Targeted Learning in R

Causal Data Science with the tlverse Software Ecosystem

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

April 04, 2021

Contents

List of Tables

List of Figures

About this book

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

0.1 Outline

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

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

What this book is not

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

About the authors

Mark van der Laan

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

Jeremy Coyle

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

Nima Hejazi

Nima Hejazi is 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; her research integrates personalized medicine, human-computer interaction, experimental design, and regulatory policy.

Alan Hubbard

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

0.2 Learning resources

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

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

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

For a general introduction to causal inference, we recommend

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

0.3 Setup instructions

0.3.1 R and RStudio

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

0.3.1.1 Windows

0.3.1.1.1 If you already have R and RStudio installed

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

0.3.1.1.2 If you don't have R and RStudio installed

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

0.3.1.2 macOS / Mac OS X

0.3.1.2.1 If you already have R and RStudio installed

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

0.3.1.2.2 If you don't have R and RStudio installed

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

0.3.1.3 Linux

- Follow the instructions for your distribution from CRAN, they provide information to get the most recent version of R for common distributions. For most distributions, you could use your package manager (e.g., for Debian/Ubuntu run `sudo apt-get install r-base`, and for Fedora `sudo yum install R`), but we don't recommend this approach as the versions provided by this are usually out of date. In any case, make sure you have at least R 3.3.1.
- Go to the RStudio download page
- Under *Installers* select the version that matches your distribution, and install it with your preferred method (e.g., with Debian/Ubuntu `sudo dpkg -i rstudio-x.yy.zzz-amd64.deb` at the terminal).

- Once it's installed, open RStudio to make sure it works and you don't get any error messages.

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

Chapter 1

Robust Statistics and Reproducible Science

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

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; Editorial, 2015).

> "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)

The objective for this handbook 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. For an introduction to Targeted Learning, we recommend the recent review paper from Coyle et al. (2021).

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

Chapter 2

The Roadmap for Targeted Learning

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.

Introduction

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

2.1 The Roadmap

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: 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 Step 1. 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 subscribe to the belief that the random variable O being observed has, for example, a normal distribution with mean μ and variance σ^2. Formally, a parametric model may be defined

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

Sadly, the assumption that the data-generating distribution has a specific, parametric form is all too common, especially since this is a leap of faith or an assumption made of convenience. 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 such a practice often forces starkly different 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) $\mathbb{R}$. That is, $\Psi : \mathcal{M} \to \mathbb{R}$. The estimand may be seen as a representation of the quantity that we wish to learn from the data, the answer to a well-specified (often causal) question of interest. In contrast to purely statistical estimands, causal estimands require *identification from*

the observed data, based on causal models that include several untestable assumptions, described in 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 some estimators can be theoretically validated to be approximately normally distributed by a Central Limit Theorem (CLT).

A **Central Limit Theorem** (CLTs) is a statement 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 that the confidence interval procedure generates intervals containing the true estimand value (or 95% confidence of "covering" the true value). That is, any single estimated confidence interval either will contain the true estimand or will not (also called "coverage").

2.2 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, such that $O_1, \ldots, O_n \sim P_0$. 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 - hence, 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).

2.3 Causal Target Parameters

In many cases, we are interested in problems that ask questions regarding the effect of an intervention on a future outcome of interest. These questions can be represented as causal estimands.

The Causal Model

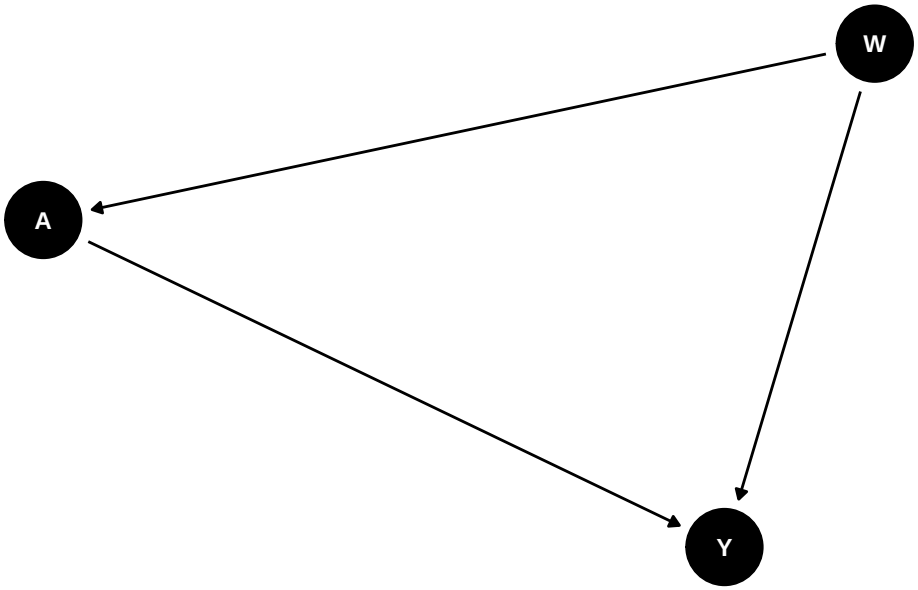
After formalizing the data and the statistical model, we can define a causal model to express causal parameters of interest. Directed acyclic graphs (DAGs) are one useful tool to express what we know about the causal relations among variables. Ignoring exogenous U terms (explained below), we assume the following ordering of the variables in the observed data O. We do this below using `DAGitty` (Textor et al., 2011):

```
library(dagitty)
library(ggdag)

# make DAG by specifying dependence structure
dag <- dagitty(
  "dag {
    W -> A
    W -> Y
    A -> Y
    W -> A -> Y
```

```
  }"
)
exposures(dag) <- c("A")
outcomes(dag) <- c("Y")
tidy_dag <- tidy_dagitty(dag)

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



While 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.

Identifiability

Because we can never observe both $Y(0)$ (the counterfactual outcome when $A = 0$) and $Y(1)$ (similarly, the counterfactual outcome when $A = 1$), we cannot estimate the quantity in Equation (2.1) directly. 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

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

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

Chapter 3

Welcome to the `tlverse`

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

What is the `tlverse`?

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

By analogy to the `tidyverse`:

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

So, the `tlverse` is

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

Anatomy of the tlverse

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.

3.1 Installation

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

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

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

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

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

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

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

1. Type `usethis::browse_github_pat()` in your R console, which will direct you to GitHub's page to create a New Personal Access Token (PAT).
2. Create a PAT simply by clicking "Generate token" at the bottom of the page.
3. Copy your PAT, a long string of lowercase letters and numbers.
4. Type `usethis::edit_r_environ()` in your R console, which will open your `.Renviron` file in the source window of RStudio.
    a. If your `.Renviron` file does not pop-up after calling `usethis::edit_r_environ()`; then try inputting `Sys.setenv(GITHUB_PAT = "yourPAT")`, replacing your PAT with inside the quotes. If this does not error, then skip to step 8.
5. In your `.Renviron` file, type `GITHUB_PAT=` and then paste your PAT after the equals symbol with no space.
6. In your `.Renviron` file, press the enter key to ensure that your `.Renviron` ends with a new line.
7. Save your `.Renviron` file. The example below shows how this syntax should look.

```
GITHUB_PAT=yourPAT
```

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

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

Chapter 4

Meet the Data

4.1 WASH Benefits Example Dataset

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

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

In the handbook, we concentrate on child growth (size for age) as the outcome

of interest. For reference, this trial was registered with ClinicalTrials.gov as NCT01590095.

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

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

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

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

skim_type	skim_variable	n_missing	complete_rate	character.min	character.max	character.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 socio-economic status of the study participants. Notice also the uniform distribution of the treatment groups (with twice as many controls); this is, of course, by design.

4.2 International Stroke Trial Example Dataset

The International Stroke Trial database contains individual patient data from the International Stroke Trial (IST), a multi-national randomized trial conducted between 1991 and 1996 (pilot phase between 1991 and 1993) that aimed to assess whether early administration of aspirin, heparin, both aspirin and heparin, or neither influenced the clinical course of acute ischaemic stroke (Sandercock et al., 1997). The IST dataset includes data on 19,435 patients with acute stroke, with 99% complete follow-up. De-identified data are available for download at `https://datashare.is.ed.ac.uk/handle/10283/128`. This study is described in more detail in Sandercock et al. (2011). The example data for this handbook considers a sample of 5,000 patients and the binary outcome of recurrent ischemic stroke within 14 days after randomization. Also in this example data, we ensure that we have subjects with a missing outcome.

```
# read in data
ist <- read_csv(
  paste0(
    "https://raw.githubusercontent.com/tlverse/tlverse-handbook/master/",
    "data/ist_sample.csv"
  )
)
```

We have 26 variables measured, and the outcome of interest, Y, indicates recurrent ischemic stroke within 14 days after randomization (`DRSISC` in `ist`); the treatment of interest, A, is the randomized aspirin vs. no aspirin treatment allocation (`RXASP` in `ist`); and the adjustment set, W, consists of all 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 it is automatically detected when `NA` are present in the outcome. This observed data structure can be denoted as n i.i.d. copies of $O_i = (W_i, A_i, \Delta_i, \Delta Y_i)$, for $i = 1, \ldots, n$, where Δ denotes the binary indicator that the outcome is observed.

Like before, we can summarize the variables measured in the IST sample data set with `skimr`:

skim_type	skim_variable	n_missing	complete_rate	character.min	character.max
character	RCONSC	0	1.000	1	1
character	SEX	0	1.000	1	1
character	RSLEEP	0	1.000	1	1
character	RATRIAL	0	1.000	1	1
character	RCT	0	1.000	1	1
character	RVISINF	0	1.000	1	1
character	RHEP24	0	1.000	1	1
character	RASP3	0	1.000	1	1
character	RDEF1	0	1.000	1	1
character	RDEF2	0	1.000	1	1
character	RDEF3	0	1.000	1	1
character	RDEF4	0	1.000	1	1
character	RDEF5	0	1.000	1	1
character	RDEF6	0	1.000	1	1
character	RDEF7	0	1.000	1	1
character	RDEF8	0	1.000	1	1
character	STYPE	0	1.000	3	4
character	RXHEP	0	1.000	1	1
character	REGION	0	1.000	10	26
numeric	RDELAY	0	1.000	NA	NA
numeric	AGE	0	1.000	NA	NA
numeric	RSBP	0	1.000	NA	NA
numeric	MISSING_RATRIAL_RASP3	0	1.000	NA	NA
numeric	MISSING_RHEP24	0	1.000	NA	NA
numeric	RXASP	0	1.000	NA	NA
numeric	DRSISC	10	0.998	NA	NA

4.3 NHANES I Epidemiologic Follow-up Study (NHEFS)

This data is from the National Health and Nutrition Examination Survey (NHANES) Data I Epidemiologic Follow-up Study. More coming soon.

```
# read in data
nhefs_data <- read_csv(
```

```
  paste0(
    "https://raw.githubusercontent.com/tlverse/tlverse-handbook/master/",
    "data/NHEFS.csv"
  )
)
```

A snapshot of the data set is shown below:

skim_type	skim_variable	n_missing	complete_rate	numeric.mean	numeric.sd	numeric.p0
numeric	seqn	0	1.00000	16552.36464	7498.91820	233.00000
numeric	qsmk	0	1.00000	0.26274	0.44026	0.00000
numeric	death	0	1.00000	0.19521	0.39649	0.00000
numeric	yrdth	1311	0.19521	87.56918	2.65941	83.00000
numeric	modth	1307	0.19767	6.25776	3.61530	1.00000
numeric	dadth	1307	0.19767	15.87267	8.90549	1.00000
numeric	sbp	77	0.95273	128.70941	19.05156	87.00000
numeric	dbp	81	0.95028	77.74483	10.63486	47.00000
numeric	sex	0	1.00000	0.50952	0.50006	0.00000
numeric	age	0	1.00000	43.91529	12.17043	25.00000
numeric	race	0	1.00000	0.13198	0.33858	0.00000
numeric	income	62	0.96194	17.94767	2.66328	11.00000
numeric	marital	0	1.00000	2.50338	1.08237	2.00000
numeric	school	0	1.00000	11.13505	3.08960	0.00000
numeric	education	0	1.00000	2.70350	1.19010	1.00000
numeric	ht	0	1.00000	168.74096	9.05313	142.87500
numeric	wt71	0	1.00000	71.05213	15.72959	36.17000
numeric	wt82	63	0.96133	73.46922	16.15805	35.38020
numeric	wt82_71	63	0.96133	2.63830	7.87991	-41.28047
numeric	birthplace	92	0.94352	31.59532	14.50050	1.00000
numeric	smokeintensity	0	1.00000	20.55126	11.80375	1.00000
numeric	smkintensity82_71	0	1.00000	-4.73788	13.74136	-80.00000
numeric	smokeyrs	0	1.00000	24.87109	12.19807	1.00000
numeric	asthma	0	1.00000	0.04850	0.21488	0.00000
numeric	bronch	0	1.00000	0.08533	0.27946	0.00000
numeric	tb	0	1.00000	0.01412	0.11802	0.00000
numeric	hf	0	1.00000	0.00491	0.06993	0.00000
numeric	hbp	0	1.00000	1.05095	0.95821	0.00000
numeric	pepticulcer	0	1.00000	0.10374	0.30502	0.00000
numeric	colitis	0	1.00000	0.03376	0.18067	0.00000
numeric	hepatitis	0	1.00000	0.01719	0.13001	0.00000
numeric	chroniccough	0	1.00000	0.05402	0.22613	0.00000
numeric	hayfever	0	1.00000	0.08963	0.28573	0.00000
numeric	diabetes	0	1.00000	0.97974	0.99579	0.00000
numeric	polio	0	1.00000	0.01412	0.11802	0.00000
numeric	tumor	0	1.00000	0.02333	0.15099	0.00000
numeric	nervousbreak	0	1.00000	0.02885	0.16744	0.00000
numeric	alcoholpy	0	1.00000	0.87600	0.33887	0.00000
numeric	alcoholfreq	0	1.00000	1.92020	1.30714	0.00000
numeric	alcoholtype	0	1.00000	2.47575	1.20816	1.00000
numeric	alcoholhowmuch	417	0.74401	3.28713	2.98470	1.00000
numeric	pica	0	1.00000	0.97545	0.99785	0.00000
numeric	headache	0	1.00000	0.62983	0.48300	0.00000
numeric	otherpain	0	1.00000	0.24616	0.43091	0.00000
numeric	weakheart	0	1.00000	0.02210	0.14705	0.00000
numeric	allergies	0	1.00000	0.06200	0.24123	0.00000

Chapter 5

Cross-validation

Ivana Malenica

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

Updated: 2021-04-04

5.1 Learning Objectives

1. Differentiate between training, validation and test sets.
2. Understand the concept of a loss function, risk and cross-validation.
3. Select a loss function that is appropriate for the functional parameter to be estimated.
4. Understand and contrast different cross-validation schemes for i.i.d. data.
5. Understand and contrast different cross-validation schemes for time dependent data.
6. Setup the proper fold structure, build custom fold-based function, and cross-validate the proposed function using the `origami R` package.
7. Setup the proper cross-validation structure for the use by the Super Learner using the the `origami R` package.

5.2 Introduction

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

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

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

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

5.3 Background

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

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

The training set is used to fit algorithm(s) of interest; we evaluate the performance of the fit(s) on a validation set, which can be used to estimate prediction error (e.g., for

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

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

5.3.1 Introducing: cross-validation

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

In great generality, cross-validation works by partitioning a sample into complementary subsets, applying a particular algorithm(s) on a subset (the training set), and evaluating the method of choice on the complementary subset (the validation/test set). This procedure is repeated across multiple partitions of the data. A variety of

different partitioning schemes exist, depending on the problem of interest, data size, prevalence of the outcome, and dependence structure. The `origami` package provides a suite of tools that generalize the application of cross-validation to arbitrary data analytic procedures. In the the following, we describe different types of cross-validation schemes readily available in `origami`, introduce the general structure of the `origami` package, and show their use in applied settings.

5.4 Estimation Roadmap: how does it all fit together?

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

1. **The loss funtion**: Define the target parameter as the minimizer of the expected loss (risk) for a full data loss function chosen to represent the desired performance measure. Map the full data loss function into an observed data loss function, having the same expected value and leading to an efficient estimator of risk.

2. **The algorithms**: Construct a finite collection of candidate estimators for the parameter of interest.

3. **The cross-validation scheme**: Apply appropriate cross-validation to select an optimal estimator among the candidates, and assess the overall performance of the resulting estimator.

Step 1 of the Estimation Roadmap allows us to unify a broad range of problems that are traditionally treated separately in the statistical literature, including density estimation, prediction of polychotomous and continuous outcomes. For example, if we are interested in estimating the full joint conditional density, we could use

the negative log-likelihood loss. If instead we are interested in the conditional mean with continuous outcome, one could use the squared error loss; had the outcome been binary, one could resort to the indicator (0-1) loss. The unified loss-based framework also reconciles censored and full data estimation methods, as full data estimators are recovered as special cases of censored data estimators.

5.5 Example: cross-validation and prediction

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

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

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

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

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

5.6 Cross-validation schemes in `origami`

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

WASH Benefits Study Example

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

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

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

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

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

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

5.6.1.1 Re-substitution

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

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

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

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

