Heterogeneous Treatment Effect Estimation Using Machine Learning

by

Soeren R Kuenzel

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Statistics

in the

Graduate Division

of the

University of California, Berkeley

Summer 2019

Heterogeneous Treatment Effect Estimation Using Machine Learning

Abstract

Heterogeneous Treatment Effect Estimation Using Machine Learning

by

Soeren R Kuenzel

Doctor of Philosophy in Statistics

University of California, Berkeley

Professor Jasjeet Singh Sekhon, Co-chair

Professor Bin Yu, Co-chair

With the rise of large and fine-grained data sets, there is a desire for researchers, physicians, businesses, and policymakers to estimate the treatment effect heterogeneity across individuals and contexts at an ever-greater precision to effectively allocate resources, to adequately assign treatments, and to understand the underlying causal mechanism. In this thesis, we provide tools for estimating and understanding the treatment heterogeneity.

Chapter 1 introduces a unifying framework for many estimators of the Conditional Average Treatment Effect (CATE), a function that describes the treatment heterogeneity. We introduce meta-learners as algorithms that can be combined with any machine learning/regression method to estimate the CATE. We also propose a new meta-learner, the X-learner, that can adapt to structural properties such as the smoothness and sparsity of the underlying treatment effect. We then present its desirable properties through simulations and theory and apply it to two field experiments.

As part of this thesis, we created an R package, `CausalToolbox`, that implements eight CATE estimators and several tools that are useful to estimate the CATE and understand the underlying causal mechanism. Chapter 2 focuses on the `CausalToolbox` package and explains how the package is structured and implemented. The package uses the same syntax for all implemented CATE estimators. That makes it easy for appliers to switch between estimators and compare different estimators on a given data set. We give examples of how it can be used to find a well-performing estimator for a given data set, how confidence intervals for the CATE can be computed, and how estimating the CATE for a unit with many CATE estimators simultaneously can give practitioners a sense for which estimates are unstable and depend heavily on the chosen estimator.

Chapter 3 is an application of the `CausalToolbox` package. It shows how useful it is in a simulation study that has been set up for the *Empirical Investigation of Methods for Heterogeneity Workshop* at the *2018 Atlantic Causal Inference Conference* by Carlos Carvalho, Jennifer Hill, Jared Murray, and Avi Feller, based on the *National Study of Learning Mindsets.*

When implementing the CATE estimators, we noticed that there was a need for a variation of the Random Forests (RF) algorithm that works particularly well for statistical inference. We designed an R package, `forestry`, that implements a new version of the RF algorithm and several tools for statistical inference with it. In Chapter 4, we describe the problem that confidence interval estimation with RF can perform poorly in areas where RF are biased or in areas outside of the support of the training data. We then introduce a new method that allows us to screen for points for which our confidence intervals methods should not be used.

CATE estimates can be used to assign treatments to subjects, but in many studies, estimating the CATE is not the ultimate goal. Researchers often want to understand the underlying causal mechanisms. In Chapter 5, we discuss a modification of the RF algorithm that is particularly interpretable and allows practitioners to understand the underlying mechanism better. Usually, RF are based on deep regression trees that are difficult to understand. In this new version of the RF, we use linear response functions and very shallow trees to make the results more easily understandable. The algorithm finds splits in quasi-linear time and locally adapts to the smoothness of the underlying response functions. In an experimental study, we show that it leads to shallow and interpretable trees that compare favorably to other regression estimators on a broad range of real-world data sets.

To my family

Contents

Contents **ii**

List of Figures **iv**

List of Tables **vi**

I Heterogeneous Treatment Effects 1

1 Meta-learners for Estimating Heterogeneous Treatment Effects using Machine Learning **2**

1.1 Introduction 2
1.2 Meta-algorithms 6
1.3 Simulation Results 10
1.4 Comparison of Convergence Rates 11
1.5 Applications 17
1.6 Conclusion 23

2 CausalToolbox Package **25**

2.1 CATE Estimators 25
2.2 Evaluating the Performance of CATE Estimators 27
2.3 Version Two 29

3 Causaltoolbox—Estimator Stability for Heterogeneous Treatment Effects **32**

3.1 Introduction 32
3.2 Methods 33
3.3 Workshop Results 38
3.4 Postworkshop results 40
3.5 Discussion 40

II Statistical Inference based on Random Forests 43

4 Detachment Index for Evaluating Trustworthiness of Confidence Intervals and Predictons in Random Forests 44
4.1 Introduction 44
4.2 The RF-detachment Index 47
4.3 Applications 54
4.4 Conclusion 59

5 Linear Aggregation in Tree-based Estimators 61
5.1 Introduction 61
5.2 The Splitting Algorithm 65
5.3 Predictive Performance 71
5.4 Interpretability 77
5.5 Conclusion 82

Bibliography 84

A Supporting Information for Meta-learners for Estimating Heterogeneous Treatment Effects using Machine Learning 90
A.1 Simulation Studies 90
A.2 Notes on the ITE 97
A.3 Confidence Intervals for the Social Pressure Analysis 98
A.4 Stability of the Social Pressure Analysis across Meta-learners 103
A.5 The Bias of the S-learner in the Reducing Transphobia Study 104
A.6 Adaptivity to Different Settings and Tuning 104
A.7 Conditioning on the Number of Treated Units 106
A.8 Convergence Rate Results for the T-learner 109
A.9 Convergence Rate Results for the X-learner 112
A.10 Pseudocode 126

B `CausalToolbox` Documentation 132

C Supporting Information for the Detachment Index 147
C.1 Example A 148
C.2 Example B 149

D Supporting Information for Linear Aggregation in Tree-based Estimators 150
D.1 Splitting on a Categorical Feature 150
D.2 Tuned Simulation Hyperparameters 153
D.3 Generating Random Step Function 154

E `forestry` Documentation 155

List of Figures

1.1 Intuition behind the X-learner with an unbalanced design. 8
1.2 Social pressure and voter turnout. 18
1.3 RMSE, bias, and variance for a simulation based on the social pressure and voter turnout experiment. 20
1.4 Histograms for the distribution of the CATE estimates in the Reducing Transphobia study. 22

2.1 Package Structure of the main classes 27

3.1 CATE estimation for ten units. For each unit, the CATE is estimated using 28 different estimators. 35
3.2 Marginal CATE and Partial Dependence Plot (PDP) of the CATE as a function of school-level pre-existing mindset norms and school achiemvent level. 39
3.3 Marginal CATE and PDP of Urbanicity and self-reported expectations. 41

4.1 Histogram of the price distirbution in the cars data set. 46
4.2 Projection of the support of the cars data set onto two covariates colored by error magnitude. 47
4.3 Scatterplot and RF-weights for the four points in the cars data set. 48
4.4 RMSE vs proportion of excluded units for a given threshold. 50
4.5 Error Distribution of detached and trustworthy test points. 52
4.6 Detachment indices of four points compared to its neighbors. 53
4.7 Confidence interval coverage and length of X_RF and X_BART for Simulation 1 and 2. 57
4.8 Confidence interval coverage and length of X_RF and X_BART for the trustworthy points in Simulations 1 and 2. 57
4.9 CI estimation with RF outside the support of the training data. 59
4.10 CI estimation with RF outside the support of the training data on trustworthy points. 59

5.1 Comparison of the classical CART and LRT. 63
5.2 Comparison of estimators with different levels of smoothness. 75
5.3 Four Trees in a linear RF for the GOTV data set. 83

A.1 Illustration of the structural form of the trees in T–RF, S–RF, and CF. 91
A.2 Comparison of S–, T–, and X–BART (left) and S–, T–, and X–RF and CF (right) for Simulation 1. 93
A.3 Comparison of the S-, T-, and X-learners with BART (left) and RF (right) as base learners for Simulation 2 (top) and Simulation 3 (bottom). 95
A.4 Comparison of S-, T-, and X-learners with BART (left) and RF (right) as base learners for Simulation 4 (top) and Simulation 5 (bottom). 96
A.5 Comparison of S–, T–, and X–BART (left) and S–, T–, and X–RF (right) for Simulation 6. 97
A.6 Comparison of normal approximated CI (Algorithm 10) and smoothed CI (Algorithm 11). The blue line is the identity function. 99
A.7 Coverage and average confidence interval length of the three meta-learners. . . . 101
A.8 Approximated bias using Algorithm 12 versus estimated bias using Algorithm 13 and X–RF. 102
A.9 Results for the S-learner and the T-learner for the get-out-the-vote experiment. 104
A.10 Histogram for how often S-RF ignores the treatment effect. 105
A.11 Adaptivity of the S-, T-, and X-Learner. 106

C.1 Confidence intervals and confidence interval lengths for Simulations 1 and 2 seperated by the Overlap regions. 149

List of Tables

2.1 Simulation of 40 RCTs to evaluate the eight implemented CATE estimators. . . 31

4.1 Continuous features of the Cars data set. 45
4.2 Discrete features of the Cars data set. 46
4.3 Summary of the OOB detachment indices for the continuous covariates. 51
4.4 Best five thresholds out of a sample of 100,000. 52
4.5 Number of "trustworthy" units in the overlap area and the area without overlap. 58

5.1 Summary of real data sets. 74
5.2 Estimator RMSE compared across real data sets 76

C.1 Best 40 thresholds out of a sample of 100,000. The numbers in the brackets are the corresponding percentiles. 148

D.1 Selected hyperparameters. 153

Acknowledgments

I would like to start by acknowledging Peter J. Bickel, who has been one of my three advisors. He has supported me from the first day I arrived in Berkeley. And when we learned that it was impossible to have three co-chairs for my dissertation committee, he helped me even then by selflessly stepping down. Nonetheless, he has been as involved with this work as my other advisors and deserves the credit of a co-chair.

Throughout my life, so many people have contributed to my growth that it would be impossible to thank every one of them. Nevertheless, I would like to recognize a few of these people who have made the most significant impacts on the culmination of this thesis.

First and foremost, I would like to express my deepest gratitude to my academic and thesis advisers, Peter J. Bickel, Jasjeet S. Sekhon, and Bin Yu, without whom none of this work would have been possible. To work with and learn from these renowned professors has truly been my honor and privilege. Their guidance during my Ph.D. studies has been invaluable for my intellectual growths and success. Peter J. Bickel, with his encyclopedic knowledge, patience, and kindness, has been always available to me. He gave me the freedom to explore my own ideas while always guiding me in the right direction when I strayed. He has become a wonderful role model for me to follow, and he has taught me many essential skills that will guide me in my future endeavors. I am also extremely grateful for the support and advice of Jasjeet S. Sekhon, whom I thank for being not only my mentor, but also my friend. The encouragement and opportunities he provided during these past five years have been invaluable. His knowledge of machine learning, statistics, causal inference, and political science was one of the driving factors in our joint work. Furthermore, his intuition has been the core idea of the two software packages, `forestry` and `CausalToolbox`. His door was always open for me, and I cannot imagine a more supportive and engaged advocate. Finally, I would like to thank Bin Yu for holding me to the highest of standards and always pushing me to become a better scientist and person. Her persistence, critical feedback, and wealth of knowledge have greatly enhanced the quality of my work and prepared me for my future. From her, I did not merely learn about statistics, but also about the importance of friendship, a strong community, and an honest and direct conversation culture. Even after knowing her for five years, I am still amazed by the energy and commitment she has for research, the future of her students, and the field of statistics.

In addition to my thesis advisers, I am deeply indebted to Peter L. Bartlett, Peter Bühlmann, Peng Ding, Avi Feller, Lisa R. Goldberg, Adityanand Guntuboyina, Michael I. Jordan, and Nicolai Meinshausen who have shaped my academic career by challenging my thinking, leading me to new ideas, and giving me advice on the best course of action.

I would also like to thank my collaborators without whom this dissertation would not have been possible: Dominik Rothenhäusler has been my collaborator and close friend since our first year at the University of Bonn. He has been a source of inspiration and critical feedback for most topics in this dissertation, and we have worked closely on Chapter 4. I am also extremely grateful for having had the opportunity to work with Simon Walter. He has been one of my closest friends here at Berkeley, and we have collaborated extensively on Chapter 3. Finally, I would like to thank Rowan Cassius, Edward W. Liu, Theo Saarinen, and Allen Tang. They have collaborated with me as undergraduate researchers, and without them, Chapter 4 and 5 would not have been possible.

Before my time in Berkeley, I spent one year at Yale, where I was fortunate enough to have many exceptional teachers and collaborators. From this time, I would like to thank, in particular, John W. Emerson and David Pollard for showing me my passion and inspiring me to pursue a career in statistics.

In addition, I would like to thank the undergraduate researchers, Rowan Cassius, Olivia Koshy, Edward W. Liu, Varsha Ramakrishnan, Theo Saarinen, Allen Tang, Nikita Vemuri, Dannver Y. Wu, and Ling Xie, for their commitment to our projects. Despite asking much of them, they always impress me with their perseverance and desire to learn. It has been a pleasure watching them grow as researchers, and I will miss working with them.

During my time as a Ph.D. student, I also had the fantastic opportunity to work with exceptionally driven and hard-working researchers who applied their knowledge in industry. Of the many people I interacted with, I would like to mention Ethan Bashky, Brad Mann, Valerie R. Yakich, and James Yeh, who have shown me how to do impactful research in different industries.

Next, I would like to thank my friends and collaborators whose expertise and dedication greatly benefited my research and who have made my time at Berkeley exponentially more enjoyable. Working with this group of brilliant people has inspired me to raise my standards of work ethnicity. It has served as a wealth of friendships, helpful advice, and collaboration. I could not have asked for a better group of people to have spent these past few years with. Thank you to Reza Abbasi-Asl, Taejoo Ahn, Nicholas Altieri, Sivaraman Balakrishnan, Rebecca L. Barter, Zsolt Bartha, Sumanta Basu, Merle Behr, Eli Ben-Michael, Adam Bloniarz, Yuansi Chen, Raaz Dwivedi, Arturo Fernandez, Ryan Giordano, Geno Guerra, Wooseok Ha, Steve Howard, Christine Kuang, Karl Kumbier, Lihua Lei, Hongwei Li, Xiao Li, Hanzhong Liu, Jamie Murdoch, Kellie Ottoboni, Yannik Pitcan, Dominik Rothenhäusler, Sujayam Saha, Florian Schäfer, Chandan Singh, Jake Soloff, Bradly Stadie, Simon Walter, Yu Wang, and Siqi Wu. I wish you all the best in your future endeavors.

I would also like to thank my gymnastics coach and life mentor, Manfred Thumser, for instilling in me from a young age the value of hard work and discipline. He taught me what it meant to push myself and to always strive for improvement. His wisdom and encouragement have guided me through my time in Berkeley. Without him, I would not be where I am today.

Lastly, I would like to thank my sister, my brother, and my parents for always encouraging my curiosity of the world, for investing in my education, and for their unconditional love and support, even from half a world away.

Thank you!

Part I

Heterogeneous Treatment Effects

Chapter 1

Meta-learners for Estimating Heterogeneous Treatment Effects using Machine Learning

1.1 Introduction

With the rise of large data sets containing fine-grained information about humans and their behavior, researchers, businesses, and policymakers are increasingly interested in how treatment effects vary across individuals and contexts. They wish to go beyond the information provided by estimating the Average Treatment Effect (ATE) in randomized experiments and observational studies. Instead, they often seek to estimate the Conditional Average Treatment Effect (CATE) to personalize treatment regimes and to better understand causal mechanisms. We introduce a new estimator called the X-learner, and we characterize it and many other CATE estimators within a unified meta-learner framework. Their performance is compared using broad simulations, theory, and two data sets from randomized field experiments in political science.

In the first randomized experiment, we estimate the effect of a mailer on voter turnout [27] and, in the second, we measure the effect of door-to-door conversations on prejudice against gender-nonconforming individuals [9]. In both experiments, the treatment effect is found to be non-constant, and we quantify this heterogeneity by estimating the CATE. We obtain insights into the underlying mechanisms, and the results allow us to better target the treatment.

To estimate the CATE, we build on regression or supervised learning methods in statistics and machine learning, which are successfully used in a wide range of applications. Specifically, we study meta-algorithms (or meta-learners) for estimating the CATE in a binary treatment setting. Meta-algorithms decompose estimating the CATE into several sub-regression problems that can be solved with any regression or supervised learning method.

The most common meta-algorithm for estimating heterogeneous treatment effects takes two steps. First, it uses so-called base learners to estimate the conditional expectations of the outcomes separately for units under control and those under treatment. Second, it takes the difference between these estimates. This approach has been analyzed when the base learners are linear regression [22] or tree-based methods [2]. When used with trees, this has been called the *Two-Tree* estimator and we will therefore refer to the general mechanism of estimating the response functions separately as the *T-learner*, "T" being short for "*two*."

Closely related to the T-learner is the idea of estimating the outcome using all of the features and the treatment indicator, without giving the treatment indicator a special role. The predicted CATE for an individual unit is then the difference between the predicted values when the treatment assignment indicator is changed from control to treatment, with all other features held fixed. This meta-algorithm has been studied with BART [35, 29] and regression trees [2] as the base learners. We refer to this meta-algorithm as the *S-learner*, since it uses a "*single*" estimator.

Not all methods that aim to capture the heterogeneity of treatment effects fall in the class of meta-algorithms. For example, some researchers analyze heterogeneity by estimating average treatment effects for meaningful subgroups [32]. Another example is causal forests [80]. Since causal forests are RF-based estimators, they can be compared to meta-learners with RFs in simulation studies. We will see that causal forests and the meta-learners used with RFs perform comparably well, but the meta-learners with other base learners can significantly outperform causal forests.

The main contribution of this paper is the introduction of a new meta-algorithm: the *X-learner*, which builds on the T-learner and uses each observation in the training set in an "X"–like shape. Suppose that we could observe the individual treatment effects directly. We could then estimate the CATE function by regressing the difference of individual treatment effects on the covariates. Structural knowledge about the CATE function (e.g., linearity, sparsity, or smoothness) could be taken into account by either picking a particular regression estimator for CATE or using an adaptive estimator that could learn these structural features. Obviously, we do not observe individual treatment effects because we observe the outcome either under control or under treatment, but never both. The X-learner uses the observed outcomes to estimate the unobserved individual treatment effects. It then estimates the CATE function in a second step as if the individual treatment effects were observed.

The X-learner has two key advantages over other estimators of the CATE. First, it can provably adapt to structural properties such as the sparsity or smoothness of the CATE. This is particularly useful since the CATE is often zero or approximately linear [38, 70]. Secondly, it is particularly effective when the number of units in one treatment group (usually the control group) is much larger than in the other. This occurs because (control) outcomes and covariates are easy to obtain using data collected by administrative agencies, electronic medical record systems, or online platforms. This is the case in our first data example, where election turnout decisions in the U.S. are recorded by local election administrators for all registered individuals.

The rest of the paper is organized as follows. We start with a formal introduction of the meta-learners and provide intuitions for why we can expect the X-learner to perform well when the CATE is smoother than the response outcome functions and when the sample sizes between treatment and control are unequal. We then present the results of an extensive simulation study and provide advice for practitioners before we present theoretical results on the convergence rate for different meta-learners. Finally, we examine two field experiments using several meta-algorithms and illustrate how the X-learner can find useful heterogeneity with fewer observations.

