[Workshop] Targeted Learning in the tlverse

Causal Inference Meets Machine Learning

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

updated: October 12, 2021

Contents

0.1 Reproduciblity with the `tlverse` 13

0.2 Setup instructions 14

0.2.1 R and RStudio 14

1 Welcome to the `tlverse` 21

1.1 Learning Objectives 21

1.2 What is the `tlverse`? 21

1.3 `tlverse` components 22

1.4 Installation 23

2 The Roadmap for Targeted Learning 25

2.1 Learning Objectives 25

2.2 Introduction 25

2.3 The Roadmap 26

2.4 Summary of the Roadmap 29

2.5 Causal Target Parameters 30

2.5.1 The Causal Model 30

2.5.2 Identifiability 32

2.6 The WASH Benefits Example Dataset 33

3 Super (Machine) Learning **37**

3.0.1 General Overview of the Algorithm 41

3.1 Exercises 61

3.1.1 Predicting Myocardial Infarction with `sl3` 61

3.2 Concluding Remarks 62

4 The TMLE Framework **65**

4.1 Learning Objectives 65

4.2 Introduction 65

4.3 Substitution Estimators 66

4.4 Targeted Maximum Likelihood Estimation 68

4.4.1 TMLE Updates 68

4.4.2 Statistical Inference 69

4.5 Easy-Bake Example: `tmle3` for ATE 69

4.5.1 Load the Data 69

4.5.2 Define the variable roles 70

4.5.3 Handle Missingness 70

4.5.4 Create a "Spec" Object 71

4.5.5 Define the learners 71

4.5.6 Fit the TMLE 72

4.5.7 Evaluate the Estimates 72

4.6 `tmle3` Components 73

4.6.1 `tmle3_task` 73

4.6.2 Initial Likelihood 73

4.6.3 Targeted Likelihood (updater) 74

4.6.4 Parameter Mapping 75

4.6.5 Putting it all together 75

4.7 Fitting `tmle3` with multiple parameters 75

4.7.1 Delta Method . . . 76
4.7.2 Fit . . . 76
4.8 Exercises . . . 78
4.8.1 Estimation of the ATE with `tmle3` . . . 78
4.8.2 Estimation of Strata-Specific ATEs with `tmle3` . . . 79
4.9 Summary . . . 80

5 Optimal Individualized Treatment Regimes (optional) 81
5.1 Learning Objectives . . . 81
5.2 Introduction to Optimal Individualized Interventions . . . 82
5.3 Data Structure and Notation . . . 84
5.4 Defining the Causal Effect of an Optimal Individualized Intervention 84
5.4.1 Why CV-TMLE? . . . 85
5.5 Binary Treatment . . . 86
5.5.1 Evaluating the Causal Effect of an optimal ITR with Binary Treatment . . . 87
5.6 Categorical Treatment . . . 90
5.6.1 Evaluating the Causal Effect of an optimal ITR with Categorical Treatment . . . 91
5.7 Extensions to Causal Effect of an OIT . . . 94
5.7.1 Simpler Rules . . . 94
5.7.2 Realistic Optimal Individual Regimes . . . 95
5.7.3 Variable Importance Analysis . . . 96
5.8 Exercise . . . 97
5.8.1 Real World Data and `tmle3mopttx` . . . 97
5.9 Summary . . . 99
5.9.1 Solutions . . . 99

6 Stochastic Treatment Regimes (optional) 103

6.1 Learning Objectives . . . 103

6.2 Introduction . . . 104

6.3 Stochastic Interventions . . . 104

6.3.1 Identifying the Causal Effect of a Stochastic Intervention . . . 105

6.4 Estimating the Causal Effect of a Stochastic Intervention with `tmle3shift` . . . 106

6.4.1 Simulate Data . . . 108

6.4.2 Targeted Estimation of Stochastic Interventions Effects . . . 109

6.5 Stochastic Interventions over a Grid of Counterfactual Shifts . . . 110

6.5.1 Initializing `vimshift` through its `tmle3_Spec` . . . 111

6.5.2 Targeted Estimation of Stochastic Intervention Effects . . . 111

6.5.3 Inference with Marginal Structural Models . . . 112

6.5.4 Directly Targeting the MSM Parameter β . . . 113

6.5.5 Example with the WASH Benefits Data . . . 114

6.6 Exercises . . . 117

7 A Primer on the `R6` Class System 119

7.1 Classes, Fields, and Methods . . . 119

7.2 Object Oriented Programming: `Python` and `R` . . . 120

List of Tables

List of Figures

5.1 Illustration of a Dynamic Treatment Regime in a Clinical Setting . . 83

Welcome!

This open source, reproducible vignette is for a half-day workshop on the Targeted Learning framework for statistical and causal inference with machine learning. Beyond introducing Targeted Learning, the workshop focuses on applying the methodology in practice using the `tlverse` software ecosystem. These materials are based on a working draft of the book *Targeted Learning in R: Causal Data Science with the tlverse Software Ecosystem*, which includes in-depth discussion of these topics and much more, and may serve as a useful reference to accompany these workshop materials.

Important links

- **Software installation**: Please install the relevant software before the workshop using the installation script.
- You will probably exceed the GitHub API rate limit during this installation, which will throw an error. This issue and the solution are addressed here.
- **Code**: `R` script files for each section of the workshop are available via the GitHub repository for the workshop at `https://github.com/tlverse/tlverse-workshops/tree/master/R_code`

About this workshop

This workshop will provide a comprehensive introduction to the field of *Targeted Learning* for causal inference, and the corresponding `tlverse` software ecosystem. Emphasis will be placed on targeted minimum loss-based estimators of the

causal effects of single timepoint interventions, including extensions for missing covariate and outcome data. These multiply robust, efficient plug-in estimators use state-of-the-art, ensemble machine learning tools to flexibly adjust for confounding while yielding valid statistical inference. In particular, we will discuss targeted estimators of the causal effects of static and dynamic interventions; time permitting, additional topics to be discussed will include estimation of the causal effects of optimal dynamic and stochastic interventions.

In addition to discussion, this workshop will incorporate both interactive activities and hands-on, guided `R` programming exercises, to allow participants the opportunity to familiarize themselves with methodology and tools that will translate to real-world data analysis. It is highly recommended for participants to have an understanding of basic statistical concepts such as confounding, probability distributions, confidence intervals, hypothesis testing, and regression. Advanced knowledge of mathematical statistics is useful but not necessary. Familiarity with the `R` programming language will be essential.

Outline

- *Warm-up*: The Roadmap of Targeted Learning and Why We Need A Statistical Revolution with an *introductory video lecture by Mark van der Laan and Alan Hubbard* (**Please watch this hour-long lecture before the workshop.**)
- 09:00-09:30A: Introduction to the `tlverse` Software Ecosystem and the WASH Benefits data
- 09:30-10:00A: Super learning with the `sl3` `R` package
- 10:00-11:00A: Programming exercises with `sl3`
- 11:00-11:15A: Morning Coffee Break and Q&A
- 11:15-12:00P: Targeted Learning for causal inference with the `tmle3` `R` package
- 12:00-12:45P: Programming exercises with `tmle3`
- 12:45-01:30P: Lunch Break
- 01:30-02:15P: Optimal treatment regimes with the `tmle3mopttx` `R` package
- 02:15-03:00P: Programming exercises with `tmle3mopttx`
- 03:00-03:15P: Afternoon Coffee Break
- 03:15-04:00P: Stochastic treatment regimes with the `tmle3shift` `R` package
- 04:00-04:30P: Programming exercises with `tmle3shift`
- 04:30-05:00P: Concluding remarks and discussion

About the instructors

Mark van der Laan

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

Alan Hubbard

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

Jeremy Coyle

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

Nima Hejazi

Nima Hejazi is a PhD candidate in biostatistics, working under the collaborative direction of Mark van der Laan and Alan Hubbard. Nima is affiliated with UC Berkeley's Center for Computational Biology and NIH Biomedical Big Data training program, as well as with the Fred Hutchinson Cancer Research Center. Previously, he earned an MA in Biostatistics and a BA (with majors in Molecular and Cell Biology, Psychology, and Public Health), both at UC Berkeley. His research interests fall at the intersection of causal inference and machine learning, drawing on ideas from non/semi-parametric estimation in large, flexible statistical models to develop efficient and robust statistical procedures for evaluating complex target estimands in observational and randomized studies. Particular areas of current emphasis include mediation/path analysis, outcome-dependent sampling designs, targeted loss-based estimation, and vaccine efficacy trials. Nima is also passionate about statistical computing and open source software development for applied statistics.

Ivana Malenica

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

Rachael Phillips

Rachael Phillips is a PhD student in biostatistics, advised by Alan Hubbard and Mark van der Laan. She has an MA in Biostatistics, BS in Biology, and BA in Mathematics. A student of targeted learning and causal inference, Rachael's research integrates semiparametric statistical estimation and inference. She is motivated by applied projects and some of her current work involves personalized online learning from data streams of vital signs, human-computer interaction, automated machine learning, and developing statistical analysis plans using targeted learning.

0.1 Reproduciblity with the tlverse

The tlverse software ecosystem is a growing collection of packages, several of which are quite early on in the software lifecycle. The team does its best to maintain backwards compatibility. Once this work reaches completion, the specific versions of the tlverse packages used will be archived and tagged to produce it.

This book was written using bookdown, and the complete source is available on GitHub. This version of the book was built with R version 4.1.1 (2021-08-10), pandoc version 2.7.3, and the following packages:

package	version	source
bookdown	0.24.1	Github (rstudio/bookdown@84bde8e)
bslib	0.3.1	Github (rstudio/bslib@efc475c)
data.table	1.14.2	CRAN (R 4.1.1)
delayed	0.4.0	Github (tlverse/delayed@bf7ca82)
devtools	2.4.2	CRAN (R 4.1.1)
downlit	0.2.1	CRAN (R 4.1.1)
dplyr	1.0.7	CRAN (R 4.1.1)
ggplot2	3.3.5	CRAN (R 4.1.1)
here	1.0.1	CRAN (R 4.1.1)
kableExtra	1.3.4	CRAN (R 4.1.1)
knitr	1.36	CRAN (R 4.1.1)
mvtnorm	1.1-2	CRAN (R 4.1.1)
origami	1.0.5	Github (tlverse/origami@e1b8fe6)
readr	2.0.2	CRAN (R 4.1.1)
rmarkdown	2.11	CRAN (R 4.1.1)
skimr	2.1.3	CRAN (R 4.1.1)
sl3	1.4.3	Github (tlverse/sl3@20834ae)
stringr	1.4.0	CRAN (R 4.1.1)
tibble	3.1.5	CRAN (R 4.1.1)
tidyr	1.1.4	CRAN (R 4.1.1)
tidyverse	1.3.1	CRAN (R 4.1.1)
tmle3	0.2.0	Github (tlverse/tmle3@425e21c)
tmle3mopttx	0.1.0	Github (tlverse/tmle3mopttx@9fb1a3b)
tmle3shift	0.2.0	Github (tlverse/tmle3shift@43f6fc0)

0.2 Setup instructions

0.2.1 R and RStudio

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

0.2.1.1 Windows

0.2.1.1.1 If you already have R and RStudio installed

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

0.2.1.1.2 If you don't have R and RStudio installed

- Download R from the CRAN website.
- Run the `.exe` file that was just downloaded
- Go to the RStudio download page
- Under *Installers* select **RStudio x.yy.zzz - Windows XP/Vista/7/8** (where x, y, and z represent version numbers)
- Double click the file to install it
- Once it's installed, open RStudio to make sure it works and you don't get any error messages.

0.2.1.2 macOS / Mac OS X

0.2.1.2.1 If you already have R and RStudio installed

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

0.2.1.2.2 If you don't have R and RStudio installed

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

0.2.1.3 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 at least R 3.3.1.
- Go to the RStudio download page
- Under *Installers* select the version that matches your distribution, and install it with your preferred method (e.g., with Debian/Ubuntu `sudo dpkg -i rstudio-x.yy.zzz-amd64.deb` at the terminal).
- Once it's installed, open RStudio to make sure it works and you don't get any error messages.

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

Motivation

> "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 (2015b)

Scientific research is at a unique point in history. The need to improve rigor and reproducibility in our field is greater than ever; corroboration moves science forward, yet there is a growing alarm about results that cannot be reproduced and that report false discoveries (Baker, 2016). Consequences of not meeting this need will result in further decline in the rate of scientific progression, the reputation of the sciences, and the public's trust in its findings (Munafò et al., 2017; Nature Editorial, 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, at its current state the culture of data analysis and statistics actually enables human bias through improper model selection. All hypothesis tests and estimators are derived from statistical models, so to obtain valid estimates and inference it is critical that the statistical model contains the process that generated the data. Perhaps treatment was randomized or only depended on a small number of baseline covariates; this knowledge should and can be incorporated in the model. Alternatively, maybe the data is observational, and there is no knowledge about the data-generating process (DGP). If this is the case, then the statistical model should contain *all* data distributions. In practice; however, models are not selected based on knowledge of the DGP, instead models are often selected based on (1)

the p-values they yield, (2) their convenience of implementation, and/or (3) an analysts loyalty to a particular model. This practice of "cargo-cult statistics — the ritualistic miming of statistics rather than conscientious practice," (Stark and Saltelli, 2018) is characterized by arbitrary modeling choices, even though these choices often result in different answers to the same research question. That is, "increasingly often, [statistics] is used instead to aid and abet weak science, a role it can perform well when used mechanically or ritually," as opposed to its original purpose of safeguarding against weak science (Stark and Saltelli, 2018). This presents a fundamental drive behind the epidemic of false findings that scientific research is suffering from (van der Laan and Starmans, 2014).

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

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

Our objective is to provide training to students, researchers, industry professionals, faculty in science, public health, statistics, and other fields to empower them with the necessary knowledge and skills to utilize the sound methodology of Targeted Learning — a technique that provides tailored pre-specified machines for answering queries, so that each data analysis is completely reproducible, and estimators are efficient, minimally biased, and provide formal statistical inference.

Just as the conscientious use of modern statistical methodology is necessary to ensure that scientific practice thrives, it remains critical to acknowledge the role that robust software plays in allowing practitioners direct access to published results. We recall 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," thus making the availability and adoption of robust statistical software key to enhancing the transparency that is an inherent aspect of science (Buckheit and Donoho, 1995).

For a statistical methodology to be readily accessible in practice, it is crucial that it is accompanied by robust user-friendly software (Pullenayegum et al., 2016; Stromberg et al., 2004). The `tlverse` software ecosystem was developed to fulfill this need for the Targeted Learning methodology. Not only does this software facilitate computationally reproducible and efficient analyses, it is also a tool for Targeted Learning education since its workflow mirrors that of the methodology. In particular, the `tlverse` paradigm does not focus on implementing a specific estimator or a small set of related estimators. Instead, the focus is on exposing the statistical framework of Targeted Learning itself — all `R` 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` `R` packages share a core set of design principles centered on extensibility, allowing for them to be used in conjunction with each other and built upon one other in a cohesive fashion.