```
folds_resubstitution(nrow(washb_data))
#> [[1]]
#> $v
#> [1] 1
#>
#> $training_set
#>  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
#> [26] 26 27 28 29 30
#>
#> $validation_set
#>  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
#> [26] 26 27 28 29 30
#>
```

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

5.6.1.2 Holdout method

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

The holdout method is intuitive, conceptually easy, and computationally not too demanding. However, if we repeat the process of randomly splitting the data into the training and validation set, we might get a different validation loss (e.g., MSE). In particular, the loss over the validation sets might be highly variable, depending on which samples were included in the training/validation split. For classification problems, there is a possibility of an uneven distribution of different classes in the training and validation set unless data is stratified. Finally, note that we are not using all of the data to train and evaluate the performance of the proposed algorithm, which might result in bias.

5.6.1.3 Leave-one-out

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

We can repeat the process of spiting the data into training and validation set until all samples are part of the validation set at some point. For example, next iteration of the cross-validation might have $x_2 = (w_2, y_2)$ as the validation set and all the rest of $n - 1$ samples as the training set. Repeating this approach n times results in, for example, n squared errors $MSE_1, MSE_2, \ldots, MSE_n$. The estimate of the

true risk is the average over the n squared errors. While the leave-one-out cross-validation results in a less biased (albeit, more variable) estimate of risk than the holdout method, it could be expensive to implement if n is large.

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

```
folds <- folds_loo(nrow(washb_data))
folds[[1]]
#> $v
#> [1] 1
#>
#> $training_set
#>  [1]  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
#> [26] 27 28 29 30
#>
#> $validation_set
#> [1] 1
#>
#> attr(,"class")
#> [1] "fold"
folds[[2]]
#> $v
#> [1] 2
#>
#> $training_set
#>  [1]  1  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
#> [26] 27 28 29 30
#>
#> $validation_set
#> [1] 2
#>
#> attr(,"class")
#> [1] "fold"
```

5.6.1.4 V-fold

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

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

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

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

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

```
#> [1] 1
#>
#> $training_set
#>  [1]  2  3  4  6  7  8 11 12 14 15 19 22 23 24 28
#>
#> $validation_set
#>  [1]  1  5  9 10 13 16 17 18 20 21 25 26 27 29 30
#>
#> attr(,"class")
#> [1] "fold"
folds[[2]]
#> $v
#> [1] 2
#>
#> $training_set
#>  [1]  1  5  9 10 13 16 17 18 20 21 25 26 27 29 30
#>
#> $validation_set
#>  [1]  2  3  4  6  7  8 11 12 14 15 19 22 23 24 28
#>
#> attr(,"class")
#> [1] "fold"
```

5.6.1.5 Monte Carlo

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

Since the partitions are independent across folds, the same sample can appear in the validation set multiple times – note that this is a stark difference between Monte Carlo and V-fold cross-validation. With Monte Carlo cross-validation, one is able to explore many more available partitions than with V-fold cross-validation – resulting in a possibly less variable estimate of the risk, at a cost of an increase in bias.

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

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

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

```
folds <- folds_montecarlo(nrow(washb_data), V = 2, pvalidation = 0.2)
folds[[1]]
#> $v
#> [1] 1
#>
#> $training_set
#>  [1] 19 27 16 29 23 12  1  3 18 11  5  7  8  6  9 22 10 25 20 28 15  2 24 26
#>
#> $validation_set
#> [1]  4 13 14 17 21 30
#>
#> attr(,"class")
#> [1] "fold"
folds[[2]]
#> $v
#> [1] 2
#>
#> $training_set
#>  [1] 19 15 28 25 29 11 20 17 14  4  9 12 30  8 27 18 16 10 13  6 24  3 26  1
#>
#> $validation_set
#> [1]  2  5  7 21 22 23
#>
#> attr(,"class")
#> [1] "fold"
```

5.6.1.6 Bootstrap

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

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

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

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

```
folds <- folds_bootstrap(nrow(washb_data), V = 2)
folds[[1]]
#> $v
#> [1] 1
#>
#> $training_set
#>  [1]  2  5 30  1 29 16 10 11  8 25 28  2 11  2 16 28 15 28  1 27  9 19 20 30 18
#> [26] 11 13  2 18 12
#>
#> $validation_set
#>  [1]  3  4  6  7 14 17 21 22 23 24 26
#>
#> attr(,"class")
#> [1] "fold"
folds[[2]]
#> $v
#> [1] 2
```

```
#>
#> $training_set
#>  [1] 12 16 10 29 22 15 27  9 27 16 12 28 10 28 26  1 14  6 23 14 21 16  5 20
#> [26] 23 25  8 27  5
#>
#> $validation_set
#>  [1]  2  3  4  7 11 13 17 18 19 24 30
#>
#> attr(,"class")
#> [1] "fold"
```

5.6.2 Cross-validation for dependent data

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

AirPassenger Example

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

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

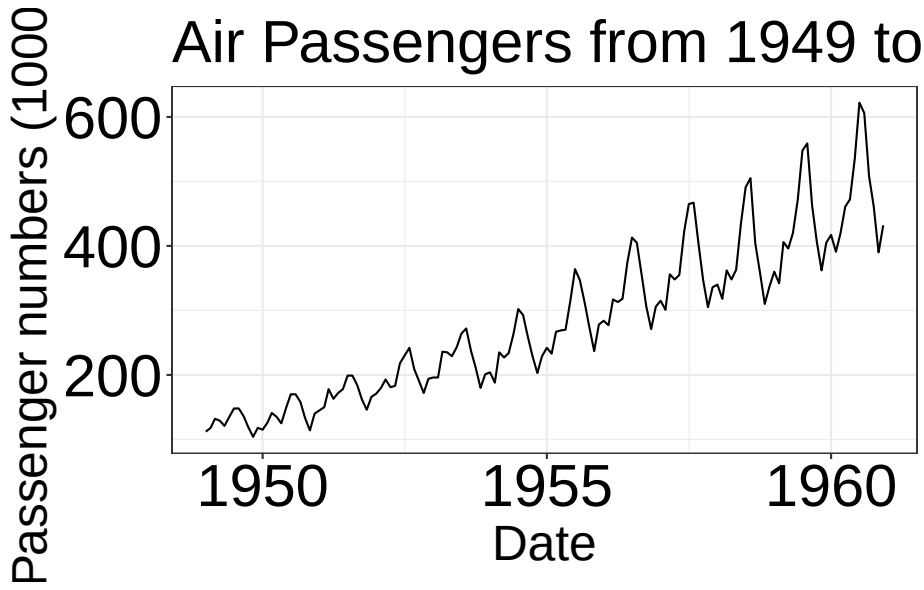
```
library(ggfortify)

data(AirPassengers)
AP <- AirPassengers

autoplot(AP) +
  labs(
    x = "Date",
```

```
    y = "Passenger numbers (1000's)",
    title = "Air Passengers from 1949 to 1961"
  )

t <- length(AP)
```



5.6.2.1 Rolling origin

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

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

original 15 we started with. We then evaluate its performance on 10 time points in the future.

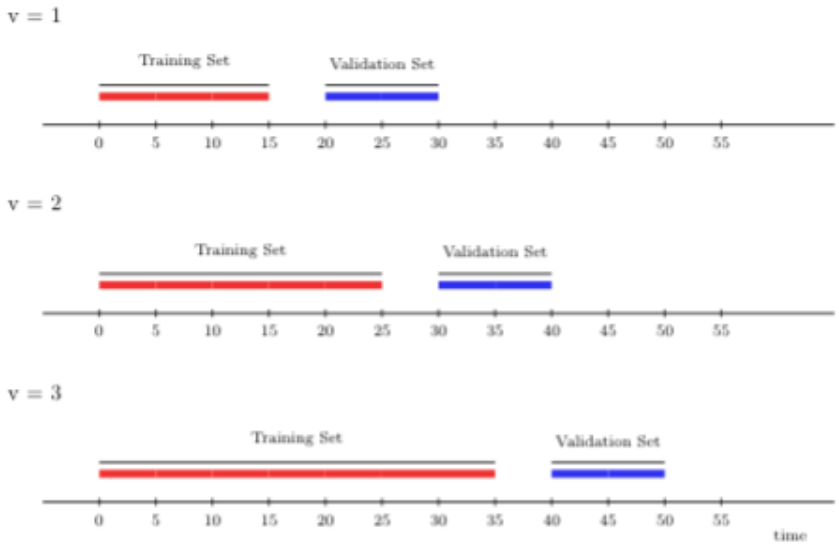


Figure 5.1: Rolling origin CV

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

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

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

```
folds <- folds_rolling_origin(
  t,
  first_window = 50, validation_size = 10, gap = 5, batch = 20
)
folds[[1]]
#> $v
#> [1] 1
#>
#> $training_set
#>  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
#> [26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
#>
#> $validation_set
#>  [1] 56 57 58 59 60 61 62 63 64 65
#>
#> attr(,"class")
#> [1] "fold"
folds[[2]]
#> $v
#> [1] 2
#>
#> $training_set
#>  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
#> [26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
#> [51] 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70
#>
#> $validation_set
#>  [1] 76 77 78 79 80 81 82 83 84 85
#>
#> attr(,"class")
#> [1] "fold"
```

5.6.2.2 Rolling window

Instead of adding more time points to the training set per each iteration, the rolling window cross-validation scheme "rolls" the training sample forward by m time units. The rolling window scheme might be considered in parametric settings when one

wishes to guard against moment or parameter drift that is difficult to model explicitly; it is also more efficient for computationally demanding settings such as streaming data, in which large amounts of training data cannot be stored. In contrast to rolling origin CV, the training sample for each iteration of the rolling window scheme is always the same.

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

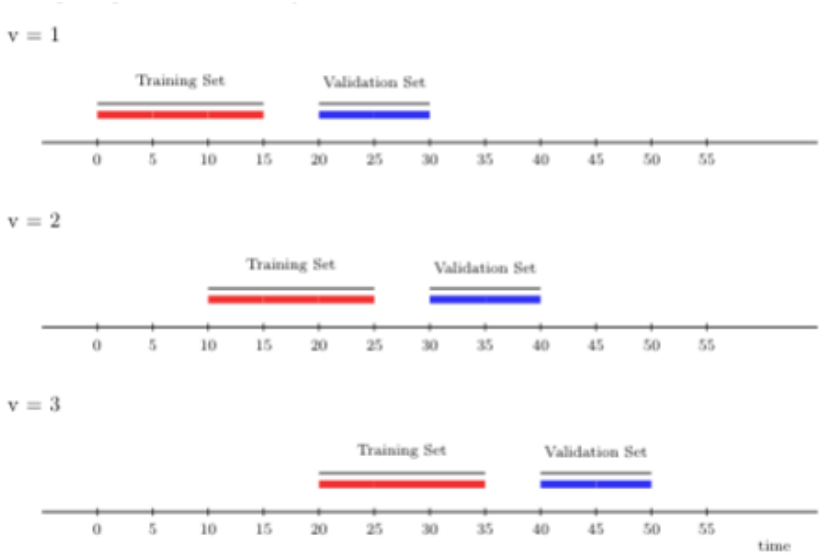


Figure 5.2: Rolling window CV

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

1. the total number of time points we want to cross-validate

2. the size of the training sets
3. the size of the validation set
4. the gap between training and validation set
5. the size of the update on the training set per each iteration of CV

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

```
folds <- folds_rolling_window(
  t,
  window_size = 50, validation_size = 10, gap = 5, batch = 20
)
folds[[1]]
#> $v
#> [1] 1
#>
#> $training_set
#>  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
#> [26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
#>
#> $validation_set
#>  [1] 56 57 58 59 60 61 62 63 64 65
#>
#> attr(,"class")
#> [1] "fold"
folds[[2]]
#> $v
#> [1] 2
#>
#> $training_set
#>  [1] 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45
#> [26] 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70
#>
#> $validation_set
#>  [1] 76 77 78 79 80 81 82 83 84 85
#>
#> attr(,"class")
#> [1] "fold"
```

5.6.2.3 Rolling origin with V-fold

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

5.6.2.4 Rolling window with v-fold

Analogous to the previous section, we can extend rolling window CV to support multiple time-series with arbitrary sample dependence. One can use the rolling-window-V-fold cross-validation with `origami` package using the function `folds_vfold_rolling_window_pooled(n, t, id, time, V, window_size, validation_size, gap, batch)`. In the figure below, we show $V = 2$ V-folds, and 2 time-series CV folds.

5.7 General workflow of `origami`

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

5.7.1 (1) Define folds

The `folds` object passed to `cross_validate` is a list of folds; such lists can be generated using the `make_folds` function. Each fold consists of a list with a `training` index vector, a `validation` index vector, and a `fold_index` (its order in the list of

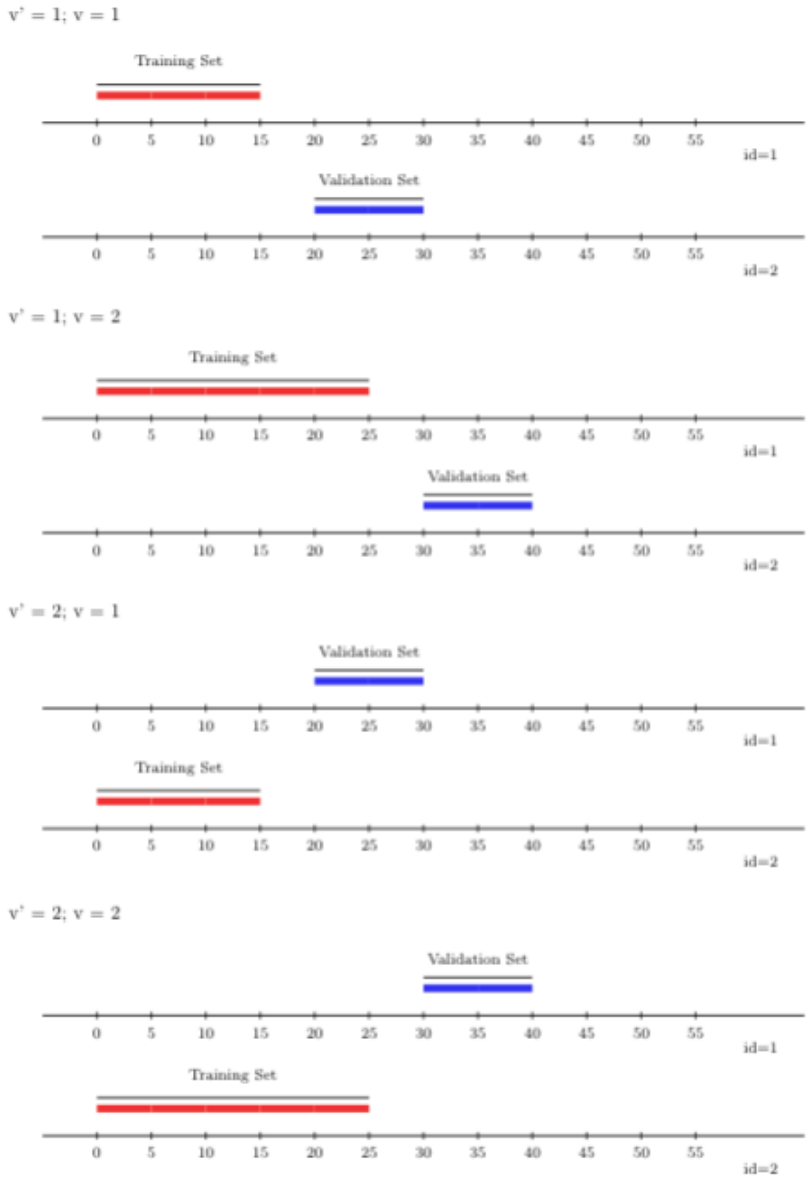


Figure 5.3: Rolling origin V-fold CV

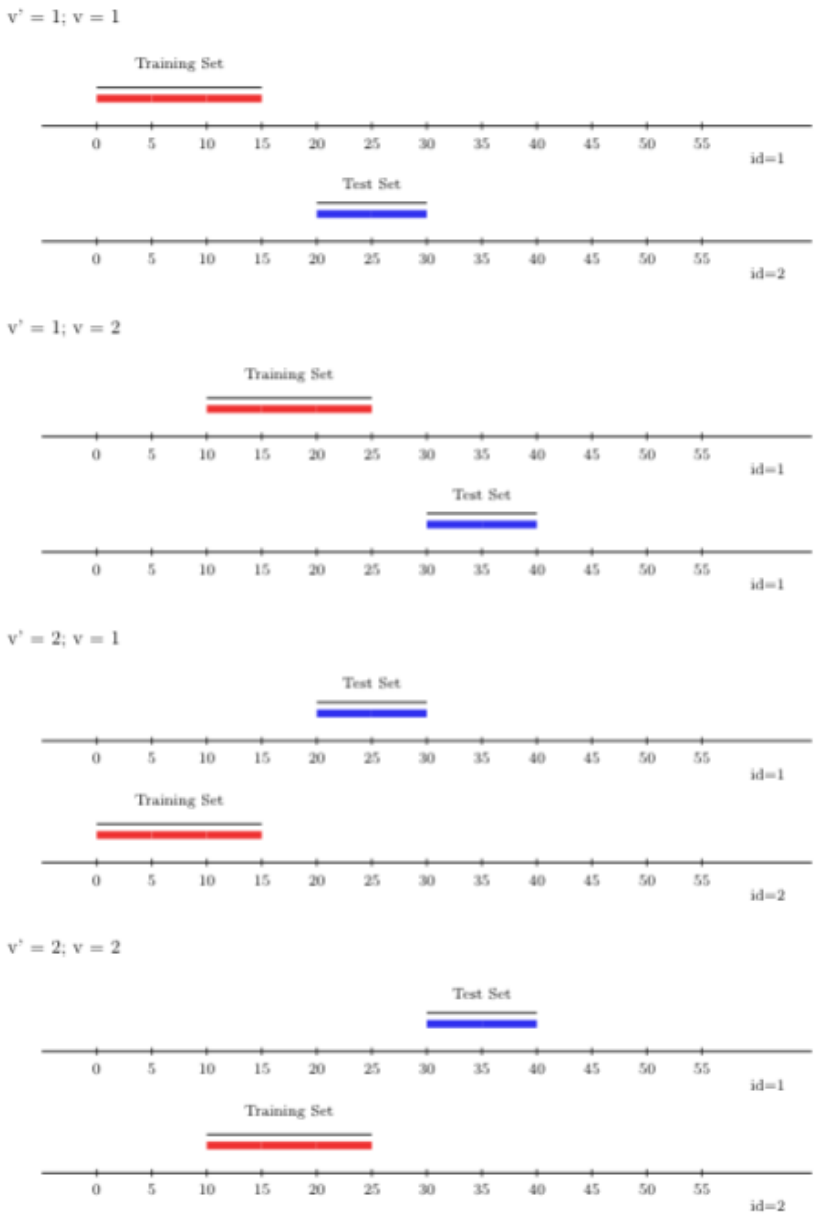


Figure 5.4: Rolling window V-fold CV

folds). This function supports a variety of cross-validation schemes we describe in the following section. The `make_folds` can balance across levels of a variable (`strata_ids`), and it can also keep all observations from the same independent unit together (`cluster`).

5.7.2 (2) Define fold function

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

5.7.3 (3) Apply `cross_validate`

After defining folds, `cross_validate` can be used to map the `cv_fun` across the `folds` using `future_lapply`. This means that it can be easily parallelized by specifying a parallelization scheme (i.e., a `plan` from the future parallelization framework for `R` (Bengtsson, 2020)). The application of `cross_validate` generates a list of results. As described above, each call to `cv_fun` itself returns a list of results, with different elements for each type of result we care about. The main loop generates a list of these individual lists of results (a sort of "meta-list"). This "meta-list" is then inverted such that there is one element per result type (this too is a list of the results for each fold). By default, `combine_results` is used to combine these results type lists in a sensible manner. How results are combined is determined automatically by examining the data types of the results from the first fold. This can be modified by specifying a list of arguments to `.combine_control`.

5.8 Cross-validation in action

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

5.8.1 Cross-validation with linear regression

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

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

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

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

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

Here's a look at the data:

whz	tr	fracode	month	aged	sex	momage	momedu	momheight	hfiacat
0.00	Control	N05265	9	268	male	30	Primary (1-5y)	146.40	Food 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

We can see the covariates used in the prediction:

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

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