Framework and Definitions

We employ the Neyman–Rubin potential outcome framework [66, 71], and assume a super-population or distribution $\mathcal{P}$ from which a realization of N independent random variables is given as the training data. That is, $(Y_i(0), Y_i(1), X_i, W_i) \sim \mathcal{P}$, where $X_i \in \mathbb{R}^d$ is a d-dimensional covariate or feature vector, $W_i \in \{0,1\}$ is the treatment assignment indicator (to be defined precisely later), $Y_i(0) \in \mathbb{R}$ is the potential outcome of unit i when i is assigned to the control group, and $Y_i(1)$ is the potential outcome when i is assigned to the treatment group. With this definition, the Average Treatment Effect is defined as

$$\text{ATE} := \mathbb{E}[Y(1) - Y(0)].$$

It is also useful to define the response under control, μ_0, and the response under treatment, μ_1, as

$$\mu_0(x) := \mathbb{E}[Y(0)|X = x] \qquad \text{and} \qquad \mu_1(x) := \mathbb{E}[Y(1)|X = x].$$

Furthermore, we use the following representation of $\mathcal{P}$:

$$\begin{aligned} X &\sim \Lambda, \\ W &\sim \text{Bern}(e(X)), \\ Y(0) &= \mu_0(X) + \varepsilon(0), \\ Y(1) &= \mu_1(X) + \varepsilon(1), \end{aligned} \tag{1.1}$$

where Λ is the marginal distribution of X, $\varepsilon(0)$ and $\varepsilon(1)$ are zero-mean random variables and independent of X and W, and $e(x) = \mathbb{P}(W = 1|X = x)$ is the propensity score.

The fundamental problem of causal inference is that for each unit in the training data set, we observe either the potential outcome under control ($W_i = 0$), or the potential outcome under treatment ($W_i = 1$) but never both. Hence we denote the observed data as

$$\mathcal{D} = (Y_i, X_i, W_i)_{1 \le i \le N},$$

with $Y_i = Y_i(W_i)$. Note that the distribution of $\mathcal{D}$ is specified by $\mathcal{P}$. To avoid the problem that with a small but non-zero probability all units are under control or under treatment, we will analyze the behavior of different estimators conditional on the number of treated units. That is, for a fixed n with $0 < n < N$, we condition on the event that

$$\sum_{i=1}^{N} W_i = n.$$

This will enable us to state the performance of an estimator in terms of the number of treated units n and the number of control units $m = N - n$.

For a new unit i with covariate vector x_i, in order to decide whether to give the unit the treatment, we wish to estimate the Individual Treatment Effect (ITE) of unit i, D_i, which is defined as

$$D_i := Y_i(1) - Y_i(0).$$

However, we do not observe D_i for any unit, and D_i is not identifiable without strong additional assumptions in the sense that one can construct data-generating processes with the same distribution of the observed data, but a different D_i (Example 3). Instead, we will estimate the CATE function, which is defined as

$$\tau(x) := \mathbb{E}\Big[D\Big|X = x\Big] = \mathbb{E}\Big[Y(1) - Y(0)\Big|X = x\Big],$$

and we note that the best estimator for the CATE is also the best estimator for the ITE in terms of the MSE. To see that, let $\hat{\tau}_i$ be an estimator for D_i and decompose the MSE at x_i

$$\begin{aligned} &\mathbb{E}\left[(D_i - \hat{\tau}_i)^2 | X_i = x_i\right] \\ =&\mathbb{E}\left[(D_i - \tau(x_i))^2 | X_i = x_i\right] + \mathbb{E}\left[(\tau(x_i) - \hat{\tau}_i)^2\right]. \end{aligned} \tag{1.2}$$

Since we cannot influence the first term in the last expression, the estimator that minimizes the MSE for the ITE of i also minimizes the MSE for the CATE at x_i.

In this paper, we are interested in estimators with a small Expected Mean Squared Error (EMSE) for estimating the CATE,

$$\text{EMSE}(\mathcal{P}, \hat{\tau}) = \mathbb{E}\left[(\tau(\mathcal{X}) - \hat{\tau}(\mathcal{X}))^2\right].$$

The expectation is here taken over $\hat{\tau}$ and $\mathcal{X} \sim \Lambda$, where $\mathcal{X}$ is independent of $\hat{\tau}$.

To aid our ability to estimate τ, we need to assume that there are no hidden confounders [64]:

Condition 1

$$(\varepsilon(0), \varepsilon(1)) \perp W | X.$$

This assumption is, however, not sufficient to identify the CATE. One additional assumption that is often made to obtain identifiability of the CATE in the support of X is to assume that the propensity score is bounded away from 0 and 1:

Condition 2 *There exists $e_{\min}$ and $e_{\max}$ such that for all x in the support of X,*

$$0 < e_{\min} < e(x) < e_{\max} < 1.$$

1.2 Meta-algorithms

In this section, we formally define a meta-algorithm (or meta-learner) for the CATE as the result of combining supervised learning or regression estimators (i.e., base learners) in a specific manner while allowing the base learners to take any form. Meta-algorithms thus have the flexibility to appropriately leverage different sources of prior information in separate sub-problems of the CATE estimation problem: they can be chosen to fit a particular type of data, and they can directly take advantage of existing data analysis pipelines.

We first review both S- and T-learners, and we then propose the X-learner, which is a new meta-algorithm that can take advantage of unbalanced designs (i.e., the control or the treated group is much larger than the other group) and existing structures of the CATE (e.g., smoothness or sparsity). Obviously, flexibility is a gain only if the base learners in the meta-algorithm match the features of the data and the underlying model well.

The T-learner takes two steps. First, the control response function,

$$\mu_0(x) = \mathbb{E}[Y(0)|X = x],$$

is estimated by a base learner, which could be any supervised learning or regression estimator using the observations in the control group, $\{(X_i, Y_i)\}_{W_i=0}$. We denote the estimated function as $\hat{\mu}_0$. Second, we estimate the treatment response function,

$$\mu_1(x) = \mathbb{E}[Y(1)|X = x],$$

with a potentially different base learner, using the treated observations and denoting the estimator by $\hat{\mu}_1$. A T-learner is then obtained as

$$\hat{\tau}_T(x) = \hat{\mu}_1(x) - \hat{\mu}_0(x). \tag{1.3}$$

Pseudocode for this T-learner can be found in Algorithm 5.

In the S-learner, the treatment indicator is included as a feature similar to all the other features without the indicator being given any special role. We thus estimate the combined response function,

$$\mu(x, w) := \mathbb{E}[Y^{obs}|X = x, W = w],$$

using any base learner (supervised machine learning or regression algorithm) on the entire data set. We denote the estimator as $\hat{\mu}$. The CATE estimator is then given by

$$\hat{\tau}_S(x) = \hat{\mu}(x, 1) - \hat{\mu}(x, 0), \tag{1.4}$$

and pseudocode is provided in Algorithm 6.

There are other meta-algorithms in the literature, but we do not discuss them here in detail because of limited space. For example, one may transform the outcomes so that any regression method can estimate the CATE directly (Algorithm 8) [2, 76, 59]. In our simulations, this algorithm performs poorly, and we do not discuss it further, but it may do well in other settings.

X-learner

We propose the X-learner, and provide an illustrative example to highlight its motivations. The basic idea of the X-learner can be described in three stages:

1. Estimate the response functions

   $$\mu_0(x) = \mathbb{E}[Y(0)|X = x], \text{ and} \tag{1.5}$$
   $$\mu_1(x) = \mathbb{E}[Y(1)|X = x], \tag{1.6}$$

   using any supervised learning or regression algorithm and denote the estimated functions $\hat{\mu}_0$ and $\hat{\mu}_1$. The algorithms used are referred to as the base learners for the first stage.

2. Impute the treatment effects for the individuals in the treated group, based on the control outcome estimator, and the treatment effects for the individuals in the control group, based on the treatment outcome estimator, that is,

   $$\tilde{D}_i^1 := Y_i^1 - \hat{\mu}_0(X_i^1), \text{ and} \tag{1.7}$$
   $$\tilde{D}_i^0 := \hat{\mu}_1(X_i^0) - Y_i^0, \tag{1.8}$$

   and call these the imputed treatment effects. Note that if $\hat{\mu}_0 = \mu_0$ and $\hat{\mu}_1 = \mu_1$, then $\tau(x) = \mathbb{E}[\tilde{D}^1|X = x] = \mathbb{E}[\tilde{D}^0|X = x]$.

   Employ any supervised learning or regression method(s) to estimate $\tau(x)$ in two ways: using the imputed treatment effects as the response variable in the treatment group to obtain $\hat{\tau}_1(x)$, and similarly in the control group to obtain $\hat{\tau}_0(x)$. Call the supervised learning or regression algorithms base learners of the second stage.

3. Define the CATE estimate by a weighted average of the two estimates in Stage 2:

   $$\hat{\tau}(x) = g(x)\hat{\tau}_0(x) + (1 - g(x))\hat{\tau}_1(x), \tag{1.9}$$

   where $g \in [0, 1]$ is a weight function.

See Algorithm 7 for pseudocode.

Remark 1 *$\hat{\tau}_0$ and $\hat{\tau}_1$ are both estimators for τ, while g is chosen to combine these estimators to one improved estimator $\hat{\tau}$. Based on our experience, we observe that it is good to use an estimate of the propensity score for g, so that $g = \hat{e}$, but it also makes sense to choose $g = 1$ or 0, if the number of treated units is very large or small compared to the number of control units. For some estimators, it might even be possible to estimate the covariance matrix of $\hat{\tau}_1$ and $\hat{\tau}_0$. One may then wish to choose g to minimize the variance of $\hat{\tau}$.*

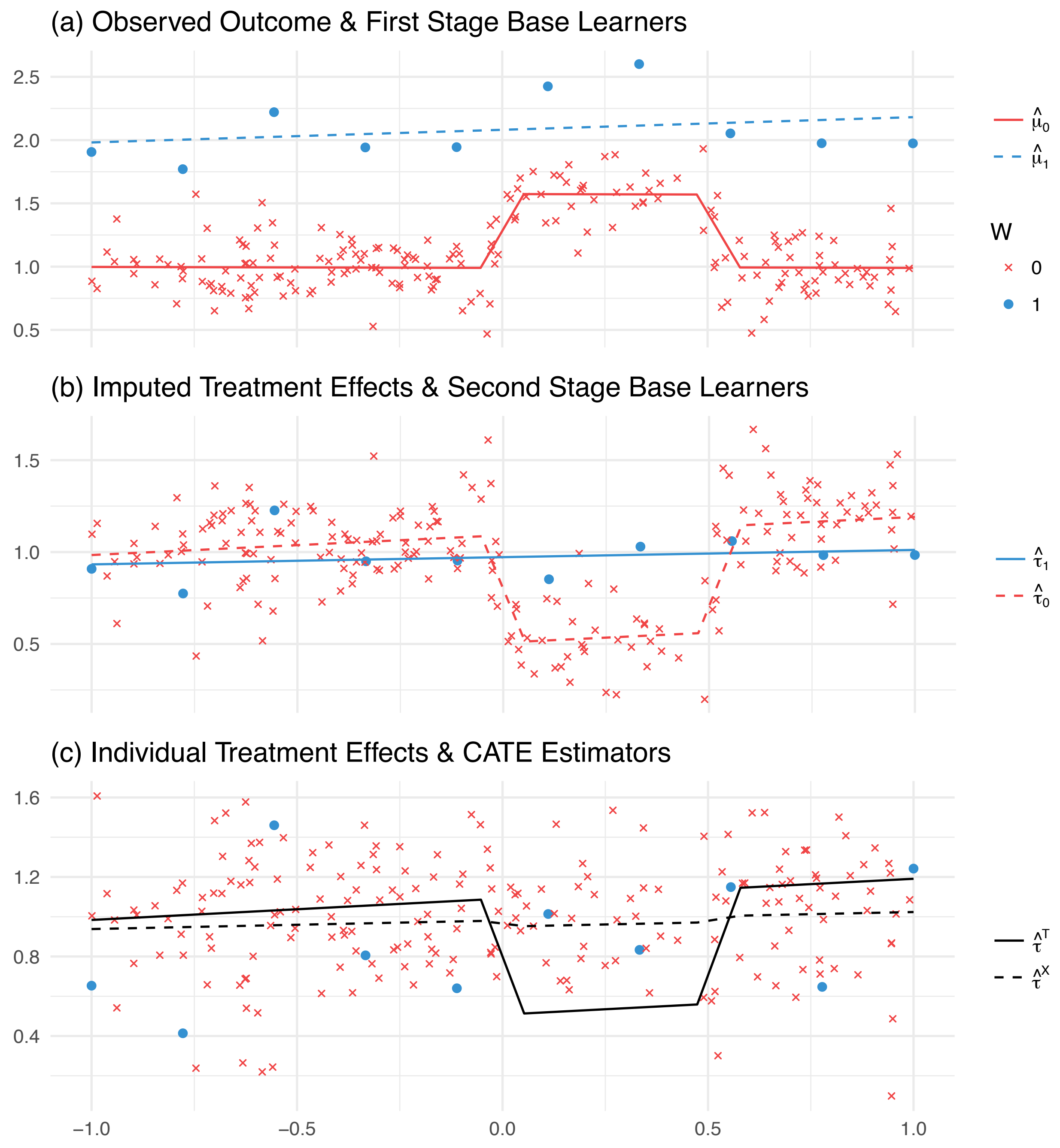


Figure 1.1: Intuition behind the X-learner with an unbalanced design.

Intuition behind the meta-learners

The X-learner can use information from the control group to derive better estimators for the treatment group and vice versa. We will illustrate this using a simple example. Suppose that we want to study a treatment, and we are interested in estimating the CATE as a function of one covariate x. We observe, however, very few units in the treatment group and many units in the control group. This situation often arises with the growth of administrative and online data sources: data on control units is often far more plentiful than data on treated units. Figure 1.1(a) shows the outcome for units in the treatment group (circles) and the outcome of unit in the untreated group (crosses). In this example, the CATE is constant and equal to one.

For the moment, let us look only at the treated outcome. When we estimate $\mu_1(x) = \mathbb{E}[Y(1)|X = x]$, we must be careful not to overfit the data since we observe only 10 data points. We might decide to use a linear model, $\hat{\mu}_1(x)$ (dashed line), to estimate μ_1. For the control group, we notice that observations with $x \in [0, 0.5]$ seem to be different, and we end up modeling $\mu_0(x) = \mathbb{E}[Y(0)|X = x]$ with a piecewise linear function with jumps at 0 and 0.5 (solid line). This is a relatively complex function, but we are not worried about overfitting since we observe many data points.

The *T-learner* would now use estimator $\hat{\tau}_T(x) = \hat{\mu}_1(x) - \hat{\mu}_0(x)$ (see Figure 1.1(c), solid line), which is a relatively complicated function with jumps at 0 and 0.5, while the true $\tau(x)$ is a constant. This is, however, problematic because we are estimating a complex CATE function, based on ten observations in the treated group.

When choosing an estimator for the treatment group, we correctly avoided overfitting, and we found a good estimator for the treatment response function and, as a result, we chose a relatively complex estimator for the CATE, namely, the quantity of interest. We could have selected a piecewise linear function with jumps at 0 and 0.5, but this, of course, would have been unreasonable when looking only at the treated group. If, however, we were to also take the control group into account, this function would be a natural choice. In other words, we should change our objective for $\hat{\mu}_1$ and $\hat{\mu}_0$. We want to estimate $\hat{\mu}_1$ and $\hat{\mu}_0$ in such a way that their difference is a good estimator for τ.

The *X-learner* enables us to do exactly that. It allows us to use structural information about the CATE to make efficient use of an unbalanced design. The first stage of the X-learner is the same as the first stage of the T-learner, but in its second stage, the estimator for the controls is subtracted from the observed treated outcomes and similarly the observed control outcomes are subtracted from estimated treatment outcomes to obtain the imputed treatment effects,

$$\begin{aligned}\tilde{D}^1_i &:= Y^1_i - \hat{\mu}_0(X^1_i),\\ \tilde{D}^0_i &:= \hat{\mu}_1(X^0_i) - Y^0_i.\end{aligned}$$

Here we use the notation that Y^0_i and Y^1_i are the ith observed outcome of the control and the treated group, respectively. X^1_i, X^0_i are the corresponding feature vectors. Figure 1.1(b) shows the imputed treatment effects, $\tilde{D}$. By choosing a simple—here linear—function to estimate $\tau_1(x) = \mathbb{E}[\tilde{D}^1|X^1 = x]$ we effectively estimate a model for $\mu_1(x) = \mathbb{E}[Y^1|X^1 = x]$, which has a similar shape to $\hat{\mu}_0$. By choosing a relatively poor model for $\mu_1(x)$, $\tilde{D}^0$ (the red crosses in Figure 1.1(b)) are relatively far away from $\tau(x)$, which is constant and equal to 1. The model for $\tau_0(x) = \mathbb{E}[\tilde{D}^0|X = x]$ will thus be relatively poor. However, our final estimator combines these two estimators according to

$$\hat{\tau}(x) = g(x)\hat{\tau}_0(x) + (1 - g(x))\hat{\tau}_1(x).$$

If we choose $g(x) = \hat{e}(x)$, an estimator for the propensity score, $\hat{\tau}$ will be very similar to $\hat{\tau}_1(x)$, since we have many more observations in the control group; i.e., $\hat{e}(x)$ is small. Figure 1.1(c) shows the T-learner and the X-learner.

It is difficult to assess the general behavior of the S-learner in this example because we must choose a base learner. For example, when we use RF as the base learner for this data set, the S-learner's first split is on the treatment indicator in 97.5% of all trees in our simulations because the treatment assignment is very predictive of the observed outcome, Y (see also Figure A.10). From there on, the S-learner and the T-learner are the same, and we observe them to perform similarly poorly in this example.

1.3 Simulation Results

In this section, we conduct a broad simulation study to compare the different meta-learners. In particular, we summarize our findings and provide general remarks on the strengths and weaknesses of the S-, T-, and X-learners, while deferring the details to the Supporting Information (SI). The simulations are key to providing an understanding of the performance of the methods we consider for model classes that are not covered by our theoretical results.

Our simulation study is designed to consider a range of situations. We include conditions under which the S-learner or the T-learner is likely to perform the best, as well as simulation setups proposed by previous researchers [80]. We consider cases where the treatment effect is zero for all units (and so pooling the treatment and control groups would be beneficial), and cases where the treatment and control response functions are completely different (and so pooling would be harmful). We consider cases with and without confounding,[1] and cases with equal and unequal sample sizes across treatment conditions. All simulations discussed in this section are based on synthetic data. For details, please see Section A.1. We provide additional simulations based on actual data when we discuss our applications.

We compare the S-, T-, and X-learners with RF and BART as base learners. We implement a version of RF for which the tree structure is independent of the leaf prediction given the observed features, the so-called honest RF in an R package called `hte` [42]. This version of RF is particularly accessible from a theoretical point of view, it performs well in noisy settings, and it is better suited for inference [68, 80]. For BART, our software uses the `dbarts` [15] implementation for the base learner.

[1]Confounding here refers to the existence of an unobserved covariate that influences both the treatment variable, W, and at least one of the portential outcomes $Y(0), Y(1)$.

Comparing different base learners enables us to demonstrate two things. On the one hand, it shows that the conclusions we draw about the S-, T-, and X-learner are not specific to a particular base learner and, on the other hand, it demonstrates that the choice of base learners can make a large difference in prediction accuracy. The latter is an important advantage of meta-learners since subject knowledge can be used to choose base learners that perform well. For example, in Simulations 2 and 4 the response functions are globally linear, and we observe that estimators that act globally such as BART have a significant advantage in these situations or when the data set is small. If, however, there is no global structure or when the data set is large, then more local estimators such as RF seem to have an advantage (Simulations 3 and 5).

