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 Figures

Welcome

Targeted Learning in `R`*: Causal Data Science with the* `tlverse` *Software Ecosystem* is an fully reproducible, open source, electronic handbook for applying Targeted Learning methodology in practice using the software stack provided by the `tlverse` ecosystem. This work is a draft phase and is publicly available to solicit input from the community. To view or contribute, visit the GitHub repository.

Outline

The contents of this handbook are meant to serve as a reference guide for both applied research and for the teaching of short courses illustrating successful applications of the Targeted Learning statistical paradigm. Each section introduces a set of distinct causal inference questions, often motivated by a case study, alongside statistical methodology and open source software for assessing the scientific (causal) claim of interest. The set of materials currently includes

- Motivation: Why we need a statistical revolution
- The Roadmap and introductory case study: the WASH Benefits Bangladesh dataset
- 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

This book does **not** focus on providing in-depth technically sophisticated descriptions of modern statistical methodology or recent advancements in Targeted Learning. Instead, the goal is to convey key details of these state-of-the-art statistical techniques in a manner that is clear, complete, and intuitive, while simultaneously avoiding the cognitive burden carried by extraneous details (e.g., mathematically niche theoretical arguments). Our aim is for the presentations herein to serve as a coherent reference for researchers – applied methodologists and domain specialists alike – that empower them to deploy the central statistical tools of Targeted Learning in a manner efficient for their scientific pursuits. For a mathematically sophisticated treatment of some of these topics, inclusive of in-depth technical details, in the field of Targeted Learning, the interested reader is invited to consult van der Laan and Rose [2011] and van der Laan and Rose [2018], among numerous other works, as appropriate. The primary literature in causal inference, machine learning, and non/semi-parametric statistical theory include many of the most recent advances in Targeted Learning and related areas. For background in causal inference, Hernán and Robins [2022] serves as an introductory modern reference.

About the authors

Mark van der Laan

Mark van der Laan is Professor of Biostatistics and of Statistics at UC Berkeley and co-director of UC Berkeley's Center for Targeted Machine Learning and Causal Inference. 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. Since mid-2017, 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 2017, primarily under the supervision of Alan Hubbard.

Nima Hejazi

Nima Hejazi is an Assistant Professor of Biostatistics at the Harvard T.H. Chan School of Public Health. He obtained his PhD in Biostatistics at UC Berkeley, working with Mark van der Laan and Alan Hubbard, and held an NSF Mathematical Sciences Postdoctoral Research Fellowship afterwards. Nima's research interests blend causal inference, machine learning, non- and semi-parametric inference, and computational statistics, with areas of recent emphasis having included causal mediation analysis; efficient estimation under biased, outcome-dependent sampling designs; and sieve estimation for causal machine learning. His methodological work is motivated principally by scientific collaborations in clinical trials and observational studies of infectious diseases, in infectious disease epidemiology, and in computational biology. Nima is also passionate about high-performance statistical computing and open source software design for applied statistics and statistical data science.

Ivana Malenica

Ivana Malenica is a Postdoctoral Researcher in the Department of Statistics at Harvard and a Wojcicki and Troper Data Science Fellow at the Harvard Data Science Initiative. She obtained her PhD in Biostatistics at UC Berkeley working with Mark van der Laan, where she was a Berkeley Institute for Data Science and a NIH Biomedical Big Data Fellow. Her research interests span non/semi-parametric theory, causal inference and machine learning, with emphasis on personalized health and dependent settings. Most of her current work involves causal inference with time and network dependence, online learning, optimal individualized treatment, reinforcement learning, 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. As a student of targeted learning, Rachael integrates causal inference, machine learning, and statistical theory to answer causal questions with statistical confidence. She is motivated by issues arising in healthcare, and is especially interested in clinical algorithm frameworks and guidelines. Related to to this, she is also interested in experimental design; human-computer interaction; statistical analysis pre-specification, automation, and reproducibility; and open-source software.

Alan Hubbard

Alan Hubbard is Professor of Biostatistics at UC Berkeley, current chair of the Division of Biostatistics of the UC Berkeley School of Public Health, head of the data analytics core of UC Berkeley's SuperFund research program, and co-director of UC Berkeley's Center for Targeted Machine Learning and Causal Inference. 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.

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, modern introduction to causal inference, we recommend

- Miguel A. Hernán and James M. Robins' *Causal Inference: What If* (2022)
- Jason A. Roy's *A Crash Course in Causality: Inferring Causal Effects from Observational Data* on Coursera

Feel free to suggest a resource!

Want to help?

Any feedback on the book is very welcome. Feel free to open an issue, or to make a Pull Request if you spot a typo.

Introduction

> "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 is greater than ever. Corroboration moves science forward, yet there is growing alarm that results cannot be reproduced or validated, suggesting that many discoveries may be false or less robust than initially claimed [Baker, 2016]. A failure to meet this need will result in further declines in the rate of scientific progress, tarnishing of the reputation of the scientific enterprise as a whole, and erosion of 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 into statistical data analysis efforts in the form of improper model selection. All procedures for estimation and hypothesis testing are derived based on a choice of a statistical model; thus, obtaining valid estimates and statistical inference relies critically on the chosen model containing an accurate representation of the process that generated the data.

Consider, for example, a hypothetical study in which a treatment is assigned to a group of patients – was the treatment assigned randomly, or were characteristics of the individuals (i.e., "baseline covariates") taken into account in making the treatment assignment decision? What's more, in light of patient characteristics and heterogeneity in clinician decision-making, are patients assigned to the treatment arm uniformly receiving

the same treatment? Any such knowledge can – indeed, *must* – be incorporated into the choice of statistical model. Alternatively, the data could arise from an observational study (or "quasi-experiment"), in which there is no (or very limited) control over the treatment assignment mechanism. In such cases, available knowledge about the data-generating process (DGP) is even more limited. In these situations, 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).

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 when these choices often result in different answers to the same research question. As opposed to its original purpose of safeguarding the scientific process – by providing formal techniques for evaluating the veracity of a claim using properly designed experiments and data collection procedures – Statistics is increasingly often used "to aid and abet weak science, a role it can perform well when used mechanically or ritual[istically]" [Stark and Saltelli, 2018]. The current trend in deriving scientific discoveries by way of abusing statistical methods helps to explain the modern 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 now spreading rapidly through his students and colleagues who have greatly enriched the field, the aptly named

"Targeted Learning" paradigm emphasizes a focus upon (i.e., "targeting of") the scientific question motivating a given study or dataset. The philosophy of Targeted Learning runs counter to the current cultural problem of "convenience statistics," which opens the door to biased estimation, misleading data analytic results, and erroneous discoveries. Targeted Learning (TL) embraces the fundamentals that formalized the field of Statistics, notably including the dual notions that a statistical model must represent real knowledge about the data-generating experiment and that a target parameter (a particular feature of the data-generating probability distribution) represents what we seek to learn from the data [van der Laan and Starmans, 2014]. In this way, TL defines a ground truth and establishes a principled standard for inference, thereby curtailing opportunities for our all-too-human biases (e.g., hindsight bias, confirmation bias, and outcome bias) to infiltrate our efforts at objective data analysis.

> "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]

Inspired loosely by R.A. Fisher's influential classic *Statistical Methods for Research Workers* [Fisher, 1946], this handbook aims to provide practical training for students, researchers, industry professionals, and academicians in the sciences (broadly considered – whether biological, physical, economic, or social), medicine and public health, statistics, and numerous other allied disciplines, equipping them with the necessary knowledge and skills to utilize the methodological developments of TL. The Targeted Learning paradigm encompasses a principled set of techniques, united by a single philosophy, for developing answers to queries with confidence, utilizing advances in causal inference, state-of-the-art non/semi-parametric statistical theory, and machine learning — so that each and every data analysis is realistic, reflecting appropriately what is known (and unknown) about the process that generated the data, while remaining fully compatible 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 access the published results of a given scientific investigation. We concur with the view put forth by Buckheit and Donoho [1995] that "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," 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.

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 packages for the `R` language and environment for statistical computing [R Core Team, 2021], was developed to fulfill this need for the TL methodological framework. Not only does this suite of software tools facilitate computationally reproducible and efficient analyses, it is also a tool for TL education. Rather than focusing on implementing a specific estimator or a small set of related estimators, the design paradigm of the `tlverse` ecosystem focuses 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 to be used in conjunction with each other and used cohesively as building blocks for formulating sophisticated statistical analyses. For an introduction to the TL framework, we recommend Coyle et al. [2021]'s recent review paper.

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 this toolbox for applying the TL statistical methodology, which will translate to real-world causal analyses. Some preliminaries are required prior to this learning endeavor – for this, we provide a list of recommended learning resources.

Part I

Part 1: Preliminaries

1

About the `tlverse`

1.1 What is the `tlverse`?

The `tlverse` is a new framework for 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 design philosophy, grammar, and core set of data structures. The `tlverse` aims to provide tools both for building Targeted Learning-based data analyses and for implementing novel, state-of-the-art Targeted Learning methods.

1.2 Anatomy of the `tlverse`

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

1. Flexibly learning particular components of the data-generating distribution often through machine learning (e.g., Super Learning), resulting in *initial estimates* of nuisance parameters.

2. Use of a carefully constructed parametric model-based update, via maximum likelihood estimation (i.e., MLE), incorporating the initial estimates produced by the prior step to produce a TML estimator.

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

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

- sl3: Modern Super Machine Learning
  - *What?* A modern object-oriented 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 both the Super Learner ensemble modeling algorithm and in the construction of TML 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 novel Targeted Learning estimators as they reach 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 rely upon the core machinery provided by sl3 and tmle3. 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 based on Modified Treatment Policies in the tlverse
  - *What?* Stochastic shift interventions for evaluating changes in 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.

1.3 Primer on the R6 Class System

The tlverse is designed using basic object oriented programming (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.

1.3.1 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].

Object Oriented Programming: Python and R

OO concepts (classes with inheritance) 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.

2

Software Setup

2.1 Setting up 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.

2.1.1 Windows

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.

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.

2.1.2 macOS / Mac OS X

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.

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.

2.1.3 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 the most recent version of R.
- 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.

2.2 Install `tlverse`

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.

3

Example Datasets

3.1 WASH Benefits Bangladesh Study

The example 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. handwashing with soap;
4. combined water, sanitation, and handwashing;
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 under registration number 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 instructional purposes, we start by treating the data as independent and identically distributed (i.i.d.) random draws from a large target population. We could account for the clustering of the data (within sampled geographic units), but, we avoid these details in this handbook for the sake of clarity of illustration. Modifications of TL methodology for biased samples, repeated measures, and related complications, are readily 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 (potential baseline confounders), 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	1.0000	3	15	0	
character	fracode	0	1.0000	2	6	0	
character	sex	0	1.0000	4	6	0	
character	momedu	0	1.0000	12	15	0	
character	hfiacat	0	1.0000	11	24	0	
numeric	whz	0	1.0000	NA	NA	NA	
numeric	month	0	1.0000	NA	NA	NA	
numeric	aged	0	1.0000	NA	NA	NA	
numeric	momage	18	0.9962	NA	NA	NA	
numeric	momheight	31	0.9934	NA	NA	NA	
numeric	Nlt18	0	1.0000	NA	NA	NA	
numeric	Ncomp	0	1.0000	NA	NA	NA	
numeric	watmin	0	1.0000	NA	NA	NA	
numeric	elec	0	1.0000	NA	NA	NA	
numeric	floor	0	1.0000	NA	NA	NA	
numeric	walls	0	1.0000	NA	NA	NA	
numeric	roof	0	1.0000	NA	NA	NA	
numeric	asset_wardrobe	0	1.0000	NA	NA	NA	
numeric	asset_table	0	1.0000	NA	NA	NA	
numeric	asset_chair	0	1.0000	NA	NA	NA	
numeric	asset_khat	0	1.0000	NA	NA	NA	
numeric	asset_chouki	0	1.0000	NA	NA	NA	
numeric	asset_tv	0	1.0000	NA	NA	NA	
numeric	asset_refrig	0	1.0000	NA	NA	NA	
numeric	asset_bike	0	1.0000	NA	NA	NA	
numeric	asset_moto	0	1.0000	NA	NA	NA	
numeric	asset_sewmach	0	1.0000	NA	NA	NA	
numeric	asset_mobile	0	1.0000	NA	NA	NA	

A convenient summary of the relevant variables appears above, complete with a sparkline visualizations describing the marginal characteristics of each covariate. Note that the *asset* variables reflect socioeconomic 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.

Part II

Part 2: Foundations

4

Learning from Data: A Roadmap

Learning Objectives

1. Translate scientific questions to statistical questions.
2. Define a statistical model based on knowledge about the scientific experiment or study 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 alongside their implications: independent and identically distributed (i.i.d.), consistency, no unmeasured confounding, interference, positivity.

Introduction

The roadmap of statistical learning is concerned with the process of translating real-world scientific questions to mathematical formalisms necessary for formulating relevant statistical inference problems. This involves viewing data as a random variable (complete with its own underlying probability distribution), incorporating scientific knowledge into the choice of statistical model, selecting a statistical target parameter that represents an answer to the scientific question of interest, and developing efficient estimators of the statistical estimand.

4.1 The Roadmap

The roadmap is a six-stage process:

1. Define the data as a random variable with a probability distribution, $O \sim P_0$
2. Specify the statistical model $\mathcal{M}$ realistically, such that $P_0 \in \mathcal{M}$
3. Translate the scientific question of interest into a statistical target parameter Ψ and establish the target population
4. Choose an estimator $\hat{\Psi}$ for Ψ under realistic $\mathcal{M}$
5. Construct a measure of uncertainty for the estimate $\hat{\Psi}(P_n)$
6. Make substantive conclusion

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

The dataset we are confronted with is the collection of the results of a scientific (or natural) experiment. We can view the data as a *random variable*; that is, if the same experiment were to be repeated, we should expect to see a different realization of the data generated by the same underlying law governing the experiment in question. In particular, if the experiment were repeated many times, the underlying probability distribution generating the data, P_0, would be revealed. The observed data on a single unit, O, may be thought of as being drawn from this probability distribution P_0. Most often, we have n *independent and identically distributed* (i.i.d.) observations of the random variable O in our dataset. Then, 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. There are a number of techniques for handling non-i.i.d. data, including establishing conditional independence, such that conditional on some variable (e.g., subject ID for repeated measures data) the i.i.d. assumption holds, and incorporating inferential corrections for repeated or clustered observations, to name but a few.

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 of variables, say W, to be the proportion of observations that

belong in W. That is,

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

In order to understand the scope for learning from a particular dataset, we next need to ask *"What do we know about the process that led to the data's generation?"* 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 the set of all possible probability distributions that could describe the process by which our observed data have been generated, appropriately constrained by background scientific knowledge. Often, $\mathcal{M}$ is necessarily very large (i.e., non-parametric), reflecting the fact that statistical knowledge about P_0 is limited.

If P_0 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 O has a Normal distribution with mean μ and variance σ^2. More generally, a parametric model may be defined as

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

which describes a constrained statistical model consisting of all distributions P_θ that are indexed by some finite, d-dimensional parameter θ.

The assumption that P_0 has a specific, parametric form is made quite commonly. Unfortunately, this is even the case when such assumptions are not supported by domain knowledge about the data-generating process. This practice of oversimplification in the current, and traditional, culture of statistical data analysis typically complicates or entirely thwarts any attempt to reliably answer the scientific question at hand. Why, you ask? Consider how much knowledge one must have to *know* (beyond a shadow of a doubt) that the data-generating distribution underlying a given dataset is, in fact, governed by just two parameters, as is the case with the ubiquitously relied upon Normal distribution. Similarly, main terms Cox proportional hazards, logistic regression, and linear models imply a highly constrained statistical model, and if any of the assumptions are unwarranted then there will be bias in their result (except when treatment is randomized). The philosophy used to justify parametric assumptions is rooted in misinterpretations of the often-quoted saying of George Box, that "All models are wrong but some are useful," which has been irresponsibly used to encourage the data analyst to make arbitrary modeling choices. However, when one makes such unfounded assumptions, it is more likely that $\mathcal{M}$ does

not contain P_0, in which case the statistical model is said to be misspecified. Statistical model misspecification introduces a bias that leads to misleading, unreliable results and inference.

The result of unwarranted assumptions and oversimplifications is a practice of statistical data science in which starkly disparate answers to the same scientific problem emerge. Practically, this is owed to the application of distinct statistical techniques under differing modeling decisions and assumptions made (but not communicated well) by different data analysts. Even in the nascent days of statistical 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 roadmap avoids this bias by defining the statistical model through a representation of the true data-generating distribution underlying the observed data. The ultimate goal is to formulate the statistical estimation problem *precisely* (up to the constraints imposed by available scientific knowledge), so that one can then tailor the estimation procedure to the motivating scientific problem.

It is crucial that the domain scientist(s) have absolute clarity about what is *actually known* about the process/experiment that generated the data, and that this is communicated to data scientists with as much detail as possible. This knowledge is rarely ground truth itself, but instead comes in the form of scientific conventions, accepted hypotheses, and operational assumptions. It is then the data scientist's responsibility to translate the domain knowledge into statistical knowledge about P_0, and then to define the statistical model $\mathcal{M}$ so that it respects what is known about P_0 and makes no further restrictions. In this manner, we can ensure that P_0 is contained in $\mathcal{M}$, which we refer to generally as defining a *realistic* statistical model $\mathcal{M}$.

Defining $\mathcal{M}$ realistically requires a shift in the paradigm of statistical problem solving. Instead of considering the methods/software one is familiar with and then trying to solve most problems with that toolbox, one must obtain a deep understanding of the experiment and scientific question first and then formulate a plan for learning from the data in a way that respects this. This requires statisticians to have not only solid methodological and theoretical foundations, but good communication skills, as several meetings with domain experts are typically required to review details of the study, possibly refine of the question of interest, translate technical details, and interpret the findings in a way that is statistically correct and agreeable with non-statistician domain experts. Unfortunately, communication between statisticians and non-statistician researchers is often fraught with misinterpretation. This is to be expected, as each have their own expertise, but proper communication about the underlying science and the motivating study can help to ensure

each have appropriate context for a given statistical data analysis. The roadmap provides a principled mechanism for learning from data realistically, so that what is learned from the data represents a reliable and reproducible approximation of the answer to the scientific question of interest. As the roadmap provides a rigorous method for translating scientific knowledge and questions into a statistical framework that can be used to learn from data, it is an invaluable tool to guide communication between statisticians and non-statistician domain scientists. This brings us to our next step in the roadmap, *"What are we trying to learn from the data?"*

(3) The statistical target parameter Ψ and statistical estimand ψ_0

The statistical target parameter, Ψ, is defined as a mapping from the statistical model, $\mathcal{M}$, to the parameter space. Usually, the parameter space is a real number (but not necessarily so), in which case we can formally define the target parameter as the mapping $\Psi : \mathcal{M} \to \mathbb{R}$. The statistical 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 about a particular target population. In contrast to ordinary statistical estimands, causal estimands require an extra set of assumptions to allow for their *identification from the observed data*. Based on causal models [Pearl, 2009, Hernán and Robins, 2022], identification assumptions are untestable and must be justified through a combination of knowledge about the system under study or the process by which the experiment was conducted. These assumptions are described in greater detail in the following section on causal target parameters.

For a simple example, consider a dataset containing observations of a survival time on every adult, for which our question of interest is "What's the probability that an adult lives longer than five years?" We have,

$$\psi_0 = \Psi(P_0) = \mathbb{E}_{P_0}(O > 5) = \int_5^\infty dP_0(o).$$

This answer to this question is the **statistical estimand,** $\Psi(P_0) = \psi_0$, which is the quantity we wish to learn from the data. As discussed above, back-and-forth communication between domain scientists and statisticians is often required to define $\mathcal{M}$ realistically, and to finalize Ψ and the target population such that the question is supported in the data. For instance, say we are interested in learning the average effect of a headache medication for treating migraines in adults and we learn that no one with high blood pressure can receive the medication. In the next meeting with domain scientists, we might suggest that the target population be modified to adults without high blood pressure or ask a question involving a

dynamic treatment such that within Ψ adults with high blood pressure are never considered as individuals who could receive treatment. Once we have defined O, $\mathcal{M}$ realistically and Ψ, we have formally defined the statistical estimation problem. Next comes Step 4: "*How do we learn from the data the approximate answer to the question of interest?*"

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

To obtain a good approximation of the statistical estimand, we need an estimator $\hat{\Psi}$, an *a priori*-specified algorithm defined as a mapping from the set of the set of possible empirical distributions P_n (which live in a non-parametric statistical model $\mathcal{M}_{NP}$) to the parameter space for our target parameter of interest: $\hat{\Psi} : \mathcal{M}_{NP} \rightarrow \mathbb{R}$. In other words, $\hat{\Psi}$ is a function that takes as input the observed data, a realization of P_n, and then outputs a value in the parameter space. Where the estimator may be seen as an operator that maps the observed data's corresponding empirical distribution to a value in the parameter space, the numerical output produced by such a function is the **estimate,** $\hat{\Psi}(P_n) = \psi_n$. Thus, ψ_n is an element of the parameter space as informed by the empirical probability distribution P_n of the observed data $O_1, \ldots, O_n$. 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 ψ_n of the true parameter value ψ_0. As we have motivated in step 2, it is imperative to consider realistic statistical models for estimation. Therefore, flexible estimators that allow for parts of the data-generating process to be unrestricted are necessary. Semiparametric theory and empirical process theory provide a framework for constructing, benchmarking, and understanding the behavior of estimators that depend on flexible estimation strategies in realistic statistical models. In general, desirable properties of an estimator are that it is regular asymptotically linear (RAL) and efficient, thereby admitting a Normal limit distribution that has minimal variance. Substitution/plug-in RAL estimators are also advantageous: they are guaranteed to remain within the bounds of $\mathcal{M}$ and, relative to estimators that are not plug-in, have improved bias and variance in finite samples. In-depth discussion of the theory and these properties are available in the literature [e.g., Kennedy, 2016, van der Laan and Rose, 2011]. We review a few key concepts in the following step.

In order to quantify the uncertainty in our estimate of the target parameter, part of the process of conducting statistical inference, an understanding of the sampling distribution of our estimator is necessary. This brings us to Step 5: "*How confident should we be in our statistical answer to the scientific question?*"

(5) A measure of uncertainty for the estimate ψ_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. Therefore, if we repeat the experiment of drawing n observations, we would every time end up with a different realization of our estimate. The hypothetical distribution of these estimates is the sampling distribution of the estimator.

A primary goal in the construction of estimators is to be able to derive their asymptotic sampling distribution through a theoretical analysis involving empirical process theory. In this regard, an important property of the estimators on which we focus is their asymptotic linearity. In particular, asymptotic linearity states that the difference between the estimator and the target parameter (i.e., the truth) can be represented, asymptotically, as an average of i.i.d. random variables plus an asymptotically negligible remainder term:

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

where the influence curve (IC) is a function of the observed data O but the function itself is defined by the underlying data-generating distribution P_0. Based on this asymptotic approximation, the Central Limit Theorem can be used 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 $IC(P_0)(O)$. 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 takes the form

$$\psi_n \pm Z\frac{\hat{\sigma}_{IC}}{\sqrt{n}},$$

where Z is the $(1-\alpha/2)^{\text{th}}$ quantile of the standard Normal distribution. Following convention, we will often be interested in constructing 95% two-tailed CIs, corresponding to probability mass $\alpha/2 = 0.025$ in each tail of the limit distribution; thus, we will take $Z \approx 1.96$ as the quantile.

Steps (1)–(5) of the roadmap define the statistical analysis plan, all of which can be done before any data is revealed. The last step of the roadmap involves interpreting the results obtained in step (4) and (5) and therefore requires the data to be analyzed; however, any additional analysis that may take place as part of step (6) can be pre-specified as well. This final step of the roadmap addresses the question, "*what is the interpretation and robustness of the study's findings, and what conclusions can be drawn from them?*"

(6) Make substantive conclusion

Making the substantive conclusion involves interpreting the study findings. It also provides an opportunity to ask follow-up questions that might be addressed later and/or discuss issues that can inform future studies. Statistical estimands ψ_0 can have statistical (noncausal) and causal interpretations. Both are often of interest and can be provided. The target population should be clearly mentioned in the interpretation, regardless of whether it's a purely statistical or causal interpretation, to curtail extrapolation of results.

The major distinction between statistical versus causal interpretations is that the latter relies on untestable so-called "identifiability" assumptions. In the following section, we review these assumptions one-by-one. Here, we focus on the interpretation and robustness of the study findings with respect to them. Specifically, causal target parameters cannot be estimated from observed data without additional identifiability assumptions, and so the validity of a result's causal interpretation hinges on them holding in the data. The more these assumptions do not hold, the larger the *causal gap*, the difference between the statistical estimand and the causal estimand. In a perfect randomized control trial with no loss to follow-up, the causal gap will be zero as the statistical and causal estimands are equivalent. In Díaz and van der Laan [2013], a non-parametric sensitivity analysis for assessing the impact of a hypothesized causal gaps on estimates and inference is proposed. In Gruber et al. [2023] and Gruber et al. [2022], there are example implementations of the methods proposed in Díaz and van der Laan [2013]; in particular, the difference between adjusted and unadjusted effect estimates is used to define a range of possible causal gaps relative to this difference. If the question of interest is causal, then such a model-free sensitivity analysis (possibly as a complement to other sensitivity analyses) is recommended to assess the robustness of the study findings.