```
lm_mod <- lm(whz ~ ., data = washb_data)
summary(lm_mod)
#>
#> Call:
#> lm(formula = whz ~ ., data = washb_data)
#>
#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -2.8890 -0.6799 -0.0169  0.6595  3.1005
```

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

```
#> Nlt18                          -0.02557    0.04060   -0.63   0.5291
#> Ncomp                           0.00179    0.00762    0.23   0.8145
#> watmin                          0.01347    0.00861    1.57   0.1182
#> elec                            0.08906    0.10700    0.83   0.4057
#> floor                          -0.17763    0.17734   -1.00   0.3171
#> walls                          -0.03001    0.21445   -0.14   0.8888
#> roof                           -0.03716    0.49214   -0.08   0.9399
#> asset_wardrobe                 -0.05754    0.13736   -0.42   0.6755
#> asset_table                    -0.22079    0.12276   -1.80   0.0728 .
#> asset_chair                     0.28012    0.13750    2.04   0.0422 *
#> asset_khat                      0.02306    0.11766    0.20   0.8447
#> asset_chouki                   -0.13943    0.14084   -0.99   0.3227
#> asset_tv                        0.17723    0.12972    1.37   0.1726
#> asset_refrig                    0.12613    0.23162    0.54   0.5863
#> asset_bike                     -0.02568    0.10083   -0.25   0.7990
#> asset_moto                     -0.32094    0.19944   -1.61   0.1083
#> asset_sewmach                   0.05090    0.17795    0.29   0.7750
#> asset_mobile                    0.01420    0.14972    0.09   0.9245
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 0.984 on 447 degrees of freedom
#> Multiple R-squared:  0.129,  Adjusted R-squared:  0.0277
#> F-statistic: 1.27 on 52 and 447 DF,  p-value: 0.104
```

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

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

The mean squared error is 0.86568. There is an important problem that arises when we assess the model in this way - that is, we have trained our linear regression model on the full data set and assessed the error on the full data set, using up all of our data. We, of course, are generally not interested in how well the model explains variation

in the observed data; rather, we are interested in how the explanation provided by the model generalizes to a target population from which the sample is presumably derived. Having used all of our available data, we cannot honestly evaluate how well the model fits (and thus explains) variation at the population level.

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

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

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

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

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

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

Our `cv_lm` function is rather simple: we merely split the available data into a training and validation sets, using the eponymous functions provided in `origami`, fit the linear model on the training set, and evaluate the model on the testing set. This is a

simple example of what `origami` considers to be `cv_fun` – functions for using cross-validation to perform a particular routine over an input data set. Having defined such a function, we can simply generate a set of partitions using `origami`'s `make_folds` function, and apply our `cv_lm` function over the resultant `folds` object. Below, we replicate the re-substitution estimate of the error – we did this "by hand" above – using the functions `make_folds` and `cv_lm`.

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

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

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

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

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

5.8.2 Cross-validation with random forests

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

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

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

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

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

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

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

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

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

```
)
mean(cvrf_results$SE)
#> [1] 1.0271
```

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

5.8.3 Cross-validation with arima

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

```
data(AirPassengers)
print(AirPassengers)
#>      Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
#> 1949 112 118 132 129 121 135 148 148 136 119 104 118
#> 1950 115 126 141 135 125 149 170 170 158 133 114 140
#> 1951 145 150 178 163 172 178 199 199 184 162 146 166
#> 1952 171 180 193 181 183 218 230 242 209 191 172 194
#> 1953 196 196 236 235 229 243 264 272 237 211 180 201
#> 1954 204 188 235 227 234 264 302 293 259 229 203 229
#> 1955 242 233 267 269 270 315 364 347 312 274 237 278
#> 1956 284 277 317 313 318 374 413 405 355 306 271 306
#> 1957 315 301 356 348 355 422 465 467 404 347 305 336
#> 1958 340 318 362 348 363 435 491 505 404 359 310 337
#> 1959 360 342 406 396 420 472 548 559 463 407 362 405
#> 1960 417 391 419 461 472 535 622 606 508 461 390 432
```

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

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

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

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

By default, `folds_rolling_origin` will increase the size of the training set by one

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

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

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

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

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

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

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

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

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

5.9 Exercises

5.9.1 Review of Key Concepts

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

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

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

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

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

5.9.2 The Ideas in Action

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

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

5.9.3 Advanced Topics

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

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

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

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

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

Chapter 6

Super (Machine) Learning

Rachael Phillips

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

Updated: 2021-04-04

Learning Objectives

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

1. Select a loss function that is appropriate for the functional parameter to be estimated.
2. Assemble an ensemble of learners based on the properties that identify what features they support.
3. Customize learner tuning parameters to incorporate a diversity of different settings.
4. Select a subset of available covariates and pass only those variables to the modeling algorithm.
5. Fit an ensemble with nested cross-validation to obtain an estimate of the performance of the ensemble itself.
6. Obtain `sl3` variable importance metrics.
7. Interpret the discrete and continuous Super Learner fits.
8. Rationalize the need to remove bias from the Super Learner to make an optimal bias-variance tradeoff for the parameter of interest.

Motivation

- A common task in data analysis is prediction – using the observed data to learn a function, which can be used to map new input variables into a predicted outcome.
- For some data, algorithms that can model a complex function are necessary to adequately model the data. For other data, a main terms regression model might fit the data quite well.
- The Super Learner (SL), an ensemble learner, solves this issue, by allowing a combination of algorithms from the simplest (intercept-only) to most complex (neural nets, random forests, SVM, etc).
- It works by using cross-validation in a manner which guarantees that the resulting fit will be as good as possible, given the learners 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. Data as a random variable, or equivalently, a realization of a particular experiment/study. We assume the observations in the data are independent and identically distributed.
2. A statistical model as the set of possible probability distributions that could have given rise to the observed data.
3. A translation of the scientific question, which is often causal, into a target estimand.

Note that if the estimand is causal, step 3 also requires establishing identifiability of the estimand from the observed data, under possible non-testable assumptions that may not necessarily be reasonable. Still, the target quantity does have a valid statistical interpretation. See causal target parameters for more detail on causal models and identifiability.

Now that we have defined the statistical estimation problem, we are ready to construct the TMLE; an asymptotically linear and efficient substitution estimator of

this estimand. The first step in this estimation procedure is an initial estimate of the data-generating distribution, or the relevant part of this distribution that is needed to evaluate the target parameter. For this initial estimation, we use the Super Learner (SL) (van der Laan et al., 2007).

The SL provides an important step in creating a robust estimator. It is a loss-function-based tool that uses cross-validation to obtain the best prediction of our target parameter, based on a weighted average of a library of machine learning algorithms. The library of machine learning algorithms consists of functions ("learners" in the `sl3` nomenclature) that we think might be consistent with the true data-generating distribution. By "consistent with the true data-generating distribution", we mean that the algorithms selected should not violate subject-matter knowledge about the experiment that generated the data. Also, the library should contain a diversity of algorithms that range from parametric regression models to multi-step algorithms involving screening covariates, penalizations, optimizing tuning parameters, etc.

The ensembling of the collection of algorithms with weights ("metalearning" in the `sl3` nomenclature) has been shown to be adaptive and robust, even in small samples (Polley and van der Laan, 2010). The SL is proven to be asymptotically as accurate as the best possible prediction algorithm in the library (van der Laan and Dudoit, 2003; Van der Vaart et al., 2006).

Step-by-step overview

Consider the scenario in which we have n independent and identically distributed observations in the data, and our data structure is not a time series. Also, let's say we have k number of candidate learners/algorithms.

1. Create the validation data for all $V = v$ folds. Break up the data evenly into $V = v$ splits; such that no observation is contained more than one split, and the splits contain about the same number of observations (e.g., about n/V observations in each split).

    - If a rare binary outcome (or highly important binary predictor, such as a treatment) is present in the data, we should consider making the prevalence of this binary outcome in the splits similar to the prevalence that exists in the data. We can achieve this by specifying, for the `strata_ids`

argument in `origami::make_folds()`, the vector of binary outcomes (or important binary covariate).
- If we have repeated measures or cluster-level dependence in the data, then all observations within a subject/cluster should be placed in the same split.

2. For each fold v:

   a. Separate (i) from (ii):

   b. The data that was selected for fold v in Step 1, which contains roughly n/V total observations. We will refer to this subset of the data as the "validation" data, and it's also commonly referred to as the "test" data. Let's call $n_{\text{validation}}$ as the number of observations in then validation data,

   ii. The data that was NOT selected for fold v in Step 1, which contains roughly $n - n_{\text{validation}}$ total observations. We will refer to this subset of the data as the "training", and n_{training} as the number of observations in the training data.

   b. Fit each of the k learners on the training data (ii).
   c. Using each of the k trained learners, predict the outcomes in the validation data (i). We can call these predictions "cross-validated predictions"; since they were obtained from the validation sample's covariate information, which was never seen while fitting these models. We end up with a $n_{validation} \times k$ matrix of cross-validated predictions.

3. Bind together the rows of all v $n_{validation} \times k$ matrices of cross-validated predictions, to obtain an $n \times k$ matrix of cross-validated predictions. This matrix $n \times k$ matrix of cross-validated predictions is often referred to as the "level-one" or "Z" matrix.

4. Retain the observed outcome Y for all of the n observations, using them to measure the "loss" of each cross-validated prediction (e.g., $(Y - \hat{Y}^2)$.

5. For each k column, take the (potentially weighted) mean across all of the n losses, which we call the "cross-validated empirical risk". The cross-validated empirical risk provides measure of performance, summarized across all n, for each of the k learners. The weights that could be incorporated in the data, and used to calculate a weighted mean, serve to to up weight or down weight samples whose loss should be considered less or more important, respectively.

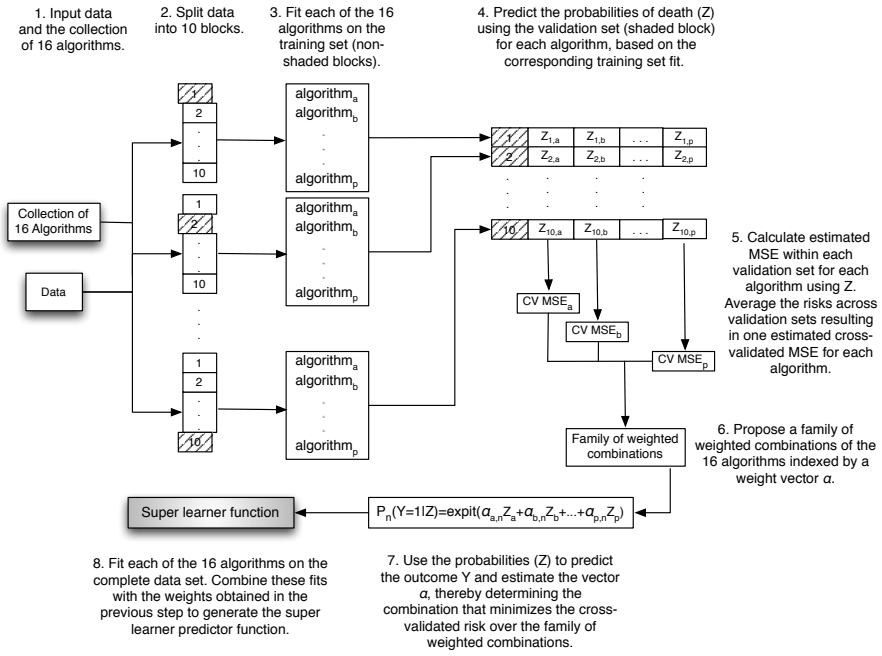
6. Establish the ensemble/combination of the k learners by fitting the so-called "metalearner". The ensemble is just a weighted combination of the learners, so the weights here are just a k-dimensional vector. The metalearner is a function that decides on the weights to be assigned to each of the k learners; taking as input the cross-validated empirical risk for all k learners (a), or taking as input a loss function and the Z matrix (b).

    a. The discrete SL (or cross-validated selector) employs a simple metalearner that takes as input the cross-validated empirical risk for all k learners. This metalearner assigns weight of 1 to the single learner with smallest cross-validated risk, and a weight of 0 to all other learners.
    b. The ensemble SL (often referred to as the "Super Learner") employs metalearners that take as input the Z matrix, and the loss function of interest (unless the loss is implied by the metalearning function itself). These metalearners assign the weights such that the weighted combination of Z matrix predictions is optimized to minimize the cross-validated empirical risk. This often results in more than one learner having positive weight. Aggressive metalearning (e.g., assigning negative weight) can be problematic, leading to overfitting.

7. Fit the learners (or only the learners with non-zero weight from Step 5) on the entire sample of n observations, and use the weights that were obtained in Step 5 to get the SL fit. That's it! The SL fit is just all of k learner fits — the weights don't come in to play until we obtain the predictions from the SL; where the SL predictions are the weighted combination of the k learner predictions, as determined by the metalearner. Notice that, we use this rigorous, optimal, and fair procedure to derive the weights from the $n \times k$ Z matrix of cross-validated predictions; but once we've done that, we capitalize on the entire sample of observations, transitioning our focus to obtaining the best fit possible of our k learners.

8. SL predictions, variable importance, and/or a cross-validated SL can be obtained from an SL fit (like most other learners). The cross-validated SL provides an estimate of the performance of the SL on unseen data, and incorporates a outer layer of cross-validation in order to cross-validate this entire procedure.

Below is a figure from [ADD REF] describing the same step-by-step procedure. This figure considers $k = 16$ learners, and in the figure $p = k$; and the squared error loss function, thus mean squared error (MSE) is the risk.



Theoretical Foundations and Further Reading

- Cross-validation is proven to be optimal for selection among estimators. This result was established through the oracle inequality for the cross-validation selector among a collection of candidate estimators (van der Laan and Dudoit, 2003; Van der Vaart et al., 2006). The only condition is that loss function is uniformly bounded, which is guaranteed in `sl3`.

- We use a *loss function* L to assign a measure of performance to each learner ψ when applied to the data O, and subsequently compare performance across the learners. More generally, L maps every $\psi \in \mathbb{R}$ to $L(\psi) : (O) \mapsto L(\psi)(O)$. We use the terms "learner", "algorithm", and "estimator" interchangeably.

  - It is important to recall that ψ is an estimator of ψ_0, the unknown and true parameter value under P_0.
  - A valid loss function will have mean/expectation (risk) that is minimized at the true value of the parameter ψ_0. Thus, minimizing the expected loss will bring an estimator ψ closer to the true ψ_0.
  - For example, say we observe a learning data set $O_i = (Y_i, X_i)$, of $i = 1, \ldots, n$ independent and identically distributed observations, where Y_i is

a continuous outcome of interest, X_i is a set of covariates. Also, let our objective be to estimate the function $\psi_0 : X \mapsto \psi_0(X) = E_0(Y \mid X)$, which is the conditional mean outcome given covariates. This function can be expressed as the minimizer of the expected squared error loss: $\psi_0 = \text{argmin}_\psi E[L(O, \psi(X))]$, where $L(O, \psi(X)) = (Y - \psi(X))^2$.
  - We can estimate the loss by substituting the empirical distribution of the data P_n for the true and unknown distribution of the observed data P_0.
  - Also, we can use the cross-validated risk to empirically determine the relative performance of an estimator (i.e., a candidate learner), and perhaps how it's performance compares to other estimators.
  - Once we have tested different estimators on actual data, and looked at the performance (e.g., MSE of predictions across all learners), we can see which algorithm (or weighted combination) has the lowest risk, and thus is closest to the true ψ_0.

- The *cross-validated empirical risk* of an algorithm is defined as the empirical mean over a validation sample of the loss of the algorithm fitted on the training sample, averaged across the splits of the data.

  - The *discrete Super Learner*, or *cross-validation selector*, is the algorithm in the library that minimizes the cross-validated empirical risk.
  - The *continuous/ensemble Super Learner*, often referred to as *Super Learner* is a weighted average of the library of algorithm predictions, where the weights are chosen to minimize the cross-validated empirical risk of the library. This notion of weighted combinations was introduced in Wolpert (1992) for neural networks and adapted for regressions in Breiman (1996). Restricting the weights to be positive and sum to one (i.e., a convex combination) has been shown to perform well in practice (Polley and van der Laan, 2010; van der Laan et al., 2007).

`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 in this handbook, is contained in Chapter 3. 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
)
```

A quick look at the data:

whz	tr	fracode	month	aged	sex	momage	momedu	momheight	hfiacat
0.00	Control	N05265	9	268	male	30	Primary (1-5y)	146.40	Food 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 probability 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.

```r
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
```

R6 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_lstm"
#> [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_sum"
#> [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_main`.

```
# we already instantiated a linear model learner above, no need to do it 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()`. The default method for parameter selection criterion with is set to "CV" instead of the `caret::train()` default `boot`. The summary metric used to select the optimal model is `RMSE` for continuous outcomes and `Accuracy` for categorical and binomial outcomes.

