this a "Node List" as it corresponds to the nodes in a Directed Acyclic Graph (DAG), a way of displaying causal relationships between variables.

```
node_list <- list(
  W = c(
    "month", "aged", "sex", "momage", "momedu",
    "momheight", "hfiacat", "Nlt18", "Ncomp", "watmin",
    "elec", "floor", "walls", "roof", "asset_wardrobe",
    "asset_table", "asset_chair", "asset_khat",
    "asset_chouki", "asset_tv", "asset_refrig",
    "asset_bike", "asset_moto", "asset_sewmach",
    "asset_mobile"
  ),
  A = "tr",
  Y = "whz"
)
```

7.5.3 Handle Missingness

Currently, missingness in `tmle3` is handled in a fairly simple way:

- Missing covariates are median- (for continuous) or mode- (for discrete) imputed, and additional covariates indicating imputation are generated, just as described in the `sl3` chapter.

- Missing treatment variables are excluded – such observations are dropped.
- Missing outcomes are efficiently handled by the automatic calculation (and incorporation into estimators) of *inverse probability of censoring weights* (IPCW); this is also known as IPCW-TMLE and may be thought of as a joint intervention to remove missingness and is analogous to the procedure used with classical inverse probability weighted estimators.

These steps are implemented in the `process_missing` function in `tmle3`:

```
processed <- process_missing(washb_data, node_list)
washb_data <- processed$data
node_list <- processed$node_list
```

7.5.4 Create a "Spec" Object

`tmle3` is general, and allows most components of the TMLE procedure to be specified in a modular way. However, most users will not be interested in manually specifying all of these components. Therefore, `tmle3` implements a `tmle3_Spec` object that bundles a set of components into a *specification* ("Spec") that, with minimal additional detail, can be run to fit a TMLE.

We'll start with using one of the specs, and then work our way down into the internals of `tmle3`.

```
ate_spec <- tmle_ATE(
  treatment_level = "Nutrition + WSH",
  control_level = "Control"
)
```

7.5.5 Define the learners

Currently, the only other thing a user must define are the `sl3` learners used to estimate the relevant factors of the likelihood: Q and g.

This takes the form of a list of `sl3` learners, one for each likelihood factor to be estimated with `sl3`:

```
# choose base learners
lrnr_mean <- make_learner(Lrnr_mean)
lrnr_rf <- make_learner(Lrnr_ranger)

# define metalearners appropriate to data types
```

```
ls_metalearner <- make_learner(Lrnr_nnls)
mn_metalearner <- make_learner(
  Lrnr_solnp, metalearner_linear_multinomial,
  loss_loglik_multinomial
)
sl_Y <- Lrnr_sl$new(
  learners = list(lrnr_mean, lrnr_rf),
  metalearner = ls_metalearner
)
sl_A <- Lrnr_sl$new(
  learners = list(lrnr_mean, lrnr_rf),
  metalearner = mn_metalearner
)
learner_list <- list(A = sl_A, Y = sl_Y)
```

Here, we use a Super Learner as defined in the previous chapter. In the future, we plan to include reasonable defaults learners.

7.5.6 Fit the TMLE

We now have everything we need to fit the tmle using `tmle3`:

```
tmle_fit <- tmle3(ate_spec, washb_data, node_list, learner_list)
print(tmle_fit)
A tmle3_Fit that took 1 step(s)
   type                                   param  init_est tmle_est       se
1:  ATE ATE[Y_{A=Nutrition + WSH}-Y_{A=Control}] -0.005231  0.00812 0.050679
       lower   upper psi_transformed lower_transformed upper_transformed
1: -0.091208 0.10745         0.00812         -0.091208           0.10745
```

7.5.7 Evaluate the Estimates

We can see the summary results by printing the fit object. Alternatively, we can extra results from the summary by indexing into it:

```
estimates <- tmle_fit$summary$psi_transformed
print(estimates)
[1] 0.00812
```

7.6 tmle3 Components

Now that we've successfully used a spec to obtain a TML estimate, let's look under the hood at the components. The spec has a number of functions that generate the objects necessary to define and fit a TMLE.

7.6.1 tmle3_task

First is, a tmle3_Task, analogous to an sl3_Task, containing the data we're fitting the TMLE to, as well as an NPSEM generated from the node_list defined above, describing the variables and their relationships.

```
tmle_task <- ate_spec$make_tmle_task(washb_data, node_list)

tmle_task$npsem
$W
tmle3_Node: W
    Variables: month, aged, sex, momedu, hfiacat, Nlt18, Ncomp, watmin, elec, floor, walls,
    Parents:

$A
tmle3_Node: A
    Variables: tr
    Parents: W

$Y
tmle3_Node: Y
    Variables: whz
    Parents: A, W
```

7.6.2 Initial Likelihood

Next, is an object representing the likelihood, factorized according to the NPSEM described above:

```
initial_likelihood <- ate_spec$make_initial_likelihood(
  tmle_task,
  learner_list
)
print(initial_likelihood)
W: Lf_emp
```

```
A: LF_fit
Y: LF_fit
```

These components of the likelihood indicate how the factors were estimated: the marginal distribution of W was estimated using NP-MLE, and the conditional distributions of A and Y were estimated using `sl3` fits (as defined with the `learner_list`) above.

We can use this in tandem with the `tmle_task` object to obtain likelihood estimates for each observation:

```
initial_likelihood$get_likelihoods(tmle_task)
               W       A        Y
   1: 0.00021299 0.34925 -0.35834
   2: 0.00021299 0.36117 -0.93261
   3: 0.00021299 0.34740 -0.80873
   4: 0.00021299 0.34248 -0.94020
   5: 0.00021299 0.34134 -0.57866
  ---
4691: 0.00021299 0.23375 -0.58997
4692: 0.00021299 0.23366 -0.22769
4693: 0.00021299 0.22660 -0.74235
4694: 0.00021299 0.28944 -0.91796
4695: 0.00021299 0.19533 -1.03878
```

7.6.3 Targeted Likelihood (updater)

We also need to define a “Targeted Likelihood” object. This is a special type of likelihood that is able to be updated using an `tmle3_Update` object. This object defines the update strategy (e.g., submodel, loss function, CV-TMLE or not).

```
targeted_likelihood <- Targeted_Likelihood$new(initial_likelihood)
```

When constructing the targeted likelihood, you can specify different update options. See the documentation for `tmle3_Update` for details of the different options. For example, you can disable CV-TMLE (the default in `tmle3`) as follows:

```
targeted_likelihood_no_cv <-
  Targeted_Likelihood$new(initial_likelihood,
    updater = list(cvtmle = FALSE)
  )
```

7.6.4 Parameter Mapping

Finally, we need to define the parameters of interest. Here, the spec defines a single parameter, the ATE. In the next section, we'll see how to add additional parameters.

```
tmle_params <- ate_spec$make_params(tmle_task, targeted_likelihood)
print(tmle_params)
[[1]]
Param_ATE: ATE[Y_{A=Nutrition + WSH}-Y_{A=Control}]
```

7.6.5 Putting it all together

Having used the spec to manually generate all these components, we can now manually fit a tmle3:

```
tmle_fit_manual <- fit_tmle3(
  tmle_task, targeted_likelihood, tmle_params,
  targeted_likelihood$updater
)
print(tmle_fit_manual)
A tmle3_Fit that took 1 step(s)
   type                                    param   init_est tmle_est       se
1:  ATE ATE[Y_{A=Nutrition + WSH}-Y_{A=Control}] -0.0045451  0.01174 0.050807
      lower   upper psi_transformed lower_transformed upper_transformed
1: -0.08784 0.11132         0.01174          -0.08784           0.11132
```

The result is equivalent to fitting using the tmle3 function as above.

7.7 Fitting tmle3 with multiple parameters

Above, we fit a tmle3 with just one parameter. tmle3 also supports fitting multiple parameters simultaneously. To illustrate this, we'll use the tmle_TSM_all spec:

```
tsm_spec <- tmle_TSM_all()
targeted_likelihood <- Targeted_Likelihood$new(initial_likelihood)
all_tsm_params <- tsm_spec$make_params(tmle_task, targeted_likelihood)
print(all_tsm_params)
[[1]]
Param_TSM: E[Y_{A=Control}]

[[2]]
```

```
Param_TSM: E[Y_{A=Handwashing}]

[[3]]
Param_TSM: E[Y_{A=Nutrition}]

[[4]]
Param_TSM: E[Y_{A=Nutrition + WSH}]

[[5]]
Param_TSM: E[Y_{A=Sanitation}]

[[6]]
Param_TSM: E[Y_{A=WSH}]

[[7]]
Param_TSM: E[Y_{A=Water}]
```

This spec generates a Treatment Specific Mean (TSM) for each level of the exposure variable. Note that we must first generate a new targeted likelihood, as the old one was targeted to the ATE. However, we can recycle the initial likelihood we fit above, saving us a super learner step.

7.7.1 Delta Method

We can also define parameters based on Delta Method Transformations of other parameters. For instance, we can estimate a ATE using the delta method and two of the above TSM parameters:

```
ate_param <- define_param(
  Param_delta, targeted_likelihood,
  delta_param_ATE,
  list(all_tsm_params[[1]], all_tsm_params[[4]])
)
print(ate_param)
Param_delta: E[Y_{A=Nutrition + WSH}] - E[Y_{A=Control}]
```

This can similarly be used to estimate other derived parameters like Relative Risks, and Population Attributable Risks

7.7.2 Fit

We can now fit a TMLE simultaneously for all TSM parameters, as well as the above defined ATE parameter

```
all_params <- c(all_tsm_params, ate_param)

tmle_fit_multiparam <- fit_tmle3(
  tmle_task, targeted_likelihood, all_params,
  targeted_likelihood$updater
)

print(tmle_fit_multiparam)
A tmle3_Fit that took 1 step(s)
   type                                       param   init_est  tmle_est
1:  TSM                            E[Y_{A=Control}] -0.5959678 -0.620830
2:  TSM                        E[Y_{A=Handwashing}] -0.6188184 -0.660230
3:  TSM                          E[Y_{A=Nutrition}] -0.6111402 -0.606586
4:  TSM                    E[Y_{A=Nutrition + WSH}] -0.6005128 -0.608949
5:  TSM                         E[Y_{A=Sanitation}] -0.5857464 -0.578472
6:  TSM                                E[Y_{A=WSH}] -0.5205610 -0.448252
7:  TSM                              E[Y_{A=Water}] -0.5657364 -0.537709
8:  ATE E[Y_{A=Nutrition + WSH}] - E[Y_{A=Control}] -0.0045451  0.011881
         se     lower    upper psi_transformed lower_transformed
1: 0.029901 -0.679435 -0.56223       -0.620830         -0.679435
2: 0.041719 -0.741998 -0.57846       -0.660230         -0.741998
3: 0.042047 -0.688996 -0.52418       -0.606586         -0.688996
4: 0.041285 -0.689867 -0.52803       -0.608949         -0.689867
5: 0.042396 -0.661566 -0.49538       -0.578472         -0.661566
6: 0.045506 -0.537442 -0.35906       -0.448252         -0.537442
7: 0.039253 -0.614644 -0.46077       -0.537709         -0.614644
8: 0.050801 -0.087688  0.11145        0.011881         -0.087688
   upper_transformed
1:          -0.56223
2:          -0.57846
3:          -0.52418
4:          -0.52803
5:          -0.49538
6:          -0.35906
7:          -0.46077
8:           0.11145
```

7.8 Exercises

7.8.1 Estimation of the ATE with tmle3

Follow the steps below to estimate an average treatment effect using data from the Collaborative Perinatal Project (CPP), available in the s13 package. To simplify this example, we define a binary intervention variable, parity01 – an indicator of having one or more children before the current child and a binary outcome, haz01 – an indicator of having an above average height for age.

```
# load the data set
data(cpp)
cpp <- cpp %>%
  as_tibble() %>%
  dplyr::filter(!is.na(haz)) %>%
  mutate(
    parity01 = as.numeric(parity > 0),
    haz01 = as.numeric(haz > 0)
  )
```

1. Define the variable roles (W, A, Y) by creating a list of these nodes. Include the following baseline covariates in W: apgar1, apgar5, gagebrth, mage, meducyrs, sexn. Both A and Y are specified above. The missingness in the data (specifically, the missingness in the columns that are specified in the node list) will need to be taking care of. The process_missing function can be used to accomplish this, like the washb_data example above.
2. Define a tmle3_Spec object for the ATE, tmle_ATE().
3. Using the same base learning libraries defined above, specify s13 base learners for estimation of $\overline{Q}_0 = \mathbb{E}_0(Y \mid A, W)$ and $g_0 = \mathbb{P}(A = 1 \mid W)$.
4. Define the metalearner like below.

```
metalearner <- make_learner(
  Lrnr_solnp,
  loss_function = loss_loglik_binomial,
  learner_function = metalearner_logistic_binomial
)
```

5. Define one super learner for estimating $\overline{Q}_0$ and another for estimating g_0. Use the metalearner above for both super learners.

6. Create a list of the two super learners defined in the step above and call this object `learner_list`. The list names should be `A` (defining the super learner for estimation of g_0) and `Y` (defining the super learner for estimation of $\overline{Q}_0$).
7. Fit the TMLE with the `tmle3` function by specifying (1) the `tmle3_Spec`, which we defined in Step 2; (2) the data; (3) the list of nodes, which we specified in Step 1; and (4) the list of super learners for estimation of g_0 and $\overline{Q}_0$, which we defined in Step 6. *Note*: Like before, you will need to explicitly make a copy of the data (to work around `data.table` optimizations), e.g., (`cpp2 <-data.table::copy(cpp)`), then use the `cpp2` data going forward.

7.8.2 Estimation of Strata-Specific ATEs with `tmle3`

For this exercise, we will work with a random sample of 5,000 patients who participated in the International Stroke Trial (IST). This data is described in the Chapter 3.2 of the `tlverse` handbook. We included the data below and a summarized description that is relevant for this exercise.

The outcome, Y, indicates recurrent ischemic stroke within 14 days after randomization (`DRSISC`); the treatment of interest, A, is the randomized aspirin vs. no aspirin treatment allocation (`RXASP` in `ist`); and the adjustment set, W, consists simply of other variables measured at baseline. In this data, the outcome is occasionally missing, but there is no need to create a variable indicating this missingness (such as Δ) for analyses in the `tlverse`, since the missingness is automatically detected when `NA` are present in the outcome. Covariates with missing values (`RATRIAL`, `RASP3` and `RHEP24`) have already been imputed. Additional covariates were created (`MISSING_RATRIAL_RASP3` and `MISSING_RHEP24`), which indicate whether or not the covariate was imputed. The missingness was identical for `RATRIAL` and `RASP3`, which is why only one covariate indicating imputation for these two covariates was created.

1. Estimate the average effect of randomized aspirin treatment (`RXASP` = 1) on recurrent ischemic stroke. Even though the missingness mechanism on Y, Δ, does not need to be specified in the node list, it does still need to be accounted for in the TMLE. In other words, for this estimation problem, Δ is a relevant factor of the likelihood. Thus, when defining the list of `sl3` learners for each likelihood factor, be sure to include a list of learners for estimation of Δ, say `sl_Delta`, and specify this in the learner list, like so `learner_list <-list(A = sl_A, delta_Y = sl_Delta, Y = sl_Y)`.
2. Recall that this RCT was conducted internationally. Suppose there is concern that the dose of aspirin may have varied across geographical regions, and an average across

all geographical regions may not be warranted. Calculate the strata specific ATEs according to geographical region (REGION).

```
ist_data <- fread(
  paste0(
    "https://raw.githubusercontent.com/tlverse/deming2019-workshop/",
    "master/data/ist_sample.csv"
  )
)
```

7.9 Summary

tmle3 is a general purpose framework for generating TML estimates. The easiest way to use it is to use a predefined spec, allowing you to just fill in the blanks for the data, variable roles, and sl3 learners. However, digging under the hood allows users to specify a wide range of TMLEs. In the next sections, we'll see how this framework can be used to estimate advanced parameters such as optimal treatments and stochastic shift interventions.

8

Optimal Individualized Treatment Regimes

Ivana Malenica

Based on the `tmle3mopttx` R package by *Ivana Malenica, Jeremy Coyle, and Mark van der Laan*.

Updated: 2021-12-30

8.1 Learning Objectives

1. Differentiate dynamic and optimal dynamic treatment interventions from static interventions.
2. Explain the benefits, and challenges, associated with using optimal individualized treatment regimes in practice.
3. Contrast the impact of implementing an optimal individualized treatment regime in the population with the impact of implementing static and dynamic treatment regimes in the population.
4. Estimate causal effects under optimal individualized treatment regimes with the `tmle3mopttx` R package.
5. Assess the mean under optimal individualized treatment with resource constraints.
6. Implement optimal individualized treatment rules based on sub-optimal rules, or "simple" rules, and recognize the practical benefit of these rules.
7. Construct "realistic" optimal individualized treatment regimes that respect real data and subject-matter knowledge limitations on interventions by only considering interventions that are supported by the data.
8. Interpret the estimated optimal individualized treatment rule.
9. Measure variable importance as defined in terms of the optimal individualized treatment interventions.

8.2 Introduction to Optimal Individualized Interventions

Identifying which intervention will be effective for which patient based on lifestyle, genetic and environmental factors is a common goal in precision medicine. To put it in context, Abacavir and Tenofovir are commonly prescribed as part of the antiretroviral therapy to Human Immunodeficiency Virus (HIV) patients. However, not all individuals benefit from the two medications equally. In particular, patients with renal dysfunction might further deteriorate if prescribed Tenofovir, due to the high nephrotoxicity caused by the medication. While Tenofovir is still highly effective treatment option for HIV patients, in order to maximize the patient's well-being, it would be beneficial to prescribe Tenofovir only to individuals with healthy kidney function. As an another example, consider a HIV trial where our goal is to improve retention in HIV care. In a randomized clinical trial, several interventions show efficacy- including appointment reminders through text messages, small cash incentives for on time clinic visits, and peer health workers. Ideally, we want to improve effectiveness by assigning each patient the intervention they are most likely to benefit from, as well as improve efficiency by not allocating resources to individuals that do not need them, or would not benefit from it.

One opts to administer the intervention to individuals who will profit from it, instead of assigning treatment on a population level. But how do we know which intervention works for which patient? This aim motivates a different type of intervention, as opposed to the static exposures we described in previous chapters. In particular, in this chapter we learn about dynamic or "individualized" interventions that tailor the treatment decision based on the collected covariates. Formally, dynamic treatments represent interventions that at each treatment-decision stage are allowed to respond to the currently available treatment and covariate history. A dynamic treatment rule can be thought of as a rule where the input is the available set of collected covariates, and the output is an individualized treatment for each patient (Bembom and van der Laan, 2007, Robins (1986a), Chakraborty and Moodie (2013)).