4.2 Summary of the Roadmap

Data collected across n i.i.d. units, $O_1, \ldots, O_n$, may be viewed as a collection of random variables arising from the same underlying probability distribution $\mathbb{P}_0$. This is expressed by denoting the collection of data as being generated as $O_1, \ldots, O_n \sim P_0$. Domain knowledge about the experiment that generated the data (e.g., if the treatment was randomized, if the treatment decision or loss to follow-up depended on a subset of covariates, time ordering in which the variables were added to the data) is translated by the statistician / data scientist to define the statistical model $\mathcal{M}$, a postulated space of candidate probability distributions

that is supposed to contain P_0. In particular, the roadmap emphasizes the critical role of defining $\mathcal{M}$ such that P_0 is guaranteed to be encapsulated by it, $P_0 \in \mathcal{M}$. By only limiting $\mathcal{M}$ based on domain knowledge about the experiment (i.e., reality) — opposed to constraining it unrealistically (e.g., assuming a restrictive functional form, like a main terms linear/logistic model, describes P_0) — it can be ensured that $P_0 \in \mathcal{M}$, and we refer to this as defining a realistic statistical model. Often, knowledge that can be used to constrain $\mathcal{M}$ is very limited, and so $\mathcal{M}$ must be very large to define it such that $P_0 \in \mathcal{M}$; hence, 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, a function defined by the true data-generating distribution P_0, that maps $\mathcal{M}$ into the statistical estimand, ψ_0. At this stage, the statistical estimation problem is formally defined, allowing for the use of statistical theory to guide the construction of estimators, which are algorithms that approximate the answer the question of interest by learning from the data. Desirable properties of an estimator are that it is unbiased, efficient, plug-in, and robust in finite samples. If the question of interest is causal, then a model-free sensitivity analysis is recommended to assess the robustness of the study's findings under various hypothesized causal gaps.

4.3 Causal Target Parameters

In many cases, we are interested in problems that ask questions regarding the *causal effect* of an intervention, whether an assigned treatment (e.g., a prescribed drug) or a "naturally occurring" exposure (e.g., pollution from a nearby factory), on a future outcome of interest. These causal effects may be defined as summaries of the population of interest (e.g., population mean of a particular outcome) under contrasting interventions (e.g., comparing the treated to the untreated condition). For example, a causal effect could be defined as the mean difference of a disease outcome between two *causal contrasts*, counterfactual cases in which the study population were set to uniformly experience low pollution levels for some pollutant, and in which the same population were set to uniformly experience high levels of the same pollutant.

There are different ways of operationalizing the theoretical experiments that generate the counterfactual data necessary for describing such causal contrasts of interest. We could simply assume that the counterfactual outcomes exist in theory for all treatment contrasts

of interest [Neyman, 1938, Rubin, 2005, Imbens and Rubin, 2015], which may be encoded in so-called "science tables". Alternatively, we could consider interventions on structural causal models (SCMs) [Pearl, 1995, 2009], which may be represented by directed acyclic graphs (DAGs). Both frameworks allow for the known or hypothesized set of relationships between variables in the system under study to be encoded and mathematically formalized.

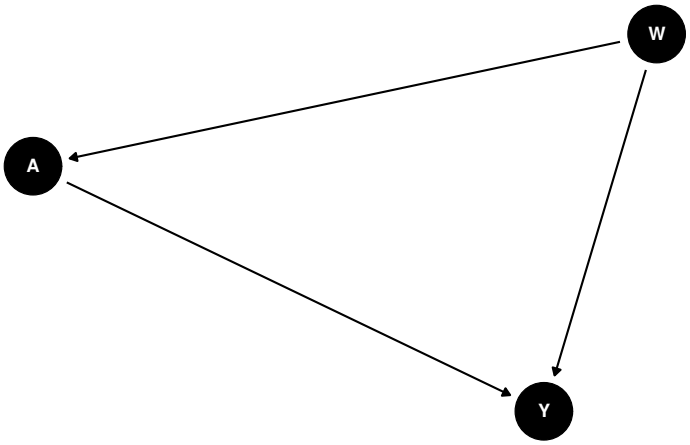
The Causal Model

Throughout, we will focus on the use of DAGs and SCMs 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 visually 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 that compose the observed data O. We demonstrate the construction of a DAG below using `DAGitty` [Textor et al., 2011]:

```
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 express the causal relations between variables, these same causal relations can be equivalently represented by an SCM:

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

where the f's are unspecified deterministic functions that generate the corresponding random variables as a function of the variable's "parents" (i.e., upstream nodes with arrows into the given random variable) in the DAG, and the unobserved, exogenous error terms (i.e., the U's). An SCM 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 functional form f_Y governing the outcome variable 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 distinct interventions:

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

These interventions may be thought of as operations that imply changes to the structural

equations in the system under study. 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 the intervention on the system set A deterministically to one of the values 1 or 0 consistent with the intervention performed. The new symbols $Y(1)$ and $Y(0)$ indicate the values the outcome variable would take in the population of interest when it is generated by removing the contribution of A to f_Y and instead setting A to the values 1 and 0, respectively. The variables $Y(1)$ and $Y(0)$ are often called counterfactuals (since they arise from interventions that run contrary to fact) and are, in other frameworks, called the *potential outcomes* of Y [Neyman [1938]; rubin2005causal; imbens2015causal]. The difference in the counterfactual means of the outcome under these two interventions defines a well known causal parameter that is most often called the "average treatment effect" (ATE) and is denoted

$$ATE = \mathbb{E}_X[Y(1) - Y(0)], \tag{4.1}$$

where $\mathbb{E}_X(\cdot)$ is the expectation taken over the theoretical (unobservable) full data (i.e., $X = (W, Y(1), Y(0))$) distribution P_X. Note that the full data structure X is, by its very definition, unobservable since one can never observe both of $Y(1)$ and $Y(0)$ for the same observational unit.

We can define much more complicated interventions on SCMs, such as interventions based upon dynamic rules (which assign particular interventions based on a function of the covariates W), stochastic rules (which can even account for the natural value of A observed in the absence of the intervention), and much more. Each results in a different target causal parameter and entails different identifiability assumptions discussed below.

Identifiability

Since we can never simultaneously observe $Y(0)$, the counterfactual outcome when $A = 0$, and $Y(1)$, the counterfactual outcome when $A = 1$, we cannot estimate their difference $Y(1) - Y(0)$ (the individual treatment effect), which appears in Equation (4.1) (inside the

expectation $\mathbb{E}_X(\cdot)$ that defines ATE). This is called the *Fundamental Problem of Causal Inference* [Holland, 1986]. Thus, one of the primary activities in causal inference is to *identify* the assumptions necessary to express causal quantities of interest as functions of the data-generating distribution of the observed data. To do this, we must make assumptions under which such quantities may be estimated from the observed data $O \sim P_0$ and its corresponding data-generating distribution P_0. Fortunately, given the causal model specified in the SCM above, we can, with a handful of untestable assumptions, estimate the ATE from observational data. These assumptions may be summarized as follows.

Definition 4.1 (Consistency)**.** The outcome for unit i is $Y_i(a)$ whenever $A_i = a$, which may be thought of as "no other versions of treatment" or "no side effects of treatment."

Definition 4.2 (No Interference)**.** The outcome for unit i, Y_i, cannot be affected by the exposure of unit j, A_j, for all $i \neq j$.

Definition 4.3 (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.

Definition 4.4 (Positivity/Overlap)**.** All observed units, across strata defined by W, must have a bounded probability of receiving treatment – that is, $\epsilon < \mathbb{P}(A = a \mid W) < 1 - \epsilon$ for all a and W and for some $\epsilon > 0$) .

Technically speaking, only the latter two of these assumptions are necessary when working within the SCM framework, as the first two are implied properties of an SCM for i.i.d. data (if you're really curious, see this commentary of Pearl [2010] for an extended discussion). We introduce all four identification assumptions because they are most often considered together, and all four are necessary when working within the potential outcomes framework [Rubin, 2005, Imbens and Rubin, 2015].

Under these assumptions, the ATE may be re-written as a function of P_0, the distribution of the observed data:

$$\begin{aligned} \psi_{\text{ATE}} &= \mathbb{E}_0[Y(1) - Y(0)] \\ &= \mathbb{E}_0[\mathbb{E}_0[Y \mid A = 1, W] - \mathbb{E}_0[Y \mid A = 0, W]] \, . \end{aligned} \tag{4.2}$$

In words, the ATE is the mean difference in the predicted outcome values for each subject, under the contrast of treatment conditions ($A = 0$ versus $A = 1$), in the population (when averaged over all observations). Thus, a parameter of a theoretical complete (or "full")

data distribution can be represented as an estimand of the observed data distribution. Significantly, there is nothing about the representation in Equation (**??**) that requires parameteric assumptions; thus, the regression functions on the right hand side may be estimated without restrictive assumptions about their underlying functional forms. 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 subsequent chapters.

5

Cross-validation

Ivana Malenica

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

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

Following the *Roadmap for Targeted Learning*, we start to elaborate on the estimation step in the current chapter. In order to generate an estimate of the target parameter, we need to decide how to evaluate the quality of our estimation procedure's performance. The performance, or error, of any algorithm (estimator) corresponds to its generalizability to

independent datasets arising from the same data-generating process. Assessment of the performance of an algorithm is extremely important — it provides a quantitative measure of how well the algorithm performs, and it guides the choice of selecting among a set (or "library") of algorithms. In order to assess the performance of an algorithm, or a library of them, we introduce the concept of a **loss function**, which defines the **risk** or the **expected prediction error**. Our goal is to estimate the true performance (risk) of our estimator. In the next chapter, we elaborate on how to estimate the performance of a library of algorithms in order to choose the best-performing one. In the following, we propose a method to do so using the observed data and **cross-validation** procedures implemented in the `origami` package [Coyle and Hejazi, 2018, Coyle et al.].

5.2 Background

Ideally, in a data-rich scenario (i.e., one with unlimited observations), we would split our dataset into three parts:

1. the training set,
2. the validation set, and
3. the test (or holdout) 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 algorithm tuning or selection). The final error of the selected algorithm is obtained by using the test (or holdout) set, which is kept entirely separate such that the algorithms never encounter these observations until the final model evaluation step. 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 a biased estimate of a fitted algorithm's generalizability, since it uses the same data for fitting and evaluation.

Since data are often scarce, separating a dataset into training, validation and test sets can prove too limiting, on account of decreasing the available data for use in training by too much. In the absence of a large dataset and a designated test set, we must resort to methods that estimate the algorithm's true performance by efficient sample re-use. Re-sampling methods, like the bootstrap, involve repeatedly sampling from the training set and fitting the algorithms to these samples. While often computationally intensive, re-sampling methods are particularly useful for evaluating an algorithm and selecting among a set of

them. In addition, they provide more insight on the variability and robustness of a fitted algorithm, relative to fitting an algorithm only once to all of the training data.

5.2.1 Introducing: cross-validation

In this chapter, we focus on **cross-validation** — an essential tool for evaluating how any algorithm extends from a sample of data to the target population from which it arose. Cross-validation has seen widespread application in all facets of modern statistics, and perhaps most notably in statistical machine learning. The cross-validation procedure has been proven to be optimal for algorithm selection in large samples, i.e. asymptotically. In particular,cross-validated algorithm 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 — a hypothetical procedure with free access to the true, unknown data-generating distribution. For further details on the theoretical results, we suggest consulting 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].

The `origami` package provides a suite of tools for cross-validation. In the following, we describe different types of cross-validation schemes readily available in `origami`, introduce the general structure of the `origami` package, and demonstrate the use of these procedures in various 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** as a guide for the estimation process. In particular, the unified loss-based estimation framework [van der Laan and Dudoit, 2003, van der Laan et al., 2004, Dudoit and van der Laan, 2005, van der Vaart et al., 2006, van der Laan et al., 2007], which relies on cross-validation for estimator construction, selection, and performance assessment, consists of three main steps:

1. **Loss function**: 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. By

full data, we refer to the complete data including missingness process, for example. Map the full data loss function into an observed data loss function, having the same expected value and leading to an estimator of risk.

2. **Algorithms**: Construct a finite collection of candidate estimators of the parameter of interest.

3. **Cross-validation scheme**: Apply appropriate cross-validation, and use the cross-validated risk in order to select the best performing estimator among the candidates. Assess the overall performance of the resulting estimator.

5.4 Example: Cross-validation and Prediction

Having introduced the Estimation Roadmap, we can more precisely define our objective using prediction as an example. Let the observed data be defined as $O = (W, Y)$, where a unit specific data structure can be written as $O_i = (W_i, Y_i)$, for $i = 1, \ldots, n$. We denote Y_i as the outcome/dependent variable of interest, and W_i as a p-dimensional set of covariate (predictor) variables. We assume the n units are independent, or conditionally independent, and identically distributed. Let $\psi_0(W)$ denote the target parameter of interest, the quantity we wish to estimate (estimand). For this example, we are interested in estimating the conditional expectation of the outcome given the covariates, $\psi_0(W) = \mathbb{E}(Y \mid W)$. Following the Estimation Roadmap, we choose the appropriate loss function, L, such that $\psi_0(W) = \text{argmin}_\psi \mathbb{E}_0[L(O, \psi(W))]$. Note that $\psi_0(W)$, the true target parameter, is a minimizer of the risk (expected value of the chosen loss function). The appropriate loss function for conditional expectation with continuous outcome could be a mean squared error, for example. Then we can define L as $L(O, \psi(W)) = (Y_i - \psi(W_i)^2$. Note that there can be many different algorithms which estimate the estimand (many different ψs). How do we know how well each of the candidate estimators of $\psi_0(W)$ are doing? To pick the best-performing candidate estimator and assess its overall performance, we use cross-validation. Observations in the training set are used to fit (or train) the estimator, while those in validation set are used to assess the risk of (or validate) it.

Next, we introduce notation flexible enough to represent any cross-validation scheme. In particular, we define a **split vector**, $B_n = (B_n(i) : i = 1, \ldots, n) \in \{0, 1\}^n$. A realization of B_n defines a random split of the data into training and validation subsets 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 the training and validation sets, respectively. The particular distribution of the split vector B_n defines the type of cross-validation scheme, tailored to the problem and dataset at hand.

5.5 Cross-validation schemes in `origami`

A variety of different partitioning schemes exist, each tailored to the salient details of the problem of interest, including data size, prevalence of the outcome, and dependence structure (between units or across time). In the following, we describe different cross-validation schemes available in the `origami` package, and we go on to demonstrate their use in practical data analysis examples.

WASH Benefits Study Example

In order to illustrate different cross-validation schemes, we will be using the WASH Benefits example dataset (detailed information 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 from an unknown distribution P_0. 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. sampled unit, indexed by the row number.

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

# 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)	146.4	Food Secu
-1.16	Control	N05265	9	286	male	25	Primary (1-5y)	148.8	Moderatel
-1.05	Control	N08002	9	264	male	25	Primary (1-5y)	152.2	Food Secu
-1.26	Control	N08002	9	252	female	28	Primary (1-5y)	140.2	Food Secu
-0.59	Control	N06531	9	336	female	19	Secondary (¿5y)	150.9	Food Secu
-0.51	Control	N06531	9	304	male	20	Secondary (¿5y)	154.2	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 perhaps the simplest strategy for estimating the risk of a fitted algorithm. With this cross-validation scheme, all observed data units are used in both the training and validation set.

We illustrate the usage of the re-substitution method with `origami` below, using the function `folds_resubstitution`. In order to set up `folds_resubstitution`, we need only to specify the total number of sampled units that we want to allocate to the training and validation sets; remember that each row of the dataset is a unique i.i.d. sampled unit. Also, notice the structure of the `origami` output:

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

The structure of the `origami` output, a `list` of fold(s), holds across all of the cross-validation schemes presented in this chapter. Below, we show the fold generated by the re-substitution method:

```
folds <- folds_resubstitution(nrow(washb_data))
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
```

```
$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

The holdout method, or the validation set approach, consists of randomly dividing the available data into training and validation (holdout) sets. The model is then fitted (i.e., "trained") using the observations in the training set and subsequently evaluated (i.e., "validated") using the observations in the validation set. Typically, the dataset is split into $60:40$, $70:30$, $80:20$ or even $90:10$ training-to-validation splits.

The holdout method is intuitive and computationally inexpensive; however, it does carry a disadvantage: If we were to repeat the process of randomly splitting the data into training and validation sets, we could get very different cross-validated estimates of the empirical risk. In particular, the empirical mean of the loss function (i.e., the empirical risk) evaluated over the validation set(s) could be highly variable, depending on which samples were included in the training and validation splits. Overall, the cross-validated empirical risk for the holdout method is more variable, since in includes variability of the random split as well — this is not desirable. For classification problems (with a binary or categorical outcome variable), there is an additional disadvantage: it is possible for the training and validation sets to end up with uneven distributions of the two (or more) outcome classes, leading to better training and poor validation, or vice-versa — though this may be corrected by incorporating stratification into the cross-validation process. Finally, note that we are not using all of the data in training or in evaluating the performance of the proposed algorithm, which could itself introduce bias.

5.5.1.3 Leave-one-out

The leave-one-out cross-validation scheme is closely related to the holdout method, as it also involves splitting the dataset into training and validation sets; however, instead of partitioning the dataset into sets of similar size, a single observation is used as the validation set. In doing so, the vast majority of the sampled units are employed for fitting (or training) the candidate learning algorithm. Since only a single sampled unit (for example $O_1 = (W_1, Y_1)$) is left out of the fitting process, leave-one-out cross-validation can result in a less biased estimate of the risk. Typically, the leave-one-out approach will

not overestimate the risk as much as the holdout method does. On the other hand, since the estimate of risk is based on a single sampled unit, it is usually a highly variable estimate.

We can repeat the process of spiting the dataset into training and validation sets until all of the sampled units have had a chance to act as the validation set. Continuing the example above, a subsequent iteration of the leave-one-out cross-validation scheme may use $O_2 = (W_2, Y_2)$ as the validation set (where, before, $O_1 = (W_1, Y_1)$ played that role), while the remaining $n - 1$ sampled units are included in the training set. Repeating this approach n times results in n risk estimates, for example, $MSE_1, MSE_2, \ldots, MSE_n$ (note that these are the mean squared error (MSE) estimates when unit i is the validation set). The estimate of the true risk is then the average over the n leave-one-out risk estimates. While the leave-one-out cross-validation scheme results in a less biased (albeit, more variable) estimate of risk than the holdout method, it can be computationally very expensive to implement when n is large.

We illustrate the usage of the leave-one-out cross-validation scheme with `origami` below, using the `folds_loo(n)` function. In order to set up `folds_loo(n)`, similarly to the case of the re-substitution method, we need only the total number of sampled units over which the cross-validation procedure is to operate. We show the first two folds generated by 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 the leave-one-out scheme is V-fold cross-validation. This cross-validation scheme randomly divides the dataset into V splits of equal (or approximately equal) size. For each split $v = 1, \ldots, V$, the vth fold is defined by the vth split (which defines the validation set for fold v) and the complement of the vth split (which defines the training set for fold v). The algorithms are fit V times separately, to each of the V training sets. The risk of each fold's fitted algorithms is then evaluated via predictions obtained from the validation set. The cross-validated risk for a fitted algorithm, for example the MSE, is its average risk across all folds. With V-fold cross-validation, all of the observations are used in the training and validation stages, preventing the candidate learning algorithm from overfitting to only a subset of the data (e.g., a given training set).

For a dataset with n sampled units, V-fold cross-validation with $v = n$ merely reduces to leave-one-out. Similarly, if we set $n = 1$, we can get the holdout method's estimate of the candidate learning algorithm's performance. Beyond its computational advantages, V-fold cross-validation often yields more accurate estimates of the true, underlying risk. This is rooted in the differing bias-variance trade-offs associated with these two cross-validation schemes: While the leave-one-out scheme may be less biased, it has much greater variance (since only a single unit is included in the validation set). This difference becomes more obvious as n becomes much greater than v. With the V-fold cross-validation scheme, we end up averaging risk estimates across the v validation folds, which are typically less correlated than the risk estimates from the leave-one-out fits. Owing to the fact that the mean of many highly correlated quantities has higher variance, leave-one-out estimates of the risk will have higher variance than the corresponding estimates based on V-fold cross-validation.

Now, let's see V-fold cross-validation with `origami` in action! In the next chapter, we will turn to studying the Super Learner algorithm — an algorithm capable of selecting the "best" algorithm from among a large library of candidate learning algorithms – which we'd like to fit *and* evaluate the performance of. The Super Learner algorithm relies on V-fold cross-validation as its default cross-validation scheme. In order to set up V-fold cross-validation, we need to call `origami`'s `folds_vfold(n, V)` function. The two required arguments for `folds_vfold(n, V)` are the total number of sample units to be cross-validated and the number of folds we wish to have.

For example, at $V = 2$, we will get two folds, each with approximately $n/2$ sampled units in the training and validation sets.

```
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

In the Monte Carlo cross-validation scheme, we randomly select some fraction of the data, *without replacement*, to form the training set, assigning the remainder of the sampled units to the validation set. In this way, the dataset is randomly divided into two independent splits: A training set of $n_0 = n \cdot (1 - p)$ observations and a validation set of $n_1 = n \cdot p$ observations. By repeating this procedure many times, the Monte Carlo cross-validation scheme generates, at random, many training and validation partitions of the dataset.

Since the partitions are independent across folds, the same observational unit can appear in the validation set multiple times; note that this is a stark difference between the Monte Carlo and V-fold cross-validation schemes. For a given sampling fraction p, the Monte Carlo cross-validation scheme would be optimal if repeated infinitely many times — of course, this is not computationally feasible. With Monte Carlo cross-validation, it is possible to explore many more partitions of the dataset than with V-fold cross-validation,

resulting in (possibly) less variable estimates of the risk (across partitions), though this comes at the cost of an increase in bias (because the splits are correlated). Because Monte Carlo cross-validation generates many splits with overlaps in the sampled units, more splits (and thus more computational time) will be necessary to achieve the level of performance (in terms of unbiasedness) that the V-fold cross-validation scheme achieves with only V splits.

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

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

For example, setting $V = 2$ and $pvalidation = 0.2$, we obtain two folds, each with approximately 6 sampled units in the validation set for each 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

Like the Monte Carlo cross-validation scheme, the bootstrap cross-validation scheme also consists of randomly selecting sampled units, *with replacement*, for the training set; the rest of the sampled units are allocated to the validation set. This process is then repeated multiple times, generating (at random) new training and validation partitions of the dataset each time. In contrast to the Monte Carlo cross-validation scheme, the total number of sampled units in training and validation sets (i.e., the sizes of the two partitions) across folds is not held constant. Also, as the name suggests, sampling is performed with replacement (as in the bootstrap [Davison and Hinkley, 1997]), hence the exact same observational units may be included in multiple training sets. The proportion of observational units in the validation sets is a random variable, with expectation ~ 0.368.

We illustrate the usage of the bootstrap cross-validation scheme with `origami` below, using the `folds_bootstrap(n, V)` function. In order to set up `folds_bootstrap(n, V)`, we need to specify the following arguments:

1. the total number of observations we wish to cross-validate; and
2. the number of folds.

For example, setting $V = 2$, we obtain two folds, each with different numbers of sampled units in the validation sets across the 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 Time-series 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 Data 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, included in R, provides monthly totals of international airline passengers between the years 1949 and 1960.

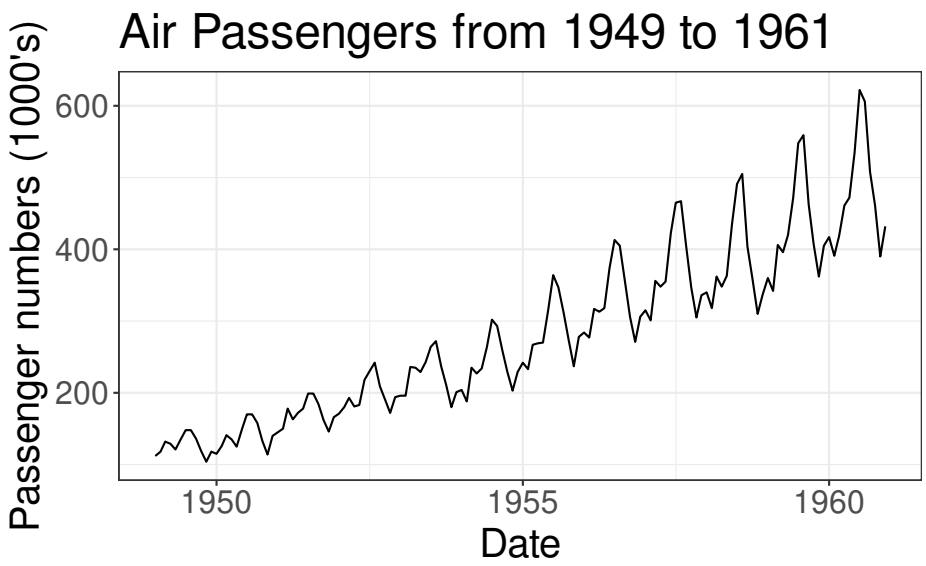
Goal: we want to forecast the number of airline passengers at time h horizon using the historical data from 1949 to 1960.

```
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

The rolling origin cross-validation scheme lends itself to "online" learning algorithms, in which large streams of data have to be fit continually (respecting time), where the fit of the learning algorithm is (constantly) updated as more data accrues. In general, the rolling origin scheme defines an initial training set, and, with each iteration, the size of the training set grows by a batch of m observations, the size of the validation set remains constant, there might be a gap between training and validation times of size h (a lag window), and new folds are added until time t is reached in the validation set. The time points included in the training set always lag behind behind those in the validation set.

To further illustrate rolling origin cross-validation, we show below an example that yields three folds. Here, the first window size is fifteen time points, on which we first train the candidate learning algorithm. We then evaluate its performance on ten time points with a gap (h) of five time points between the training and validation sets.

In the following, we train the learning algorithm on a longer stream of data, 25 time points, including the original fifteen with which we initially started. Then, we evaluate its performance at a (temporal) distance ten time points ahead.