```
# 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 – I will confess: Bob Ross and I both know that 20 trees makes for a lonely forest, and I shouldn't consider it, but these are the sacrifices I have to make for this chapter to build 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"      "asset_tv"    "asset_chair"
```

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 with no internal screening 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`.

```r
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.

```r
# we did it! now we have SL predictions
sl_preds <- sl_fit$predict()
head(sl_preds)
#> [1] -0.64698 -0.76514 -0.64312 -0.68991 -0.68068 -0.66422
```

We can also obtain a summary of the results.

```r
sl_fit_summary <- sl_fit$print()
#> [1] "SuperLearner:"
#> List of 8
#>  $ glm                : chr "Lrnr_glm_TRUE"
#>  $ polspline          : chr "Lrnr_polspline_5"
#>  $ enet.5             : chr "Lrnr_glmnet_NULL_deviance_10_0.5_100_TRUE_FALSE"
#>  $ ridge              : chr "Lrnr_glmnet_NULL_deviance_10_0_100_TRUE_FALSE"
#>  $ lasso              : chr "Lrnr_glmnet_NULL_deviance_10_1_100_TRUE_FALSE"
#>  $ xgboost50          : chr "Lrnr_xgboost_50_1"
#>  $ randomforest_screen: chr "Pipeline(Lrnr_screener_importance_5->Stack)"
#>  $ lasso_screen       : chr "Pipeline(Lrnr_screener_coefs_0_NULL_2->Stack)"
#> [1] "Lrnr_solnp_TRUE_TRUE_FALSE_1e-05"
#> $pars
#>  [1] 0.055565 0.055551 0.055558 0.055564 0.055558 0.055583 0.055556 0.055573
#>  [9] 0.055555 0.055555 0.055555 0.055546 0.055553 0.055553 0.055553 0.055552
#> [17] 0.055553 0.055516
#>
#> $convergence
#> [1] 0
#>
#> $values
```

```
#> [1] 1.01 1.01
#>
#> $lagrange
#>             [,1]
#> [1,] -0.0050956
#>
#> $hessian
#>       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
#>  [1,]    1    0    0    0    0    0    0    0    0     0     0     0     0
#>  [2,]    0    1    0    0    0    0    0    0    0     0     0     0     0
#>  [3,]    0    0    1    0    0    0    0    0    0     0     0     0     0
#>  [4,]    0    0    0    1    0    0    0    0    0     0     0     0     0
#>  [5,]    0    0    0    0    1    0    0    0    0     0     0     0     0
#>  [6,]    0    0    0    0    0    1    0    0    0     0     0     0     0
#>  [7,]    0    0    0    0    0    0    1    0    0     0     0     0     0
#>  [8,]    0    0    0    0    0    0    0    1    0     0     0     0     0
#>  [9,]    0    0    0    0    0    0    0    0    1     0     0     0     0
#> [10,]    0    0    0    0    0    0    0    0    0     1     0     0     0
#> [11,]    0    0    0    0    0    0    0    0    0     0     1     0     0
#> [12,]    0    0    0    0    0    0    0    0    0     0     0     1     0
#> [13,]    0    0    0    0    0    0    0    0    0     0     0     0     1
#> [14,]    0    0    0    0    0    0    0    0    0     0     0     0     0
#> [15,]    0    0    0    0    0    0    0    0    0     0     0     0     0
#> [16,]    0    0    0    0    0    0    0    0    0     0     0     0     0
#> [17,]    0    0    0    0    0    0    0    0    0     0     0     0     0
#> [18,]    0    0    0    0    0    0    0    0    0     0     0     0     0
#>       [,14] [,15] [,16] [,17] [,18]
#>  [1,]     0     0     0     0     0
#>  [2,]     0     0     0     0     0
#>  [3,]     0     0     0     0     0
#>  [4,]     0     0     0     0     0
#>  [5,]     0     0     0     0     0
#>  [6,]     0     0     0     0     0
#>  [7,]     0     0     0     0     0
#>  [8,]     0     0     0     0     0
#>  [9,]     0     0     0     0     0
#> [10,]     0     0     0     0     0
```

```
#> [11,]     0     0     0     0     0
#> [12,]     0     0     0     0     0
#> [13,]     0     0     0     0     0
#> [14,]     1     0     0     0     0
#> [15,]     0     1     0     0     0
#> [16,]     0     0     1     0     0
#> [17,]     0     0     0     1     0
#> [18,]     0     0     0     0     1
#>
#> $ineqx0
#> NULL
#>
#> $nfuneval
#> [1] 23
#>
#> $outer.iter
#> [1] 1
#>
#> $elapsed
#> Time difference of 0.011938 secs
#>
#> $vscale
#>  [1] 1.01001 0.00001 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000
#> [10] 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000
#> [19] 1.00000 1.00000
#>
#> $coefficients
#>                            glm                     polspline
#>                       0.055565                      0.055551
#>                          enet.5                         ridge
#>                       0.055558                      0.055564
#>                          lasso                     xgboost50
#>                       0.055558                      0.055583
#>       randomforest_screen_glm randomforest_screen_polspline
#>                       0.055556                      0.055573
#>    randomforest_screen_enet.5     randomforest_screen_ridge
#>                       0.055555                      0.055555
```

```
#>     randomforest_screen_lasso randomforest_screen_xgboost50
#>                      0.055555                      0.055546
#>              lasso_screen_glm        lasso_screen_polspline
#>                      0.055553                      0.055553
#>           lasso_screen_enet.5            lasso_screen_ridge
#>                      0.055553                      0.055552
#>            lasso_screen_lasso        lasso_screen_xgboost50
#>                      0.055553                      0.055516
#>
#> $training_offset
#> [1] FALSE
#>
#> $name
#> [1] "solnp"
#>
#> [1] "Cross-validated risk:"
#>                           learner coefficients   risk       se  fold_sd
#>  1:                           glm     0.055565 1.0202 0.023955 0.067500
#>  2:                      polspline     0.055551 1.0208 0.023577 0.067921
#>  3:                         enet.5     0.055558 1.0131 0.023598 0.065732
#>  4:                          ridge     0.055564 1.0153 0.023739 0.065299
#>  5:                          lasso     0.055558 1.0130 0.023592 0.065840
#>  6:                       xgboost50     0.055583 1.1136 0.025262 0.077580
#>  7:       randomforest_screen_glm     0.055556 1.0173 0.023830 0.069913
#>  8: randomforest_screen_polspline     0.055573 1.0135 0.023814 0.069240
#>  9:    randomforest_screen_enet.5     0.055555 1.0177 0.023842 0.070142
#> 10:     randomforest_screen_ridge     0.055555 1.0176 0.023855 0.069787
#> 11:     randomforest_screen_lasso     0.055555 1.0177 0.023840 0.070155
#> 12: randomforest_screen_xgboost50     0.055546 1.1277 0.026043 0.078673
#> 13:              lasso_screen_glm     0.055553 1.0164 0.023542 0.065018
#> 14:        lasso_screen_polspline     0.055553 1.0177 0.023520 0.065566
#> 15:           lasso_screen_enet.5     0.055553 1.0163 0.023544 0.065017
#> 16:            lasso_screen_ridge     0.055552 1.0166 0.023553 0.064869
#> 17:            lasso_screen_lasso     0.055553 1.0163 0.023544 0.065020
#> 18:        lasso_screen_xgboost50     0.055516 1.1256 0.025939 0.084270
#> 19:                   SuperLearner           NA 1.0100 0.023524 0.068184
#>     fold_min_risk fold_max_risk
#>  1:       0.89442        1.1200
```

```
#>  2:        0.89892        1.1255
#>  3:        0.88839        1.1058
#>  4:        0.88559        1.1063
#>  5:        0.88842        1.1060
#>  6:        0.96019        1.2337
#>  7:        0.88579        1.1119
#>  8:        0.89370        1.1190
#>  9:        0.88593        1.1137
#> 10:        0.88620        1.1128
#> 11:        0.88593        1.1136
#> 12:        1.00223        1.2465
#> 13:        0.90204        1.1156
#> 14:        0.89742        1.1162
#> 15:        0.90184        1.1154
#> 16:        0.90120        1.1146
#> 17:        0.90183        1.1154
#> 18:        0.96251        1.2327
#> 19:        0.88036        1.1041
```

From the table of the printed SL fit, we note that the SL had a mean risk of 1.01 and that this ensemble weighted the `ranger` and `glmnet` learners highest while not weighting the `mean` learner highly.

We can also see that the `glmnet` learner had the lowest cross-validated mean risk, thus making it the cross-validated selector (or the *discrete* SL). The mean risk of the SL is calculated using all of the data, and not a separate hold-out, so the SL's mean risk that is reported here is an underestimation.

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
)
```

Let's take a look at a table summarizing the performance:

```
if (knitr::is_latex_output()) {
  CVsl %>%
    kable(format = "latex")
} else if (knitr::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_max_risk
glm	0.05555	1.0494	0.02687	0.07973	0.99301	1.1058
polspline	0.05557	1.0173	0.02354	0.06840	0.96890	1.0656
enet.5	0.05556	1.0239	0.02413	0.07188	0.97310	1.0748
ridge	0.05557	1.0271	0.02418	0.06797	0.97906	1.0752
lasso	0.05556	1.0243	0.02417	0.07238	0.97314	1.0755
xgboost50	0.05555	1.1789	0.02668	0.00524	1.17521	1.1826
randomforest_screen_glm	0.05555	1.0324	0.02392	0.05939	0.99043	1.0744
randomforest_screen_polspline	0.05556	1.0259	0.02372	0.07722	0.97129	1.0805
randomforest_screen_enet.5	0.05555	1.0284	0.02390	0.06515	0.98234	1.0745
randomforest_screen_ridge	0.05555	1.0310	0.02395	0.06113	0.98782	1.0743
randomforest_screen_lasso	0.05555	1.0285	0.02390	0.06508	0.98244	1.0745
randomforest_screen_xgboost50	0.05554	1.1721	0.02685	0.03247	1.14913	1.1950
lasso_screen_glm	0.05556	1.0257	0.02381	0.05370	0.98771	1.0636
lasso_screen_polspline	0.05556	1.0267	0.02390	0.05510	0.98771	1.0656
lasso_screen_enet.5	0.05556	1.0261	0.02384	0.05463	0.98753	1.0648
lasso_screen_ridge	0.05556	1.0255	0.02383	0.05439	0.98703	1.0639
lasso_screen_lasso	0.05556	1.0262	0.02384	0.05464	0.98753	1.0648
lasso_screen_xgboost50	0.05553	1.1519	0.02577	0.04334	1.12129	1.1826
SuperLearner	NA	1.0187	0.02366	0.06090	0.97563	1.0618

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, more risky) 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.

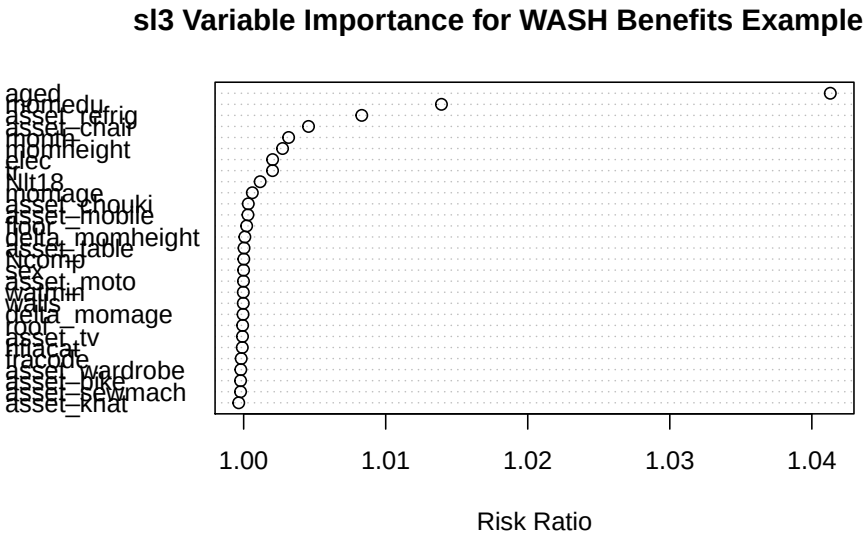
Let's explore the `sl3` variable importance measurements for the `washb` data.

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

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

X	risk_ratio
aged	1.04130
momedu	1.01392
asset_refrig	1.00831
asset_chair	1.00457
month	1.00316
momheight	1.00274
elec	1.00204
tr	1.00203
Nlt18	1.00116
momage	1.00061
asset_chouki	1.00032
asset_mobile	1.00030
floor	1.00021
delta_momheight	1.00008
asset_table	1.00003
Ncomp	1.00001
sex	1.00000
asset_moto	0.99999
watmin	0.99997
walls	0.99997
delta_momage	0.99996
roof	0.99993
asset_tv	0.99992
hfiacat	0.99990
fracode	0.99983
asset_wardrobe	0.99980
asset_bike	0.99978
asset_sewmach	0.99978
asset_khat	0.99965

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



6.1 Exercises

6.1.1 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)
```

Let's take a quick peek at the data:

waist	alcoh	hdl	beta	smoke	ace	ldl	bmi	aspirin	gend	age	estrgn	
110.164	0.0000	66.497	0	0	1	114.216	27.997	0	0	73.518	0	159
89.976	0.0000	50.065	0	0	0	103.777	20.893	0	0	61.772	0	153
106.194	8.4174	40.506	0	0	0	165.716	28.455	1	1	72.931	0	121
90.057	0.0000	36.175	0	0	0	45.203	23.961	0	0	79.119	0	53
78.614	2.9790	71.064	0	1	0	131.312	10.966	0	1	69.018	0	94
91.659	0.0000	59.496	0	0	0	171.187	29.132	0	1	81.835	0	212

1. Create an `sl3` task, setting myocardial infarction `mi` as the outcome and using all available covariate data.
2. Make a library of seven relatively fast base learning algorithms (i.e., do not consider BART or HAL). Customize hyperparameters 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. Fit the metalearning step with the default metalearner.
5. With the metalearner and base learners, make the Super Learner (SL) and train it on the task.
6. Print your SL fit by calling `print()` with `$`.
7. Cross-validate your SL fit to see how well it performs on unseen data. Specify a valid loss function to evaluate the SL.
8. Use the `importance()` function to identify the "most important" predictor of myocardial infarction, according to `sl3` importance metrics.

6.1.2 Predicting Recurrent Ischemic Stroke in an RCT with `sl3`

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 Chapter 3.2 of the `tlverse` handbook.

1. Train a SL to predict recurrent stroke `DRSISC` with the available covariate data (the 25 other variables). Of course, you can consider feature selection in the machine learning algorithms. In this data, the outcome is occasionally missing, so be sure to specify `drop_missing_outcome = TRUE` when defining the task.
2. Use the SL-based predictions to calculate the area under the ROC curve (AUC).

3. Calculate the cross-validated AUC to evaluate the performance of the SL on unseen data.
4. Which covariates are the most predictive of 14-day recurrent stroke, according to `sl3` variable importance measures?

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

# number 3 help
ist_task_CVsl <- make_sl3_Task(
  data = ist_data,
  outcome = "DRSISC",
  covariates = colnames(ist_data)[-which(names(ist_data) == "DRSISC")],
  drop_missing_outcome = TRUE,
  folds = origami::make_folds(
    n = sum(!is.na(ist_data$DRSISC)),
    fold_fun = folds_vfold,
    V = 5
  )
)
```

6.2 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: $\Psi_0 = E_{0,W}[E_0(Y|A = 1, W) - E_0(Y|A = 0, W)]$.
  - We could use the SL that was trained on the original data (let's call this `sl_fit`) to predict the outcome for all subjects under each intervention. All we would need to do is take the average difference between the counterfactual outcomes under each intervention of interest.