In this workshop, the reader will embark on a journey through the `tlverse` ecosystem. Guided by `R` programming exercises, case studies, and intuitive explanation readers will build a toolbox for applying the Targeted Learning statistical methodology, which will translate to real-world causal inference analyses. Participants need not be a fully trained statistician to begin understanding and applying these methods.

However, it is highly recommended for participants 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 introductory materials for learning the `R` programming language we recommend the following free resources:

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

For causal inference learning materials we recommend the following resources:

- Hernán MA, Robins JM (2019). *Causal Inference.*
- Jason A. Roy's coursera Course *A Crash Course in Causality: Inferring Causal Effects from Observational Data*

Chapter 1

Welcome to the `tlverse`

1.1 Learning Objectives

1. Understand the `tlverse` ecosystem conceptually
2. Identify the core components of the `tlverse`
3. Install `tlverse` `R` packages
4. Understand the Targeted Learning roadmap
5. Learn about the WASH Benefits example data

1.2 What is the `tlverse`?

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

By analogy to the `tidyverse`:

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

So, the `tlverse` is

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

1.3 tlverse components

These are the main packages that represent the **core** of the tlverse:

- sl3: Modern Super Learning with Pipelines
  - *What?* A modern object-oriented re-implementation of the Super Learner algorithm, employing recently developed paradigms for R programming.
  - *Why?* A design that leverages modern tools for fast computation, is forward-looking, and can form 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 is still flexible enough to remain the engine even as new techniques are developed.

In addition to the engines that drive development in the tlverse, there are some supporting packages – in particular, we have two...

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

A key principle of the tlverse is extensibility. That is, we want to support new Targeted Learning estimators as they are developed. The model for this is new estimators are implemented in additional packages using the core packages above. There are currently two featured examples of this:

- `tmle3mopttx`: Optimal Treatments in `tlverse`
  - *What?* Learn an optimal rule and estimate the mean outcome under the rule
  - *Why?* Optimal Treatment is a powerful tool in precision healthcare and other settings where a one-size-fits-all treatment approach is not appropriate.
- `tmle3shift`: Shift Interventions in `tlverse`
  - *What?* Shift interventions for continuous treatments
  - *Why?* Not all treatment variables are discrete. Being able to estimate the effects of continuous treatment represents a powerful extension of the Targeted Learning approach.

1.4 Installation

The `tlverse` ecosystem of packages are currently hosted at `https://github.com/tlverse`, not yet on CRAN. You can use the `devtools` 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.
2. Create a Personal Access Token simply by clicking "Generate token" at the bottom of the page.
3. Copy your Personal Access Token, 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. If you are not able to access your `.Renviron` file with this command, then try inputting `Sys.setenv(GITHUB_PAT = )` with your Personal Access Token inserted as a string after the equals symbol; and if this does not error, then skip to step 8.
5. In your `.Renviron` file, type `GITHUB_PAT=` and then paste your Personal Access Token after the equals symbol with no space.
6. In your `.Renviron` file, press the enter key to ensure that your `.Renviron` ends with a newline.
7. Save your `.Renviron` file.
8. Restart R for changes to take effect. You can restart R via the drop-down menu on the "Session" tab. The "Session" tab is at the top of the RStudio interface.

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

Chapter 2

The Roadmap for Targeted Learning

2.1 Learning Objectives

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

1. Translate scientific questions to statistical questions.
2. Define a statistical model based on the knowledge of the experiment that generated the data.
3. Identify a causal parameter as a function of the observed data distribution.
4. Explain the following causal and statistical assumptions and their implications: i.i.d., consistency, interference, positivity, SUTVA.

2.2 Introduction

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

2.3 The Roadmap

Following the roadmap is a process of five stages.

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

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

The data set we're confronted with is the result of an experiment and we can view the data as a random variable, O, because if we repeat the experiment we would have a different realization of this experiment. In particular, if we repeat the experiment many times we could learn the probability distribution, P_0, of our data. So, the observed data O with probability distribution P_0 are n independent identically distributed (i.i.d.) observations of the random variable $O; O_1, \ldots, O_n$. Note that while not all data are i.i.d., there are ways to handle non-i.i.d. data, such as establishing conditional independence, stratifying data to create sets of identically distributed data, etc. It is crucial that researchers be absolutely clear about what they actually know about the data-generating distribution for a given problem of interest. Unfortunately, communication between statisticians and researchers is often fraught with misinterpretation. The roadmap provides a mechanism by which to ensure clear communication between research and statistician – it truly helps with this communication!

The empirical probability measure, P_n

Once we have n of such i.i.d. observations we have 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 our data. For example, we can define the empirical probability measure of a set, A, to be the proportion of observations which end up in A. That is,

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

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

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

The statistical model $\mathcal{M}$ is defined by the question we asked at the end of **??**. It is defined as the set of possible probability distributions for our observed data. Often $\mathcal{M}$ is very large (possibly infinite-dimensional), to reflect the fact that statistical knowledge is limited. In the case that $\mathcal{M}$ is infinite-dimensional, we deem this a nonparametric statistical model.

Alternatively, if the probability distribution of the data at hand is described by a finite number of parameters, then the statistical model is parametric. In this case, we prescribe to the belief that the random variable O being observed has, e.g., a normal distribution with mean μ and variance σ^2. More formally, a parametric model may be defined

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

Sadly, the assumption that the data-generating distribution has a specific, parametric forms is all-too-common, even when such is a leap of faith. This practice of oversimplification in the current culture of data analysis typically derails any attempt at trying to answer the scientific question at hand; alas, such statements as the ever-popular quip of Box that "All models are wrong but some are useful," encourage the data analyst to make arbitrary choices even when that often force significant differences in answers to the same estimation problem. The Targeted Learning paradigm does not suffer from this bias since it defines the statistical model through a representation of the true data-generating distribution corresponding to the observed data.

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

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

The statistical target parameter, Ψ, is defined as a mapping from the statistical model, $\mathcal{M}$, to the parameter space (i.e., a real number) $\mathcal{R}$. That is, $\Psi : \mathcal{M} \to \mathbb{R}$. The target parameter may be seen as a representation of the quantity that we wish to learn from the data, the answer to a well-specified (often causal) question of interest. In contrast to purely statistical target parameters, causal target parameters require

identification from the observed data, based on causal models that include several untestable assumptions, described in more detail in the section on causal target parameters.

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

$$\Psi(P_0) = \mathbb{P}(O > 5)$$

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

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

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

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

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

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

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

different realization of our estimate and our estimator has a sampling distribution. The sampling distribution of an estimator can be theoretically validated to be approximately normally distributed by a Central Limit Theorem (CLT).

A class of **Central Limit Theorems** (CLTs) are statements regarding the convergence of the **sampling distribution of an estimator** to a normal distribution. In general, we will construct estimators whose limit sampling distributions may be shown to be approximately normal distributed as sample size increases. For large enough n we have,

$$\hat{\Psi}(P_n) \sim N\left(\Psi(P_0), \frac{\sigma^2}{n}\right),$$

permitting statistical inference. Now, we can proceed to quantify the uncertainty of our chosen estimator by construction of hypothesis tests and confidence intervals. For example, we may construct a confidence interval at level $(1-\alpha)$ for our estimand, $\Psi(P_0)$:

$$\hat{\Psi}(P_n) \pm z_{1-\frac{\alpha}{2}}\left(\frac{\sigma}{\sqrt{n}}\right),$$

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

Note: we will typically have to estimate the standard error, $\frac{\sigma}{\sqrt{n}}$.

A 95% confidence interval means that if we were to take 100 different samples of size n and compute a 95% confidence interval for each sample then approximately 95 of the 100 confidence intervals would contain the estimand, $\Psi(P_0)$. More practically, this means that there is a 95% probability (or 95% confidence) that the confidence interval procedure will contain the true estimand. However, any single estimated confidence interval either will contain the true estimand or will not.

2.4 Summary of the Roadmap

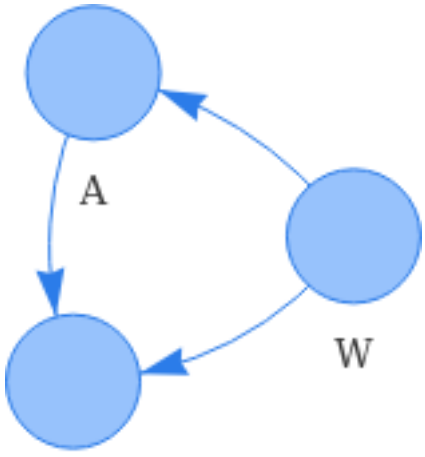
Data, O, is viewed as a random variable that has a probability distribution. We often have n units of independent identically distributed units with probability distribution P_0 ($O_1, \ldots, O_n \sim P_0$). We have statistical knowledge about the experiment that generated this data. In other words, we make a statement that the true data distribution P_0 falls in a certain set called a statistical model, $\mathcal{M}$. Often these

sets are very large because statistical knowledge is very limited so these statistical models are often infinite dimensional models. Our statistical query is, "What are we trying to learn from the data?" denoted by the statistical target parameter, Ψ, which maps the P_0 into the estimand, $\Psi(P_0)$. At this point the statistical estimation problem is formally defined and now we will need statistical theory to guide us in the construction of estimators. There's a lot of statistical theory we will review in this course that, in particular, relies on the Central Limit Theorem, allowing us to come up with estimators that are approximately normally distributed and also allowing us to come with statistical inference (i.e., confidence intervals and hypothesis tests).

2.5 Causal Target Parameters

2.5.1 The Causal Model

After formalizing the data and the statistical model, we can define a causal model to express causal parameters of interest. Directed acyclic graphs (DAGs) are one useful tool to express what we know about the causal relations among variables. Ignoring exogenous U terms (explained below), we assume the following ordering of the variables in the observed data O.



\begin{center}

While directed acyclic graphs (DAGs) like above provide a convenient means by which to visualize causal relations between variables, the same causal relations among variables can be represented via a set of structural equations, which define the

non-parametric structural equation model (NPSEM):

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

where U_W, U_A, and U_Y represent the unmeasured exogenous background characteristics that influence the value of each variable. In the NPSEM, f_W, f_A and f_Y denote that each variable (for W, A and Y, respectively) is a function of its parents and unmeasured background characteristics, but note that there is no imposition of any particular functional constraints(e.g., linear, logit-linear, only one interaction, etc.). For this reason, they are called non-parametric structural equation models (NPSEMs). The DAG and set of nonparametric structural equations represent exactly the same information and so may be used interchangeably.

The first hypothetical experiment we will consider is assigning exposure to the whole population and observing the outcome, and then assigning no exposure to the whole population and observing the outcome. On the nonparametric structural equations, this corresponds to a comparison of the outcome distribution in the population under two interventions:

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

These interventions imply two new nonparametric structural equation models. For the case $A = 1$, we have

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

and 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 we have intervened on the system, setting A deterministically to either of the values 1 or 0. The new symbols $Y(1)$ and $Y(0)$ indicate the outcome variable in our population if it were generated by the respective NPSEMs above; these are often called *counterfactuals* (since they

run contrary-to-fact). The difference between the means of the outcome under these two interventions defines a parameter that is often called the "average treatment effect" (ATE), denoted

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

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

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

2.5.2 Identifiability

Because we can never observe both $Y(0)$ (the counterfactual outcome when $A = 0$) and $Y(1)$ (similarly, the counterfactual outcome when $A = 1$), we cannot estimate 2.1 directly. Instead, we have to make assumptions under which this quantity may be estimated from the observed data $O \sim P_0$ under the data-generating distribution P_0. Fortunately, given the causal model specified in the NPSEM above, we can, with a handful of untestable assumptions, estimate the ATE, even from observational data. These assumptions may be summarized as follows

1. The causal graph implies $Y(a) \perp A$ for all $a \in \mathcal{A}$, which is the *randomization* assumption. In the case of observational data, the analogous assumption is *strong ignorability* or *no unmeasured confounding* $Y(a) \perp A \mid W$ for all $a \in \mathcal{A}$;
2. Although not represented in the causal graph, also required is the assumption of no interference between units, that is, the outcome for unit i Y_i is not affected by exposure for unit j A_j unless $i = j$;
3. *Consistency* of the treatment mechanism is also required, i.e., the outcome for unit i is $Y_i(a)$ whenever $A_i = a$, an assumption also known as "no other versions of treatment";
4. It is also necessary that all observed units, across strata defined by W, have a bounded (non-deterministic) probability of receiving treatment – that is, $0 < \mathbb{P}(A = a \mid W) < 1$ for all a and W). This assumption is referred to as *positivity* or *overlap*.

Remark: Together, (2) and (3), the assumptions of no interference and consistency, respectively, are jointly referred to as the *stable unit treatment value assumption* (SUTVA).

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

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

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

2.6 The WASH Benefits Example Dataset

The data come from a study of the effect of water quality, sanitation, hand washing, and nutritional interventions on child development in rural Bangladesh (WASH Benefits Bangladesh): a cluster-randomised controlled trial (Luby 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-randomised, using a random number generator, into six intervention groups (all of which received weekly visits from a community health promoter for the first 6 months and every 2 weeks for the next 18 months) and a double-sized control group (no intervention or health promoter visit). The six intervention groups were:

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

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

```
library(tidyverse)

# read in data
dat <- read_csv("https://raw.githubusercontent.com/tlverse/tlverse-data/master/w
dat
# A tibble: 4,695 x 28
    whz tr      fracode month  aged sex    momage momedu momheight hfiacat Nlt18
  <dbl> <chr>   <chr>   <dbl> <dbl> <chr>   <dbl> <chr>       <dbl> <chr>   <dbl>
1  0    Control N05265      9   268 male       30 Prima~       146. Food S~     3
2 -1.16 Control N05265      9   286 male       25 Prima~       149. Modera~     2
3 -1.05 Control N08002      9   264 male       25 Prima~       152. Food S~     1
4 -1.26 Control N08002      9   252 female     28 Prima~       140. Food S~     3
5 -0.59 Control N06531      9   336 female     19 Secon~       151. Food S~     2
# ... with 4,690 more rows, and 17 more variables: Ncomp <dbl>, watmin <dbl>,
#   elec <dbl>, floor <dbl>, walls <dbl>, roof <dbl>, asset_wardrobe <dbl>,
#   asset_table <dbl>, asset_chair <dbl>, asset_khat <dbl>, asset_chouki <dbl>,
#   asset_tv <dbl>, asset_refrig <dbl>, asset_bike <dbl>, asset_moto <dbl>,
#   asset_sewmach <dbl>, asset_mobile <dbl>
```

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

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

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

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

A convenient summary of the relevant variables is given just above, complete with a small visualization describing the marginal characteristics of each covariate. Note that the *asset* variables reflect 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.