We illustrate the usage of the rolling origin cross-validation scheme with `origami` below, using the `folds_rolling_origin(n, first_window, validation_size, gap, batch)` function. In order to set up `folds_rolling_origin(n, first_window, validation_size, gap, batc` we need the following,

1. the total number of time points we wish to cross-validate (`n`);

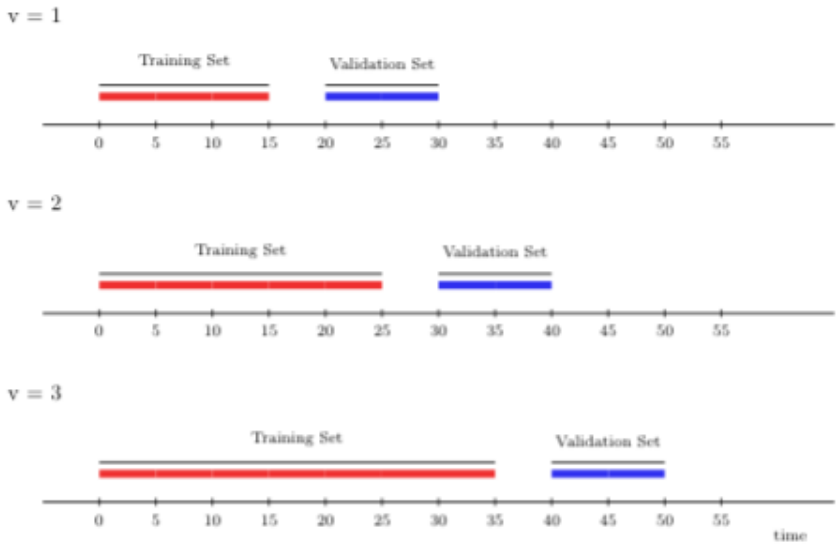


Figure 5.1: Rolling origin CV

2. the size of the first training set (`first_window`);
3. the size of the validation set (`validation_size`);
4. the gap between training and validation set(`gap`); and
5. the size of the training set update per iteration of cross-validation (`batch`).

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

```
folds <- folds_rolling_origin(
  n = 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

Rather than adding more and more time points to the training set in each iteration of cross-validation (as under the rolling origin scheme), the rolling window cross-validation scheme "rolls" the training sample forward in time by m units (of time). This strategy can be useful, for example, in settings with parametric learning algorithms, which are often very sensitive to moment (e.g., mean, variance) or parameter drift, which is itself challenging to explicitly account for in the model construction step. The rolling window scheme is also computationally more efficient, and possibly warranted over rolling origin when working in streaming data analysis where the training data is too large for convenient access. In contrast to the rolling origin scheme, the sampled units in the training set are always the same for each iteration of the rolling window scheme.

The illustration below depicts rolling window cross-validation using three time-series folds. The first window size is 15 time points, on which we first train the candidate learning algorithm. As in the previous illustration, we evaluate its performance on 10 time points, with a gap of size 5 time points between the training and validation sets. However, for the next fold, we train the learning 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 (both include 15 time points). This setup keeps the training sets comparable over time (and across folds), unlike under the rolling origin cross-validation scheme. We then evaluate the performance of the candidate learning algorithm on 10 time points in the future.

We demonstrate the usage of the rolling window cross-validation scheme with `origami`

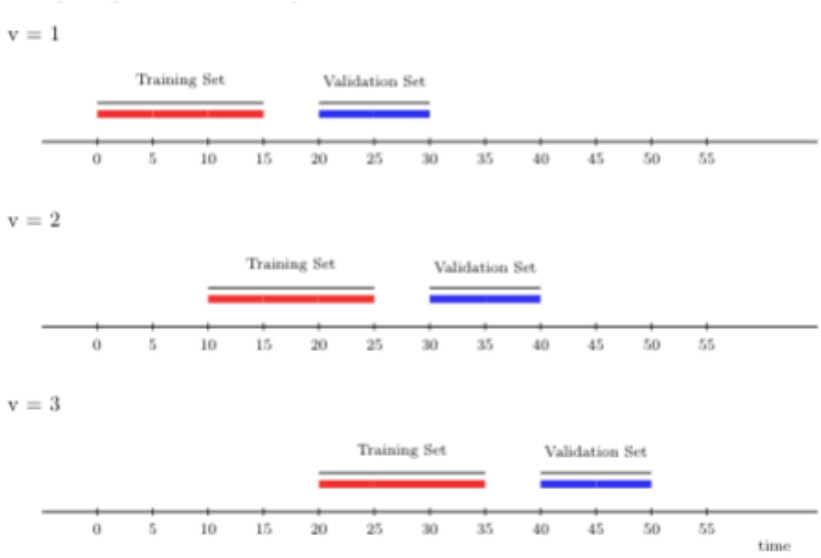


Figure 5.2: Rolling window CV

below, using the `folds_rolling_window(n, window_size, validation_size, gap, batch)` function. In order to set up `folds_rolling_window(n, window_size, validation_size, gap, batch)`, we need to specify the following arguments:

1. the total number of time points we wish to cross-validate (`n`);
2. the size of the training sets (`window_size`);
3. the size of the validation set (`validation_size`);
4. the gap between training and validation set (`gap`); and
5. the size of the training set update per iteration of cross-validation (`batch`).

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(
  n = 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 the rolling origin cross-validation scheme, accounting for sample dependence, is the rolling-origin-V-fold cross-validation scheme. In contrast to the canonical rolling origin scheme, under this hybrid scheme, sampled units in the training and validation sets are *not* the same, which this scheme accomplishes by incorporating V-fold cross-validation within the time-series setup. Here, the learning algorithm's predictions are evaluated on the future time points of the time-series observational units excluded from the training step, accommodating potential dependence not only across time but also across observational units. To use the rolling-origin-V-fold cross-validation scheme with `origami`, we can invoke the `folds_vfold_rolling_origin_pooled(n, t, id, time, V, first_window, validation_size, g` function. In the figure below, we show $V = 2$ folds, alongside two time-series (rolling origin) cross-validation folds.

5.5.2.4 Rolling window with V-fold

Just like the scheme described above, the rolling window approach, like the rolling origin approach, can be extended to support multiple time-series with arbitrary sample-level dependence by incorporating a V-fold splitting component. This

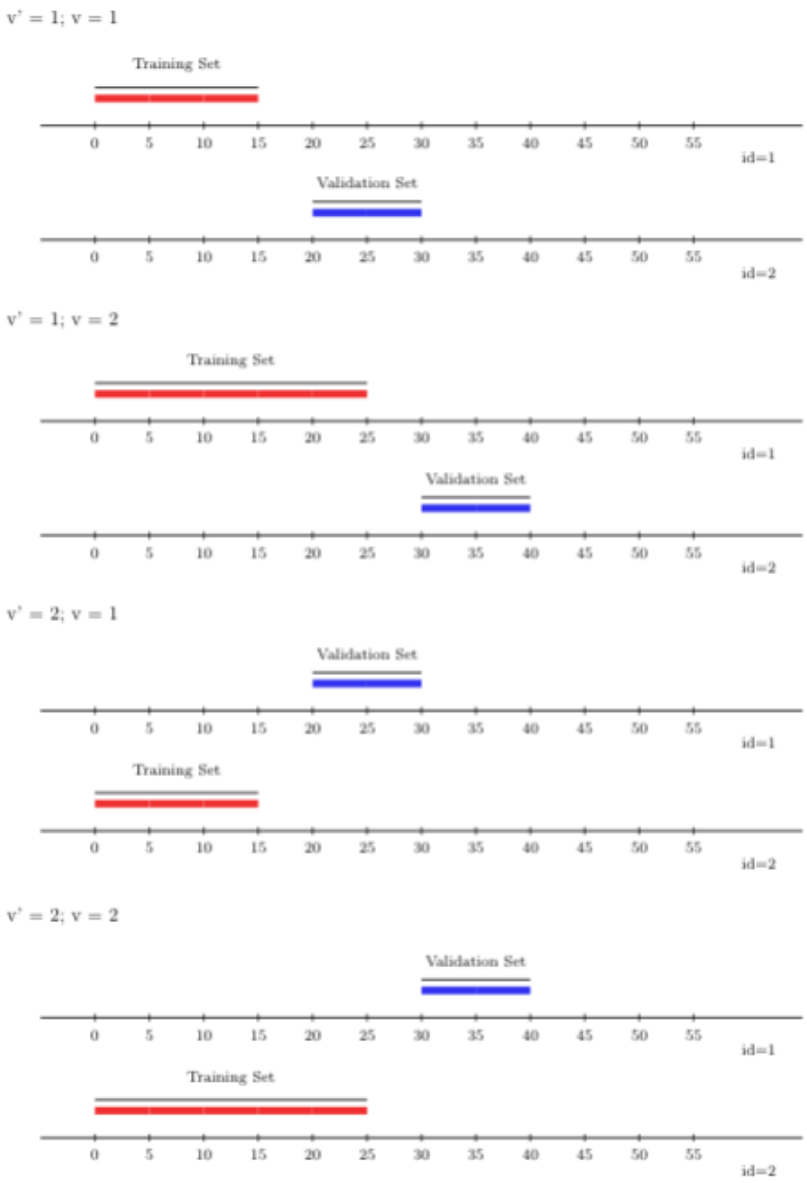


Figure 5.3: Rolling origin V-fold CV

rolling-window-V-fold cross-validation scheme can be used through `origami` via the `folds_vfold_rolling_window_pooled(n, t, id, time, V, window_size, validation_size, ga` function. The figure below displays $V = 2$ folds and two time-series (rolling window) cross-validation 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 the `origami R` package. The main workhorse function in `origami` is `cross_validate()`. To start off, the user must define the fold structure and a function that operates on each fold (this `cv_fun()`, in `origami`'s parlance, usually dictates how the candidate learning algorithm is trained and its predictions validated).

Once passed to `cross_validate()`, the workhorse function will iteratively apply the specified function (i.e., `cv_fun()`) to each fold, combining 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 `list` objects are generated using the `make_folds()` helper function. Each fold consists of a `list` with a `"training"` index vector, a `"validation"` index vector, and a `"fold_index"` (its order in the overall `list` of folds). The `make_folds()` function supports a variety of cross-validation schemes, described in the preceding section. The `make_folds()` function can also ensure balance across levels of a given variable (through the `strata_ids` arguments), and it can also keep all observations on the same independent unit together (via the `cluster_ids` argument).

5.6.2 (2) Define the fold function

The `cv_fun` argument to `cross_validate()` is a custom function that performs some operation on each fold (again, *usually* this specifies the training of the candidate learning algorithm and its evaluation on a given training/validation split, i.e., in a single fold). The first argument to this function is the `fold`, which specifies the indices of the units in a

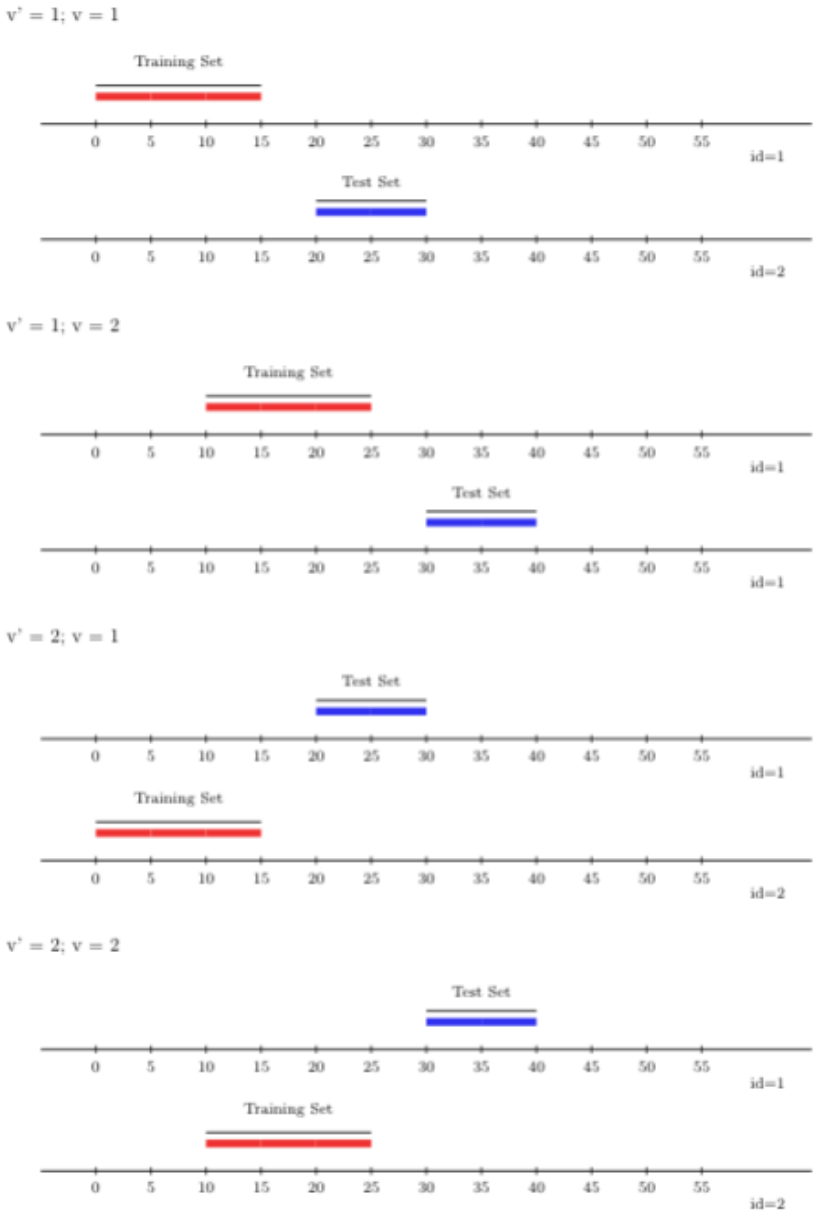


Figure 5.4: Rolling window V-fold CV

given training/validation split (note that this first argument is automatically passed to the `cv_fun()` by `cross_validate()`, which queries the folds object from `make_folds()` in doing so). Additional arguments can be passed to the `cv_fun()` through the `...` argument to `cross_validate()`. Within this function, the convenience functions `training()`, `validation()` and `fold_index()` can be used to return the various components of a fold object. When the `training()` or `validation()` functions are passed an object of a particular class, they will index that object in a sensible way. For instance, if the input object is a vector, these helper functions will index the vector directly, but if the input object is a `data.frame` or `matrix`, these functions will automatically index the 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 desired.

5.6.3 (3) Apply `cross_validate()`

After defining the folds, the `cross_validate()` function can be used to map the `cv_fun()` across the `folds`; internally, this uses either `lapply()` or `future_lapply()` (a parallelized variant of the same). In this way, `cross_validate()` 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, matching the customized `list` specified in the relevant `cv_fun()`. As noted above, each call to `cv_fun()` itself returns a `list` of results, with different named slots for each type of result we wish to store. The main `cross_validate()` loop generates a `list` of these individual, fold-specific `list`s of results (a `list` of `list`s or "meta-list"). Internally, this "meta-list" is cleaned up (by concatenation) such that only a single slot per type of result specified by the `cv_fun()` is returned (this too is a `list` of the results for each fold). By default, the `combine_results()` helper function is used to combine the individual, fold-specific `list`s of results 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 in the `.combine_control` argument.

5.7 Cross-validation in action

We've waited long enough. Now, let's see `origami` in action! In the next chapter, we will learn how to use cross-validation with the Super Learner algorithm, and how we can

utilize the power of cross-validation to build optimal ensembles of algorithms — going far beyond the application of cross-validation to a single statistical learning method.

5.7.1 Cross-validation with linear regression

First, let's load the relevant `R` packages, set a seed (for reproducibility), and once again load the WASH Benefits example dataset. For illustrative purposes, we'll examine the application of cross-validation to simple linear regression with `origami`, focusing on predicting the weight-for-height Z-score (`whz`) using all of the other available covariates in the dataset. As mentioned before, we will assume the dataset contains only independent and identically distributed units, ignoring the clustering structure imposed by the trial design. For the sake of illustration, we will work with only a subset of the data, removing all observational units with missing covariate data from the analysis-ready dataset. In the prior chapter, we discussed 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 with drop_na(), then pick just the first 500 rows
washb_data <- washb_data %>%
  drop_na() %>%
  slice(1:500)

# specify the outcome and covariates as character vectors
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)	146.4	Food Secu
-1.16	Control	N05265	9	286	male	25	Primary (1-5y)	148.8	Moderatel
-1.05	Control	N08002	9	264	male	25	Primary (1-5y)	152.2	Food Secu
-1.26	Control	N08002	9	252	female	28	Primary (1-5y)	140.2	Food Secu
-0.59	Control	N06531	9	336	female	19	Secondary (¿5y)	150.9	Food Secu
-0.51	Control	N06531	9	304	male	20	Secondary (¿5y)	154.2	Severely F

Let's remind ourselves of the covariates to be used in the prediction step:

```
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, let's fit a simple main-terms linear regression model to the analysis-ready dataset. Here, our goal is to predict the weight-for-height Z-score (`"whz"`, which we assigned to the variable `outcome`) 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 the quality of the model fit on the dataset by comparing the linear model's predictions of the weight-for-height Z-score against the observations of the same in the dataset. This is the well-known, and standard, mean squared error (MSE). We can extract this summary measure from the `lm` model object like so

```
(err <- mean(resid(lm_mod)^2))
[1] 0.8657
```

The MSE estimate is 0.8657, which, from examination of the above, is merely the mean of the squared residuals of the model fit. An important problem arises when we assess the learning algorithm's quality in this way — that is, because we have trained our linear regression model on the complete analysis-ready dataset and then assessed its performance (the MSE) on the same dataset, all of the data is used for both model training and validation. Unfortunately, this simple estimate of the MSE is overly optimistic. Why? The linear regression model is trained on the same dataset used in its evaluation, not unlike reusing problems from a homework assignment in a course examination. Of course, we are generally not interested in how well the algorithm explains variation in the observed dataset; rather, we are interested in how well the explanations provided by the learning algorithm generalize to a target population from which this particular sample is drawn. By using all of the data available to us for training the learning algorithm, we are left unable to honestly evaluate how well the algorithm fits (and, thus, explains) variation at the level of the target population. To resolve this issue, cross-validation allows for a particular procedure (e.g., linear regression) to be implemented over training and validation splits of the dataset, evaluating how well the procedure fits on a holdout (or validation) set. This evaluation of the learning algorithm's quality on data unseen during the training phase provides an honest evaluation of the algorithm's generalization error.

We can easily incorporate cross-validation into our linear regression procedure using `origami`. First, let's define a new function to perform linear regression on a specific partition of the dataset (i.e., 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 quite simple: It merely splits the available data into distinct training and validation sets (using the eponymous functions provided in `origami`), fits the linear model on the training set, and evaluates the quality of the trained linear regression model on the validation set. This is a simple example of what `origami` considers to be a `cv_fun()` — functions for applying a particular routine over an input dataset in cross-validated manner.

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 using `cross_validate()`. 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] 0.8657
```

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 dataset into V subsets, fitting the algorithm on $V - 1$ of the subsets (training) and evaluating on the subset that was held out from fitting (validation). This is repeated such that each holdout subset takes a turn being used for validation. We can easily apply our `cv_lm()` function in this way using `origami`'s `cross_validate()` (note that by default this function 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 V-fold cross-validation with 10 folds (the default), we quickly notice that our previous estimate of the model error (by re-substitution) was a bit optimistic. The honest estimate of the linear regression model's error is larger!

5.7.2 Cross-validation with random forests

To examine `origami` further, let's return to our example analysis using the WASH Benefits dataset. Here, we will write a new `cv_fun()` function. As an example, we will use Breiman's random forest algorithm [Breiman, 2001], implemented in the `randomForest()` function (from the `randomForest` package):

```
# 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)
}
```

The `cv_rf()` function, which cross-validates the training and evaluation of the `randomForest` algorithm, used our previous `cv_lm()` function as a template. For now, individual `cv_fun()`s must be written by hand; however, in future releases of the package,

a wrapper may be made available to support auto-generating `cv_fun`s for use with `origami`.

Below, we use `cross_validate()` to apply our custom `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] 1.027
```

Using V-fold cross-validation with 10 folds, we obtain an honest estimate of the prediction error of this random forest. This is one example of how `origami`'s `cross_validate()` procedure can be generalized to arbitrary estimation techniques, as long as an appropriate `cv_fun()` function is available.

5.7.3 Cross-validation with ARIMA

Cross-validation can also be used for the selection of forecasting models in settings with time-series data. Here, the partitioning scheme mirrors the application of the forecasting model: We'll train the learning algorithm on past observations (either all available or a recent, in time, subset), and then use the fitted model to predict the next (again, in time) few observations. To demonstrate this, we return to the `AirPassengers` dataset, a monthly time-series of passenger air traffic for thousands of travelers.

```
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 (AutoRegressive Integrated Moving Average) model configurations. We can choose among such models by evaluating their forecasting performance. First, we set up an appropriate cross-validation scheme for use with time-series data. Here, we pick the rolling origin cross-validation scheme described above.

```
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 in each training fold. Had we followed the default option, we would have 85 folds to train! Luckily, we can pass the `batch` as an option to `folds_rolling_origin`, telling it to increase the size of the training set by 10 points in each iteration (so that we don't have so many training folds). Since we want to forecast the immediately following time point, the `gap` argument remains at its default of zero.

```
# 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$mse
  fold   arima arima2
1    1   68.21  137.3
2    2  319.68  313.2
3    3  578.35  713.4
4    4  428.69  505.3
5    5  407.33  371.3
6    6  281.82  251.0
7    7  827.56  910.1
8    8 2099.59 2213.1
9    9  398.37  293.4
colMeans(mses$mse[, c("arima", "arima2")])
 arima arima2
 601.1  634.2
```

By applying `cross_validate()` with this `cv_forecasts()` custom function, we find that the ARIMA model with no AR (autoregressive) component seems to be a better fit for this dataset.

5.8 Exercises

5.8.1 Review of Key Concepts

1. Compare and contrast V-fold cross-validation with re-substitution 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 cross-validation relative to:
    a. holdout cross-validation?
    b. leave-one-out cross-validation?
3. Why is V-fold cross-validation inappropriate for use with time-series data?
4. Would you use rolling window or rolling origin cross-validation for non-stationary time-series? Why? ### The Ideas in Action

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

6. Consider the WASH Benefits example dataset discussed in this chapter. How can we incorporate cluster-level information into a cross-validation scheme? How can we implement this strategy with the `origami` package?

5.8.2 Advanced Topics

1. Think about a dataset with a spatial dependence structure, in which the degree of dependence is known such that the groups formed by this dependence structure are clear and where there are no spillover effects. What kind of cross-validation scheme would be appropriate in this case?

2. Continuing from the previous problem, what kind of procedure, and cross-validation scheme, can we use if the spatial dependence is not as clearly defined as in assumptions made in the preceding problem?

3. Consider a classification problem with a large number of predictors and a binary outcome. Your friendly neighborhood statistician proposes the following analysis:

    a. First, screen the predictors, isolating only those covariates that are strongly correlated with the (binary) outcome labels.
    b. Next, train a learning algorithm using only this subset of covariates that are highly correlated with the outcome.
    c. Finally, use cross-validation to estimate the tuning parameters and the performance of the learning algorithm.

    Is this application of cross-validation correct? Why or why not?

6

Super Learning

Rachael Phillips

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

Learning Objectives

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

1. Select a performance metric that is optimized by the true prediction function, or define the true prediction prediction of interest as the optimizer of the performance metric.
2. Assemble a diverse set ("library") of learners to be considered in the super learner. In particular, you should be able to:
    a. Customize a learner by modifying its tuning parameters.
    b. Create variations of the same base learner with different tuning parameter specifications.
    c. Couple screener(s) with learner(s) to create learners that consider as covariates a reduced, screener-selected subset of them.
3. Specify a meta-learner that optimizes the objective function of interest.
4. Justify the library and the meta-learner in terms of the prediction problem at hand, intended use of the analysis in the real world, statistical model, sample size, number of covariates, and outcome prevalence for discrete outcomes.
5. Interpret the fit for a super learner from the table of cross-validated risk estimates and the super learner coefficients.

6.1 Introduction

A common task in data analysis is prediction, or using the observed data to learn a function that takes as input data on covariates/predictors and outputs a predicted value. Occasionally, the scientific question of interest lends itself to causal effect estimation. Even in these scenarios, where prediction is not in the forefront, prediction tasks are embedded in the procedure. For instance, in targeted minimum loss-based estimation (TMLE) for the average treatment effect, predictive modeling is necessary for estimating outcome regressions and propensity scores.

There are various strategies that can be employed to model relationships from data, which we refer to interchangeably as "estimators", "algorithms", and "learners". For some data, algorithms that can pick up on complex relationships among variables are necessary to adequately model it. For other data, parametric regression learners might fit the data reasonably well. It is generally impossible to know in advance which approach will be the best for a given data set and prediction problem.

The Super Learner (SL) solves the issue of selecting an algorithm, as it can consider many of them - from the simplest parametric regressions to the most complex machine learning algorithms (e.g., neural nets, support vector machines, etc). Additionally, it is proven to perform as well as possible (as good as the unknown oracle) in large samples, given the learners specified [van der Laan and Dudoit, 2003, van der Laan et al., 2004, Dudoit and van der Laan, 2005, van der Vaart et al., 2006]. The SL represents an entirely pre-specified, flexible, and theoretically grounded approach for predictive modeling. It has been shown to be adaptive and robust in a variety of applications, even in very small samples. Detailed descriptions outlining the SL procedure are widely available [Polley and van der Laan, 2010, Naimi and Balzer, 2018]. Practical considerations for specifying the SL, including how to specify a rich and diverse library of learners, choose a performance metric for the SL, and specify a cross-validation (CV) scheme, are described in a pre-print article [Phillips et al., 2023]. Here, we focus on introducing `sl3`, the standard `tlverse` software package for SL.

Super Learning with `sl3`:

6.2 How to Fit the Super Learner

In this section, the core functionality for fitting any SL with `sl3` is illustrated. In the sections that follow, additional `sl3` functionality is presented.

Fitting any SL with `sl3` consists of the following three steps:

1. Define the prediction task with `make_sl3_Task`.
2. Instantiate the SL with `Lrnr_sl`.
3. Fit the SL to the task with `train`.

Running example with WASH Benefits dataset

We will use the WASH Benefits Bangladesh study as an example to guide this overview of `sl3`. In this study, we are interested in predicting the child development outcome, weight-for-height z-score, from covariates/predictors, including socio-economic status variables, gestational age, and maternal features. More information on this dataset is described in the "Meet the Data" chapter of the `tlverse` handbook.

Preliminaries

First, we need to load the data and relevant packages into the R session.