In the statistics community such a treatment strategy is termed an **individualized treatment regime** (ITR), also known as the optimal dynamic treatment rule, optimal treatment regime, optimal strategy, and optimal policy (Murphy, 2003, Robins (2004)). The (counterfactual) population mean outcome under an ITR is the value of the ITR (Murphy, 2003, Robins (2004)). Even more, suppose one wishes to maximize the population mean of an outcome, where for each individual we have access to some set of measured covariates. This means, for example, that we can learn for which individual characteristics assigning treatment increases the probability of a beneficial outcome. An

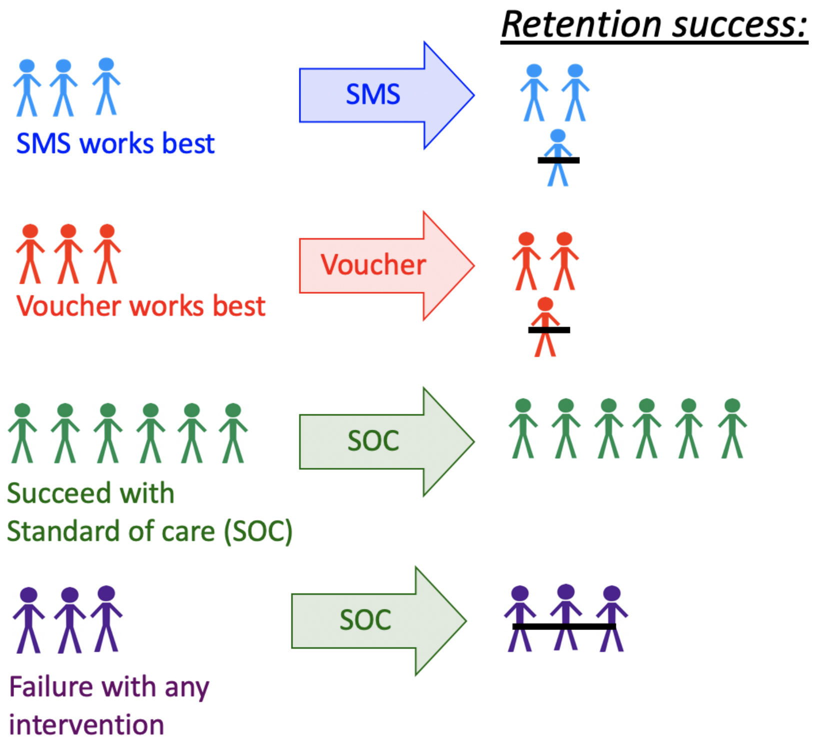


Figure 8.1: Dynamic Treatment Regime in a Clinical Setting

ITR with the maximal value is referred to as an optimal ITR or the **optimal individualized treatment**. Consequently, the value of an optimal ITR is termed the optimal value, or the **mean under the optimal individualized treatment**.

The problem of estimating the optimal individualized treatment has received much attention in the statistics literature over the years, especially with the advancement of precision medicine; see Murphy (2003), Robins (2004), Zhang et al. (2016), Zhao et al. (2012), Chakraborty and Moodie (2013) and Robins and Rotnitzky (2014) to name a few. However, much of the early work depends on parametric assumptions. As such, even in a randomized trial, the statistical inference for the optimal individualized treatment relies on assumptions that are generally believed to be false, and can lead to biased results.

In this chapter, we consider estimation of the mean outcome under the optimal individualized treatment where the candidate rules are restricted to depend only on user-supplied subset of the baseline covariates. The estimation problem is addressed in a statistical model for the data distribution that is nonparametric, and at most places restrictions on the probability of a patient receiving treatment given covariates (as in a randomized trial). As such, we don't need to make any assumptions about the relationship of the outcome with the treatment and covariates, or the relationship between the treatment and covariates. Further, we provide a Targeted Maximum Likelihood Estimator for the mean under the optimal individualized treatment that allows us to generate valid inference for our parameter, without having any parametric assumptions.

In the following, we provide a brief overview of the methodology with a focus on building intuition for the target parameter and its importance — aided with simulations, data examples and software demonstrations. For more information on the technical aspects of the algorithm, further practical advice and overview, the interested reader is invited to additionally consult van der Laan and Luedtke (2015), Luedtke and van der Laan (2016a), Montoya et al. (2021b) and Montoya et al. (2021a).

8.3 Data Structure and Notation

Suppose we observe n independent and identically distributed observations of the form $O = (W, A, Y) \sim P_0$. We denote A as categorical treatment, and Y as the final outcome. In particular, we define $A \in \mathcal{A}$ where $\mathcal{A} \equiv \{a_1, \cdots, a_{n_A}\}$ and $n_A = |\mathcal{A}|$, with n_A denoting the number of categories (possibly only two, for a binary setup). Note that we treat W as vector-valued, representing all of our collected baseline covariates. Therefore, for a single random individual i, we have that their observed data is O_i: with corresponding baseline covariates W_i, treatment A_i, and final outcome Y_i. Let $O^n = \{O_i\}_{i=1}^n$ denote n observed samples. Then, we say that $O^n \sim P_0$, or that all data was drawn from some true probability distribution P_0. Let $\mathcal{M}$ denote a statistical model for the probability distribution of the data that is nonparametric, beyond possible knowledge of the treatment mechanism. In words, this means that we make no assumptions on the relationship between variables, but might be able to say something about the relationship of A and W, as is the case in a randomized trial. In general, the more we know, or are willing to assume about the experiment that produces the data, the smaller the model. The true data generating distribution P_0 is part of the statistical model $\mathcal{M}$, and we write $P_0 \in \mathcal{M}$. As in previous chapters, we denote P_n as the empirical distribution which gives each observation weight $1/n$.

We use the structural equation model (SEM) in order to define the process that gives rise to the observed (endogenous) and not observed (exogenous) variables, as described by Pearl (2009). In particular, we denote $U = (U_W, U_A, U_Y)$ as the exogenous random variables, drawn from $U \sim P_U$. The endogenous variables, written as $O = (W, A, Y)$, correspond to the observed data. We can define the relationships between variables with the following structural equations:

$$W = f_W(U_W) \tag{8.1}$$
$$A = f_A(W, U_A) \tag{8.2}$$
$$Y = f_Y(A, W, U_Y), \tag{8.3}$$

where the collection $f = (f_W, f_A, f_Y)$ denotes unspecified functions, beyond possible knowledge of the treatment mechanism function, f_A. Note that in the case of a randomized trial, we can write the above NPSEM as

$$W = f_W(U_W) \tag{8.4}$$
$$A = U_A \tag{8.5}$$
$$Y = f_Y(A, W, U_Y), \tag{8.6}$$

where U_A has a known distribution and U_A is independent of U_W. We will discuss this more in later sections on identifiability.

The likelihood of the data admits a factorization, implied by the time ordering of O. We denote the true density of O as p_0, corresponding to the distribution P_0 and dominating measure μ.

$$p_0(O) = p_{Y,0}(Y \mid A, W) p_{A,0}(A \mid W) p_{W,0}(W) = q_{Y,0}(Y \mid A, W) g_{A,0}(A \mid W) q_{W,0}(W), \tag{8.7}$$

where $p_{Y,0}(Y|A, W)$ is the conditional density of Y given (A, W) with respect to some dominating measure μ_Y, $p_{A,0}$ is the conditional density of A given W with respect to a counting measure μ_A, and $p_{W,0}$ is the density of W with respect to dominating measure μ_W. In order to match relevant Targeted Learning literature, we also write $P_{Y,0}(Y \mid A, W) = Q_{Y,0}(Y \mid A, W)$, $P_{A,0}(A \mid W) = g_0(A \mid W)$ and $P_{W,0}(W) = Q_{W,0}(W)$ as the corresponding conditional distribution of Y given (A, W), treatment mechanism A given W, and distribution of baseline covariates. For notational simplicity, we additionally define $\bar{Q}_{Y,0}(A, W) \equiv \mathbb{E}_0[Y \mid A, W]$ as the conditional expectation of Y given (A, W).

Lastly, we define V as a subset of the baseline covariates the optimal individualized rule depends on, where $V \in W$. Note that V could be all of W, or an empty set, depending on the subject matter knowledge. In particular, a researcher might want to consider known effect modifiers available at the time of treatment decision as possible V covariates, or consider dynamic treatment rules based on measurments that can be easily obtained in a clinical setting. Defining V as a more restrictive set of baseline covariates allows us to consider possibly sub-optimal rules that are easier to estimate, and thereby allows for statistical inference for the counterfactual mean outcome under the sub-optimal rule; we will elaborate on this in later sections.

8.4 Defining the Causal Effect of an Optimal Individualized Intervention

Consider dynamic treatment rules, denoted as d, in the set of all possible rules $\mathcal{D}$. Then, in a point treatment setting, d is a deterministic function that takes as input V and outputs a treatment decision where $V \to d(V) \in \{a_1, \cdots, a_{n_A}\}$. We will use dynamic treatment rules, and the corresponding treatment decision, to describe an intervention on the treatment mechanism and the corresponding outcome under a dynamic treatment rule.

As mentioned in the previous section, causal effects are defined in terms of hypothetical interventions on the SEM (8.3). For a given rule d, our modified system then takes the following form:

$$\begin{aligned} W &= f_W(U_W) && (8.8)\\ A &= d(V) && (8.9)\\ Y_{d(V)} &= f_Y(d(V), W, U_Y), && (8.10) \end{aligned}$$

where the dynamic treatment regime may be viewed as an intervention in which A is set equal to a value based on a hypothetical regime $d(V)$. The couterfactual outcome $Y_{d(V)}$ denotes the outcome for a patient had their treatment been assigned using the dynamic rule $d(V)$, possibly contrary to the fact. Similarly, the counterfactual outcomes had all patients been assigned treatment ($A = 1$), or given control ($A = 0$), are written as Y_1 and Y_0. Finally, we denote the distribution of the counterfactual outcomes as $P_{U,X}$, implied by the distribution of exogenous variables U and structural equations f. The set of all possible counterfactual distributions are encompased by the causal model $\mathcal{M}^F$, where $P_{U,X} \in \mathcal{M}^F$.

The goal of any causal analysis motivated by such dynamic interventions is to estimate a parameter defined as the counterfactual mean of the outcome with respect to the modified intervention distribution. That is, subject's outcome if, possibly contrary to the fact, the subject received treatment that would have been assigned by rule $d(V)$. Equivalently, we ask the following causal question: "What is the expected outcome had every subject received treatment according to the (optimal) individualized treatment?" In order to estimate the optimal individualized treatment, we set the following optimization problem:

$$d_{opt}(V) \equiv \text{argmax}_{d(V) \in \mathcal{D}} \mathbb{E}_{P_{U,X}}[Y_{d(V)}],$$

where the optimal individualized rule is the rule with the maximal value. We note that, in case the problem at hand requires minimizing the mean of an outcome, our optimal individualized rule will be the rule with the minimal value instead.

With that in mind, we can consider different treatment rules, all in the set $\mathcal{D}$:

1. The true rule, $d_{0,\text{opt}}$, and the corresponding causal parameter $\mathbb{E}_{U,X}[Y_{d_{0,\text{opt}}(V)}]$ denoting the expected outcome under the true optimal treatment rule $d_{0,\text{opt}}(V)$.
2. The estimated rule, $d_{n,\text{opt}}$, and the corresponding causal parameter $\mathbb{E}_{U,X}[Y_{d_{n,\text{opt}}(V)}]$ denoting the expected outcome under the estimated optimal treatment rule $d_{n,\text{opt}}(V)$.

In this chapter, we will focus on the value under the estimated optimal rule $d_{n,\text{opt}}$, a **data-adaptive parameter**. Note that its true value depends on the sample! Finally, our causal target parameter of interest is the expected outcome under the estimated optimal individualized rule:

$$\Psi_{d_{n,\text{opt}}(V)}(P_{U,X}) := \mathbb{E}_{P_{U,X}}[Y_{d_{n,\text{opt}}(V)}].$$

8.4.1 Identification and Statistical Estimand

The optimal individualized rule, as well as the value of an optimal individualized rule, are causal parameters based on the unobserved counterfactuals. In order for the causal quantities to be estimated from the observed data, they need to be identified with statistical parameters. This step of the roadmap requires we make a few assumptions:

1. *Strong ignorability*: $A \perp\!\!\!\perp Y^{d_{n,\text{opt}}(v)} \mid W$, for all $a \in \mathcal{A}$.
2. *Positivity (or overlap)*: $P_0(\min_{a\in\mathcal{A}} g_0(a \mid W) > 0) = 1$

Under the above assumptions, we can identify the causal target parameter with observed data using the G-computation formula. The value of an individualized rule can now be expressed as

$$\mathbb{E}_0[Y_{d_{n,\text{opt}}(V)}] = \mathbb{E}_{0,W}[\bar{Q}_{Y,0}(A = d_{n,\text{opt}}(V), W)],$$

which, under assumptions, is interpreted as the mean outcome if (possibly contrary to fact), treatment was assigned according to the optimal rule. Finally, the statistical counterpart to the causal parameter of interest is defined as

$$\psi_0 = \mathbb{E}_{0,W}[\bar{Q}_{Y,0}(A = d_{n,\text{opt}}(V), W)].$$

Inference for the optimal value has been shown to be difficult at exceptional laws, defined as probability distributions for which there is a positive probability on a set of W values for which conditional expectation of Y given A and W is constant in a - so all treatments are equally benefitial. Inference is similarly difficult in finite samples if the treatment effect is very small in all strata, even though valid asymptotic estimators exist in this setting. With that in mind, we address the estimation problem under the assumption of non-exceptional laws in effect.

Many methods for learning the optimal rule from data have been developed (Murphy, 2003; Robins, 2004; Zhang et al., 2016; Zhao et al., 2012; Chakraborty and Moodie, 2013). In this chapter, we focus on the methods discussed in Luedtke and van der Laan (2016a) and van der Laan and Luedtke (2015). Note however, that `tmle3mopttx` also supports the widely used Q-learning approach, where the optimal individualized rule is based on the initial estimate of $\bar{Q}_{Y,0}(A, W)$ (Sutton et al., 1998).

We follow the methodology outlined in Luedtke and van der Laan (2016a) and van der Laan and Luedtke (2015), where we learn the optimal ITR using Super Learner (van der Laan et al., 2007), and estimate its value with cross-validated Targeted Minimum Loss-based Estimation (CV-TMLE) (Zheng and van der Laan, 2010). In great generality, we first need to estimate the true individual treatment regime, $d_0(V)$, which corresponds to dynamic treatment rule that takes a subset of covariates V and assigns treatment to each individual based on their observed covariates v. With the estimate of the true optimal ITR in hand, we can estimate its corresponding value.

8.4.2 Binary treatment

How do we estimate the optimal individualized treatment regime? In the case of a binary treatment, a key quantity for optimal ITR is the **blip function**. One can show that any optimal ITR assigns treatment to individuals falling in strata in which the stratum specific average treatment effect, the blip, is positive and does not assign treatment to individuals for which this quantity is negative. Therefore for a binary treatment, under causal assumptions, we define the blip function as:

$$\bar{Q}_0(V) \equiv \mathbb{E}_0[Y_1 - Y_0 \mid V] \equiv \mathbb{E}_0[\bar{Q}_{Y,0}(1, W) - \bar{Q}_{Y,0}(0, W) \mid V],$$

or the average treatment effect within a stratum of V. The note that the optimal individualized rule can now be derived as $d_{n,\text{opt}}(V) = \mathbb{I}(\bar{Q}_n(V) > 0)$.

The package `tmle3mopttx` relies on using the Super Learner to estimate the blip function.

With that in mind, the loss function utilized for learning the optimal individualized rule corresponds to conditional mean type losses. It is however worth mentioning that Luedtke and van der Laan (2016a) present three different approaches for learning the optimal rule. Namely, they focus on:

1. Super Learner of the blip function using the squared error loss,
2. Super Learner of d_0 using the weighted classification loss function,
3. Super Learner of d_0 that uses a library of candidate estimators that are implied by estimators of the blip as well as estimators that directly go for d_0 through weighted classification.

A benefit of relying on the blip function, as implemented in `tmle3mopttx`, is that one can look at the distribution of the predicted outcomes of the blip for a given sample. Having an estimate of the blip allows one to identify patients in the sample who benefit the most (or the least) from treatment. Additionally, blip-based approach allows for straight-forward extension to the categorical treatment, interpretable rules, and OIT under resource constrains, where only a percent of the population can receive treatment (Luedtke and van der Laan, 2016b).

Relying on the Targeted Maximum Likelihood (TML) estimator and the Super Learner estimate of the blip function, we follow the below steps in order to obtain value of the ITR:

1. Estimate $\bar{Q}_{Y,0}(A,W)$ and $g_0(A \mid W)$ using `sl3`. We denote such estimates as $\bar{Q}_{Y,n}(A,W)$ and $g_n(A \mid W)$.
2. Apply the doubly robust Augmented-Inverse Probability Weighted (A-IPW) transform to our outcome (double-robust pseudo-outcome), where we define:
$$D_{\bar{Q}_Y,g,a}(O) \equiv \frac{\mathbb{I}(A=a)}{g(A \mid W)}(Y - \bar{Q}_Y(A,W)) + \bar{Q}_Y(A=a,W).$$

Note that under the randomization and positivity assumptions we have that $\mathbb{E}[D_{\bar{Q}_Y,g,a}(O) \mid V] = \mathbb{E}[Y_a \mid V]$. We emphasize the double robust nature of the A-IPW transform — consistency of $\mathbb{E}[Y_a \mid V]$ will depend on correct estimation of either $\bar{Q}_{Y,0}(A,W)$ or $g_0(A \mid W)$. As such, in a randomized trial, we are guaranteed a consistent estimate of $\mathbb{E}[Y_a \mid V]$ even if we get $\bar{Q}_{Y,0}(A,W)$ wrong! An alternative to the double-robust pseudo-outcome just presented would be single stage Q-learning, where an estimate $\bar{Q}_{Y,0}(A,W)$ is used to predict at $\bar{Q}_{Y,n}(A=1,W)$ and $\bar{Q}_{Y,n}(A=0,W)$. This provides an estimate of the blip function, $\bar{Q}_{Y,n}(A=1,W)-\bar{Q}_{Y,n}(A=0,W)$, but relies on doing a good job on estimating $\bar{Q}_{Y,0}(A,W)$.

Using the double-robust pseudo-outcome, we can define the following contrast:

$$D_{\bar{Q}_Y,g}(O) = D_{\bar{Q}_Y,g,a=1}(O) - D_{\bar{Q}_Y,g,a=0}(O).$$

We estimate the blip function, $\bar{Q}_{0,a}(V)$, by regressing $D_{\bar{Q}_Y,g}(O)$ on V using the specified `sl3` library of learners and an appropriate loss function. Finally, we are ready for the final steps.

3. Our estimated rule corresponds to $\text{argmax}_{a\in\mathcal{A}}\bar{Q}_{0,a}(V)$.
4. We obtain inference for the mean outcome under the estimated optimal rule using CV-TMLE.

8.4.3 Categorical treatment

In line with the approach considered for binary treatment, we extend the blip function to allow for categorical treatment. We denote such blip function extensions as *pseudo-blips*, which are our new estimation targets in a categorical setting. We define pseudo-blips as vector-valued entities where the output for a given V is a vector of length equal to the number of treatment categories, n_A. As such, we define it as:

$$\bar{Q}_0^{pblip}(V) = \{\bar{Q}_{0,a}^{pblip}(V) : a \in \mathcal{A}\}$$

We implement three different pseudo-blips in `tmle3mopttx`.

1. *Blip1* corresponds to choosing a reference category of treatment, and defining the blip for all other categories relative to the specified reference. Hence we have that:

   $$\bar{Q}_{0,a}^{pblip-ref}(V) \equiv \mathbb{E}_0[Y_a - Y_0 \mid V]$$

   where Y_0 is the specified reference category with $A=0$. Note that, for the case of binary treatment, this strategy reduces to the approach described for the binary setup.

2. *Blip2* approach corresponds to defining the blip relative to the average of all categories. As such, we can define $\bar{Q}_{0,a}^{pblip-avg}(V)$ as:

   $$\bar{Q}_{0,a}^{pblip-avg}(V) \equiv \mathbb{E}_0[Y_a - \frac{1}{n_A}\sum_{a\in\mathcal{A}} Y_a \mid V].$$

   In the case where subject-matter knowledge regarding which reference category to use is not available, blip2 might be a viable option.