Chapter 3

Super (Machine) Learning

Ivana Malenica and *Rachael Phillips*

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

Updated: 2021-10-12

Learning Objectives

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

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

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

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

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

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

5. Obtain `sl3` variable importance metrics.

6. Interpret the fit for discrete and continuous Super Learners' from the cross-validated risk table and the coefficients.

7. Justify the base library of machine learning algorithms and the ensembling learner in terms of the prediction problem, statistical model $\mathcal{M}$, data sparsity, and the dimensionality of the covariates.

Motivation

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

Introduction

In Chapter 1, we introduced the *Roadmap for Targeted Learning* as a general template to translate real-world data applications into formal statistical estimation problems.

The first steps of this roadmap define the *statistical estimation problem*, which establish

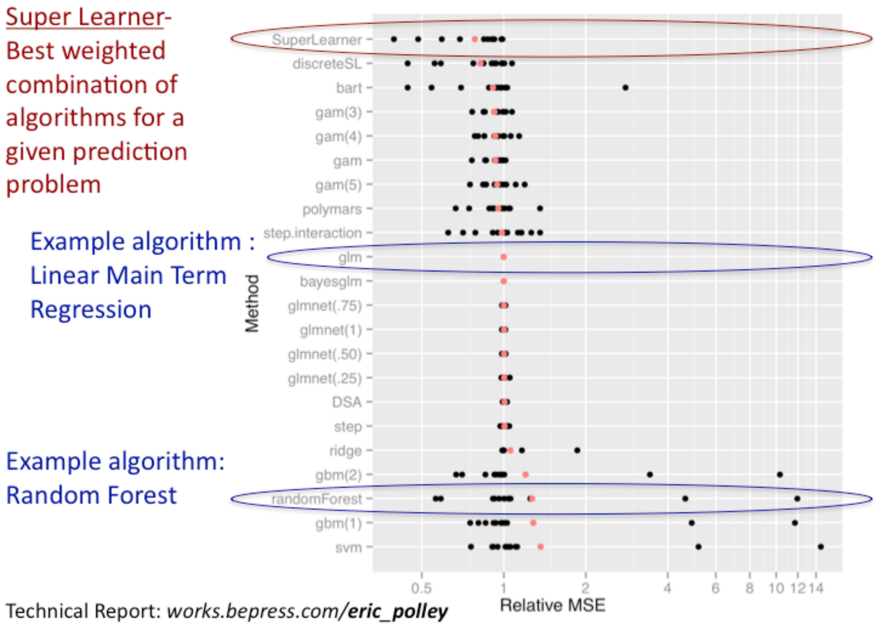
1. **The data O as a random variable, or equivalently, a realization of a particular experiment/study, which has probability distribution P_0.** This is written $O \sim P_0$, and P_0 is also commonly referred to as the data-generating process (DGP) and also the data-generating distribution (DGD). The data structure O is comprised of variables, such as a vector of covariates W, a treatment or exposure A, and an outcome Y, $O = (W, A, Y) \sim P_0$. We often observe the random variable O n times, by repeating the common experiment n times. For example, $O_1, \ldots, O_n$ random variables could be the result of a random sample of n subjects from a population, collecting baseline characteristics W, randomly assigning treatment A, and then later measuring an outcome Y.
2. **A statistical model $\mathcal{M}$ as a set of possible probability distributions that could have given rise to the data.** It's essential for $\mathcal{M}$ to only constrained by factual subject-matter knowledge in order to guarantee P_0 resides in the statistical model, written $P_0 \in \mathcal{M}$. Continuing the example from step 1, the following restrictions could be placed on the statistical model: the $O_1, \ldots, O_n$ observations in the data are independent and identically distributed (i.i.d.), the assignment of treatment A was random and not based on covariates W.
3. **A translation of the scientific question of interest into a function of P_0, the target statistical estimand $\Psi(P_0)$.** For example, we might be interested in the average difference in mean outcomes under treatment $A = 1$ versus placebo $A = 0$: $\Psi(P_0) = E_{P_0}\Big[E_{P_0}(Y|A = 1, W) - E_{P_0}(Y|A = 0, W)\Big]$. Note that, if the scientific question is causal, then it's translation will produce a target *causal* estimand; another layer of translation, identifiability, is required to express the target causal estimand as a function of the observed data distribution P_0. See causal target parameters for more information on causal quantities, causal models and identifiability. Note that if the target estimand is causal, step 3 also requires establishing so-called "identifiability" of this estimand from the observed data. See causal target parameters for more detail on causal models and identifiability.

After finalizing steps 1–3 above, the estimation procedure can be specified. We advocate for the use of the Super Learner (SL) algorithm in the estimation procedure

it is flexible and grounded in optimality theory (van der Laan et al., 2007).

Why use the Super Learner?

- It offers a system for combining/ensembling many machine learning (ML) algorithms into an improved ML algorithm.
- In large samples, SL is proven to perform at least as well as the unknown best candidate ML algorithm (van der Laan and Dudoit, 2003; Van der Vaart et al., 2006).
- When we have tested different ML algorithms on actual data and looked at the performance, never does one algorithm always win (see below).



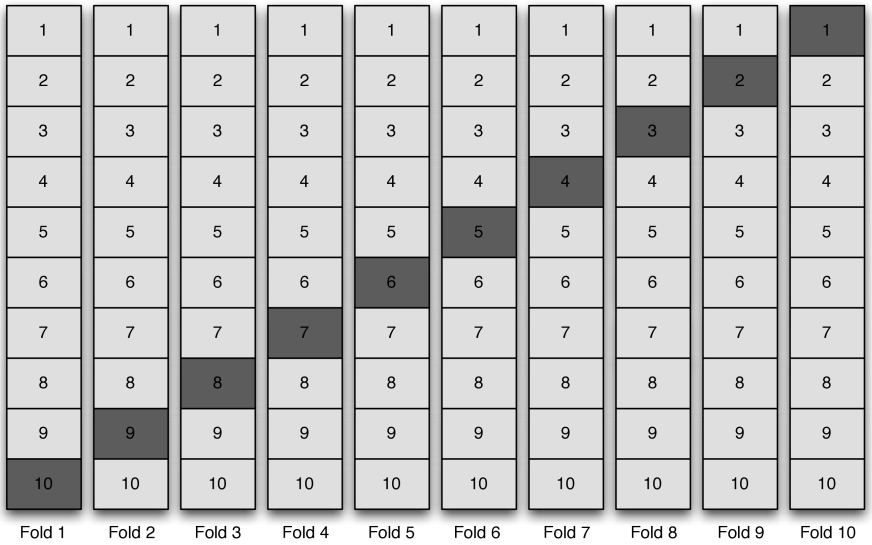
The figure above shows the performance of several different ML algorithms, including the SL, across many datasets. Here, the performance was assessed with the relative mean squared error and the target estimand that was being estimated for all datasets was the conditional mean outcome, given covariates (Polley and van der Laan, 2010).

3.0.1 General Overview of the Algorithm

- SL uses cross-validation and an objective function (e.g., loss function) to optimize the fit of the target parameter, based on a weighted combination of a so-called "library" of candidate ML algorithms.
- The library of ML algorithms consists of functions ("learners" in the SL nomenclature). These functions should align with the statistical model, both in terms of it's vastness and any potential constraints, where the statistical model's constraints are restrictions based on subject-matter knowledge regarding the process that generated the data.
- The so-called "metalearning", or ensembling of the library of ML algorithms has been shown to be adaptive and robust, even in small samples (Polley and van der Laan, 2010).

3.0.1.1 Cross-validation

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



The figure above shows an example of V-fold cross-validation with $V = 10$ folds, and this is the default cross-validation structure in the `sl3` `R` package. The darker

boxes represent the so-called "validation data" and the lighter boxes represent the so-called "training data". The following details are important to notice:

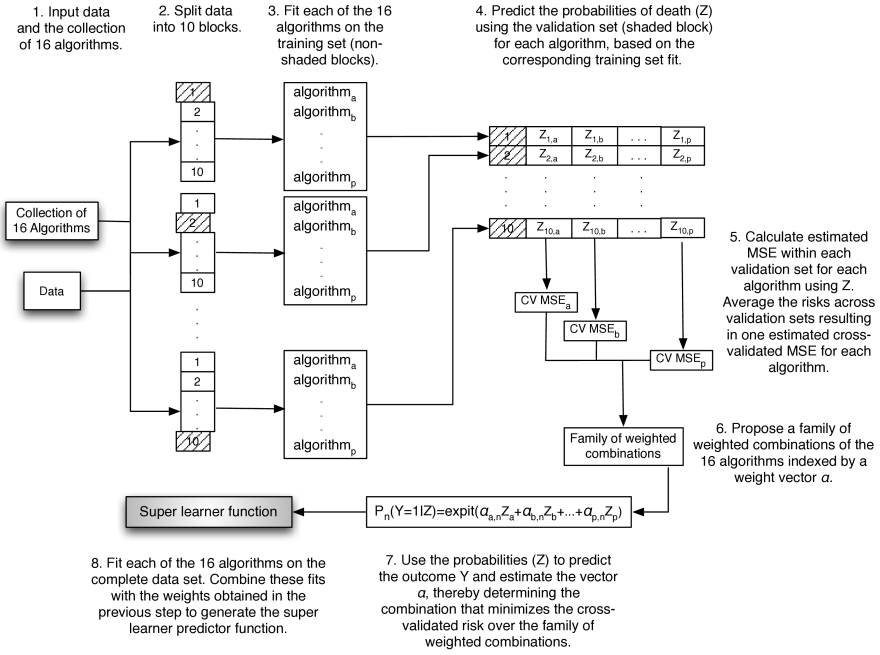
- Across all folds, there are V (10) copies of the dataset. The only difference between each copy is the coloring, which distinguishes the subset of the data that's considered as the training data from the subset that's considered as the validation data.
- Within each fold $1/V$ (1/10) of the data is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

3.0.1.2 Step-by-step procedure with V-fold cross-validation

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

which are the variables in the Z matrix, the K learners' predictions. Once the metalearner is fit, it can be used to obtain predicted outcomes from new input data; that is, new K learners predictions' can be supplied to the fitted metalearner in order to obtain predicted outcomes.

6. The fitted metalearner and the full-fit learners define the weighted combination of the K learners, finalizing the Super Learner (SL) fit. To obtain SL predictions the full-fit learners' predictions are first obtained and then fed as input to the fitted metalearner; the metalearner's output is the SL predictions.



3.0.1.3 How to pick the library of candidate ML algorithms?

- The library of candidate ML algorithms should be chosen based on contextual knowledge regarding the study/experiment that generated the data, and on the information available in the data.

- Having a "go-to" library to use as a "default" when the sample size is relatively large can be useful in practice.
- The algorithms may range from a simple linear regression model to multi-step algorithms involving screening covariates, penalizations, optimizing tuning

parameters, etc.

3.0.1.4 Theoretical Foundations

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

`sl3` "Microwave Dinner" Implementation

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

The `sl3` implementation consists of the following steps:

0. Load the necessary libraries and data
1. Define the machine learning task
2. Make an SL by creating library of base learners and a metalearner
3. Train the SL on the machine learning task
4. Obtain predicted values

WASH Benefits Study Example

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

0. Load the necessary libraries and data

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

```
library(data.table)
library(dplyr)
library(readr)
```

```
library(ggplot2)
library(SuperLearner)
library(origami)
library(sl3)
library(knitr)
library(kableExtra)

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

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

1. Define the machine learning task

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

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

Also, if we had missing outcomes, we would need to set `drop_missing_outcome = TRUE` when we create the task. In the next analysis, with the IST stroke trial data,

we do have a missing outcome. In the following chapter, we need to estimate this missingness mechanism; which is the conditional probably that the outcome is observed, given the history (i.e., variables that were measured before the missingness). Estimating the missingness mechanism requires learning a prediction function that outputs the predicted probability that a unit is missing, given their history.

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

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

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

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

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

Let's visualize our `washb_task`:

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

```
$outcome
[1] "whz"

$id
NULL

$weights
NULL

$offset
NULL

$time
NULL
```

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

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

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

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

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

2. Make a Super Learner

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

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

- A metalearner, to ensemble the base learners.

We might also incorporate

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

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

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

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

```
sl3_list_learners("continuous")
 [1] "Lrnr_arima"                      "Lrnr_bartMachine"
 [3] "Lrnr_bayesglm"                   "Lrnr_bilstm"
 [5] "Lrnr_bound"                      "Lrnr_caret"
 [7] "Lrnr_cv_selector"                "Lrnr_dbarts"
 [9] "Lrnr_earth"                      "Lrnr_expSmooth"
[11] "Lrnr_gam"                        "Lrnr_gbm"
[13] "Lrnr_glm"                        "Lrnr_glm_fast"
[15] "Lrnr_glmnet"                     "Lrnr_grf"
[17] "Lrnr_gru_keras"                  "Lrnr_gts"
[19] "Lrnr_h2o_glm"                    "Lrnr_h2o_grid"
[21] "Lrnr_hal9001"                    "Lrnr_HarmonicReg"
[23] "Lrnr_hts"                        "Lrnr_lightgbm"
[25] "Lrnr_lstm_keras"                 "Lrnr_mean"
[27] "Lrnr_multiple_ts"                "Lrnr_nnet"
[29] "Lrnr_nnls"                       "Lrnr_optim"
[31] "Lrnr_pkg_SuperLearner"           "Lrnr_pkg_SuperLearner_method"
[33] "Lrnr_pkg_SuperLearner_screener" "Lrnr_polspline"
[35] "Lrnr_randomForest"               "Lrnr_ranger"
[37] "Lrnr_rpart"                      "Lrnr_rugarch"
[39] "Lrnr_screener_correlation"       "Lrnr_solnp"
[41] "Lrnr_stratified"                 "Lrnr_svm"
[43] "Lrnr_tsDyn"                      "Lrnr_xgboost"
```

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

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

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

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

lrn_polspline <- Lrnr_polspline$new()

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

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

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

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

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

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

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

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

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

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

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

grid_params <- list(
  max_depth = c(2, 4, 6),
  eta = c(0.001, 0.1, 0.3),
  nrounds = 100
)
grid <- expand.grid(grid_params, KEEP.OUT.ATTRS = FALSE)
grid
  max_depth   eta nrounds
1         2 0.001     100
2         4 0.001     100
3         6 0.001     100
4         2 0.100     100
5         4 0.100     100
6         6 0.100     100
7         2 0.300     100
8         4 0.300     100
9         6 0.300     100

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

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

[[3]]
```

```
[1] "Lrnr_xgboost_100_1_6_0.001"

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

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

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

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

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

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

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

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

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

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

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

```
[4] "Lrnr_glmnet_NULL_deviance_10_0_100_TRUE_FALSE"
[5] "Lrnr_glmnet_NULL_deviance_10_1_100_TRUE_FALSE"
[6] "Lrnr_xgboost_50_1"
```

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

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

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

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

Screening Algorithms for Feature Selection

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

- `Lrnr_screener_importance` selects `num_screen` (default = 5) covariates based on the variable importance ranking provided by the `learner`. Any learner with an importance method can be used in `Lrnr_screener_importance`; and this currently includes `Lrnr_ranger`, `Lrnr_randomForest`, and `Lrnr_xgboost`.
- `Lrnr_screener_coefs`, which provides screening of covariates based on the magnitude of their estimated coefficients in a (possibly regularized) GLM.

The `threshold` (default = 1e-3) defines the minimum absolute size of the coefficients, and thus covariates, to be kept. Also, a `max_retain` argument can be optionally provided to restrict the number of selected covariates to be no more than `max_retain`.

- `Lrnr_screener_correlation` provides covariate screening procedures by running a test of correlation (Pearson default), and then selecting the (1) top ranked variables (default), or (2) the variables with a pvalue lower than some pre-specified threshold.
- `Lrnr_screener_augment` augments a set of screened covariates with additional covariates that should be included by default, even if the screener did not select them. An example of how to use this screener is included below.

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

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

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

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

# which covariates are selected on the full data?
screen_rf$train(washb_task)
[1] "Lrnr_screener_importance_5"
$selected
[1] "aged"         "month"        "momedu"       "Nlt18"        "asset_refrig"
```

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

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

```
screen_augment_rf
[1] "Lrnr_screener_augment_c(\"aged\", \"momage\")"
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3. Train the Super Learner on the machine learning task

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

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

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

4. Obtain predicted values

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

```
# we did it! now we have SL predictions
sl_preds <- sl_fit$predict()
head(sl_preds)
[1] -0.61014 -0.76503 -0.69304 -0.65588 -0.65689 -0.64811
```

We can also obtain a summary of the results.