We observe that the choice of meta-learner can make a large difference, and for each meta-learner there exist cases where it is the best-performing estimator.

The S-learner treats the treatment indicator like any other predictor. For some base learners such as k-nearest neighbors it is not a sensible estimator, but for others it can perform well. Since the treatment indicator is given no special role, algorithms such as the lasso and RFs can completely ignore the treatment assignment by not choosing/splitting on it. This is beneficial if the CATE is in many places 0 (Simulations 4 and 5), but—as we will see in our second data example—the S-learner can be biased toward 0.

The T-learner, on the other hand, does not combine the treated and control groups. This can be a disadvantage when the treatment effect is simple because by not pooling the data, it is more difficult for the T-learner to mimic a behavior that appears in both the control and treatment response functions (e.g., Simulation 4). If, however, the treatment effect is very complicated, and there are no common trends in μ_0 and μ_1, then the T-learner performs especially well (Simulations 2 and 3).

The X-learner performs particularly well when there are structural assumptions on the CATE or when one of the treatment groups is much larger than the other (Simulation 1 and 3). In the case where the CATE is 0, it usually does not perform as well as the S-learner, but it is significantly better than the T-learner (Simulations 4, 5, and 6), and in the case of a very complex CATE, it performs better than the S-learner and it often outperforms even the T-learner (Simulations 2 and 3). These simulation results lead us to the conclusion that unless one has a strong belief that the CATE is mostly 0, then, as a rule of thumb, one should use the X-learner with BART for small data sets and RF for bigger ones. In the sequel, we will further support these claims with additional theoretical results and empirical evidence from real data and data-inspired simulations.

1.4 Comparison of Convergence Rates

In this section, we provide conditions under which the X-learner can be proven to outperform the T-learner in terms of pointwise estimation rate. These results can be viewed as attempts to rigorously formulate intuitions regarding when the X-learner is desirable. They corroborate our intuition that the X-learner outperforms the T-learner when one group is much larger than the other group and when the CATE function has a simpler form than those of the underlying response functions themselves.

Let us start by reviewing some of the basic results in the field of minimax nonparametric regression estimation [30]. In the standard regression problem, one observes N independent and identically distributed tuples $(X_i, Y_i)_i \in \mathbb{R}^{d\times N} \times \mathbb{R}^N$ generated from some distribution $\mathcal{P}$ and one is interested in estimating the conditional expectation of Y given some feature vector x, $\mu(x) = \mathbb{E}[Y|X = x]$. The error of an estimator $\hat{\mu}_N$ can be evaluated by the Expected Mean Squared Error (EMSE),

$$\text{EMSE}(\mathcal{P}, \hat{\mu}_N) = \mathbb{E}[(\hat{\mu}_N(\mathcal{X}) - \mu(\mathcal{X}))^2].$$

For a fixed $\mathcal{P}$, there are always estimators that have a very small EMSE. For example, choosing $\hat{\mu}_N \equiv \mu$ would have no error. However, $\mathcal{P}$ and thus μ would be unknown. Instead, one usually wants to find an estimator that achieves a small EMSE for a relevant set of distributions (such a set is relevant if it captures domain knowledge or prior information about the problem). To make this problem feasible, a typical approach is the minimax approach where one analyzes the worst performance of an estimator over a family, F, of distributions [79]. The goal is to find an estimator that has a small EMSE for all distributions in this family. For example, if F_0 is the family of distributions $\mathcal{P}$ such that $X \sim \text{Unif}[0, 1]$, $Y = \beta X + \varepsilon$, $\varepsilon \sim N(0, 1)$, and $\beta \in \mathbb{R}$, then it is well known that the OLS estimator achieves the optimal parametric rate. That is, there exists a constant $C \in \mathbb{R}$ such that for all $\mathcal{P} \in F_0$,

$$\text{EMSE}(\mathcal{P}, \hat{\mu}_N^{\text{OLS}}) \leq CN^{-1}.$$

If, however, F_1 is the family of all distributions $\mathcal{P}$ such that $X \sim \text{Unif}[0, 1]$, $Y \sim \mu(X) + \varepsilon$, and μ is a Lipschitz continuous function with a bounded Lipschitz constant, then there exists no estimator that achieves the parametric rate uniformly for all possible distributions in F_1. To be precise, we can at most expect to find an estimator that achieves a rate of $N^{-2/3}$ and that there exists a constant C', such that

$$\liminf_{N\to\infty} \inf_{\hat{\mu}_N} \sup_{\mathcal{P}\in F_1} \frac{\text{EMSE}(\mathcal{P}, \hat{\mu}_N)}{N^{-2/3}} > C' > 0.$$

The Nadaraya–Watson and the k-nearest neighbors estimators can achieve this optimal rate [4, 30].

Crucially, the fastest rate of convergence that holds uniformly for a family F is a property of the family to which the underlying data-generating distribution belongs. It will be useful for us to define sets of families for which particular rates are achieved.

Definition 1 (Families with bounded minimax rate) *For $a \in (0, 1]$, we define $S(a)$ to be the set of all families, F, with a minimax rate of at most N^{-a}.*

Note that for any family $F \in S(a)$ there exists an estimator $\hat{\mu}$ and a constant C such that for all $N \geq 1$,

$$\sup_{\mathcal{P}\in F} \text{EMSE}(\mathcal{P}, \hat{\mu}_N) \leq CN^{-a}.$$

From the examples above, it is clear that $F_0 \in S(1)$ and $F_1 \in S(2/3)$.

Even though the minimax rate of the EMSE is not very practical since one rarely knows that the true data-generating process is in some reasonable family of distributions, it is nevertheless one of the very few useful theoretical tools to compare different nonparametric estimators. If for a big class of distributions, the worst EMSE of an estimator $\hat{\mu}^A$ is smaller than the worst EMSE of an estimator $\hat{\mu}^B$, then one might prefer estimator $\hat{\mu}^A$ over estimator $\hat{\mu}^B$. Furthermore, if the estimator of choice does not have a small error for a family that we believe based on domain information could be relevant in practice, then we might expect $\hat{\mu}$ to have a large EMSE in real data.

Implication for CATE estimation

Let us now apply the minimax approach to the problem of estimating the CATE. Recall that we assume a superpopulation, $\mathcal{P}$, of random variables $(Y(0), Y(1), X, W)$ according to 1.1 and we observe N outcomes, $(X_i, W_i, Y_i^{obs})_{i=1}^N$. To avoid the problem that with a small but non-zero probability all units are treated or untreated, we analyze the expected mean squared error of an estimator given that there are $0 < n < N$ treated units,

$$\text{EMSE}(\mathcal{P}, \hat{\tau}^{mn}) = \mathbb{E}\left[(\tau(\mathcal{X}) - \hat{\tau}^{mn}(\mathcal{X}))^2 \middle| \sum_{i=1}^{N} W_i = n\right].$$

The expectation is taken over the observed data,[2] $(X_i, W_i, Y_i)_{i=1}^N$, given that we observe n treated units, and over $\mathcal{X}$, which is distributed according to $\mathcal{P}$.

As in Definition 1, we characterize families of superpopulations by the rates at which the response functions and the CATE function can be estimated.

Definition 2 (Superpopulations with given rates) *For $a_\mu, a_\tau \in (0, 1]$, we define $S(a_\mu, a_\tau)$ to be the set of all families of distributions $\mathcal{P}$ of $(Y(0), Y(1), X, W)$ such that ignorability holds (Condition 1), the overlab condition (Condition 2) is satisfied, and the following conditions hold:*

1. *The distribution of $(X, Y(0))$ given $W = 0$ is in a family $F_0 \in S(a_\mu)$,*
2. *The distribution of $(X, Y(1))$ given $W = 1$ is in a family $F_1 \in S(a_\mu)$,*
3. *The distribution of $(X, \mu_1(X) - Y(0))$ given $W = 0$ is in a family $F_{\tau 0} \in S(a_\tau)$, and*
4. *The distribution of $(X, Y(1) - \mu_0(X))$ given $W = 1$ is in a family $F_{\tau 1} \in S(a_\tau)$.*

A simple example of a family in $S(2/3, 1)$ is the set of distributions $\mathcal{P}$ for which $X \sim \text{Unif}([0, 1])$, $W \sim \text{Bern}(1/2)$, μ_0 is any Lipschitz continuous function, τ is linear, and $\varepsilon(0), \varepsilon(1)$ are independent and standard normal-distributed.

We can also build on existing results from the literature to characterize many families in terms of smoothness conditions on the CATE and on the response functions.

[2]Refer to Section A.7 for a careful treatment of the distributions involved.

Example 1 *Let $C > 0$ be an arbitrary constant and consider the family, F_2, of distributions for which X has compact support in $\mathbb{R}^d$, the propensity score e is bounded away from 0 and 1 (Condition 2), μ_0, μ_1 are C Lipschitz continuous, and the variance of ε is bounded. Then it follows [30] that*

$$F_2 \in S\left(\frac{2d}{2+d}, \frac{2d}{2+d}\right).$$

Note that we don't have any assumptions on X apart from its support being bounded. If we are willing to make assumptions on the density (e.g., X is uniformly distributed), then we can characterize many distributions by the smoothness conditions of μ_0, μ_1, and τ.

Definition 3 ($(\mathbf{p}, \mathbf{C})$-smooth functions [30]) *Let $p = k + \beta$ for some $k \in \mathbb{N}$ and $0 < \beta \leq 1$, and let $C > 0$. A function $f : \mathbb{R}^d \longrightarrow \mathbb{R}$ is called (p, C)-smooth if for every $\alpha = (\alpha_1, \dots, \alpha_d)$, $\alpha_i \in \mathbb{N}$, $\sum_{j=1}^d \alpha_j = k$, the partial derivative $\frac{\partial^k f}{\partial x_1^{\alpha_1} \dots \partial x_d^{\alpha_d}}$ exists and satisfies*

$$\left| \frac{\partial^k f}{\partial x_1^{\alpha_1} \dots \partial x_d^{\alpha_d}}(x) - \frac{\partial^k f}{\partial x_1^{\alpha_1} \dots \partial x_d^{\alpha_d}}(z) \right| \leq C \|x - z\|^\beta.$$

Example 2 *Let C_1, C_2 be arbitrary constants and consider the family, F_3, of distributions for which $X \sim$ Unif$([0,1]^d)$, $e \equiv c \in (0,1)$, ε is two-dimensional normally distributed, μ_0 and μ_1 are (p_μ, C_1)-smooth, and τ is (p_τ, C_2)-smooth.[3] Then it follows [73, 30] that*

$$F_3 \in S\left(\frac{2d}{2p_\mu + d}, \frac{2d}{2p_\tau + d}\right).$$

Let us intuitively understand the difference between the T- and X-learners. The T-learner splits the problem of estimating the CATE into the two subproblems of estimating μ_0 and μ_1 separately. By appropriately choosing the base learners, we can expect to achieve the minimax optimal rates of m^{-a_μ} and n^{-a_μ}, respectively,

$$\begin{aligned} \sup_{\mathcal{P}_0 \in F_0} \text{EMSE}(\mathcal{P}_0, \hat{\mu}_0^m) &\leq C m^{-a_\mu}, \quad \text{and} \\ \sup_{\mathcal{P}_1 \in F_1} \text{EMSE}(\mathcal{P}_1, \hat{\mu}_1^n) &\leq C n^{-a_\mu}, \end{aligned} \tag{1.10}$$

where C is some constant. Those rates translate immediately to rates for estimating τ,

$$\sup_{\mathcal{P} \in F} \text{EMSE}(\mathcal{P}, \hat{\tau}_T^{nm}) \leq C_\tau \left(m^{-a_\mu} + n^{-a_\mu}\right).$$

In general, we cannot expect to do better than this when using an estimation strategy that falls in the class of T-learners, because the subproblems in Equation 1.10 are treated

[3] The assumption that X is uniformly distributed and the propensity score is constant can be generalized if one uses a slightly different risk [31, 47, 73].

completely independently and there is nothing to be learned from the treatment group about the control group and vice versa.

In Section A.8, we present a careful analysis of this result and we prove the following theorem.

Theorem 1 (Minimax rates of the T-learner) *For a family of superpopulations, $F \in S(a_\mu, a_\tau)$, there exist base learners to be used in the T-learner so that the corresponding T-learner estimates the CATE at a rate*

$$\mathcal{O}(m^{-a_\mu} + n^{-a_\mu}). \tag{1.11}$$

The X-learner, on the other hand, can be seen as a locally weighted average of the two estimators, $\hat{\tau}_0$ and $\hat{\tau}_1$ (Eq. 1.9). Take for the moment, $\hat{\tau}_1$. It consists of an estimator for the outcome under control, which achieves a rate of m^{-a_μ}, and an estimator for the imputed treatment effects, which should intuitively achieve a rate of n^{-a_τ}. We therefore expect that under some conditions on $F \in S(a_\mu, a_\tau)$, there exist base learners such that $\hat{\tau}_0$ and $\hat{\tau}_1$ in the X-learner achieve the rates

$$\mathcal{O}(m^{-a_\tau} + n^{-a_\mu}) \quad \text{and} \quad \mathcal{O}(m^{-a_\mu} + n^{-a_\tau}), \tag{1.12}$$

respectively.

Even though it is theoretically possible that a_τ is similar to a_μ, our experience with real data suggests that it is often larger (i.e., the treatment effect is *simpler* to estimate than the potential outcomes), because the CATE function is often smoother or sparsely related to the feature vector. In this case, the X-learner converges at a faster rate than the T-learner.

Remark 2 (Unbalanced groups) *In many real-world applications, we observe that the number of control units is much larger than the number of treated units, $m \gg n$. This happens, for example, if we test a new treatment and we have a large number of previous (untreated) observations that can be used as the control group. In that case, the bound on the EMSE of the T-learner will be dominated by the regression problem for the treated response function,*

$$\sup_{\mathcal{P} \in F} EMSE(\mathcal{P}, \hat{\tau}_T^{nm}) \le C_1 n^{-a_\mu}. \tag{1.13}$$

The EMSE of the X-learner, however, will be dominated by the regression problem for the imputed treatment effects and it will achieve a faster rate of n^{-a_τ},

$$\sup_{\mathcal{P} \in F} EMSE(\mathcal{P}, \hat{\tau}_X^{nm}) \le C_2 n^{-a_\tau}. \tag{1.14}$$

This is a substantial improvement on 1.13 when $a_\tau > a_\mu$, and it demonstrates that in contrast to the T-learner, the X-learner can exploit structural conditions on the treatment effect. We therefore expect the X-learner to perform particularly well when one of the treatment groups is larger than the other. This can also be seen in our extensive simulation study presented in Section A.1 and in the field experiment on social pressure on voter turnout presented in the Applications section of this paper.

Example when the CATE is linear

It turns out to be mathematically very challenging to give a satisfying statement of the extra conditions needed on F in 1.12. However, they are satisfied under weak conditions when the CATE is Lipschitz continuous (cf. Section A.9) and, as we discuss in the rest of this section, when the CATE is linear. We emphasize that we believe that this result holds in much greater generality.

Let us discuss the result in the following families of distributions with a linear CATE, but without assumptions on the response functions other than that they can be estimated at some rate a.

Condition 3 *The treatment effect is linear, $\tau(x) = x^T\beta$, with $\beta \in \mathbb{R}^d$.*

Condition 4 *There exists an estimator $\hat{\mu}_0^m$ and constants $C_0, a > 0$ with*

$$EMSE(\mathcal{P}, \hat{\mu}_0^m) = \mathbb{E}[(\mu_0(X) - \hat{\mu}_0^m(X))^2 | W = 0] \leq C_0 m^{-a}.$$

To help our analysis, we also assume that the noise terms are independent given X and that the feature values are well behaved.

Condition 5 *The error terms ε_i are independent given X, with $\mathbb{E}[\varepsilon_i | X = x] = 0$ and $Var[\varepsilon_i | X = x] \leq \sigma^2$.*

Condition 6 *X has finite second moments,*

$$\mathbb{E}[\|X\|_2^2] \leq C_X,$$

and the eigenvalues of the sample covariance matrix of X^1 are well conditioned, in the sense that there exists an $n_0 \in \mathbb{N}$ and a constant $C_\Sigma \in \mathbb{R}$ such that for all $n > n_0$,

$$\mathbb{P}\left(\gamma_{min}^{-1}(\hat{\Sigma}_n) \leq C_\Sigma\right) = 1. \tag{1.15}$$

Under these conditions, we can prove that the X-learner achieves a rate of $O(m^{-a} + n^{-1})$.

Theorem 2 *Assume that we observe m control units and n treated units from a superpopulation that satisfies Conditions 1–6; then $\hat{\tau}_1$ of the X-learner with $\hat{\mu}_0^m$ in the first stage and OLS in the second stage achieves a rate of $O(m^{-a} + n^{-1})$. Specifically, for all $n > n_0, m > 1$,*

$$EMSE(\mathcal{P}, \hat{\tau}_1^{mn}) \leq C(m^{-a} + n^{-1}),$$

with $C = \max\left(\frac{e_{\max} - e_{\max}e_{\min}}{e_{\min} - e_{\max}e_{\min}} C_0, \sigma^2 d\right) C_X C_\Sigma$.

We note that an equivalent statement also holds for the pointwise MSE (Theorem 4) and for $\hat{\tau}_0$.

This example also suppports Remark 2, because if there are many control units, $m \geq n^{1/a}$, then the X-learner achieves the parametric rate in n,

$$\text{EMSE}(\mathcal{P}, \hat{\tau}_1^{mn}) \leq Cn^{-1}.$$

In fact, as Theorem 5 shows, even if the number of control units is of the same order as the number of treated units, we can often achieve the parametric rate.

1.5 Applications

In this section, we consider two data examples. In the first example, we consider a large Get-Out-The-Vote (GOTV) experiment that explored if social pressure can be used to increase voter turnout in elections in the United States [27]. In the second example, we consider an experiment that explored if door-to-door canvassing can be used to durably reduce transphobia in Miami [9]. In both examples, the original authors failed to find evidence of heterogeneous treatment effects when using simple linear models without basis expansion, and subsequent researchers and policy makers have been acutely interested in treatment effect heterogeneity that could be used to better target the interventions. We use our honest random forest implementation [42] because of the importance of obtaining useful confidence intervals in these applications. Confidence intervals are obtained using a bootstrap procedure (Algorithm 10). We have evaluated several bootstrap procedures, and we have found that the results for all of them were very similar. We explain this particular bootstrap choice in detail in SI.3.

Social pressure and voter turnout

In a large field experiment, Gerber et al. show that substantially higher turnout was observed among registered voters who received a mailing promising to publicize their turnout to their neighbors [27]. In the United States, whether someone is registered to vote and their past voting turnout are a matter of public record. Of course, *how* individuals voted is private. The experiment has been highly influential both in the scholarly literature and in political practice. In our reanalysis, we focus on two treatment conditions: the control group, which was assigned to 191,243 individuals, and the "neighbor's" treatment group, which was assigned to 38,218 individuals. Note the unequal sample sizes. The experiment was conducted in Michigan before the August 2006 primary election, which was a statewide election with a wide range of offices and proposals on the ballot. The authors randomly assigned households with registered voters to receive mailers. The outcome, whether someone voted, was observed in the primary election. The "neighbors" mailing opens with a message that states "DO YOUR CIVIC DUTY—VOTE!" It then continues by not only listing the household's voting records but also the voting records of those living nearby. The mailer informed individuals that "we intend to mail an updated chart" after the primary.