3. *Blip3* reflects an extension of Blip2, where the average is now a weighted average:

$$\bar{Q}_{0,a}^{pblip-wavg}(V) \equiv \mathbb{E}_0[Y_a - \frac{1}{n_A}\sum_{a\in\mathcal{A}} Y_a P(A=a \mid V) \mid V].$$

Just like in the binary case, pseudo-blips are estimated by regressing contrasts composed using the A-IPW transform on V.

8.4.4 Technical Note: Inference and data-adaptive parameter

In a randomized trial, statistical inference relies on the second-order difference between the estimate of the optimal individualized treatment and the optimal individualized treatment itself to be asymptotically negligible. This is a reasonable condition if we consider rules that depend on a small number of covariates, or if we are willing to make smoothness assumptions. Alternatively, we can consider TMLEs and statistical inference for data-adaptive target parameters defined in terms of an estimate of the optimal individualized treatment. In particular, instead of trying to estimate the mean under the true optimal individualized treatment, we aim to estimate the mean under the estimated optimal individualized treatment. As such, we develop cross-validated TMLE approach that provides asymptotic inference under minimal conditions for the mean under the estimate of the optimal individualized treatment. In particular, considering the data adaptive parameter allows us to avoid consistency and rate condition for the fitted optimal rule, as required for asymptotic linearity of the TMLE of the mean under the actual, true optimal rule. Practically, the estimated (data-adaptive) rule should be preferred, as this possibly sub-optimal rule is the one implemented in the population.

8.4.5 Technical Note: Why CV-TMLE?

As discussed in van der Laan and Luedtke (2015), CV-TMLE is necessary as the non-cross-validated TMLE is biased upward for the mean outcome under the rule, and therefore overly optimistic. More generally however, using CV-TMLE allows us more freedom in estimation and therefore greater data adaptivity, without sacrificing inference.

8.5 Interpreting the Causal Effect of an Optimal Individualized Intervention

In summary, the mean outcome under the optimal individualized treatment is a counterfactual quantity of interest representing what the mean outcome would have been if everybody, contrary to the fact, received treatment that optimized their outcome. The optimal individualized treatment regime is a rule that optimizes the mean outcome under the dynamic treatment, where the candidate rules are restricted to only respond to a user-supplied subset of the baseline covariates. In essence, our target parameter answers the key aim of precision medicine: allocating the available treatment by tailoring it to the individual characteristics of the patient, with the goal of optimizing the final outcome.

8.6 Evaluating the Causal Effect of an OIT with Binary Treatment

Finally, we demonstrate how to evaluate the mean outcome under the optimal individualized treatment using tmle3mopptx. To start, let's load the packages we'll use and set a seed:

```
library(data.table)
library(sl3)
library(tmle3)
library(tmle3mopttx)
library(devtools)

set.seed(111)
```

8.6.1 Simulated Data

First, we load the simulated data. We will start with the more general setup where the treatment is a binary variable; later in the chapter we will consider another data-generating distribution where A is categorical. In this example, our data generating distribution is of the following form:

$$W \sim \mathcal{N}(\mathbf{0}, \mathbf{I_{3\times 3}})$$

$$\mathbb{P}(A = 1 \mid W) = \frac{1}{1 + \exp^{(-0.8 * W_1)}}$$

$$\mathbb{P}(Y = 1 \mid A, W) = 0.5\text{logit}^{-1}[-5I(A = 1)(W_1 - 0.5) + 5I(A = 0)(W_1 - 0.5)] + 0.5\text{logit}^{-1}(W_2 W_3)$$

```
data("data_bin")
```

The above composes our observed data structure $O = (W, A, Y)$. Note that the truth is $\psi = 0.578$ for this data generating distribution.

To formally express this fact using the tlverse grammar introduced by the tmle3 package, we create a single data object and specify the functional relationships between the nodes in the *directed acyclic graph* (DAG) via *structural equation models* (SEMs), reflected in the node list that we set up:

```
# organize data and nodes for tmle3
data <- data_bin
node_list <- list(
  W = c("W1", "W2", "W3"),
  A = "A",
  Y = "Y"
)
```

We now have an observed data structure (data) and a specification of the role that each variable in the dataset plays as the nodes in a DAG.

8.6.2 Constructing Optimal Stacked Regressions with s13

To easily incorporate ensemble machine learning into the estimation procedure, we rely on the facilities provided in the s13 R package. Using the framework provided by the s13 package, the nuisance parameters of the TML estimator may be fit with ensemble learning, using the cross-validation framework of the Super Learner algorithm of van der Laan et al. (2007).

```
# Define sl3 library and metalearners:
lrn_xgboost_50 <- Lrnr_xgboost$new(nrounds = 50)
lrn_xgboost_100 <- Lrnr_xgboost$new(nrounds = 100)
lrn_xgboost_500 <- Lrnr_xgboost$new(nrounds = 500)

lrn_mean <- Lrnr_mean$new()
lrn_glm <- Lrnr_glm_fast$new()
lrn_lasso <- Lrnr_glmnet$new()

## Define the Q learner:
Q_learner <- Lrnr_sl$new(
  learners = list(lrn_lasso, lrn_mean, lrn_glm),
  metalearner = Lrnr_nnls$new()
)

## Define the g learner:
g_learner <- Lrnr_sl$new(
  learners = list(lrn_lasso, lrn_glm),
  metalearner = Lrnr_nnls$new()
)
```

```
## Define the B learner:
b_learner <- Lrnr_sl$new(
  learners = list(lrn_lasso,lrn_mean, lrn_glm),
  metalearner = Lrnr_nnls$new()
)
```

As seen above, we generate three different ensemble learners that must be fit, corresponding to the learners for the outcome regression (Q), propensity score (g), and the blip function (B). We make the above explicit with respect to standard notation by bundling the ensemble learners into a list object below:

```
# specify outcome and treatment regressions and create learner list
learner_list <- list(Y = Q_learner, A = g_learner, B = b_learner)
```

The `learner_list` object above specifies the role that each of the ensemble learners we've generated is to play in computing initial estimators. Recall that we need initial estimators of relevant parts of the likelihood in order to build a TMLE for the parameter of interest. In particular, `learner_list` makes explicit the fact that our `Y` is used in fitting the outcome regression, while `A` is used in fitting the treatment mechanism regression, and finally `B` is used in fitting the blip function.

8.6.3 Targeted Estimation of the Mean under the Optimal Individualized Interventions Effects

To start, we will initialize a specification for the TMLE of our parameter of interest simply by calling `tmle3_mopttx_blip_revere`. We specify the argument `V = c("W1", "W2", "W3")` when initializing the `tmle3_Spec` object in order to communicate that we're interested in learning a rule dependent on `V` covariates. Note that we don't have to specify `V` — this will result in a rule that is not based on any collected covariates; we will see an example like this shortly. We also need to specify the type of (pseudo) blip we will use in this estimation problem, the list of learners used to estimate the blip function, whether we want to maximize or minimize the final outcome, and few other more advanced features including searching for a less complex rule, realistic interventions and possible resource constraints.

```
# initialize a tmle specification
tmle_spec <- tmle3_mopttx_blip_revere(
  V = c("W1", "W2", "W3"), type = "blip1",
  learners = learner_list,
  maximize = TRUE, complex = TRUE,
  realistic = FALSE, resource = 1
)
```

As seen above, the `tmle3_mopttx_blip_revere` specification object (like all `tmle3_Spec` objects) does *not* store the data for our specific analysis of interest. Later, we'll see that passing a data object directly to the `tmle3` wrapper function, alongside the instantiated `tmle_spec`, will serve to construct a `tmle3_Task` object internally.

We elaborate more on the initialization specifications. In initializing the specification for the TMLE of our parameter of interest, we have specified the set of covariates the rule depends on (`V`), the type of (pseudo) blip to use (`type`), and the learners used for estimating the relevant parts of the likelihood and the blip function. In addition, we need to specify whether we want to maximize the mean outcome under the rule (`maximize`), and whether we want to estimate the rule under all the covariates V provided by the user (`complex`). If `FALSE`, `tmle3mopttx` will instead consider all the possible rules under a smaller set of covariates including the static rules, and optimize the mean outcome over all the subsets of V. As such, while the user might have provided a full set of collected covariates as input for V, it is possible that the true rule only depends on a subset of the set provided by the user. In that case, our returned mean under the optimal individualized rule will be based on the smaller subset. In addition, we provide an option to search for realistic optimal individualized interventions via the `realistic` specification. If `TRUE`, only treatments supported by the data will be considered, therefore alleviating concerns regarding practical positivity issues. Finally, we can incorporate source constrains by setting `resource` argument to less than 1. We explore all the important extensions of `tmle3mopttx` in later sections.

```
# fit the TML estimator
fit <- tmle3(tmle_spec, data, node_list, learner_list)
fit
A tmle3_Fit that took 1 step(s)
   type         param init_est tmle_est      se   lower   upper psi_transformed
1:  TSM E[Y_{A=NULL}]  0.35553  0.55371 0.02598 0.50279 0.60463         0.55371
   lower_transformed upper_transformed
1:           0.50279           0.60463
```

By studying the output generated, we can see that the confidence interval covers the true parameter, as expected.

8.6.3.1 Resource constraint

We can restrict the number of individuals that get the treatment by only treating k percent of samples. With that, only patients with the biggest benefit (according to the estimated blip) receive treatment. In order to impose a resource constraint, we only have to specify the percent of individuals that can get treatment. For example, if `resource=1`, all individuals with blip higher than zero will get treatment; if `resource=0`, noone will be treated.

```
# initialize a tmle specification
tmle_spec_resource <- tmle3_mopttx_blip_revere(
  V = c("W1", "W2", "W3"), type = "blip1",
  learners = learner_list,
  maximize = TRUE, complex = TRUE,
  realistic = FALSE, resource = 0.90
)

# fit the TML estimator
fit_resource <- tmle3(tmle_spec_resource, data, node_list, learner_list)
fit_resource
A tmle3_Fit that took 1 step(s)
   type         param init_est tmle_est      se   lower   upper psi_transformed
1:  TSM E[Y_{A=NULL}]  0.34304  0.55841 0.02612 0.50721 0.6096         0.55841
   lower_transformed upper_transformed
1:           0.50721            0.6096
```

We can compare the number of individuals that got treatment with and without the resource constraint:

```
# Number of individuals getting treatment (no resource constraint):
table(tmle_spec$return_rule)

  0   1
275 725

# Number of individuals getting treatment (resource constraint):
table(tmle_spec_resource$return_rule)

  0   1
351 649
```

8.6.3.2 Empty V

Below we the show an example where V is not specified, under the resource constraint.

```
# initialize a tmle specification
tmle_spec_V_empty <- tmle3_mopttx_blip_revere(
  type = "blip1",
  learners = learner_list,
  maximize = TRUE, complex = TRUE,
  realistic = FALSE, resource = 0.90
)

# fit the TML estimator
fit_V_empty <- tmle3(tmle_spec_V_empty, data, node_list, learner_list)
fit_V_empty
A tmle3_Fit that took 1 step(s)
   type         param init_est tmle_est       se   lower   upper
1:  TSM E[Y_{A=NULL}]  0.31575  0.51694 0.013528 0.49043 0.54346
   psi_transformed lower_transformed upper_transformed
1:         0.51694           0.49043           0.54346
```

8.7 Evaluating the Causal Effect of an optimal ITR with Categorical Treatment

In this section, we consider how to evaluate the mean outcome under the optimal individualized treatment when A has more than two categories. While the procedure is analogous to the previously described binary treatment, we now need to pay attention to the type of blip we define in the estimation stage, as well as how we construct our learners.

8.7.1 Simulated Data

First, we load the simulated data. Our data generating distribution is of the following form:

$$W \sim \mathcal{N}(\mathbf{0}, \mathbf{I_{4\times 4}})$$

$$\mathbb{P}(A = a \mid W) = \frac{1}{1 + \exp^{(-0.8*W_1)}}$$

$$\begin{aligned}\mathbb{P}(Y = 1 \mid A, W) = 0.5\text{logit}^{-1}[15I(A = 1)(W_1 - 0.5)-\\ 3I(A = 2)(2W_1 + 0.5)+\\ 3I(A = 3)(3W_1 - 0.5)] + \text{logit}^{-1}(W_2W_1)\end{aligned}$$

We can just load the data available as part of the package as follows:

```
data("data_cat_realistic")
```

The above composes our observed data structure $O = (W, A, Y)$. Note that the truth is now $\psi_0 = 0.658$, which is the quantity we aim to estimate.

```
# organize data and nodes for tmle3
data <- data_cat_realistic
node_list <- list(
  W = c("W1", "W2", "W3", "W4"),
  A = "A",
  Y = "Y"
)
```

We can see the number of observed categories of treatment below:

```
# organize data and nodes for tmle3
table(data$A)

  1   2   3
 24 528 448
```

8.7.2 Constructing Optimal Stacked Regressions with `sl3`

QUESTION: With categorical treatment, what is the dimension of the blip now? What is the dimension for the current example? How would we go about estimating it?

We will now create new ensemble learners using the `sl3` learners initialized previously:

```
# Initialize few of the learners:
lrn_xgboost_50 <- Lrnr_xgboost$new(nrounds = 50)
lrn_xgboost_100 <- Lrnr_xgboost$new(nrounds = 100)
lrn_xgboost_500 <- Lrnr_xgboost$new(nrounds = 500)
lrn_mean <- Lrnr_mean$new()
lrn_glm <- Lrnr_glm_fast$new()

## Define the Q learner, which is just a regular learner:
Q_learner <- Lrnr_sl$new(
  learners = list(lrn_xgboost_100, lrn_mean, lrn_glm),
  metalearner = Lrnr_nnls$new()
)

# Define the g learner, which is a multinomial learner:
# specify the appropriate loss of the multinomial learner:
mn_metalearner <- make_learner(Lrnr_solnp,
  loss_function = loss_loglik_multinomial,
  learner_function = metalearner_linear_multinomial
)
g_learner <- make_learner(Lrnr_sl, list(lrn_xgboost_100, lrn_xgboost_500, lrn_mean),

# Define the Blip learner, which is a multivariate learner:
```

```
learners <- list(lrn_xgboost_50, lrn_xgboost_100, lrn_xgboost_500, lrn_mean, lrn_glm)
b_learner <- create_mv_learners(learners = learners)
```

As seen above, we generate three different ensemble learners that must be fit, corresponding to the learners for the outcome regression, propensity score, and the blip function. Note that we need to estimate $g_0(A \mid W)$ for a categorical A — therefore, we use the multinomial Super Learner option available within the `sl3` package with learners that can address multi-class classification problems. In order to see which learners can be used to estimate $g_0(A \mid W)$ in `sl3`, we run the following:

```
# See which learners support multi-class classification:
sl3_list_learners(c("categorical"))
 [1] "Lrnr_bound"                  "Lrnr_caret"
 [3] "Lrnr_cv_selector"            "Lrnr_glmnet"
 [5] "Lrnr_grf"                    "Lrnr_gru_keras"
 [7] "Lrnr_h2o_glm"                "Lrnr_h2o_grid"
 [9] "Lrnr_independent_binomial"   "Lrnr_lightgbm"
[11] "Lrnr_lstm_keras"             "Lrnr_mean"
[13] "Lrnr_multivariate"           "Lrnr_nnet"
[15] "Lrnr_optim"                  "Lrnr_polspline"
[17] "Lrnr_pooled_hazards"         "Lrnr_randomForest"
[19] "Lrnr_ranger"                 "Lrnr_rpart"
[21] "Lrnr_screener_correlation"   "Lrnr_solnp"
[23] "Lrnr_svm"                    "Lrnr_xgboost"
```

Since the corresponding blip will be vector valued, we will have a column for each additional level of treatment. As such, we need to create multivariate learners with the helper function `create_mv_learners` that takes a list of initialized learners as input.

We make the above explicit with respect to the standard notation by bundling the ensemble learners into a list object below:

```
# specify outcome and treatment regressions and create learner list
learner_list <- list(Y = Q_learner, A = g_learner, B = b_learner)
```

8.7.3 Targeted Estimation of the Mean under the Optimal Individualized Interventions Effects

```
# initialize a tmle specification
tmle_spec_cat <- tmle3_mopttx_blip_revere(
  V = c("W1", "W2", "W3", "W4"), type = "blip2",
  learners = learner_list, maximize = TRUE, complex = TRUE,
  realistic = FALSE
)
```

```
# fit the TML estimator
fit_cat <- tmle3(tmle_spec_cat, data, node_list, learner_list)
fit_cat
A tmle3_Fit that took 1 step(s)
   type         param init_est tmle_est       se   lower   upper
1:  TSM E[Y_{A=NULL}]  0.53783  0.62117 0.065863 0.49208 0.75025
   psi_transformed lower_transformed upper_transformed
1:         0.62117           0.49208           0.75025

# How many individuals got assigned each treatment?
table(tmle_spec_cat$return_rule)

  1   2   3
250 432 318
```

We can see that the confidence interval covers the truth.

NOTICE the distribution of the assigned treatment! We will need this shortly.

8.8 Extensions to Causal Effect of an OIT

In this section, we consider two extensions to the procedure described for estimating the value of the OIT. First one considers a setting where the user might be interested in a grid of possible sub-optimal rules, corresponding to potentially limited knowledge of potential effect modifiers. The second extension concerns implementation of a realistic optimal individual interventions where certain regimes might be preferred, but due to practical or global positivity restraints, are not realistic to implement.

8.8.1 Simpler Rules

In order to not only consider the most ambitious fully V-optimal rule, we define S-optimal rules as the optimal rule that considers all possible subsets of V covariates, with card(S) $\leq$ card(V) and $\emptyset \in S$. In particular, this allows us to define a Super Learner for d_0 that includes a range of estimators from very simple (e.g., statis rules) to more complex (e.g. full V), and let the discrete Super Learner select a simple rule when appropriate. This allows us to consider sub-optimal rules that are easier to estimate and potentially provide more realistic rules. Within the `tmle3mopttx` paradigm, we just need to change the `complex` parameter to `FALSE`:

```
# initialize a tmle specification
tmle_spec_cat_simple <- tmle3_mopttx_blip_revere(
  V = c("W4", "W3", "W2", "W1"), type = "blip2",
  learners = learner_list,
  maximize = TRUE, complex = FALSE, realistic = FALSE
)

# fit the TML estimator
fit_cat_simple <- tmle3(tmle_spec_cat_simple, data, node_list, learner_list)
fit_cat_simple
A tmle3_Fit that took 1 step(s)
   type           param init_est tmle_est       se   lower   upper
1:  TSM E[Y_{d(V=W1)}]  0.54336  0.61838 0.060848 0.49912 0.73764
   psi_transformed lower_transformed upper_transformed
1:         0.61838           0.49912           0.73764
```

Even though we specified all baseline covariates as the basis for rule estimation, a simpler rule is sufficient to maximize the mean outcome.

QUESTION: How does the set of covariates picked by `tmle3mopttx` compare to the baseline covariates the true rule depends on?

8.8.2 Realistic Optimal Individual Regimes

In addition to considering less complex rules, `tmle3mopttx` also provides an option to estimate the mean under the realistic, or implementable, optimal individualized treatment. It is often the case that assigning particular regime might have the ability to fully maximize (or minimize) the desired outcome, but due to global or practical positivity constrains, such treatment can never be implemented in real life (or is highly unlikely). As such, specifying `realistic` to `TRUE`, we consider possibly suboptimal treatments that optimize the outcome in question while being supported by the data.

```
# initialize a tmle specification
tmle_spec_cat_realistic <- tmle3_mopttx_blip_revere(
  V = c("W4", "W3", "W2", "W1"), type = "blip2",
  learners = learner_list,
  maximize = TRUE, complex = TRUE, realistic = TRUE
)

# fit the TML estimator
fit_cat_realistic <- tmle3(tmle_spec_cat_realistic, data, node_list, learner_list)
fit_cat_realistic
A tmle3_Fit that took 1 step(s)
   type          param init_est tmle_est      se   lower   upper psi_transformed
```

```
1:  TSM E[Y_{A=NULL}]  0.54035  0.65821 0.02135 0.61636 0.70005         0.65821
   lower_transformed upper_transformed
1:           0.61636           0.70005

# How many individuals got assigned each treatment?
table(tmle_spec_cat_realistic$return_rule)

  2   3
506 494
```

QUESTION: Referring back to the data-generating distribution, why do you think the distribution of allocated treatment changed from the distribution we had under the “non-realistic”” rule?