```
sl_fit$cv_risk(loss_fun = loss_squared_error)
                          learner coefficients   risk        se  fold_sd
 1:                           glm     0.055568 1.0202 0.023955 0.067500
 2:                    polspline     0.055554 1.0208 0.023577 0.067921
 3:                       enet.5     0.055561 1.0133 0.023618 0.065940
 4:                        ridge     0.055566 1.0154 0.023736 0.065991
 5:                        lasso     0.055561 1.0130 0.023605 0.065828
 6:                    xgboost50     0.055589 1.1136 0.025262 0.077580
 7:       randomforest_screen_glm     0.055549 1.0229 0.023733 0.067706
 8: randomforest_screen_polspline     0.055567 1.0178 0.023753 0.067091
 9:    randomforest_screen_enet.5     0.055549 1.0229 0.023729 0.067516
10:     randomforest_screen_ridge     0.055548 1.0232 0.023746 0.067528
11:     randomforest_screen_lasso     0.055549 1.0226 0.023722 0.067520
12: randomforest_screen_xgboost50     0.055548 1.1207 0.025542 0.088250
13:              lasso_screen_glm     0.055556 1.0165 0.023561 0.064937
14:        lasso_screen_polspline     0.055556 1.0184 0.023537 0.065581
15:           lasso_screen_enet.5     0.055556 1.0164 0.023561 0.064912
16:            lasso_screen_ridge     0.055555 1.0166 0.023567 0.064759
17:            lasso_screen_lasso     0.055556 1.0164 0.023560 0.064898
18:        lasso_screen_xgboost50     0.055511 1.1296 0.025767 0.084180
19:                   SuperLearner           NA 1.0122 0.023500 0.068013
    fold_min_risk fold_max_risk
 1:       0.89442        1.1200
 2:       0.89892        1.1255
 3:       0.88927        1.1082
 4:       0.88570        1.1104
 5:       0.89037        1.1090
 6:       0.96019        1.2337
 7:       0.89923        1.1081
 8:       0.90342        1.1098
 9:       0.89912        1.1080
10:       0.89882        1.1073
11:       0.89912        1.1076
12:       0.95622        1.2330
13:       0.90204        1.1156
14:       0.89742        1.1162
15:       0.90184        1.1154
16:       0.90132        1.1148
17:       0.90183        1.1154
18:       0.96251        1.2327
19:       0.88510        1.1087
```

Cross-validated Super Learner

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

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

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

```
washb_task_new <- make_sl3_Task(
  data = washb_data,
  covariates = covars,
  outcome = outcome,
  folds = origami::make_folds(washb_data, fold_fun = folds_vfold, V = 2)
)

CVsl <- CV_lrnr_sl(
  lrnr_sl = sl_fit, task = washb_task_new, loss_fun = loss_squared_error
)

if (is_latex_output()) {
   CVsl %>%
     kable(format = "latex")
} else if (is_html_output()) {
   CVsl %>%
     kable() %>%
     kable_styling(fixed_thead = TRUE) %>%
     scroll_box(width = "100%", height = "300px")
}
```

learner	coefficients	risk	se	fold_sd	fold_min_risk	fold_m
glm	0.05556	1.0448	0.02548	0.06295	1.00030	
polspline	0.05556	1.0206	0.02354	0.05104	0.98449	
enet.5	0.05557	1.0191	0.02366	0.05005	0.98372	
ridge	0.05557	1.0264	0.02404	0.05195	0.98963	
lasso	0.05556	1.0201	0.02370	0.05134	0.98375	
xgboost50	0.05555	1.1871	0.02737	0.03028	1.16566	
randomforest_screen_glm	0.05557	1.0171	0.02370	0.05001	0.98176	
randomforest_screen_polspline	0.05556	1.0231	0.02376	0.05456	0.98449	
randomforest_screen_enet.5	0.05557	1.0171	0.02370	0.04994	0.98183	
randomforest_screen_ridge	0.05557	1.0171	0.02372	0.05005	0.98173	
randomforest_screen_lasso	0.05557	1.0172	0.02370	0.04992	0.98185	
randomforest_screen_xgboost50	0.05549	1.1967	0.02737	0.04170	1.16725	
lasso_screen_glm	0.05556	1.0197	0.02372	0.04487	0.98793	
lasso_screen_polspline	0.05556	1.0229	0.02374	0.04655	0.98997	
lasso_screen_enet.5	0.05556	1.0195	0.02372	0.04475	0.98790	
lasso_screen_ridge	0.05556	1.0198	0.02374	0.04513	0.98785	
lasso_screen_lasso	0.05556	1.0195	0.02372	0.04474	0.98791	
lasso_screen_xgboost50	0.05549	1.1884	0.02720	0.08492	1.12834	
SuperLearner	NA	1.0179	0.02379	0.04688	0.98479	

Variable Importance Measures with `sl3`

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

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

The intuition of this measure is that it calculates the risk (in terms of the average loss in predictive accuracy) of losing one covariate, while keeping everything else fixed, and compares it to the risk if the covariate was not lost. If this risk ratio is one, or risk difference is zero, then losing that covariate had no impact, and is thus not important

by this measure. We do this across all of the covariates. As stated above, we can remove the covariate and refit the SL without it, or we just permute the covariate (faster) and hope for the shuffling to distort any meaningful information that was present in the covariate. This idea of permuting instead of removing saves a lot of time, and is also incorporated in the `randomForest` variable importance measures. However, the permutation approach is risky, so the importance function default is to remove and refit.

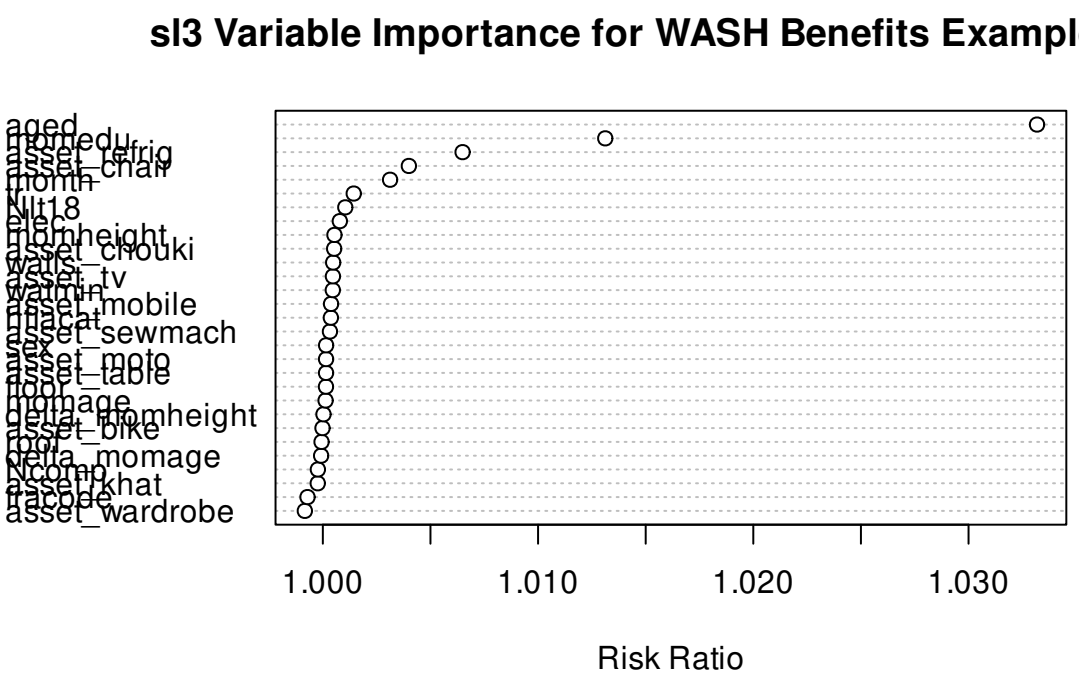
Let's explore the `sl3` variable importance measurements for the `washb` data.

```
washb_varimp <- importance(sl_fit, loss = loss_squared_error, type = "permute")

if (is_latex_output()) {
  washb_varimp %>%
    kable(format = "latex")
} else if (is_html_output()) {
  washb_varimp %>%
    kable() %>%
    kable_styling(fixed_thead = TRUE) %>%
    scroll_box(width = "100%", height = "300px")
}
```

X	risk_ratio
aged	1.03318
momedu	1.01312
asset_refrig	1.00649
asset_chair	1.00400
month	1.00312
tr	1.00143
Nlt18	1.00104
elec	1.00079
momheight	1.00054
asset_chouki	1.00053
walls	1.00048
asset_tv	1.00047
watmin	1.00046
asset_mobile	1.00038
hfiacat	1.00037
asset_sewmach	1.00033
sex	1.00015
asset_moto	1.00015
asset_table	1.00015
floor	1.00015
momage	1.00013
delta_momheight	1.00003
asset_bike	0.99998
roof	0.99994
delta_momage	0.99992
Ncomp	0.99977
asset_khat	0.99976
fracode	0.99929
asset_wardrobe	0.99916

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



3.1 Exercises

3.1.1 Predicting Myocardial Infarction with `sl3`

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

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

# take a quick peek
if (is_latex_output()) {
   head(chspred) %>%
     kable(format = "latex")
```

```
} else if (is_html_output()) {
   head(chspred) %>%
     kable() %>%
     kable_styling(fixed_thead = TRUE) %>%
     scroll_box(width = "100%", height = "300px")
}
```

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

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. Customize tuning parameters for one of your learners. Feel free to use learners from `sl3` or `SuperLearner`. You may use the same base learning library that is presented above.
3. Incorporate at least one pipeline with feature selection. Any screener and learner(s) can be used.
4. With the default metalearner and base learners, make the Super Learner (SL) and train it on the task.
5. Print your SL fit by calling `print()` with `$`.
6. Cross-validate your SL fit to see how well it performs on unseen data. Specify a valid loss function to evaluate the SL.
7. Use the `importance()` function to identify the "most important" predictor of myocardial infarction, according to `sl3` importance metrics.

3.2 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: $\Psi_0 = \mathbb{E}_{0,W}[\mathbb{E}_0(Y \mid A = 1, W) - \mathbb{E}_0(Y \mid A = 0, W)]$.
  - 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 $A = 1$, $Y_i \mid A_i = 1, W_i$ for all $i = 1, \ldots, n$. The other vector would represent the predicted outcomes under an intervention that sets all subjects to receive $A = 0$, $Y_i \mid A_i = 0, W_i$ 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 optimized for our target estimand of interest. Here, we cared about doing a good job estimating the ATE. The SL is an essential step to help us get there. In fact, we will use the counterfactual predicted outcomes that were explained at length above. However, SL might not be not the end of the estimation procedure. Plugging in the Super Learner in the target parameter representation would generally not result in an asymptotically linear estimator of the target estimand. This begs the question, why is it important for an estimator to possess these properties?

  - An asymptotically linear estimator converges to the estimand at $\frac{1}{\sqrt{n}}$ rate – and thus behaves as sample mean – so it's sampling distribution can be estimated in order to conduct formal statistical inference (i.e., confidence intervals and p-values).
  - Substitution, or plug-in, estimators of the estimand are desirable because they respect both the local and global constraints of the statistical model (e.g., bounds), and have they have better finite-sample properties.
  - 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.
    * 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. Various canonical gradient are shown in the chapters that follow.
    * Practitioner's do not need to know how to calculate a canonical gradient in order to understand efficiency and use Targeted Maximum Likelihood Estimation (TMLE). Metaphorically, you do not need to be Yoda in order to be a Jedi.

- TMLE is a general strategy that succeeds in constructing efficient and asymptotically linear plug-in estimators.
- SL is fantastic for pure prediction, and for obtaining an initial estimate in the first step of TMLE, but we need the second step of TMLE to have the desirable statistical properties mentioned above.
- In the chapters that follow, we focus on the Targeted Maximum Likelihood Estimator and it's generalization to Targeted Minimum Loss-based Estimator, both referred to as TMLE.

Chapter 4

The TMLE Framework

Jeremy Coyle and *Nima Hejazi*

Based on the `tmle3` `R` package.