  - Considering Ψ_0 above, we would first need two n-length vectors of predicted outcomes under each intervention. One vector would represent the predicted outcomes under an intervention that sets all subjects to receive $A = 1$, $Y_i|A_i = 1, W_i$ for all $i = 1, \ldots, n$. The other vector would represent the predicted outcomes under an intervention that sets all subjects to receive $A = 0$, $Y_i|A_i = 0, W_i$ for all $i = 1, \ldots, n$.
  - After obtaining these vectors of counterfactual predicted outcomes, all we would need to do is average and then take the difference in order to "plug-in" the SL estimator into the target parameter mapping.
  - In `sl3` and with our current ATE example, this could be achieved with `mean(sl_fit$predict(A1_task)) - mean(sl_fit$predict(A0_task))`; where `A1_task$data` would contain all 1's (or the level that pertains to receiving the treatment) for the treatment column in the data (keeping all else the same), and `A0_task$data` would contain all 0's (or the level that pertains to not receiving the treatment) for the treatment column in the data.

- It's a worthwhile exercise to obtain the predicted counterfactual outcomes and create these counterfactual `sl3` tasks. It's too biased; however, to plug the SL fit into the target parameter mapping, (e.g., calling the result of `mean(sl_fit$predict(A1_task)) - mean(sl_fit$predict(A0_task))` the estimated ATE. We would end up with an estimator for the ATE that was optimized for estimation of the prediction function, and not the ATE!

- At the end of the "analysis day", we want an estimator that is optimized for our target estimand of interest. We ultimately care about doing a good job estimating ψ_0. The SL is an essential step to help us get there. In fact, we will use the counterfactual predicted outcomes that were explained at length above. However, SL is not the end of the estimation procedure. Specifically, the Super Learner would not be an asymptotically linear estimator of the target estimand; and it is not an efficient substitution estimator. This begs the question, why is it so important for an estimator to possess these properties?

  - An asymptotically linear estimator converges to the estimand a $\frac{1}{\sqrt{n}}$ rate, thereby permitting formal statistical inference (i.e., confidence intervals and p-values).
  - 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 the targeted minimum loss-based estimator, both referred to as TMLE.

Appendix

6.2.1 Exercise 1 Solution

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

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

# make task
```

```r
chspred_task <- make_sl3_Task(
  data = chspred,
  covariates = head(colnames(chspred), -1),
  outcome = "mi"
)

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

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

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

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

```
sl_fit$print()

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

varimp <- importance(sl_fit, type = "permute")
varimp %>%
  importance_plot(
    main = "sl3 Variable Importance for Myocardial Infarction Prediction"
  )
```

6.2.2 Exercise 2 Solution

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

```
library(ROCR) # for AUC calculation

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

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

# learner library
lrn_glm <- Lrnr_glm$new()
lrn_lasso <- Lrnr_glmnet$new(alpha = 1)
lrn_ridge <- Lrnr_glmnet$new(alpha = 0)
lrn_enet <- Lrnr_glmnet$new(alpha = 0.5)
lrn_mean <- Lrnr_mean$new()
```

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

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

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

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

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

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

Chapter 7

The TMLE Framework

Jeremy Coyle

Based on the `tmle3` `R` package.

7.1 Learning Objectives

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

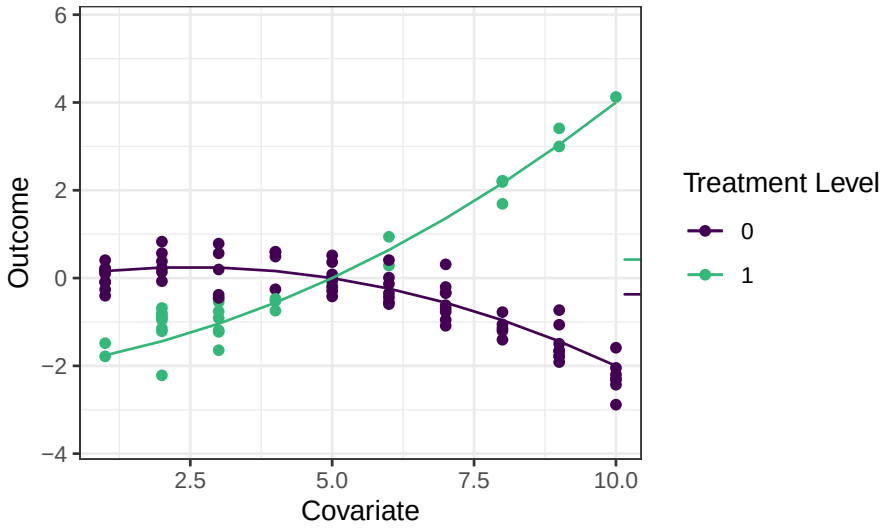
1. Understand why we use TMLE for effect estimation.
2. Use `tmle3` to estimate an Average Treatment Effect (ATE).
3. Understand how to use `tmle3` "Specs" objects.
4. Fit `tmle3` for a custom set of target parameters.
5. Use the delta method to estimate transformations of target parameters.

7.2 Introduction

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

Going back to the roadmap for targeted learning, suppose we'd like to estimate the effect of a treatment variable A on an outcome Y. As discussed, one potential

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



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

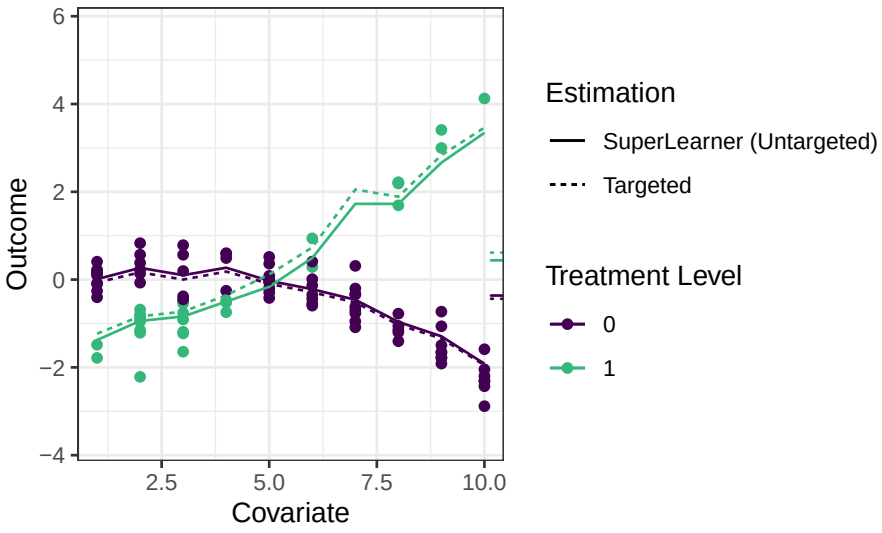
While we hope to motivate the application of TMLE in this chapter, we refer the interested reader to the two Targeted Learning books and associated works for full technical details.

7.3 Substitution Estimators

We can use `sl3` to fit a Super Learner or other regression model to estimate the outcome regression function $\mathbb{E}_0[Y \mid A, W]$, which we often refer to as $\overline{Q}_0(A, W)$ and whose estimate we denote $\overline{Q}_n(A, W)$. To construct an estimate of the ATE ψ_n, we need only "plug-in" the estimates of $\overline{Q}_n(A, W)$, evaluated at the two intervention contrasts, to the corresponding ATE "plug-in" formula: $\psi_n = \frac{1}{n}\sum(\overline{Q}_n(1, W) -$

$\overline{Q}_n(0, W)$). This kind of estimator is called a *plug-in* or *substitution* estimator, since accurate estimates ψ_n of the parameter ψ_0 may be obtained by substituting estimates $\overline{Q}_n(A, W)$ for the relevant regression functions $\overline{Q}_0(A, W)$ themselves.

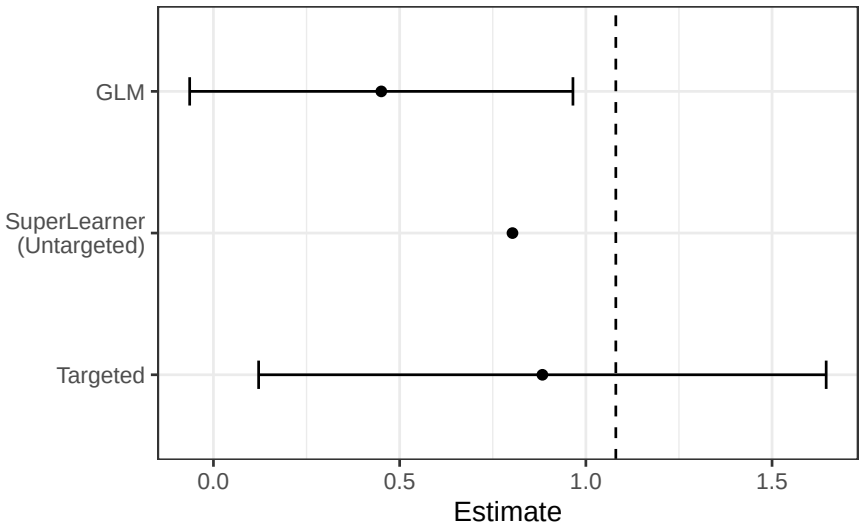
Applying `sl3` to estimate the outcome regression in our example, we can see that the ensemble machine learning predictions fit the data quite well:



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

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

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



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

7.4 Targeted Maximum Likelihood Estimation

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

7.4.1 TMLE Updates

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

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

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

7.4.2 Statistical Inference

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

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

7.5 Easy-Bake Example: `tmle3` for ATE

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

7.5.1 Load the Data

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

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

7.5.2 Define the variable roles

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

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

7.5.3 Handle Missingness

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

- Missing covariates are median- (for continuous) or mode- (for discrete) imputed, and additional covariates indicating imputation are generated, just as described in the sl3 chapter.
- Missing treatment variables are excluded – such observations are dropped.
- Missing outcomes are efficiently handled by the automatic calculation (and incorporation into estimators) of *inverse probability of censoring weights* (IPCW); this is also known as IPCW-TMLE and may be thought of as a joint intervention to remove missingness and is analogous to the procedure used with classical inverse probability weighted estimators.

These steps are implemented in the process_missing function in tmle3:

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

7.5.4 Create a "Spec" Object

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

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

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

7.5.5 Define the learners

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

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

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

# define metalearners appropriate to data types
ls_metalearner <- make_learner(Lrnr_nnls)
mn_metalearner <- make_learner(
  Lrnr_solnp, metalearner_linear_multinomial,
  loss_loglik_multinomial
)
sl_Y <- Lrnr_sl$new(
  learners = list(lrnr_mean, lrnr_rf),
  metalearner = ls_metalearner
)
sl_A <- Lrnr_sl$new(
  learners = list(lrnr_mean, lrnr_rf),
  metalearner = mn_metalearner
)
learner_list <- list(A = sl_A, Y = sl_Y)
```

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

7.5.6 Fit the TMLE

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

```
tmle_fit <- tmle3(ate_spec, washb_data, node_list, learner_list)
print(tmle_fit)
#> A tmle3_Fit that took 1 step(s)
```

```
#>    type                                      param  init_est tmle_est       se
#> 1:  ATE ATE[Y_{A=Nutrition + WSH}-Y_{A=Control}] -0.005231  0.00812 0.050679
#>        lower   upper psi_transformed lower_transformed upper_transformed
#> 1: -0.091208 0.10745         0.00812         -0.091208           0.10745
```

7.5.7 Evaluate the Estimates

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

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

7.6 tmle3 Components

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

7.6.1 tmle3_task

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

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

```
tmle_task$npsem
#> $W
#> tmle3_Node: W
#>  Variables: month, aged, sex, momedu, hfiacat, Nlt18, Ncomp, watmin, elec, floor, u
#>  Parents:
#>
```

```
#> $A
#> tmle3_Node: A
#>  Variables: tr
#>  Parents: W
#>
#> $Y
#> tmle3_Node: Y
#>  Variables: whz
#>  Parents: A, W
```

7.6.2 Initial Likelihood

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

```
initial_likelihood <- ate_spec$make_initial_likelihood(
  tmle_task,
  learner_list
)
print(initial_likelihood)
#> W: Lf_emp
#> A: LF_fit
#> Y: LF_fit
```

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

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

```
initial_likelihood$get_likelihoods(tmle_task)
#>                 W       A        Y
#>    1: 0.00021299 0.34925 -0.35834
#>    2: 0.00021299 0.36117 -0.93261
#>    3: 0.00021299 0.34740 -0.80873
```

```
#>    4: 0.00021299 0.34248 -0.94020
#>    5: 0.00021299 0.34134 -0.57866
#>   ---
#> 4691: 0.00021299 0.23375 -0.58997
#> 4692: 0.00021299 0.23366 -0.22769
#> 4693: 0.00021299 0.22660 -0.74235
#> 4694: 0.00021299 0.28944 -0.91796
#> 4695: 0.00021299 0.19533 -1.03878
```

7.6.3 Targeted Likelihood (updater)

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

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

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

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

7.6.4 Parameter Mapping

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

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

7.6.5 Putting it all together

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

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

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

7.7 Fitting `tmle3` with multiple parameters

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

```
tsm_spec <- tmle_TSM_all()
targeted_likelihood <- Targeted_Likelihood$new(initial_likelihood)
all_tsm_params <- tsm_spec$make_params(tmle_task, targeted_likelihood)
print(all_tsm_params)
#> [[1]]
#> Param_TSM: E[Y_{A=Control}]
#>
#> [[2]]
#> Param_TSM: E[Y_{A=Handwashing}]
#>
#> [[3]]
#> Param_TSM: E[Y_{A=Nutrition}]
#>
```

```
#> [[4]]
#> Param_TSM: E[Y_{A=Nutrition + WSH}]
#>
#> [[5]]
#> Param_TSM: E[Y_{A=Sanitation}]
#>
#> [[6]]
#> Param_TSM: E[Y_{A=WSH}]
#>
#> [[7]]
#> Param_TSM: E[Y_{A=Water}]
```

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

7.7.1 Delta Method

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

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

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

7.7.2 Fit

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

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

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

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

```
#> 6:          -0.35906
#> 7:          -0.46077
#> 8:           0.11145
```

7.8 Exercises

7.8.1 Estimation of the ATE with `tmle3`

Follow the steps below to estimate an average treatment effect using data from the Collaborative Perinatal Project (CPP), available in the `sl3` package. To simplify this example, we define a binary intervention variable, `parity01` – an indicator of having one or more children before the current child and a binary outcome, `haz01` – an indicator of having an above average height for age.

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

1. Define the variable roles (W, A, Y) by creating a list of these nodes. Include the following baseline covariates in W: `apgar1`, `apgar5`, `gagebrth`, `mage`, `meducyrs`, `sexn`. Both A and Y are specified above. The missingness in the data (specifically, the missingness in the columns that are specified in the node list) will need to be taking care of. The `process_missing` function can be used to accomplish this, like the `washb_data` example above.
2. Define a `tmle3_Spec` object for the ATE, `tmle_ATE()`.
3. Using the same base learning libraries defined above, specify `sl3` base learners for estimation of $\overline{Q}_0 = \mathbb{E}_0(Y \mid A, Y)$ 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.

7.8.2 Estimation of Strata-Specific ATEs with `tmle3`

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

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

1. Estimate the average effect of randomized 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"
  )
)
```

7.9 Summary

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

Chapter 8

Optimal Individualized Treatment Regimes

Ivana Malenica

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

Updated: 2021-04-04

8.1 Learning Objectives

1. Differentiate dynamic and optimal dynamic treatment interventions from static interventions.
2. Explain the benefits, and challenges, associated with using optimal individualized treatment regimes in practice.
3. Contrast the impact of implementing an optimal individualized treatment regime in the population with the impact of implementing static and dynamic treatment regimes in the population.
4. Estimate causal effects under optimal individualized treatment regimes with the `tmle3mopttx` `R` package.
5. Assess the mean under optimal individualized treatment with resource constraints.
6. Implement optimal individualized treatment rules based on sub-optimal rules, or "simple" rules, and recognize the practical benefit of these rules.

7. Construct "realistic" optimal individualized treatment regimes that respect real data and subject-matter knowledge limitations on interventions by only considering interventions that are supported by the data.
8. Measure variable importance as defined in terms of the optimal individualized treatment interventions.

8.2 Introduction to Optimal Individualized Interventions

Identifying which intervention will be effective for which patient based on lifestyle, genetic and environmental factors is a common goal in precision medicine. To put it in context, Abacavir and Tenofovir are commonly prescribed as part of the antiretroviral therapy to Human Immunodeficiency Virus (HIV) patients. However, not all individuals benefit from the two medications equally. In particular, patients with renal dysfunction might further deteriorate if prescribed Tenofovir, due to the high nephrotoxicity caused by the medication. While Tenofovir is still highly effective treatment option for HIV patients, in order to maximize the patient's well-being, it would be beneficial to prescribe Tenofovir only to individuals with healthy kidney function. Along the same lines, 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 it.