8.8.3 Missingness and `tmle3mopttx`

In this section, we present how to use the `tmle3mopttx` package when the data is subject to missingness in Y. Let’s start by add some missingness to our outcome, first.

```
data_missing <- data_cat_realistic

#Add some random missingless:
rr <- sample(nrow(data_missing), 100, replace = FALSE)
data_missing[rr,"Y"]<-NA

summary(data_missing$Y)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  0.000   0.000   0.000   0.464   1.000   1.000     100
```

To start, we must first add to our library — we now also need to estimate the missingness process as well.

```
delta_learner <- Lrnr_sl$new(
  learners = list(lrn_mean, lrn_glm),
  metalearner = Lrnr_nnls$new()
)

# specify outcome and treatment regressions and create learner list
learner_list <- list(Y = Q_learner, A = g_learner, B = b_learner, delta_Y=delta_learn
```

The `learner_list` object above specifies the role that each of the ensemble learners we’ve generated is to play in computing the initial estimators needed for building the TMLE for the parameter of interest. In particular, it makes explicit the fact that `Y` is used in fitting the outcome regression while A is used in fitting our treatment mechanism regression, B for fitting the blip function, and delta_Y fits the missing outcome process.

Now, with the additional estimation step associated with missingness added, we can proceed as usual.

```
# initialize a tmle specification
tmle_spec_cat_miss <- tmle3_mopttx_blip_revere(
  V = c("W1", "W2", "W3", "W4"), type = "blip2",
  learners = learner_list, maximize = TRUE, complex = TRUE,
  realistic = FALSE
)

# fit the TML estimator
fit_cat_miss <- tmle3(tmle_spec_cat_miss, data_missing, node_list, learner_list)
fit_cat_miss
A tmle3_Fit that took 1 step(s)
   type                        param init_est tmle_est       se   lower   upper
1:  TSM E[Y_{A=NULL, delta_Y=1}]  0.53537    0.727 0.061309 0.60683 0.84716
   psi_transformed lower_transformed upper_transformed
1:           0.727           0.60683           0.84716
```

8.8.4 Q-learning

Alternatively, we could estimate the mean under the optimal individualized treatment using Q-learning. The optimal rule can be learned through fitting the likelihood, and consequently estimating the optimal rule under this fit of the likelihood (Sutton et al., 1998; Murphy, 2003).

Below we outline how to use tmle3mopttx package in order to estimate the mean under the ITR using Q-learning. As demonstrated in the previous sections, we first need to initialize a specification for the TMLE of our parameter of interest. As opposed to the previous section however, we will now use tmle3_mopttx_Q instead of tmle3_mopttx_blip_revere in order to indicate that we want to use Q-learning instead of TMLE.

```
# initialize a tmle specification
tmle_spec_Q <- tmle3_mopttx_Q(maximize = TRUE)

# Define data:
tmle_task <- tmle_spec_Q$make_tmle_task(data, node_list)

# Define likelihood:
initial_likelihood <- tmle_spec_Q$make_initial_likelihood(
  tmle_task,
  learner_list
)

# Estimate the parameter:
Q_learning(tmle_spec_Q, initial_likelihood, tmle_task)[1]
```

8.9 Variable Importance Analysis with OIT

Suppose one wishes to assess the importance of each observed covariate, in terms of maximizing (or minimizing) the population mean of an outcome under an optimal individualized treatment regime. In particular, a covariate that maximizes (or minimizes) the population mean outcome the most under an optimal individualized treatment out of all other considered covariates under optimal assignment might be considered *more important* for the outcome. To put it in context, perhaps optimal allocation of treatment 1, denoted A_1, results in a larger mean outcome than optimal allocation of another treatment 2, denoted A_2. Therefore, we would label A_1 as having a higher variable importance with regard to maximizing (or minimizing) the mean outcome under the optimal individualized treatment.

8.9.1 Simulated Data

For illustration purpose, we bin baseline covariates corresponding to the data-generating distribution described previously:

```
# bin baseline covariates to 3 categories:
data$W1<-ifelse(data$W1<quantile(data$W1)[2],1,ifelse(data$W1<quantile(data$W1)[3],2,

node_list <- list(
  W = c("W3", "W4", "W2"),
  A = c("W1", "A"),
  Y = "Y"
)
```

Our node list now includes W_1 as treatments as well! Don't worry, we will still properly adjust for all baseline covariates.

8.9.2 Variable Importance using Targeted Estimation of the value of the ITR

In the previous sections we have seen how to obtain a contrast between the mean under the optimal individualized rule and the mean under the observed outcome for a single covariate — we are now ready to run the variable importance analysis for all of our specified covariates. In order to run the variable importance analysis, we first need to initialize a specification for the TMLE of our parameter of interest as we have done before. In addition, we need to specify the data and the corresponding list of nodes, as well as the appropriate learners for the outcome regression, propensity score, and the blip function. Finally, we need to specify whether we should adjust for all the other covariates we are assessing variable importance for. We will adjust for all Ws in our analysis, and if `adjust_for_other_A=TRUE`, also for all A covariates that are not treated as exposure in the variable importance loop.

To start, we will initialize a specification for the TMLE of our parameter of interest (called a `tmle3_Spec` in the `tlverse` nomenclature) simply by calling `tmle3_mopttx_vim`. First, we indicate the method used for learning the optimal individualized treatment by specifying the `method` argument of `tmle3_mopttx_vim`. If `method="Q"`, then we will be using Q-learning for rule estimation, and we do not need to specify `V`, `type` and `learners` arguments in the spec, since they are not important for Q-learning. However, if `method="SL"`, which corresponds to learning the optimal individualized treatment using the above outlined methodology, then we need to specify the type of (pseudo) blip we will use in this estimation problem, whether we want to maximize or minimize the outcome, complex and realistic rules, resource constraint. Finally, for `method="SL"` we also need to communicate that we're interested in learning a rule dependent on `V` covariates by specifying the `V` argument. For both `method="Q"` and `method="SL"`, we need to indicate whether we want to maximize or minimize the mean under the optimal individualized rule. Finally, we also need to specify whether the final comparison of the mean under the optimal individualized rule and the mean under the observed outcome should be on the multiplicative scale (risk ratio) or linear (similar to average treatment effect).

```
# initialize a tmle specification
tmle_spec_vim <- tmle3_mopttx_vim(
  V=c("W2"),
  type = "blip2",
  learners = learner_list,
  maximize = FALSE,
  method = "SL",
  complex = TRUE,
  realistic = FALSE
)
```

```
# fit the TML estimator
vim_results <- tmle3_vim(tmle_spec_vim, data, node_list, learner_list,
  adjust_for_other_A = TRUE
)

print(vim_results)
   type                param    init_est  tmle_est       se      lower
1:  ATE E[Y_{A=NULL}] - E[Y]  0.00066272  0.040712 0.016866  0.0076547
2:  ATE E[Y_{A=NULL}] - E[Y] -0.01042913 -0.039754 0.022095 -0.0830588
       upper psi_transformed lower_transformed upper_transformed  A           W
1: 0.0737684        0.040712         0.0076547         0.0737684  A W3,W4,W2,W1
2: 0.0035505       -0.039754        -0.0830588         0.0035505 W1  W3,W4,W2,A
    Z_stat      p_nz p_nz_corrected
1:  2.4138 0.0078932       0.015786
2: -1.7993 0.0359882       0.035988
```

The final result of `tmle3_vim` with the `tmle3mopttx` spec is an ordered list of mean outcomes under the optimal individualized treatment for all categorical covariates in our dataset.

8.10 Exercises

8.10.1 Real World Data and `tmle3mopttx`

Finally, we cement everything we learned so far with a real data application.

As in the previous sections, we will be using the WASH Benefits data, corresponding to the effect of water quality, sanitation, hand washing, and nutritional interventions on child development in rural Bangladesh.

The main aim of the cluster-randomized controlled trial was to assess the impact of six intervention groups, including:

1. control;
2. hand-washing with soap;

3. improved nutrition through counseling and provision of lipid-based nutrient supplements;
4. combined water, sanitation, hand-washing, and nutrition;
5. improved sanitation;
6. combined water, sanitation, and hand-washing;
7. chlorinated drinking water.

We aim to estimate the optimal ITR and the corresponding value under the optimal ITR for the main intervention in WASH Benefits data.

Our outcome of interest is the weight-for-height Z-score, whereas our primary treatment is the six intervention groups aimed at improving living conditions.

Questions:

1. Define V as mother's education (`momedu`), current living conditions (`floor`), and possession of material items including the refrigerator (`asset_refrig`). Why do you think we use these covariates as V? Do we want to minimize or maximize the outcome? Which (pseudo) blip type should we use?
2. Load the WASH Benefits data, and define the appropriate nodes for treatment and outcome. Use all the rest of the covariates as W except for `momheight` for now. Construct an appropriate `sl3` library for A, Y and B.
3. Based on the V defined in the previous question, estimate the mean under the ITR for the main randomized intervention used in the WASH Benefits trial with weight-for-height Z-score as the outcome. What's the TMLE value of the optimal ITR? How does it change from the initial estimate? Which intervention is the most prominent? Why do you think that is?
4. Using the same formulation as in questions 1 and 2, estimate the realistic optimal ITR and the corresponding value of the realistic ITR. Did the results change? Which intervention is the most prominent under realistic rules? Why do you think that is?
5. Consider simpler rules for the WASH benefits data example. Which covariates does the final rule depend on?

6. Change the treatment to a binary variable (`asset_sewmach`), and estimate the value under the ITR in this setting under a 60% resource constraint. What do the results indicate?

7. Change the treatment once again, now to mother's education (`momedu`), and estimate the value under the ITR in this setting. What do the results indicate? Can we intervene on such a variable?

8.10.2 Review of Key Concepts

1. What is the difference between dynamic and optimal individualized regimes?

2. What's the intuition behind using different blip types? Why did we switch from `blip1` to `blip2` when considering categorical treatment? What are some of the advantages of each?

3. Look back at the results generated in the section on categorical treatments, and compare them to the mean under the optimal individualized treatment in the section on complex categorical treatments. How does the set of covariates picked by `tmle3mopttx` compare to the baseline covariates the true rule depends on?

4. Compare the distribution of treatments assigned under the true optimal individualized treatment and realistic optimal individualized treatment. Referring back to the data-generating distribution, why do you think the distribution of allocated treatment changed?

5. Using the same simulation, perform a variable importance analysis using Q-learning. How do the results change and why?

8.10.3 Advanced Topics

1. How can we extend the current approach to include exceptional laws?

2. How can we extend the current approach to continuous interventions?

9

Stochastic Treatment Regimes

Nima Hejazi

Based on the `tmle3shift` `R` package by *Nima Hejazi, Jeremy Coyle, and Mark van der Laan*.

Updated: 2021-12-30

Learning Objectives

1. Differentiate stochastic treatment regimes from static, dynamic, and optimal dynamic treatment regimes. 2. Describe how a real-world data analysis may incorporate assessing the causal effects of stochastic treatment regimes. 3. Contrast a population-level (general) stochastic treatment regime from an (individualized) modified treatment policy. 4. Estimate the population-level causal effects of modified treatment policies with the 'tmle3shift' 'R' package. 5. Specify and interpret a set of causal effects based upon differing modified treatment policies arising from a grid of counterfactual shifts. 6. Construct marginal structural models to measure variable importance in terms of stochastic interventions, using a grid of counterfactual shifts. 7. Implement, with the 'tmle3shift' 'R' package, modified treatment policies that shift individual units only to the extent supported by the observed data.

9.1 Why *Stochastic* Interventions?

Stochastic treatment regimes, or *stochastic interventions*, constitute a relatively simple yet extremely flexible and expressive framework for defining *realistic* causal effects. In contrast to intervention regimens discussed previously, stochastic interventions may be applied to nearly any manner of treatment variable – binary, ordinal, continuous – allowing for a rich set of causal effects to be defined through this formalism. This chapter focuses on examining a few types of stochastic interventions that may be applied to *continuous* treatment variables, to which static and dynamic treatment regimes cannot easily be applied. Notably, the resultant causal effects conveniently are endowed with an interpretation echoing that of ordinary regression adjustment.

In the next chapter, we will introduce two alternative uses of stochastic interventions – a recently formulated intervention applicable to binary treatment variables (Kennedy, 2019) and the definition of causal effects in the presence of post-treatment, or mediating, variables. Here, we will focus on the tools provided in the `tmle3shift` R package, which exposes targeted minimum loss-based estimators of the causal effects of stochastic interventions that additively shift the observed value of the treatment variable. More comprehensive, technical presentations of some aspects of the material in this chapter appear in Díaz and van der Laan (2012), Díaz and van der Laan (2018), Hejazi et al. (2020c), and Hejazi (2021).

9.2 Data Structure and Notation

Let us return to the familiar data unit $O = (W, A, Y)$, where W denote baseline covariates (e.g., age, biological sex, education level), A a treatment variable (e.g., dose of nutritional supplements), and Y an outcome of interest (e.g., disease status). Here, we consider A that are continuous-valued (i.e., $A \in \mathbb{R}$) or ordinal with many levels. For a given study, we consider observing n independent and identically distributed units $O_1, \ldots, O_n$.

Following the roadmap, let $O \sim P \in \mathcal{M}$, where $\mathcal{M}$ is the nonparametric statistical model, minimizing any restrictions on the form of the data-generating distribution P. To formalize the definition of stochastic interventions and their corresponding causal effects, we introduce a structural causal model (SCM), based on Pearl (2009), to define how the system changes under posited interventions:

$$\begin{aligned} W &= f_W(U_W) \\ A &= f_A(W, U_A) \\ Y &= f_Y(A, W, U_Y). \end{aligned} \tag{9.1}$$

The set of structural equations provide a mechanistic model describing the relationships between variables composing the observed data unit O. The SCM describes a temporal ordering between the variables (i.e., that Y occurs after A, which occurs after W); specifies deterministic functions $\{f_W, f_A, f_Y\}$ generating each variable $\{W, A, Y\}$ based on those preceding it and exogenous (unobserved) variable $\{U_W, U_A, U_Y\}$; and requires that each exogenous variable is assumed to contain all unobserved causes of the corresponding observed variable.

We can factorize the likelihood of the data unit O as follows, revealing orthogonal components of the density, p_0^O, when evaluated on a typical observation o:

$$p_0(x) = q_{0,Y}(y \mid A = a, W = w) g_{0,A}(a \mid W = w) q_{0,W}(w), \tag{9.2}$$

where $q_{0,Y}$ is the conditional density of Y given $\{A, W\}$ with respect to some dominating measure, $g_{0,A}$ is the conditional density of A given W with respect to dominating measure μ, and $q_{0,W}$ is the density of W with respect to dominating measure ν. In the interest of continuing to use familiar notation, we let $Q(A, W) = \mathbb{E}[Y \mid A, W]$, $g(A \mid W) = g_A(A \mid W)$, and q_W the marginal distribution of W. Importantly, the SCM parameterizes p_0 in terms of the distribution of random variables (O, U) modeled by the system of equations. In turn, this implies a model for the distribution of counterfactual random variables generated by interventions on the data-generating process.

9.3 Defining the Causal Effect of a Stochastic Intervention

Causal effects are defined in terms of contrasts of hypothetical interventions on the SCM (**??**). Stochastic interventions modifying components of the SCM may be thought of in two equivalent ways. A *general* stochastic intervention replaces the equation f_A, which gives rise to A, and $g(A \mid W)$, the natural conditional density of A, with a candidate density $g_{A_\delta}(A \mid W)$. In the absence of the intervention, we would consider any given value $a \in \mathcal{A}$, the support of A – that is, the result of evaluating the function f_A at a given value $W = w$ – as the result of a random draw from the distribution defined by the conditional density $g(A \mid W)$, that is, $A_\delta \sim g_{A_\delta}(\cdot \mid W)$. In applying the intervention, we simply remove the structural equation f_A, instead drawing the post-intervention value A_δ from the distribution defined by the candidate density $g_{A_\delta}(A \mid W)$. The post-intervention value A_δ is stochastically modified in the sense that it has been drawn from an arbitrary (in practice, user-defined) distribution. Note that the familiar case of static interventions can be recovered by choosing degenerate candidate distributions, which place all mass on just a single value. Stock (1989) first considered estimating the total effects of such stochastic interventions.

While there are few restrictions on the choice of the candidate post-treatment density $g_{A_\delta}(A \mid W)$, in practice, it is often chosen based on knowledge of the natural (or pre-intervention) density $g(A \mid W)$. When $g_{A_\delta}(A \mid W)$ is *piecewise smooth invertible* (more below) (Haneuse and Rotnitzky, 2013), there is a direct correspondence between the post-intervention density $g_{A_\delta}(A \mid W)$ and a function $d(A, W; \delta)$ that maps an observed pair $\{A, W\}$ to the post-intervention quantity A_δ. In such cases, the stochastic intervention, defined by $d(A, W; \delta)$, is said to depend on the natural value of treatment and has been termed a *modified treatment policy* (MTP) (Haneuse and Rotnitzky, 2013; Díaz and van der Laan, 2018; Hejazi, 2021). Haneuse and Rotnitzky (2013) and Young et al. (2014) provide detailed discussions contrasting the interpretations of the causal effects under modified treatment policies and general stochastic interventions.

Definition 9.1 (Piecewise Smooth Invertibility)**.** For each $w \in \mathcal{W}$, assume that the interval $\mathcal{I}(w) = (l(w,), u(w))$ may be partitioned into subintervals $\mathcal{I}_{\delta,j}(w) : j = 1, \ldots, J(w)$ such that $d(a, w; \delta)$ is equal to some $d_j(a, w; \delta)$ in $\mathcal{I}_{\delta,j}(w)$ and $d_j(\cdot, w; \delta)$ has inverse function $h_j(\cdot, w; \delta)$ with derivative $h_j'(\cdot, w; \delta)$.

A stochastic intervention gives rise to a counterfactual random variable $Y_{A_\delta} := f_Y(A_\delta, W, U_Y)$, where the counterfactual outcome $Y_{A_\delta} \sim \mathcal{P}_0^{A_\delta}$ arises from replacing the natural value of A with A_δ (whether as a draw from $g_{A_\delta}(A \mid W)$ or by evaluating $d(A, W; \delta)$). For the remainder of this chapter, we will focus on additive MTPs of the form

$$d(a, w; \delta) = \begin{cases} a + \delta & \text{if } a + \delta \leq u(w) \\ a & \text{if } a + \delta > u(w), \end{cases} \tag{9.3}$$

where $\delta \in \mathbb{R}^+$ defines the degree to which an observed $A = a$ ought to be shifted, in the context of the stratum $W = w$, and $l(w)$ and $u(w)$ are the minimum and maximum values of the treatment A in the stratum $W = w$. Consider, for example, the case where A denotes a (continuous-valued) dosage of nutritional supplements (e.g., number of vitamin pills) and assume that the distribution of A conditional on $W = w$ has support in the interval $(l(w), u(w))$. That is, the minimum number of pills taken for an individual with in the covariate stratum defined by $W = w$ is $l(w)$; similarly, the maximum is $u(w)$. Such a stochastic intervention may be interpreted as the result of a clinic policy encouraging individuals to consume δ more vitamin pills ($A\delta$) than they would normally be recommended (A) based on their baseline characteristics W. This class of stochastic interventions was introduced by Díaz and van der Laan (2012) and has been further discussed in Haneuse and Rotnitzky (2013), Díaz and van der Laan (2018), Hejazi et al. (2020c), and Hejazi (2021). This class of interventions may be expressed as a general stochastic intervention, as per Díaz and van der Laan (2012), by considering the random draw $\mathbb{P}_{A_\delta}(g_{0,A})(A = a \mid W) = g_{0,A}(a - \delta(W) \mid W)$.