4.1 Learning Objectives

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

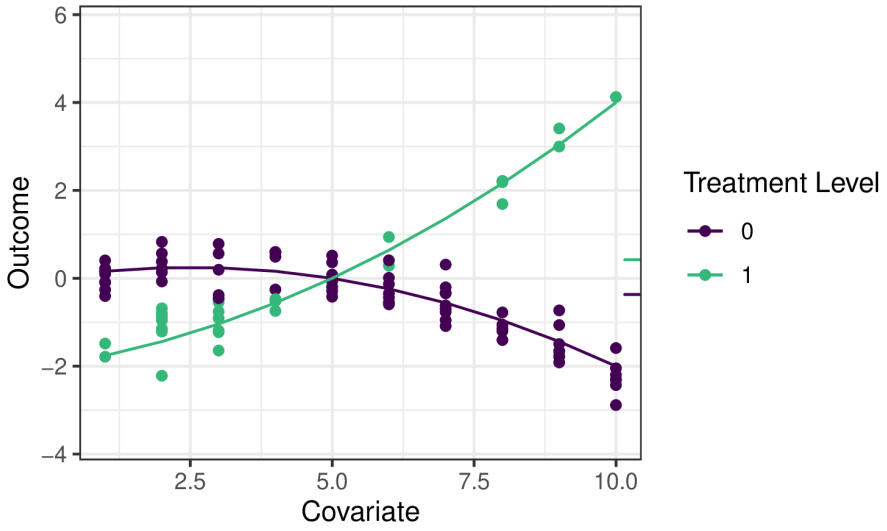
1. Understand why we use TMLE for effect estimation.
2. Use `tmle3` to estimate an Average Treatment Effect (ATE).
3. Understand how to use `tmle3` "Specs" objects.
4. Fit `tmle3` for a custom set of target parameters.
5. Use the delta method to estimate transformations of target parameters.

4.2 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.

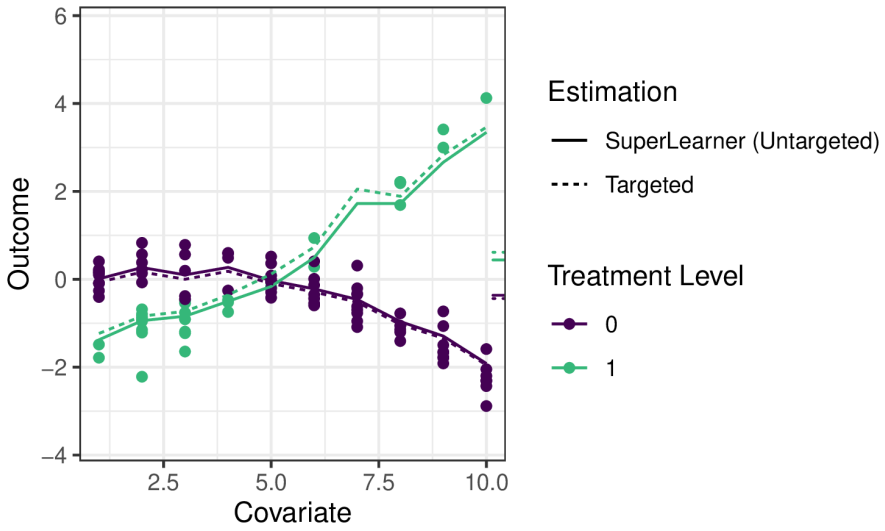
While we hope to motivate the application of TMLE in this chapter, we refer the interested reader to the two Targeted Learning books and associated works for full technical details.

4.3 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.

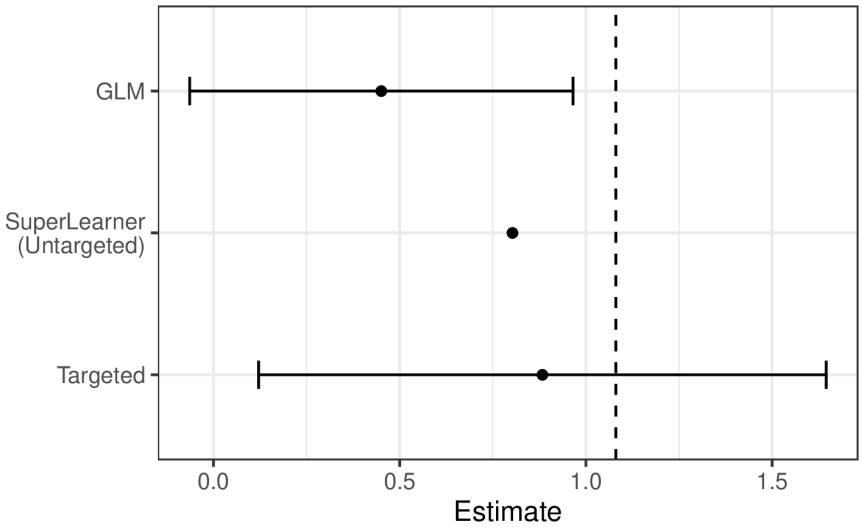
Applying `sl3` to estimate the outcome regression in our example, we can see that the ensemble machine learning predictions fit the data quite well:



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 $\bar{Q}_0(A, W)$ 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.

4.4 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.

4.4.1 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., $\text{logit}(x) = \log\left(\frac{x}{1-x}\right)$), 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).

4.4.2 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 $\overline{Q}_n^\star$ 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.

4.5 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.

4.5.1 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
)
```

4.5.2 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"
)
```

4.5.3 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
```

4.5.4 Create a "Spec" Object

`tmle3` is general, and allows most components of the TMLE procedure to be specified in a modular way. However, most end-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 by an end-user.

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"
)
```

4.5.5 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.

4.5.6 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}] -0.0031611 0.010044 0.050853
       lower   upper psi_transformed lower_transformed upper_transformed
1: -0.089626 0.10971        0.010044          -0.089626            0.10971
```

4.5.7 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] 0.010044
```

4.6 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.

4.6.1 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, w
    Parents:

$A
tmle3_Node: A
    Variables: tr
    Parents: W

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

4.6.2 Initial Likelihood

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

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

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

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

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

```
initial_likelihood$get_likelihoods(tmle_task)
                W       A        Y
   1: 0.00021299 0.34702 -0.32696
   2: 0.00021299 0.37305 -0.88218
   3: 0.00021299 0.34685 -0.79300
   4: 0.00021299 0.33625 -0.89157
   5: 0.00021299 0.34098 -0.63477
  ---
4691: 0.00021299 0.24334 -0.61095
4692: 0.00021299 0.24620 -0.21534
4693: 0.00021299 0.22401 -0.79223
4694: 0.00021299 0.27641 -0.94319
4695: 0.00021299 0.20158 -1.08201
```

4.6.3 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)
  )
```

4.6.4 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}]
```

4.6.5 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}] -0.0062324 0.017515 0.050591
       lower   upper psi_transformed lower_transformed upper_transformed
1: -0.081641 0.11667        0.017515          -0.081641            0.11667
```

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

4.7 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.

4.7.1 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

4.7.2 Fit

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

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

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

print(tmle_fit_multiparam)
A tmle3_Fit that took 1 step(s)
   type                                          param   init_est tmle_est
1:  TSM                              E[Y_{A=Control}] -0.5953314 -0.61981
2:  TSM                          E[Y_{A=Handwashing}] -0.6179897 -0.66114
3:  TSM                            E[Y_{A=Nutrition}] -0.6119870 -0.60338
4:  TSM                     E[Y_{A=Nutrition + WSH}] -0.6015639 -0.60250
5:  TSM                           E[Y_{A=Sanitation}] -0.5866311 -0.58147
6:  TSM                                 E[Y_{A=WSH}] -0.5213051 -0.45027
7:  TSM                               E[Y_{A=Water}] -0.5653576 -0.53554
8:  ATE E[Y_{A=Nutrition + WSH}] - E[Y_{A=Control}] -0.0062324  0.01731
         se     lower    upper psi_transformed lower_transformed
1: 0.030069 -0.678746 -0.56088        -0.61981         -0.678746
2: 0.041821 -0.743111 -0.57917        -0.66114         -0.743111
3: 0.041553 -0.684825 -0.52194        -0.60338         -0.684825
4: 0.040925 -0.682712 -0.52229        -0.60250         -0.682712
5: 0.042313 -0.664402 -0.49854        -0.58147         -0.664402
6: 0.045216 -0.538891 -0.36165        -0.45027         -0.538891
7: 0.039290 -0.612551 -0.45854        -0.53554         -0.612551
8: 0.050596 -0.081857  0.11648         0.01731         -0.081857
   upper_transformed
1:          -0.56088
2:          -0.57917
3:          -0.52194
4:          -0.52229
5:          -0.49854
6:          -0.36165
7:          -0.45854
8:           0.11648
```

4.8 Exercises

4.8.1 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.
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 $\overline{Q}_0 = \mathbb{E}_0(Y \mid A, Y)$ 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.

4.8.2 Estimation of Strata-Specific ATEs with `tmle3`

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

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

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

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

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

4.9 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.

Chapter 5

Optimal Individualized Treatment Regimes (optional)

Ivana Malenica

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

Updated: 2021-10-13

5.1 Learning Objectives

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

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. Implement optimal individualized treatment rules based on sub-optimal rules, or "simple" rules, and recognize the practical benefit of these rules.

6. 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.
7. Measure variable importance as defined in terms of the optimal individualized treatment interventions.

5.2 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. One opts to administer the intervention to individuals who will benefit from it, instead of assigning treatment on a population level.

- This aim motivates a different type of an intervention, as opposed to the static exposures we might be used to.
- In this chapter, we learn about dynamic (individualized) interventions that tailor the treatment decision based on the collected covariates.
- In the statistics community, such a treatment strategy is termed **individualized treatment regimes** (ITR), and the (counterfactual) population mean outcome under an ITR is the **value of the ITR**.
- 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. An ITR with the maximal value is referred to as an **optimal ITR** or the **optimal individualized treatment**. Consequently, the value of an optimal ITR is termed the **optimal value**, or the **mean under the optimal individualized treatment**.
- 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?
- For example, one might seek 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 an intervention.

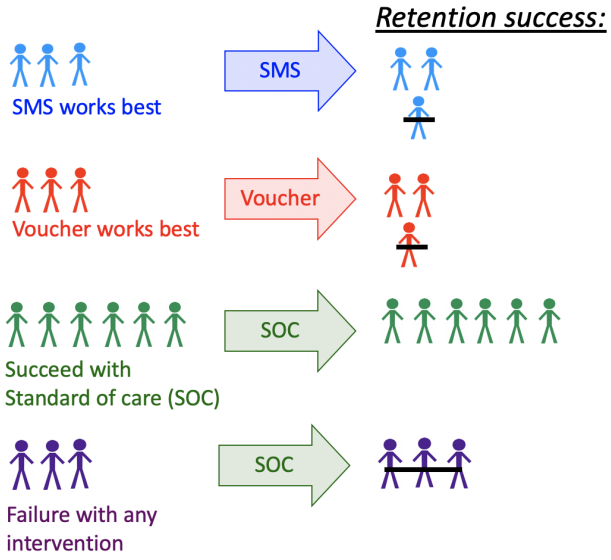


Figure 5.1: Illustration of a Dynamic Treatment Regime in a Clinical Setting

This aim motivates a different type of intervention, as opposed to the static exposures we might be used to.

- In this chapter, we examine multiple examples of optimal individualized treatment regimes and estimate the mean outcome under the ITR where the candidate rules are restricted to depend only on user-supplied subset of the baseline covariates.
- In order to accomplish this, we present the `tmle3mopttx` R package, which features an implementation of a recently developed algorithm for computing targeted minimum loss-based estimates of a causal effect based on optimal ITR for categorical treatment.
- In particular, we will use `tmle3mopttx` to estimate optimal ITR and the corresponding population value, construct realistic optimal ITRs, and perform variable importance in terms of the mean under the optimal individualized treatment.

5.3 Data Structure and Notation

- Suppose we observe n independent and identically distributed observations of the form $O = (W, A, Y) \sim P_0$. $P_0 \in \mathcal{M}$, where $\mathcal{M}$ is the fully nonparametric model.
- Denote $A \in \mathcal{A}$ as categorical treatment, where $\mathcal{A} \equiv \{a_1, \ldots, a_{n_A}\}$ and $n_A = |\mathcal{A}|$, with n_A denoting the number of categories.
- Denote Y as the final outcome, and W a vector-valued collection of baseline covariates.
- The likelihood of the data admits a factorization, implied by the time ordering of O.
$$p_0(O) = p_{Y,0}(Y|A,W)p_{A,0}(A|W)p_{W,0}(W) = q_{Y,0}(Y|A,W)q_{A,0}(A|W)q_{W,0}(W),$$
- Consequently, we define $P_{Y,0}(Y|A,W) = Q_{Y,0}(Y|A,W)$, $P_{A,0}(A|W) = g_0(A|W)$ and $P_{W,0}(W) = Q_{W,0}(W)$ as the corresponding conditional distributions of Y, A and W.
- We also define $\bar{Q}_{Y,0}(A,W) \equiv E_0[Y|A,W]$.
- Finally, denote V as a subset of the baseline covariates W that the optimal individualized rule depends on.

5.4 Defining the Causal Effect of an Optimal Individualized Intervention

- Consider dynamic treatment rules $V \rightarrow d(V) \in \{a_1, \ldots, a_{n_A}\} \times \{1\}$, for assigning treatment A based on V.
- Dynamic treatment regime may be viewed as an intervention in which A is set equal to a value based on a hypothetical regime $d(V)$, and $Y_{d(V)}$ is the corresponding counterfactual outcome under $d(V)$.
- The goal of any causal analysis motivated by an optimal individualized intervention is to estimate a parameter defined as the counterfactual mean of the outcome with respect to the modified intervention distribution.

- Recall causal assumptions:

1. **Consistency**: $Y_i^{d(v_i)} = Y_i$ in the event $A_i = d(v_i)$, for $i = 1, \ldots, n$.
2. **Stable unit value treatment assumption (SUTVA)**: $Y_i^{d(v_i)}$ does not depend on $d(v_j)$ for $i = 1, \ldots, n$ and $j \neq i$, or lack of interference.
3. **Strong ignorability**: $A \perp\!\!\!\perp Y^{d(v)} \mid W$, for all $a \in \mathcal{A}$.
4. **Positivity (or overlap)**: $P_0(\min_{a \in \mathcal{A}} g_0(a|W) > 0) = 1$

- Here, we also assume non-exceptional law is in effect.

- We are primarily interested in the value of an individualized rule,
$$E_0[Y_{d(V)}] = E_{0,W}[Q_{Y,0}(A = d(V), W)].$$

- The optimal rule is the rule with the maximal value:
$$d_{opt}(V) \equiv \text{argmax}_{d(V) \in \mathcal{D}} E_0[Y_{d(V)}]$$

  where $\mathcal{D}$ represents the set of possible rules, d, implied by V.

- The target causal estimand of our analysis is:
$$\psi_0 := E_0[Y_{d_{opt}(V)}] = E_{0,W}[Q_{Y,0}(A = d_{opt}(V), W)].$$

- General, high-level idea:

1. Learn the optimal ITR using the Super Learner.

2. Estimate its value with the cross-validated Targeted Minimum Loss-based Estimator (CV-TMLE).

5.4.1 Why CV-TMLE?

- 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!