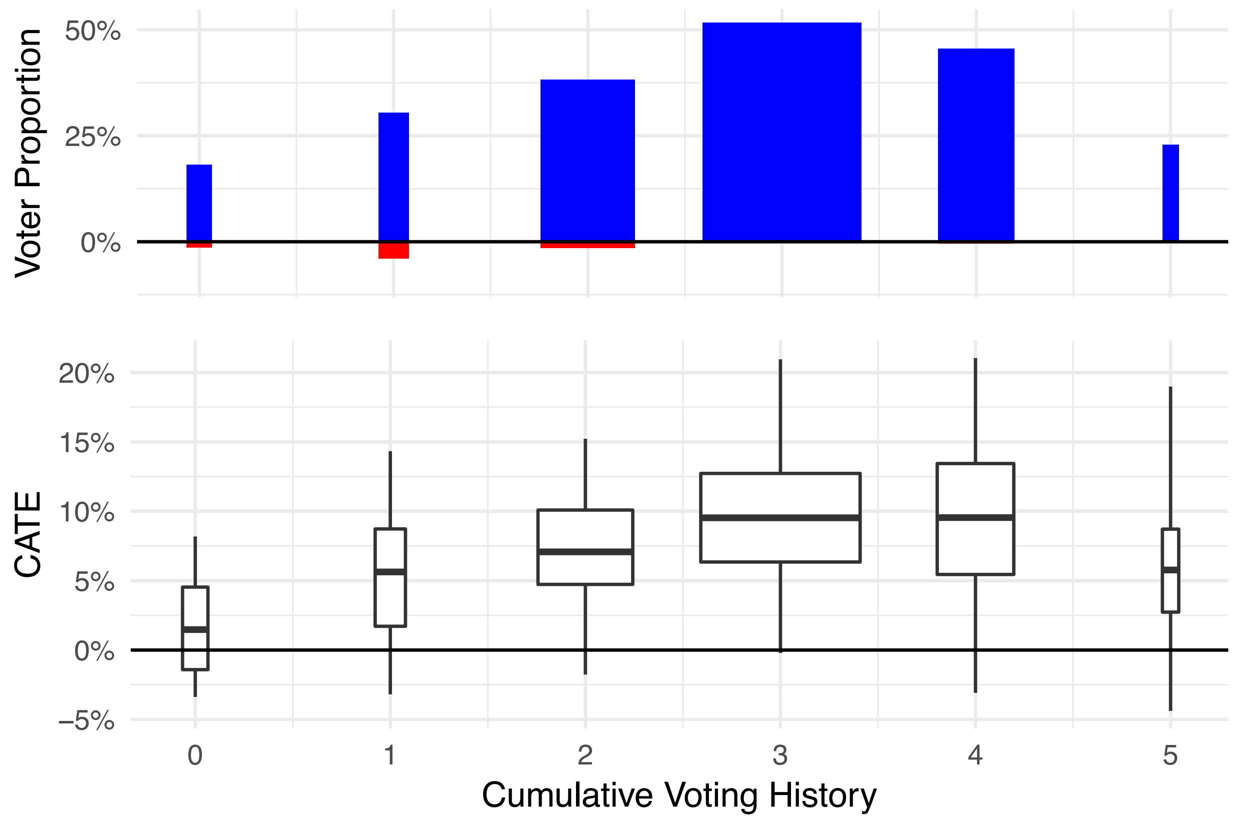


Figure 1.2: Social pressure and voter turnout. Potential voters are grouped by the number of elections they participated in, ranging from 0 (potential voters who did not vote during the past five elections) to 5 (voters who participated in all five past elections). The width of each group is proportional to the size of the group. Positive values in the first plot correspond to the percentage of voters for which the predicted CATE is significantly positive, while negative values correspond to the percentage of voters for which the predicted CATE is significantly negative. The second plot shows the CATE estimate distribution for each bin.

The study consists of seven key individual-level covariates, most of which are discrete: gender, age, and whether the registered individual voted in the primary elections in 2000, 2002, and 2004 or the general election in 2000 and 2002. The sample was restricted to voters who had voted in the 2004 general election. The outcome of interest is turnout in the 2006 primary election, which is an indicator variable. Because compliance is not observed, all estimates are of the Intention-to-Treat (ITT) effect, which is identified by the randomization. The average treatment effect estimated by the authors is 0.081 with a standard error of (0.003). Increasing voter turnout by 8.1% using a simple mailer is a substantive effect, especially considering that many individuals may never have seen the mailer.

Figure 1.2 presents the estimated treatment effects, using X–RF where the potential voters are grouped by their voting history. The upper panel of the figure shows the proportion of voters with a significant positive (blue) and a significant negative (red) CATE estimate. We can see that there is evidence of a negative backlash among a small number of people who voted only once in the past five elections prior to the general election in 2004. Applied researchers have observed a backlash from these mailers; e.g., some recipients called their Secretary of States office or local election registrar to complain [53, 55]. The lower panel shows the distribution of CATE estimates for each of the subgroups. Having estimates of the heterogeneity enables campaigns to better target the mailers in the future. For example, if the number of mailers is limited, one should target potential voters who voted three times during the past five elections, since this group has the highest average treatment effect and it is a very big group of potential voters.[4]

S–RF, T–RF, and X–RF all provide similar CATE estimates. This is unsurprising given the very large sample size, the small number of covariates, and their distributions. For example, the correlation between the CATE estimates of S–RF and T–RF is 0.99 (results for S–RF and T–RF can be found in Figure A.9).

We conducted a data-inspired simulation study to see how these estimators would behave in smaller samples. We take the CATE estimates produced by T–RF, and we assume that they are the truth. We can then impute the potential outcomes under both treatment and control for every observation. We then sample training data from the complete data and predict the CATE estimates for the test data using, S–, T–, and X–RF. We keep the unequal treatment proportion observed in the full data fixed, i.e., $\mathbb{P}(W = 1) = 0.167$. Figure 1.3 presents the results of this simulation. They show that in small samples both X–RF and S–RF outperform T–RF, with X–RF performing the best, as one may conjecture given the unequal sample sizes.

Reducing transphobia: A field experiment on door-to-door canvassing

In an experiment that received widespread media attention, Broockman et al. show that brief (10 minutes) but high-quality door-to-door conversations can markedly reduce prejudice against gender-nonconforming individuals for at least three months [9, 10]. There are important methodological differences between this example and our previous one. The experiment is a placebo-controlled experiment with a parallel survey that measures attitudes, which are the outcomes of interest. The authors follow the design of [11]. The authors first recruited registered voters ($n = 68,378$) via mail for an unrelated online survey to measure baseline outcomes. They then randomly assigned respondents of the baseline survey to either the treatment group ($n = 913$) or the placebo group that was targeted with a conversation about recycling ($n = 912$). Randomization was conducted at the household level ($n = 1295$),

[4]In praxis, it is not necessary to identify a particular subgroup. Instead, one can simply target units for which the predicted CATE is large. If the goal of our analysis were to find subgroups with different treatment effects, one should validate those subgroup estimates. We suggest either splitting the data and letting the X-learner use part of the data to find subgroups and the other part to validate the subgroup estimates, or to use the suggested subgroups to conduct further experiments.

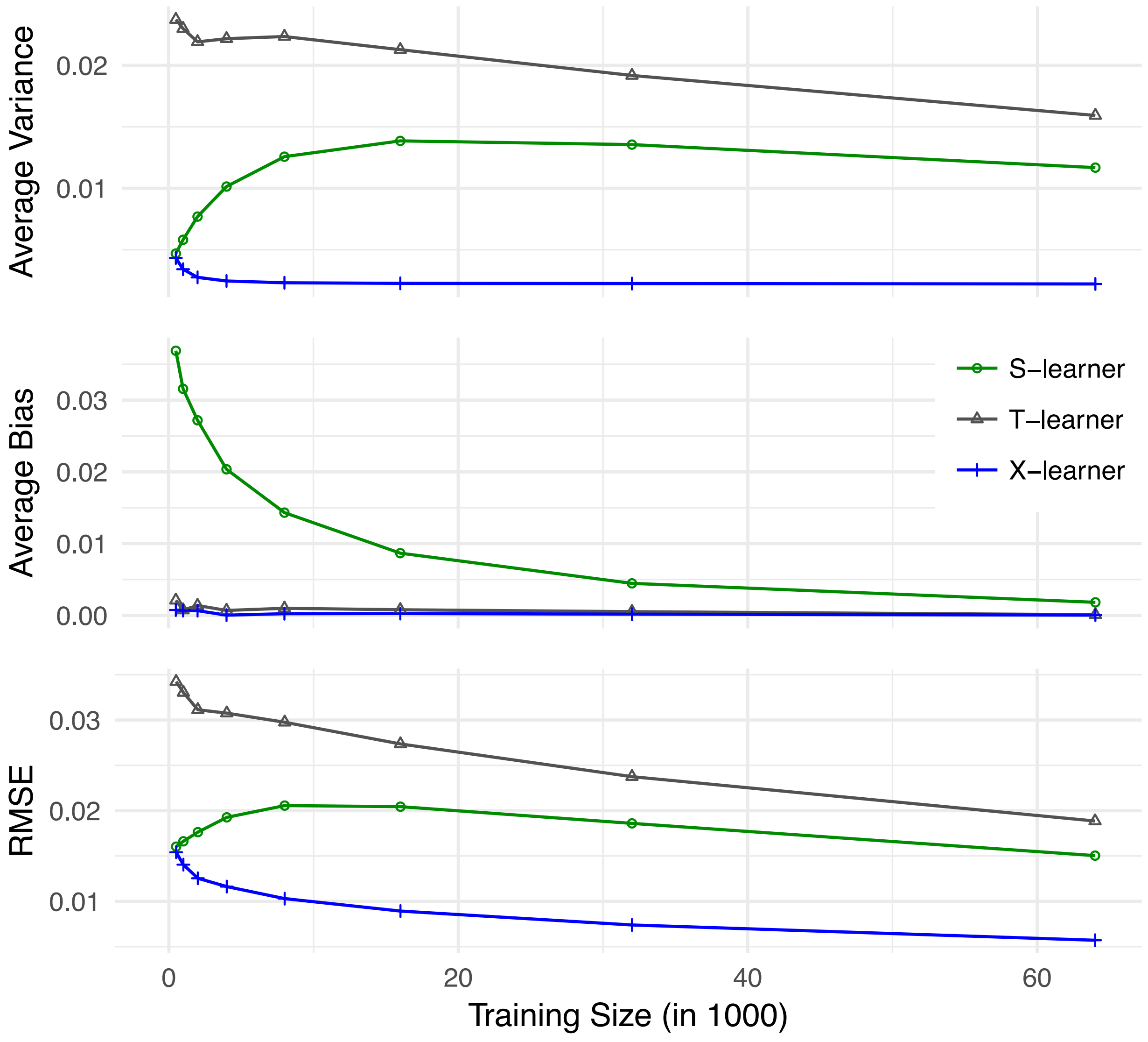


Figure 1.3: RMSE, bias, and variance for a simulation based on the social pressure and voter turnout experiment.

and because the design employs a placebo control, the estimand of interest is the complier-average-treatment effect. Outcomes were measured by the online survey three days, three weeks, six weeks, and three months after the door-to-door conversations. We analyze results for the first follow-up.

The final experimental sample consists of only 501 observations. The experiment was well powered despite its small sample size because it includes a baseline survey of respondents as well as post-treatment surveys. The survey questions were designed to have high over-time stability. The R^2 of regressing the outcomes of the placebo-control group on baseline covariates using OLS is 0.77. Therefore, covariate adjustment greatly reduces sampling variation. There are 26 baseline covariates that include basic demographics (gender, age, ethnicity) and baseline measures of political and social attitudes and opinions about prejudice in general and Miami's nondiscrimination law in particular.

The authors find an average treatment effect of 0.22 (SE: 0.072, t-stat: 3.1) on their transgender tolerance scale.[5] The scale is coded so that a larger number implies greater tolerance. The variance of the scale is 1.14, with a minimum observed value of -2.3 and a maximum of 2. This is a large effect given the scale. For example, the estimated decrease in transgender prejudice is greater than Americans' average decrease in homophobia from 1998 to 2012, when both are measured as changes in standard deviations of their respective scales.

The authors report finding no evidence of heterogeneity in the treatment effect that can be explained by the observed covariates. Their analysis is based on linear models (OLS, lasso, and elastic net) without basis expansions.[6] Figure 1.4(a) presents our results for estimating the CATE, using X–RF. We find that there is strong evidence that the positive effect that the authors find is only found among a subset of respondents that can be targeted based on observed covariates. The average of our CATE estimates is within half a standard deviation of the ATE that the authors report.

Unlike in our previous data example, there are marked differences in the treatment effects estimated by our three learners. Figure 1.4(b) presents the estimates from T–RF. These estimates are similar to those of X–RF, but with a larger spread. Figure 1.4(c) presents the estimates from S–RF. Note that the average CATE estimate of S–RF is much lower than the ATE reported by the original authors and the average CATE estimates of the other two learners. Almost none of the CATE estimates are significantly different from zero. Recall that the ATE in the experiment was estimated with precision, and was large both substantively and statistically (t-stat=3.1).

In this data, S–RF shrinks the treatment estimates toward zero. The ordering of the estimates we see in this data application is what we have often observed in simulations: the S-learner has the least spread around zero, the T-learner has the largest spread, and the X-learner is somewhere in between. Unlike in the previous example, the covariates are strongly

[5]The authors' transgender tolerance scale is the first principal component of combining five -3 to $+3$ Likert scales. See [9] for details.

[6][9] estimates the CATE using Algorithm 8.

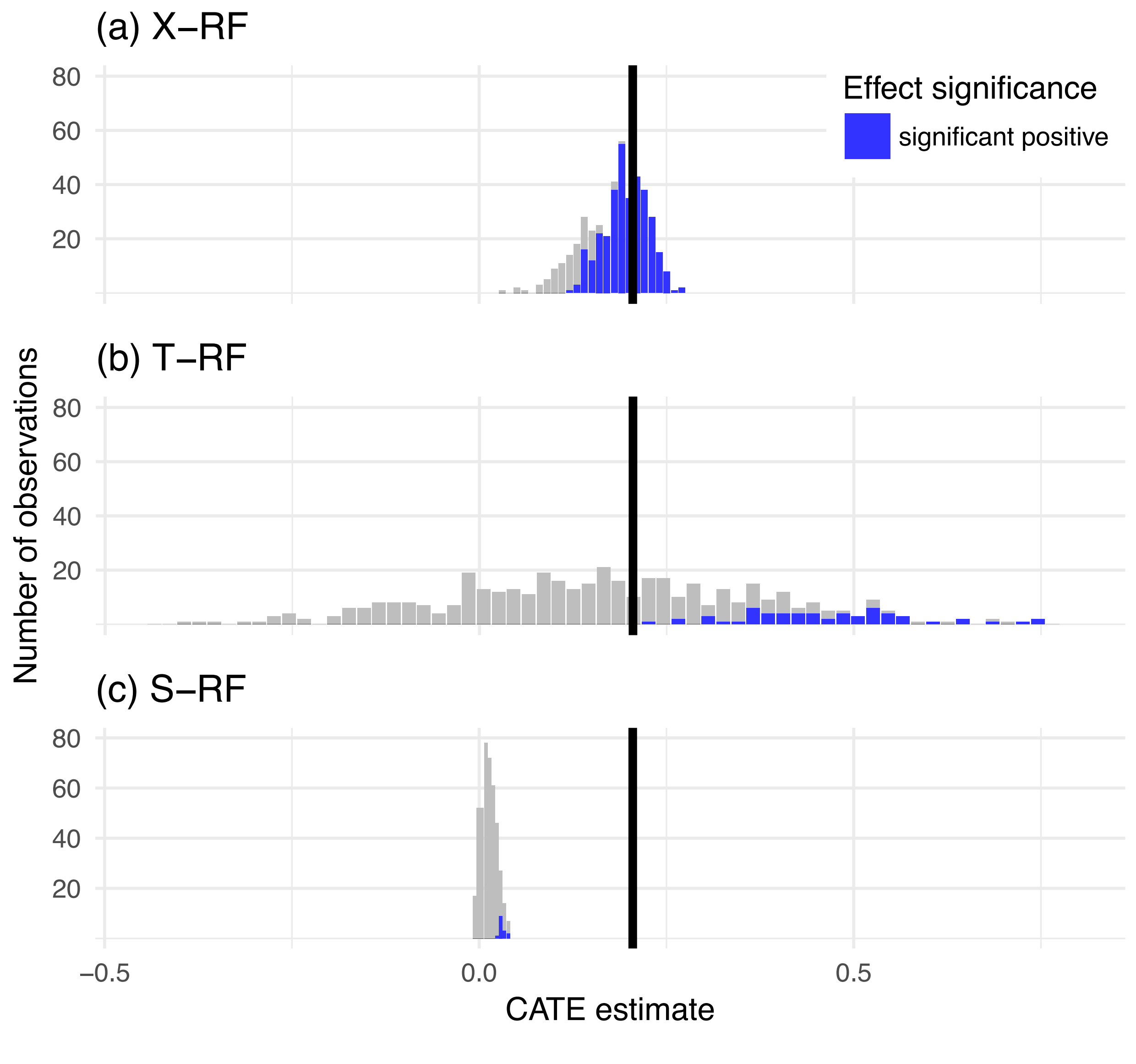


Figure 1.4: Histograms for the distribution of the CATE estimates in the Reducing Transphobia study. The horizontal line shows the position of the estimated ATE.

predictive of the outcomes, and the splits in the S–RF are mostly on the features rather than the treatment indicator, because they are more predictive of the observed outcomes than the treatment assignment (cf., Figure A.10).

1.6 Conclusion

This paper reviewed meta-algorithms for CATE estimation including the S- and T-learners. It then introduced a new meta-algorithm, the X-learner, that can translate any supervised learning or regression algorithm or a combination of such algorithms into a CATE estimator. The X-learner is adaptive to various settings. For example, both theory and data examples show that it performs particularly well when one of the treatment groups is much larger than the other or when the separate parts of the X-learner are able to exploit the structural properties of the response and treatment effect functions. Specifically, if the CATE function is linear, but the response functions in the treatment and control group satisfy only the Lipschitz-continuity condition, the X-learner can still achieve the parametric rate if one of the groups is much larger than the other (Theorem 2). If there are no regularity conditions on the CATE function and the response functions are Lipschitz continuous, then both the X-learner and the T-learner obtain the same minimax optimal rate (Theorem 7). We conjecture that these results hold for more general model classes than those in our theorems.