In order to evaluate the causal effect of our intervention, we consider as a parameter of interest the counterfactual mean of the outcome under our stochastically modified intervention distribution. This target causal estimand is $\psi_{0,\delta} := \mathbb{E}_{P_0^{A_\delta}}\{Y_{A_\delta}\}$, the mean of the counterfactual outcome variable Y_{A_δ}. Díaz and van der Laan (2018) showed that $\psi_{0,\delta}$ may be identified by a functional of the distribution of O:

$$\psi_{0,\delta} = \int_{\mathcal{W}} \int_{\mathcal{A}} \mathbb{E}_{P_0}\{Y \mid A = d(a,w), W = w\} \\ q_{0,A}(a \mid W = w) q_{0,W}(w) d\mu(a) d\nu(w). \tag{9.4}$$

Under certain identification conditions, which we will enumerate shortly, the statistical parameter in Equation (9.4) matches exactly the counterfactual mean $\psi_{0,\delta}$. While this book is not concerned with the identification of causal parameters – that is, establishing statistical functionals of the observed data that have causal interpretations under certain assumptions – we review key assumptions for identifying the counterfactual mean $\psi_{0,\delta}$ below. As the SCM introduced prior generates independent and identically distributed units O, the common identification assumptions of consistency ($Y_i^{A_{\delta,i}} = Y_i$ in the event $A_i = d(a_i, w_i)$, for $i = 1, \ldots, n$) and lack of interference ($Y_i^{A_{\delta,i}}$ does not depend on $d(a_j, w_j)$ for $i = 1, \ldots, n$ and $j \neq i$) hold. Beyond these, we require no unmeasured confounding (the analog to the randomization assumption in observational studies) and positivity.

1. *No unmeasured confounders*: $A_i \perp\!\!\!\perp Y_i^{A_{\delta,i}} \mid W_i$, for $i = 1, \ldots, n$. This is the observational study analog to the well-known randomization assumption.
2. *Positivity (or overlap)*: $a_i \in \mathcal{A} \implies d(a_i, w_i) \in \mathcal{A}$ for all $w \in \mathcal{W}$, where $\mathcal{A}$ denotes the support of $A \mid W = w_i \quad \forall i = 1, \ldots n$.

9.4 Estimating the Causal Effect of a Stochastic Intervention

Díaz and van der Laan (2012) provided a derivation of the efficient influence function (EIF), a key quantity for constructing efficient estimators, in the nonparametric model $\mathcal{M}$ and developed both classical and efficient estimators of this quantity, including substitution, inverse probability weighted, one-step and targeted maximum likelihood (TML) estimators. Both the one-step and TML estimators allow for semiparametric-efficient estimation and inference on the target quantity of interest $\psi_{0,\delta}$. As described by Díaz and van der Laan (2018), the EIF of $\psi_{0,\delta}$, with respect to the nonparametric model $\mathcal{M}$, is

$$D(P_0)(x) = H(a,w)(y - \overline{Q}(a,w)) + \overline{Q}(d(a,w),w) - \Psi(P_0), \tag{9.5}$$

where the auxiliary covariate $H(a, w)$ may be expressed

$$H(a, w) = \mathbb{I}(a + \delta < u(w)) \frac{g_0(a - \delta \mid w)}{g_0(a \mid w)} + \mathbb{I}(a + \delta \geq u(w)), \tag{9.6}$$

which may be reduced to

$$H(a, w) = \frac{g_0(a - \delta \mid w)}{g_0(a \mid w)} + 1 \tag{9.7}$$

when the treatment A lies within the limits defined by the covariate strata W, that is, for $A_i \in (u(w) - \delta, u(w))$. The efficient influence function is a key ingredient in the construction of semiparametric-efficient estimators. Next, we focus on a targeted maximum likelihood (TML) estimator, for which Díaz and van der Laan (2018) give the following recipe:

1. Construct initial estimators g_n of $g_0(A, W)$ and Q_n of $\overline{Q}_0(A, W)$, ideally using data-adaptive regression techniques.
2. For each observation i, compute an estimate $H_n(a_i, w_i)$ of the auxiliary covariate $H(a_i, w_i)$.
3. Construct the one-dimensional logistic regression model,
$$\text{logit} Q_{\epsilon,n}(a, w) = \text{logit} Q_n(a, w) + \epsilon H_n(a, w),$$
or an analogous regression model incorporating H_n as weights. Estimate the regression model's parameter ϵ, obtaining ϵ_n. The outcome of this regression model yields $\overline{Q}_n^\star$.
4. Compute TML estimator Ψ_n of the target parameter, defining update $\overline{Q}_n^\star$ of the initial estimate $\overline{Q}_{n,\epsilon_n}$:
$$\psi_n = \Psi(P_n^\star) = \frac{1}{n} \sum_{i=1}^{n} \overline{Q}_n^\star(d(A_i, W_i), W_i). \tag{9.8}$$

As discussed previously, TML estimators are constructed so as to be *asymptotically linear* and are usually *doubly robust*. Asymptotic linearity means that the asymptotic difference between the estimator ψ_n and the target parameter ψ_0 can be expressed in terms of the EIF, that is,

$$\sqrt{n}(\psi_n - \psi_0) = \frac{1}{\sqrt{n}} \sum_{i=1}^{n} D(P_0)(O_i) + o_p(1). \tag{9.9}$$

Together with regularity, asymptotic linearity establishes a class of estimators whose asymptotic variance is bounded from below by the asymptotic variance of the EIF. This means that such estimators are solutions to the EIF estimating equation (i.e., plugging the TML estimator ψ_n into the EIF equation results in a solution close to zero) and that their sampling variance may be approximated by the variance of the EIF in closed form. This latter fact is computationally convenient, as resampling methods (e.g., the bootstrap) are not strictly necessary for variance estimation. A central limit theorem establishes that the asymptotic distribution of the estimator ψ_n is centered at ψ_0 and is Gaussian:

$$\sqrt{n}(\psi_n - \psi_0) \to \text{Normal}(0, \sigma^2(D(P_0))). \tag{9.10}$$

Thus, an estimate σ_n^2 of the variance $\sigma^2(D(P_0))$ may be computed

$$\sigma_n^2 = \frac{1}{n}\sum_{i=1}^{n} D^2(\overline{Q}_n^\star, g_n)(O_i), \tag{9.11}$$

allowing for Wald-style confidence intervals at coverage level $(1-\alpha)$ to be computed as $\psi_n \pm z_{(1-\alpha/2)} \cdot \sigma_n/\sqrt{n}$. Under certain conditions, the resampling based on the bootstrap may also be used to compute σ_n^2 (van der Laan and Rose, 2011).

9.5 Evaluating the Causal Effect of a Stochastic Intervention

To start, let's load the packages we'll be using throughout our simple data example

```
library(data.table)
library(haldensify)
library(sl3)
library(tmle3)
library(tmle3shift)
```

We need to estimate two components of the likelihood in order to construct a TML estimator. The first of these components is the outcome regression, $\overline{Q}_n$, which is a simple regression of the form $\mathbb{E}[Y \mid A, W]$. An estimate for such a quantity may be constructed using the Super Learner algorithm. We construct the components of an sl3-style Super Learner for a regression below, using a small variety of parametric and nonparametric regression techniques:

```
# learners used for conditional mean of the outcome
mean_lrnr <- Lrnr_mean$new()
fglm_lrnr <- Lrnr_glm_fast$new()
rf_lrnr <- Lrnr_ranger$new()
hal_lrnr <- Lrnr_hal9001$new(max_degree = 3, n_folds = 3)

# SL for the outcome regression
sl_reg_lrnr <- Lrnr_sl$new(
  learners = list(mean_lrnr, fglm_lrnr, rf_lrnr, hal_lrnr),
  metalearner = Lrnr_nnls$new()
)
```

The second of these is an estimate of the treatment mechanism, g_n, i.e., the *generalized propensity score*. In the case of a continuous intervention node A, such a quantity takes the form $p(A \mid W)$, which is a conditional density. Generally speaking, conditional density estimation is a challenging problem that has received much attention in the literature. To estimate the treatment mechanism, we must make use of learning algorithms specifically suited to conditional density estimation; a list of such learners may be extracted from `sl3` by using `sl3_list_learners()`:

```
sl3_list_learners("density")
[1] "Lrnr_density_discretize"     "Lrnr_density_hse"
[3] "Lrnr_density_semiparametric" "Lrnr_haldensify"
[5] "Lrnr_solnp_density"
```

To proceed, we'll select two of the above learners, `Lrnr_haldensify` for using the highly adaptive lasso for conditional density estimation, based on an algorithm given by Díaz and van der Laan (2011) and implemented in Hejazi et al. (2020a), and semiparametric location-scale conditional density estimators implemented in the `sl3` package. A Super Learner may be constructed by pooling estimates from each of these modified conditional density regression techniques (note that we exclude the approach based on the `haldensify` learner from our Super Learner on account of the computationally intensive nature of the approach).

```
# learners used for conditional densities for (g_n)
haldensify_lrnr <- Lrnr_haldensify$new(
  n_bins = c(5, 10, 20),
  lambda_seq = exp(seq(-1, -10, length = 200))
)
# semiparametric density estimator with homoscedastic errors (HOSE)
hose_hal_lrnr <- make_learner(Lrnr_density_semiparametric,
  mean_learner = hal_lrnr
)
# semiparametric density estimator with heteroscedastic errors (HESE)
hese_rf_glm_lrnr <- make_learner(Lrnr_density_semiparametric,
  mean_learner = rf_lrnr,
  var_learner = fglm_lrnr
)

# SL for the conditional treatment density
sl_dens_lrnr <- Lrnr_sl$new(
  learners = list(hose_hal_lrnr, hese_rf_glm_lrnr),
  metalearner = Lrnr_solnp_density$new()
)
```

Finally, we construct a `learner_list` object for use in constructing a TML estimator of our target parameter of interest:

```
learner_list <- list(Y = sl_reg_lrnr, A = sl_dens_lrnr)
```

The `learner_list` object above specifies the role that each of the ensemble learners we have generated is to play in computing initial estimators to be used in building a TMLE for the parameter of interest here. In particular, it makes explicit the fact that our `Q_learner` is used in fitting the outcome regression while our `g_learner` is used in estimating the treatment mechanism.

9.5.1 Example with Simulated Data

```
# simulate simple data for tmle-shift sketch
n_obs <- 400 # number of observations
tx_mult <- 2 # multiplier for the effect of W = 1 on the treatment

## baseline covariates -- simple, binary
W <- replicate(2, rbinom(n_obs, 1, 0.5))

## create treatment based on baseline W
A <- rnorm(n_obs, mean = tx_mult * W, sd = 1)

## create outcome as a linear function of A, W + white noise
Y <- rbinom(n_obs, 1, prob = plogis(A + W))

# organize data and nodes for tmle3
data <- data.table(W, A, Y)
setnames(data, c("W1", "W2", "A", "Y"))
node_list <- list(
  W = c("W1", "W2"),
  A = "A",
  Y = "Y"
)
head(data)
   W1 W2         A Y
1:  1  1  0.271651 1
2:  0  0 -0.663368 1
3:  0  0  0.113366 0
4:  0  1 -0.732558 0
5:  1  1  0.388835 1
6:  0  0  0.043986 0
```

The above composes our observed data structure $O = (W, A, Y)$. To formally express this fact using the tlverse grammar introduced by the tmle3 package, we create a single data object and specify the functional relationships between the nodes in the *directed acyclic graph* (DAG) via an SCM, reflected in the node list we set up.

We now have an observed data structure (data) and a specification of the role that each variable in the data set plays as the nodes in a DAG.

To start, we will initialize a specification for the TMLE of our parameter of interest (called a tmle3_Spec in the tlverse nomenclature) simply by calling tmle_shift. We specify the argument shift_val = 0.5 when initializing the tmle3_Spec object to communicate that we're interested in a shift of 0.5 on the scale of the treatment A – that is, we specify $\delta = 0.5$ (an arbitrarily chosen value for this example).

```
# initialize a tmle specification
tmle_spec <- tmle_shift(
  shift_val = 0.5,
  shift_fxn = shift_additive,
  shift_fxn_inv = shift_additive_inv
)
```

As seen above, the tmle_shift specification object (like all tmle3_Spec objects) does *not* store the data for our specific analysis of interest. Later, we'll see that passing a data object directly to the tmle3 wrapper function, alongside the instantiated tmle_spec, will serve to construct a tmle3_Task object internally (see the tmle3 documentation for details).

9.5.2 Targeted Estimation of Stochastic Interventions Effects

```
tmle_fit <- tmle3(tmle_spec, data, node_list, learner_list)

Iter: 1 fn: 548.8338     Pars:  0.94736 0.05264
Iter: 2 fn: 548.8338     Pars:  0.94736 0.05264
solnp--> Completed in 2 iterations
tmle_fit
A tmle3_Fit that took 1 step(s)
   type          param init_est tmle_est       se   lower   upper
1:  TSM E[Y_{A=NULL}]  0.76372  0.76011 0.022838 0.71535 0.80488
   psi_transformed lower_transformed upper_transformed
1:         0.76011           0.71535           0.80488
```

The print method of the resultant tmle_fit object conveniently displays the results of computing our TML estimator ψ_n. The standard error estimate is computed based on the estimated EIF.

9.6 Selecting Stable Stochastic Interventions

At times, a particular choice of the shift parameter δ may lead to positivity violations and downstream instability in the estimation process. In order to curb such issues, we can make choices of δ based on the impact of the candidate values on the estimator. Recall that a simplified expression of the auxiliary covariate for the TMLE of ψ is $H = \frac{g(a-\delta|w)}{g(a|w)}$, where $g(a - \delta \mid w)$ is defined by the stochastic intervention of interest. We can design our stochastic intervention to avoid violations of the positivity assumption by by considering a bound $C(\delta) = \frac{g(a-\delta|w)}{g(a|w)} < M$, where M is a potentially user-specified upper bound of $C(\delta)$. Note that $C(\delta)$ corresponds to the inverse weight assigned to the unit with counterfactual treatment value $A = a + \delta$, natural treatment value $A = a$, and covariates $W = w$. So, $C(\delta)$ can be viewed as a measure of the influence that a given observation has on the estimator ψ_n. By limiting $C(\delta)$, whether through a choice of M or δ, we can limit the potential instability of our estimator. We can formalize this procedure by defining a new shift function $\delta(A, W)$:

$$\delta(a,w) = \begin{cases} \delta, & \delta_{\text{min}}(a,w) \leq \delta \leq \delta_{\text{max}}(a,w) \\ \delta_{\text{max}}(a,w), & \delta \geq \delta_{\text{max}}(a,w) \\ \delta_{\text{min}}(a,w), & \delta \leq \delta_{\text{min}}(a,w) \end{cases}, \tag{9.12}$$

where

$$\delta_{\text{max}}(a,w) = \text{argmax}_{\left\{\delta \geq 0, \frac{g(a-\delta|w)}{g(a|w)} \leq M\right\}} \frac{g(a-\delta \mid w)}{g(a \mid w)}$$

and

$$\delta_{\text{min}}(a,w) = \text{argmin}_{\left\{\delta \leq 0, \frac{g(a-\delta|w)}{g(a|w)} \leq M\right\}} \frac{g(a-\delta \mid w)}{g(a \mid w)}.$$

The above provides a strategy for implementing a shift at the level of a given observation (a_i, w_i), thereby allowing for all observations to be shifted to an appropriate value, whether δ_{min}, δ, or δ_{max}. The `tmle3shift` package implements the functions `shift_additive_bounded` and `shift_additive_bounded_inv`, which define a variation of this strategy:

$$\delta(a,w) = \begin{cases} \delta, & C(\delta) \leq M \\ 0, \text{otherwise} & \end{cases}, \tag{9.13}$$

corresponding to an intervention in which the natural value of treatment $A = a$ is shifted by a value δ when the ratio $C(\delta)$ of the post-intervention density $g(a-\delta \mid w)$ to the natural treatment density $g(a \mid w)$ does not exceed a bound M. When $C(\delta)$ exceeds the bound M, the stochastic intervention exempts the given unit from the treatment modification, leaving them to their natural value of treatment $A = a$.

9.6.1 Initializing vimshift through its tmle3_Spec

To start, we will initialize a specification for the TMLE of our parameter of interest (called a tmle3_Spec in the tlverse nomenclature) simply by calling tmle_shift. We specify the argument shift_grid = seq(-1, 1, by = 1) when initializing the tmle3_Spec object to communicate that we're interested in assessing the mean counterfactual outcome over a grid of shifts $\delta \in \{-1, 0, 1\}$ on the scale of the treatment A (n.b., we make an arbitrary choice of shift values for this example).

```
# what's the grid of shifts we wish to consider?
delta_grid <- seq(-1, 1, 1)

# initialize a tmle specification
tmle_spec <- tmle_vimshift_delta(
  shift_grid = delta_grid,
  max_shifted_ratio = 2
)
```

As seen above, the tmle_vimshift specification object (like all tmle3_Spec objects) does *not* store the data for our specific analysis of interest. Later, we'll see that passing a data object directly to the tmle3 wrapper function, alongside the instantiated tmle_spec, will serve to construct a tmle3_Task object internally (see the tmle3 documentation for details).

9.6.2 Targeted Estimation of Stochastic Interventions Effects

One may walk through the step-by-step procedure for fitting the TML estimator of the mean counterfactual outcome under each shift in the grid, using the machinery exposed by the tmle3 R package.

One may invoke the tmle3 wrapper function (a user-facing convenience utility) to fit the series of TML estimators (one for each parameter defined by the grid delta) in a single function call:

```
tmle_fit <- tmle3(tmle_spec, data, node_list, learner_list)
```

```
Iter: 1 fn: 547.4323     Pars:  0.9998973 0.0001027
Iter: 2 fn: 547.4323     Pars:  0.99997059 0.00002941
solnp--> Completed in 2 iterations
```

```
tmle_fit
A tmle3_Fit that took 1 step(s)
         type          param init_est tmle_est        se   lower   upper
1:        TSM  E[Y_{A=NULL}]  0.56957  0.57130 0.0213472 0.52946 0.61314
2:        TSM  E[Y_{A=NULL}]  0.69881  0.69750 0.0229958 0.65243 0.74257
3:        TSM  E[Y_{A=NULL}]  0.81733  0.81516 0.0171125 0.78162 0.84870
4: MSM_linear MSM(intercept)  0.69524  0.69465 0.0190943 0.65723 0.73208
5: MSM_linear     MSM(slope)  0.12388  0.12193 0.0085622 0.10515 0.13871
   psi_transformed lower_transformed upper_transformed
1:         0.57130           0.52946           0.61314
2:         0.69750           0.65243           0.74257
3:         0.81516           0.78162           0.84870
4:         0.69465           0.65723           0.73208
5:         0.12193           0.10515           0.13871
```

Remark: The `print` method of the resultant `tmle_fit` object conveniently displays the results from computing our TML estimator.