Load the data

We will use the `fread` function in the `data.table` R package to load the WASH Benefits example dataset:

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

Next, we will take a peek at the first few rows of our dataset:

```
head(washb_data)
```

whz	tr	fracode	month	aged	sex	momage	momedu	momheight	hfiacat
0.00	Control	N05265	9	268	male	30	Primary (1-5y)	146.4	Food Secu
-1.16	Control	N05265	9	286	male	25	Primary (1-5y)	148.8	Moderatel
-1.05	Control	N08002	9	264	male	25	Primary (1-5y)	152.2	Food Secu
-1.26	Control	N08002	9	252	female	28	Primary (1-5y)	140.2	Food Secu
-0.59	Control	N06531	9	336	female	19	Secondary (¿5y)	150.9	Food Secu
-0.51	Control	N06531	9	304	male	20	Secondary (¿5y)	154.2	Severely F

Install `sl3` software (as needed)

To install any package, we recommend first clearing the R workspace and then restarting the R session. In RStudio, this can be achieved by clicking the tab "Session" then "Clear Workspace", and then clicking "Session" again then "Restart R".

We can install `sl3` using the function `install_github` provided in the `devtools` R package. We are using the development ("devel") version of `sl3` in these materials, so we show how to install that version below.

```
library(devtools)
install_github("tlverse/sl3@devel")
```

Once the R package is installed, we recommend restarting the R session again.

Load `sl3` software

Once `sl3` is installed, we can load it like any other `R` package:

```
library(sl3)
```

1. Define the prediction task with `make_sl3_Task`

The `sl3_Task` object defines the prediction task of interest. Recall that our task in this illustrative example is to use the WASH Benefits Bangladesh example dataset to learn a function of the covariates for predicting weight-for-height Z-score `whz`.

```
# create the task (i.e., use washb_data to predict outcome using covariates)
task <- make_sl3_Task(
  data = washb_data,
```

```
  outcome = "whz",
  covariates = c("tr", "fracode", "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")
)

# let's examine the task
task
An 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
```

The `sl3_Task` keeps track of the roles the variables play in the prediction problem. Additional information relevant to the prediction task (such as observational-level weights, offset, id, CV folds) can also be specified in `make_sl3_Task`. The default CV fold structure in `sl3` is V-fold CV (VFCV) with V=10 folds. If `id` is specified in the task, a clustered V=10 VFCV scheme is considered; if the outcome type is binary or categorical, then a stratified V=10 VFCV scheme is considered. Different CV schemes can be specified by inputting an `origami` folds object, as generated by the `make_folds`

function in the origami R package. For more information, refer to the previous Chapter on cross-validation, or consult documentation on origami's make_folds function (e.g., in RStudio, by loading the origami R package and then inputting "?make_folds" in the Console). For more details on sl3_Task, refer to its documentation (e.g., by inputting "?sl3_Task" in R).

Tip: If you type task$ and then press the tab key (press tab twice if not in RStudio), you can view all of the active and public fields, as well as methods that can be accessed from the task$ object. This $ is like the key to access many internals of an object. In the next section, will see how we can use $ to dig into SL fit objects as well - to obtain predictions from an SL fit or candidate learners, examine an SL fit or its candidates, and summarize an SL fit.

2. Instantiate the Super Learner with `Lrnr_sl`

In order to create Lrnr_sl we need to specify, at the minimum, a set of learners for the SL to consider as candidates. This set of algorithms is also commonly referred to as the "library". We might also specify the meta-learner, which is the algorithm that ensembles the learners (note that this part is optional since there are already defaults set up in sl3). See "Practical considerations for specifying a super learner" for step-by-step guidelines for tailoring the SL specification (including the library and meta-learner(s)) that optimizes the prediction task at hand [Phillips et al., 2023].

Learners have properties that indicate what features they support. We may use the sl3_list_properties() function 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 whz is a continuous outcome, we can identify the learners that support this outcome type with sl3_list_learners():

```
sl3_list_learners(properties = "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_ga"                       "Lrnr_gam"
```

```
[13] "Lrnr_gbm"                          "Lrnr_glm"
[15] "Lrnr_glm_fast"                     "Lrnr_glm_semiparametric"
[17] "Lrnr_glmnet"                       "Lrnr_glmtree"
[19] "Lrnr_grf"                          "Lrnr_gru_keras"
[21] "Lrnr_gts"                          "Lrnr_h2o_glm"
[23] "Lrnr_h2o_grid"                     "Lrnr_hal9001"
[25] "Lrnr_HarmonicReg"                  "Lrnr_hts"
[27] "Lrnr_lightgbm"                     "Lrnr_lstm_keras"
[29] "Lrnr_mean"                         "Lrnr_multiple_ts"
[31] "Lrnr_nnet"                         "Lrnr_nnls"
[33] "Lrnr_optim"                        "Lrnr_pkg_SuperLearner"
[35] "Lrnr_pkg_SuperLearner_method"      "Lrnr_pkg_SuperLearner_screener"
[37] "Lrnr_polspline"                    "Lrnr_randomForest"
[39] "Lrnr_ranger"                       "Lrnr_rpart"
[41] "Lrnr_rugarch"                      "Lrnr_screener_correlation"
[43] "Lrnr_solnp"                        "Lrnr_stratified"
[45] "Lrnr_svm"                          "Lrnr_tsDyn"
[47] "Lrnr_xgboost"
```

Now that we have an idea of some learners, let's instantiate a few of them. Below we instantiate `Lrnr_glm` and `Lrnr_mean`, a main terms generalized linear model (GLM) and a mean model, respectively.

```
lrn_glm <- Lrnr_glm$new()
lrn_mean <- Lrnr_mean$new()
```

For both of the learners created above, we just used the default tuning parameters. We can also customize a learner's tuning parameters to incorporate a diversity of different settings, and consider the same learner with different tuning parameter specifications.

Below, we consider the same base learner, `Lrnr_glmnet` (i.e., GLMs with elastic net regression), and create two different candidates from it: an L2-penalized/ridge regression and an L1-penalized/lasso regression.

```
# penalized regressions:
lrn_ridge <- Lrnr_glmnet$new(alpha = 0)
lrn_lasso <- Lrnr_glmnet$new(alpha = 1)
```

By setting `alpha` in `Lrnr_glmnet` above, we customized this learner's tuning parameter. When we instantiate `Lrnr_hal9001` below we show how multiple tuning parameters (specifically, `max_degree`and `num_knots`) can be modified at the same time.

Let's also instantiate some more learners that do not enforce relationships to be linear or monotonic, which further diversifies the set of candidates to include nonparametric learners.

```
# spline regressions:
lrn_polspline <- Lrnr_polspline$new()
lrn_earth <- Lrnr_earth$new()

# fast highly adaptive lasso (HAL) implementation
lrn_hal <- Lrnr_hal9001$new(max_degree = 2, num_knots = c(3,2), nfolds = 5)

# tree-based methods
lrn_ranger <- Lrnr_ranger$new()
lrn_xgb <- Lrnr_xgboost$new()
```

Let's also include a generalized additive model (GAM) and Bayesian GLM to further diversify the pool that we will consider as candidates in the SL.

```
lrn_gam <- Lrnr_gam$new()
lrn_bayesglm <- Lrnr_bayesglm$new()
```

Now that we've instantiated a set of learners, we need to put them together so the SL can consider them as candidates. In `sl3`, we do this by creating a so-called `Stack` of learners. A `Stack` is created in the same way we created the learners. This is because `Stack` is a learner itself; it has the same interface as all of the other learners. What makes a stack special is that it considers multiple learners at once: it can train them simultaneously, so that their predictions can be combined and/or compared.

```
stack <- Stack$new(
  lrn_glm, lrn_mean, lrn_ridge, lrn_lasso, lrn_polspline, lrn_earth, lrn_hal,
  lrn_ranger, lrn_xgb, lrn_gam, lrn_bayesglm
)
stack
 [1] "Lrnr_glm_TRUE"
 [2] "Lrnr_mean"
 [3] "Lrnr_glmnet_NULL_deviance_10_0_100_TRUE"
 [4] "Lrnr_glmnet_NULL_deviance_10_1_100_TRUE"
 [5] "Lrnr_polspline"
 [6] "Lrnr_earth_2_3_backward_0_1_0_0"
 [7] "Lrnr_hal9001_2_1_c(3, 2)_5"
 [8] "Lrnr_ranger_500_TRUE_none_1"
 [9] "Lrnr_xgboost_20_1"
[10] "Lrnr_gam_NULL_NULL_GCV.Cp"
[11] "Lrnr_bayesglm_TRUE"
```

We can see that the names of the learners in the stack are long. This is because the default naming of a learner in `sl3` is clunky: for each learner, every tuning parameter in `sl3` is contained in the name. In the next section, "Naming Learners", we show a few different ways for the user to name learners as they wish.

Now that we have instantiated a set of learners and stacked them together, we are ready to instantiate the SL. We will use the default meta-learner, which is non-negative least squares (NNLS) regression (`Lrnr_nnls`) for continuous outcomes. For illustrative purposes, we will still go ahead and specify it in the following.

```
sl <- Lrnr_sl$new(learners = stack, metalearner = Lrnr_nnls$new())
```

3. Fit the Super Learner to the prediction task with `train`

The last step for fitting the SL to the prediction task is to call `train` and supply the task. Before we call `train`, we will set a random number generator so the results are reproducible, and we will also time it.

```
start_time <- proc.time() # start time

set.seed(4197)
sl_fit <- sl$train(task = task)

runtime_sl_fit <- proc.time() - start_time # end time - start time = run time
runtime_sl_fit
   user  system elapsed
281.498   6.225 277.831
```

It took 277.8 seconds (4.6 minutes) to fit the SL.

Summary

In this section, the core functionality for fitting any SL with `sl3` was illustrated. This consists of the following three steps:

1. Define the prediction task with `make_sl3_Task`.
2. Instantiate the SL with `Lrnr_sl`.
3. Fit the SL to the task with `train`.

This example was for demonstrative purposes only. See Phillips et al. [2023] for step-by-step guidelines for constructing a SL that is well-specified for the prediction task at hand.

Additional `sl3` Topics:

6.3 Obtaining Predictions

6.3.1 Super learner and candidate learner predictions

We will draw on the fitted SL object from above, `sl_fit`, to obtain the SL's predicted `whz` value for each subject.

```
sl_preds <- sl_fit$predict(task = task)
head(sl_preds)
[1] -0.5719 -0.8717 -0.6881 -0.7342 -0.6308 -0.6596
```

We can also obtain predicted values from a candidate learner in the SL. Below we obtain predictions for the GLM learner.

```
glm_preds <- sl_fit$learner_fits$Lrnr_glm_TRUE$predict(task = task)
head(glm_preds)
[1] -0.7262 -0.9361 -0.7085 -0.6492 -0.7013 -0.8462
```

Note that the predicted values for the SL correspond to so-called "full fits" of the candidate learners, in which the candidates are fit to the entire analytic dataset, i.e., all of the data supplied as `data` to `make_sl3_Task`. Figure 2 in Phillips et al. [2023] provides a visual overview of the SL fitting procedure.

```
# we can also access the candidate learner full fits directly and obtain
# the same "full fit" candidate predictions from there
# (we split this into two lines to avoid overflow)
stack_full_fits <- sl_fit$fit_object$full_fit$learner_fits$Stack$learner_fits
glm_preds_full_fit <- stack_full_fits$Lrnr_glm_TRUE$predict(task)

# check that they are identical
identical(glm_preds, glm_preds_full_fit)
[1] TRUE
```

Below we visualize the observed values for `whz` and predicted `whz` values for SL, GLM and the mean.

```
# table of observed and predicted outcome values and arrange by observed values
df_plot <- data.table(
  Obs = washb_data[["whz"]], SL_Pred = sl_preds, GLM_Pred = glm_preds,
  Mean_Pred = sl_fit$learner_fits$Lrnr_mean$predict(task)
```

```
)
df_plot <- df_plot[order(df_plot$Obs), ]

head(df_plot)
```

Obs	SL_Pred	GLM_Pred	Mean_Pred
-4.67	-1.487	-0.9096	-0.5861
-4.18	-1.170	-0.6391	-0.5861
-4.17	-1.147	-0.8098	-0.5861
-4.03	-1.447	-0.8960	-0.5861
-3.95	-1.579	-1.1952	-0.5861
-3.90	-1.285	-0.9849	-0.5861

```
# melt the table so we can plot observed and predicted values
df_plot$id <- seq(1:nrow(df_plot))
df_plot_melted <- melt(
  df_plot, id.vars = "id",
  measure.vars = c("Obs", "SL_Pred", "GLM_Pred", "Mean_Pred")
)

library(ggplot2)
ggplot(df_plot_melted, aes(id, value, color = variable)) +
  geom_point(size = 0.1) +
  labs(x = "Subjects (ordered by increasing whz)",
       y = "whz") +
  theme(legend.position = "bottom", legend.title = element_blank(),
        axis.text.x = element_blank(), axis.ticks.x = element_blank()) +
  guides(color = guide_legend(override.aes = list(size = 1)))
```

6.3.2 Cross-validated predictions

We can also obtain the cross-validated (CV) predictions for the candidate learners. We can do this in a few different ways.

```
# one way to obtain the CV predictions for the candidate learners
cv_preds_option1 <- sl_fit$fit_object$cv_fit$predict_fold(
  task = task, fold_number = "validation"
)
# another way to obtain the CV predictions for the candidate learners
cv_preds_option2 <- sl_fit$fit_object$cv_fit$predict(task = task)

# we can check that they are identical
identical(cv_preds_option1, cv_preds_option2)
[1] TRUE
```

Figure 6.1: Observed and predicted values for weight-for-height z-score (whz)

```
head(cv_preds_option1)
```

Lrnr_glm_TRUE	Lrnr_mean	Lrnr_glmnet_NULL_deviance_10_0_100_TRUE	Lrnr_glmnet_NULL_dev
-0.7453	-0.5931	-0.6949	
-0.9447	-0.5865	-0.8150	
-0.6494	-0.5931	-0.7004	
-0.6211	-0.5846	-0.6237	
-0.7647	-0.5846	-0.6711	
-0.8873	-0.5763	-0.8106	

predict_fold

Our first option to get CV predictions, `cv_preds_option1`, used the `predict_fold` function to obtain validation set predictions across all folds. This function only exists for learner fits that are cross-validated in `sl3`, like those in `Lrnr_sl`. In addition to supplying `fold_number = "validation"` in `predict_fold`, we can set `fold_number = "full"` to obtain predictions from learners fit to the entire analytic dataset (i.e., all of the data supplied to `make_sl3_Task`). For instance, below we show that `glm_preds` we calculated above can also be obtained by setting `fold_number = "full"`.

```
full_fit_preds <- sl_fit$fit_object$cv_fit$predict_fold(
  task = task, fold_number = "full"
)
glm_full_fit_preds <- full_fit_preds$Lrnr_glm_TRUE
```

```
# check that they are identical
identical(glm_preds, glm_full_fit_preds)
[1] TRUE
```

We can also supply a specific integer between 1 and the number of CV folds to the `fold_number` argument in `predict_fold`; example of this functionality is shown in the next part.

Cross-validated predictions by hand

We can get the CV predictions "by hand", by tapping into each of the folds, and then using the fitted candidate learners (which were trained to the training set for each fold) to predict validation set outcomes (which were not seen in training).

```
##### CV predictions "by hand" #####
# for each fold, i, we obtain validation set predictions:
cv_preds_list <- lapply(seq_along(task$folds), function(i){

  # get validation dataset for fold i:
  v_data <- task$data[task$folds[[i]]$validation_set, ]

  # get observed outcomes in fold i's validation dataset:
  v_outcomes <- v_data[["whz"]]

  # make task (for prediction) using fold i's validation dataset as data,
  # and keeping all else the same:
  v_task <- make_sl3_Task(covariates = task$nodes$covariates, data = v_data)

  # get predicted outcomes for fold i's validation dataset, using candidates
  # trained to fold i's training dataset
  v_preds <- sl_fit$fit_object$cv_fit$predict_fold(
    task = v_task, fold_number = i
  )
  # note: v_preds is a matrix of candidate learner predictions, where the
  # number of rows is the number of observations in fold i's validation dataset
  # and the number of columns is the number of candidate learners (excluding
  # any that might have failed)

  # an identical way to get v_preds, which is used when we calculate the
  # cv risk by hand in a later part of this chapter:
  # v_preds <- sl_fit$fit_object$cv_fit$fit_object$fold_fits[[i]]$predict(
  #   task = v_task
  # )
```

```
  # we will also return the row indices for fold i's validation set, so we
  # can later reorder the CV predictions and make sure they are equal to what
  # we obtained above
  return(list("v_preds" = v_preds, "v_index" = task$folds[[i]]$validation_set))
})

# extract the validation set predictions across all folds
cv_preds_byhand <- do.call(rbind, lapply(cv_preds_list, "[[", "v_preds"))

# extract the indices of validation set observations across all folds
# then reorder cv_preds_byhand to correspond to the ordering in the data
row_index_in_data <- unlist(lapply(cv_preds_list, "[[", "v_index"))
cv_preds_byhand_ordered <- cv_preds_byhand[order(row_index_in_data), ]
# now we can check that they are identical
identical(cv_preds_option1, cv_preds_byhand_ordered)
[1] TRUE
```

6.3.3 Predictions with new data

If we wanted to obtain predicted values for new data then we would need to create a new `sl3_Task` from the new data. Also, the covariates in this new `sl3_Task` must be identical to the covariates in the `sl3_Task` for training. As an example, let's assume we have new covariate data `washb_data_new` for which we want to use the fitted SL to obtain predicted weight-for-height z-score values.

```
# we do not evaluate this code chunk, as 'washb_data_new' does not exist
prediction_task <- make_sl3_Task(
  data = washb_data_new, # assuming we have some new data for predictions
  covariates = c("tr", "fracode", "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")
)
sl_preds_new_task <- sl_fit$predict(task = prediction_task)
```

6.3.4 Counterfactual predictions

Counterfactual predictions are predicted values under an intervention of interest. Recall from above that we can obtain predicted values for new data by creating a `sl3_Task` with the new data whose covariates match the set considered for training. As an

example that draws on the WASH Benefits Bangladesh study, suppose we would like to obtain predictions for every subject's weight-for-height z-score (`whz`) outcome under an intervention on treatment (`tr`) that sets it to the nutrition, water, sanitation, and handwashing regime.

First we need to create a copy of the dataset, and then we can intervene on `tr` in the copied dataset, create a new `sl3_Task` using the copied data and the same covariates as the training task, and finally obtain predictions from the fitted SL (which we named `sl_fit` in the previous section).

```
### 1. Copy data
tr_intervention_data <- data.table::copy(washb_data)

### 2. Define intervention in copied dataset
tr_intervention <- rep("Nutrition + WSH", nrow(washb_data))
# NOTE: When we intervene on a categorical variable (such as "tr"), we need to
#       define the intervention as a categorical variable (ie a factor).
#       Also, even though not all levels of the factor will be represented in
#       the intervention, we still need this factor to reflect all of the
#       levels that are present in the observed data
tr_levels <- levels(washb_data[["tr"]])
tr_levels
[1] "Control"         "Handwashing"     "Nutrition"       "Nutrition + WSH"
[5] "Sanitation"      "WSH"             "Water"
tr_intervention <- factor(tr_intervention, levels = tr_levels)
tr_intervention_data[,"tr" := tr_intervention, ]

### 3. Create a new sl3_Task
# note that we do not need to specify the outcome in this new task since we are
# only using it to obtain predictions
tr_intervention_task <- make_sl3_Task(
  data = tr_intervention_data,
  covariates = c("tr", "fracode", "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")
)
### 4. Get predicted values under intervention of interest
# SL predictions of what "whz" would have been had everyone received "tr"
# equal to "Nutrition + WSH"
counterfactual_pred <- sl_fit$predict(tr_intervention_task)
```

Note that this type of intervention, where every subject receives the same intervention,

is referred to as “static”. Interventions that vary depending on the characteristics of the subject are referred to as “dynamic”. For instance, we might consider an intervention that sets the treatment to the desired (nutrition, water, sanitation, and handwashing) regime if the subject has a refridgerator, and a nutrition-omitted (water, sanitation, and handwashing) regime otherwise.

```
dynamic_tr_intervention_data <- data.table::copy(washb_data)

dynamic_tr_intervention <- ifelse(
  washb_data[["asset_refrig"]] == 1, "Nutrition + WSH", "WSH"
)
dynamic_tr_intervention <- factor(dynamic_tr_intervention, levels = tr_levels)
dynamic_tr_intervention_data[,"tr" := dynamic_tr_intervention, ]

dynamic_tr_intervention_task <- make_sl3_Task(
  data = dynamic_tr_intervention_data,
  covariates = c("tr", "fracode", "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")
)
### 4. Get predicted values under intervention of interest
# SL predictions of what "whz" would have been had every subject received "tr"
# equal to "Nutrition + WSH" if they had a fridge and "WSH" if they didn't have
# a fridge
counterfactual_pred <- sl_fit$predict(dynamic_tr_intervention_task)
```

6.4 Summarizing Super Learner Fits

6.4.1 Super Learner coefficients / fitted meta-learner summary

We can see how the meta-learner created a function of the learners in a few ways. In our illustrative example, we considered the default, NNLS meta-learner for continuous outcomes. For meta-learners that simply learn a weighted combination, we can examine their coefficients.

```
round(sl_fit$coefficients, 3)
```

```
Lrnr_glm_TRUE                                    Lrnr_mean
            0.000                                        0.000
```

```
Lrnr_glmnet_NULL_deviance_10_0_100_TRUE Lrnr_glmnet_NULL_deviance_10_1_100_TRUE
                                  0.096                                   0.000
                          Lrnr_polspline         Lrnr_earth_2_3_backward_0_1_0_0
                                  0.168                                   0.399
              Lrnr_hal9001_2_1_c(3, 2)_5             Lrnr_ranger_500_TRUE_none_1
                                  0.000                                   0.337
                     Lrnr_xgboost_20_1               Lrnr_gam_NULL_NULL_GCV.Cp
                                  0.000                                   0.000
                   Lrnr_bayesglm_TRUE
                                  0.000
```

We can also examine the coefficients by directly accessing the meta-learner's fit object.

```
metalrnr_fit <- sl_fit$fit_object$cv_meta_fit$fit_object
round(metalrnr_fit$coefficients, 3)
```

```
                           Lrnr_glm_TRUE                                Lrnr_mean
                                   0.000                                    0.000
Lrnr_glmnet_NULL_deviance_10_0_100_TRUE Lrnr_glmnet_NULL_deviance_10_1_100_TRUE
                                   0.096                                    0.000
                           Lrnr_polspline         Lrnr_earth_2_3_backward_0_1_0_0
                                   0.168                                    0.399
               Lrnr_hal9001_2_1_c(3, 2)_5             Lrnr_ranger_500_TRUE_none_1
                                   0.000                                    0.337
                      Lrnr_xgboost_20_1               Lrnr_gam_NULL_NULL_GCV.Cp
                                   0.000                                    0.000
                    Lrnr_bayesglm_TRUE
                                   0.000
```

Direct access to the meta-learner fit object is also handy for more complex meta-learners (e.g., non-parametric meta-learners) that are not defined by a simple set of main terms regression coefficients.

6.4.2 Cross-validated predictive performance

We can obtain a table of the cross-validated (CV) predictive performance, i.e., the CV risk, for each learner included in the SL. Below, we use the squared error loss for the evaluation function, which equates to the mean squared error (MSE) as the metric to summarize predictive performance. The reason why we use the MSE is because it is a valid metric for estimating the conditional mean, which is what we're learning the prediction function for in the WASH Benefits example. For more information on selecting an appropriate performance metric, see Phillips et al. [2023].

```
cv_risk_table <- sl_fit$cv_risk(eval_fun = loss_squared_error)
```

```
cv_risk_table[,c(1:3)]
```

learner	coefficients	MSE
Lrnr_glm_TRUE	0.0000	1.022
Lrnr_mean	0.0000	1.065
Lrnr_glmnet_NULL_deviance_10_0_100_TRUE	0.0957	1.017
Lrnr_glmnet_NULL_deviance_10_1_100_TRUE	0.0000	1.015
Lrnr_polspline	0.1678	1.016
Lrnr_earth_2_3_backward_0_1_0_0	0.3993	1.013
Lrnr_hal9001_2_1_c(3, 2)_5	0.0000	1.018
Lrnr_ranger_500_TRUE_none_1	0.3372	1.014
Lrnr_xgboost_20_1	0.0000	1.079
Lrnr_gam_NULL_NULL_GCV.Cp	0.0000	1.024
Lrnr_bayesglm_TRUE	0.0000	1.022

Cross-validated predictive performance by hand

Similar to how we got the CV predictions "by hand", we can also calculate the CV performance/risk in a way that exposes the procedure. Specifically, this is done by tapping into each of the folds, and then using the fitted candidate learners (which were trained to the training set for each fold) to predict validation set outcomes (which were not seen in training) and then measure the predictive performance (i.e., risk). Each candidate learner's fold-specific risk is then averaged across all folds to obtain the CV risk. The function `cv_risk` does all of this internally and we show how to do it by hand below, which can be helpful for understanding the CV risk and how it is calculated.

```
##### CV risk "by hand" #####
# for each fold, i, we obtain predictive performance/risk for each candidate:
cv_risks_list <- lapply(seq_along(task$folds), function(i){

  # get validation dataset for fold i:
  v_data <- task$data[task$folds[[i]]$validation_set, ]

  # get observed outcomes in fold i's validation dataset:
  v_outcomes <- v_data[["whz"]]

  # make task (for prediction) using fold i's validation dataset as data,
  # and keeping all else the same:
  v_task <- make_sl3_Task(covariates = task$nodes$covariates, data = v_data)

  # get predicted outcomes for fold i's validation dataset, using candidates
  # trained to fold i's training dataset
```

```
  v_preds <- sl_fit$fit_object$cv_fit$fit_object$fold_fits[[i]]$predict(v_task)
  # note: v_preds is a matrix of candidate learner predictions, where the
  # number of rows is the number of observations in fold i's validation dataset
  # and the number of columns is the number of candidate learners (excluding
  # any that might have failed)

  # calculate predictive performance for fold i for each candidate
  eval_function <- loss_squared_error # valid for estimation of conditional mean
  v_losses <- apply(v_preds, 2, eval_function, v_outcomes)
  cv_risks <- colMeans(v_losses)
  return(cv_risks)
})
# average the predictive performance across all folds for each candidate
cv_risks_byhand <- colMeans(do.call(rbind, cv_risks_list))
cv_risk_table_byhand <- data.table(
  learner = names(cv_risks_byhand), MSE = cv_risks_byhand
)
# check that the CV risks are identical when calculated by hand and function
# (ignoring small differences by rounding to the fourth decimal place)
identical(
  round(cv_risk_table_byhand$MSE,4), round(as.numeric(cv_risk_table$MSE),4)
)
[1] TRUE
```

6.4.3 Cross-validated Super Learner

We can see from the CV risk table above that the SL is not listed. This is because we do not have a CV risk for the SL unless we cross-validate it or include it as a candidate in another SL; the latter is shown in the next subsection. Below, we show how to obtain a CV risk estimate for the SL using function `cv_sl`. Like before when we called `sl$train`, we will set a random number generator so the results are reproducible, and we will also time this.

```
start_time <- proc.time()

set.seed(569)
cv_sl_fit <- cv_sl(lrnr_sl = sl_fit, task = task, eval_fun = loss_squared_error)

runtime_cv_sl_fit <- proc.time() - start_time
runtime_cv_sl_fit

   user  system elapsed
 2792.6   159.6  3051.4
```

It took 3051.4 seconds (50.9 minutes) to fit the CV SL.

```
cv_sl_fit$cv_risk[,c(1:3)]
```

learner	MSE	se
Lrnr_glm_TRUE	1.022	0.0240
Lrnr_mean	1.065	0.0250
Lrnr_glmnet_NULL_deviance_10_0_100_TRUE	1.017	0.0237
Lrnr_glmnet_NULL_deviance_10_1_100_TRUE	1.014	0.0236
Lrnr_polspline	1.016	0.0237
Lrnr_earth_2_3_backward_0_1_0_0	1.013	0.0235
Lrnr_hal9001_2_1_c(3, 2)_5	1.018	0.0237
Lrnr_ranger_500_TRUE_none_1	1.014	0.0236
Lrnr_xgboost_20_1	1.079	0.0248
Lrnr_gam_NULL_NULL_GCV.Cp	1.024	0.0239
Lrnr_bayesglm_TRUE	1.022	0.0240
SuperLearner	1.007	0.0234

The CV risk of the SL is 0.0234, which is lower than all of the candidates' CV risks.