We have presented a broad set of simulations to understand the finite sample behaviors of different implementations of these learners, especially for model classes that are not covered by our theoretical results. We have also examined two data applications. Although none of the meta-algorithms is always the best, the X-learner performs well overall, especially in the real-data examples. In practice, in finite samples, there will always be gains to be had if one accurately judges the underlying data-generating process. For example, if the treatment effect is simple, or even zero, then pooling the data across treatment and control conditions will be beneficial when estimating the response model (i.e., the S-learner will perform well). However, if the treatment effect is strongly heterogeneous and the response surfaces of the outcomes under treatment and control are very different, pooling the data will lead to worse finite sample performance (i.e., the T-learner will perform well). Other situations are possible and lead to different preferred estimators. For example, one could slightly change the S-learner so that it shrinks to the estimated ATE instead of zero, and it would then be preferred when the treatment effect is constant and non-zero. One hopes that the X-learner can adapt to these different settings. The simulations and real-data studies presented have demonstrated the X-learner's adaptivity. However, further studies and experience with more real data sets are necessary. To enable practitioners to benchmark these learners on their own data sets, we have created an easy-to-use software library called `hte`. It implements several methods of selecting the best CATE estimator for a particular data set, and it implements confidence interval estimators for the CATE.

In ongoing research, we are investigating using other supervised learning algorithms. For example, we are creating a deep learning architecture for estimating the CATE that is based on the X-learner with a particular focus on transferring information between different data sets and treatment groups. Furthermore, we are concerned with finding better confidence intervals for the CATE. This might enable practitioners to better design experiments, and determine the required sample size before an experiment is conducted.

Chapter 2

CausalToolbox Package

In this section, we describe the first version of the `CausalToolbox` package. It has been designed to make heterogeneous treatment effect estimation simple by providing useful tools and functions in the ***R*** programming language. As of the writing of this thesis, it has not been published on the Comprehensive R Archive Network (CRAN) but it can be found on Github (`https://github.com/soerenkuenzel/causalToolbox/`).

We will first introduce the `CATEestimator` class. It is the main class of the package, and all other estimators are implemented as classes that inherit the `CATEestimator` class. We will show the basic syntax and principles that have guided us when implementing the package. We then demonstrate in an example that it is effortless to run many estimators at the same time. In Chapter 2.2, we will provide one additional method that can be used to select a good estimator for a given data set. We then conclude with a brief outlook on functionalities that we expect to release in Version 2.

2.1 CATE Estimators

The CATE estimators

In the following, we describe the different CATE estimators that are implemented in this package. The package only implements meta-learners, but it is possible to extend it to other estimators.

Meta-learners are algorithms that can be combined with any regression or machine learning algorithm (c.f. Chapter 1). We implemented the M-, S-, T-, and X-Learner and combined each of them with the Random Forests algorithm (RF) and the Bayesian Additive Regression Trees algorithm (BART). Thus, in total, we implemented eight different estimators. Each estimator is implemented as a class, and its name reveals which meta-learner and base learner it is based on. For example, `X_RF` is the `X-Learner` with RF as the base learner.

Chapter B in the Appendix contains the precise documentation of the different functions. In the following, we will only give a high-level explanation of the M-Learner, as the other meta-learners have been discussed in Chapter 1.

M-Learner

The M-Learner [65] estimates the CATE in three steps:

1. Estimate the response functions and the propensity score,

$$\begin{aligned}\mu_0(x) &= \mathbb{E}[Y(0)|X = x],\\ \mu_1(x) &= \mathbb{E}[Y(1)|X = x],\\ e(x) &= \mathbb{P}[W = 1|X = x],\end{aligned}$$

   using the base learner, and denote the estimates as $\hat{\mu}_0$, $\hat{\mu}_1$, and $\hat{e}$.

2. Define

$$\tilde{R}_i := \frac{W_i - \hat{e}(X_i)}{\hat{e}(X_i)[1 - \hat{e}(X_i)]}\Big(Y_i - \hat{\mu}_1(X_i)[1 - \hat{e}(X_i)] - \hat{\mu}_0(X_i)\hat{e}(X_i)\Big).$$

3. Now employ the base learner using $\tilde{R}_i$ as the dependent variable to obtain the CATE estimator.

Structure of the Classes

The fundamental class of this package is the `CATEestimator` class. For each derived class from `CATEestimator`, there exists at least two methods:

- `EstimateCate` - Estimates the Conditional Average Treatment Effect (CATE) for given features.
- `CateCI` - Estimates confidence intervals for the CATE using the i.i.d. bootstrap.

This unifying framework makes it very easy to switch between different estimators. Specifically, the same syntax can be used for any CATE estimator that is part of the package or an extension thereof. We will see how this can be used to iterate through different learners to find the best one and to understand how robust the CATE estimates are for the chosen learner [84]. The following example shows how the CATE can be estimated using two different CATE estimators. Note that the syntax is in both cases the same.

```
xl_rf <- X_RF(feat = feat, tr = tr, yobs = yobs)
sl_bt <- S_BART(feat = feat, tr = tr, yobs = yobs)

cate_est_xrf <- EstimateCate(xl_rf, feature_test)
cate_est_sbt <- EstimateCate(sl_bt, feature_test)
```

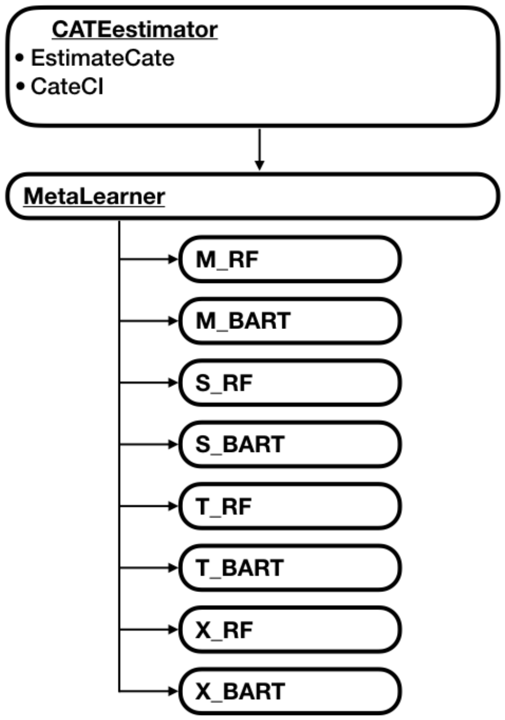


Figure 2.1: Package Structure of the main classes

Figure 2.1 shows the different CATE estimators that are implemented in the package and the structure of these estimators. Extending it to other estimators is straightforward by simply inheriting the `CATEestimator` class. For example, if one wanted to implement another estimator, it is enough to define the `EstimateCate` function. `CateCI` is then automatically inherited from the `CATEestimator` class. This avoids repetitive code, errors, and makes extensions to other estimators simple. It is, of course, also possible to override this option and implement a specialized method.

2.2 Evaluating the Performance of CATE Estimators

In a standard prediction task, researchers are interested in predicting the regression function, $\mu(x) = \mathbb{E}[Y|X = x]$, and they can use several strategies to evaluate the performance of an estimator. A popular strategy is to set aside a part of the data as a test set, S_V. An estimator, $\hat{\mu}$, can then be evaluated by its ability to minimize the prediction error for the test set,

$$\sum_{i \in S_V} \left(Y_i - \hat{\mu}_i\right)^2.$$

This package is concerned with estimating the CATE, $\tau(x) = \mathbb{E}[Y(1) - Y(0)|X = x]$. Naturally, one would also like to set aside a test set to evaluate a CATE estimator, $\hat{\tau}$. One would then want to calculate the prediction error for the individual treatment effects,

$$\sum_{i \in S_V} \big(Y_i(1) - Y_i(0) - \hat{\tau}_i\big)^2.$$

This, however, is impossible since for any unit in the training set, one either observes the outcome under treatment, $Y_i(1)$, or the outcome under control, $Y_i(0)$, but never both.

Nonetheless, there are several methods that try to overcome this problem to evaluate CATE estimators. In the following, we describe a method we have implemented to evaluate the CATE estimators.

Simulate Causal Experiments

The basic idea of our new method takes the features from the experimental data, and it simulates the observed outcome and the treatment assignment using a wide range of different setups. Since the outcomes are simulated, the true underlying data generating distribution and therefore, also the CATE is known. An estimator can then simply be evaluated by the prediction error for the CATE,

$$\sum_{i \in S_V} \big(\tau_i - \hat{\tau}_i\big)^2.$$

The Function `simulate_causal_experiment` implements more than 100 different setups. Each setup specifies the response functions, μ_0 and μ_1, and the propensity score, $e(x)$ and it can be used with almost any feature matrix. A function call generates a list of all the potential outcomes, the observed outcome, a treatment assignment, and a split in training and testing set.

The following code, for example, simulates an experiment with the features based on the `iris` data set and it then trains and evaluates the default versions of the X-learner with BART and RF.

```
ce <- simulate_causal_experiment(
  given_features = iris,
  pscore = "rct5",
  mu0 = "fullLinearWeak",
  tau = "fullLinearWeak")

xl_rf <- X_RF(feat = ce$feat_tr, tr = ce$W_tr, yobs = ce$Yobs_tr)
xl_bt <- X_BART(feat = ce$feat_tr, tr = ce$W_tr, yobs = ce$Yobs_tr)

mean((EstimateCate(xl_rf, ce$feat_te) - ce$tau_te)^2)
mean((EstimateCate(xl_bt, ce$feat_te) - ce$tau_te)^2)
```

In this particular example, we find that `X_BART` performs better than `X_RF`. This matches our intuition that for small data sets, estimators based on BART perform better than those based on RF.

There are many possible settings. In the example above, we used `pscore = "rct5"`, which specifies that the propensity score is constant and equal to 0.5. Furthermore, we chose `mu0="fullLinearWeak"` and `tau="fullLinearWeak"`. This determines that the response functions are both linear and given by the following equations:

$$\mu_0(x) = x^T \beta_0,$$
$$\mu_1(x) = \mu_0(x) + x^T \beta_T,$$

where β_0 and $\beta_T \in \mathbb{R}^d$ are independently sampled from a uniform distribution over $[-5, 5]^d$.

If we want to choose a well-performing estimator, it is essential to avoid making a decision based on only one simulation setup. It is, therefore, useful to be able to iterate through all setups. Assume, for example, we had a list of estimators, `estimators`, that extended the `CATEestimator` class, and we wanted to select the best of these estimators. The following code loops through the different setups.

```
for(tau_name in names(tau.simulate_causal_experiment)){
  for(mu0_name in names(mu0.simulate_causal_experiment)) {
    for(estimator in estimators) {
      ce <- simulate_causal_experiment(
        given_features = iris,
        pscore = "rct5",
        mu0 = mu0_name,
        tau = tau_name)

      trained_e <- estimator(ce$feat_tr, ce$W_tr, ce$Yobs_tr)
      performance[estimator$name, setup] <-
        mean((EstimateCate(trained_e, ce$feat_te) - ce$tau_te)^2)
    }
  }
}
```

Table 2.1 shows the `performance` table of the example code above averaged over 100 random seeds. We can see that the BART-based CATE estimators outperform the RF-based ones. It is, however, not obvious which estimator to choose, since `X_RF`, `S_BART`, and `X_BART` all perform well.

2.3 Version Two

This is only a brief description of the first version of the published package. Looking ahead, the goal of this package is to provide a comprehensive library that provides software for estimating, analyzing, and describing treatment effect heterogeneity.

For the second version, we have already implemented three goodness-of-fit tests that are based on matching [69], the modified outcome estimator [1], and the Robinson transform [62]. We believe the user should be able to choose between different methods. We want to connect them with the popular `caret` package to find ways to tune our estimators to the underlying data generating distribution quickly.

Another line of work focuses on confidence intervals. We implemented a customized version of the RF algorithm in a package called `forestry` that achieves provably valid confidence intervals under some conditions. However, we have noticed that these conditions are sometimes not met and are particularly problematic when the overlap condition (i.e., the propensity score is bounded away from 0 and 1) is violated. We have developed a novel method that indicates points for which we cannot provide reliable predictions and confidence intervals (c.f. Chapter 4).

Estimating the CATE is rarely the final goal. Researchers are often interested in understanding the underlying treatment mechanism. For the second version, we provide tools for better visualizing an estimated CATE function. We have already implemented a method to represent the data with a new regression tree algorithm that leads to shallow trees that can be used to understand the underlying mechanism better and to analyze how a particular prediction was made (c.f. Chapter 5).

μ_0	τ	**M_RF**	**S_RF**	**T_RF**	**X_RF**	**M_BT**	**S_BT**	**T_BT**	**X_BT**
SLW	const	0.25	0.44	0.52	**0.18**	1.33	0.21	0.94	0.26
SLW	SLW	1.53	1.21	**0.93**	2.8	2.23	1.12	1.43	1.21
SLW	FLW	1.5	5.23	1.47	3.04	2.49	1.38	1.61	**1.24**
SLW	FLLW	0.53	0.88	0.54	0.44	1.33	0.58	0.96	**0.34**
SLW	SNS3	0.25	0.41	0.52	0.18	1.33	**0.18**	0.94	0.29
SLS	const	4.72	50.73	24.31	**2.8**	10.88	3.6	32.61	6.11
SLS	SLW	5.15	58.75	24.01	**3.05**	13.74	9.5	36.5	9.79
SLS	FLW	9.8	62.89	29.12	8.81	12.94	8.3	34.06	**6.88**
SLS	FLLW	6.31	4.11	24.08	**3.6**	11.88	6.81	35.72	7.97
SLS	SNS3	4.55	10.08	24.2	**2.79**	10.88	3.37	32.61	7.45
FLW	const	0.15	0.59	0.49	**0.08**	0.54	0.17	1.78	0.22
FLW	SLW	1.52	2.1	2.03	2.87	0.8	1.32	2.02	**0.77**
FLW	FLW	1.76	1.71	**1.38**	2.57	2.16	1.49	2.95	1.39
FLW	FLLW	**0.39**	1.87	0.64	0.4	0.56	0.52	1.9	0.47
FLW	SNS3	0.15	0.94	0.5	**0.08**	0.54	0.14	1.78	0.2
FLLW	const	0.35	0.54	0.6	0.23	8.31	**0.15**	0.49	0.21
FLLW	SLW	1.65	1.52	1.29	3.07	8.12	1.01	0.9	**0.57**
FLLW	FLW	1.88	2.42	1.77	2.97	10.69	1.81	**1.31**	1.41
FLLW	FLLW	0.59	0.78	0.82	0.6	7.7	0.68	0.49	**0.28**
FLLW	SNS3	0.35	0.58	0.6	0.23	8.31	**0.13**	0.49	0.28
FLWS	const	0.21	0.57	0.77	**0.16**	0.73	0.29	1.02	0.25
FLWS	SLW	1.53	1.57	1.44	2.85	0.92	0.89	1.02	**0.56**
FLWS	FLW	1.78	2.57	1.84	3.01	1.51	1.56	1.35	**1.23**
FLWS	FLLW	0.39	2.43	0.82	0.44	0.65	0.53	0.96	**0.35**
FLWS	SNS3	0.21	2.23	0.77	**0.15**	0.73	0.19	1.02	0.19
SNS1	const	0.2	0.25	0.31	0.13	3.36	**0.11**	0.22	0.18
SNS1	SLW	1.52	1.16	0.88	2.8	4.65	0.99	**0.57**	0.81
SNS1	FLW	1.57	1.76	1.16	2.59	4.41	1.56	**0.99**	1.06
SNS1	FLLW	0.41	0.42	0.42	0.46	3.28	0.64	**0.36**	0.41
SNS1	SNS3	0.2	0.27	0.31	0.13	3.36	**0.07**	0.22	0.18
SNS2	const	0.2	0.32	0.37	0.15	2.74	**0.14**	0.38	0.2
SNS2	SLW	1.55	1.38	1.15	2.9	3.67	1.05	**0.67**	0.69
SNS2	FLW	1.54	1.55	1.14	2.71	3.53	1.39	1.05	**0.94**
SNS2	FLLW	0.42	**0.38**	0.43	0.5	2.6	0.66	0.45	0.46
SNS2	SNS3	0.2	0.33	0.37	0.15	2.74	**0.12**	0.38	0.18
SNS3	const	0.19	0.26	0.3	0.12	3.22	**0.11**	0.21	0.16
SNS3	SLW	1.53	1.21	0.88	2.8	4.23	0.98	**0.56**	0.61
SNS3	FLW	1.55	1.74	1.12	2.58	4.59	1.48	**0.94**	1.1
SNS3	FLLW	0.41	0.41	0.41	0.46	3.24	0.57	0.35	**0.28**
SNS3	SNS3	0.19	0.27	0.3	0.12	3.22	**0.07**	0.21	0.15

Table 2.1: Simulation of 40 RCTs to evaluate the eight implemented CATE estimators.

Chapter 3

Causaltoolbox—Estimator Stability for Heterogeneous Treatment Effects

3.1 Introduction

Heterogeneous Treatment Effect (HTE) estimation is now a mainstay in many disciplines, including personalized medicine [34, 59], digital experimentation [75], economics [2], political science [29], and statistics [76]. Its prominence has been driven by the combination of the rise of big data, which permits the estimation of fine-grained heterogeneity, and recognition that many interventions have heterogeneous effects, suggesting that much can be gained by targeting only the individuals likely to experience the most positive response. This increase in interest amongst applied statisticians has been accompanied by a burgeoning methodological and theoretical literature: there are now many methods to characterize and estimate heterogeneity; some recent examples include [35, 2, 45, 80, 58]. Many of these methods are accompanied by guarantees suggesting they possess desirable properties when specific assumptions are met, however, verifying these assumptions may be impossible in many applications; so practitioners are given little guidance for choosing the best estimator for a particular data set. As an alternative to verifying these assumptions we suggest practitioners construct a large family of HTE estimators and consider their similarities and differences.

Treatment effect estimation contrasts with prediction problems, where researchers can use cross-validation (CV) or a validation set to compare the performance of different estimators or to combine them in an ensemble. This is infeasible for treatment effect estimation because of the fundamental problem of causal inference: we can never observe the treatment effect for any individual unit directly, so we have no source of truth to validate or cross-validate against. Partial progress has been made in addressing this problem; for example, [2] suggest using the transformed outcome as the truth, a quantity equal in expectation to the individual treatment effect and [45] suggests using matching to impute a quantity similar to the unobserved potential outcome. However, even if there were a reliable procedure for identifying the estimator with the best predictive performance, we maintain that using multiple estimates can still be superior, because the best performing method or ensemble of methods may perform well in some regions of the feature space and badly in others; using many estimates simultaneously may permit identification of this phenomenon. For example, researchers can construct a worst-case estimator that is equal to the most pessimistic point estimate, for each point in the feature space, or they can use the idea of stability [83] to assess whether one can trust estimates for a particular subset of units.

3.2 Methods

Study setting

The data set we analyzed was constructed for the Empirical Investigation of Methods for Heterogeneity Workshop at the 2018 Atlantic Causal Inference Conference. The organizers of the workshop: Carlos Carvalho, Jennifer Hill, Jared Murray, and Avi Feller used the National Study of Learning Mindsets, a randomized controlled trial in a probability sample of U.S. public high schools, to simulate an observational study. The organizers did not disclose how the simulated observational data were derived from the experimental data because the workshop was intended to evaluate procedures for analyzing observational studies, where the mechanism of treatment assignment is not known *a priori*.

Measured variables

The outcome was a measure of student achievement; the treatment was the completion of online exercises designed to foster a learning mindset. Eleven covariates were available for each student: four are specific to the student and describe the self-reported expectations for success in the future, race, gender and whether the student is the first in the family to go to college; the remaining seven variables describe the school the student is attending measuring urbanicity, poverty concentration, racial composition, the number of pupils, average student performance, and the extent to which students at the school had fixed mindsets; an anonymized school id recorded which students went to the same school.