9.6.3 Estimation and Inference with Marginal Structural Models

It can be challenging to select a value of the shift parameter δ in advance. One solution is to specify a *grid* of such shifts δ to be used in defining a set of related stochastic interventions (Hejazi et al., 2020c). When we consider estimating the counterfactual mean ψ_n under several choices of δ, a single summary measure of these estimated quantities can be established through working marginal structural models (MSMs). Summarizing the estimates ψ_n through a working MSM allows for inference on the *trend* appearing through the grid in δ, which may be evaluating through a simple hypothesis test on the slope parameter β_0 of the working MSM. Consider a grid of δ, $\{\delta_1, \ldots, \delta_k\}$, corresponding to counterfactual means $\{\psi_{\delta_1}, \ldots, \psi_{\delta_k}\}$. Next, let $\psi(\delta) = (\psi_\delta : \delta)$ denote the grid of counterfactual means in the grid defined by δ and let $\psi_n(\delta)$ denote TML estimators of $\psi(\delta)$. The MSM summarizing the change in ψ_n as a function of δ may be expressed $m_\beta(\psi_\delta) = \beta_0 + \beta_1\delta$. This simple working model summarizes the changes in ψ_δ as a function of the parameters (β_0, β_1), where the latter is the slope of the line resulting from projecting the counterfactual means onto this simple two-parameter working model.

A more general expression for the MSM $m_\beta(\delta)$ is $\beta_0 = \text{argmin}_\beta \sum_\delta (\psi_\delta(P_0) - m_\beta(\delta))^2 h(\delta)$, the solution to the estimating equation

$$u(\beta, (\psi_\delta : \delta)) = \sum_\delta h(\delta) \left(\psi_\delta(P_0) - m_\beta(\delta)\right) \frac{d}{d\beta} m_\beta(\delta) = 0.$$

Now, say, $\psi = (\psi(\delta) : \delta)$ is d-dimensional. We may express the EIF of the MSM parameter

β_0 in terms of the EIFs of the individual counterfactual means:

$$D_\beta(O) = \left(\sum_\delta h(\delta)\frac{d}{d\beta}m_\beta(\delta)\frac{d}{d\beta}m_\beta(\delta)^t\right)^{-1}\sum_\delta h(\delta)\frac{d}{d\beta}m_\beta(\delta)D_{\psi_\delta}(O). \tag{9.14}$$

Here, in Equation (9.14), the first component is of dimension $d \times d$ and the second is of dimension $d \times 1$. In the above, we assume a linear working MSM; however, an analogous procedure may be applied for working MSMs based on GLMs.

Above, we utilized a straightforward application of the delta method to obtain the EIF of β. Inference for this parameter of a working MSM follows from evaluation of its EIF D_β, which is expressed in terms of the EIFs of each of the corresponding estimates $\psi_n(\delta)$. The limit distribution of β_n may be expressed

$$\sqrt{n}(\beta_n - \beta_0) \to N(0, \Sigma),$$

where Σ is the empirical covariance matrix of $D_\beta(O)$. With this, we can not only estimate the trend through the counterfactual means across a grid in δ, but we can also evaluate whether the slope estimate is statistically significant, in terms of hypothesis tests of the form $(H_0 : \beta_0 = 0; H_1 : \beta_0 \neq 0)$ and equivalent Wald-style confidence intervals. Note that the estimator β_n of the parameter β_0 of the MSM is asymptotically linear (and, in fact, a TML estimator) as a consequence of its construction from individual TML estimators.

The strategy just discussed constructs an estimate β_n of the working MSM slope β_0 by first evaluating the TML estimates of the counterfactual means $\psi_{n,\delta}$ in the grid $\{\delta_1, \ldots, \delta_k\}$; however, this is not necessarily the best strategy, especially when giving consideration to estimation stability in small samples. In smaller samples, it may be prudent to perform TML estimation targeting directly the parameter β_0, as opposed to constructing it by applying the delta method to several independently targeted TML estimates.

To do so, consider a TML estimator targeting β_0 (the parameter of the working MSM m_β), which uses a targeting update step of the form $\overline{Q}_{n,\epsilon}(A, W) = \overline{Q}_n(A, W) + \epsilon(H_{\beta_0}(g), H_{\beta_1}(g))$, for all δ, where $H_{\beta_0}(g)$ is the auxiliary covariate for β_0 (the intercept) and $H_\beta(g)$ is the auxiliary covariate for β_1 (the slope). Note that the forms of these auxiliary covariates depend on the EIF D_β. Such a TML estimator avoids estimating each of the ψ_δ in the grid directly, instead cleverly concatenating their auxiliary covariates into those appropriate for β_0 and β_1. To construct a targeted maximum likelihood estimator that directly targets the parameters of the working MSM, we may use the `tmle_vimshift_msm` Spec (instead of the `tmle_vimshift_delta` Spec).

```
# initialize a tmle specification
tmle_msm_spec <- tmle_vimshift_msm(
  shift_grid = delta_grid,
  max_shifted_ratio = 2
)

# fit the TML estimator and examine the results
tmle_msm_fit <- tmle3(tmle_msm_spec, data, node_list, learner_list)
tmle_msm_fit
```

9.6.4 Example with the WASH Benefits Data

To complete our walk through, let's turn to using stochastic interventions to investigate the data from the WASH Benefits trial. To start, let's load the data, convert all columns to be of class `numeric`, and take a quick look at it

```
washb_data <- fread(
  paste0(
    "https://raw.githubusercontent.com/tlverse/tlverse-data/master/",
    "wash-benefits/washb_data.csv"
  ),
  stringsAsFactors = TRUE
)
washb_data <- washb_data[!is.na(momage), lapply(.SD, as.numeric)]
head(washb_data, 3)
     whz tr fracode month aged sex momage momedu momheight hfiacat Nlt18 Ncomp
1:  0.00  1       4     9  268   2     30      2    146.40       1     3    11
2: -1.16  1       4     9  286   2     25      2    148.75       3     2     4
3: -1.05  1      20     9  264   2     25      2    152.15       1     1    10
   watmin elec floor walls roof asset_wardrobe asset_table asset_chair
1:      0    1     0     1    1              0           1           1
2:      0    1     0     1    1              0           1           0
3:      0    0     0     1    1              0           0           1
   asset_khat asset_chouki asset_tv asset_refrig asset_bike asset_moto
1:          1            0        1            0          0          0
2:          1            1        0            0          0          0
3:          0            1        0            0          0          0
   asset_sewmach asset_mobile
1:             0            1
2:             0            1
3:             0            1
```

Next, we specify our NPSEM via the `node_list` object. For our example analysis, we'll consider the outcome to be the weight-for-height Z-score (as in previous chapters), the intervention of interest to be the mother's age at time of child's birth, and take all other covariates to be potential confounders.

```
node_list <- list(
  W = names(washb_data)[!(names(washb_data) %in%
    c("whz", "momage"))],
  A = "momage",
  Y = "whz"
)
```

Were we to consider the counterfactual weight-for-height Z-score under shifts in the age of the mother at child's birth, how would we interpret estimates of our parameter? To simplify our interpretation, consider a shift of just a year in the mother's age (i.e., $\delta = 1$); in this setting, a stochastic intervention would correspond to a policy advocating that potential mothers defer having a child for a single calendar year, possibly implemented through an encouragement design deployed in a clinical setting.

For this example, we'll use the variable importance strategy of considering a grid of stochastic interventions to evaluate the weight-for-height Z-score under a shift in the mother's age down by two years ($\delta = -2$) or up by two years ($\delta = 2$). To do this, we simply initialize a `Spec` `tmle_vimshift_delta` just as we did in a previous example:

```
# initialize a tmle specification for the variable importance parameter
washb_vim_spec <- tmle_vimshift_delta(
  shift_grid = c(-2, 2),
  max_shifted_ratio = 2
)
```

Prior to running our analysis, we'll modify the `learner_list` object we had created such that the density estimation procedure we rely on will be only the location-scale conditional density estimation procedure, as the nonparametric conditional density approach based on the highly adaptive lasso (Díaz and van der Laan, 2011; Benkeser and van der Laan, 2016; Coyle et al., 2020; Hejazi et al., 2020b,a) is currently unable to accommodate larger datasets.

```
# we need to turn on cross-validation for the HOSE learner
cv_hose_hal_lrnr <- Lrnr_cv$new(
  learner = hose_hal_lrnr,
  full_fit = TRUE
)

# modify learner list, using existing SL for Q fit
learner_list <- list(Y = sl_reg_lrnr, A = cv_hose_hal_lrnr)
```

Having made the above preparations, we're now ready to estimate the counterfactual mean of the weight-for-height Z-score under a small grid of shifts in the mother's age at child's birth. Just as before, we do this through a simple call to our `tmle3` wrapper function:

```
washb_tmle_fit <- tmle3(washb_vim_spec, washb_data, node_list, learner_list)
washb_tmle_fit
```

9.7 Exercises

9.7.1 The Ideas in Action

Exercise 9.1. Set the `sl3` library of algorithms for the Super Learner to a simple, interpretable library and use this new library to estimate the counterfactual mean of mother's age at child's birth (`momage`) under a shift $\delta = 0$. What does this counterfactual mean equate to in terms of the observed data?

Solution. Forthcoming

Exercise 9.2. Using a grid of values of the shift parameter δ (e.g., $\{-1, 0, +1\}$), repeat the analysis on the variable chosen in the preceding question, summarizing the trend for this sequence of shifts using a marginal structural model.

Solution. Forthcoming

Exercise 9.3. Repeat the preceding analysis, using the same grid of shifts, but instead directly targeting the parameters of the marginal structural model. Interpret the results – that is, what does the slope of the marginal structural model tell us about the trend across the chosen sequence of shifts?

Solution. Forthcoming

9.7.2 Review of Key Concepts

Exercise 9.4. Describe two (equivalent) ways in which the causal effects of stochastic interventions may be interpreted.

Solution. Forthcoming

Exercise 9.5. How can the information provided by estimates across several shifts $\{\delta_1, \dots, \delta_k\}$ and the marginal structural model parameter summarizing the trend in δ be used to enrich the interpretation of our findings?

Solution. Forthcoming

Exercise 9.6. What advantages, if any, are there to targeting directly the parameters of a marginal structural model?

Solution. Forthcoming

10

Causal Mediation Analysis

Nima Hejazi

Based on the `tmle3mediate` `R` package by *Nima Hejazi, James Duncan, and David McCoy*.

Updated: 2021-12-30

\begin{VT1} *Learning Objectives*

1. Examine how the presence of post-treatment mediating variables can complicate a causal analysis, and how direct and indirect effects can be defined to resolve these complications.
2. Describe the essential similarities and differences between direct and indirect causal effects, including their definition in terms of stochastic interventions.
3. Differentiate the joint interventions required to define direct and indirect effects from the static, dynamic, and stochastic interventions that yield *total* causal effects.
4. Describe the assumptions needed for identification of the natural direct and indirect effects, as well as the limitations of these effect definitions.
5. Estimate the natural direct and indirect effects for a binary treatment using the `tmle3mediate` `R` package.
6. Differentiate the population intervention direct and indirect effects of stochastic interventions from the natural direct and indirect effects, including differences in the assumptions required for their identification.
7. Estimate the population intervention direct effect of a binary treatment using the `tmle3mediate` `R` package.

\end{VT1}

10.1 Causal Mediation Analysis

In applications ranging from biology and epidemiology to economics and psychology, scientific inquires are often concerned with ascertaining the effect of a treatment on an outcome variable only through particular pathways between the two. In the presence of post-treatment intermediate variables affected by exposure (that is, *mediators*), path-specific effects allow for such complex, mechanistic relationships to be teased apart. These causal effects are of such wide interest that their definition and identification has been the object of study in statistics for nearly a century – indeed, the earliest examples of modern causal mediation analysis can be traced back to work on path analysis (Wright, 1934). In recent decades, renewed interest has resulted in the formulation of novel direct and indirect effects within both the potential outcomes and nonparametric structural equation modeling frameworks (Robins, 1986b; Pearl, 1995, 2009; Spirtes et al., 2000; Dawid, 2000). Generally, the indirect effect (IE) is the portion of the total effect found to work *through* mediating variables, while the direct effect (DE) encompasses *all other components* of the total effect, including both the effect of the treatment directly on the outcome *and* its effect through all paths not explicitly involving the mediators. The mechanistic knowledge conveyed by the direct and indirect effects can be used to improve understanding of both *why* and *how* treatments may be efficacious.

Modern approaches to causal inference have allowed for significant advances over the methodology of traditional path analysis, overcoming significant restrictions imposed by the use of parametric modeling approaches (VanderWeele, 2015). Using distinct frameworks, Robins and Greenland (1992) and Pearl (2001) provided equivalent nonparametric decompositions of the average treatment effect into the *natural* direct and indirect effects. VanderWeele (2015) provides a comprehensive overview of classical causal mediation analysis. We provide an alternative perspective, focusing instead on the construction of efficient estimators of these quantities, which have appeared only recently (Tchetgen Tchetgen and Shpitser, 2012; Zheng and van der Laan, 2012), as well as on more flexible direct and indirect definitions based upon stochastic interventions (Díaz and Hejazi, 2020).

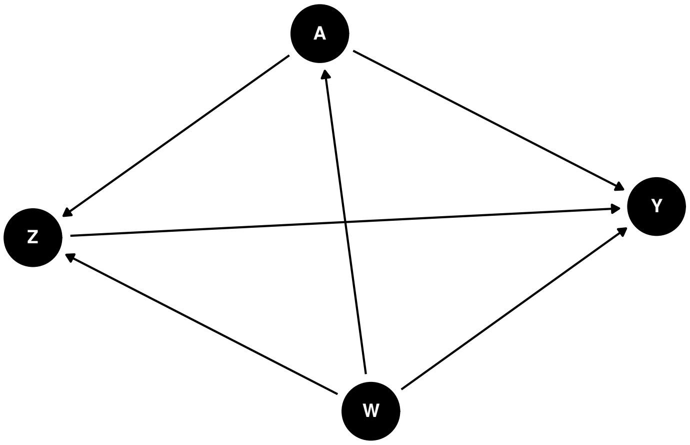
10.2 Data Structure and Notation

Let us return to our familiar sample of n units $O_1, \ldots, O_n$, where we now consider a slightly more complex data structure $O = (W, A, Z, Y)$ for any given observational unit. As before, W represents a vector of observed covariates, A a binary or continuous treatment, and Y a binary or continuous outcome; the new post-treatment variable Z represents a (possibly multivariate) set of mediators. Avoiding assumptions unsupported by background scientific knowledge, we assume only that $O \sim P_0 \in \mathcal{M}$, where $\mathcal{M}$ is the nonparametric statistical model that places no assumptions on the form of the data-generating distribution P_0.

As in preceding chapters, a structural causal model (SCM) (Pearl, 2009) helps to formalize the definition of our counterfactual variables:

$$\begin{aligned} W &= f_W(U_W) \\ A &= f_A(W, U_A) \\ Z &= f_Z(W, A, U_Z) \\ Y &= f_Y(W, A, Z, U_Y). \end{aligned} \tag{10.1}$$

This set of equations constitutes a mechanistic model generating the observed data O; furthermore, the SCM encodes several fundamental assumptions. Firstly, there is an implicit temporal ordering: W occurs first, depending only on exogenous factors U_W; A happens next, based on both W and exogenous factors U_A; then come the mediators Z, which depend on A, W, and another set of exogenous factors U_Z; and finally appears the outcome Y. We assume neither access to the set of exogenous factors $\{U_W, U_A, U_Z, U_Y\}$ nor knowledge of the forms of the deterministic generating functions $\{f_W, f_A, f_Z, f_Y\}$. In practice, any available knowledge about the data-generating experiment should be incorporated into this model – for example, if the data from a randomized controlled trial (RCT), the form of f_A may be known. The SCM corresponds to the following DAG:



By factorizing the likelihood of the data O, we can express p_0, the density of O with respect to the product measure, when evaluated on a particular observation o, in terms of several orthogonal components:

$$\begin{aligned} p_0(x) =& q_{0,Y}(y \mid Z = z, A = a, W = w) \\ & q_{0,Z}(z \mid A = a, W = w) \\ & g_{0,A}(a \mid W = w) \\ & q_{0,W}(w). \end{aligned} \tag{10.2}$$

In Equation (**??**), $q_{0,Y}$ is the conditional density of Y given $\{Z, A, W\}$, $q_{0,Z}$ is the conditional density of Z given $\{A, W\}$, $g_{0,A}$ is the conditional density of A given W, and $q_{0,W}$ is the marginal density of W. For convenience and consistency of notation, we will define $\overline{Q}_Y(Z, A, W) := \mathbb{E}[Y \mid Z, A, W]$ and $g(A \mid W) := \mathbb{P}(A \mid W)$ (i.e., the propensity score).

We have explicitly excluded potential confounders of the mediator-outcome relationship affected by exposure (i.e., variables affected by A and affecting both Z and Y). Mediation analysis in the presence of such variables is challenging (Avin et al., 2005); thus, most efforts to develop definitions of causal direct and indirect effects explicitly assume the absence of such confounders. While we will refrain from discussing the matter in detail, the interested reader may wish to consult recent advances in the vast literature on causal mediation analysis, including *interventional* direct and indirect effects (Didelez et al., 2006; VanderWeele et al., 2014; Lok, 2016; Vansteelandt and Daniel, 2017; Rudolph et al., 2017; Nguyen et al., 2019), whose identification is robust to this complex form of post-treatment confounding. Within this thread of the literature, Díaz et al. (2020) and Benkeser and Ran (2021) provide considerations of nonparametric effect decompositions and efficiency theory, while Hejazi et al. (2021) formulate a novel class of effects utilizing stochastic interventions.

10.3 Defining the Natural Direct and Indirect Effects

10.3.1 Decomposing the Average Treatment Effect

The natural direct and indirect effects arise from a decomposition of the ATE:

$$\mathbb{E}[Y(1)-Y(0)] = \underbrace{\mathbb{E}[Y(1,Z(0))-Y(0,Z(0))]}_{\text{NDE}} + \underbrace{\mathbb{E}[Y(1,Z(1))-Y(1,Z(0))]}_{\text{NIE}}.$$

In particular, the natural indirect effect (NIE) measures the effect of the treatment $A \in \{0,1\}$ on the outcome Y through the mediators Z, while the natural direct effect (NDE) measures the effect of the treatment on the outcome *through all other paths*. Identification of the natural direct and indirect effects requires the following non-testable causal assumptions. Note that the standard assumptions of consistency and no interference (i.e., SUTVA (Rubin, 1978, 1980)) hold owing to the fact that (1) the SCM we consider is restricted so as to give rise only to independent and identically distributed units; and (2) consistency is an implied property of the SCM, as counterfactuals are derived quantities (as opposed to primitive quantities in the potential outcomes framework). Pearl (2010) provides an illuminating discussion on this latter point.