One opts to administer the intervention to individuals who will profit from it instead, instead of assigning treatment on a population level. But how do we know which intervention works for which patient? This aim motivates a different type of intervention, as opposed to the static exposures we described in previous chapters. In particular, in this chapter we learn about dynamic or "individualized" interventions that tailor the treatment decision based on the collected covariates. Formally, dynamic treatments represent interventions that at each treatment-decision stage are allowed to respond to the currently available treatment and covariate history.

In the statistics community such a treatment strategy is termed an **individualized treatment regime** (ITR), and the (counterfactual) population mean outcome under an ITR is the value of the ITR (Neyman, 1990; Robins, 1986; Pearl, 2009a). Even

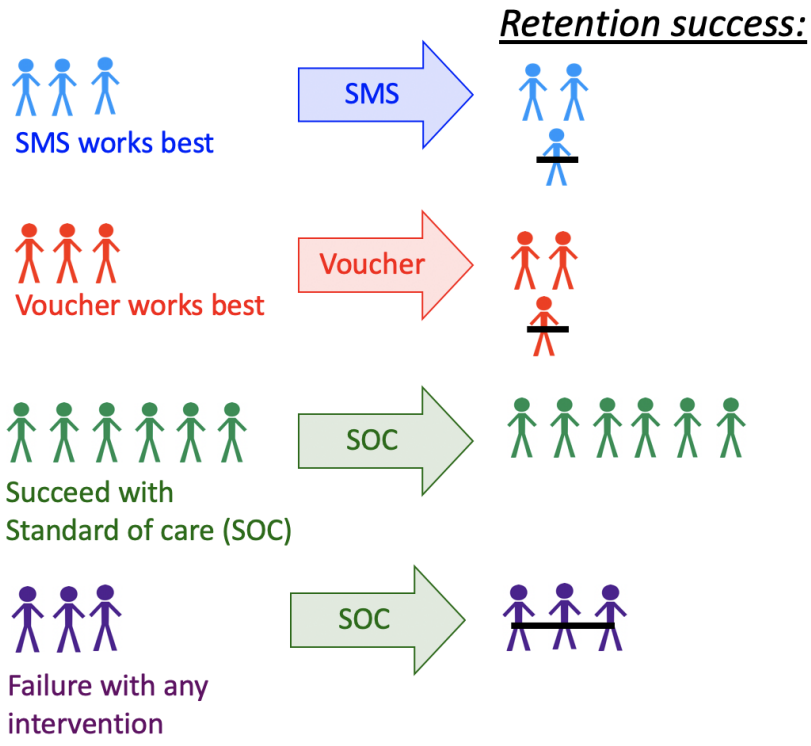


Figure 8.1: Dynamic Treatment Regime in a Clinical Setting

more, suppose one wishes to maximize the population mean of an outcome, where for each individual we have access to some set of measured covariates. This means, for example, that we can learn for which individual characteristics assigning treatment increases the probability of a beneficial outcome. An ITR with the maximal value is referred to as an optimal ITR or the **optimal individualized treatment**. Consequently, the value of an optimal ITR is termed the optimal value, or the **mean under the optimal individualized treatment**.

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

In this chapter, we consider estimation of the mean outcome under the optimal individualized treatment where the candidate rules are restricted to depend only on user-supplied subset of the baseline covariates. The estimation problem is addressed in a statistical model for the data distribution that is nonparametric, and at most places restrictions on the probability of a patient receiving treatment given covariates (as in a randomized trial). As such, we don't need to make any assumptions about the relationship of the outcome with the treatment and covariates, or the relationship between the treatment and covariates. Further, we provide a Targeted Maximum Likelihood Estimator for the mean under the optimal individualized treatment that allows us to generate valid inference for our parameter, without having any parametric assumptions. For a technical presentation of the algorithm, the interested reader is invited to further consult van der Laan and Luedtke (2015) and Luedtke and van der Laan (2016).

8.3 Data Structure and Notation

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

We use the nonparametric structural equation model (NPSEM) in order to define the process that gives rise to the observed (endogenous) and not observed (exogenous) variables, as described by Pearl (2009b). In particular, we denote $U = (U_W, U_A, U_Y)$

as the exogenous random variables, and $O = (W, A, Y)$ as endogenous variables we observe. The joint distribution of exogenous and endogenous random variables in $\mathcal{M}^F$ (defined by the NPSEM) is $P_{U,X}$. We can define the relationships between variables with the following structural equations:

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

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

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

indicating no dependence of treatment on baseline covariates.

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

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

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

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

8.4 Defining the Causal Effect of an Optimal Individualized Intervention

Consider dynamic treatment rules d in the set of all possible rules $\mathcal{D}$. Then, d is a function that takes as input V and outputs a treatment decision, $V \rightarrow d(V) \in \{a_1, \cdots, a_{n_A}\} \times \{1\}$. We will use dynamic treatment rules, and the corresponding treatment decision, to describe an intervention on the treatment mechanism and the corresponding outcome under a dynamic treatment rule.

As mentioned in the previous section, causal effects are defined in terms of hypothetical interventions on the NPSEM (8.3). We can define counterfactuals $Y_{d(V)}$ defined by a modified system in which the equation for A is replaced by the rule $d(V)$, dependent on covariates V. Our modified system then takes the following form:

$$W = f_W(U_W) \tag{8.8}$$

$$A = d(V) \tag{8.9}$$

$$Y_{d(V)} = f_Y(d(V), W, U_Y), \tag{8.10}$$

where the dynamic treatment regime may be viewed as an intervention in which A is set equal to a value based on a hypothetical regime $d(V)$, possibly contrary to the fact, and $Y_{d(V)}$ is the corresponding outcome under $d(V)$. We denote the distribution of the counterfactual quantities as $P_{0,d(V)}$.

The goal of any causal analysis motivated by such dynamic interventions is to estimate a parameter defined as the counterfactual mean of the outcome with respect to the modified intervention distribution. That is, subject's outcome if, possibly contrary to the fact, the subject received treatment that would have been assigned by rule $d(V)$. We can consider different treatment rules, all in the set $\mathcal{D}$:

1. The true rule, d_0, and the corresponding causal parameter $\mathbb{E}_{U,X}[Y_{d_0(V)}]$;
2. The estimated rule, d_n, and the corresponding causal parameter $\mathbb{E}_{U,X}[Y_{d_n(V)}]$.

In this chapter, we will focus on the estimated rule d_n, and the corresponding data-adaptive parameter.

The optimal individualized rule is the rule with the maximal value:

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

.

We note that, in case the problem at hand requires minimizing the mean of an outcome, our optimal individualized rule will be the rule with the minimal value instead. Our causal target parameter of interest is the expected outcome under the estimated optimal individualized rule:

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

8.4.1 Identification and Statistical Estimand

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

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

Under the above causal assumptions, we can identify $P_{0,d}$ with observed data using the G-computation formula:

$$P_{0,d_{n,\text{opt}}}(O) = Q_{Y,0}(Y \mid A = d_{n,\text{opt}}(V), W) g_0(A = d_{n,\text{opt}}(V) \mid W) Q_{W,0}(W).$$

The value of an individualized rule can now be expressed as

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

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

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

Inference for the optimal value has been shown to be difficult at exceptional laws, defined as probability distributions for which treatment is neither beneficial nor harmful. Inference is similarly difficult in finite samples if the treatment effect is very small in all strata, even though valid asymptotic estimators exist in this setting. With that in mind, we address the estimation problem under the assumption of non-exceptional laws in effect.

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

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

8.4.2 Binary treatment

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

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

or the average treatment effect within a stratum of V. The note that the optimal individualized rule can now be derived as $d_{n,\text{opt}}(V) = \mathbb{I}(\bar{Q}_0(V) > 0)$.

The package `tmle3mopttx` relies on using the Super Learner to estimate the blip function, as it easily extends to more general categorical treatment. With that in mind, the loss function utilized for learning the optimal individualized rule corresponds to conditional mean type losses. It is however worth mentioning that Luedtke and van der Laan (2016) present three different approaches for learning the optimal rule. Namely, they focus on:

1. Super Learning the Blip Function,
2. Super Learning the Weighted Classification Problem,

3. Joint Super Learner of the Blip and Weighted Classification Problem.

We refer the interested reader to Luedtke and van der Laan (2016) for further reference on advantages of each approach.

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

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

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

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

8.4.3 Categorical treatment

In line with the approach considered for binary treatment, we extend the blip function to allow for categorical treatment. We denote such blip function extensions as *pseudo-blips*, which are our new estimation targets in a categorical setting. We define pseudo-blips as vector-valued entities where the output for a given V is a vector of length equal to the number of treatment categories, n_A. As such, we define it as:
$$\bar{Q}_0^{pblip}(V) = \{\bar{Q}_{0,a}^{pblip}(V) : a \in \mathcal{A}\}$$

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

1. *Blip1* corresponds to choosing a reference category of treatment, and defining the blip for all other categories relative to the specified reference. Hence we have that:
$$\bar{Q}_{0,a}^{pblip-ref}(V) \equiv \mathbb{E}_0(Y_a - Y_0 \mid V)$$
where Y_0 is the specified reference category with $A = 0$. Note that, for the case of binary treatment, this strategy reduces to the approach described for the binary setup.

2. *Blip2* approach corresponds to defining the blip relative to the average of all categories. As such, we can define $\bar{Q}_{0,a}^{pblip-avg}(V)$ as:
$$\bar{Q}_{0,a}^{pblip-avg}(V) \equiv \mathbb{E}_0(Y_a - \frac{1}{n_A} \sum_{a \in \mathcal{A}} Y_a \mid V)$$
In the case where subject-matter knowledge regarding which reference category to use is not available, blip2 might be a viable option.

3. *Blip3* reflects an extension of Blip2, where the average is now a weighted average:
$$\bar{Q}_{0,a}^{pblip-wavg}(V) \equiv \mathbb{E}_0(Y_a - \frac{1}{n_A} \sum_{a \in \mathcal{A}} Y_a P(A = a \mid V) \mid V)$$

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

8.4.4 Note on Inference and data-adaptive parameter

In a randomized trial, statistical inference relies on the second-order difference between the estimator of the optimal individualized treatment and the optimal individualized treatment itself to be asymptotically negligible. This is a reasonable condition if we consider rules that depend on a small number of covariates, or if we are willing to make smoothness assumptions. Alternatively, we can consider TMLEs and statistical inference for data-adaptive target parameters defined in terms of an estimate of the optimal individualized treatment. In particular, instead of trying to estimate the mean under the true optimal individualized treatment, we aim to estimate the mean under the estimated optimal individualized treatment. As such, we develop cross-validated TMLE approach that provides asymptotic inference under minimal conditions for the mean under the estimate of the optimal individualized

treatment. In particular, considering the data adaptive parameter allows us to avoid consistency and rate condition for the fitted optimal rule, as required for asymptotic linearity of the TMLE of the mean under the actual, true optimal rule. Practically, the estimated (data-adaptive) rule should be preferred, as this possibly sub-optimal rule is the one implemented in the population.

8.4.5 Why CV-TMLE?

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

8.5 Interpreting the Causal Effect of an Optimal Individualized Intervention

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

8.6 Evaluating the Causal Effect of an OIT with Binary Treatment

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

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

8.6.1 Simulated Data

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

$$W \sim \mathcal{N}(\mathbf{0}, \mathbf{I_{3\times 3}})$$
$$P(A = 1|W) = \frac{1}{1 + \exp^{(-0.8*W_1)}}$$
$$P(Y = 1|A, W) = 0.5\text{logit}^{-1}[-5I(A = 1)(W_1 - 0.5) + 5I(A = 0)(W_1 - 0.5)] + 0.5\text{logit}^{-1}(W_2W_3)$$

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

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

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

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

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

8.6.2 Constructing Optimal Stacked Regressions with `sl3`

To easily incorporate ensemble machine learning into the estimation procedure, we rely on the facilities provided in the `sl3` R package. Using the framework provided by the `sl3` package, the nuisance parameters of the TML estimator may be fit with ensemble learning, using the cross-validation framework of the Super Learner algorithm of van der Laan et al. (2007).

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

## 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()
)
```

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

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

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

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

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

```
# 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
)
```

As seen above, the `tmle3_mopttx_blip_revere` specification object (like all `tmle3_Spec` objects) does *not* store the data for our specific analysis of interest.

Later, we'll see that passing a data object directly to the `tmle3` wrapper function, alongside the instantiated `tmle_spec`, will serve to construct a `tmle3_Task` object internally.

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

```
# fit the TML estimator
fit <- tmle3(tmle_spec, data, node_list, learner_list)
fit
#> A tmle3_Fit that took 1 step(s)
#>    type         param init_est tmle_est       se   lower   upper
#> 1:  TSM E[Y_{A=NULL}]  0.42223  0.56606 0.027015 0.51311 0.61901
#>    psi_transformed lower_transformed upper_transformed
#> 1:         0.56606           0.51311           0.61901
```

We can see that the estimate of psi_0 is 0.56, and that the confidence interval covers our true mean under the true optimal individualized treatment.

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

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

8.7.1 Simulated Data

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

$$P(Y = 1|A, W) = 0.5\text{logit}^{-1}[15I(A = 1)(W_1 - 0.5) - 3I(A = 2)(2W_1 + 0.5) + 3I(A = 3)(3W_1 - 0.$$

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 $\psi = 0.658$, 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"
)
```

8.7.2 Constructing Optimal Stacked Regressions with `sl3`

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

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

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

As seen above, we generate three different ensemble learners that must be fit, corre-

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

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

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

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

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

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

```
# initialize a tmle specification
tmle_spec <- 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
#> 1:  TSM E[Y_{A=NULL}]    0.551  0.61063 0.065332 0.48258 0.73867
#>    psi_transformed lower_transformed upper_transformed
#> 1:         0.61063           0.48258           0.73867
```

We can see that the estimate of psi_0 is 0.60, and that the confidence interval covers our true mean under the true optimal individualized treatment.

8.8 Extensions to Causal Effect of an OIT

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

8.8.1 Simpler Rules

In order to not only consider the most ambitious fully V-optimal rule, we define S-optimal rules as the optimal rule that considers all possible subsets of V covariates, with $\text{card}(S) \leq \text{card}(V)$ 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. Within the `tmle3mopttx` paradigm, we just need to change the `complex` parameter to `FALSE`:

```
# 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_{A=W3,W2,W1}]  0.54309  0.61946 0.070471 0.48134 0.75758
#>    psi_transformed lower_transformed upper_transformed
#> 1:         0.61946           0.48134           0.75758
```

Therefore even though the user specified all baseline covariates as the basis for rule estimation, a simpler rule based on only W_2 and W_1 is sufficient to maximize the mean under the optimal individualized treatment.

8.8.2 Realistic Optimal Individual Regimes

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

```r
# 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
)
```

```r
# 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}]  0.54847  0.65455 0.021603 0.61221 0.69689
#>    psi_transformed lower_transformed upper_transformed
#> 1:         0.65455           0.61221           0.69689

# How many individuals got assigned each treatment?
table(tmle_spec$return_rule)
#>
#>   2   3
#> 507 493
```

8.8.3 Q-learning

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

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

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

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

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

# Estimate the parameter:
Q_learning(tmle_spec_Q, initial_likelihood, tmle_task)[1]
#> [1] 0.47964
```

8.9 Variable Importance Analysis with OIT

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

8.9.1 Simulated Data

In order to run `tmle3mopttx` variable importance measure, we need to consider covariates to be categorical variables. For illustration purpose, we bin baseline covariates corresponding to the data-generating distribution described previously:

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

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.

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

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

To start, we will initialize a specification for the TMLE of our parameter of interest (called a `tmle3_Spec` in the `tlverse` nomenclature) simply by calling `tmle3_mopttx_vim`. First, we indicate the method used for learning the optimal individualized treatment by specifying the `method` argument of `tmle3_mopttx_vim`. If `method="Q"`, then we will be using Q-learning for rule estimation, and we do not need to specify `V`, `type` and `learners` arguments in the spec, since they are not important for Q-learning. However, if `method="SL"`, which corresponds to learning the optimal individualized treatment using the above outlined methodology, then

we need to specify the type of pseudo-blip we will use in this estimation problem, whether we want to maximize or minimize the outcome, complex and realistic rules. Finally, for `method="SL"` we also need to communicate that we're interested in learning a rule dependent on `V` covariates by specifying the `V` argument. For both `method="Q"` and `method="SL"`, we need to indicate whether we want to maximize or minimize the mean under the optimal individualized rule. Finally, we also need to specify whether the final comparison of the mean under the optimal individualized rule and the mean under the observed outcome should be on the multiplicative scale (risk ratio) or linear (similar to average treatment effect).