We can see how the SL fits varied across the folds by the coefficients for the SL on each fold.

```
round(cv_sl_fit$coef, 3)
```

fold	Lrnr_glm_TRUE	Lrnr_mean	Lrnr_glmnet_NULL_deviance_10_0_100_TRUE	Lrnr_glmnet_NUL
1	0.000	0	0.047	
2	0.000	0	0.000	
3	0.000	0	0.030	
4	0.050	0	0.000	
5	0.000	0	0.075	
6	0.025	0	0.248	
7	0.000	0	0.000	
8	0.189	0	0.007	
9	0.113	0	0.000	
10	0.000	0	0.000	

6.4.4 Revere-cross-validated predictive performance of Super Learner

We can also use so-called "revere", to obtain a partial CV risk for the SL, where the SL candidate learner fits are cross-validated but the meta-learner fit is not. It takes

essentially no extra time to calculate a revere-CV performance/risk estimate of the SL, since we already have the CV fits of the candidates. This isn't to say that revere-CV SL performance can replace that obtained from actual CV SL. Revere can be used to very quickly examine an approximate lower bound on the SL's CV risk *when the meta-learner is a simple model*, like NNLS. We can output the revere-based CV risk estimate by setting `get_sl_revere_risk = TRUE` in `cv_risk`.

```
cv_risk_w_sl_revere <- sl_fit$cv_risk(
  eval_fun = loss_squared_error, get_sl_revere_risk = TRUE
)

cv_risk_w_sl_revere[,c(1:3)]
```

learner	coefficients	MSE
Lrnr_glm_TRUE	0.0000	1.022
Lrnr_mean	0.0000	1.065
Lrnr_glmnet_NULL_deviance_10_0_100_TRUE	0.0957	1.017
Lrnr_glmnet_NULL_deviance_10_1_100_TRUE	0.0000	1.015
Lrnr_polspline	0.1678	1.016
Lrnr_earth_2_3_backward_0_1_0_0	0.3993	1.013
Lrnr_hal9001_2_1_c(3, 2)_5	0.0000	1.018
Lrnr_ranger_500_TRUE_none_1	0.3372	1.014
Lrnr_xgboost_20_1	0.0000	1.079
Lrnr_gam_NULL_NULL_GCV.Cp	0.0000	1.024
Lrnr_bayesglm_TRUE	0.0000	1.022
SuperLearner	NA	1.003

Revere-cross-validated predictive performance of Super Learner by hand

We show how to calculate the revere-CV predictive performance/risk of the SL by hand below, as this might be helpful for understanding revere and how it can be used to obtain a partial CV performance/risk estimate for the SL.

```
##### revere-based risk "by hand" #####
# for each fold, i, we obtain predictive performance/risk for the SL
sl_revere_risk_list <- lapply(seq_along(task$folds), function(i){
  # get validation dataset for fold i:
  v_data <- task$data[task$folds[[i]]$validation_set, ]

  # get observed outcomes in fold i's validation dataset:
  v_outcomes <- v_data[["whz"]]
```

```
  # make task (for prediction) using fold i's validation dataset as data,
  # and keeping all else the same:
  v_task <- make_sl3_Task(
    covariates = task$nodes$covariates, data = v_data
  )

  # get predicted outcomes for fold i's validation dataset, using candidates
  # trained to fold i's training dataset
  v_preds <- sl_fit$fit_object$cv_fit$fit_object$fold_fits[[i]]$predict(v_task)

  # make a metalevel task (for prediction with sl):
  v_meta_task <- make_sl3_Task(
    covariates = sl_fit$fit_object$cv_meta_task$nodes$covariates,
    data = v_preds
  )

  # get predicted outcomes for fold i's metalevel dataset, using the fitted
  # metalearner, cv_meta_fit
  sl_revere_v_preds <- sl_fit$fit_object$cv_meta_fit$predict(task=v_meta_task)
  # note: cv_meta_fit was trained on the metalevel dataset, which contains the
  # candidates' cv predictions and validation dataset outcomes across ALL folds,
  # so cv_meta_fit has already seen fold i's validation dataset outcomes.

  # calculate predictive performance for fold i for the SL
  eval_function <- loss_squared_error # valid for estimation of conditional mean
  # note: by evaluating the predictive performance of the SL using outcomes
  # that were already seen by the metalearner, this is not a cross-validated
  # measure of predictive performance for the SL.
  sl_revere_v_loss <- eval_function(
    pred = sl_revere_v_preds, observed = v_outcomes
  )
  sl_revere_v_risk <- mean(sl_revere_v_loss)
  return(sl_revere_v_risk)
})
# average the predictive performance across all folds for the SL
sl_revere_risk_byhand <- mean(unlist(sl_revere_risk_list))
sl_revere_risk_byhand
[1] 1.003

# check that our calculation by hand equals what is output in cv_risk_table_revere
sl_revere_risk <- as.numeric(cv_risk_w_sl_revere[learner=="SuperLearner","MSE"])
sl_revere_risk
[1] 1.003
```

The reason why this is not a fully cross-validated risk estimate is because the `cv_meta_fit` object above (which is the trained meta-learner), was previously fit to the *entire* matrix

of CV predictions from *every* fold (i.e., the meta-level dataset; see Figure 2 in Phillips et al. [2023] for more detail). This is why revere-based risks are not a true CV risk. If the meta-learner is not a simple regression function, and instead a more flexible learner (e.g., random forest) is used as the meta-learner, then the revere-CV risk estimate of the resulting SL will be a worse approximation of the CV risk estimate. This is because more flexible learners are more likely to overfit. When simple parametric regressions are used as a meta-learner, like what we considered in our SL (NNLS with `Lrnr_nnls`), and like all of the default meta-learners in `sl3`, then the revere-CV risk is a quick way to examine an approximation of the CV risk estimate of the SL and it can thought of as a ballpark lower bound on it. This idea holds in our example; that is, with the simple NNLS meta-learner the revere risk estimate of the SL (1.0033) is very close to, and slightly lower than, the CV risk estimate for the SL (1.0067).

6.5 Discrete Super Learner

From the glossary (Table 1) entry for discrete SL (dSL) in Phillips et al. [2023], the dSL is "a SL that uses a winner-take-all meta-learner called the cross-validated selector. The dSL is therefore identical to the candidate with the best cross-validated performance; its predictions will be the same as this candidate's predictions". The cross-validated selector is `Lrnr_cv_selector` in `sl3` (see `Lrnr_cv_selector` documentation for more detail) and a dSL is instantiated in `sl3` by using `Lrnr_cv_selector` as the meta-learner in `Lrnr_sl`.

```
cv_selector <- Lrnr_cv_selector$new(eval_function = loss_squared_error)
dSL <- Lrnr_sl$new(learners = stack, metalearner = cv_selector)
```

Just like before, we use the learner's `train` method to fit it to the prediction task.

```
set.seed(4197)
dSL_fit <- dSL$train(task)
```

Following from subsection "Summarizing Super Learner Fits" above, we can see how the `Lrnr_cv_selector` meta-learner created a function of the candidates.

```
round(dSL_fit$coefficients, 3)
```

```
                            Lrnr_glm_TRUE                                      Lrnr_mean
                                        0                                               0
Lrnr_glmnet_NULL_deviance_10_0_100_TRUE Lrnr_glmnet_NULL_deviance_10_1_100_TRUE
                                        0                                               0
                           Lrnr_polspline         Lrnr_earth_2_3_backward_0_1_0_0
```

```
                                0                                 1
Lrnr_hal9001_2_1_c(3, 2)_5      0      Lrnr_ranger_500_TRUE_none_1      0
          Lrnr_xgboost_20_1     0        Lrnr_gam_NULL_NULL_GCV.Cp      0
        Lrnr_bayesglm_TRUE      0
```

We can also examine the CV risk of the candidates alongside the coefficients:

```
dSL_cv_risk_table <- dSL_fit$cv_risk(eval_fun = loss_squared_error)

dSL_cv_risk_table[,c(1:3)]
```

learner	coefficients	MSE
Lrnr_glm_TRUE	0	1.022
Lrnr_mean	0	1.065
Lrnr_glmnet_NULL_deviance_10_0_100_TRUE	0	1.017
Lrnr_glmnet_NULL_deviance_10_1_100_TRUE	0	1.014
Lrnr_polspline	0	1.016
Lrnr_earth_2_3_backward_0_1_0_0	1	1.013
Lrnr_hal9001_2_1_c(3, 2)_5	0	1.018
Lrnr_ranger_500_TRUE_none_1	0	1.013
Lrnr_xgboost_20_1	0	1.079
Lrnr_gam_NULL_NULL_GCV.Cp	0	1.024
Lrnr_bayesglm_TRUE	0	1.022

The multivariate adaptive splines regression candidate (`Lrnr_earth`) has the lowest CV risk. Indeed, our winner-take-all meta-learner `Lrnr_cv_selector` gave it a weight of one and all others zero weight; the resulting dSL will be defined by this weighted combination, i.e., `dSL_fit` will be identical to the full fit `Lrnr_earth`. We verify that the `dSL_fit`'s predictions are identical to `Lrnr_earth`'s below.

```
dSL_pred <- dSL_fit$predict(task)
earth_pred <- dSL_fit$learner_fits$Lrnr_earth_2_3_backward_0_1_0_0$predict(task)
identical(dSL_pred, earth_pred)
[1] TRUE
```

6.5.1 Including ensemble Super Learner(s) as candidate(s) in discrete Super Learner

We recommend using CV to evaluate the predictive performance of the SL. We showed how to do this with `cv_sl` above. We have also seen that when we include a learner as

a candidate in the SL (in `sl3` terms, when we include a learner in the `Stack` passed to `Lrnr_sl` as `learners`), we are able to examine its CV risk. Also, when we use the dSL, the candidate that achieved the lowest CV risk defines the resulting SL. We therefore can use the dSL automate a procedure for obtaining a final SL that represents the candidate with the best cross-validated predictive performance. When the ensemble SL (eSL) and its candidate learners are considered in a dSL as candidates, the eSL's CV performance can be compared to that from the learners from which it was constructed, and the final SL will be the candidate that achieved the lowest CV risk. From the glossary (Table 1) entry for eSL in Phillips et al. [2023], an eSL is "a SL that uses any parametric or non-parametric algorithm as its meta-learner. Therefore, the eSL is defined by a combination of multiple candidates; its predictions are defined by a combination of multiple candidates' predictions." In the following, we show how to include the eSL, and multiple eSLs, as candidates in the dSL.

Recall the SL object, `sl`, defined in section 2:

```
# in the section 2 we defined Lrnr_sl as
# sl <- Lrnr_sl$new(learners = stack, metalearner = Lrnr_nnls$new())
```

`sl` is an eSL since it used NNLS as the meta-learner. We rename `sl` to `eSL_metaNNLS` below to clarify that this is an eSL that uses NNLS as its meta-learner. Note that the candidate learners in this eSL are those passed to the `learners` argument, i.e., `stack`.

```
# let's rename it to clarify that this is an eSL that uses NNLS as meta-learner
eSL_metaNNLS <- sl
```

To consider the `eSL_metaNNLS` as an additional candidate in `stack`, we can create a new stack that includes the original candidate learners and the eSL.

```
stack_with_eSL <- Stack$new(stack, eSL_metaNNLS)
```

To instantiate the dSL that considers as its candidates `eSL_metaNNLS` and the individual learners from which `eSL_metaNNLS` was constructed, we define a new `Lrnr_sl` that considers `stack_with_eSL` as candidates and `Lrnr_cv_selector` as the meta-learner.

```
cv_selector <- Lrnr_cv_selector$new(eval_function = loss_squared_error)
dSL <- Lrnr_sl$new(learners = stack_with_eSL, metalearner = cv_selector)
```

When we include an eSL as a candidate in the dSL, this allows the eSL's CV performance to be compared to that from the other learners from which it was constructed. This is similar to calling CV SL, `cv_sl`, above. The difference between including the eSL as a candidate in the dSL and calling `cv_sl` is that the former automates a procedure for the final SL to be the learner that achieved the best CV predictive performance, i.e., lowest CV risk. If the eSL outperforms any other candidate, the dSL will end up selecting it and the

resulting SL will be the eSL. As mentioned in Phillips et al. [2023], "another advantage of this approach is that multiple eSLs that use more flexible meta-learner methods (e.g., non-parametric machine learning algorithms like HAL) can be evaluated simultaneously."

Below, we show how multiple eSLs can be included as candidates in a dSL:

```
# instantiate more eSLs
eSL_metaNNLSconvex <- Lrnr_sl$new(
  learners = stack, metalearner = Lrnr_nnls$new(convex = TRUE)
)
eSL_metaLasso <- Lrnr_sl$new(learners = stack, metalearner = lrn_lasso)
eSL_metaEarth <- Lrnr_sl$new(learners = stack, metalearner = lrn_earth)
eSL_metaRanger <- Lrnr_sl$new(learners = stack, metalearner = lrn_ranger)
eSL_metaHAL <- Lrnr_sl$new(learners = stack, metalearner = lrn_hal)
# adding the eSLs to the stack that defined them
stack_with_eSLs <- Stack$new(
  stack, eSL_metaNNLS, eSL_metaNNLSconvex, eSL_metaLasso, eSL_metaEarth,
  eSL_metaRanger, eSL_metaHAL
)
# specify dSL
dSL <- Lrnr_sl$new(learners = stack_with_eSLs, metalearner = cv_selector)
```

We included as candidates in the dSL:

1. the same eSL as before, `eSL_metaNNLS`;
2. the learners considered as candidates in (1);
3. an eSL that considered the same candidate learners as (1) and a convex combination-constrained NNLS as the meta-learner;
4. an eSL that considered the same candidate learners as (1) and a lasso meta-learner, using `lrn_lasso` which was instantiated in section 2;
5. an eSL that considered the same candidate learners as (1) and a multivariate adaptive regression splines (earth) meta-learner, using `lrn_earth` which was instantiated in section 2;
6. an eSL that considered the same candidate learners as (1) and a ranger meta-learner, using `lrn_ranger` which was instantiated in section 2; and
7. an eSL that considered the same candidate learners as (1) and a HAL meta-learner, using `lrn_hal` which was instantiated in section 2.

Running this many eSLs in the dSL is currently very computationally intensive in `sl3`, as it is akin to running cross-validated SL for each eSL. Parallel programming (reviewed below) is recommended for training learners that are computationally intensive, like the `dSL` defined above. That is, a parallel processing scheme should be defined before calling `dSL$train(task)` in order to speed up the run time.

6.6 Parallel Processing

It's straightforward to take advantage of `sl3`'s built-in parallel processing support, which draws on the `future` R package, which provides a lightweight, unified Future API for sequential and parallel processing of R expressions via futures. From the `future` package documentation: "This package implements sequential, multicore, multisession, and cluster futures. With these, R expressions can be evaluated on the local machine, in parallel a set of local machines, or distributed on a mix of local and remote machines. Extensions to this package implement additional backends for processing futures via compute cluster schedulers, etc. Because of its unified API, there is no need to modify any code in order switch from sequential on the local machine to, say, distributed processing on a remote compute cluster. Another strength of this package is that global variables and functions are automatically identified and exported as needed, making it straightforward to tweak existing code to make use of futures."

To use `future` with `sl3`, you can simply choose a futures `plan()`, as shown below.

```
# let's load the future package and set n-1 cores for parallel processing
library(future)
ncores <- availableCores()-1
ncores
system
     1
plan(multicore, workers = ncores)
# now, let's re-train sl in parallel for demonstrative purposes
# we will also set a stopwatch so we can see how long this takes
start_time <- proc.time()

set.seed(4197)
sl_fit_parallel <- sl$train(task)

runtime_sl_fit_parallel <- proc.time() - start_time
runtime_sl_fit_parallel
   user  system elapsed
276.429   5.703 272.591
```

6.7 Default Data Pre-processing

In `sl3` it is required that the analytic dataset (i.e., the dataset consisting of observations on an outcome and covariates) does not contain any missing values, and it does not contain character and factor covariates. In this subsection, we review the default functionality in `sl3` that takes care of this internally; specifically, this data pre-processing occurs when `make_sl3_Task` is called.

Users can also perform any pre-processing before creating the `sl3_Task` (as needed) to bypass the default functionality discussed in the following. See Phillips et al. [2023], section "Preliminaries: Analytic dataset pre-processing" for more information and general guidelines to follow for pre-processing of the analytic dataset, including considerations for pre-processing in high dimensional settings.

Recall that the `sl3_Task` object defines the prediction task of interest. Our task in the illustrative example from above was to use the WASH Benefits Bangladesh data to learn a function of the covariates for predicting weight-for-height Z-score `whz`. For more details on `sl3_Task`, refer to the documentation (e.g., by inputting "?sl3_Task" in R). We will instantiate the task in order to examine the pre-processing of `washb_data`.

```
# create the task (i.e., use washb_data to predict outcome using covariates)
task <- make_sl3_Task(
  data = washb_data,
  outcome = "whz",
  covariates = c("tr", "fracode", "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")
)
Warning in process_data(data, nodes, column_names = column_names, flag = flag,
: Imputing missing values and adding missingness indicators for the following
covariates with missing values: momage, momheight. See documentation of the
process_data function for details.
```

6.7.1 Imputation and missingness indicators

Notice the warning that appeared when we created the task above. (We muted this warning when we created the task in the previous section). This warning states that missing

covariate data was detected and imputed. For each covariate column with missing values, `sl3` uses the median to impute missing continuous covariates, and the mode to impute discrete (binary and categorical) covariates.

Also, for each covariate with missing values, an additional column indicating whether the value was imputed is incorporated. The so-called "missingness indicator" covariates can be helpful, as the pattern of covariate missingness might be informative for predicting the outcome.

Users are free to handle missingness in their covariate data before creating the sl3 task. In any case, we do recommend the inclusion of the missingness indicator as a covariate. Let's examine this in greater detail for completeness. It's also easier to see what's going on here by examining it with an example.

First, let's examine the missingness in the data:

```
# which columns have missing values, and how many observations are missing?
colSums(is.na(washb_data))
```

```
           whz             tr        fracode          month           aged
             0              0              0              0              0
           sex         momage         momedu      momheight        hfiacat
             0             18              0             31              0
         Nlt18          Ncomp         watmin           elec          floor
             0              0              0              0              0
         walls           roof asset_wardrobe    asset_table    asset_chair
             0              0              0              0              0
    asset_khat   asset_chouki       asset_tv   asset_refrig     asset_bike
             0              0              0              0              0
    asset_moto  asset_sewmach   asset_mobile
             0              0              0
```

We can see that covariates `momage` and `momheight` have missing observations. Let's check out a few rows in the data with missing values:

```
some_rows_with_missingness <- which(!complete.cases(washb_data))[31:33]
# note: we chose 31:33 because missingness in momage & momheight is there
washb_data[some_rows_with_missingness, c("momage", "momheight")]
```

```
   momage momheight
1:     NA     153.2
2:     17        NA
3:     23        NA
```

When we called `make_sl3_Task` using `washb_data` with missing covariate values, `momage` and `momheight` were imputed with their respective medians (since they are continuous), and a missingness indicator (denoted by prefix "delta_") was added for each of them. See below:

```
task$data[some_rows_with_missingness,
          c("momage", "momheight", "delta_momage", "delta_momheight")]
   momage momheight delta_momage delta_momheight
1:     23     153.2            0               1
2:     17     150.6            1               0
3:     23     150.6            1               0
colSums(is.na(task$data))
            tr         fracode           month            aged             sex
             0               0               0               0               0
        momage          momedu       momheight         hfiacat            Nlt18
             0               0               0               0               0
         Ncomp          watmin            elec           floor           walls
             0               0               0               0               0
          roof  asset_wardrobe     asset_table     asset_chair      asset_khat
             0               0               0               0               0
  asset_chouki        asset_tv    asset_refrig      asset_bike      asset_moto
             0               0               0               0               0
 asset_sewmach    asset_mobile    delta_momage delta_momheight             whz
             0               0               0               0               0
```

Indeed, we can see that `washb_task$data` has no missing values. The missingness indicators take a value of 0 when the observation *was not* in the original data and a value of 1 when the observation *was* in the original data.

If the data supplied to `make_sl3_Task` contains missing outcome values, then an error will be thrown. Missing outcomes in the data can easily be dropped when the task is created, by setting `drop_missing_outcome = TRUE`. In general, we do not recommend dropping missing outcomes during data pre-processing, unless the problem of interest is purely prediction. This is because complete case analyses are generally biased; it is typically unrealistic to assume the missingness is completely random and therefore unsafe to just drop the observations with missing outcomes. For instance, in the estimation of estimands that admit Targeted Minimum Loss-based Estimators (i.e., pathwise differentiable estimands, including most parameters arising in causal inference that do not violate positivity, and those reviewed in the following chapters), the missingness that should be reflected in the expression of the question of interest (e.g., what would have been the average effect of treatment with Drug A compared to standard of care under no loss to follow-up) is also incorporated in the estimation procedure. That is, the probability of loss to follow-up is a prediction function that is approximated (e.g., with SL) and incorporated that in the estimation of the target parameter and the inference / uncertainty quantification.

6.7.2 Character and categorical covariates

First any character covariates are converted to factors. Then all factor covariates are one-hot encoded, i.e., the levels of a factor become a set of binary indicators. For example, the factor `cats` and it's one-hot encoding are shown below:

```
cats <- c("calico", "tabby", "cow", "ragdoll", "mancoon", "dwarf", "calico")
cats <- factor(cats)
cats_onehot <- factor_to_indicators(cats)
cats_onehot
     cow dwarf mancoon ragdoll tabby
[1,]   0     0       0       0     0
[2,]   0     0       0       0     1
[3,]   1     0       0       0     0
[4,]   0     0       0       1     0
[5,]   0     0       1       0     0
[6,]   0     1       0       0     0
[7,]   0     0       0       0     0
```

The second value for `cats` was "tabby" so the second row of `cats_onehot` has value 1 under tabby. Every level of `cats` except for one is represented in the `cats_onehot` table. The first and last `cats` are "calico" so the first and last rows of `cats_onehot` are zero across all columns, to denote this level that does not appear explicitly in the table.

The learners in `sl3` are trained to the object `X` in the task, or a sample of `X` for learners that use CV. Let's check out the first six rows of our task's `X` object:

```
head(task$X)
```

tr.Handwashing	tr.Nutrition	tr.Nutrition...WSH	tr.Sanitation	tr.WSH	tr.Water	fracode.N04681	fracode.I
0	0	0	0	0	0	0	
0	0	0	0	0	0	0	
0	0	0	0	0	0	0	
0	0	0	0	0	0	0	
0	0	0	0	0	0	0	
0	0	0	0	0	0	0	

We can see that any character columns in the WASH Benefits dataset were converted to factors and all factors (`tr`, `momedu`, `hfiacat` and `fracode`) were one-hot encoded. We can also see that the missingness indicators reviewed above are the last two columns in `task$X`: `delta_momage` and `delta_momage`. The imputed `momage` and `momheight` are also in the task's `X` object.

6.8 Learner Documentation

Documentation for the learners and some of their tuning parameters can be found in the R session (e.g., to see `Lrnr_glmnet`'s parameters, one could type "?Lrnr_glmnet" in RStudio's R console) or online at the `sl3` Learners Reference. All of the learners in `sl3` are simply wrappers around existing functions from other software packages in R. For example, `sl3`'s `Lrnr_xgboost` is a learner in `sl3` for fitting the XGBoost (eXtreme Gradient Boosting) algorithm. As described in the `Lrnr_xgboost` documentation, "this learner provides fitting procedures for `xgboost` models, using the `xgboost` package, via `xgb.train`". In general, the documentation in `sl3` for a learner refers the reader to the original function and package that `sl3` has wrapped a learner around. With that in mind, the `sl3` learner documentation is a good first place to look up any learner, as it will show us exactly which package and function the learner is based on. However, any thorough investigation of a learner (such as a detailed explanation of all tuning parameters or how it models the data) typically involves referencing the original package. Continuing the example from above, this means that, while some information will be provided in `Lrnr_xgboost` documentation, such as learning that `Lrnr_xgboost` uses the `xgboost` package's `xgb.train` function, the deepest understanding of the XGBoost algorithm available in `sl3` will come from referencing the `xgboost` R package and its `xgb.train` function.