::: {.definition name = "Exchangeability"} $Y(a,z) \perp\!\!\!\perp (A,Z) \mid W$, which further implies that $\mathbb{E}\{Y(a,z) \mid A=a, W=w, Z=z\} \equiv \mathbb{E}\{Y(a,z) \mid W=w\}$. This is a special, more restrictive case of the standard assumption of no unmeasured counfounding in the presence of mediators. The analogous randomization assumption is simply the standard randomization assumption applied to a joint intervention on both treatment and mediators. :::

::: {.definition name = "Treatment positivity"} For any $a \in \mathcal{A}$ and $w \in \mathcal{W}$, the conditional probability of treatment $g(a \mid w)$ is bounded away from the limits of the unit interval by a small factor $\xi > 0$. More precisely, $\xi < g(a \mid w) < 1-\xi$. This mirrors the standard positivity assumption required for static interventions, discussed previously. :::

::: {.definition name = "Mediator positivity"} For any $z \in \mathcal{Z}$, $a \in \mathcal{A}$, and $w \in \mathcal{W}$, the conditional mediator density must be bounded away from zero by a small factor $\epsilon > 0$, specifically, $\epsilon < q_{0,Z}(z \mid a,w)$. Essentially, this requires that the conditional mediator density be bounded away from zero for all $\{z,a,w\}$ in their joint support $\mathcal{Z} \times \mathcal{A} \times \mathcal{W}$, which is to say that it must be possible to observe any given mediator value across all strata defined by both treatment and baseline covariates. A less restrictive form of this assumption is also possible – specifically, that the ratio of the mediator densities under both treatment contrasts be bounded for both realizations of the mediator density. :::

::: {.definition name = "Cross-world counterfactual independence"} For all $a \neq a'$, where $a, a' \in \mathcal{A}$, and $z \in \mathcal{Z}$, $Y(a', z)$ must be independent of $Z(a)$, given W. That is, the counterfactual outcome under the treatment contrast $a' \in \mathcal{A}$ and the counterfactual mediator value $Z(a) \in \mathcal{Z}$ (under the alternative contrast $a \in \mathcal{A}$) must both be observable. The term "cross-world" refers to the two counterfactuals $Z(a)$ and $Y(a', z)$ existing under two differing treatment contrasts. Though the joint distributions of these counterfactuals are well-defined, they cannot be jointly realized. :::

While the first three assumptions may be familiar based on their simpler analogs, the cross-world independence requirement is unique to the NDE and NIE. This assumption resolves a thorny complication to the identification of these path-specific effects, which has been termed the "recanting witness" by Avin et al. (2005), who introduce a graphical resolution equivalent to this assumption. This independence of counterfactuals indexed by distinct interventions is, in fact, a serious limitation to the scientific relevance of these effect definitions, as it results in the NDE and NIE being unidentifiable in randomized trials (Robins and Richardson, 2010), directly implying that corresponding scientific claims cannot be falsified through experimentation (Popper, 1934; Dawid, 2000), contradicting a key pillar of the scientific method.

While many attempts have been made to weaken this last assumption (Petersen et al., 2006; Imai et al., 2010; Vansteelandt et al., 2012; Vansteelandt and VanderWeele, 2012), these results either impose stringent modeling assumptions, propose alternative interpretations of the natural effects, or provide a limited degree of additional flexibility by developing conditions that may more easily be satisfied. The motivated reader may wish to further examine these details independently. We next review estimation of the NDE and NIE, which remain widely used in modern applications of causal mediation analysis.

10.3.2 Estimating the Natural Direct Effect

The NDE is defined as

$$\begin{aligned}\Psi_{NDE} =& \mathbb{E}[Y(1, Z(0)) - Y(0, Z(0))] \\ \overset{\text{rand.}}{=}& \sum_w \sum_z [\underbrace{\mathbb{E}(Y \mid A = 1, z, w)}_{\overline{Q}_Y(A=1,z,w)} - \underbrace{\mathbb{E}(Y \mid A = 0, z, w)}_{\overline{Q}_Y(A=0,z,w)}] \\ & \times \underbrace{p(z \mid A = 0, w)}_{q_Z(Z|0,w))} \underbrace{p(w)}_{q_W},\end{aligned}$$

where the likelihood factors arise from a factorization of the joint likelihood:

$$p(w, a, z, y) = \underbrace{p(y \mid w, a, z)}_{q_Y(A,W,Z)} \underbrace{p(z \mid w, a)}_{q_Z(Z|A,W)} \underbrace{p(a \mid w)}_{g(A|W)} \underbrace{p(w)}_{q_W}.$$

The process of estimating the NDE begins by constructing $\overline{Q}_{Y,n}$, an estimate of the conditional mean of the outcome, given Z, A, and W. With an estimate of this conditional mean in hand, predictions of the quantities $\overline{Q}_Y(Z, 1, W)$ (setting $A = 1$) and, likewise, $\overline{Q}_Y(Z, 0, W)$ (setting $A = 0$) are readily obtained. We denote the difference of these conditional means $\overline{Q}_{\text{diff}} = \overline{Q}_Y(Z, 1, W) - \overline{Q}_Y(Z, 0, W)$. $\overline{Q}_{\text{diff}}$ represents the difference in the conditional mean of Y across contrasts of A.

A procedure for constructing a targeted maximum likelihood (TML) estimator of the NDE treats $\overline{Q}_{\text{diff}}$ itself as a nuisance parameter, regressing its estimate $\overline{Q}_{\text{diff},n}$ on baseline covariates W, among observations in the control condition only (i.e., those for whom $A = 0$ is observed); the goal of this step is to remove part of the marginal impact of Z on $\overline{Q}_{\text{diff}}$, since the covariates W precede the mediators Z in time. Regressing this difference on W among the controls recovers the expected $\overline{Q}_{\text{diff}}$, had all individuals been set to the control condition $A = 0$. Any residual additive effect of Z on $\overline{Q}_{\text{diff}}$ is removed during the TML estimation step using the auxiliary (or "clever") covariate, which accounts for the mediators Z. This auxiliary covariate takes the form

$$C_Y(q_Z, g)(O) = \left\{ \frac{\mathbb{I}(A = 1)}{g(1 \mid W)} \frac{q_Z(Z \mid 0, W)}{q_Z(Z \mid 1, W)} - \frac{\mathbb{I}(A = 0)}{g(0 \mid W)} \right\}.$$

Breaking this down, $\mathbb{I}(A = 1)/g(1 \mid W)$ is the inverse propensity score weight for $A = 1$ and, likewise, $\mathbb{I}(A = 0)/g(0 \mid W)$ is the inverse propensity score weight for $A = 0$. The middle term is the ratio of the conditional densities of the mediator under the control ($A = 0$) and treatment ($A = 1$) conditions (n.b., recall the mediator positivity condition above).

This subtle appearance of a ratio of conditional densities is concerning – tools to estimate such quantities are sparse in the statistics literature, unfortunately, and the problem is still more complicated (and computationally taxing) when Z is high-dimensional. As only the ratio of these conditional densities is required, a convenient re-parametrization may be achieved, that is,

$$\frac{p(A = 0 \mid Z, W)g(0 \mid W)}{p(A = 1 \mid Z, W)g(1 \mid W)}.$$

Going forward, we will denote this re-parameterized conditional probability $e(A \mid Z, W) := p(A \mid Z, W)$. The same re-parameterization technique has been used by Zheng and van der Laan (2012), Tchetgen Tchetgen (2013), Díaz and Hejazi (2020), and Díaz et al. (2020) in similar contexts. This reformulation is particularly useful for the fact that it reduces the estimation problem to one requiring only the estimation of conditional means, opening the door to the use of a wide range of machine learning algorithms, as discussed previously.

Underneath the hood, the outcome difference $\overline{Q}_{\text{diff}}$ and $e(A \mid Z, W)$, the conditional probability of A given Z and W, are used in constructing the auxiliary covariate for TML estimation. These nuisance parameters play an important role in the bias-correcting update step of the TML estimation procedure.

10.3.3 Estimating the Natural Indirect Effect

Derivation and estimation of the NIE is analogous to that of the NDE. Recall that the NIE is the effect of A on Y *only through the mediator(s)* Z. This quantity, which may be expressed $\mathbb{E}(Y(Z(1), 1) - \mathbb{E}(Y(Z(0), 1)$, corresponds to the difference of the conditional mean of Y given $A = 1$ and $Z(1)$ (the values the mediator would take under $A = 1$) and the conditional expectation of Y given $A = 1$ and $Z(0)$ (the values the mediator would take under $A = 0$).

As with the NDE, re-parameterization can be used to estimate $\mathbb{E}(A \mid Z, W)$, avoiding estimation of a possibly multivariate conditional density. However, in this case, the mediated mean outcome difference, denoted $\Psi_Z(Q)$, is instead estimated as follows

$$\Psi_{\text{NIE}}(Q) = \mathbb{E}(\Psi_{\text{NIE},Z}(Q)(1, W) - \Psi_{\text{NIE},Z}(Q)(0, W))$$

Here, $\bar{Q}_Y(Z, 1, W)$, the predicted values for Y given Z and W when $A = 1$, is regressed on W, among only the treated units (i.e., for whom $A = 1$ is observed) to obtain the conditional mean $\Psi_{\text{NIE},Z}(Q)(1, W)$. Performing the same procedure, instead regressing $\bar{Q}_Y(Z, 1, W)$ on W among only the control units (i.e., for whom $A = 0$ is observed) yields $\Psi_{\text{NIE},Z}(Q)(0, W)$. The difference of these two estimates is the NIE and can be thought of as the additive marginal effect of treatment on the conditional mean of Y given

$\{W, A = 1, Z\}$ through its effect on Z. So, in the case of the NIE, our estimate ψ_n is slightly different, but the same quantity $e(A \mid Z, W)$ comes into play as the auxiliary covariate.

10.4 The Population Intervention Direct and Indirect Effects

At times, the natural direct and indirect effects may prove too limiting, as these effect definitions are based on *static interventions* (i.e., setting $A = 0$ or $A = 1$), which may be unrealistic for real-world interventions. In such cases, one may turn instead to the population intervention direct effect (PIDE) and the population intervention indirect effect (PIIE), which are based on decomposing the effect of the population intervention effect (PIE) of flexible stochastic interventions (Díaz and Hejazi, 2020).

A particular type of stochastic intervention well-suited to working with binary treatments is the *incremental propensity score intervention* (IPSI), first proposed by Kennedy (2019). Such interventions do not deterministically set the treatment level of an observed unit to a fixed quantity (i.e., setting $A = 1$), but instead *alter the odds of receiving the treatment* by a fixed amount ($0 \leq \delta \leq \infty$) for each individual. In particular, this intervention takes the form

$$g_\delta(1 \mid w) = \frac{\delta g(1 \mid w)}{\delta g(1 \mid w) + 1 - g(1 \mid w)},$$

where the scalar $0 < \delta < \infty$ specifies a *change in the odds of receiving treatment*. As described by Díaz and Hejazi (2020), this stochastic intervention is a special case of exponential tilting, a framework that unifies post-intervention treatment values that are draws from an altered distribution.

Unlike the natural direct and indirect effects, the conditions required for identifiability of the population intervention direct and indirect effects are more lax. Most importantly, these differences involve a (1) treatment positivity assumption that only requires that the counterfactual treatment be in the observed support of the treatment $\mathcal{A}$, and (2) no requirement of the independence any cross-world counterfactuals.

10.4.1 Decomposing the Population Intervention Effect

We may decompose the population intervention effect (PIE) in terms of the *population intervention direct effect* (PIDE) and the *population intervention indirect effect* (PIIE):

$$\mathbb{E}\{Y(A_\delta)\} - \mathbb{E}Y = \overbrace{\mathbb{E}\{Y(A_\delta, Z(A_\delta)) - Y(A_\delta, Z)\}}^{\text{PIIE}} + \overbrace{\mathbb{E}\{Y(A_\delta, Z) - Y(A, Z)\}}^{\text{PIDE}}.$$

This decomposition of the PIE as the sum of the population intervention direct and indirect effects has an interpretation analogous to the corresponding standard decomposition of the average treatment effect. In the sequel, we will compute each of the components of the direct and indirect effects above using appropriate estimators as follows

- For $\mathbb{E}\{Y(A, Z)\}$, the sample mean $\frac{1}{n}\sum_{i=1}^{n} Y_i$ is consistent;
- for $\mathbb{E}\{Y(A_\delta, Z)\}$, a TML estimator for the effect of a joint intervention altering the treatment mechanism but not the mediation mechanism, based on the proposal in Díaz and Hejazi (2020); and,
- for $\mathbb{E}\{Y(A_\delta, Z_{A_\delta})\}$, an efficient estimator for the effect of a joint intervention altering both the treatment and mediation mechanisms, as proposed in Kennedy (2019) and implemented in the `npcausal` R package.

10.4.2 Estimating the Effect Decomposition Term

As described by Díaz and Hejazi (2020), the statistical functional identifying the decomposition term that appears in both the PIDE and PIIE $\mathbb{E}\{Y(A_\delta, Z)\}$, which corresponds to altering the treatment mechanism while keeping the mediation mechanism fixed, is

$$\theta_0(\delta) = \int m_0(a, z, w) g_{0,\delta}(a \mid w) p_0(z, w) d\nu(a, z, w),$$

for which a TML estimator is available. The corresponding *efficient influence function* (EIF) with respect to the nonparametric model $\mathcal{M}$ is $D_{\eta,\delta}(o) = D^{Y}_{\eta,\delta}(o) + D^{A}_{\eta,\delta}(o) + D^{Z,W}_{\eta,\delta}(o) - \theta(\delta)$.

The TML estimator may be computed basd on the EIF estimating equation and may incorporate cross-validation (Zheng and van der Laan, 2011; Chernozhukov et al., 2018) to circumvent possibly restrictive entropy conditions (e.g., Donsker class). The resultant estimator is

$$\hat{\theta}(\delta) = \frac{1}{n}\sum_{i=1}^{n} D_{\hat{\eta}_{j(i)},\delta}(O_i) = \frac{1}{n}\sum_{i=1}^{n} \left\{ D^{Y}_{\hat{\eta}_{j(i)},\delta}(O_i) + D^{A}_{\hat{\eta}_{j(i)},\delta}(O_i) + D^{Z,W}_{\hat{\eta}_{j(i)},\delta}(O_i) \right\},$$

which is implemented in `tmle3mediate` (a one-step estimator is also avaialble, in the `medshift` R package). We demonstrate the use of `tmle3mediate` to obtain $\mathbb{E}\{Y(A_\delta, Z)\}$ via its TML estimator.

10.5 Evaluating the Direct and Indirect Effects

We now turn to estimating the natural direct and indirect effects, as well as the population intervention direct effect, using the WASH Benefits data, introduced in earlier chapters. Let's first load the data:

```
library(data.table)
library(sl3)
library(tmle3)
library(tmle3mediate)

# download data
washb_data <- fread(
  paste0(
    "https://raw.githubusercontent.com/tlverse/tlverse-data/master/",
    "wash-benefits/washb_data.csv"
  ),
  stringsAsFactors = TRUE
)

# make intervention node binary and subsample
washb_data <- washb_data[sample(.N, 600), ]
washb_data[, tr := as.numeric(tr != "Control")]
```

We'll next define the baseline covariates W, treatment A, mediators Z, and outcome Y nodes of the NPSEM via a "Node List" object:

```
node_list <- list(
  W = c(
    "momage", "momedu", "momheight", "hfiacat", "Nlt18", "Ncomp", "watmin",
    "elec", "floor", "walls", "roof"
  ),
  A = "tr",
  Z = c("sex", "month", "aged"),
  Y = "whz"
)
```

Here the `node_list` encodes the parents of each node – for example, Z (the mediators) have parents A (the treatment) and W (the baseline confounders), and Y (the outcome) has parents Z, A, and W. We'll also handle any missingness in the data by invoking `process_missing`:

```
processed <- process_missing(washb_data, node_list)
```

```
washb_data <- processed$data
node_list <- processed$node_list
```

We'll now construct an ensemble learner using a handful of popular machine learning algorithms:

```
# SL learners used for continuous data (the nuisance parameter Z)
enet_contin_learner <- Lrnr_glmnet$new(
  alpha = 0.5, family = "gaussian", nfolds = 3
)
lasso_contin_learner <- Lrnr_glmnet$new(
  alpha = 1, family = "gaussian", nfolds = 3
)
fglm_contin_learner <- Lrnr_glm_fast$new(family = gaussian())
mean_learner <- Lrnr_mean$new()
contin_learner_lib <- Stack$new(
  enet_contin_learner, lasso_contin_learner, fglm_contin_learner, mean_learner
)
sl_contin_learner <- Lrnr_sl$new(learners = contin_learner_lib)

# SL learners used for binary data (nuisance parameters G and E in this case)
enet_binary_learner <- Lrnr_glmnet$new(
  alpha = 0.5, family = "binomial", nfolds = 3
)
lasso_binary_learner <- Lrnr_glmnet$new(
  alpha = 1, family = "binomial", nfolds = 3
)
fglm_binary_learner <- Lrnr_glm_fast$new(family = binomial())
binary_learner_lib <- Stack$new(
  enet_binary_learner, lasso_binary_learner, fglm_binary_learner, mean_learner
)
sl_binary_learner <- Lrnr_sl$new(learners = binary_learner_lib)

# create list for treatment and outcome mechanism regressions
learner_list <- list(
  Y = sl_contin_learner,
  A = sl_binary_learner
)
```

10.5.1 Targeted Estimation of the Natural Indirect Effect

We demonstrate calculation of the NIE below, starting by instantiating a "Spec" object that encodes exactly which learners to use for the nuisance parameters $e(A \mid Z, W)$ and Ψ_Z. We then pass our Spec object to the `tmle3` function, alongside the data, the node list (created above), and a learner list indicating which machine learning algorithms to use for estimating the nuisance parameters based on A and Y.

```
tmle_spec_NIE <- tmle_NIE(
  e_learners = Lrnr_cv$new(lasso_binary_learner, full_fit = TRUE),
  psi_Z_learners = Lrnr_cv$new(lasso_contin_learner, full_fit = TRUE),
  max_iter = 1
)
washb_NIE <- tmle3(
  tmle_spec_NIE, washb_data, node_list, learner_list
)
washb_NIE
A tmle3_Fit that took 1 step(s)
   type                  param  init_est  tmle_est       se     lower    upper
1:  NIE NIE[Y_{A=1} - Y_{A=0}] 0.0022912 0.0026608 0.044295 -0.084156 0.089478
   psi_transformed lower_transformed upper_transformed
1:       0.0026608         -0.084156          0.089478
```

Based on the output, we conclude that the indirect effect of the treatment through the mediators (sex, month, aged) is 0.00266.

10.5.2 Targeted Estimation of the Natural Direct Effect

An analogous procedure applies for estimation of the NDE, only replacing the Spec object for the NIE with `tmle_spec_NDE` to define learners for the NDE nuisance parameters:

```
tmle_spec_NDE <- tmle_NDE(
  e_learners = Lrnr_cv$new(lasso_binary_learner, full_fit = TRUE),
  psi_Z_learners = Lrnr_cv$new(lasso_contin_learner, full_fit = TRUE),
  max_iter = 1
)
washb_NDE <- tmle3(
  tmle_spec_NDE, washb_data, node_list, learner_list
)
washb_NDE
A tmle3_Fit that took 1 step(s)
   type                  param init_est tmle_est      se   lower   upper
1:  NDE NDE[Y_{A=1} - Y_{A=0}] 0.012983 0.012983 0.10285 -0.1886 0.21457
   psi_transformed lower_transformed upper_transformed
1:        0.012983           -0.1886           0.21457
```

From this, we can draw the conclusion that the direct effect of the treatment (through all paths not involving the mediators (sex, month, aged)) is 0.01298. Note that, together, the estimates of the natural direct and indirect effects approximately recover the *average* *treatment effect*, that is, based on these estimates of the NDE and NIE, the ATE is roughly 0.01564.