5.5 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.
- Optimal ITR ideally assigns treatment to individuals falling in strata in which the stratum specific average treatment effect, the **blip** function, is positive and does not assign treatment to individuals for which this quantity is negative.
- We define the blip function as:
$$\bar{Q}_0(V) \equiv E_0[Y_1 - Y_0|V] \equiv E_0[\bar{Q}_{Y,0}(1, W) - \bar{Q}_{Y,0}(0, W)|V],$$
or the average treatment effect within a stratum of V.
- Optimal individualized rule can now be derived as $d_{opt}(V) = I(\bar{Q}_0(V) > 0)$.
- 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 $g_0(A|W)$ using `sl3`. We denote such estimates as $\bar{Q}_{Y,n}(A, W)$ and $g_n(A|W)$.
2. Apply the doubly robust Augmented-Inverse Probability Weighted (A-IPW) transform to our outcome, where we define:

$$D_{\bar{Q}_Y,g,a}(O) \equiv \frac{I(A = a)}{g(A|W)}(Y - \bar{Q}_Y(A, W)) + \bar{Q}_Y(A = a, W)$$

Note that under the randomization and positivity assumptions we have that $E[D_{\bar{Q}_Y,g,a}(O)|V] = E[Y_a|V]$. We emphasize the double robust nature of the A-IPW transform: consistency of $E[Y_a|V]$ will depend on correct estimation of either $\bar{Q}_{Y,0}(A, W)$ or $g_0(A|W)$. As such, in a randomized trial, we are guaranteed a consistent estimate of $E[Y_a|V]$ even if we get $\bar{Q}_{Y,0}(A, W)$ wrong!

Using this transform, 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.

3. Our estimated rule is $d(V) = \text{argmax}_{a \in \mathcal{A}} \bar{Q}_{0,a}(V)$.

4. We obtain inference for the mean outcome under the estimated optimal rule using CV-TMLE.

5.5.1 Evaluating the Causal Effect of an optimal ITR with Binary Treatment

To start, let us load the packages we will use and set a seed for simulation:

```
library(here)
library(data.table)
library(sl3)
library(tmle3)
library(tmle3mopttx)
library(devtools)
set.seed(116)
```

5.5.1.1 Simulate Data

Our data generating distribution is of the following form:

$$W \sim \mathcal{N}(\mathbf{0}, \mathbf{I_{3\times 3}})$$
$$P(A = 1|W) = \frac{1}{1 + \exp^{(-0.8*W_1)}}$$
$$P(Y = 1|A, W) = 0.5\text{logit}^{-1}[-5I(A = 1)(W_1 - 0.5) + 5I(A = 0)(W_1 - 0.5)] + 0.5\text{logit}^{-1}(W_2 W_3)$$

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

- The above composes our observed data structure $O = (W, A, Y)$.
- Note that the mean under the true optimal rule is $\psi_0 = 0.578$ for this data generating distribution.
- Next, we specify the role that each variable in the data set plays as the nodes in a DAG.

```
# 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 data set plays as the nodes in a DAG.

5.5.1.2 Constructing Optimal Stacked Regressions with `sl3`

- We generate three different ensemble learners that must be fit, corresponding to the learners for the outcome regression, propensity score, and the blip function.

```
# 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()

## Define the Q learner:
Q_learner <- Lrnr_sl$new(
  learners = list(
    lrn_xgboost_50, lrn_xgboost_100,
    lrn_xgboost_500, lrn_mean, lrn_glm
  ),
  metalearner = Lrnr_nnls$new()
)

## Define the g learner:
g_learner <- Lrnr_sl$new(
  learners = list(lrn_xgboost_100, lrn_glm),
  metalearner = Lrnr_nnls$new()
)

## Define the B learner:
b_learner <- Lrnr_sl$new(
  learners = list(
    lrn_xgboost_50, lrn_xgboost_100,
    lrn_xgboost_500, lrn_mean, lrn_glm
```

```
  ),
  metalearner = Lrnr_nnls$new()
)
```

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)
```

5.5.1.3 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.
- We also need to specify the type of blip we will use in this estimation problem, and the list of learners used to estimate relevant parts of the likelihood and the blip function.
- In addition, we need to specify whether we want to maximize or minimize the mean outcome under the rule (`maximize=TRUE`).
- If `complex=FALSE`, `tmle3mopttx` will consider all the possible rules under a smaller set of covariates including the static rules, and optimize the mean outcome over all the suboptimal rules dependent on V.
- If `realistic=TRUE`, only treatments supported by the data will be considered, therefore alleviating concerns regarding practical positivity issues.

```
# 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
)
```

```
# 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
1:  TSM E[Y_{A=NULL}]  0.43164  0.55855 0.027047 0.50554 0.61156
   psi_transformed lower_transformed upper_transformed
1:         0.55855           0.50554           0.61156
```

We can see that the confidence interval covers our true mean under the true optimal individualized treatment!

5.6 Categorical Treatment

QUESTION: Can we still use the blip function if the treatment is categorical?

- In this section, we consider how to evaluate the mean outcome under the optimal individualized treatment when A has more than two categories!
- We define **pseudo-blips** as vector valued entities where the output for a given V is a vector of length equal to the number of treatment categories, n_A. As such, we define it as:
$$\bar{Q}_0^{pblip}(V) = \{\bar{Q}_{0,a}^{pblip}(V) : a \in \mathcal{A}\}$$
- We implement three different pseudo-blips in `tmle3mopttx`.

1. **Blip1** corresponds to choosing a reference category of treatment, and defining the blip for all other categories relative to the specified reference:
$$\bar{Q}_{0,a}^{pblip-ref}(V) \equiv E_0(Y_a - Y_0|V)$$
2. **Blip2** approach corresponds to defining the blip relative to the average of all categories:
$$\bar{Q}_{0,a}^{pblip-avg}(V) \equiv E_0(Y_a - \frac{1}{n_A}\sum_{a\in\mathcal{A}} Y_a|V)$$
3. **Blip3** reflects an extension of Blip2, where the average is now a weighted average:
$$\bar{Q}_{0,a}^{pblip-wavg}(V) \equiv E_0(Y_a - \frac{1}{n_A}\sum_{a\in\mathcal{A}} Y_a P(A=a|V)|V)$$

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

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.

5.6.1.1 Simulated Data

- First, we load the simulated data. Here, our data generating distribution was of the following form:

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

$$P(A|W) = \frac{1}{1+\exp^{(-(0.05*I(A=1)*W_1+0.8*I(A=2)*W_1+0.8*I(A=3)*W_1))}}$$

$$P(Y|A,W) = 0.5\text{logit}^{-1}[15I(A=1)(W_1-0.5)-3I(A=2)(2W_1+0.5)+3I(A=3)(3W_1-0.5)]+\text{logit}^{-1}(W_2W$$

- 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 mean under the true optimal rule is $\psi_0 = 0.658$, 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"
)
```

5.6.1.2 Constructing Optimal Stacked Regressions with `sl3`

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

```
# 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$new(
  learners = list(lrn_xgboost_50, lrn_xgboost_100, lrn_xgboost_500, lrn_mean, lr
  metalearner = Lrnr_nnls$new()
)

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

# Define the Blip learner, which is a multivariate learner:
learners <- list(lrn_xgboost_50, lrn_xgboost_100, lrn_xgboost_500, lrn_mean, lrn
b_learner <- create_mv_learners(learners = learners)
```

- 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 $g_0(A|W)$ 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 $g_0(A|W)$ in `sl3`, we run the following:

```
# See which learners support multi-class classification:
sl3_list_learners(c("categorical"))
 [1] "Lrnr_bound"                "Lrnr_caret"
 [3] "Lrnr_cv_selector"          "Lrnr_glmnet"
```

```
 [5] "Lrnr_grf"                    "Lrnr_gru_keras"
 [7] "Lrnr_h2o_glm"                "Lrnr_h2o_grid"
 [9] "Lrnr_independent_binomial"   "Lrnr_lightgbm"
[11] "Lrnr_lstm_keras"             "Lrnr_mean"
[13] "Lrnr_multivariate"           "Lrnr_nnet"
[15] "Lrnr_optim"                  "Lrnr_polspline"
[17] "Lrnr_pooled_hazards"         "Lrnr_randomForest"
[19] "Lrnr_ranger"                 "Lrnr_rpart"
[21] "Lrnr_screener_correlation"   "Lrnr_solnp"
[23] "Lrnr_svm"                    "Lrnr_xgboost"
```

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

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

```
# initialize a tmle specification
tmle_spec <- 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 <- 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}]  0.54926  0.62506 0.06366 0.50029 0.74983         0.62506
   lower_transformed upper_transformed
1:           0.50029           0.74983

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

  1   2   3
447 382 171
```

We can see that the confidence interval covers our true mean under the true optimal individualized treatment.

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

5.7 Extensions to Causal Effect of an OIT

- We consider two extensions to the procedure described for estimating the value of the ITR.
- The first one considers a setting where the user might be interested in a grid of possible suboptimal rules, corresponding to potentially limited knowledge of potential effect modifiers (**Simpler Rules**).
- The second extension concerns implementation of realistic optimal individual interventions where certain regimes might be preferred, but due to practical or global positivity restraints are not realistic to implement (**Realistic Interventions**).

5.7.1 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 $\text{card}(S) \leq \text{card}(V)$ and $\emptyset \in S$.
- This allows us to consider sub-optimal rules that are easier to estimate and potentially provide more realistic rules- as such, we allow for statistical inference for the counterfactual mean outcome under the sub-optimal rule.

```
# initialize a tmle specification
tmle_spec <- 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 <- 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
1:  TSM E[Y_{d(V=W2,W1)}]  0.53979  0.61659 0.068671 0.482 0.75118
   psi_transformed lower_transformed upper_transformed
1:         0.61659             0.482           0.75118
```

Even though the user specified all baseline covariates as the basis for rule estimation, a simpler rule is sufficient to maximize the mean under the optimal individualized treatment!

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

5.7.2 Realistic Optimal Individual Regimes

- `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).
- Specifying `realistic="TRUE"`, we consider possibly suboptimal treatments that optimize the outcome in question while being supported by the data.

```
# initialize a tmle specification
tmle_spec <- 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 <- 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
1:  TSM E[Y_{A=NULL}]  0.54688  0.66017 0.021458 0.61812 0.70223
   psi_transformed lower_transformed upper_transformed
1:         0.66017           0.61812           0.70223

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

  2   3
516 484
```

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

5.7.3 Variable Importance Analysis

- 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 observed covariates!
- In order to run `tmle3mopttx` variable importance measure, we need considered covariates to be categorical variables.
- For illustration purpose, we bin baseline covariates corresponding to the data-generating distribution described in the previous section:

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

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

- Note that our node list now includes W_1 as treatments as well! Don't worry, we will still properly adjust for all baseline covariates when considering A as treatment.

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

- 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`.

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

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

print(vim_results)
   type                   param    init_est tmle_est       se     lower     upper
1:   RR RR(E[Y_{A=NULL}]/E[Y])  0.0011065  0.14769 0.032279  0.084421 0.210952
2:   RR RR(E[Y_{A=NULL}]/E[Y]) -0.0227483 -0.01578 0.047608 -0.109090 0.077529
   psi_transformed lower_transformed upper_transformed  A           W   Z_stat
1:         1.15915           1.08809            1.2349  A W3,W4,W2,W1  4.57534
2:         0.98434           0.89665            1.0806 W1  W3,W4,W2,A -0.33146
         p_nz p_nz_corrected
1: 2.3772e-06     4.7544e-06
2: 3.7015e-01     3.7015e-01
```

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.

5.8 Exercise

5.8.1 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 trial.

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

1. Control
2. Handwashing with soap
3. Improved nutrition through counselling and provision of lipid-based nutrient supplements
4. Combined water, sanitation, handwashing, and nutrition.
5. Improved sanitation
6. Combined water, sanitation, and handwashing
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 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 blip type should we use? Construct an appropriate `sl3` library for A, Y and B.
2. 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 dominant? Why do you think that is?
3. 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 dominant under realistic rules? Why do you think that is?

5.9 Summary

- 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.
- `tmle3mopttx` estimates the mean outcome under the optimal individualized treatment, where the candidate rules are restricted to only respond to a user-supplied subset of the baseline and intermediate covariates. In addition it provides options for realistic, data-adaptive interventions.
- 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.

5.9.1 Solutions

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("https://raw.githubusercontent.com/tlverse/tlverse-data/master/was
washb_data <- washb_data[!is.na(momage), lapply(.SD, as.numeric)]
head(washb_data, 3)
```

As before, we specify the NPSEM via the `node_list` object.

```
node_list <- list(
  W = names(washb_data)[!(names(washb_data) %in% c("whz", "tr"))],
  A = "tr", Y = "whz"
)
```

We pick few potential effect modifiers, including mother's education, current living conditions (floor), and possession of material items including the refrigerator. We concentrate of these covariates as they might be indicative of the socio-economic status of individuals involved in the trial. We can explore the distribution of our V, A and Y:

```
# V1, V2 and V3:
table(washb_data$momedu)
table(washb_data$floor)
table(washb_data$asset_refrig)
```

```
# A:
table(washb_data$tr)

# Y:
summary(washb_data$whz)
```

We specify a simply library for the outcome regression, propensity score and the blip function. Since our treatment is categorical, we need to define a multinomial learner for A and multivariate learner for B. We will define the `xgboost` over a grid of parameters, and initialize a mean learner.

```
# Initialize few of the learners:
grid_params <- list(
  nrounds = c(100, 500),
  eta = c(0.01, 0.1)
)
grid <- expand.grid(grid_params, KEEP.OUT.ATTRS = FALSE)
xgb_learners <- apply(grid, MARGIN = 1, function(params_tune) {
  do.call(Lrnr_xgboost$new, c(as.list(params_tune)))
})
lrn_mean <- Lrnr_mean$new()

## Define the Q learner, which is just a regular learner:
Q_learner <- Lrnr_sl$new(
  learners = list(
    xgb_learners[[1]], xgb_learners[[2]], xgb_learners[[3]],
    xgb_learners[[4]], lrn_mean
  ),
  metalearner = Lrnr_nnls$new()
)

# Define the g learner, which is a multinomial learner:
# specify the appropriate loss of the multinomial learner:
mn_metalearner <- make_learner(Lrnr_solnp,
  loss_function = loss_loglik_multinomial,
  learner_function = metalearner_linear_multinomial
)
g_learner <- make_learner(Lrnr_sl, list(xgb_learners[[4]], lrn_mean), mn_metalea

# Define the Blip learner, which is a multivariate learner:
learners <- list(
  xgb_learners[[1]], xgb_learners[[2]], xgb_learners[[3]],
  xgb_learners[[4]], lrn_mean
)
b_learner <- create_mv_learners(learners = learners)
```

```
learner_list <- list(Y = Q_learner, A = g_learner, B = b_learner)
```

As seen before, we initialize the `tmle3_mopttx_blip_revere` Spec in order to answer Question 1. We want to maximize our outcome, with higher the weight-for-height Z-score the better. We will also use `blip2` as the blip type, but note that we could have used `blip1` as well since "Control" is a good reference category.