6.9 Naming Learners

Recall that our `Stack` from the example above had long names.

```
stack
 [1] "Lrnr_glm_TRUE"
 [2] "Lrnr_mean"
 [3] "Lrnr_glmnet_NULL_deviance_10_0_100_TRUE"
 [4] "Lrnr_glmnet_NULL_deviance_10_1_100_TRUE"
 [5] "Lrnr_polspline"
 [6] "Lrnr_earth_2_3_backward_0_1_0_0"
 [7] "Lrnr_hal9001_2_1_c(3, 2)_5"
 [8] "Lrnr_ranger_500_TRUE_none_1"
 [9] "Lrnr_xgboost_20_1"
[10] "Lrnr_gam_NULL_NULL_GCV.Cp"
```

```
[11] "Lrnr_bayesglm_TRUE"
```

Here, we show a few different ways for the user to name learners. The first way to name a learner is upon instantiation, as shown below:

```
lrn_glm <- Lrnr_glm$new(name = "GLM")
```

We can specify the `name` for any learner upon instantiating it. Above, we named the GLM learner "GLM".

Also, we can specify the names of the learners upon creation of the `Stack`:

```
learners_pretty_names <- c(
  "GLM" = lrn_glm, "Mean" = lrn_mean, "Ridge" = lrn_ridge,
  "Lasso" = lrn_lasso, "Polspline" = lrn_polspline, "Earth" = lrn_earth,
  "HAL" = lrn_hal, "RF" = lrn_ranger, "XGBoost" = lrn_xgb, "GAM" = lrn_gam,
  "BayesGLM" = lrn_bayesglm
)
stack_pretty_names <- Stack$new(learners_pretty_names)
stack_pretty_names
 [1] "GLM"       "Mean"      "Ridge"     "Lasso"     "Polspline" "Earth"
 [7] "HAL"       "RF"        "XGBoost"   "GAM"       "BayesGLM"
```

6.10 Defining Learners over Grid of Tuning Parameters

Customized learners can be created over a grid of tuning parameters. For highly flexible learners that require careful tuning, it is oftentimes very helpful to consider different tuning parameter specifications. However, this is time consuming, so computational feasibility should be considered. Also, when the effective sample size is small, highly flexible learners will likely not perform well since they typically require a lot of data to fit their models. See Phillips et al. [2023] for information on the effective sample size, and step-by-step guidelines for tailoring the SL specification to perform well for the prediction task at hand.

We show two ways to customize learners over a grid of tuning parameters. The first, "do-it-yourself" approach requires that the user or a collaborator has knowledge of the algorithm and their tuning parameters, so they can adequately specify a set of tuning parameters themselves. The second approach does not require the user to have specialized knowledge of an algorithm (although some understanding is still helpful); it uses the `caret` software to automatically select an "optimal" set of tuning parameters over a grid of them.

6.10.1 Do-it-yourself grid

Below, we show how we can create several variations of an XGBoost learner, `Lrnr_xgboost`, by hand. This example is just for demonstrative purposes; users should consult the documentation, and consider computational feasibility and their prediction task to specify an appropriate grid of tuning parameters for their task.

```
grid_params <- list(
  max_depth = c(3, 5, 8),
  eta = c(0.001, 0.1, 0.3),
  nrounds = 100
)
grid <- expand.grid(grid_params, KEEP.OUT.ATTRS = FALSE)

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_3_0.001"

[[2]]
[1] "Lrnr_xgboost_100_1_5_0.001"

[[3]]
[1] "Lrnr_xgboost_100_1_8_0.001"

[[4]]
[1] "Lrnr_xgboost_100_1_3_0.1"

[[5]]
[1] "Lrnr_xgboost_100_1_5_0.1"

[[6]]
[1] "Lrnr_xgboost_100_1_8_0.1"

[[7]]
[1] "Lrnr_xgboost_100_1_3_0.3"

[[8]]
[1] "Lrnr_xgboost_100_1_5_0.3"

[[9]]
[1] "Lrnr_xgboost_100_1_8_0.3"
```

In the example above, we considered every possible combination in the grid to create nine

XGBoost learners. If we wanted to create custom names for each of these learners we could do that as well:

```
names(xgb_learners) <- c(
  "XGBoost_depth3_eta.001", "XGBoost_depth5_eta.001", "XGBoost_depth8_eta.001",
  "XGBoost_depth3_eta.1", "XGBoost_depth5_eta.1", "XGBoost_depth8_eta.1",
  "XGBoost_depth3_eta.3", "XGBoost_depth5_eta.3", "XGBoost_depth8_eta.3"
)
```

6.10.2 Automatic grid and selection with caret

We can use the Lrnr_caret to use the caret software. As described in the Lrnr_caret documentation, Lrnr_caret "uses the caret package's train function to automatically tune a predictive model". Below, we instantiate a neural network that will be automatically tuned with caret and we name the learner "NNET_autotune".

```
lrnr_nnet_autotune <- Lrnr_caret$new(method = "nnet", name = "NNET_autotune")
```

6.11 Learners with Interactions and formula Interface

As described in in Phillips et al. [2023], if it's known/possible that there are interactions among covariates then we can include learners that pick up on that explicitly (e.g., by including in the library a parametric regression learner with interactions specified in a formula) or implicitly (e.g., by including in the library tree-based algorithms that learn interactions empirically).

One way to define interaction terms among covariates in sl3 is with a formula. The argument exists in Lrnr_base, which is inherited by every learner in sl3; even though formula does not explicitly appear as a learner argument, it is via this inheritance. This implementation allows formula to be supplied to all learners, even those without native formula support. Below, we show how to specify a GLM learner that considers two-way interactions among all covariates.

```
lrnr_glm_interaction <- Lrnr_glm$new(formula = "~.^2")
```

As we can see from above, the general behavior of formulain R applies in sl3. See Details of formula in the stats R package for more details on this syntax (e.g,. in RStudio, type "?formula" in the Console and information will appear in the Help tab).

6.12 Covariate Screening

One characteristic of a rich library of learners is that it is effective at handling covariates of high dimension. When there are many covariates in the data relative to the effective sample size (see Figure 1 Flowchart in Phillips et al. [2023]), candidate learners should be coupled with a range of so-called "screeners". A screener is simply a function that returns a subset of covariates. A screener is intended to be coupled with a candidate learner, to define a new candidate learner that considers the reduced set of screener-returned covariates as its covariates.

As stated in Phillips et al. [2023], "covariate screening is essential when the dimensionality of the data is very large, and it can be practically useful in any SL or machine learning application. Screening of covariates that considers associations with the outcome must be cross validated to avoid biasing the estimate of an algorithm's predictive performance". By including screener-learner couplings as additional candidates in the SL library, we are cross validating the screening of covariates. Covariates retained in each CV fold may vary.

A "range of screeners" is a set of screeners that exhibits varying degrees of dimension reduction and incorporates different fitting procedures (e.g., lasso-based screeners that retain covariates with non-zero coefficients, and importance-based screeners that retain the top j most important covariates according to some importance metric. The current set of screeners available in `sl3` is described in each part below.

We will see that, to define a screener and learner coupling in `sl3`, 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.

6.12.1 Variable importance-based screeners

Variable importance-based screeners retain the top j most important covariates according to some importance metric. This screener is provided by `Lrnr_screener_importance` in `sl3` and the parameter j (default is five) is provided by the user via the `num_screen` argument. The user also gets to choose the importance metric considered via the `learner` argument. Any learner with an importance method can be used in `Lrnr_screener_importance`; this currently includes the following:

```
sl3_list_learners(properties = "importance")
[1] "Lrnr_lightgbm"     "Lrnr_randomForest" "Lrnr_ranger"
[4] "Lrnr_xgboost"
```

Let's consider screening covariates based on `Lrnr_ranger` variable importance ranking that selects the top ten most important covariates, according to `ranger`'s "impurity_corrected" importance. We will couple this screener with `Lrnr_glm` to define a new learner that (1) selects the top ten most important covariates, according to `ranger`'s "impurity_corrected" importance, and then (2) passes the screener-selected covariates to `Lrnr_glm`, so `Lrnr_glm` fits a model according to this reduced set of covariates. As mentioned above, this coupling establishes a new learner and requires defining a `Pipeline`. The `Pipeline` is `sl3`'s way of going from (1) to (2).

```
ranger_with_importance <- Lrnr_ranger$new(importance = "impurity_corrected")
RFscreen_top10 <- Lrnr_screener_importance$new(
  learner = ranger_with_importance, num_screen = 10
)
RFscreen_top10_glm <- Pipeline$new(RFscreen_top10, lrn_glm)
```

We could even define the `Pipeline` for the entire `Stack`, so that every learner in it is fit to the screener-selected, reduced set of ten covariates.

```
RFscreen_top10_stack <- Pipeline$new(RFscreen_top10, stack)
```

6.12.2 Coefficient threshold-based screeners

`Lrnr_screener_coefs` 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`.

Let's consider screening covariates with `Lrnr_screener_coefs` to select the variables with non-zero lasso regression coefficients. We will couple this screener with `Lrnr_glm` to define a new learner that (1) selects the covariates with non-zero lasso regression coefficients, and then (2) passes the screener-selected covariates to `Lrnr_glm`, so `Lrnr_glm` fits a model according to this reduced set of covariates. The structure is very similar to above.

```
lasso_screen <- Lrnr_screener_coefs$new(learner = lrn_lasso, threshold = 0)
lasso_screen_glm <- Pipeline$new(lasso_screen, lrn_glm)
```

We could even define the `Pipeline` for the entire `Stack`, so that every learner in it is fit to the lasso screener-selected, reduced set of covariates.

```
lasso_screen_stack <- Pipeline$new(lasso_screen, stack)
```

6.12.3 Correlation-based screeners

`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 p-value lower than some user-specified threshold.

Let's consider screening covariates with `Lrnr_screener_coefs`. We will illustrate how to set up a pipeline with a `Stack`, which looks the same as previous examples. The `Pipeline` with a single learner also looks the same as previous examples.

```
# select top 10 most correlated covariates
corRank_screen <- Lrnr_screener_correlation$new(
  type = "rank", num_screen = 10
)
corRank_screen_stack <- Pipeline$new(corRank_screen, stack)

# select covariates with correlation p-value below 0.05, and a minimum of 3
corP_screen <- Lrnr_screener_correlation$new(
  type = "threshold", pvalue_threshold = 0.05, min_screen = 3
)
corP_screen_stack <- Pipeline$new(corP_screen, stack)
```

6.12.4 Augmented screeners

Augmented screeners are special in that they enforce certain covariates to always be included. That is, if a screener removes a "mandatory" covariate then `Lrnr_screener_augment` will reincorporate it before the learner(s) in the `Pipeline` are fit. An example of how to use this screener is included below. We assume `aged` and `momage` are covariates that must be kept in learner fitting.

```
keepme <- c("aged", "momage")
# using corRank_screen as an example, but any instantiated screener can be
# supplied as screener.
corRank_screen_augmented <- Lrnr_screener_augment$new(
  screener = corRank_screen, default_covariates = keepme
)
corRank_screen_augmented_glm <- Pipeline$new(corRank_screen_augmented, lrn_glm)
```

`Lrnr_screener_augment` is useful when subject-matter experts feel strongly that certain covariate sets must be included, even under screening procedures.

6.12.5 `Stack` with range of screeners

Above, we mentioned that we'd like to consider a range of screeners to diversify the library. Here we show how we can create a new `Stack` from other learners stacks which includes learners with no screening, and learners coupled with various screeners.

```
screeners_stack <- Stack$new(stack, corP_screen_stack, corRank_screen_stack,
                             lasso_screen_stack, RFscreen_top10_stack)
```

This `screeners_stack` could be inputted as `learners` in `Lrnr_sl` to define the SL that considers as candidates learners with no screening, and learners coupled with various screeners.

Advanced `sl3` Functionality:

6.13 Variable Importance Measures

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

The measure of importance in `sl3` is based on a ratio or difference of predictive performance between the SL fit with a removed or permuted covariate (or covariate grouping), and the SL fit with the observed covariate (or covariate grouping), across all of them. In this manner, the larger the ratio/difference in predictive performance, the more important the covariate (or covariate group) is in the SL prediction.

The intuition of this measure is that it calculates the predictive risk (e.g., MSE) of losing one covariate (or one group of covariates), while keeping everything else fixed, comparing this predictive risk to the one from the analytic dataset. If the ratio in predictive risks is one, or the difference is zero, then losing that covariate (group) had no impact, and it is thus not important according to this measure. This procedure is repeated across all of the covariates/groups. As stated above, we can remove each covariate (or covariate group) and refit the SL without it, or we just permute it (faster) and hope for this shuffling to distort any meaningful information that was present. This idea of permuting instead of removing saves

a lot of time, and is also incorporated in `randomForest` variable importance measures. However, the permutation approach is more risky. The `sl3 importance` default is to remove each covariate and then refit. Below, we use the `permute` approach because it is so much faster.

Let's explore the `sl3` variable importance measurements for `sl_fit`, the SL we fit above to the WASH Benefits example dataset. We define a grouping of covariates to consider in the importance evaluation that is based on household assets, as this collection of variables reflects the socio-economic status (SES) of the study's participants.

```
assets <- c("asset_wardrobe", "asset_table", "asset_chair", "asset_khat",
            "asset_chouki", "asset_tv", "asset_refrig", "asset_bike",
            "asset_moto", "asset_sewmach", "asset_mobile", "Nlt18", "Ncomp",
            "watmin", "elec", "floor", "walls", "roof")
set.seed(983)
washb_varimp <- importance(
  fit = sl_fit, eval_fun = loss_squared_error, type = "permute",
  covariate_groups = list("assets" = assets)
)

washb_varimp
```

covariate_group	MSE_difference
aged	0.0414
assets	0.0361
month	0.0149
momedu	0.0101
tr	0.0064
fracode	0.0043
momage	0.0008
sex	0.0006
momheight	0.0006
delta_momheight	0.0001
hfiacat	0.0000
delta_momage	0.0000

```
# plot variable importance
importance_plot(x = washb_varimp)
```

According to the `sl3` variable importance measures, which were assessed by the mean squared error (MSE) difference under permutations of each covariate, the fitted SL's (`sl_fit`) most important variables for predicting weight-for-height z-score (`whz`) are child

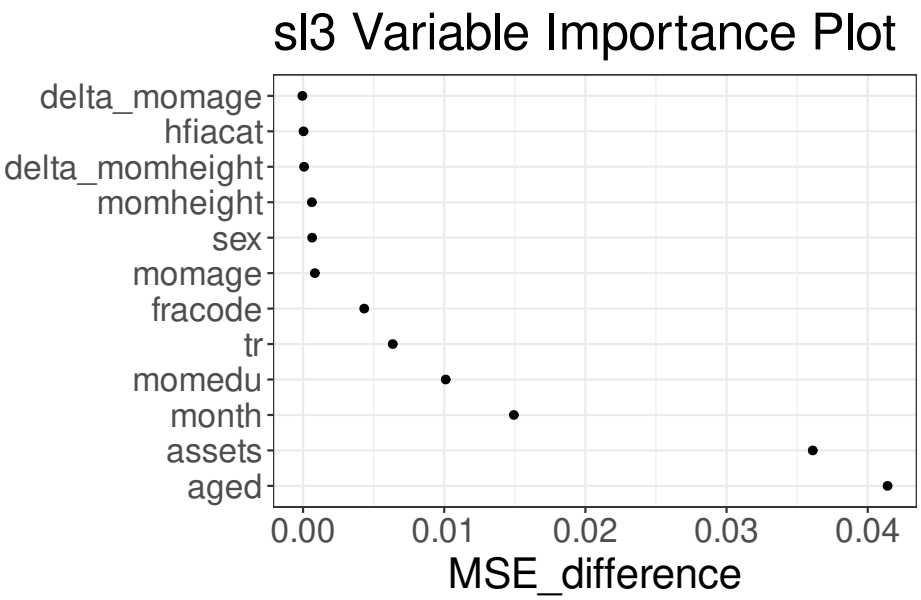


Figure 6.2: sl3 variable importance for predicting weight-for-height z-score with WASH Benefits example dataset

age (`aged`) and household assets (`assets`) that reflect the socio-economic status of the study's subjects.

6.14 Conditional Density Estimation

In certain scenarios it may be useful to estimate the conditional density of a dependent variable, given predictors/covariates that precede it. In the context of causal inference, this arises most readily when working with continuous-valued treatments. Specifically, conditional density estimation (CDE) is necessary when estimating the treatment mechanism for a continuous-valued treatment, often called the *generalized propensity score*. Compared the classical propensity score (PS) for binary treatments (the conditional probability of receiving the treatment given covariates), $\mathbb{P}(A = 1 \mid W)$, the generalized PS is the conditional density of treatment A, given covariates W, $\mathbb{P}(A \mid W)$.

CDE often requires specialized approaches tied to very specific algorithmic implementations. To our knowledge, general and flexible algorithms for CDE have been proposed only sparsely in the literature. We have implemented two such approaches

in `sl3`: a semiparametric CDE approach that makes certain assumptions about the constancy of (higher) moments of the underlying distribution, and second approach that exploits the relationship between the conditional hazard and density functions to allow CDE via pooled hazard regression. Both approaches are flexible in that they allow the use of arbitrary regression functions or machine learning algorithms for the estimation of nuisance quantities (the conditional mean or the conditional hazard, respectively). We elaborate on these two frameworks below. Importantly, per Dudoit and van der Laan [2005] and related works, a loss function appropriate for density estimation is the negative log-density loss $L(\cdot) = -\log(p_n(\cdot))$.

6.14.1 Moment-restricted location-scale

This family of semiparametric CDE approaches exploits the general form $\rho(Y - \mu(X)/\sigma(X))$, where Y is the dependent variable of interest (e.g., treatment A in the PS), X are the predictors (e.g., covariates W in the PS), ρ is a specified marginal density function, and $\mu(X) = \mathbb{E}(Y \mid X)$ and $\sigma(X) = \mathbb{E}[(Y - \mu(X))^2 \mid X]$ are nuisance functions of the dependent variable that may be estimated flexibly. CDE procedures formulated within this framework may be characterized as belonging to a *conditional location-scale* family, that is, in which $p_n(Y \mid X) = \rho((Y - \mu_n(X))/\sigma_n(X))$. While CDE with conditional location-scale families is not without potential disadvantages (e.g., the restriction on the density's functional form could lead to misspecification bias), this strategy is flexible in that it allows for arbitrary machine learning algorithms to be used in estimating the conditional mean of Y given X, $\mu(X) = \mathbb{E}(Y \mid X)$, and the conditional variance of Y given X, $\sigma(X) = \mathbb{E}[(Y - \mu(X))^2 \mid X]$.

In settings with limited data, the additional structure imposed by the assumption that the target density belongs to a location-scale family may prove advantageous by smoothing over areas of low support in the data. However, in practice, it is impossible to know whether and when this assumption holds. This procedure is not a novel contribution of our own (and we have been unable to locate a formal description of it in the literature); nevertheless, we provide an informal algorithm sketch below. This algorithm considers access to n independent and identically distributed (i.i.d.) copies of an observed data random variable $O = (Y, X)$, an *a priori*-specified kernel function ρ, a candidate regression procedure f_μ to estimate $\mu(X)$, and a candidate regression procedure f_σ to estimate $\sigma(X)$.

1. Estimate $\mu(X) = \mathbb{E}[Y \mid X]$, the conditional mean of Y given X, by applying the regression estimator f_μ, yielding $\hat{\mu}(X)$.
2. Estimate $\sigma(X) = \mathbb{V}[Y \mid X]$, the conditional variance of Y given X, by applying the

regression estimator f_σ, yielding $\hat{\sigma}^2(X)$. Note that this step involves only estimation of the conditional mean $\mathbb{E}[(Y - \hat{\mu}(X))^2 \mid X]$.
3. Estimate the one-dimensional density of $(Y - \hat{\mu}(X))^2/\hat{\sigma}^2(X)$, using kernel smoothing to obtain $\hat{\rho}(Y)$.
4. Construct the estimated conditional density $p_n(Y \mid X) = \hat{\rho}((Y - \hat{\mu}(X))/\hat{\sigma}(X))$.

This algorithm sketch encompasses two forms of this CDE approach, which diverge at the second step above. To simplify the approach, one may elect to estimate only the conditional mean $\mu(X)$, leaving the conditional variance to be assumed constant (i.e., estimated simply as the marginal mean of the residuals $\mathbb{E}[(Y - \hat{\mu}(X))^2]$). This subclass of CDE approaches have *homoscedastic error* based on the variance assumption made. The conditional variance can instead by estimated as the conditional mean of the residuals $(Y - \hat{\mu}(X))^2$ given X, $\mathbb{E}[(Y - \hat{\mu}(X))^2 \mid X]$, where the candidate algorithm f_σ is used to evaluate the expectation. Both approaches have been implemented in `sl3`, in the learner `Lrnr_density_semiparametric`. The `mean_learner` argument specifies f_μ and the optional `var_learner` argument specifies f_σ. We demonstrate CDE with this approach below.

```
# semiparametric density estimator with homoscedastic errors (HOSE)
hose_hal_lrnr <- Lrnr_density_semiparametric$new(
  mean_learner = Lrnr_hal9001$new()
)
# semiparametric density estimator with heteroscedastic errors (HESE)
hese_rf_glm_lrnr <- Lrnr_density_semiparametric$new(
  mean_learner = Lrnr_ranger$new()
  var_learner = Lrnr_glm$new()
)

# 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()
)
```

6.14.2 Pooled hazard regression

Another approach for CDE available in `sl3`, and originally proposed in Díaz and van der Laan [2011], leverages the relationship between the (conditional) hazard and density functions. To develop their CDE framework, Díaz and van der Laan [2011] proposed discretizing a continuous dependent variable Y with support $\mathcal{Y}$ based on a number of bins T and a binning procedure (e.g., cutting $\mathcal{Y}$ into T bins of exactly the same length).

The tuning parameter T conceptually corresponds to the choice of bandwidth in classical kernel density estimation. Following discretization, each unit is represented by a collection of records, and the number of records representing a given unit depends on the rank of the bin (along the discretized support) into which the unit falls.

To take an example, an instantiation of this procedure might divide the support of Y into, say, $T = 4$, bins of equal length (note this requires $T + 1$ cut points): $[\alpha_1, \alpha_2), [\alpha_2, \alpha_3), [\alpha_3, \alpha_4), [\alpha_4, \alpha_5]$ (n.b., the rightmost interval is fully closed while the others are only partially closed). Next, an artificial, repeated measures dataset would be created in which each unit would be represented by up to T records. To better see this structure, consider an individual unit $O_i = (Y_i, X_i)$ whose Y_i value is within $[\alpha_3, \alpha_4)$, the third bin. This unit would be represented by three distinct records: $\{Y_{ij}, X_{ij}\}_{j=1}^{3}$, where $\{\{Y_{ij} = 0\}_{j=1}^{2}, Y_{i3} = 1\}$ and three exact copies of X_i, $\{X_{ij}\}_{j=1}^{3}$. This representation in terms of multiple records for the same unit allows for the conditional hazard probability of Y_i falling in a given bin along the discretized support to be evaluated via standard binary regression techniques.

In fact, this proposal reformulates the binary regression problem into a corresponding set of hazard regressions: $\mathbb{P}(Y \in [\alpha_{t-1}, \alpha_t) \mid X) = \mathbb{P}(Y \in [\alpha_{t-1}, \alpha_t) \mid Y \geq \alpha_{t-1}, X) \times \prod_{j=1}^{t-1}\{1 - \mathbb{P}(Y \in [\alpha_{j-1}, \alpha_j) \mid Y \geq \alpha_{j-1}, X)\}$. Here, the probability of $Y \in \mathcal{Y}$ falling in bin $[\alpha_{t-1}, \alpha_t)$ may be directly estimated via a binary regression procedure, by re-expressing the corresponding likelihood in terms of the likelihood of a binary variable in a dataset with this repeated measures structure. Finally, the hazard estimates can be mapped into density estimates by re-scaling the hazard estimates by the bin sizes $|\alpha_t - \alpha_{t-1}|$, that is, $p_{n,\alpha}(Y \mid X) = \mathbb{P}(Y \in [\alpha_{t-1}, \alpha_t) \mid X)/|\alpha_t - \alpha_{t-1}|$, for $\alpha_{t-1} \leq a < \alpha_t$. We provide an informal sketch of this algorithm below.

1. Apply a procedure to divide the observed support of Y, $\max(Y) - \min(Y)$, into T bins: $[\alpha_1, \alpha_2), \ldots, [\alpha_{t-1}, \alpha_t), [\alpha_t, \alpha_{t+1}]$.
2. Expand the observed data into a repeated measures data structure, expressing each individual observation as a set of up to T records, recording the observation ID alongside each such record. For a single unit i, the set of records takes the form $\{Y_{ij}, X_{ij}\}_{j=1}^{T_i}$, where X_{ij} are constant in the index set $\mathcal{J}$, Y_{ij} is a binary counting process that jumps from 0 to 1 at its final index (at the bin into which Y_i falls), and $T_i \leq T$ indicates the bin along its support into which Y_i falls.
3. Estimate the hazard probability, conditional on X, of bin membership $\mathbb{P}(Y_i \in [\alpha_{t-1}, \alpha_t) \mid X)$ using any binary regression estimator or appropriate machine learning algorithm.
4. Rescale the conditional hazard probability estimates to the conditional density scale

by dividing the cumulative hazard by the width of the bin into which X_i falls, for each observation $i = 1, \ldots, n$. If the support set is partitioned into bins of equal size (approximately n/T samples in each bin), this amounts to rescaling by a constant. If the support set is partitioned into bins of equal range, then the rescaling might vary across bins.