Notation and estimands

For each student, indexed by i, we observed a continuous outcome, Y_i, a treatment indicator variable, Z_i, that is 1 if the student was in the treatment group and 0 if she was in the control group, and a feature vector X_i. We adopt the notation of the Neyman-Rubin causal model: for each student we assume there exist two potential outcomes: if a student is assigned to treatment we observe the outcome $Y_i = Y_i(1)$ and if the student is assigned to control we observe $Y_i = Y_i(0)$. Our task was to assess whether the treatment was effective and, if so whether the effect is heterogeneous. In particular, we are interested in discerning if there is a subset of units for which the treatment effect is particularly large or small.

To assess whether the treatment is effective, we considered the average treatment effect,

$$\text{ATE} := \mathbb{E}[Y_i(1) - Y_i(0)],$$

and to analyze the heterogeneity of the data, we computed average treatment effects for a selected subgroup S,

$$\mathbb{E}[Y_i(1) - Y_i(0) | X_i \in S],$$

and the Conditional Average Treatment Effect (CATE) function,

$$\tau(x) := \mathbb{E}[Y_i(1) - Y_i(0) | X_i = x].$$

Estimating average effects

Wherever we computed the ATE or the ATE for some subset, we used four estimators. Three of which were based on the `CausalGAM` package of [28]. This package uses generalized additive models to estimate the expected potential outcomes, $\hat{\mu}_0(x) := \hat{\mathbb{E}}[Y_i(0)|X_i = x]$, and $\hat{\mu}_1(x) := \hat{\mathbb{E}}[Y_i(1)|X_i = x]$, and the propensity score: $\hat{e}(x) := \hat{\mathbb{E}}[Z_i|X_i = x]$. With these estimates we computed the Inverse Probability Weighting (IPW) estimator,

$$\hat{\text{ATE}}_{\text{IPW}} := \frac{1}{n} \sum_{i=1}^{n} \left(\frac{Y_i Z_i}{\hat{e}_i} - \frac{Y_i(1 - Z_i)}{1 - \hat{e}_i} \right),$$

the regression estimator,

$$\hat{\text{ATE}}_{\text{Reg}} := \frac{1}{n} \sum_{i=1}^{n} \left[\hat{\mu}_1(X_i) - \hat{\mu}_0(X_i) \right],$$

and the Augmented Inverse Probability Weighted (AIPW) estimator,

$$\hat{\text{ATE}}_{\text{AIPW}} := \frac{1}{2n} \sum_{i=1}^{n} \left(\frac{[Y_i - \hat{\mu}_0(X_i)]Z_i}{\hat{e}_i} + \frac{[\hat{\mu}_1(X_i) - Y_i][1 - Z_i]}{1 - \hat{e}_i} \right).$$

We also used the `Matching` package of [69] to construct a matching estimator for the ATE. Matches were required to attend the same school as the student to which they were matched and be assigned to the opposite treatment status. Among possible matches satisfying these criteria we selected the student minimizing the Mahalonobis distance on the four student specific features.

Characterizing heterogeneous treatment effects

In any data set there might be some units where estimators significantly disagree; when this happens, we should not trust any estimate unless we understand why certain estimates are

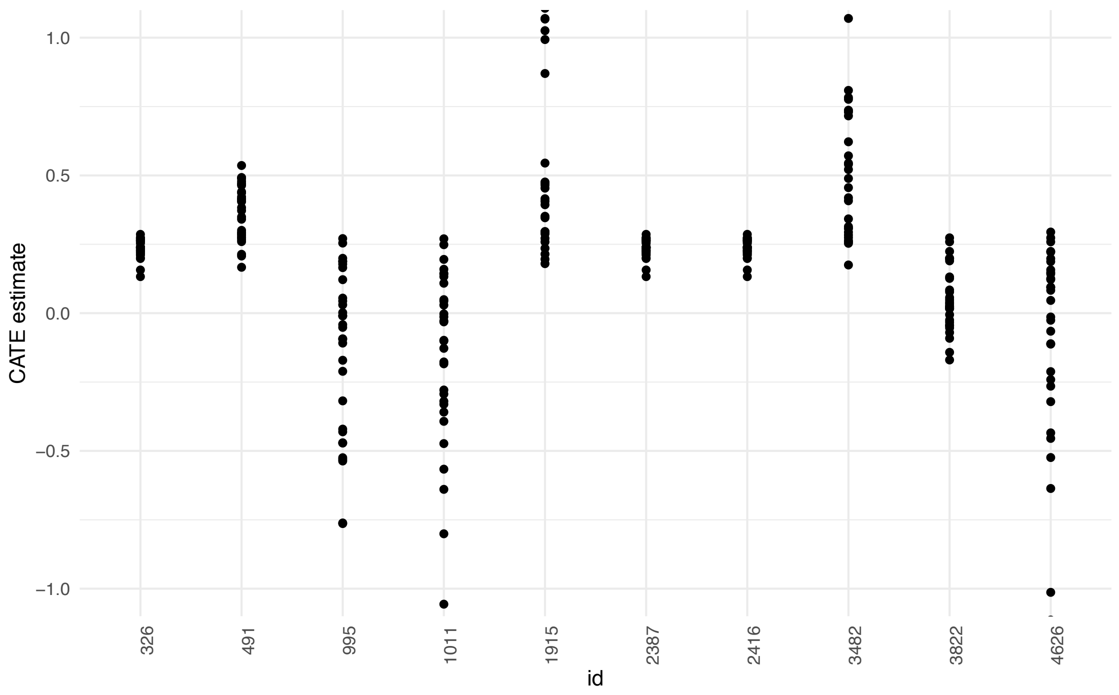


Figure 3.1: CATE estimation for ten units. For each unit, the CATE is estimated using 28 different estimators.

unreasonable for these units[1]. Instead of simply reporting an estimate that is likely wrong, we should acknowledge that a conclusions cannot be drawn and more data or domain specific knowledge is needed. Figure 3.1 demonstrates this phenomenon arising in practice. It shows the estimated treatment effect for ten subjects corresponding to 28 CATE estimators (these estimates arise from the data analyzed in the remainder of this paper). Some of these estimators may have better generalization error than others. However, a reasonable analyst could have selected any one of them. We can see that for five units the estimators all fall in a tight cluster, but for the remaining units, the estimators disagree markedly. This may be due to those units being in regions with little overlap, where the estimators overcome data scarcity by pooling information in different ways.

In this analysis, our goal was to understand and interpret the heterogeneity that exists in the treatment response. An estimate of the CATE function describes the heterogeneity. However, this estimate is hard to interpret, and drawing statistically significant conclusions based on it is difficult. Therefore, we sought large subgroups with a markedly different

[1]A standard way to capture estimation uncertainty is to report the standard errors or confidence intervals of a single estimator, and we recommend using this approach as well. However, such methods can be misleading and should not be trusted blindly. For example, in Appendix C of [45], the authors found that in regions without overlap, bootstrap confidence intervals were smaller than in regions with overlap. The confidence intervals suggested that in regions without overlap the estimates were more trustworthy, while the opposite was was true.

average treatment effects to help characterize the heterogeneity.

Specifically, we split the data into an **exploration set** and an equally sized **validation set**. We used the exploration set to identify subsets, which may have a very different behavior from the rest of the students. To do this, we used all CATE estimators trained on the exploration set and we carefully formulated hypotheses based on plots of all the CATE estimates: For example, based on plots of the CATE estimates we might theorize that students in schools with more than 900 students have a much higher treatment effect than those in schools with less than 300 students. Next, we used the validation set to verify our findings by estimating the ATEs of each of the subgroups.

The exploration and validation sets were constructed so that all students from the same school are in the same set: we do not randomize on student level, but on school-level. This is important, because it mirrors the probability sampling approach used to construct the full sample; it also means that we can argue that the estimand captured by evaluating our hypotheses on the validation set is the estimand corresponding to the population from which all schools were drawn.

CATE estimators

We use several procedures to estimate the CATE and we give a brief overview of the procedures here; however, interested readers should consult the referenced papers for a complete exposition.

Many of the procedures can be classified as meta-learners: they modify base learners designed for standard non-causal prediction problems to estimate the CATE. This is advantageous because we can select a base learner that performs well on the observed data.

1. The **T-Learner** is the most common meta-learner. Base learners are used to estimate the control and treatment response function separately, $\hat{\mu}_1(x) := \hat{\mathbb{E}}[Y_i(1)|X_i = x]$ and $\hat{\mu}_0(x) := \hat{\mathbb{E}}[Y_i(0)|X = x]$. The CATE estimate is then the difference between these two estimates, $\hat{\tau}^T(x) := \hat{\mu}_1(x) - \hat{\mu}_0(x)$.

2. The **S-Learner** uses one base learner to estimate the joint outcome function, $\hat{\mu}(x, z) := \hat{\mathbb{E}}[Y_i|X_i = x, Z_i = z]$. The predicted CATE is the difference between the predicted values when the treatment assignment indicator is changed from treatment to control, $\hat{\tau}^S(x) := \hat{\mu}(x, 1) - \hat{\mu}(x, 0)$.

3. The **MO-Learner** [65, 81] is a two stage meta-learner. It first uses the base learners to estimate the propensity score, $\hat{e}(x) := \hat{\mathbb{E}}[Z_i|X = x]$, and the control and treatment response functions. It then defines the adjusted modified outcome as

$$R_i := \frac{Z_i - \hat{e}(x_i)}{\hat{e}(x_i)[1 - \hat{e}(x_i)]}\Big(Y_i - \hat{\mu}_1(x_i)[1 - \hat{e}(x_i)] - \hat{\mu}_0(x_i)\hat{e}(x_i)\Big).$$

An estimate of the CATE is obtained by using a base learner to estimate the conditional expectation of R_i given X_i, $\hat{\tau}^{MO}(x) := \hat{\mathbb{E}}[R_i|X_i = x]$.

4. The **X-Learner** [45] also uses base learners to estimate the response functions and the propensity score. It then defines the imputed treatment effects for the treatment group and control group separately as $\tilde{D}_i^1 := Y_i(1) - \hat{\mu}_0(X_i)$ and $\tilde{D}_i^0 := \hat{\mu}_1(X_i) - Y_i(0)$. The two estimators for the CATE are obtained by using base learners to estimate the conditional expectation of the imputed treatment effects, $\hat{\tau}_1^X := \hat{\mathbb{E}}[\tilde{D}_i^1|X_i = x]$, and $\hat{\tau}_0^X := \hat{\mathbb{E}}[\tilde{D}_i^0|X_i = x]$. The final estimate is then a convex combination of these two estimators,
$$\hat{\tau}^X(x) := \hat{e}(x)\hat{\tau}_0^X(x) + (1 - \hat{e}(x))\hat{\tau}_1^X(x).$$

All of these meta-learners have different strength and weaknesses. For example, the T-Learner performs particularly well when the control and treatment response function are simpler than the CATE. The S-Learner performs particularly well when the expected treatment effect is mostly zero or constant. The X-Learner, on the other hand, has very desirable properties when either the treatment or control group is much larger than the other group.

Note, however, that all of these meta-learners need a base learners to be fully defined. We believe that tree-based estimators perform well on mostly discrete and low-dimensional data sets. Therefore, we use the `causalToolbox` package [41] that implements all of these estimators combined with RF and BART.

Using two different tree estimators is desirable because CATE estimators based on BART perform very well when the data-generating process has some global structure (e.g., global sparsity or linearity), while random forest is better when the data has local structure that does not necessarily generalize to the entire space. However, to protect our analysis from biases caused by using tree-based approaches only, we also included methods based on neural networks. We followed [46] and implemented the S, T and X-Neural Network methods.

We also included non-meta-learners that are tree-based, and we believe would work well on this data set:

5. The **causal forest** algorithm [80] is a generalization of the random forest algorithm to estimates the CATE directly. Similar to random forest, it is an ensemble of many tree estimators. Each of the tree estimators follows a greedy splitting strategy to generate leaves for which the CATE function is as homogeneous as possible. The final estimate for each tree for a unit with features x is the difference-in-means estimate of all units in the training set that fall in the same leaf as x.

6. The **R-Learner** [58] is a set of algorithms that use an approximation of the following optimization problem to estimate the CATE,
$$\arg\min_{\tau} \left\{ \frac{1}{n} \sum_{i=1}^{n} \Big(\big(Y_i - \hat{\mu}^{(-i)}(X_i)\big) - \big(W_i - \hat{e}^{(-i)}(X_i)\big)\,\tau(X_i) \Big)^2 + \Lambda_n(\tau(\cdot)) \right\}.$$
$\Lambda_n(\tau(\cdot))$ is a regularizer and $\mu^{(-i)}(x)$ and $\hat{e}^{(-i)}(X_i)$ are held-out predictions of $\mu(x) = \mathbb{E}[Y_i|X_i = x]$ and the propensity score, $e(x)$, respectively. There are several versions

of the R-Learner; we have decided to use one that is based on XGBoost [14] and one that is based on RF.

Although we expected there would be school-level effects, and that both the expected performance of each student and the CATE would vary from school to school, it was not clear how to incorporate the school id. The two choices we considered were to include a categorical variable recording the school id, or to ignore it entirely. The former makes parameters associated with the six school-level features essentially uninterpretable because they cannot be identified separately from the school id; the second may lead to less efficient estimates because we are denying our estimation procedure the use of all data that was available to us. Because we do not want our inference to depend on this decision, we fit each of our estimators twice, once including school id as a feature and once excluding it. We considered 14 different CATE estimation procedures; since each procedure was applied twice, a total of 28 estimators were computed.

3.3 Workshop Results

Our sample consisted of about 10,000 students enrolled at 76 different schools. The intervention was applied to 33% of the students. Pre-treatment features were similar in the treatment and control groups but some statistically significant differences were present. Most importantly a variable capturing self-reported expectations for success in the future had mean 5.22 (95% CI, 5.20-5.25) in the control group and mean 5.36 (5.33-5.40) in the treatment group. This meant students with higher expectations of achievement were more likely to be treated.

We assessed whether overlap held by fitting a propensity score model and we found that propensity score estimate for all students in the study was between 0.15 and 0.46 therefore, the overlap condition is likely to be satisfied.

Average treatment effects

The IPW, regression, and AIPW estimator yielded estimates identical up to two significant figures: 0.25 with 95% bootstrap confidence interval of $(0.22, 0.27)$. The matching estimator gave a similar ATE estimate of 0.26 with confidence interval $(0.23, 0.28)$.

The similarity of all the estimates we evaluated is reassuring, but we cannot exclude the possibility that the experiment is affected by an unobserved confounder that affects all estimators in a similar away. To address this we characterize the extent of hidden bias required to explain away our conclusion. We conducted a sensitivity analysis for the matching estimator using the `sensetivitymv` package of [63]. We found that a permutation test for the matching estimator still finds a significant positive treatment effect provided the ratio of the odds of treatment assignment for the treated unit relative to the odds of treatment assignment for the control unit in each pair can be bounded by 0.40 and 2.52. This bound is not very large, and it is plausible that there exists an unobserved confounder that increases the treatment assignment probability for some unit by a factor of more than 2.52.

More information about the treatment assignment mechanism would be required to conclude whether this extent of confounding exists.

Heterogeneous effects

The marginal distribution and partial dependence plots for the 28 CATE estimators as a function of school-level pre-existing mindset norms are shown on the left hand side of Figure 3.2. There appears to be substantial heterogeneity present: students at schools with mindset norms lower than 0.15 may have a larger treatment effect than students at schools with higher mindset norms. However the Figure suggests the conclusion is not consistent for all of the 28 estimators. A similar analysis of the feature recording the school achievement level is shown on the right hand side of this Figure. Again we appear to find the existence of heterogeneity: students with school achievement level near the middle of the range had the most positive response to treatment. On the basis of this figure, we identified thresholds of -0.8 and 1.1 for defining a low achievement level, a middle achievement level, and a high achievement level subgroup.

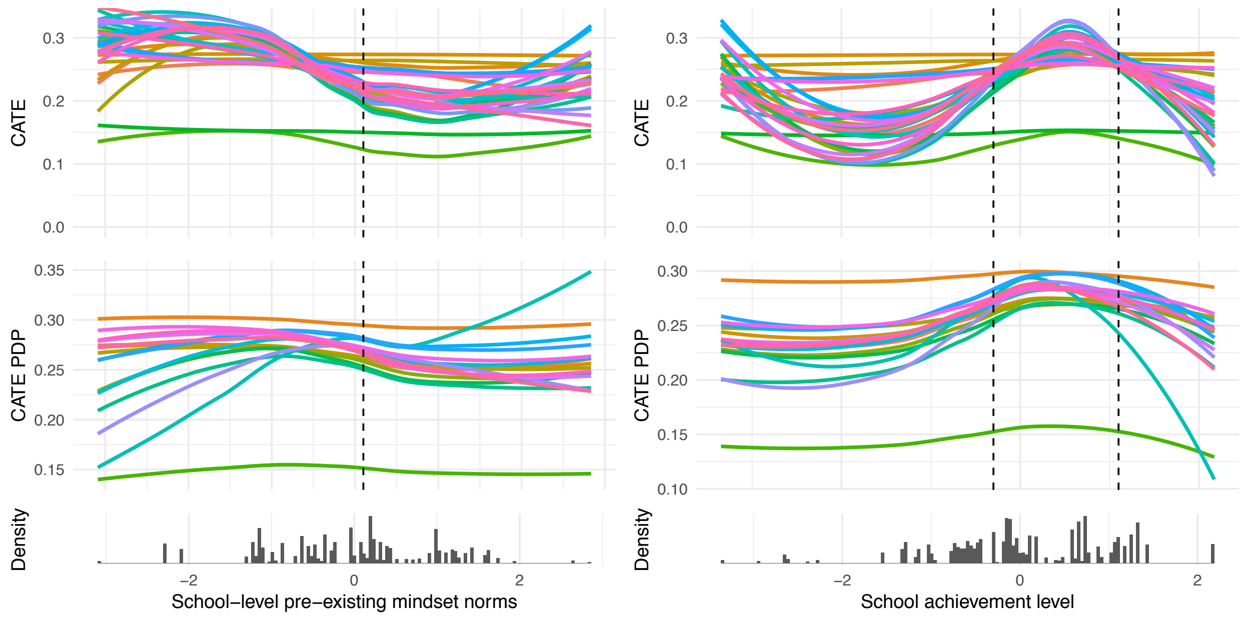


Figure 3.2: Marginal CATE and Partial Dependence Plot (PDP) of the CATE as a function of school-level pre-existing mindset norms and school achiemvent level.

We then used the validation set to construct ATE estimates for each of the subgroups. We found that students who attended schools where the measure of fixed mindsets was less than 0.15 had a higher treatment effect (0.31, 95% CI 0.26-0.35) than students where the fixed mindset was more pronounced (0.21, 0.17–0.26). Testing for equality of the ATE for these two groups yielded a p-value of 0.003. However, when we considered the subsets defined by school achievement level the differences were not so pronounced. Students at the lowest achieving schools had the smallest ATE estimate 0.19 (0.10–0.32); while students at middle and high-achieving schools had similar ATE estimates: 0.28 (0.24–0.32) and 0.24 (0.16-0.31) respectively. However, none of the pairwise difference between the three groups were significant.