```
# initialize a tmle specification
tmle_spec <- 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
#> 1:   RR RR(E[Y_{A=NULL}]/E[Y]) -0.02858893 -0.145989 0.054757 -0.253311
#> 2:   RR RR(E[Y_{A=NULL}]/E[Y])  0.00035671  0.088742 0.033090  0.023886
#>        upper psi_transformed lower_transformed upper_transformed  A
#> 1: -0.038667         0.86417           0.77623           0.96207 W1  W3,W4,W2
#> 2:  0.153598         1.09280           1.02417           1.16602  A W3,W4,W2,
#>     Z_stat      p_nz p_nz_corrected
#> 1: -2.6661 0.0038367      0.0038367
#> 2:  2.6818 0.0036614      0.0038367
```

The final result of `tmle3_vim` with the `tmle3mopttx` spec is an ordered list of mean

outcomes under the optimal individualized treatment for all categorical covariates in our dataset.

8.10 Exercises

8.10.1 Real World Data and `tmle3mopttx`

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

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

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

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

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

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

Questions:

1. Define V as mother's education (`momedu`), current living conditions (`floor`), and possession of material items including the refrigerator (`asset_refrig`). Why do you think we use these covariates as V? Do we want to minimize or maximize the outcome? Which blip type should we use?

2. Load the WASH Benefits data, and define the appropriate nodes for treatment and outcome. Use all the rest of the covariates as W except for `momheight` for now. Construct an appropriate `sl3` library for A, Y and B.

3. Based on the V defined in the previous question, estimate the mean under the ITR for the main randomized intervention used in the WASH Benefits trial with weight-for-height Z-score as the outcome. What's the TMLE value of the optimal ITR? How does it change from the initial estimate? Which intervention is the most dominant? Why do you think that is?

4. Using the same formulation as in questions 1 and 2, estimate the realistic optimal ITR and the corresponding value of the realistic ITR. Did the results change? Which intervention is the most dominant under realistic rules? Why do you think that is?

5. Consider simpler rules for the WASH benefits data example. What set of rules are picked?

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

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

8.10.2 Review of Key Concepts

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

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

3. Look back at the results generated in the section on categorical treatments, and compare then to the mean under the optimal individualized treatment in the section on binary treatments. Why do you think the estimate is higher under the less complex rule? How does the set of covariates picked by `tmle3mopttx` compare to the baseline covariates the true rule depends on?

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

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

8.10.3 Advanced Topics

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

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

Chapter 9

Stochastic Treatment Regimes

Nima Hejazi

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

Updated: 2021-04-04

9.1 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.

9.2 Introduction to Stochastic Interventions

Stochastic treatment regimes present a relatively simple, yet extremely flexible manner by which *realistic* causal effects (and contrasts thereof) may be defined. Importantly, stochastic treatment regimes 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. In this chapter, 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. In later sections, we introduce numerous extensions based on this broad class of interventions – from stochastic interventions on binary treatment variables to stochastic mediation effects and data-adaptive inference for stochastic intervention effects. 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 $d(A, W)$. For a comprehensive technical presentation of some of the material in this chapter, the interested reader is invited to consult Díaz and van der Laan (2018). Additional background on the field of Targeted Learning, as well as prior work on stochastic treatment regimes, is available in van der Laan and Rose (2011), van der Laan and Rose (2018), and Díaz and van der Laan (2012).

While stochastic treatment regimes are arguably the most general of the classes of interventions through which causal effects may be defined, such interventions are conceptually simple.

9.3 Data Structure and Notation

Consider n observed units $O_1, \ldots, O_n$, where each random variable $O = (W, A, Y)$ corresponds to a single observational unit. Let W denote baseline covariates (e.g., age, sex, education level), A an intervention variable of interest (e.g., nutritional

supplements), and Y an outcome of interest (e.g., disease status). Though it need not be the case, let A be continuous-valued, i.e. $A \in \mathbb{R}$. Let $O_i \sim \mathcal{P} \in \mathcal{M}$, where $\mathcal{M}$ is the nonparametric statistical model defined as the set of continuous densities on O with respect to some dominating measure. To formalize the definition of stochastic interventions and their corresponding causal effects, we introduce a nonparametric structural equation model (NPSEM), based on Pearl (2009b), to define how the system changes under posited interventions:

$$W = f_W(U_W) \tag{9.1}$$
$$A = f_A(W, U_A) \tag{9.2}$$
$$Y = f_Y(A, W, U_Y), \tag{9.3}$$

where the set of structural equations provide a mechanistic model by which the observed data O is assumed to have been generated. There are several standard assumptions embedded in the NPSEM – specifically, a temporal ordering that supposes that Y occurs after A, which occurs after W; each variable (i.e., $\{W, A, Y\}$) is assumed to have been generated from its corresponding deterministic function (i.e., $\{f_W, f_A, f_Y\}$) of the observed variables that precede it temporally, as well as an exogenous variable, denoted by U; lastly, each exogenous variable is assumed to contain all unobserved causes of the corresponding observed variable.

The likelihood of the data O admits a factorization, wherein, for p_0^O, the density of O with respect to the product measure, the density evaluated on a particular observation o may be a written

$$p_0^O(x) = q_{0,Y}^O(y \mid A = a, W = w) q_{0,A}^O(a \mid W = w) q_{0,W}^O(w), \tag{9.4}$$

where $q_{0,Y}$ is the conditional density of Y given (A, W) with respect to some dominating measure, $q_{0,A}$ is the conditional density of A given W with respect to dominating measure μ, and $q_{0,W}$ is the density of W with respect to dominating measure ν. Further, for ease of notation, let $Q(A, W) = \mathbb{E}[Y \mid A, W]$, $g(A \mid W) = \mathbb{P}(A \mid W)$, and q_W the marginal distribution of W. These components of the likelihood will be essential in developing an understanding of the manner in which stochastic treatment regimes pertrub a system and how a corresponding causal effect may be evaluated. Importantly, the NPSEM parameterizes p_0^O in terms of the distribution of random variables (O, U) modeled by the system of equations. In turn, this implies a model for the distribution of counterfactual random variables generated by interventions on the data-generating process.

9.4 Defining the Causal Effect of a Stochastic Intervention

As causal effects are defined in terms of hypothetical interventions on the NPSEM (9.3), we may consider stochastic interventions in two equivalent ways: (1) where the equation f_A, giving rise to A, is replaced by a probabilistic mechanism $g_\delta(A \mid W)$ that differs from the original $g(A \mid W)$, or (2) where 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 the former case, 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 $A_\delta \sim g_\delta(\cdot \mid W)$. Alternatively, in the latter 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. In either case, one may view the stochastic intervention as generating 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 the counterfactual outcome $Y_{d(A,W)} \sim \mathcal{P}_0^\delta$.

Stochastic interventions of this second variety may be referred to as depending on the *natural value of treatment* or as *modified treatment policies*. Haneuse and Rotnitzky (2013) and Young et al. (2014) provide a discussion of the critical differences and similarities in the identification and interpretation of these two classes of stochastic intervention. In the sequel, we will restrict our attention to a simple stochastic treatment regime that has been characterized as a *modified treatment policy* (MTP). Letting A denote a continuous-valued treatment, such as the taking of nutritional supplements (e.g., number of vitamin pills) and assume that the distribution of A conditional on $W = w$ has support in the interval $(l(w), u(w))$. That is, the minimum observed number of pills taken A for an individual with covariates $W = w$ is $l(w)$; similarly, the maximum is $u(w)$. Then, a simple stochastic intervention, based on a shift δ, may be defined

$$d(a, w) = \begin{cases} a - \delta & \text{if } a > l(w) + \delta \\ a & \text{if } a \leq l(w) + \delta, \end{cases} \tag{9.5}$$

where $0 \leq \delta \leq u(w)$ is an arbitrary pre-specified value that defines the degree to which the observed value A is to be shifted, where possible. Such a stochastic treatment regime may be interpreted as the result of a clinic policy that encourages individuals to consume δ more vitamin pills than they would normally, i.e., based on their baseline characteristics. The interpretation of this stochastic intervention may

be made more interesting by allowing the modification δ that it engenders to be a function of the baseline covariates W, thereby allowing for the number of vitamin pills taken to be a function of covariates such as age, sex, comorbidities, etc. This class of stochastic interventions was first introduced by Díaz and van der Laan (2012) and has been further discussed in Haneuse and Rotnitzky (2013), Díaz and van der Laan (2018), and Hejazi et al. (2020c). Note that this intervention may be written in a manner consistent with the first class of stochastic treatment regimes discussed as well – that is, as per Díaz and van der Laan (2012), $\mathbb{P}_\delta(g_0)(A = a \mid W) = g_0(a - \delta(W) \mid W)$.

The goal of any causal analysis motivated by such a stochastic intervention is to estimate a parameter defined as the counterfactual mean of the outcome with respect to the stochastically modified intervention distribution. In particular, 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 $Y_{d(A,W)}$. 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:

$$\psi_{0,d} = \int_{\mathcal{W}} \int_{\mathcal{A}} \mathbb{E}_{P_0}\{Y \mid A = d(a,w), W = w\} \cdot \tag{9.6}$$

$$q^O_{0,A}(a \mid W = w) \cdot q^O_{0,W}(w) d\mu(a) d\nu(w). \tag{9.7}$$

If the identification conditions may be assumed to hold, then the statistical parameter in (9.7) matches exactly the counterfactual outcome $\psi_{0,\delta}$ under such an intervention, allowing for the causal effect to be learned from the observed data O. Díaz and van der Laan (2012) provide a derivation based on the efficient influence function (EIF) in the nonparametric model $\mathcal{M}$ and develop several estimators of this quantity, including substitution, inverse probability weighted (IPW), one-step (OS) and targeted maximum likelihood (TML) estimators, allowing for semiparametric-efficient estimation and inference on the quantity of interest. As per Díaz and van der Laan (2018), 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.

Although the focus of this work is neither the establishment of identification results nor the development of theoretical details, we review the necessary identification details for the counterfactual mean under a stochastic intervention here, in the interest of completeness. Paraphrasing from Díaz and van der Laan (2012) and Díaz and van der Laan (2018), four standard assumptions are necessary in order to establish identifiability of the causal parameter from the observed data via the statistical functional – these are

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

$$D(P_0)(x) = H(a, w)(y - \overline{Q}(a, w)) + \overline{Q}(d(a, w), w) - \Psi(P_0), \tag{9.8}$$

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

$$H(a, w) = \mathbb{I}(a + \delta < u(w)) \frac{g_0(a - \delta \mid w)}{g_0(a \mid w)} + \mathbb{I}(a + \delta \geq u(w)), \tag{9.9}$$

which may be reduced to

$$H(a, w) = \frac{g_0(a - \delta \mid w)}{g_0(a \mid w)} + 1 \tag{9.10}$$

in the case that the treatment is within the limits that arise from conditioning on W, i.e., for $A_i \in (u(w) - \delta, u(w))$.

The efficient influence function allows the construction of a semiparametric-efficient estimators may be constructed. In the sequel, we focus on a targeted maximum likelihood (TML) estimator, for which Díaz and van der Laan (2018) give a recipe:

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
$$\text{logit}Q_{\epsilon,n}(a, w) = \text{logit}Q_n(a, w) + \epsilon H_n(a, w),$$
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 $\overline{Q}_{n,\epsilon_n}$:
$$\Psi_n = \Psi(P_n^\star) = \frac{1}{n} \sum_{i=1}^{n} \overline{Q}_n^\star(d(A_i, W_i), W_i). \tag{9.11}$$

9.5 Evaluating the Causal Effect of a Stochastic Intervention

To start, let's load the packages we'll be using throughout our simple data example

```
library(data.table)
library(haldensify)
library(sl3)
library(tmle3)
library(tmle3shift)
```

We need to estimate two components of the likelihood in order to construct a TML estimator. The first of these components is the outcome regression, $\hat{Q}_n$, which is a simple regression of the form $\mathbb{E}[Y \mid A, W]$. An estimate for such a quantity may be constructed using the Super Learner algorithm. We construct the components of an `sl3`-style Super Learner for a regression below, using a small variety of parametric and nonparametric regression techniques:

```
# learners used for conditional mean of the outcome
mean_lrnr <- Lrnr_mean$new()
fglm_lrnr <- Lrnr_glm_fast$new()
rf_lrnr <- Lrnr_ranger$new()
hal_lrnr <- Lrnr_hal9001$new(max_degree = 3, n_folds = 3)

# SL for the outcome regression
sl_reg_lrnr <- Lrnr_sl$new(
  learners = list(mean_lrnr, fglm_lrnr, rf_lrnr, hal_lrnr),
  metalearner = Lrnr_nnls$new()
)
```

The second of these is an estimate of the treatment mechanism, $\hat{g}_n$, 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. (2020a), and semiparametric location-scale conditional density estimators implemented in the `sl3` package. A Super Learner may be constructed by pooling estimates from each of these modified conditional density regression techniques.

```
# learners used for conditional densities (i.e., generalized propensity score)
haldensify_lrnr <- 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_hal_lrnr <- make_learner(Lrnr_density_semiparametric,
  mean_learner = hal_lrnr
)
# semiparametric density estimator based on heteroscedastic errors (HESE)
hese_rf_glm_lrnr <- make_learner(Lrnr_density_semiparametric,
  mean_learner = rf_lrnr,
  var_learner = fglm_lrnr
)

# SL for the conditional treatment density
sl_dens_lrnr <- Lrnr_sl$new(
  learners = list(haldensify_lrnr, hose_hal_lrnr, hese_rf_glm_lrnr),
  metalearner = Lrnr_solnp_density$new()
)
```

Finally, we construct a `learner_list` object for use in constructing a TML estimator of our target parameter of interest:

```
learner_list <- list(Y = sl_reg_lrnr, A = sl_dens_lrnr)
```

The `learner_list` object above specifies the role that each of the ensemble learners we have generated is to play in computing initial estimators to be used in building a TMLE for the parameter of interest here. In particular, it makes explicit the fact that our `Q_learner` is used in fitting the outcome regression while our `g_learner` is used in estimating the treatment mechanism.

9.5.1 Example with Simulated Data

```
# simulate simple data for tmle-shift sketch
n_obs <- 400 # number of observations
tx_mult <- 2 # multiplier for the effect of W = 1 on the treatment

## baseline covariates -- simple, binary
W <- replicate(2, rbinom(n_obs, 1, 0.5))

## create treatment based on baseline W
A <- rnorm(n_obs, mean = tx_mult * W, sd = 1)

## create outcome as a linear function of A, W + white noise
Y <- rbinom(n_obs, 1, prob = plogis(A + W))

# organize data and nodes for tmle3
data <- data.table(W, A, Y)
setnames(data, c("W1", "W2", "A", "Y"))
node_list <- list(
  W = c("W1", "W2"),
  A = "A",
  Y = "Y"
)
head(data)
#>    W1 W2         A Y
#> 1:  1  1  0.271651 1
#> 2:  0  0 -0.663368 1
#> 3:  0  0  0.113366 0
```

```
#> 4:  0  1 -0.732558 0
#> 5:  1  1  0.388835 1
#> 6:  0  0  0.043986 0
```

The above composes our observed data structure $O = (W, A, Y)$. To formally express this fact using the `tlverse` grammar introduced by the `tmle3` package, we create a single data object and specify the functional relationships between the nodes in the *directed acyclic graph* (DAG) via *nonparametric structural equation models* (NPSEMs), reflected in the node list that we set up:

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

To start, we will initialize a specification for the TMLE of our parameter of interest (called a `tmle3_Spec` in the `tlverse` nomenclature) simply by calling `tmle_shift`. We specify the argument `shift_val = 0.5` when initializing the `tmle3_Spec` object to communicate that we're interested in a shift of 0.5 on the scale of the treatment A – that is, we specify $\delta = 0.5$ (note that this is an arbitrarily chosen value for this example).