A key element of this proposal is the flexibility to use any binary regression procedure or appropriate machine learning algorithm to estimate $\mathbb{P}(Y \in [\alpha_{t-1}, \alpha_t) \mid X)$, facilitating the incorporation of flexibletechniques like ensemble learning [Breiman, 1996, van der Laan et al., 2007]. This extreme degree of flexibility integrates perfectly with the underlying design principles of `sl3`; however, we have not yet implemented this approach in its full generality. A version of this CDE approach, which limits the original proposal by replacing the use of arbitrary binary regression with the highly adaptive lasso (HAL) algorithm [Benkeser and van der Laan, 2016] is supported in the `haldensify` package [Hejazi et al., 2022a] (the HAL implementation in `haldensify` is provided the `hal9001` package [Coyle et al., 2022, Hejazi et al., 2020a]). This CDE algorithm that uses `haldensify` is incorporated as learner `Lrnr_haldensify` in `sl3`, as we demonstrate below.

```
# learners used for conditional densities for (g_n)
haldensify_lrnr <- Lrnr_haldensify$new(
  n_bins = c(5, 10)
)
```

Exercises:

Binary outcome prediction

Follow the steps below to predict the probability for myocardial infarction (`mi`) using the available covariate data. We thank Dr. David Benkeser, Assistant Professor of Biostatistics and Bioinformatics at Emory University, for making this Cardiovascular Health Study (CHS) publicly data available.

```
# load the data set
library(readr)
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:

```
head(chspred)
```

waist	alcoh	hdl	beta	smoke	ace	ldl	bmi	aspirin	gend	age	estrgn	glu	
110.16	0.000	66.50	0	0	1	114.2	28.00	0	0	73.52	0	159.93	70
89.98	0.000	50.07	0	0	0	103.8	20.89	0	0	61.77	0	153.39	33
106.19	8.417	40.51	0	0	0	165.7	28.46	1	1	72.93	0	121.71	-17
90.06	0.000	36.17	0	0	0	45.2	23.96	0	0	79.12	0	53.97	11
78.61	2.979	71.06	0	1	0	131.3	10.97	0	1	69.02	0	94.32	9
91.66	0.000	59.50	0	0	0	171.2	29.13	0	1	81.83	0	212.91	-28

1. Create an `sl3` 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 tuning parameters for one of your learners. Incorporate at least one screener-learner coupling.
3. Make the SL and train it on the task.
4. Print the SL fit results by adding `$cv_risk(loss_squared_error)` to your fit object.

6.15 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!

- Ultimately, we want an estimator that is does the best job in approximating our question of interest. That is, we care about doing a the best job possible estimating ψ_0. The SL is an essential step to help us get there: the counterfactual predicted outcome estimates (like those explained above), and other SL-derived estimates (like a propensity score) play a key role in estimating ψ_0. However, SL is not the end of the estimation procedure. Specifically, if we simply plugged in the SL estimates into the target parameter, we would not have an asymptotically linear estimator of the target estimand; the SL is not an efficient substitution estimator and does not admit statistical inference. Why does this matter?

  - An asymptotically linear estimator is one that converges to the estimand a $\frac{1}{\sqrt{n}}$ rate, thereby permitting formal statistical inference, i.e., confidence intervals and p-values, (see Chapters 4–6 of van der Laan and Rose [2011]).
  - Substitution, or plug-in, estimators are desirable because they respect both the local and global constraints of the statistical model, such as bounds on an

outcome, and have they have better finite-sample properties (see Chapter 6 of van der Laan and Rose [2011]).

- 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 (see Chapter 6 of van der Laan and Rose [2011]).
  - 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 (Chapter 5 of van der Laan and Rose [2011]). Various canonical gradients are shown in the chapters that follow.
  - Practitioners do not need to know how to calculate a canonical gradient to explain properties that are desirable for an estimator to possess (like substitution/plug-in, admits valid inference, efficient, and ability to optimize finite sample performance). These properties motivate the use TMLE, since TMLE satisfies them.

- TMLE is a general strategy that succeeds in constructing efficient and asymptotically linear plug-in estimators that are robust in finite samples.
- SL is fantastic for pure prediction, and for obtaining initial estimates of components in the likelihood (the first step of TMLE), but we need the second, targeting/updating/fluctuation, step to have the desirable statistical properties mentioned above.
- In the chapters that follow, we focus on various targeted maximum likelihood estimator and the targeted minimum loss-based estimator, both referred to as TMLE.

6.16 Appendix

6.16.1 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)
data.table::setDT(chspred)

# make task
chspred_task <- make_sl3_Task(
  data = chspred,
  covariates = 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"
curated_glm_learner <- Lrnr_glm_fast$new(covariates = c("smoke", "beta"))
mean_learner <- Lrnr_mean$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)
```

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

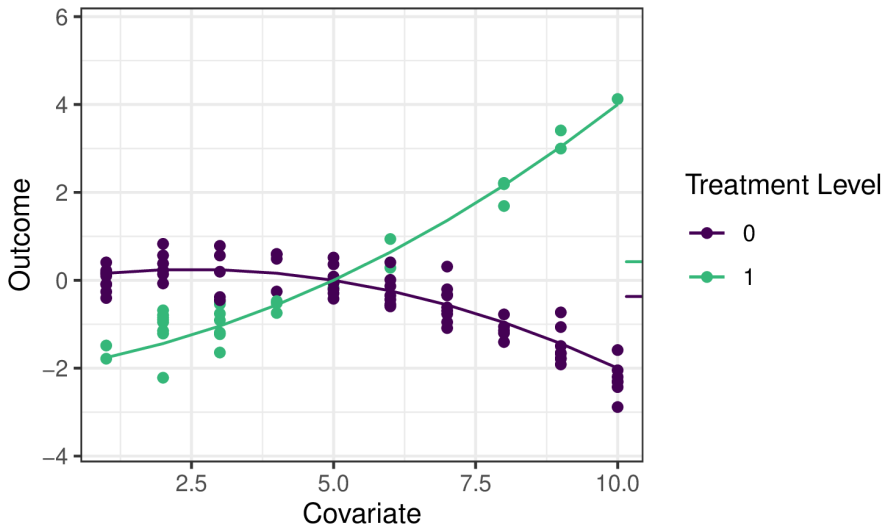
1. Understand why we use TMLE for effect estimation.
2. Use `tmle3` to estimate an Average Treatment Effect (ATE).
3. Understand how to use `tmle3` "Specs" objects.
4. Fit `tmle3` for a custom set of target parameters.
5. Use the delta method to estimate transformations of target parameters.

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:



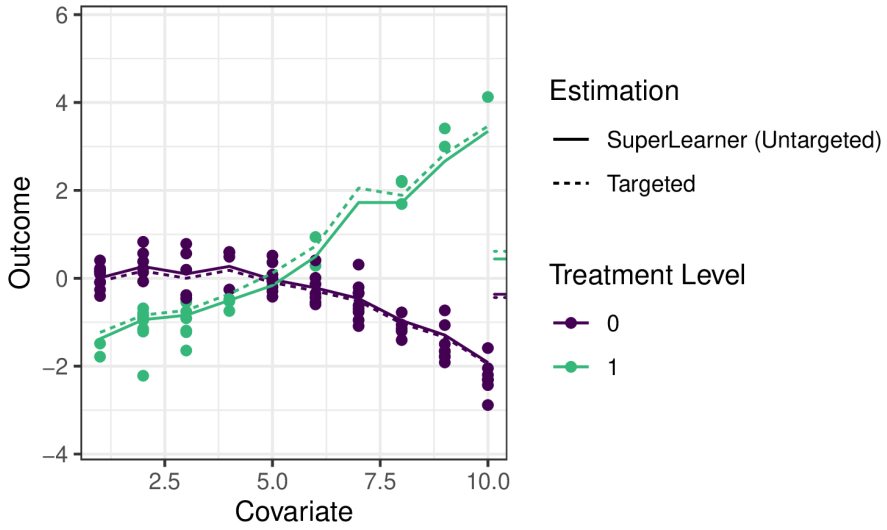
The small ticks on the right indicate the mean outcomes (averaging over W) under $A = 1$ and $A = 0$ respectively, so their difference is the quantity we'd like to estimate.

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.

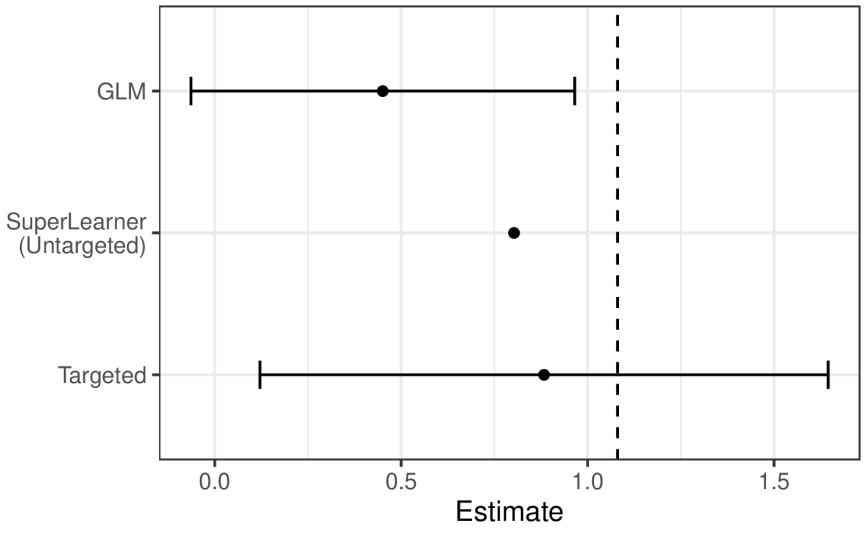
Applying `sl3` to estimate the outcome regression in our example, we can see that the ensemble machine learning predictions fit the data quite well:



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 $\overline{Q}_n^\star(A, W)$ 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. $\overline{Q}_n^\star(A, W)$ is the TMLE-augmented estimate $f(\overline{Q}_n^\star(A, W)) = f(\overline{Q}_n(A, W)) + \epsilon \cdot$

$H_n(A, W)$, 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}] -0.005233 0.007111 0.05025
      lower  upper psi_transformed lower_transformed upper_transformed
1: -0.09139 0.1056        0.007111          -0.09139            0.1056
```

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] 0.007111
```

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: 0.000213 0.3302 -0.3550
   2: 0.000213 0.3398 -0.9297
   3: 0.000213 0.3287 -0.8058
   4: 0.000213 0.3247 -0.9373
   5: 0.000213 0.3238 -0.5755
  ---
4691: 0.000213 0.2131 -0.5868
4692: 0.000213 0.2130 -0.2243
4693: 0.000213 0.2073 -0.7393
4694: 0.000213 0.2580 -0.9151
4695: 0.000213 0.1821 -1.0360
```

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}] -0.004549  0.01096 0.05046
      lower  upper psi_transformed lower_transformed upper_transformed
1: -0.08794 0.1099         0.01096          -0.08794             0.1099
```

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      se
1:  TSM                              E[Y_{A=Control}] -0.592825 -0.62093 0.02978
2:  TSM                          E[Y_{A=Handwashing}] -0.615693 -0.65811 0.04155
3:  TSM                            E[Y_{A=Nutrition}] -0.608009 -0.60779 0.04187
4:  TSM                     E[Y_{A=Nutrition + WSH}] -0.597373 -0.60990 0.04095
5:  TSM                           E[Y_{A=Sanitation}] -0.582596 -0.57852 0.04220
6:  TSM                                 E[Y_{A=WSH}] -0.517360 -0.44815 0.04515
7:  TSM                               E[Y_{A=Water}] -0.562570 -0.53743 0.03909
8:  ATE E[Y_{A=Nutrition + WSH}] - E[Y_{A=Control}] -0.004549  0.01103 0.05045
      lower   upper psi_transformed lower_transformed upper_transformed
1: -0.67929 -0.5626        -0.62093          -0.67929           -0.5626
2: -0.73955 -0.5767        -0.65811          -0.73955           -0.5767
3: -0.68986 -0.5257        -0.60779          -0.68986           -0.5257
4: -0.69015 -0.5296        -0.60990          -0.69015           -0.5296
5: -0.66123 -0.4958        -0.57852          -0.66123           -0.4958
6: -0.53664 -0.3597        -0.44815          -0.53664           -0.3597
7: -0.61405 -0.4608        -0.53743          -0.61405           -0.4608
8: -0.08785  0.1099         0.01103          -0.08785            0.1099
```

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 `sl3` 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 `sl3` 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.

7.9 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.

Part III

Part 3: Advanced Topics

8

Dynamic and Optimal Individualized Treatment Regimes

Ivana Malenica

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

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 ITR with the

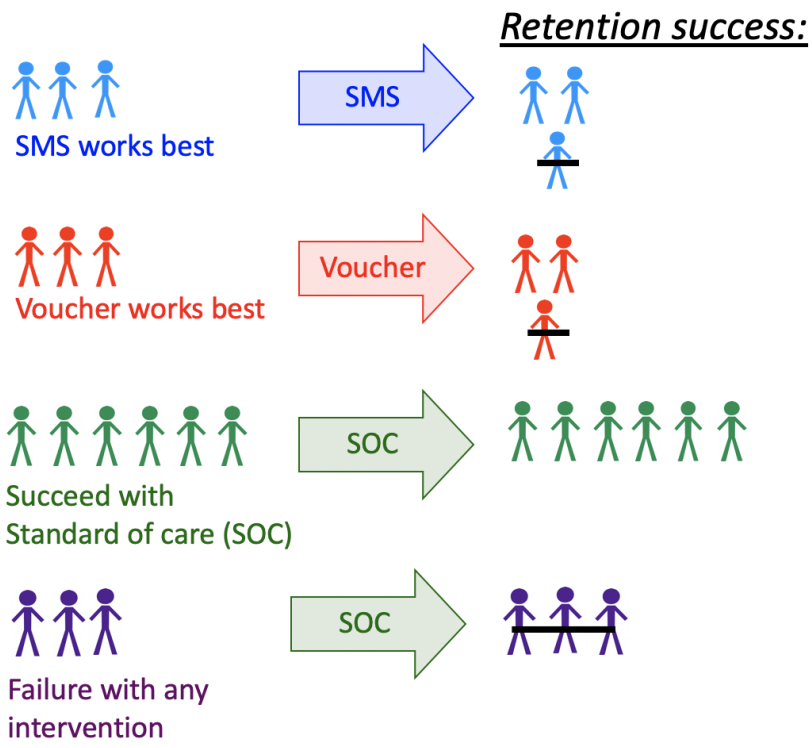


Figure 8.1: Dynamic Treatment Regime in a Clinical Setting

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. [2023a] and Montoya et al. [2023b].

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 $V \rightarrow d(V) \in \{a_1, \cdots, a_{n_A}\}$. 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:

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

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:

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

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, 2011]. 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:

$$Q_0(V) \equiv \mathbb{E}_0[Y_1 - Y_0 \mid V] \equiv \mathbb{E}_0[\bar{Q}_{Y,0}(1, W) - \bar{Q}_{Y,0}(0, W) \mid V],$$

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:

$$Q_0^{pblip}(V) = \{Q_{0,a}^{pblip}(V) : a \in \mathcal{A}\}$$

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:

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

   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 `sl3`

To easily incorporate ensemble machine learning into the estimation procedure, we rely on the facilities provided in the `sl3` R package. Using the framework provided by the `sl3` 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}]   0.3504   0.5508 0.02622 0.4994 0.6022          0.5508
   lower_transformed upper_transformed
1:            0.4994            0.6022
```

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}]   0.3566   0.5579 0.02577 0.5074 0.6084          0.5579
   lower_transformed upper_transformed
1:            0.5074            0.6084
```

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
274 726
```

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 psi_transformed
1:  TSM E[Y_{A=NULL}]   0.3259   0.5321 0.01034 0.5118 0.5523          0.5321
   lower_transformed upper_transformed
1:            0.5118            0.5523
```

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:

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

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

$$\begin{aligned}\mathbb{P}(Y = 1 \mid A, W) = 0.5\text{logit}^{-1}[15I(A = 1)(W_1 - 0.5) - \\ 3I(A = 2)(2W_1 + 0.5) + \\ 3I(A = 3)(3W_1 - 0.5)] + \text{logit}^{-1}(W_2 W_1)\end{aligned}$$

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,
  eval_function = loss_loglik_multinomial,
  learner_function = metalearner_linear_multinomial
)
g_learner <- make_learner(Lrnr_sl, list(lrn_xgboost_100, lrn_xgboost_500, lrn_mean), m

# 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_ga"
 [5] "Lrnr_glmnet"                 "Lrnr_grf"
 [7] "Lrnr_gru_keras"              "Lrnr_h2o_glm"
 [9] "Lrnr_h2o_grid"               "Lrnr_independent_binomial"
[11] "Lrnr_lightgbm"               "Lrnr_lstm_keras"
[13] "Lrnr_mean"                   "Lrnr_multivariate"
[15] "Lrnr_nnet"                   "Lrnr_optim"
[17] "Lrnr_polspline"              "Lrnr_pooled_hazards"
[19] "Lrnr_randomForest"           "Lrnr_ranger"
[21] "Lrnr_rpart"                  "Lrnr_screener_correlation"
[23] "Lrnr_solnp"                  "Lrnr_svm"
[25] "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 psi_transformed
1:  TSM E[Y_{A=NULL}]   0.5347   0.6213 0.06628 0.4914 0.7512          0.6213
   lower_transformed upper_transformed
1:            0.4914            0.7512

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

  1   2   3
249 432 319
```

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=W4,W3,W2,W1)}]   0.5301   0.5497 0.05822 0.4356 0.6638
   psi_transformed lower_transformed upper_transformed
1:          0.5497            0.4356            0.6638
```

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}]   0.5377   0.6582 0.02135 0.6163   0.7          0.6582
   lower_transformed upper_transformed
1:            0.6163               0.7

# 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
   0.00    0.00    0.00    0.46    1.00    1.00     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
```

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    upper
1:  ATE E[Y_{A=NULL}] - E[Y] -0.013019 -0.06474 0.02171 -0.10730 -0.02218
2:  ATE E[Y_{A=NULL}] - E[Y]  0.000332  0.05371 0.01688  0.02062  0.08679
   psi_transformed lower_transformed upper_transformed  A           W Z_stat
1:        -0.06474          -0.10730          -0.02218 W1  W3,W4,W2,A -2.982
2:         0.05371           0.02062           0.08679  A W3,W4,W2,W1  3.182
        p_nz p_nz_corrected
1: 0.0014338       0.001434
2: 0.0007326       0.001434
```

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

Featuring the `tmle3shift R` package.

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. [2020b], 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 $O \sim P_0 \in \mathcal{M}$, where $\mathcal{M}$ is the nonparametric statistical model, minimizing any restrictions on the form of the data-generating distribution P_0. 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, when evaluated on a typical observation o:

$$\begin{aligned} p_0(o) =& q_{0,Y}(y \mid A = a, W = w) \\ & g_{0,A}(a \mid W = w) \\ & q_{0,W}(w), \end{aligned} \tag{9.2}$$

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 $\overline{Q}(A, W) = \mathbb{E}[Y \mid A, W]$, $g(A \mid W) = g_A(A \mid W)$, 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 (9.2). 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 $b_j(\cdot, w; \delta)$ with derivative $b'_j(\cdot, w; \delta)$.

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. [2020b], 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.

Definition 9.2 (No Unmeasured Confounding). $A_i \perp\!\!\!\perp Y_i^{A_{\delta,i}} \mid W_i$, for $i = 1, \ldots, n$. This is the observational study analog to the well-known randomization assumption.

Definition 9.3 (Treatment Positivity). $a_i \in \mathcal{A} \implies d(a_i, w_i) \in \mathcal{A}$ 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 $\overline{Q}_n$ 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,
$$\text{logit}\overline{Q}_{\epsilon,n}(a,w) = \text{logit}\overline{Q}_n(a,w) + \epsilon H_n(a,w),$$
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:

$$\sqrt{n}(\psi_n - \psi_0) \rightarrow \text{Normal}(0, \sigma^2(D(P_0))). \tag{9.10}$$

Thus, an estimate σ_n^2 of the variance $\sigma^2(D(P_0))$ may be computed

$$\sigma_n^2 = \frac{1}{n} \sum_{i=1} D^2(\overline{Q}_n^\star, g_n)(O_i), \tag{9.11}$$

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. [2022a], 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  0.27165 1
2:  0  0 -0.66337 1
3:  0  0  0.11337 0
4:  0  1 -0.73256 0
5:  1  1  0.38884 1
6:  0  0  0.04399 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:  0.94735 0.05265
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 psi_transformed
1:  TSM E[Y_{A=NULL}]   0.7645   0.7601 0.02284 0.7154 0.8049          0.7601
   lower_transformed upper_transformed
1:            0.7154            0.8049
```

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 $\delta(A, W)$:

$$\delta(a,w) = \begin{cases} \delta, & \delta_{\min}(a,w) \leq \delta \leq \delta_{\max}(a,w) \\ \delta_{\max}(a,w), & \delta \geq \delta_{\max}(a,w) \\ \delta_{\min}(a,w), & \delta \leq \delta_{\min}(a,w) \end{cases}, \tag{9.12}$$

where

$$\delta_{\max}(a,w) = \text{argmax}_{\left\{\delta \geq 0, \frac{g(a-\delta|w)}{g(a|w)} \leq M\right\}} \frac{g(a-\delta \mid w)}{g(a \mid w)}$$

and

$$\delta_{\min}(a,w) = \text{argmin}_{\left\{\delta \leq 0, \frac{g(a-\delta|w)}{g(a|w)} \leq M\right\}} \frac{g(a-\delta \mid w)}{g(a \mid w)}.$$