10.5.3 Targeted Estimation of the Population Intervention Direct Effect

As previously noted, the assumptions underlying the natural direct and indirect effects may be challenging to justify; moreover, the effect definitions themselves depend on the application of a static intervention to the treatment, sharply limiting their flexibility. When considering binary treatments, incremental propensity score shifts provide an alternative class of flexible, stochastic interventions. We'll now consider estimating the PIDE with an IPSI that modulates the odds of receiving treatment by $\delta = 3$. Such an intervention may be interpreted (hypothetically) as the effect of a design that encourages study participants to opt in to receiving the treatment, thus increasing their relative odds of receiving said treatment. To exemplify our approach, we postulate a motivational intervention that *triples the odds* (i.e., $\delta = 3$) of receiving the treatment for each individual:

```r
# set the IPSI multiplicative shift
delta_ipsi <- 3

# instantiate tmle3 spec for stochastic mediation
tmle_spec_pie_decomp <- tmle_medshift(
  delta = delta_ipsi,
  e_learners = Lrnr_cv$new(lasso_binary_learner, full_fit = TRUE),
  phi_learners = Lrnr_cv$new(lasso_contin_learner, full_fit = TRUE)
)

# compute the TML estimate
washb_pie_decomp <- tmle3(
  tmle_spec_pie_decomp, washb_data, node_list, learner_list
)
washb_pie_decomp

# get the PIDE
washb_pie_decomp$summary$tmle_est - mean(washb_data[, get(node_list$Y)])
```

Recall that, based on the decomposition outlined previously, the PIDE may be denoted $\beta_{\text{PIDE}}(\delta) = \theta_0(\delta) - \mathbb{E}Y$. Thus, an estimator of the PIDE, $\hat{\beta}_{\text{PIDE}}(\delta)$ may be expressed as a composition of estimators of its constituent parameters:

$$\hat{\beta}_{\text{PIDE}}(\delta) = \hat{\theta}(\delta) - \frac{1}{n}\sum_{i=1}^{n} Y_i.$$

10.6 Exercises

10.6.1 Review of Key Concepts

Exercise 10.1. Examine the WASH Benefits dataset and choose a different set of potential mediators of the effect of the treatment on weight-for-height Z-score. Using this newly chosen set of mediators (or single mediator), estimate the natural direct and indirect effects. Provide an interpretation of these estimates.

Solution. Forthcoming

Exercise 10.2. Assess whether additivity of the natural direct and indirect effects holds. Using the natural direct and indirect effects estimated above, does their sum recover the ATE?

Solution. Forthcoming

Exercise 10.3. Evaluate whether the assumptions required for identification of the natural direct and indirect effects are plausible in the WASH Benefits example. In particular, position this evaluation in terms of empirical diagnostics of treatment and mediator positivity.

Solution. Forthcoming

10.6.2 The Ideas in Action

Forthcoming.

11

R6 *Class System Primer*

A central goal of the Targeted Learning statistical paradigm is to estimate scientifically relevant parameters in realistic (usually nonparametric) models.

The `tlverse` is designed using basic OOP principles and the `R6` OOP framework. While we've tried to make it easy to use the `tlverse` packages without worrying much about OOP, it is helpful to have some intuition about how the `tlverse` is structured. Here, we briefly outline some key concepts from OOP. Readers familiar with OOP basics are invited to skip this section.

11.1 Classes, Fields, and Methods

The key concept of OOP is that of an object, a collection of data and functions that corresponds to some conceptual unit. Objects have two main types of elements:

1. *fields*, which can be thought of as nouns, are information about an object, and
2. *methods*, which can be thought of as verbs, are actions an object can perform.

Objects are members of classes, which define what those specific fields and methods are. Classes can inherit elements from other classes (sometimes called base classes) – accordingly, classes that are similar, but not exactly the same, can share some parts of their definitions.

Many different implementations of OOP exist, with variations in how these concepts are implemented and used. R has several different implementations, including `S3`, `S4`, reference classes, and `R6`. The `tlverse` uses the `R6` implementation. In `R6`, methods and fields of a class object are accessed using the `$` operator. For a more thorough introduction to `R`'s various OOP systems, see `http://adv-r.had.co.nz/OO-essentials.html`, from Hadley Wickham's *Advanced R* (Wickham, 2014).

11.2 Object Oriented Programming: Python and R

OO concepts (classes with inherentence) were baked into Python from the first published version (version 0.9 in 1991). In contrast, R gets its OO "approach" from its predecessor, S, first released in 1976. For the first 15 years, S had no support for classes, then, suddenly, S got two OO frameworks bolted on in rapid succession: informal classes with S3 in 1991, and formal classes with S4 in 1998. This process continues, with new OO frameworks being periodically released, to try to improve the lackluster OO support in R, with reference classes (R5, 2010) and R6 (2014). Of these, R6 behaves most like Python classes (and also most like OOP focused languages like C++ and Java), including having method definitions be part of class definitions, and allowing objects to be modified by reference.

Bibliography

Avin, C., Shpitser, I., and Pearl, J. (2005). Identifiability of path-specific effects. In *IJCAI International Joint Conference on Artificial Intelligence*, pages 357–363.

Baker, M. (2016). Is there a reproducibility crisis? a nature survey lifts the lid on how researchers view the crisis rocking science and what they think will help. *Nature*, 533(7604):452–455.

Bembom, O. and van der Laan, M. J. (2007). A practical illustration of the importance of realistic individualized treatment rules in causal inference. *Electron J Stat*, 1:574–596.

Bengtsson, H. (2021). A unifying framework for parallel and distributed processing in r using futures. *The R Journal*.

Benkeser, D. and Ran, J. (2021). Nonparametric inference for interventional effects with multiple mediators. *Journal of Causal Inference*.

Benkeser, D. and van der Laan, M. J. (2016). The highly adaptive lasso estimator. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Buckheit, J. B. and Donoho, D. L. (1995). Wavelab and reproducible research. In *Wavelets and statistics*, pages 55–81. Springer.

Chakraborty, B. and Moodie, E. E. (2013). *Statistical Methods for Dynamic Treatment Regimes: Reinforcement Learning, Causal Inference, and Personalized Medicine (Statistics for Biology and Health)*. Springer.

Chernozhukov, V., Chetverikov, D., Demirer, M., Duflo, E., Hansen, C., Newey, W., and Robins, J. (2018). Double/debiased machine learning for treatment and structural parameters. *The Econometrics Journal*, 21(1).

Coyle, J. R. and Hejazi, N. S. (2018). origami: A generalized framework for cross-validation in r. *Journal of Open Source Software*, 3(21).

Coyle, J. R., Hejazi, N. S., Malenica, I., Phillips, R. V., Arnold, B. F., Mertens, A., Benjamin-Chung, J., Cai, W., Dayal, S., Colford Jr., J. M., Hubbard, A. E., and van der Laan, M. J. (2021). Targeting Learning: Robust statistics for reproducible research. *arXiv*.

Coyle, J. R., Hejazi, N. S., and van der Laan, M. J. (2020). *hal9001: The scalable highly adaptive lasso*. R package version 0.2.7.

Dawid, A. P. (2000). Causal inference without counterfactuals. *Journal of the American Statistical Association*, 95(450):407–424.

Díaz, I. and Hejazi, N. S. (2020). Causal mediation analysis for stochastic interventions. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 82(3):661–683.

Díaz, I., Hejazi, N. S., Rudolph, K. E., and van der Laan, M. J. (2020). Non-parametric efficient causal mediation with intermediate confounders. *Biometrika*.

Díaz, I. and van der Laan, M. J. (2011). Super learner based conditional density estimation with application to marginal structural models. *The International Journal of Biostatistics*, 7(1):1–20.

Díaz, I. and van der Laan, M. J. (2012). Population intervention causal effects based on stochastic interventions. *Biometrics*, 68(2):541–549.

Díaz, I. and van der Laan, M. J. (2018). Stochastic treatment regimes. In *Targeted Learning in Data Science: Causal Inference for Complex Longitudinal Studies*, pages 167–180. Springer Science & Business Media.

Didelez, V., Dawid, P., and Geneletti, S. (2006). Direct and indirect effects of sequential treatments. In *Proceedings of the 22nd Annual Conference on Uncertainty in Artificial Intelligence*, pages 138–146.

Donoho, D. (2017). 50 years of data science. *Journal of Computational and Graphical Statistics*, 26(4):745–766.

Dudoit, S. and van der Laan, M. J. (2005). Asymptotics of cross-validated risk estimation in estimator selection and performance assessment. *Statistical Methodology*, 2(2):131–154.

Haneuse, S. and Rotnitzky, A. (2013). Estimation of the effect of interventions that modify the received treatment. *Statistics in medicine*, 32(30):5260–5277.

Hejazi, N. S. (2021). *Semiparametric statistical methods for causal inference with stochastic treatment regimes*. PhD thesis, University of California, Berkeley.

Hejazi, N. S., Benkeser, D. C., and van der Laan, M. J. (2020a). *haldensify: Highly adaptive lasso conditional density estimation*. R package version 0.0.5.

Hejazi, N. S., Coyle, J. R., and van der Laan, M. J. (2020b). hal9001: Scalable highly adaptive lasso regression in R. *Journal of Open Source Software*.

Hejazi, N. S., Rudolph, K. E., van der Laan, M. J., and Díaz, I. (2021). Nonparametric causal mediation analysis for stochastic interventional (in)direct effects. revision invited at *Biostatistics*.

Hejazi, N. S., van der Laan, M. J., Janes, H. E., Gilbert, P. B., and Benkeser, D. C. (2020c). Efficient nonparametric inference on the effects of stochastic interventions under two-phase sampling, with applications to vaccine efficacy trials. *Biometrics*.

Imai, K., Keele, L., and Yamamoto, T. (2010). Identification, inference and sensitivity analysis for causal mediation effects. *Statistical science*, pages 51–71.

Imbens, G. W. and Rubin, D. B. (2015). *Causal inference in statistics, Social, and Biomedical Sciences*. Cambridge University Press.

Kennedy, E. H. (2019). Nonparametric causal effects based on incremental propensity score interventions. *Journal of the American Statistical Association*, 114(526):645–656.

Lok, J. J. (2016). Defining and estimating causal direct and indirect effects when setting the mediator to specific values is not feasible. *Statistics in Medicine*, 35(22):4008–4020.

Luedtke, A. and van der Laan, M. J. (2016a). Super-learning of an optimal dynamic treatment rule. *International Journal of Biostatistics*, 12(1):305–332. PMID: 27227726.

Luedtke, A. R. and van der Laan, M. J. (2016b). Optimal individualized treatments in resource-limited settings. *International Journal of Biostatisics*, 12(1):283–303.

Montoya, L., Skeem, J., van der Laan, M., and Petersen, M. (2021a). Performance and application of estimators for the value of an optimal dynamic treatment rule.

Montoya, L., van der Laan, M., Luedtke, A., Skeem, J., Coyle, J., and Petersen, M. (2021b). The optimal dynamic treatment rule superlearner: Considerations, performance, and application.

Munafò, M. R., Nosek, B. A., Bishop, D. V., Button, K. S., Chambers, C. D., Du Sert, N. P., Simonsohn, U., Wagenmakers, E.-J., Ware, J. J., and Ioannidis, J. P. (2017). A manifesto for reproducible science. *Nature Human Behaviour*, 1(1):0021.

Murphy, S. A. (2003). Optimal dynamic treatment regimes. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 65(2):331–355.

Nature Editorial (Anonymous) (2015a). How scientists fool themselves — and how they can stop. *Nature*, 526(7572).

Nature Editorial (Anonymous) (2015b). Let's think about cognitive bias. *Nature*, 526(7572).

Neyman, J. (1938). Contribution to the theory of sampling human populations. *Journal of the American Statistical Association*, 33(201):101–116.

Nguyen, T. Q., Schmid, I., and Stuart, E. A. (2019). Clarifying causal mediation analysis for the applied researcher: Defining effects based on what we want to learn. *arXiv preprint arXiv:1904.08515*.

Nosek, B. A., Ebersole, C. R., DeHaven, A. C., and Mellor, D. T. (2018). The preregistration revolution. *Proceedings of the National Academy of Sciences*, 115(11):2600–2606.

Pearl, J. (1995). Causal diagrams for empirical research. *Biometrika*, 82(4):669–688.

Pearl, J. (2001). Direct and indirect effects. *arXiv preprint arXiv:1301.2300*.

Pearl, J. (2009). *Causality: Models, Reasoning, and Inference*. Cambridge University Press.

Pearl, J. (2010). Brief report: On the consistency rule in causal inference: "axiom, definition, assumption, or theorem?". *Epidemiology*, pages 872–875.

Peng, R. (2015). The reproducibility crisis in science: A statistical counterattack. *Significance*, 12(3):30–32.

Petersen, M. L., Sinisi, S. E., and van der Laan, M. J. (2006). Estimation of direct causal effects. *Epidemiology*, pages 276–284.

Polley, E. C. and van der Laan, M. J. (2010). Super learner in prediction. Technical report, Division of Biostatistics, University of California, Berkeley.

Popper, K. (1934). *The Logic of Scientific Discovery*. Routledge.

Pullenayegum, E. M., Platt, R. W., Barwick, M., Feldman, B. M., Offringa, M., and Thabane, L. (2016). Knowledge translation in biostatistics: a survey of current practices, preferences, and barriers to the dissemination and uptake of new statistical methods. *Statistics in medicine*, 35(6):805–818.

R Core Team (2021). R: A language and environment for statistical computing.

Robins, J. (1986a). A new approach to causal inference in mortality studies with a sustained exposure period—application to control of the healthy worker survivor effect. *Mathematical Modelling*, 7(9):1393 – 1512.

Robins, J. and Rotnitzky, A. (2014). Discussion of "dynamic treatment regimes: Technical challenges and applications". *Electron. J. Statist.*, 8(1):1273–1289.

Robins, J. M. (1986b). A new approach to causal inference in mortality studies with sustained exposure periods — application to control of the healthy worker survivor effect. *Mathematical Modelling*, 7:1393–1512.

Robins, J. M. (2004). Optimal structural nested models for optimal sequential decisions. In *Proceedings of the Second Seattle Symposium in Biostatistics: Analysis of Correlated Data*, pages 189–326. Springer New York.

Robins, J. M. and Greenland, S. (1992). Identifiability and exchangeability for direct and indirect effects. *Epidemiology*, pages 143–155.

Robins, J. M. and Richardson, T. S. (2010). Alternative graphical causal models and the identification of direct effects. *Causality and psychopathology: Finding the determinants of disorders and their cures*, pages 103–158.

Rubin, D. B. (1978). Bayesian inference for causal effects: The role of randomization. *The Annals of statistics*, pages 34–58.

Rubin, D. B. (1980). Randomization analysis of experimental data: The fisher randomization test comment. *Journal of the American Statistical Association*, 75(371):591–593.

Rubin, D. B. (2005). Causal inference using potential outcomes: Design, modeling, decisions. *Journal of the American Statistical Association*, 100(469):322–331.

Rudolph, K. E., Sofrygin, O., Zheng, W., and van der Laan, M. J. (2017). Robust and flexible estimation of stochastic mediation effects: a proposed method and example in a randomized trial setting. *Epidemiologic Methods*, 7(1).

Spirtes, P., Glymour, C. N., Scheines, R., Heckerman, D., Meek, C., Cooper, G., and Richardson, T. (2000). *Causation, prediction, and search*. MIT press.

Stark, P. B. and Saltelli, A. (2018). Cargo-cult statistics and scientific crisis. *Significance*, 15(4):40–43.

Stock, J. H. (1989). Nonparametric policy analysis. *Journal of the American Statistical Association*, 84(406):567–575.

Stromberg, A. et al. (2004). Why write statistical software? the case of robust statistical methods. *Journal of Statistical Software*, 10(5):1–8.

Sutton, R. S., Barto, A. G., et al. (1998). *Introduction to reinforcement learning*, volume 135. MIT press Cambridge.

Szucs, D. and Ioannidis, J. (2017). When null hypothesis significance testing is unsuitable for research: a reassessment. *Frontiers in Human Neuroscience*, 11:390.

Tchetgen Tchetgen, E. J. (2013). Inverse odds ratio-weighted estimation for causal mediation analysis. *Statistics in Medicine*, 32(26):4567–4580.

Tchetgen Tchetgen, E. J. and Shpitser, I. (2012). Semiparametric theory for causal mediation analysis: efficiency bounds, multiple robustness, and sensitivity analysis. *Annals of Statistics*, 40(3):1816–1845.

Textor, J., Hardt, J., and Knüppel, S. (2011). Dagitty: a graphical tool for analyzing causal diagrams. *Epidemiology*, 22(5):745.

Tofail, F., Fernald, L. C., Das, K. K., Rahman, M., Ahmed, T., Jannat, K. K., Unicomb, L., Arnold, B. F., Ashraf, S., Winch, P. J., et al. (2018). Effect of water quality, sanitation, hand washing, and nutritional interventions on child development in rural bangladesh (wash benefits bangladesh): a cluster-randomised controlled trial. *The Lancet Child & Adolescent Health*, 2(4):255–268.

Tukey, J. W. (1962). The future of data analysis. *The Annals of Mathematical Statistics*, 33(1):1–67.

van der Laan, M. J. and Dudoit, S. (2003). Unified cross-validation methodology for selection among estimators and a general cross-validated adaptive epsilon-net estimator: Finite sample oracle inequalities and examples. Technical report, Division of Biostatistics, University of California, Berkeley.

van der Laan, M. J., Dudoit, S., and Keles, S. (2004). Asymptotic optimality of likelihood-based cross-validation. *Statistical Applications in Genetics and Molecular Biology*, 3(1):1–23.

van der Laan, M. J. and Luedtke, A. (2015). Targeted learning of the mean outcome under an optimal dynamic treatment rule. *Journal of Causal Inference*, 3(1):61–95. PMCID: PMC4517487.

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

van der Laan, M. J. and Rose, S. (2011). *Targeted learning: causal inference for observational and experimental data*. Springer Science & Business Media.

van der Laan, M. J. and Rose, S. (2018). *Targeted Learning in Data Science: Causal Inference for Complex Longitudinal Studies*. Springer Science & Business Media.

van der Laan, M. J. and Starmans, R. J. (2014). Entering the era of data science: Targeted learning and the integration of statistics and computational data analysis. *Advances in Statistics*, 2014.

Van der Vaart, A. W., Dudoit, S., and van der Laan, M. J. (2006). Oracle inequalities for multi-fold cross validation. *Statistics & Decisions*, 24(3):351–371.

VanderWeele, T. (2015). *Explanation in causal inference: methods for mediation and interaction*. Oxford University Press.

VanderWeele, T. J., Vansteelandt, S., and Robins, J. M. (2014). Effect decomposition in the presence of an exposure-induced mediator-outcome confounder. *Epidemiology*, 25(2):300.

Vansteelandt, S., Bekaert, M., and Lange, T. (2012). Imputation strategies for the estimation of natural direct and indirect effects. *Epidemiologic Methods*, 1(1):131–158.

Vansteelandt, S. and Daniel, R. M. (2017). Interventional effects for mediation analysis with multiple mediators. *Epidemiology*, 28(2):258.

Vansteelandt, S. and VanderWeele, T. J. (2012). Natural direct and indirect effects on the exposed: effect decomposition under weaker assumptions. *Biometrics*, 68(4):1019–1027.

Wickham, H. (2014). *Advanced r*. Chapman and Hall/CRC.

Wright, S. (1934). The method of path coefficients. *The Annals of Mathematical Statistics*, 5(3):161–215.

Young, J. G., Hernán, M. A., and Robins, J. M. (2014). Identification, estimation and approximation of risk under interventions that depend on the natural value of treatment using observational data. *Epidemiologic methods*, 3(1):1–19.

Zhang, B., A Tsiatis, A., Davidian, M., Zhang, M., and Laber, E. (2016). Estimating optimal treatment regimes from a classification perspective. *Stat*, 5(1):278–278.

Zhao, Y., Zeng, D., Rush, A. J., and Kosorok, M. R. (2012). Estimating individualized treatment rules using outcome weighted learning. *Journal of the American Statistical Association*, 107(499):1106–1118. PMID: 23630406.

Zheng, W. and van der Laan, M. J. (2010). Asymptotic Theory for Cross-validated Targeted Maximum Likelihood Estimation. *U.C. Berkeley Division of Biostatistics Working Paper Series.*

Zheng, W. and van der Laan, M. J. (2011). Cross-validated targeted minimum-loss-based estimation. In *Targeted Learning*, pages 459–474. Springer.

Zheng, W. and van der Laan, M. J. (2012). Targeted maximum likelihood estimation of natural direct effects. *International Journal of Biostatistics*, 8(1).