3.4 Postworkshop results

During the workshop, other contributors found that the variable recording the urbanicity of the schools might explain some of the heterogeneity and we want to analyze this phenomena in the following. The left hand side of Figure 3.3 shows the CATE as a function of urbanicity, and the right hand side of this figure shows the CATE as a function of the student's self-reported expectation of success. We formulated two hypothesis: students at schools with an urbanicity of 3 seemed to have a lower treatment effect than students at other schools; students with a self-reported evaluation of 4 might enjoy a higher treatment effect.

These hypotheses were obtained by only using the exploration set; to confirm or refute these hypotheses we used the validation set. The validation set confirmed the hypothesis that students at schools with an urbanicity of 3 had a lower treatment effect (0.16, 0.08–0.24) compared to students at schools with a different urbanicity (0.28, 0.25–0.31); however we could not reject the null hypothesis of no difference for the subsets identified by the self-reported evaluation measure. The urbanicity test yielded a p-values of 0.008 and the self-reported evaluation test yielded a p-value of 0.56.

3.5 Discussion

The importance of considering multiple estimators

The results of our analysis confirm that point estimates of the CATE can differ markedly depending on subtle modelling choices; so we confirm that an analyst's discretion may be the deciding factor in whether and what kind of heterogeneity is found. As the methodological literature on heterogeneous treatment effect estimation continues to expand this problem will become more, not less, serious. To facilitate applying many estimation procedures we have authored an R package `causalToolbox` that provides a uniform interface for constructing many common heterogeneous treatment estimators. The design of the package makes it straightforward to add new estimators as they are proposed and gain currency.

Differences that arise in our estimation of the CATE function translate directly into suboptimal real world applications of the treatment considered. To see this we propose a thought experiment: suppose we wanted to determine the treatment for a particular student: a natural treatment rule is to allocate her to treatment if her estimated CATE exceeds a small positive threshold or withhold treatment if it is below the threshold. The analyst

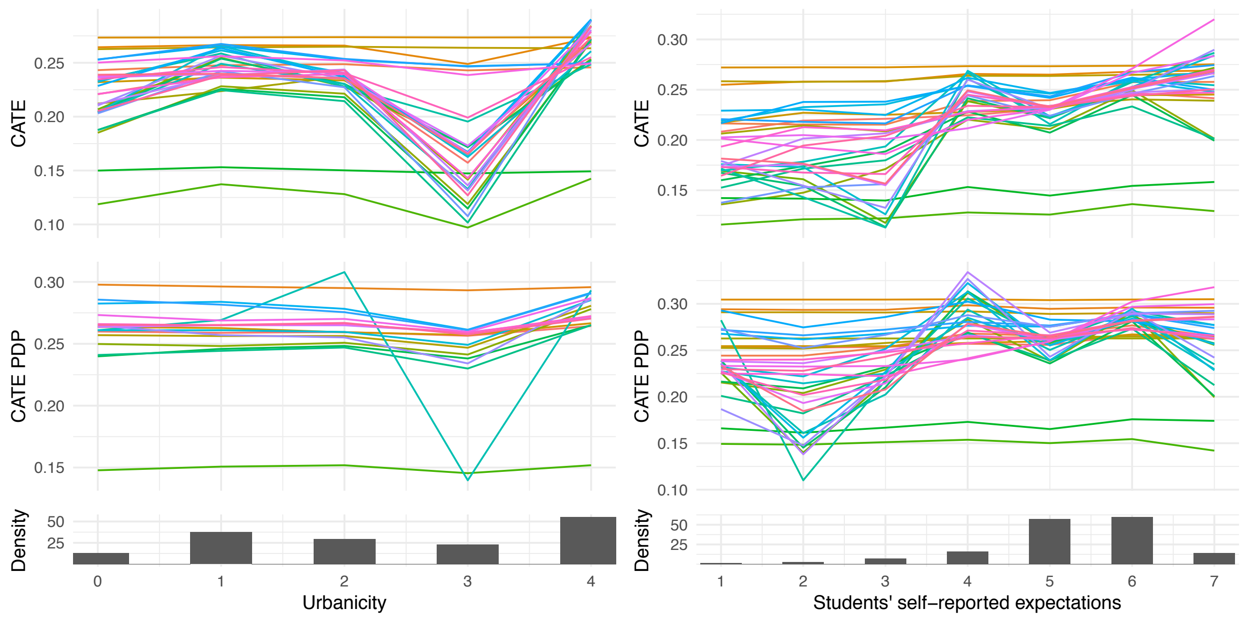


Figure 3.3: Marginal CATE and PDP of Urbanicity and self-reported expectations.

might select a CATE estimator on the basis of personal preference or prior experience and it is likely that, for some experimental subjects, the choice of estimator will affect the CATE estimated to such an extent that it changes the treatment decision. This is particularly problematic in studies where analysts have a vested interest in a particular result and are working without a pre-analysis plan, as they should not have discretion to select a procedure that pushes the results in the direction they desire. On the other hand, if analysts consider a wide variety of estimators, as we recommend, and if most estimators agree for an individual, we can be confident that our decision for that individual is not a consequence of arbitrary modelling choices. Conversely, if some estimators predict a positive and some a negative response, we should reserve judgment for that unit until more conclusive data is available and admit that we do not know what the best treatment decision is.

Would we recommend the online exercises?

We find that the overall effect of the treatment is significant and positive. We were not able to identify a subgroup of units that had significant and negative treatment effect and we would therefore recommend the treatment for every student. We are, however, concerned that an unobserved confounder exists. Our sensitivity analysis showed that our findings would still hold if the confounder is not too strong. We cannot exclude the possibility that there is a strong confounder and would have to know more about the assignment mechanism to address this question. This is particularly problematic, because we have seen that students who had higher expectations for success in the future were more likely to be in the treatment group. Uncovering the heterogeneity in the CATE function proved to be substantially more difficult. We found heterogeneity could be identified from school-level pre-exising mindset norms and urbanicity but in general we had limited power to detect heterogeneous effects. For example, experts believe that the heterogeneity might be moderated by pre-existing mindset norms and school-level achievement. For both covariates, we see that most CATE estimators produce estimates that are consistent with this theory. Domain experts also believe that there could be a "Goldilocks effect" where middle-achieving schools have the largest treatment effect. We are not able to verify this statistically, but we do observe that most CATE estimators describe such an effect.

Part II

Statistical Inference based on Random Forests

Chapter 4

Detachment Index for Evaluating Trustworthiness of Confidence Intervals and Predictons in Random Forests

4.1 Introduction

In Chapter 1, we studied several estimators for the Conditional Average Treatment Effect (CATE) and found that all of them struggled to estimate valid Confidence Intervals (CI) for the CATE in regions where either control units or treated units were missing. This problem is particularly significant in observational studies (c.f. Chapter A.3) where the propensity score can be a function of observed or—even worse—unobserved covariates.

In the following, we introduce the detachment index that measures whether a data point is close to the underlying data set. We find that methods such as Random Forests (RF) and Bayesian Additive Regression Trees (BART) perform poorly in CI estimation tasks for regression and CATE estimation tasks on points with an extraordinarily large detachment index. We generalize the problem of CI estimation by allowing estimators to either estimate an interval at a point or to report a "DO NOT TRUST" flag.[1] This flag is useful in many real world tasks. For example, it is useful if some points in the test data are outside of the support of the training data or if crucial assumptions that are necessary for correct CI estimation are violated.

We do not believe that we have found a general solution to the problem of providing CI for the CATE in observational studies. We urge practitioners to be extremely cautious when analyzing CATE estimates in observational studies since a lot of problems can occur. The problem we are addressing in this chapter arises when the overlap condition is violated or the test units are outside of the support of the training set. In fact, we argue that these

[1]A similar idea has been studied in [18].

problems are similar because in both cases the estimators have to generalize in regions where either control units, treated units, or both are missing.

The detachment index is not only useful in CATE estimation tasks, but also in general regression tasks. To motivate the detachment indices, we first introduce a regression task (Example A), where some of the test points are unusual. We will use this example in Chapter 4.2 to motivate the detachment indices. In Chapter 4.3, we will then analyze two different problems: In the first one (Example B), we study a CATE estimation problem where the overlap condition is violated. We find that the detachment indices allow us to identify regions where the overlap condition is violated. In the second part (Example C), we analyze a low-dimensional regression problem where the test data is not in the support of the training data. We are able to identify points in the test set for which a prediction and CI should not be trusted because the covariates are outside of the support of the training set.

Example A: Predicting Car Prices

We will use the following example to motivate the detachment indices and show how well they can be used to find units in the test set where predictions and CIs based on RF should not be trusted. The data set is from Ebay-Kleinanzeigen (German Ebay listings) and contains 151,547 car offerings. In each offering a car is described using eleven features. Table 4.1 and 4.2 summarize these continuous and discrete features. It should be noted that due to the many categories contained in *Model* and *Brand*, the prediction task is challening. If we encoded all categories using indicators, the data set would have 316 features. Figure 4.1 shows a histogram of the price tags of the cars. We can see that there are a few cars that are very expensive, even though most cars are offered for less than 10,000 Euros.

Days Ago Created	Registration D.	Horse Power	Kilometer Driven	ZIP Code
Min. : 748.0	Min. :1950	Min. : 1.0	Min. : 5000	Min. : 1067
1st Qu.: 757.0	1st Qu.:2000	1st Qu.: 82.0	1st Qu.:100000	1st Qu.:31303
Median : 765.0	Median :2004	Median :116.0	Median :150000	Median :50969
Mean : 765.1	Mean :2004	Mean :127.7	Mean :124919	Mean :51767
3rd Qu.: 773.0	3rd Qu.:2008	3rd Qu.:150.0	3rd Qu.:150000	3rd Qu.:72531
Max. :1132.0	Max. :2019	Max. :579.0	Max. :150000	Max. :99998

Table 4.1: Continuous features of the cars data set.

Our goal is to predict the offered price. To validate an estimator, we have split the data into a training set of 51,548 cars and a held-out test set of 100,000 cars. We have trained a RF estimator on the training data set and found that the overall Root Mean Squared Error (RMSE) is about 4,479. However, most of the RMSE is driven by very few observations. If we simply exclude the 1% of the cars with the highest prediciton error, then the RMSE is

Vehicle Type	Gearbox	Model	Fuel Type	Brand
limousine :43939	automatik: 34488	golf : 12043	benzin :94973	vw :31981
kleinwagen:34670	manuell :114942	andere : 10897	diesel :48478	bmw :17609
kombi :31167	unknown : 2117	3er : 9032	unknown: 5279	opel :15458
bus :14145		polo : 5038	lpg : 2356	mercedes:15036
cabrio :10848		unknown: 5016	cng : 256	audi :14281
coupe : 8687		corsa : 4670	hybrid : 122	ford : 9966
(Other) : 8091		(Other):104851	(Other): 83	(Other) :47216

Table 4.2: Discrete features of the Cars data set.

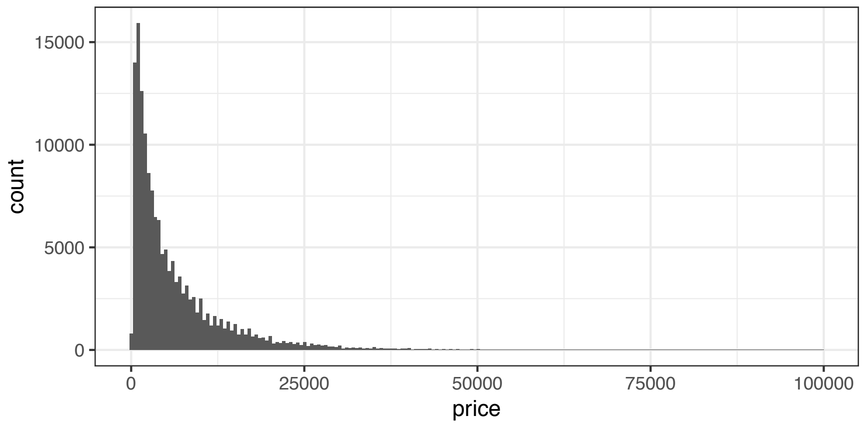


Figure 4.1: Histogram of the price distirbution in the cars data set.

would drop to 2,972. Of course, in a usual prediciton task one does know which predictions are the worst.

Figure 4.2 shows the absolute error divided by the true value. We can see that there are regions where the RF estimator performs particularly poorly. Most cars that are sold on the platform are very popular cars and have prices that are rather easy to predict. These cars are usually between 1 and 8 years old, and their horsepower is not extremely large. There are, however, some cars that are rare and very hard to predict. Some examples are vintage cars, racing cars, and specialty cars. For these cars, the RF algorithm often fails to predict a good price. We will develop a new method that can automatically flag units for which the RF estimates perform particularly poorly. More specifically, we will see that it identifies "unusual" cars. Estimates of the RF algorithm are not reliable for these cars. This is also reflected in the unreliability of CI.

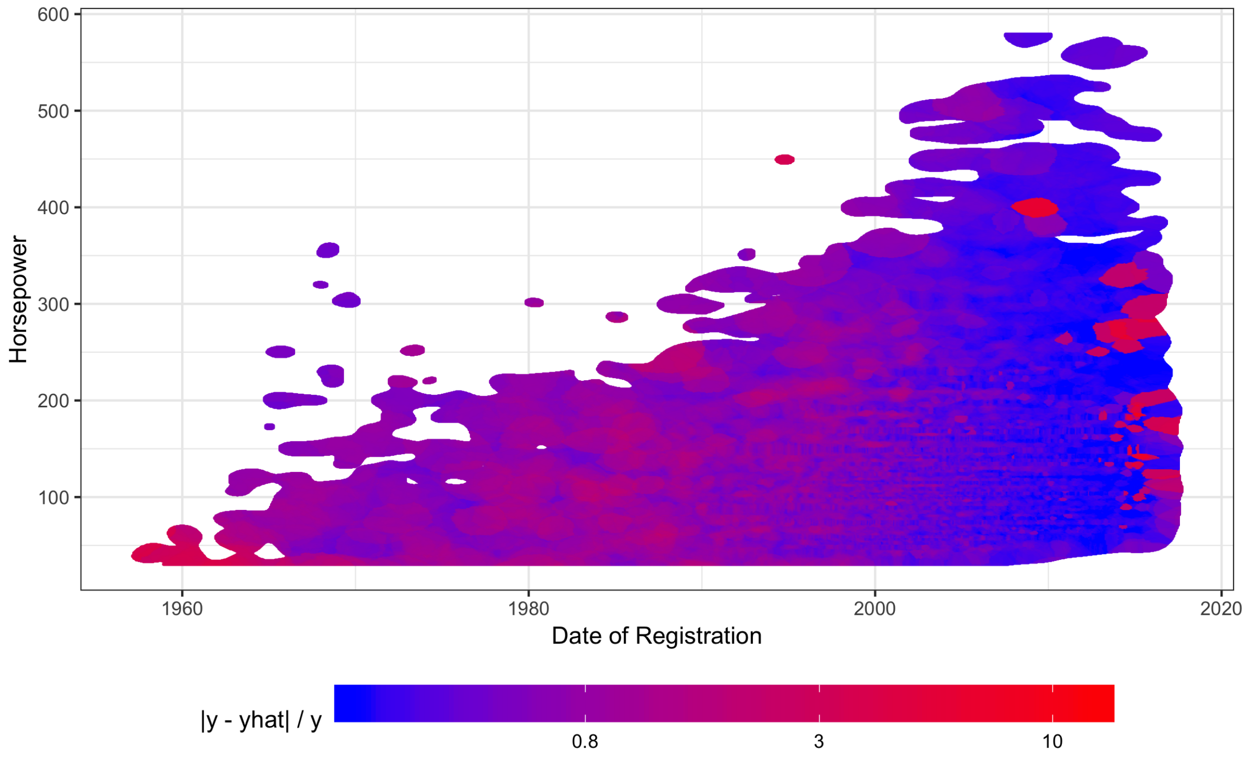
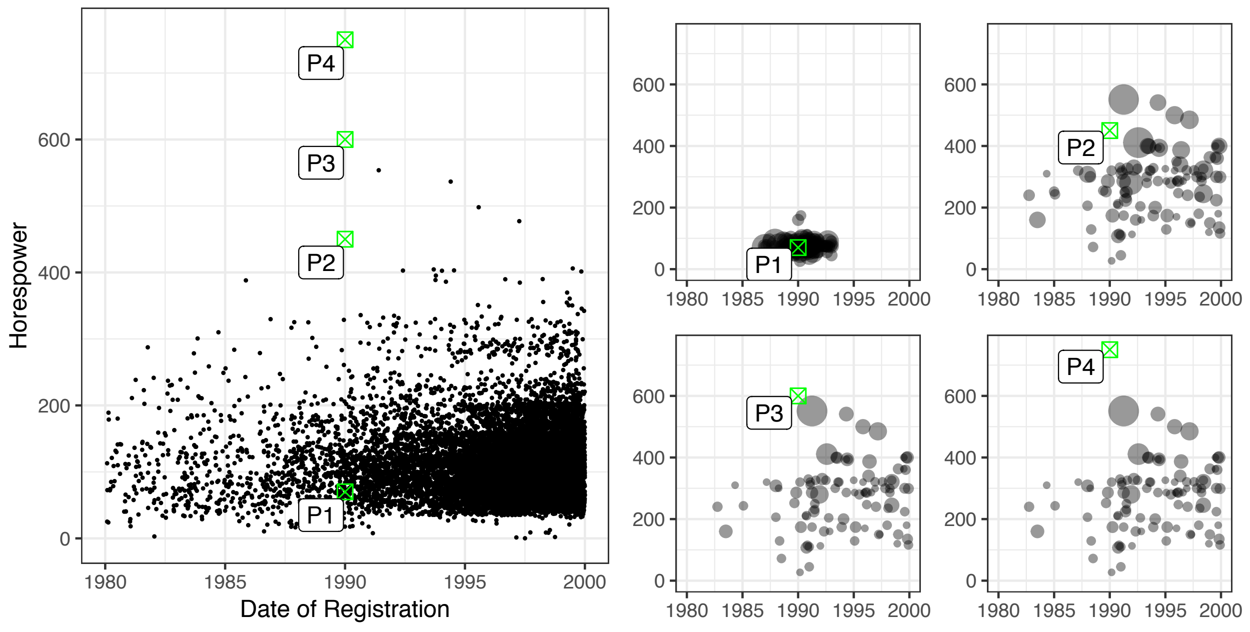


Figure 4.2: Projection of the support of the cars data set onto the two most predictive covariates as chosen by the default RF variable importance measure [6]. The colors show the absolute residuals of the RF estimates divided by the true price.

4.2 The RF-detachment Index

Motivation

The idea of the RF-detachment index is partly based on ideas by [51] and [72] to interpret random forests as an adaptive, nearest neighbors algorithm. Let us recall their interpretation of the random forests algorithm. Consider a training data set with n units, $(Y_i, X_i)_{i=1}^n$, and a random forests object with B trees. We use the notation that $L_b(x)$ is the leaf in which x falls in Tree b, and $|L_b(x)|$ is the number of training units in that leaf. We can then write



(a) Scatter plot of the first two dimensions (b) RF-weights $w(x)$ for $P1$, P2, P3, P4.