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:  0.99992954 0.00007046
Iter: 2 fn: 547.4323     Pars:  0.99996416 0.00003584
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}]   0.5681   0.5718 0.021416 0.5299 0.6138
2:        TSM  E[Y_{A=NULL}]   0.6975   0.6975 0.022996 0.6524 0.7426
3:        TSM  E[Y_{A=NULL}]   0.8167   0.8155 0.017110 0.7819 0.8490
4: MSM_linear MSM(intercept)   0.6941   0.6949 0.019107 0.6575 0.7324
5: MSM_linear     MSM(slope)   0.1243   0.1218 0.008603 0.1049 0.1387
   psi_transformed lower_transformed upper_transformed
1:          0.5718           0.5299            0.6138
2:          0.6975           0.6524            0.7426
3:          0.8155           0.7819            0.8490
4:          0.6949           0.6575            0.7324
5:          0.1218           0.1049            0.1387
```

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., 2020b]. 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 (**??**), 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     146.4       1     3    11
2: -1.16  1       4     9  286   2     25      2     148.8       3     2     4
3: -1.05  1      20     9  264   2     25      2     152.2       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., 2022, Hejazi et al., 2020a, 2022a] 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 $\{\delta_1, \ldots, \delta_k\}$ 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

Featuring the `tmle3mediate R` package.

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.

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].

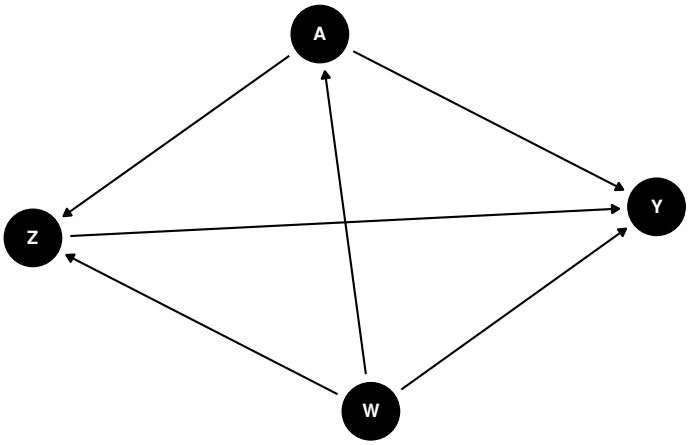
10.2 Data Structure and Notation

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:

$$\begin{aligned} p_0(o) =& q_{0,Y}(y \mid Z = z, A = a, W = w) \\ & q_{0,Z}(z \mid A = a, W = w) \\ & g_{0,A}(a \mid W = w) \\ & q_{0,W}(w). \end{aligned} \tag{10.2}$$

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. Without further assumptions, common mediation parameters (the natural direct and indirect effects) cannot be identified in the presence of such confounding, though Tchetgen Tchetgen and VanderWeele [2014] discuss a monotonicity assumption that may be useful when justified by available scientific knowledge about the system under study. The interested reader may wish to consult recent advances in the vast and quickly growing 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. [2022b] 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:

$$\begin{aligned}\mathbb{E}[Y(1) - Y(0)] = &\underbrace{\mathbb{E}[Y(1, Z(0)) - Y(0, Z(0))]}_{\text{NDE}} \\ &+ \underbrace{\mathbb{E}[Y(1, Z(1)) - Y(1, Z(0))]}_{\text{NIE}}.\end{aligned}$$

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 pathways*. 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 (iid) 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 10.1 (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 the treatment A and mediators Z.

Definition 10.2 (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 10.3 (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 A and baseline covariates W. 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 the two realizations of the mediator density under differing treatment contrasts.

Definition 10.4 (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 their joint distribution is well-defined, these counterfactuals can never be jointly realized.

While the first three assumptions may be familiar based on their analogs in simpler settings, the cross-world independence requirement is unique to identification of the natural direct and indirect effects. This assumption resolves a challenging 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], implying that corresponding scientific claims cannot be falsified through experimentation [Popper, 1934, Dawid, 2000] and, consequently, directly contradicting a foundational 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. For example, Petersen et al. [2006] weaken this assumption by requiring it only for conditional means (rather than distinct counterfactuals) and adopt a view of the natural direct effect as a weighted average of another type of direct effect, the controlled direct effect. 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

$$\begin{aligned}\psi_{\text{NDE}} &= \mathbb{E}[Y(1, Z(0)) - Y(0, Z(0))] \\ &= \sum_w \sum_z [\underbrace{\mathbb{E}(Y \mid A = 1, z, w)}_{\overline{Q}_Y(A=1,z,w)} - \underbrace{\mathbb{E}(Y \mid A = 0, z, w)}_{\overline{Q}_Y(A=0,z,w)}] \\ &\quad \times \underbrace{p(z \mid A = 0, w)}_{q_Z(Z|0,w))} \underbrace{p(w)}_{q_W},\end{aligned}$$

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 $\overline{Q}_{\text{diff}} = \overline{Q}_Y(Z, 1, W) - \overline{Q}_Y(Z, 0, W)$, which is itself only a functional parameter of the data distribution. $\overline{Q}_{\text{diff}}$ captures differences 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 $\overline{Q}_{\text{diff}}$, under the setting in which all individuals are treated as falling in 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 [Díaz and van der Laan, 2011, Hejazi et al., 2022a], 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,

$$\frac{p(A = 0 \mid Z, W)}{g(0 \mid W)} \frac{g(1 \mid W)}{p(A = 1 \mid Z, W)}.$$

Going forward, we will denote this re-parameterized conditional probability functional $e(A \mid Z, W) := p(A \mid Z, W)$. The same re-parameterization technique has been used by Zheng and van der Laan [2012], Tchetgen Tchetgen [2013], Díaz and Hejazi [2020], Díaz et al. [2020], and Hejazi et al. [2022b] 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 mean 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* Z. This counterfactual 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 replace $q_Z(Z \mid A, W)$ with $e(A \mid Z, W)$ in the estimation process, avoiding estimation of a possibly multivariate conditional density. However, in this case, the mediated mean outcome difference, previously computed by regressing $\overline{Q}_{\text{diff}}$ on W among the control units (for whom $A = 0$ is observed) is instead replaced by a two-step process. First, $\overline{Q}_Y(Z, 1, W)$, the conditional mean of 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). Then, the same quantity, $\overline{Q}_Y(Z, 1, W)$ is again regressed on W, but this time among only the control units (i.e., for whom $A = 0$ is observed). The mean difference of these two functionals of the data distribution is a valid estimator of the NIE. It 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, while our estimand ψ_{NIE} is different, the same estimation techniques useful for constructing efficient estimators of the NDE come into play.

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].

We previously discussed stochastic interventions when considering how to intervene on continuous-valued treatments; however, these intervention schemes may be applied to all manner of treatment variables. 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] in the context of causal mediation analysis, the identification assumptions required for the PIDE and the PIIE are significantly more lax than those required for the NDE and NIE. These identification assumptions include the following. Importantly, the assumption of cross-world counterfactual independence is not at all required.

Definition 10.5 (Conditional Exchangeability of Treatment and Mediators)**.** Assume that $\mathbb{E}\{Y(a, z) \mid Z, A, W\} = \mathbb{E}\{Y(a, z) \mid Z, W\} \; \forall \; (a, z) \in \mathcal{A} \times \mathcal{Z}W$. This assumption is stronger than and implied by the assumption $Y(a, z) \perp\!\!\!\perp (A, Z) \mid W$, originally proposed by Vansteelandt and VanderWeele [2012] for identification of mediated effects among the treated. In introducing this assumption Díaz and Hejazi [2020] state that "This assumption would be satisfied for any pre-exposure W in a randomized experiment in which exposure

and mediators are randomized. Thus, the direct effect for a population intervention corresponds to contrasts between treatment regimes of a randomized experiment via interventions on A and Z, unlike the natural direct effect for the average treatment effect [Robins and Richardson, 2010]."

Definition 10.6 (Common Support of Treatment and Mediators)**.** Assume that $\text{supp}\{g_\delta(\cdot \mid w)\} \subseteq \text{supp}\{g(\cdot \mid w)\}\ \forall\ w \in \mathcal{W}$. This assumption is standard and requires only that the post-intervention value of A be supported in the data. Note that this is significantly weaker than the treatment and mediator positivity conditions required for the natural direct and indirect effects, and it is a direct consequence of using stochastic (rather than static) interventions.

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 on both the treatment and mediation mechanisms, as per Kennedy [2019].

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

$$\psi_0(\delta) = \int \overline{Q}_{0,Y}(a, z, w) g_{0,\delta}(a \mid w) p_0(z, w) d\nu(a, z, w),$$

for which a TML estimator is available. In the case of $A \in \{0, 1\}$, the *efficient influence function* (EIF) with respect to the nonparametric statistical model $\mathcal{M}$ is $D_\delta(o) = D_\delta^Y(o) + D_\delta^A(o) + D_\delta^{Z,W}(o) - \psi(\delta)$, where the orthogonal components of the EIF are defined as follows:

- $D_\delta^Y(o) = \{g_\delta(a \mid w)/e(a \mid z, w)\}\{y - \overline{Q}_Y(z, a, w)\}$,
- $D_\delta^A(o) = \{\delta\phi(w)(a - g(1 \mid w))\}/\{(\delta g(1 \mid w) + g(0 \mid w))^2\}$, and where $\phi(w) := \mathbb{E}\{\overline{Q}_Y(1, Z, W) - \overline{Q}_Y(0, Z, W) \mid W = w\}$,
- $D_\delta^{Z,W}(o) = \int_{\mathcal{A}} \overline{Q}_Y(z, a, w) g_\delta(a \mid w) d\kappa(a)$.

The TML estimator may be computed by fluctuating initial estimates of the nuisance parameters so as to solve the EIF estimating equation. The resultant TML estimator is

$$\psi_n^\star(\delta) = \int_{\mathcal{A}} \frac{1}{n} \sum_{i=1}^{n} \overline{Q}_{Y,n}^\star(Z_i, a, W_i) g_{\delta,n}^\star(a \mid W_i) d\kappa(a),$$

where $g_{\delta,n}^\star(a \mid w)$ and $\overline{Q}_{Y,n}^\star(z, a, w)$ are generated by *targeting* regressions that fluctuate (or tilt) initial nuisance parameter estimates towards solutions of the score equations $n^{-1} \sum_{i=1}^n D_\delta^A(O_i) = 0$ and $n^{-1} \sum_{i=1}^n D_\delta^Y(O_i) = 0$, respectively. This TML estimator $\psi_n^\star(\delta)$ is implemented in `tmle3mediate` package. We demonstrate the use of `tmle3mediate` to obtain $\mathbb{E}\{Y(A_\delta, Z)\}$ via its TML estimator in the following worked code examples.

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}] 0.002419 0.003376 0.04342 -0.08173 0.08848
   psi_transformed lower_transformed upper_transformed
1:        0.003376          -0.08173           0.08848
```

Based on the output, we conclude that the indirect effect of the treatment through the mediators (sex, month, aged) is 0.0034.

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}]   0.0148   0.0148 0.08569 -0.1532 0.1828
   psi_transformed lower_transformed upper_transformed
1:          0.0148           -0.1532            0.1828
```

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 0.0148. 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 0.0182.

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:

```
# 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 $\beta_{0,\text{PIDE}}(\delta) = \psi_0(\delta) - \mathbb{E}Y$. Thus, a TML estimator of the PIDE, $\beta_{n,\text{PIDE}}(\delta)$ may be expressed as a composition of estimators of its constituent parameters:

$$\beta_{n,\text{PIDE}}(\delta) = \psi_n^\star(\delta) - \frac{1}{n}\sum_{i=1}^{n} Y_i.$$

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

Exercise 10.4. Forthcoming

Solution. Forthcoming

Exercise 10.5. Forthcoming

Solution. Forthcoming

Bibliography

Chen Avin, Ilya Shpitser, and Judea Pearl. Identifiability of path-specific effects. In *IJCAI International Joint Conference on Artificial Intelligence*, pages 357–363, 2005.

Monya Baker. 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, 2016.

Oliver Bembom and Mark J van der Laan. A practical illustration of the importance of realistic individualized treatment rules in causal inference. *Electronic Journal of Statistics*, 1:574–596, 2007.

Henrik Bengtsson. A unifying framework for parallel and distributed processing in r using futures. *The R Journal*, 2021. doi: 10.32614/RJ-2021-048. URL https://journal.r-project.org/archive/2021/RJ-2021-048/index.html.

David Benkeser and Jialu Ran. Nonparametric inference for interventional effects with multiple mediators. *Journal of Causal Inference*, 2021. doi: 10.1515/jci-2020-0018.

David Benkeser and Mark J van der Laan. The highly adaptive lasso estimator. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 2016. doi: 10.1109/dsaa.2016.93. URL https://doi.org/10.1109/dsaa.2016.93.

Leo Breiman. Stacked regressions. *Machine learning*, 24(1):49–64, 1996.

Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

Jonathan B Buckheit and David L Donoho. Wavelab and reproducible research. In *Wavelets and statistics*, pages 55–81. Springer, 1995.

Bibhas Chakraborty and Erica EM Moodie. *Statistical Methods for Dynamic Treatment Regimes: Reinforcement Learning, Causal Inference, and Personalized Medicine (Statistics for Biology and Health)*. Springer, 2013. ISBN 1461474272.

Jeremy R Coyle and Nima S Hejazi. origami: A generalized framework for cross-validation in r. *Journal of Open Source Software*, 3(21), January 2018. doi: 10.21105/joss.00512. URL https://doi.org/10.21105/joss.00512.

Jeremy R Coyle, Nima S Hejazi, Ivana Malenica, and Rachael V Phillips. `origami`: *Generalized framework for cross-validation*. URL https://CRAN.R-project.org/package=origami. `R` package version 1.0.5.

Jeremy R Coyle, Nima S Hejazi, Ivana Malenica, Rachael V Phillips, Benjamin F Arnold, Andrew Mertens, Jade Benjamin-Chung, Weixin Cai, Sonali Dayal, John M Colford Jr., Alan E Hubbard, and Mark J van der Laan. Targeting Learning: Robust statistics for reproducible research. *arXiv*, 2021. URL https://arxiv.org/abs/2006.07333.

Jeremy R Coyle, Nima S Hejazi, Rachael V Phillips, Lars WP van der Laan, and Mark J van der Laan. *hal9001: The scalable highly adaptive lasso*, 2022. URL https://doi.org/10.5281/zenodo.3558313. R package version 0.4.3.

Anthony Christopher Davison and David Victor Hinkley. *Bootstrap Methods and Their Application*. Cambridge University Press, 1997.

A Philip Dawid. Causal inference without counterfactuals. *Journal of the American Statistical Association*, 95(450):407–424, 2000.

Iván Díaz and Nima S Hejazi. Causal mediation analysis for stochastic interventions. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 82(3): 661–683, 2020. doi: 10.1111/rssb.12362. URL https://doi.org/10.1111/rssb.12362.

Iván Díaz and Mark J van der Laan. Super learner based conditional density estimation with application to marginal structural models. *The International Journal of Biostatistics*, 7(1):1–20, 2011.

Iván Díaz and Mark J van der Laan. Population intervention causal effects based on stochastic interventions. *Biometrics*, 68(2):541–549, 2012.

Iván Díaz and Mark J van der Laan. Sensitivity analysis for causal inference under unmeasured confounding and measurement error problems. *The International Journal of Biostatistics*, 9(2):149–160, 2013. doi: 10.1515/ijb-2013-0004.

Iván Díaz and Mark J van der Laan. Stochastic treatment regimes. In *Targeted Learning in Data Science: Causal Inference for Complex Longitudinal Studies*, pages 167–180. Springer Science & Business Media, 2018.

Iván Díaz, Nima S Hejazi, Kara E Rudolph, and Mark J van der Laan. Non-parametric efficient causal mediation with intermediate confounders. *Biometrika*, 2020. doi: 10.1093/biomet/asaa085. URL https://arxiv.org/abs/1912.09936.

Vanessa Didelez, Philip Dawid, and Sara Geneletti. Direct and indirect effects of sequential treatments. In *Proceedings of the 22nd Annual Conference on Uncertainty in Artificial Intelligence*, pages 138–146, 2006.

David Donoho. 50 years of data science. *Journal of Computational and Graphical Statistics*, 26(4):745–766, 2017.

Sandrine Dudoit and Mark J van der Laan. Asymptotics of cross-validated risk estimation in estimator selection and performance assessment. *Statistical Methodology*, 2(2): 131–154, 2005.

Ronald Aylmer Fisher. *Statistical Methods for Research Workers*. Oliver and Boyd, 10th edition, 1946.

Susan Gruber, Rachael V Phillips, Hana Lee, John Concato, and Mark van der Laan. Evaluating and improving real-world evidence with targeted learning. *arXiv preprint arXiv:2208.07283*, 2022.

Susan Gruber, Rachael V Phillips, Hana Lee, Martin Ho, John Concato, and Mark J van der Laan. Targeted Learning: Toward a future informed by real-world evidence. *Statistics in Biopharmaceutical Research*, 2023. doi: 10.1080/19466315.2023.2182356.

Sebastian Haneuse and Andrea Rotnitzky. Estimation of the effect of interventions that modify the received treatment. *Statistics in Medicine*, 32(30):5260–5277, 2013.

Nima S Hejazi. *Semiparametric statistical methods for causal inference with stochastic treatment regimes*. PhD thesis, University of California, Berkeley, 2021. URL https://www.stat.berkeley.edu/~nhejazi/publications/thesis-phd-biostat.pdf.

Nima S Hejazi, Jeremy R Coyle, and Mark J van der Laan. hal9001: Scalable highly adaptive lasso regression in R. *Journal of Open Source Software*, 2020a. doi: 10.21105/joss.02526. URL https://doi.org/10.21105/joss.02526.

Nima S Hejazi, Mark J van der Laan, Holly E Janes, Peter B Gilbert, and David C Benkeser. Efficient nonparametric inference on the effects of stochastic interventions under two-phase sampling, with applications to vaccine efficacy trials. *Biometrics*, 2020b. doi: 10.1111/biom.13375. URL https://arxiv.org/abs/2003.13771.

Nima S Hejazi, David C Benkeser, and Mark J van der Laan. *haldensify: Highly adaptive lasso conditional density estimation*, 2022a. URL https://doi.org/10.5281/zenodo.3698329. R package version 0.2.3.

Nima S Hejazi, Kara E Rudolph, Mark J van der Laan, and Iván Díaz. Nonparametric causal mediation analysis for stochastic interventional (in)direct effects. *Biostatistics*, (in press), 2022b. doi: 10.1093/biostatistics/kxac002. URL https://arxiv.org/abs/2009.06203.

Miguel A Hernán and James M Robins. *Causal Inference: What If*. CRC Press, 2022.

Paul W Holland. Statistics and causal inference. *Journal of the American Statistical Association*, 81(396):945–960, 1986.

Kosuke Imai, Luke Keele, and Teppei Yamamoto. Identification, inference and sensitivity analysis for causal mediation effects. *Statistical science*, pages 51–71, 2010.

Guido W Imbens and Donald B Rubin. *Causal inference in statistics, Social, and Biomedical Sciences*. Cambridge University Press, 2015.

Edward H Kennedy. Semiparametric theory and empirical processes in causal inference. In *Statistical Causal Inferences and Their Applications in Public Health Research*, pages 141–167. Springer, 2016.

Edward H Kennedy. Nonparametric causal effects based on incremental propensity score interventions. *Journal of the American Statistical Association*, 114(526):645–656, 2019.

Judith J Lok. 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, 2016.

Alex Luedtke and Mark J van der Laan. Super-learning of an optimal dynamic treatment rule. *International Journal of Biostatistics*, 12(1):305–332, 2016a.

Alexander R Luedtke and Mark J van der Laan. Optimal individualized treatments in resource-limited settings. *The International Journal of Biostatisics*, 12(1):283–303, 2016b. doi: 10.1515/ijb-2015-0007.

Lina M Montoya, Mark J van der Laan, Alexander R Luedtke, Jennifer L Skeem, Jeremy R Coyle, and Maya L Petersen. The optimal dynamic treatment rule superlearner: considerations, performance, and application to criminal justice interventions. *The International Journal of Biostatistics*, 19(1):217–238, 2023a. doi: 10.1515/ijb-2020-0127.

Lina M Montoya, Mark J van der Laan, Jennifer L Skeem, and Maya L Petersen. Estimators for the value of the optimal dynamic treatment rule with application to criminal justice interventions. *The International Journal of Biostatistics*, 19(1): 239–259, 2023b. doi: 10.1515/ijb-2020-0128.

Marcus R Munafò, Brian A Nosek, Dorothy VM Bishop, Katherine S Button, Christopher D Chambers, Nathalie Percie Du Sert, Uri Simonsohn, Eric-Jan Wagenmakers, Jennifer J Ware, and John PA Ioannidis. A manifesto for reproducible science. *Nature Human Behaviour*, 1(1):0021, 2017.

Susan A Murphy. Optimal dynamic treatment regimes. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 65(2):331–355, 2003.

Ashley I Naimi and Laura B Balzer. Stacked generalization: an introduction to super learning. *European Journal of Epidemiology*, 33(5):459–464, 2018.

Nature Editorial (Anonymous). How scientists fool themselves — and how they can stop. *Nature*, 526(7572), Oct 2015a.

Nature Editorial (Anonymous). Let's think about cognitive bias. *Nature*, 526(7572), Oct 2015b. doi: 10.1038/526163a.

Jerzy Neyman. Contribution to the theory of sampling human populations. *Journal of the American Statistical Association*, 33(201):101–116, 1938.

Trang Quynh Nguyen, Ian Schmid, and Elizabeth A Stuart. Clarifying causal mediation analysis for the applied researcher: Defining effects based on what we want to learn. *arXiv preprint arXiv:1904.08515*, 2019.

Brian A Nosek, Charles R Ebersole, Alexander C DeHaven, and David T Mellor. The preregistration revolution. *Proceedings of the National Academy of Sciences*, 115(11): 2600–2606, 2018.

Judea Pearl. Causal diagrams for empirical research. *Biometrika*, 82(4):669–688, 1995.

Judea Pearl. Direct and indirect effects. *arXiv preprint arXiv:1301.2300*, 2001.

Judea Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, 2009.

Judea Pearl. Brief report: On the consistency rule in causal inference: "axiom, definition, assumption, or theorem?". *Epidemiology*, pages 872–875, 2010.

Roger Peng. The reproducibility crisis in science: A statistical counterattack. *Significance*, 12(3):30–32, 2015.

Maya L Petersen, Sandra E Sinisi, and Mark J van der Laan. Estimation of direct causal effects. *Epidemiology*, pages 276–284, 2006.

Rachael V Phillips, Mark J van der Laan, Hana Lee, and Susan Gruber. Practical considerations for specifying a super learner. *International Journal of Epidemiology*, 2023. doi: 10.1093/ije/dyad023.

Eric C Polley and Mark J van der Laan. Super learner in prediction. Technical report, Division of Biostatistics, University of California, Berkeley, 2010.

Karl Popper. *The Logic of Scientific Discovery*. Routledge, 1934.

Eleanor M Pullenayegum, Robert W Platt, Melanie Barwick, Brian M Feldman, Martin Offringa, and Lehana Thabane. 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, 2016.

R Core Team. R: A language and environment for statistical computing, 2021. URL https://www.R-project.org/.

James Robins. 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, 1986a. ISSN 0270-0255. doi: https://doi.org/10.1016/0270-0255(86)90088-6.

James Robins and Andrea Rotnitzky. Discussion of "dynamic treatment regimes: Technical challenges and applications". *Electron. J. Statist.*, 8(1):1273–1289, 2014. doi: 10.1214/14-EJS908. URL https://doi.org/10.1214/14-EJS908.

James M Robins. 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, 1986b.

James M Robins. 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, 2004. doi: 10.1007/978-1-4419-9076-1_11. URL https://doi.org/10.1007/978-1-4419-9076-1_11.

James M Robins and Sander Greenland. Identifiability and exchangeability for direct and indirect effects. *Epidemiology*, pages 143–155, 1992.

James M Robins and Thomas S Richardson. Alternative graphical causal models and the identification of direct effects. *Causality and psychopathology: Finding the determinants of disorders and their cures*, pages 103–158, 2010.

Donald B Rubin. Bayesian inference for causal effects: The role of randomization. *The Annals of statistics*, pages 34–58, 1978.

Donald B Rubin. Randomization analysis of experimental data: The fisher randomization test comment. *Journal of the American Statistical Association*, 75(371):591–593, 1980.

Donald B Rubin. Causal inference using potential outcomes: Design, modeling, decisions. *Journal of the American Statistical Association*, 100(469):322–331, 2005.

Kara E Rudolph, Oleg Sofrygin, Wenjing Zheng, and Mark J van der Laan. Robust and flexible estimation of stochastic mediation effects: a proposed method and example in a randomized trial setting. *Epidemiologic Methods*, 7(1), 2017.

Peter Spirtes, Clark N Glymour, Richard Scheines, David Heckerman, Christopher Meek, Gregory Cooper, and Thomas Richardson. *Causation, Prediction, and Search*. MIT press, 2000.

Philip B Stark and Andrea Saltelli. Cargo-cult statistics and scientific crisis. *Significance*, 15(4):40–43, 2018.

James H Stock. Nonparametric policy analysis. *Journal of the American Statistical Association*, 84(406):567–575, 1989.

Arnold Stromberg et al. Why write statistical software? the case of robust statistical methods. *Journal of Statistical Software*, 10(5):1–8, 2004.

Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

Denes Szucs and John Ioannidis. When null hypothesis significance testing is unsuitable for research: a reassessment. *Frontiers in Human Neuroscience*, 11:390, 2017.

Eric J Tchetgen Tchetgen. Inverse odds ratio-weighted estimation for causal mediation analysis. *Statistics in Medicine*, 32(26):4567–4580, 2013.

Eric J Tchetgen Tchetgen and Ilya Shpitser. Semiparametric theory for causal mediation analysis: efficiency bounds, multiple robustness, and sensitivity analysis. *Annals of Statistics*, 40(3):1816–1845, jun 2012. doi: 10.1214/12-AOS990.

Eric J Tchetgen Tchetgen and Tyler J VanderWeele. On identification of natural direct effects when a confounder of the mediator is directly affected by exposure. *Epidemiology*, 25(2):282, 2014.

Johannes Textor, Juliane Hardt, and Sven Knüppel. DAGitty: a graphical tool for analyzing causal diagrams. *Epidemiology*, 22(5):745, 2011.

Fahmida Tofail, Lia CH Fernald, Kishor K Das, Mahbubur Rahman, Tahmeed Ahmed, Kaniz K Jannat, Leanne Unicomb, Benjamin F Arnold, Sania Ashraf, Peter J Winch, et al. 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, 2018.

John W Tukey. The future of data analysis. *The Annals of Mathematical Statistics*, 33(1): 1–67, 1962.

Mark J van der Laan and Sandrine Dudoit. 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, 2003.

Mark J van der Laan and Alex Luedtke. Targeted learning of the mean outcome under an optimal dynamic treatment rule. *Journal of Causal Inference*, 3(1):61–95, 2015.

Mark J van der Laan and Sherri Rose. *Targeted learning: causal inference for observational and experimental data*. Springer Science & Business Media, 2011.

Mark J van der Laan and Sherri Rose. *Targeted Learning in Data Science: Causal Inference for Complex Longitudinal Studies*. Springer Science & Business Media, 2018.

Mark J van der Laan and Richard JCM Starmans. Entering the era of data science: Targeted learning and the integration of statistics and computational data analysis. *Advances in Statistics*, 2014, 2014.

Mark J van der Laan, Sandrine Dudoit, and Sunduz Keles. Asymptotic optimality of likelihood-based cross-validation. *Statistical Applications in Genetics and Molecular Biology*, 3(1):1–23, 2004.

Mark J van der Laan, Eric C Polley, and Alan E Hubbard. Super Learner. *Statistical Applications in Genetics and Molecular Biology*, 6(1), 2007.

Aad W van der Vaart, Sandrine Dudoit, and Mark J van der Laan. Oracle inequalities for multi-fold cross validation. *Statistics & Decisions*, 24(3):351–371, 2006.

Tyler VanderWeele. *Explanation in Causal Inference: Methods for Mediation and Interaction*. Oxford University Press, 2015.

Tyler J VanderWeele, Stijn Vansteelandt, and James M Robins. Effect decomposition in the presence of an exposure-induced mediator-outcome confounder. *Epidemiology*, 25 (2):300, 2014.

Stijn Vansteelandt and Rhian M Daniel. Interventional effects for mediation analysis with multiple mediators. *Epidemiology*, 28(2):258, 2017.

Stijn Vansteelandt and Tyler J VanderWeele. Natural direct and indirect effects on the exposed: effect decomposition under weaker assumptions. *Biometrics*, 68(4): 1019–1027, 2012.

Stijn Vansteelandt, Maarten Bekaert, and Theis Lange. Imputation strategies for the estimation of natural direct and indirect effects. *Epidemiologic Methods*, 1(1):131–158, 2012.

Hadley Wickham. *Advanced r*. Chapman and Hall/CRC, 2014.

Sewall Wright. The method of path coefficients. *The Annals of Mathematical Statistics*, 5 (3):161–215, 1934.

Jessica G Young, Miguel A Hernán, and James M Robins. 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, 2014.

Baqun Zhang, Anastasios A Tsiatis, Marie Davidian, Min Zhang, and Eric Laber. Estimating optimal treatment regimes from a classification perspective. *Stat*, 5(1): 278–278, 2016. doi: 10.1002/sta4.124.

Yingqi Zhao, Donglin Zeng, A John Rush, and Michael R Kosorok. Estimating individualized treatment rules using outcome weighted learning. *Journal of the American Statistical Association*, 107(499):1106–1118, 2012. doi: 10.1080/01621459.2012.695674. PMID: 23630406.

Wenjing Zheng and Mark J van der Laan. Cross-validated targeted minimum-loss-based estimation. In Mark J van der Laan and Sherri Rose, editors, *Targeted Learning: Causal Inference for Observational and Experimental Data*, pages 459–474. Springer, 2011. doi: 10.1007/978-1-4419-9782-1_27.

Wenjing Zheng and Mark J van der Laan. Targeted maximum likelihood estimation of natural direct effects. *International Journal of Biostatistics*, 8(1), 2012. ISSN 15574679. doi: 10.2202/1557-4679.1361.