```
# initialize a tmle specification
tmle_spec <- tmle_shift(
  shift_val = 0.5,
  shift_fxn = shift_additive,
  shift_fxn_inv = shift_additive_inv
)
```

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

9.5.2 Targeted Estimation of Stochastic Interventions Effects

```
tmle_fit <- tmle3(tmle_spec, data, node_list, learner_list)
#>
#> Iter: 1 fn: 534.2313  Pars:  0.43334 0.38684 0.17982
#> Iter: 2 fn: 534.2312  Pars:  0.43334 0.38684 0.17982
#> 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}]  0.76199   0.7626 0.021965 0.71955 0.80565
#>    psi_transformed lower_transformed upper_transformed
#> 1:          0.7626           0.71955           0.80565
```

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

9.5.3 Statistical Inference for Targeted Maximum Likelihood Estimates

Recall that the asymptotic distribution of TML estimators has been studied thoroughly:

$$\psi_n - \psi_0 = (P_n - P_0) \cdot D(\bar{Q}_n^\star, g_n) + R(\hat{P}^\star, P_0),$$

which, provided the following two conditions,

1. If $D(\bar{Q}_n^\star, g_n)$ converges to $D(P_0)$ in $L_2(P_0)$ norm, and
2. the size of the class of functions considered for estimation of $\bar{Q}_n^\star$ and g_n is bounded (technically, $\exists \mathcal{F}$ such that $D(\bar{Q}_n^\star, g_n) \in \mathcal{F}$ *whp*, where $\mathcal{F}$ is a Donsker class), readily admits the conclusion that $\psi_n - \psi_0 = (P_n - P_0) \cdot D(P_0) + R(\hat{P}^\star, P_0)$.

Under the additional condition that the remainder term $R(\hat{P}^\star, P_0)$ decays as $o_P\left(\frac{1}{\sqrt{n}}\right)$, we have that

$$\psi_n - \psi_0 = (P_n - P_0) \cdot D(P_0) + o_P\left(\frac{1}{\sqrt{n}}\right),$$

which, by a central limit theorem, establishes a Gaussian limiting distribution for the estimator:

$$\sqrt{n}(\psi_n - \psi) \to N(0, V(D(P_0))),$$

where $V(D(P_0))$ is the variance of the efficient influence curve (canonical gradient) when ψ admits an asymptotically linear representation.

The above implies that ψ_n is a $\sqrt{n}$-consistent estimator of ψ, that it is asymptotically normal (as given above), and that it is locally efficient. This allows us to build Wald-type confidence intervals in a straightforward manner:

$$\psi_n \pm z_\alpha \cdot \frac{\sigma_n}{\sqrt{n}},$$

where σ_n^2 is an estimator of $V(D(P_0))$. The estimator σ_n^2 may be obtained using the bootstrap or computed directly via the following

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

Having now re-examined these facts, let's simply examine the results of computing our TML estimator:

9.6 Extensions: Variable Importance Analysis with Stochastic Interventions

9.6.1 Defining a grid of counterfactual interventions

In order to specify a *grid* of shifts δ to be used in defining a set of stochastic intervention policies in an *a priori* manner, let us 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. Then, to ascertain whether a given choice of the shift δ is admissable (in the sense of avoiding violations of the positivity assumption), let there be a bound $C(\delta) = \frac{g^\star(a|w)}{g(a|w)} < M$, where $g^\star(a \mid w)$ is a function of δ in part, and M is a potentially user-specified upper bound of $C(\delta)$. Then, $C(\delta)$ is a measure of the influence of a given observation, thereby providing a way to limit the maximum influence of a given observation (by way of the bound M placed on $C(\delta)$) through a choice of the shift δ.

We formalize and extend the procedure to determine an acceptable set of values for the shift δ in the sequel. Specifically, let there be a shift $d(A, W) = A + \delta(A, W)$,

where the shift $\delta(A, W)$ is defined as

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

where

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

and

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

The above provides a strategy for implementing a shift at the level of a given observation (a_i, w_i), thereby allowing for all observations to be shifted to an appropriate value – whether $\delta_{\min}$, δ, or $\delta_{\max}$.

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:

$$\delta(a, w) = \begin{cases} \delta, & C(\delta) \leq M \\ 0, \text{otherwise} \end{cases}, \tag{9.13}$$

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.

9.6.2 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 (note that the numerical choice of shift is an arbitrarily chosen set of values for this example).

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

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

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

9.6.3 Targeted Estimation of Stochastic Interventions Effects

One may walk through the step-by-step procedure for fitting the TML estimator of the mean counterfactual outcome under each shift in the grid, using the machinery exposed by the `tmle3` R package.

One may invoke the `tmle3` wrapper function (a user-facing convenience utility) to fit the series of TML estimators (one for each parameter defined by the grid delta) in a single function call:

```
tmle_fit <- tmle3(tmle_spec, data, node_list, learner_list)
#>
#> Iter: 1 fn: 534.0196  Pars:  0.40783 0.35788 0.23429
#> Iter: 2 fn: 534.0196  Pars:  0.40783 0.35787 0.23430
#> 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}]  0.55351  0.56184 0.0226250 0.51749 0.60618
#> 2:       TSM  E[Y_{A=NULL}]  0.69755  0.69748 0.0229975 0.65240 0.74255
#> 3:       TSM  E[Y_{A=NULL}]  0.82085  0.80031 0.0183466 0.76435 0.83627
```

```
#> 4: MSM_linear MSM(intercept)  0.69063  0.68654 0.0198658 0.64761 0.72548
#> 5: MSM_linear     MSM(slope)  0.13367  0.11924 0.0091106 0.10138 0.13709
#>    psi_transformed lower_transformed upper_transformed
#> 1:         0.56184           0.51749           0.60618
#> 2:         0.69748           0.65240           0.74255
#> 3:         0.80031           0.76435           0.83627
#> 4:         0.68654           0.64761           0.72548
#> 5:         0.11924           0.10138           0.13709
```

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

9.6.4 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 $\beta(\vec{\psi}_\delta) = \phi((\psi_\delta : \delta))$.

For a given MSM $m_\beta(\delta)$, we have that

$$\beta_0 = \text{argmin}_\beta \sum_\delta (\psi_\delta(P_0) - m_\beta(\delta))^2 h(\delta),$$

which is the solution to

$$u(\beta, (\psi_\delta : \delta)) = \sum_\delta h(\delta) \left(\psi_\delta(P_0) - m_\beta(\delta)\right) \frac{d}{d\beta} m_\beta(\delta) = 0.$$

This then leads to the following expansion

$$\beta(\vec{\psi}_n) - \beta(\vec{\psi}_0) \approx -\frac{d}{d\beta} u(\beta_0, \vec{\psi}_0)^{-1} \frac{d}{d\psi} u(\beta_0, \psi_0)(\vec{\psi}_n - \vec{\psi}_0),$$

where we have

$$\frac{d}{d\beta} u(\beta, \psi) = -\sum_\delta h(\delta) \frac{d}{d\beta} m_\beta(\delta)^t \frac{d}{d\beta} m_\beta(\delta) - \sum_\delta h(\delta) m_\beta(\delta) \frac{d^2}{d\beta^2} m_\beta(\delta),$$

which, in the case of an MSM that is a linear model (since $\frac{d^2}{d\beta^2} m_\beta(\delta) = 0$), reduces simply to

$$\frac{d}{d\beta} u(\beta, \psi) = -\sum_\delta h(\delta) \frac{d}{d\beta} m_\beta(\delta)^t \frac{d}{d\beta} m_\beta(\delta),$$

and

$$\frac{d}{d\psi} u(\beta, \psi)(\psi_n - \psi_0) = \sum_\delta h(\delta) \frac{d}{d\beta} m_\beta(\delta)(\psi_n - \psi_0)(\delta),$$

which we may write in terms of the efficient influence function (EIF) of ψ by using the first order approximation $(\psi_n - \psi_0)(\delta) = \frac{1}{n} \sum_{i=1}^n \text{EIF}_{\psi_\delta}(O_i)$, where EIF_{ψ_δ} is the efficient influence function (EIF) of $\vec{\psi}$.

Now, say, $\vec{\psi} = (\psi(\delta) : \delta)$ is d-dimensional, then we may write the efficient influence function of the MSM parameter β as follows

$$\text{EIF}_\beta(O) = \left(\sum_\delta h(\delta) \frac{d}{d\beta} m_\beta(\delta) \frac{d}{d\beta} m_\beta(\delta)^t \right)^{-1} \cdot \sum_\delta h(\delta) \frac{d}{d\beta} m_\beta(\delta) \text{EIF}_{\psi_\delta}(O),$$

where the first term is of dimension $d \times d$ and the second term is of dimension $d \times 1$. In the above, we assume a linear working MSM; however, an analogous procedure may be applied for working MSMs based on GLMs.

Inference for a parameter of an MSM may be obtained by straightforward application of the delta method (discussed previously) – that is, 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)  0.69063  0.68654 0.0198658 0.64761 0.72548
#> 2: MSM_linear     MSM(slope)  0.13367  0.11924 0.0091106 0.10138 0.13709
#>    psi_transformed lower_transformed upper_transformed
#> 1:         0.68654           0.64761           0.72548
#> 2:         0.11924           0.10138           0.13709
```

9.6.4.1 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: 536.5917  Pars:  0.33422 0.55591 0.10986
#> Iter: 2 fn: 536.5917  Pars:  0.33423 0.55591 0.10986
#> solnp--> Completed in 2 iterations
tmle_msm_fit
#> A tmle3_Fit that took 1 step(s)
#>          type          param init_est tmle_est        se   lower   upper
#> 1: MSM_linear MSM(intercept)  0.69029  0.69005 0.0200303 0.65079 0.72931
#> 2: MSM_linear     MSM(slope)  0.13238  0.13222 0.0091809 0.11423 0.15021
#>    psi_transformed lower_transformed upper_transformed
#> 1:         0.69005           0.65079           0.72931
#> 2:         0.13222           0.11423           0.15021
```

9.6.5 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(
  paste0(
    "https://raw.githubusercontent.com/tlverse/tlverse-data/master/",
    "wash-benefits/washb_data.csv"
  ),
  stringsAsFactors = TRUE
)
washb_data <- washb_data[!is.na(momage), lapply(.SD, as.numeric)]
head(washb_data, 3)
#>       whz tr fracode month aged sex momage momedu momheight hfiacat Nlt18 Ncom
#> 1:  0.00  1       4     9  268   2     30      2    146.40       1     3    1
#> 2: -1.16  1       4     9  286   2     25      2    148.75       3     2
#> 3: -1.05  1      20     9  264   2     25      2    152.15       1     1    1
#>    watmin elec floor walls roof asset_wardrobe asset_table asset_chair
#> 1:      0    1     0     1    1              0           1           1
#> 2:      0    1     0     1    1              0           1           0
#> 3:      0    0     0     1    1              0           0           1
#>    asset_khat asset_chouki asset_tv asset_refrig asset_bike asset_moto
#> 1:          1            0        1            0          0          0
#> 2:          1            1        0            0          0          0
#> 3:          0            1        0            0          0          0
#>    asset_sewmach asset_mobile
#> 1:             0            1
#> 2:             0            1
#> 3:             0            1
```

Next, we specify our NPSEM via the `node_list` object. For our example analysis, we'll consider the outcome to be the weight-for-height Z-score (as in previous chapters), the intervention of interest to be the mother's age at time of child's birth, and take all other covariates to be potential confounders.

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

Were we to consider the counterfactual weight-for-height Z-score under shifts in the age of the mother at child's birth, how would we interpret estimates of our parameter? To simplify our interpretation, consider a shift of just a year in the mother's age (i.e., $\delta = 1$); in this setting, a stochastic intervention would correspond to a policy advocating that potential mothers defer having a child for a single calendar year, possibly implemented through an encouragement design deployed in a clinical setting.

For this example, we'll use the variable importance strategy of considering a grid of stochastic interventions to evaluate the weight-for-height Z-score under a shift in the mother's age down by two years ($\delta = -2$) or up by two years ($\delta = 2$). To do this, we simply initialize a `Spec` `tmle_vimshift_delta` just as we did in a previous example:

```
# initialize a tmle specification for the variable importance parameter
washb_vim_spec <- tmle_vimshift_delta(
  shift_grid = c(-2, 2),
  max_shifted_ratio = 2
)
```

Prior to running our analysis, we'll modify the `learner_list` object we had created such that the density estimation procedure we rely on will be only the location-scale conditional density estimation procedure, as the nonparametric conditional density approach based on the highly adaptive lasso (Díaz and van der Laan, 2011; Benkeser and van der Laan, 2016; Coyle et al., 2020; Hejazi et al., 2020b,a) is currently unable to accommodate larger datasets.

```
# we need to turn on cross-validation for the HOSE learner
cv_hose_hal_lrnr <- Lrnr_cv$new(
  learner = hose_hal_lrnr,
  full_fit = TRUE
)
```

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

Having made the above preparations, we're now ready to estimate the counterfactual mean of the weight-for-height Z-score under a small grid of shifts in the mother's age at child's birth. Just as before, we do this through a simple call to our `tmle3` wrapper function:

```
washb_tmle_fit <- tmle3(washb_vim_spec, washb_data, node_list, learner_list)
washb_tmle_fit
```

9.7 Exercises

9.7.1 The Ideas in Action

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

3. Repeat the preceding analysis, using the same grid of shifts, but instead directly targeting the parameters of the marginal structural model. Interpret the results – that is, what does the slope of the marginal structural model tell us about the trend across the chosen sequence of shifts?

9.7.2 Review of Key Concepts

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

2. 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?

3. What advantages, if any, are there to targeting directly the parameters of a marginal structural model?

Chapter 10

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.

10.1 Classes, Fields, and Methods

The key concept of OOP is that of an object, a collection of data and functions that corresponds to some conceptual unit. Objects have two main types of elements:

1. *fields*, which can be thought of as nouns, are information about an object, and
2. *methods*, which can be thought of as verbs, are actions an object can perform.

Objects are members of classes, which define what those specific fields and methods are. Classes can inherit elements from other classes (sometimes called base classes) – accordingly, classes that are similar, but not exactly the same, can share some parts of their definitions.

Many different implementations of OOP exist, with variations in how these concepts are implemented and used. R has several different implementations, including `S3`,

S4, reference classes, and R6. The tlverse uses the R6 implementation. In R6, methods and fields of a class object are accessed using the $ operator. For a more thorough introduction to R's various OOP systems, see http://adv-r.had.co.nz/OO-essentials.html, from Hadley Wickham's *Advanced R* (Wickham, 2014).

10.2 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

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

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.

Bengtsson, H. (2020). A unifying framework for parallel and distributed processing in r using futures.

Benkeser, D. and van der Laan, M. J. (2016). The highly adaptive lasso estimator. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE.

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

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

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

Chakraborty, B. and Moodie, E. E. (2013). *Statistical Methods for Dynamic Treatment Regimes: Reinforcement Learning, Causal Inference, and Personalized Medicine (Statistics for Biology and Health)*. Springer.

Coyle, J. R. and Hejazi, N. S. (2018). origami: A generalized framework for cross-validation in r. *The Journal of Open Source Software*, 3(21).

Coyle, J. R., Hejazi, N. S., Malenica, I., Phillips, R. V., Arnold, B. F., Mertens, A., Benjamin-Chung, J., Cai, W., Dayal, S., Colford Jr., J. M., Hubbard, A. E., and van der Laan, M. J. (2021). Targeting Learning: Robust statistics for reproducible research. *arXiv*.

Coyle, J. R., Hejazi, N. S., and van der Laan, M. J. (2020). *hal9001: The scalable highly adaptive lasso*. R package version 0.2.7.

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.

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

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

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. (2020a). *haldensify: Highly adaptive lasso conditional density estimation*. R package version 0.0.5.

Hejazi, N. S., Coyle, J. R., and van der Laan, M. J. (2020b). hal9001: Scalable highly adaptive lasso regression in R. *Journal of Open Source Software*.

Hejazi, N. S., van der Laan, M. J., Janes, H. E., Gilbert, P. B., and Benkeser, D. C. (2020c). Efficient nonparametric inference on the effects of stochastic interventions under two-phase sampling, with applications to vaccine efficacy trials. *Biometrics*.

Luedtke, A. and van der Laan, M. J. (2016). Super-learning of an optimal dynamic treatment rule. *International Journal of Biostatistics*, 12(1):305–332. PMID: 27227726.

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

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

Neyman, J. (1990). On the application of probability to agricultural experiments. essay on principles (dm dabrowska & tp speed, trans.). *Statistical Science*, 5:465–480.

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

Pearl, J. (2009a). *Causality: Models, Reasoning and Inference*. Cambridge University Press, New York, NY, USA, 2nd edition.

Pearl, J. (2009b). *Causality: Models, Reasoning, and Inference*. Cambridge University Press.

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. Technical report.

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.

Robins, J. (1986). A new approach to causal inference in mortality studies with a sustained exposure period—application to control of the healthy worker survivor effect. *Mathematical Modelling*, 7(9):1393 – 1512.

Robins, J. and Rotnitzky, A. (2014). Discussion of "dynamic treatment regimes: Technical challenges and applications". *Electron. J. Statist.*, 8(1):1273–1289.

Robins, J. M. (2004). Optimal structural nested models for optimal sequential decisions. In *Proceedings of the Second Seattle Symposium in Biostatistics: Analysis of Correlated Data*, pages 189–326. Springer New York.

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.

Sandercock, P., Collins, R., Counsell, C., Farrell, B., Peto, R., Slattery, J., and Warlow, C. (1997). The international stroke trial (ist): a randomized trial of aspirin, subcutaneous heparin, both, or neither among 19,435 patients with acute ischemic stroke. *Lancet*, 349(9065):1569–1581.

Sandercock, P. A., Niewada, M., and Członkowska, A. (2011). The international stroke trial database. *Trials*, 12(1):101.

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.

Sutton, R. S., Barto, A. G., et al. (1998). *Introduction to reinforcement learning*, volume 135. MIT press Cambridge.

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

Textor, J., Hardt, J., and Knüppel, S. (2011). Dagitty: a graphical tool for analyzing causal diagrams. *Epidemiology*, 22(5):745.

Tofail, F., Fernald, L. C., Das, K. K., Rahman, M., Ahmed, T., Jannat, K. K., Unicomb, L., Arnold, B. F., Ashraf, S., Winch, P. J., et al. (2018). Effect of water quality, sanitation, hand washing, and nutritional interventions on child development in rural bangladesh (wash benefits bangladesh): a cluster-randomised controlled trial. *The Lancet Child & Adolescent Health*, 2(4):255–268.

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

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

van der Laan, M. J. and Luedtke, A. (2015). Targeted learning of the mean outcome under an optimal dynamic treatment rule. *Journal of Causal Inference*, 3(1):61–95. PMCID: PMC4517487.

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

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

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

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

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

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

Wolpert, D. H. (1992). Stacked generalization. *Neural networks*, 5(2):241–259.

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

Zhang, B., A Tsiatis, A., Davidian, M., Zhang, M., and Laber, E. (2016). Estimating optimal treatment regimes from a classification perspective. *Stat*, 5(1):278–278.

Zhao, Y., Zeng, D., Rush, A. J., and Kosorok, M. R. (2012). Estimating individualized treatment rules using outcome weighted learning. *Journal of the American Statistical Association*, 107(499):1106–1118. PMID: 23630406.

Zheng, W. and van der Laan, M. J. (2010). Asymptotic Theory for Cross-validated Targeted Maximum Likelihood Estimation. *U.C. Berkeley Division of Biostatistics Working Paper Series.*