```
## Question 2:

# Initialize a tmle specification
tmle_spec <- tmle3_mopttx_blip_revere(
  V = c("momedu", "floor", "asset_refrig"), type = "blip2",
  learners = learner_list, maximize = TRUE, complex = TRUE,
  realistic = FALSE
)

# Fit the TML estimator.
fit <- tmle3(tmle_spec, data = washb_data, node_list, learner_list)
fit

# Which intervention is the most dominant?
table(tmle_spec$return_rule)
```

Using the same formulation as before, we estimate the realistic optimal ITR and the corresponding value of the realistic ITR:

```
## Question 3:

# Initialize a tmle specification with "realistic=TRUE":
tmle_spec <- tmle3_mopttx_blip_revere(
  V = c("momedu", "floor", "asset_refrig"), type = "blip2",
  learners = learner_list, maximize = TRUE, complex = TRUE,
  realistic = TRUE
)

# Fit the TML estimator.
fit <- tmle3(tmle_spec, data = washb_data, node_list, learner_list)
fit

table(tmle_spec$return_rule)
```

Chapter 6

Stochastic Treatment Regimes (optional)

Nima Hejazi

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

Updated: 2021-10-13

6.1 Learning Objectives

1. Differentiate stochastic treatment regimes from static, dynamic, and optimal treatment regimes.
2. Describe how estimating causal effects of stochastic interventions informs a real-world data analysis.
3. Contrast a population level stochastic intervention policy from a modified treatment policy.
4. Estimate causal effects under stochastic treatment regimes with the `tmle3shift` `R` package.
5. Specify a grid of counterfactual shift interventions to be used for defining a set of stochastic intervention policies.
6. Interpret a set of effect estimates from a grid of counterfactual shift interventions.

7. Construct marginal structural models to measure variable importance in terms of stochastic interventions, using a grid of shift interventions.
8. Implement a shift intervention at the individual level, to facilitate shifting each individual to a value that's supported by the data.
9. Define novel shift intervention functions to extend the `tmle3shift` `R` package.

6.2 Introduction

In this section, we examine a simple example of stochastic treatment regimes in the context of a continuous treatment variable of interest, defining an intuitive causal effect through which to examine stochastic interventions more generally. As a first step to using stochastic treatment regimes in practice, we present the `tmle3shift` R package, which features an implementation of a recently developed algorithm for computing targeted minimum loss-based estimates of a causal effect based on a stochastic treatment regime that shifts the natural value of the treatment based on a shifting function $d(A, W)$. We will also use `tmle3shift` to construct marginal structural models for variable importance measures, implement shift interventions at the individual level, and define novel shift intervention functions.

6.3 Stochastic Interventions

- Present a relatively simple yet extremely flexible manner by which *realistic* causal effects (and contrasts thereof) may be defined.
- May be applied to nearly any manner of treatment variable – continuous, ordinal, categorical, binary – allowing for a rich set of causal effects to be defined through this formalism.
- Arguably the most general of the classes of interventions through which causal effects may be defined, and are conceptually simple.
- We may consider stochastic interventions in two ways:

    1. The equation f_A, which produces A, is replaced by a probabilistic mechanism $g_\delta(A \mid W)$ that differs from the original $g(A \mid W)$. The *stochastically modified* value of the treatment A_δ is drawn from a user-specified distribution $g_\delta(A \mid W)$, which may depend on the original

distribution $g(A \mid W)$ and is indexed by a user-specified parameter δ. In this case, the stochastically modified value of the treatment $A_\delta \sim g_\delta(\cdot \mid W)$.

2. The observed value A is replaced by a new value $A_{d(A,W)}$ based on applying a user-defined function $d(A, W)$ to A. In this case, the stochastic treatment regime may be viewed as an intervention in which A is set equal to a value based on a hypothetical regime $d(A, W)$, where regime d depends on the treatment level A that would be assigned in the absence of the regime as well as the covariates W. Stochastic interventions of this variety may be referred to as depending on the *natural value of treatment* or as *modified treatment policies* (Haneuse and Rotnitzky, 2013; Young et al., 2014).

6.3.1 Identifying the Causal Effect of a Stochastic Intervention

- The stochastic intervention generates a counterfactual random variable $Y_{d(A,W)} := f_Y(d(A, W), W, U_Y) \equiv Y_{g_\delta} := f_Y(A_\delta, W, U_Y)$, where $Y_{d(A,W)} \sim \mathcal{P}_0^\delta$.

- The target causal estimand of our analysis is $\psi_{0,\delta} := \mathbb{E}_{P_0^\delta}\{Y_{d(A,W)}\}$, the mean of the counterfactual outcome variable $Y_{d(A,W)}$. The statistical target parameter may also be denoted $\Psi(P_0) = \mathbb{E}_{P_0}\overline{Q}(d(A, W), W)$, where $\overline{Q}(d(A, W), W)$ is the counterfactual outcome value of a given individual under the stochastic intervention distribution (Díaz and van der Laan, 2018).

- In prior work, Díaz and van der Laan (2012) showed that the causal quantity of interest $\mathbb{E}_0\{Y_{d(A,W)}\}$ is identified by a functional of the distribution of O:
$$\psi_{0,d} = \int_{\mathcal{W}} \int_{\mathcal{A}} \mathbb{E}_{P_0}\{Y \mid A = d(a, w), W = w\} \cdot q_{0,A}^O(a \mid W = w) \cdot q_{0,W}^O(w) d\mu(a) d\nu(w).$$

- The four standard assumptions presented in 2 are necessary in order to establish identifiability of the causal parameter from the observed data via the statistical functional. These were

1. *Consistency*: $Y_i^{d(a_i,w_i)} = Y_i$ in the event $A_i = d(a_i, w_i)$, for $i = 1, \ldots, n$
2. *Stable unit value treatment assumption (SUTVA)*: $Y_i^{d(a_i,w_i)}$ does not depend on $d(a_j, w_j)$ for $i = 1, \ldots, n$ and $j \neq i$, or lack of interference (Rubin, 1978, 1980).

3. *Strong ignorability*: $A_i \perp\!\!\!\perp Y_i^{d(a_i,w_i)} \mid W_i$, for $i = 1, \ldots, n$.
4. *Positivity (or overlap)*: $a_i \in \mathcal{A} \implies d(a_i, w_i) \in \mathcal{A}$ for all $w \in \mathcal{W}$, where $\mathcal{A}$ denotes the support of $A \mid W = w_i \quad \forall i = 1, \ldots n$.

- With the identification assumptions satisfied, Díaz and van der Laan (2012) and Díaz and van der Laan (2018) provide an efficient influence function with respect to the nonparametric model $\mathcal{M}$ as
$$D(P_0)(x) = H(a, w)(y - \overline{Q}(a, w)) + \overline{Q}(d(a, w), w) - \Psi(P_0),$$
where the auxiliary covariate $H(a, w)$ may be expressed
$$H(a, w) = \mathbb{I}(a + \delta < u(w)) \frac{g_0(a - \delta \mid w)}{g_0(a \mid w)} + \mathbb{I}(a + \delta \geq u(w)),$$
which may be reduced to
$$H(a, w) = \frac{g_0(a - \delta \mid w)}{g_0(a \mid w)} + 1$$
in the case that the treatment is in the limits that arise from conditioning on W, i.e., for $A_i \in (u(w) - \delta, u(w))$.

6.4 Estimating the Causal Effect of a Stochastic Intervention with `tmle3shift`

We use `tmle3shift` to construct a targeted maximum likelihood (TML) estimator of of a causal effect of a stochastic treatment regime that shifts the natural value of the treatment based on a shifting function $d(A, W)$. We will follow the recipe provided by Díaz and van der Laan (2018), tailored to the `tmle3` framework:

1. Construct initial estimators g_n of $g_0(A, W)$ and Q_n of $\overline{Q}_0(A, W)$, perhaps 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. Estimate the parameter ϵ in the 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 alternative regression model incorporating weights.
4. Compute TML estimator Ψ_n of the target parameter, defining update $\overline{Q}_n^\star$ of the initial estimate $\overline{Q}_{n,\epsilon_n}$:
$$\Psi_n = \Psi(P_n^\star) = \frac{1}{n} \sum_{i=1}^{n} \overline{Q}_n^\star(d(A_i, W_i), W_i).$$

To start, let's load the packages we'll use and set a seed for simulation:

```
library(tidyverse)
library(data.table)
library(sl3)
library(tmle3)
library(tmle3shift)
set.seed(429153)
```

1. Construct initial estimators g_n of $g_0(A, W)$ and Q_n of $\overline{Q}_0(A, W)$.

We need to estimate two components of the likelihood in order to construct a TML estimator.

1. The outcome regression, $\hat{Q}_n$, which is a simple regression of the form $\mathbb{E}[Y \mid A, W]$.

```
# learners used for conditional expectation regression
mean_learner <- Lrnr_mean$new()
fglm_learner <- Lrnr_glm_fast$new()
xgb_learner <- Lrnr_xgboost$new(nrounds = 200)
sl_regression_learner <- Lrnr_sl$new(
  learners = list(mean_learner, fglm_learner, xgb_learner)
)
```

2. The second of these is an estimate of the treatment mechanism, $\hat{g}_n$, i.e., the *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. (2020), 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 estimation techniques.

```
# learners used for conditional densities (i.e., generalized propensity score)
haldensify_learner <- Lrnr_haldensify$new(
  n_bins = c(3, 5),
  lambda_seq = exp(seq(-1, -10, length = 200))
)
# semiparametric density estimator based on homoscedastic errors (HOSE)
hose_learner_xgb <- make_learner(Lrnr_density_semiparametric,
  mean_learner = xgb_learner
)
# semiparametric density estimator based on heteroscedastic errors (HESE)
hese_learner_xgb_fglm <- make_learner(Lrnr_density_semiparametric,
  mean_learner = xgb_learner,
  var_learner = fglm_learner
)
# SL for the conditional treatment density
sl_density_learner <- Lrnr_sl$new(
  learners = list(haldensify_learner, hose_learner_xgb,
                  hese_learner_xgb_fglm),
  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:

```
Q_learner <- sl_regression_learner
g_learner <- sl_density_learner
learner_list <- list(Y = Q_learner, A = g_learner)
```

6.4.1 Simulate Data

```
# simulate simple data for tmle-shift sketch
n_obs <- 1000 # 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  3.58065 1
2:  1  0  3.20718 1
3:  1  1  1.03584 1
4:  0  0 -0.65785 1
5:  1  1  3.01990 1
6:  1  1  2.78031 1
```

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 *directed acyclic graph* (DAG) via *nonparametric structural equation models* (NPSEMs).

To start, we will initialize a specification for the TMLE of our parameter of interest (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 $\delta = 0.5$.

```
# 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).

6.4.2 Targeted Estimation of Stochastic Interventions Effects

```
tmle_fit <- tmle3(tmle_spec, data, node_list, learner_list)

Iter: 1 fn: 1342.0479    Pars:  0.56750948 0.00000282 0.43248770
Iter: 2 fn: 1342.0479    Pars:  0.5675095127 0.0000005883 0.4324898990
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}]   0.8008  0.79801 0.012924 0.77268 0.82334
   psi_transformed lower_transformed upper_transformed
1:         0.79801           0.77268           0.82334
```

The `print` method of the resultant `tmle_fit` object conveniently displays the results from computing our TML estimator.

6.5 Stochastic Interventions over a Grid of Counterfactual Shifts

- Consider an arbitrary scalar δ that defines a counterfactual outcome $\psi_n = Q_n(d(A, W), W)$, where, for simplicity, let $d(A, W) = A + \delta$. A simplified expression of the auxiliary covariate for the TMLE of ψ is $H_n = \frac{g^\star(a|w)}{g(a|w)}$, where $g^\star(a \mid w)$ defines the treatment mechanism with the stochastic intervention implemented. In this manner, we can specify a *grid* of shifts δ to define a set of stochastic intervention policies in an *a priori* manner.

- To ascertain whether a given choice of the shift δ is acceptable, let there be a bound $C(\delta) = \frac{g^\star(a|w)}{g(a|w)} \leq M$, where $g^\star(a \mid w)$ is a function of δ in part, and M is a user-specified upper bound of $C(\delta)$. Then, $C(\delta)$ is a measure of the influence of a given observation (under a bound of the ratio of the conditional densities), which provides a way to limit the maximum influence of a given observation through a choice of the shift δ.

- For the purpose of using such a shift in practice, the present software provides the functions `shift_additive_bounded` and `shift_additive_bounded_inv`, which define a variation of this shift:
$$\delta(a, w) = \begin{cases} \delta, & C(\delta) \leq M \\ 0, \text{otherwise} \end{cases}, \tag{6.1}$$
which corresponds to an intervention in which the natural value of treatment of a given observational unit is shifted by a value δ in the case that the ratio

of the intervened density $g^\star(a \mid w)$ to the natural density $g(a \mid w)$ (that is, $C(\delta)$) does not exceed a bound M. In the case that the ratio $C(\delta)$ exceeds the bound M, the stochastic intervention policy does not apply to the given unit and they remain at their natural value of treatment a.

6.5.1 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 -1, 0, 1 on the scale of the treatment A.

```
# what's the grid of shifts we wish to consider?
delta_grid <- seq(from = -1, to = 1, by = 1)

# initialize a tmle specification
tmle_spec <- tmle_vimshift_delta(
  shift_grid = delta_grid,
  max_shifted_ratio = 2
)
```

6.5.2 Targeted Estimation of Stochastic Intervention 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, or simply invoke the `tmle3` wrapper function to fit the series of TML estimators (one for each parameter defined by the grid delta) in a single function call. For convenience, we choose the latter:

```
tmle_fit <- tmle3(tmle_spec, data, node_list, learner_list)
```

```
Iter: 1 fn: 1340.1739    Pars:  0.59196 0.12955 0.27849
Iter: 2 fn: 1340.1739    Pars:  0.59196 0.12956 0.27848
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}]  0.61243  0.61414 0.0140803 0.58655 0.64174
2:        TSM  E[Y_{A=NULL}]  0.74078  0.73900 0.0138950 0.71177 0.76623
3:        TSM  E[Y_{A=NULL}]  0.85109  0.84900 0.0112007 0.82705 0.87096
```

```
4: MSM_linear MSM(intercept)  0.73477  0.73405 0.0125574 0.70944 0.75866
5: MSM_linear     MSM(slope)  0.11933  0.11743 0.0043886 0.10883 0.12603
   psi_transformed lower_transformed upper_transformed
1:         0.61414           0.58655           0.64174
2:         0.73900           0.71177           0.76623
3:         0.84900           0.82705           0.87096
4:         0.73405           0.70944           0.75866
5:         0.11743           0.10883           0.12603
```

Remark: The `print` method of the resultant `tmle_fit` object conveniently displays the results from computing our TML estimator.

6.5.3 Inference with Marginal Structural Models

Since we consider estimating the mean counterfactual outcome ψ_n under several values of the intervention δ, taken from the aforementioned δ-grid, one approach for obtaining inference on a single summary measure of these estimated quantities involves leveraging working marginal structural models (MSMs). Summarizing the estimates ψ_n through a working MSM allows for inference on the *trend* imposed by a δ-grid to be evaluated via a simple hypothesis test on a parameter of this working MSM. Letting $\psi_\delta(P_0)$ be the mean outcome under a shift δ of the treatment, we have $\vec{\psi}_\delta = (\psi_\delta : \delta)$ with corresponding estimators $\vec{\psi}_{n,\delta} = (\psi_{n,\delta} : \delta)$. Further, let $\beta(\vec{\psi}_\delta) = \phi((\psi_\delta : \delta))$. By a straightforward application of the delta method (discussed previously), we may write the efficient influence function of the MSM parameter β in terms of the EIFs of each of the corresponding point estimates. Based on this, inference from a working MSM is rather straightforward. To wit, the limiting distribution for $m_\beta(\delta)$ may be expressed

$$\sqrt{n}(\beta_n - \beta_0) \to N(0, \Sigma),$$

where Σ is the empirical covariance matrix of $\text{EIF}_\beta(O)$.

```
tmle_fit$summary[4:5, ]
         type          param init_est tmle_est        se   lower   upper
1: MSM_linear MSM(intercept)  0.73477  0.73405 0.0125574 0.70944 0.75866
2: MSM_linear     MSM(slope)  0.11933  0.11743 0.0043886 0.10883 0.12603
   psi_transformed lower_transformed upper_transformed
1:         0.73405           0.70944           0.75866
2:         0.11743           0.10883           0.12603
```

6.5.4 Directly Targeting the MSM Parameter β

Note that in the above, a working MSM is fit to the individual TML estimates of the mean counterfactual outcome under a given value of the shift δ in the supplied grid. The parameter of interest β of the MSM is asymptotically linear (and, in fact, a TML estimator) as a consequence of its construction from individual TML estimators. In smaller samples, it may be prudent to perform a TML estimation procedure that targets the parameter β directly, as opposed to constructing it from several independently targeted TML estimates. An approach for constructing such an estimator is proposed in the sequel.

Suppose a simple working MSM $\mathbb{E}Y_{g_\delta^0} = \beta_0 + \beta_1\delta$, then a TML estimator targeting β_0 and β_1 may be constructed as

$$\overline{Q}_{n,\epsilon}(A, W) = \overline{Q}_n(A, W) + \epsilon(H_1(g), H_2(g),$$

for all δ, where $H_1(g)$ is the auxiliary covariate for β_0 and $H_2(g)$ is the auxiliary covariate for β_1.

To construct a targeted maximum likelihood estimator that directly targets the parameters of the working marginal structural model, we may use the `tmle_vimshift_msm` Spec (instead of the `tmle_vimshift_delta` Spec that appears above):

```
# 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)

Iter: 1 fn: 1338.2186    Pars:  0.621115419 0.000009161 0.378875424
Iter: 2 fn: 1338.2186    Pars:  0.6211155417 0.0000003725 0.3788840858
solnp--> Completed in 2 iterations
tmle_msm_fit
A tmle3_Fit that took 100 step(s)
         type          param init_est tmle_est        se   lower   upper
1: MSM_linear MSM(intercept)  0.73511  0.73540 0.0125979 0.71071 0.76009
2: MSM_linear     MSM(slope)  0.11894  0.11876 0.0042977 0.11033 0.12718
   psi_transformed lower_transformed upper_transformed
1:         0.73540           0.71071           0.76009
2:         0.11876           0.11033           0.12718
```

6.5.5 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("https://raw.githubusercontent.com/tlverse/tlverse-data/mast
washb_data <- washb_data[!is.na(momage) & !is.na(momheight), ]
head(washb_data, 3)
```

```
     whz          tr fracode month aged    sex momage         momedu momheight
1: -0.94 Handwashing  N06505     7  237   male     21 Primary (1-5y)    146.00
2: -1.13     Control  N06505     8  310 female     26   No education    148.90
3: -1.61     Control  N06524     3  162   male     25 Primary (1-5y)    153.75
       hfiacat Nlt18 Ncomp watmin elec floor walls roof asset_wardrobe
1: Food Secure     1    25      2    1     0     1    1               0
2: Food Secure     1     7      4    1     0     0    1               0
3: Food Secure     0    15      2    0     0     1    1               0
   asset_table asset_chair asset_khat asset_chouki asset_tv asset_refrig
1:           1           0          0            1        0            0
2:           1           1          0            1        0            0
3:           1           0          1            1        0            0
   asset_bike asset_moto asset_sewmach asset_mobile
1:          0          0             0            0
2:          0          0             0            1
3:          0          0             0            0
```

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 sections), 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 (up or down) of two years in the mother's age (i.e., $\delta = \{-2, 0, 2\}$); in this setting, a stochastic intervention would correspond to a policy advocating that potential mothers defer or accelerate plans of having a child for two calendar years, possibly implemented through the deployment of an encouragement design.

First, let's try a simple upward shift of just two years:

```
# initialize a tmle specification for just a single delta shift
washb_shift_spec <- tmle_shift(
  shift_val = 2,
  shift_fxn = shift_additive,
  shift_fxn_inv = shift_additive_inv
)
```

To examine the effect modification approach we looked at in previous chapters, we'll estimate the effect of this shift $\delta = 2$ while stratifying on the mother's education level (`momedu`, a categorical variable with three levels). For this, we augment our initialized `tmle3_Spec` object like so

```
# initialize effect modification specification around previous specification
washb_shift_strat_spec <-  tmle_stratified(washb_shift_spec)
```

Prior to running our analysis, we'll modify the `learner_list` object we had created to include just one of the semiparametric location-scale conditional density estimators, as fitting of these estimators is much faster than the more computationally intensive approach implemented in the `haldensify` package (Hejazi et al., 2020).

```
# learners used for conditional density regression (i.e., propensity score),
# but we need to turn on cross-validation for this conditional density learner
hose_learner_xgb_cv <- Lrnr_cv$new(
  learner = hose_learner_xgb,
  full_fit = TRUE
)

# modify learner list, using existing SL for Q fit
learner_list <- list(Y = Q_learner, A = hose_learner_xgb_cv)
```

Now we're ready to construct a TML estimate of the shift parameter at $\delta = 2$, stratified across levels of our variable of interest:

```
# fit stratified TMLE
strat_node_list <- copy(node_list)
strat_node_list$W <- setdiff(strat_node_list$W,"momedu")
strat_node_list$V <- "momedu"
washb_shift_strat_fit <- tmle3(washb_shift_strat_spec, washb_data, strat_node_list,
                               learner_list)
washb_shift_strat_fit
A tmle3_Fit that took 1 step(s)
             type                                 param init_est tmle_est       se
1:            TSM                     E[Y_{A=NULL}] -0.57056 -0.59444 0.058920
2: stratified TSM  E[Y_{A=NULL}] | V=Primary (1-5y) -0.60763 -0.70816 0.075504
```

```
3: stratified TSM    E[Y_{A=NULL}] | V=No education -0.65272 -0.85506 0.125655
4: stratified TSM E[Y_{A=NULL}] | V=Secondary (>5y) -0.52368 -0.44815 0.094647
      lower    upper psi_transformed lower_transformed upper_transformed
1: -0.70992 -0.47896        -0.59444          -0.70992          -0.47896
2: -0.85614 -0.56017        -0.70816          -0.85614          -0.56017
3: -1.10134 -0.60878        -0.85506          -1.10134          -0.60878
4: -0.63365 -0.26264        -0.44815          -0.63365          -0.26264
```

For the next 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 ($\delta = -2$) through up by two years ($\delta = 2$), incrementing by a single year between the two. To do this, we simply initialize a `Spec tmle_vimshift_delta` similar to how we did in a previous example:

```
# initialize a tmle specification for the variable importance parameter
washb_vim_spec <- tmle_vimshift_delta(
  shift_grid = seq(from = -2, to = 2, by = 1),
  max_shifted_ratio = 2
)
```

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
A tmle3_Fit that took 1 step(s)
         type          param init_est   tmle_est        se      lower
1:        TSM  E[Y_{A=NULL}] -0.56197 -0.5606288 0.0459486 -0.6506865
2:        TSM  E[Y_{A=NULL}] -0.56388 -0.5604088 0.0464538 -0.6514565
3:        TSM  E[Y_{A=NULL}] -0.56579 -0.5652941 0.0466314 -0.6566901
4:        TSM  E[Y_{A=NULL}] -0.56770 -0.5688357 0.0468849 -0.6607283
5:        TSM  E[Y_{A=NULL}] -0.56961 -0.5653488 0.0479085 -0.6592478
6: MSM_linear MSM(intercept) -0.56579 -0.5641032 0.0466524 -0.6555403
7: MSM_linear     MSM(slope) -0.00191 -0.0017867 0.0015524 -0.0048293
       upper psi_transformed lower_transformed upper_transformed
1: -0.4705711      -0.5606288        -0.6506865        -0.4705711
2: -0.4693611      -0.5604088        -0.6514565        -0.4693611
3: -0.4738982      -0.5652941        -0.6566901        -0.4738982
4: -0.4769430      -0.5688357        -0.6607283        -0.4769430
5: -0.4714498      -0.5653488        -0.6592478        -0.4714498
6: -0.4726662      -0.5641032        -0.6555403        -0.4726662
7:  0.0012559      -0.0017867        -0.0048293         0.0012559
```

6.6 Exercises

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?

2. Describe two (equivalent) ways in which the causal effects of stochastic interventions may be interpreted.

3. Using a grid of values of the shift parameter δ (e.g., $\{-1, 0, +1\}$), repeat the analysis on the variable of interest (`momage`), summarizing the trend for this sequence of shifts using a marginal structural model.

4. For either the grid of shifts in the example preceding the exercises or that estimated in (3) above, plot the resultant estimates against their respective counterfactual shifts. Graphically add to the scatterplot a line with slope and intercept equivalent to the MSM fit through the individual TML estimates.

5. How does the marginal structural model we used to summarize the trend along the sequence of shifts previously help to contextualize the estimated effect for a single shift? That is, how does access to estimates across several shifts and the marginal structural model parameters allow us to more richly interpret our findings?

Chapter 7

A Primer on the R6 Class System

A central goal of the Targeted Learning statistical paradigm is to estimate scientifically relevant parameters in realistic (usually nonparametric) models.

The tlverse is designed using basic OOP principles and the R6 OOP framework. While we've tried to make it easy to use the tlverse packages without worrying much about OOP, it is helpful to have some intuition about how the tlverse is structured. Here, we briefly outline some key concepts from OOP. Readers familiar with OOP basics are invited to skip this section.

7.1 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 R6, see https://adv-r.hadley.nz/r6.html, from Hadley Wickham's *Advanced R* (Wickham, 2014).

7.2 Object Oriented Programming: Python and R

OO concepts (classes with inherentence) were baked into Python from the first published version (version 0.9 in 1991). In contrast, R gets its OO "approach" from its predecessor, S, first released in 1976. For the first 15 years, S had no support for classes, then, suddenly, S got two OO frameworks bolted on in rapid succession: informal classes with S3 in 1991, and formal classes with S4 in 1998. This process continues, with new OO frameworks being periodically released, to try to improve the lackluster OO support in R, with reference classes (R5, 2010) and R6 (2014). Of these, R6 behaves most like Python classes (and also most like OOP focused languages like C++ and Java), including having method definitions be part of class definitions, and allowing objects to be modified by reference.

Bibliography

Baker, M. (2016). Is there a reproducibility crisis? a nature survey lifts the lid on how researchers view the crisis rocking science and what they think will help. *Nature*, 533(7604):452–455.

Buckheit, J. B. and Donoho, D. L. (1995). Wavelab and reproducible research. In *Wavelets and statistics*, pages 55–81. Springer.

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

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

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

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

Hejazi, N. S., Benkeser, D. C., and van der Laan, M. J. (2020). haldensify: Highly adaptive lasso conditional density estimation. https://github.com/nhejazi/haldensify. R package version 0.0.5.

Luby, S. P., Rahman, M., Arnold, B. F., Unicomb, L., Ashraf, S., Winch, P. J., Stewart, C. P., Begum, F., Hussain, F., Benjamin-Chung, J., et al. (2018). Effects of water quality, sanitation, handwashing, and nutritional interventions on diarrhoea and child growth in rural bangladesh: a cluster randomised controlled trial. *The Lancet Global Health*, 6(3):e302–e315.

Munafò, M. R., Nosek, B. A., Bishop, D. V., Button, K. S., Chambers, C. D., Du Sert, N. P., Simonsohn, U., Wagenmakers, E.-J., Ware, J. J., and Ioannidis, J. P. (2017). A manifesto for reproducible science. *Nature Human Behaviour*, 1(1):0021.

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

Nature Editorial (2015b). Let's think about cognitive bias. *Nature*, 526(7572).

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

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

Polley, E. C. and van der Laan, M. J. (2010). Super learner in prediction. *bepress*.

Pullenayegum, E. M., Platt, R. W., Barwick, M., Feldman, B. M., Offringa, M., and Thabane, L. (2016). Knowledge translation in biostatistics: a survey of current practices, preferences, and barriers to the dissemination and uptake of new statistical methods. *Statistics in medicine*, 35(6):805–818.

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

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

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

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

Szucs, D. and Ioannidis, J. (2017). When null hypothesis significance testing is unsuitable for research: a reassessment. *Frontiers in human neuroscience*, 11:390.

van der Laan, M. J. and Dudoit, S. (2003). Unified cross-validation methodology for selection among estimators and a general cross-validated adaptive epsilon-net estimator: Finite sample oracle inequalities and examples. *bepress*.

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

van der Laan, M. J. and Starmans, R. J. (2014). Entering the era of data science: Targeted learning and the integration of statistics and computational data analysis. *Advances in Statistics*, 2014.

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

Wickham, H. (2014). *Advanced r*. Chapman and Hall/CRC.

Young, J. G., Hernán, M. A., and Robins, J. M. (2014). Identification, estimation and approximation of risk under interventions that depend on the natural value of treatment using observational data. *Epidemiologic methods*, 3(1):1–19.