Figure 4.3: Scatterplot and RF-weights for the four points in the cars data set.

the random forests prediction $\hat{\mu}(x)$ as,

$$\hat{\mu}(x) = \frac{1}{B}\sum_{b=1}^{B}\sum_{i=1}^{n} Y_i \frac{1\{X_i \in L_b(x)\}}{|L_b(x)|} \tag{4.1}$$

$$= \sum_{i=1}^{n} Y_i \frac{1}{B}\sum_{b=1}^{B} \frac{1\{X_i \in L_b(x)\}}{|L_b(x)|} \tag{4.2}$$

$$= \sum_{i=1}^{n} Y_i w_i(x). \tag{4.3}$$

Here, $w_i(x) = \sum_{b=1}^{B} \frac{1\{X_i \in L_b(x)\}}{|L_b(x)|}$ can be interpreted as a local weighting function that is given by the random forest algorithm.

Note that for every x, the weights assigned to the n training units are usually very different. To better understand the local weighting function, we have plotted it for four points in Figure 4.3. P1 is an undamaged VW Golf with 70 horsepower and a manual gear that has been created in January 1990. P2, P3, and P4 are the same car, but we artificially set the strength of the engines to be 450, 600, and 750 horsepower respectively. Those are unreasonably large numbers, and we can see that in the left part of Figure 4.3 there was no offering of a car with 750 Horsepower in the training set.

Now let us take a look at the weight functions of the four cars. The weight function assigns positive weight to units that are in some relevant metric close to the point x. We can see that for point P1, the positive weight is very concentrated around point P1. This is because P1 is a relatively common car. There are many VW Golfs with about the same power and date of registration. For points P2, P3, and P4, the weight function is much further spread out. This is because there are very few, if any, cars that are similar to these artificial cars. Even more interestingly, the weight functions for P3 and P4 are the same and they are very similar to the one of P2. This is because P3 and P4 always fall in exactly the same leaf in every tree. Point predictions and confidence intervals for P3 and P4 are exactly the same and very similar to those of P2.

The Detachment Index

As motivated in the preceding section, a prediction is potentially untrustworthy if RF averages over data points that are dissimilar from the new data point x_{new} in relevant dimensions. This motivates us to define the RF-detachment index that measures the average distance of the point distance of the point x_{new} to the training points with a positive RF weight, w.

Definition 4 *For a covariate $1 \leq j \leq d$ and a point $x \in \mathbb{R}^d$, define the RF-detachment index in the jth covariate as*

$$d^j(x) = \sum_{i=1}^{n} w_i(x) \left| X_i^j - x^j \right|,$$

where X_i^j and x^j are the jth covariate of vector X_i and x, respectively.

In the cars example, we know that the *Horsepower* covariate is the most important covariate as measured by the default RF variable importance test [6]. We find that the RF-detachment indices in the *Horsepower* covariate are 9.9 for P1, 116.6 for P2, 236.5 for P3, and 386.5 for P4. Unsurprisingly, the detachment index is increasing from P1 to P4. Intuitively, one would be more cautious when using RF for inference on P4 instead of P1.

However, to flag a prediction or a confidence interval for being not trustworthy, we need a threshold or rule that determines whether a detachment index is too large. Deriving such a rule is difficult and depends on the use of the index. We propose two different ways to determine whether a unit is detached.

Threshold 1: CV-Error Calibration

In our first approach, we use a global quantity to find a threshold that can be applied to all units. More precisely, we want to find a threshold t and label all units with a detachment index of greater than t as "detached" and all units with a detachment index of less than t as "trustworthy."

Let us first demonstrate this in a one-dimensional example based on the *Horsepower* covariate in the cars data set: Using Cross-Validation, we first compute the detachment index and the CV-prediction errors for the observations in the training set. We then choose

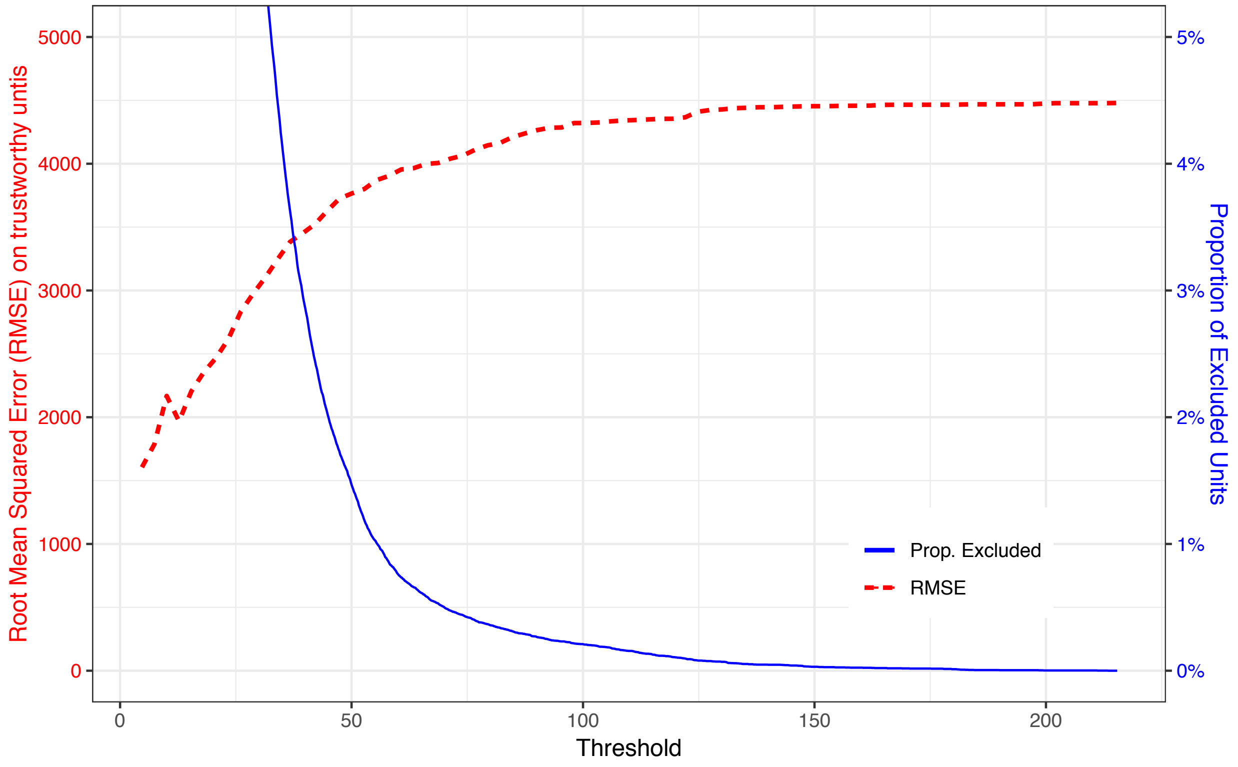


Figure 4.4: RMSE vs proportion of excluded units for a given threshold.

the threshold t in such a way that we do not exclude too many units while we try to minimize the RMSE on the "trustworthy" units. Figure 4.4 shows this trade off for the cars data set and the *Horespower* covariate. The blue, solid line shows that increasing the threshold leads to fewer units being excluded while the RMSE (red, dashed line) is being increased. For example, if we chose a threshold of $t = 32$, we expect to exclude only about 5% of the test units while decreasing the RMSE from 4,479 to about 3400.[2]

However, usually there is more than one variable. One has to select a threshold for each variable. For example, in the cars data set there are eleven covariates. To avoid dealing with categorical variables, we will choose a threshold for each of the five continuous covariates (c.f., Table 4.1).[3] Table 4.3 shows a summary of the detachment indices for the five continuous covariates. We can see that the indices are small for most units in the training set, and the 3rd quantile is up to 50 times smaller than the maximum. This suggests that most of the units are in a dense area, but there are still some outliers.

The goal is to find a threshold for each of the five continuous covariates of the cars data set. Let us call these thresholds t_1, t_2, t_3, t_4, and t_5. We want to choose them in such a way that the number of "trustworthy" units does not fall below a predefined threshold M, but

[2]Instead of optimizing the RMSE, it is also possible to evaluate Confidence Intervals.

[3]We refer to the `forestry` package for a treatment of the categorical variables.

Days Ago Created	Registration D.	Horse Power	Kilometer Driven	ZIP Code
Min. : 4.117	Min. : 0.4393	Min. : 5.727	Min. : 136.4	Min. : 7729
1st Qu.: 5.892	1st Qu.: 1.1558	1st Qu.: 11.619	1st Qu.: 841.7	1st Qu.:14973
Median : 6.392	Median : 1.6782	Median : 13.933	Median : 2077.2	Median :16324
Mean : 6.578	Mean : 2.0629	Mean : 16.925	Mean : 6367.1	Mean :16762
3rd Qu.: 7.003	3rd Qu.: 2.2624	3rd Qu.: 18.800	3rd Qu.:11289.8	3rd Qu.:17984
Max. :355.982	Max. :34.1566	Max. :204.668	Max. :54380.9	Max. :34993

Table 4.3: Summary of the OOB detachment indices for the continuous covariates.

the MSE for the "trustworthy" units is as small as possible. We can formulate this via the following optimization problem:

$$\begin{aligned} \underset{t\in\mathbb{R}^5}{\text{minimize}} \quad & \frac{1}{\#S(t)} \sum_{i\in S(t)} \left(Y_i - \tilde{Y}_i\right)^2 \\ \text{subject to} \quad & \#S(t) \geq M \end{aligned} \tag{4.4}$$

where we choose $M = 95\% * 51,548 \approx 48,971$. $\tilde{Y}_i$ is the CV-prediction of Y_i, and $S(t)$ is the set of trusted units:

$$S(t) := \left\{i \mid d^j(X_i) < t_j \text{ for all } j \in \{1,\ldots,5\}\right\}.$$

While it is an interesting problem on its own, it is not necessary to find the exact optimizer of this problem. In the following, we outline a simple approximation which has shown promising results in preliminary simulation experiments. In this particular example, we have sampled 100,000 potential threshold vectors (t_1, t_2, t_3, t_4, and t_5) and have evaluated their performance. Table 4.4 shows the five best vectors, $t \in \mathbb{R}^5$, as evaluated by the objective in Equation (4.4).[4]

First of all, it is interesting to note that *Date of Registration* and *Power* are the only relevant features. The others appear to matter relatively little since all their thresholds are larger than the 99.95% quantile. This is in line with our subject knowledge that suggests that *Power* is the most predictive covariate followed by *Date of Registration.* Furthermore, when we apply this five-dimensional threshold, we find that we can decrease the total RMSE in the test set from about 4,479 to about 3,052 by labeling slightly more than 4.7 % of the data as "detached." Figure 4.5 shows the distribution of the data points that are labeled as "detached" versus those that are labeled as "trustworthy."

Threshold 2: Neighborhood Calibration

Using the same threshold for every observation has worked well for the cars data set. However, some data sets exhibit a very different behaviour on a local and on a global scale. In

[4]Table C.1 in the Appendix contains the 40 best thresholds.

Days Ago C.	Registration D.	Horse Power	Kilometer Dr.	ZIP Code	RMSE
50.7 (100%)	9.3 (98.8%)	37.1 (96.1%)	47155.5 (100%)	30653.6 (100%)	2981.65
137.1 (100%)	9.2 (98.8%)	38.1 (96.3%)	47496.9 (100%)	30915.4 (100%)	2984.89
50.6 (100%)	7.7 (98.1%)	41.5 (97.2%)	34628.5 (99.5%)	34944.2 (100%)	2988.02
58.5 (100%)	7.3 (98.0%)	40.8 (97.1%)	52064.3 (100%)	31871.9 (100%)	2988.98
59.4 (100%)	7.9 (98.2%)	41.1 (97.1%)	36530.9 (99.6%)	32554.2 (100%)	2989.33

Table 4.4: Best 5 thresholds out of a sample of 100,000. The number in brackets is the corresponding percentile.

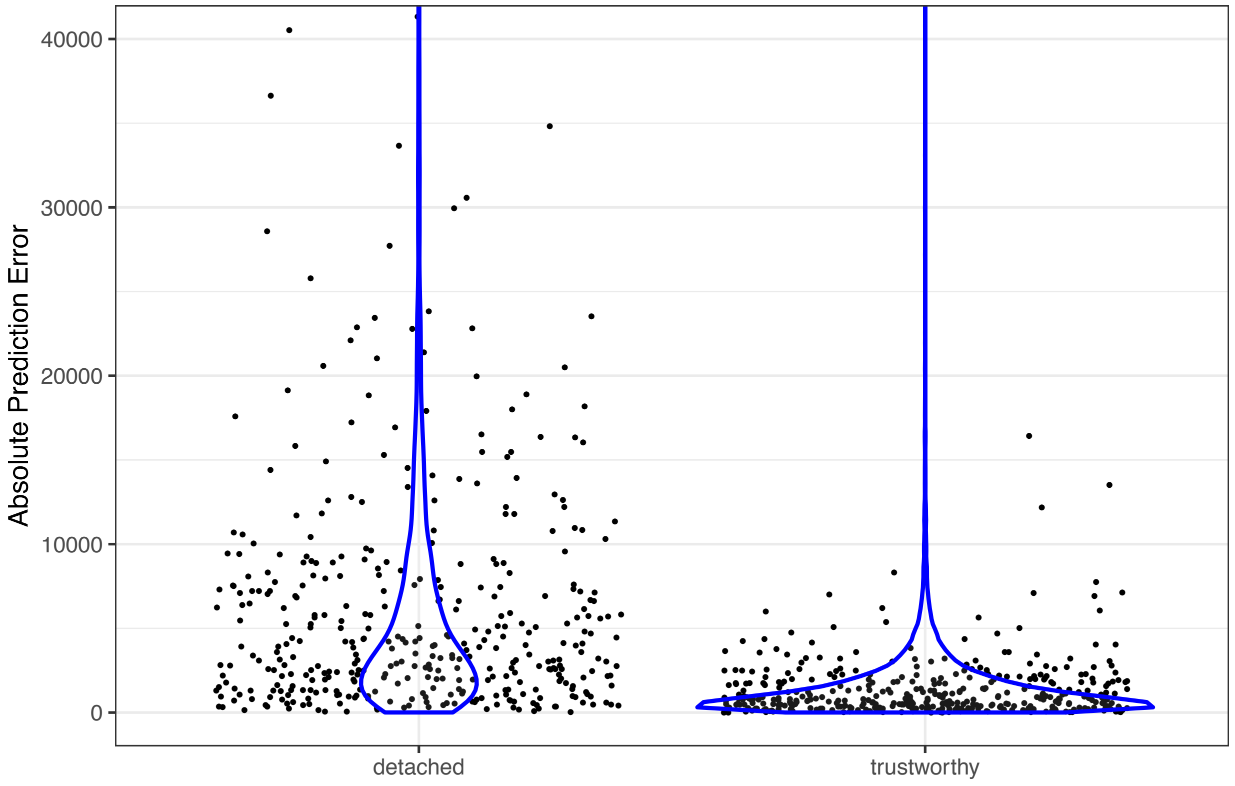


Figure 4.5: Error Distribution of detached and trustworthy test points.

this section, we want to discuss another approach that computes an individualized threshold for every unit. Recall the four cars, P1, P2, P3, and P4, that we have introduced in the beginning of this section. Figure 4.6 shows the detachment indices for *Horsepower* for each of the four points and the distribution of the OOB detachment indices for the neighbors of the four points.

It is immediately clear that P1 not only has a small detachment index, but it is also not particularly extreme compared to its neighbors. Interestingly, P2 has a large detachment index, but it is not very extreme, since about 12 % of its neighbors have even more extreme

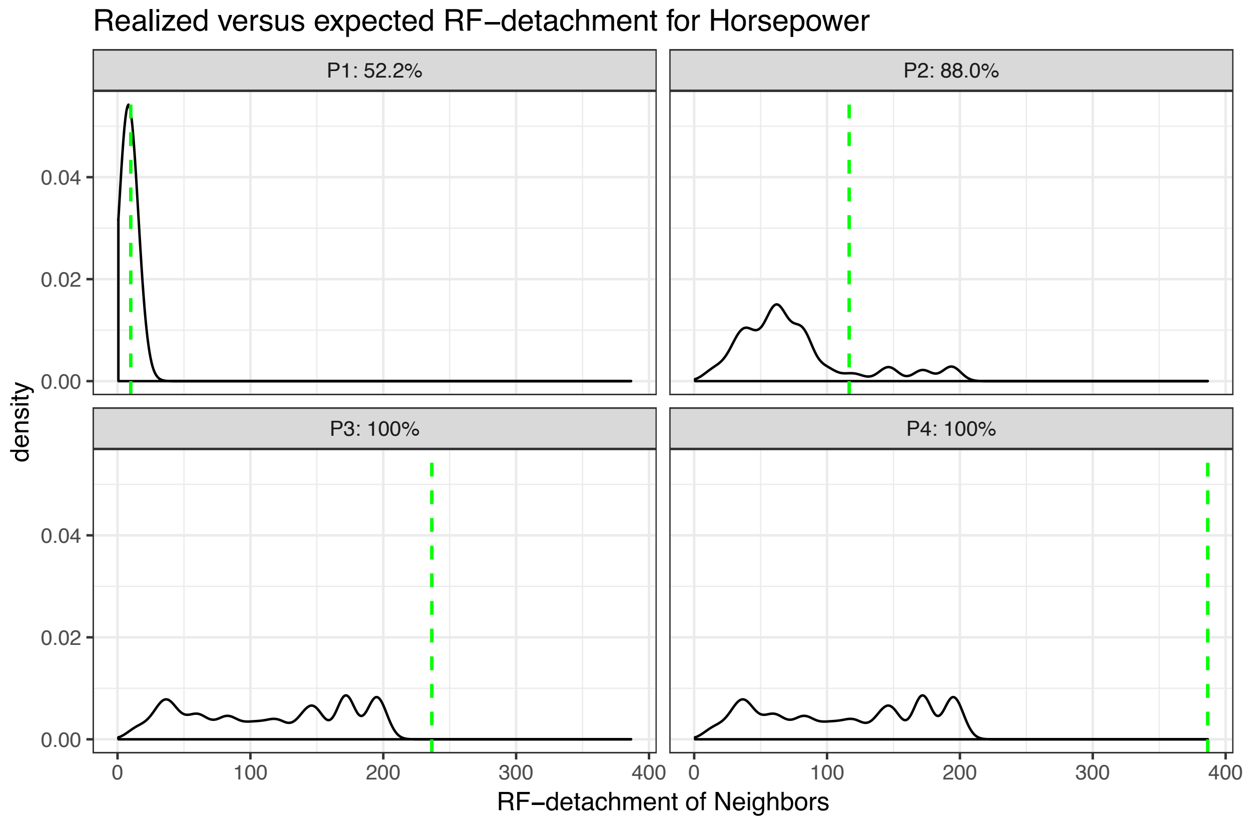


Figure 4.6: Detachment indices of cars P1, P2, P3, and P4 compared to their neighbors. The green line corresponds to the detachment index. The solid, black line shows the density of the detachment indices of its neighbors, weighted by the RF-weight function.

detachment indices. In comparison, P3 and P4 are very extreme, and all its neighbors have smaller detachment indices. Based on these insights, we conclude that P3 and P4 are atypical. We would then flag those as "detached" to emphasize the fact that inference with RF cannot not be trusted. We propose to flag a point as "detached" and suggest to not make a prediction on it if its detachment index is more extreme in a relevant direction than 99% of its neighbors. Algorithm 1 describes the exact algorithm that determines which predictions are trustworthy. We will show its effectiveness in Example B and C in the following section.

Software

We have carefully evaluated and implemented the stated algorithms in our software package `forestry`. Even though we stated everything in this section in terms of the RF-weight function, $w_i(x)$, our implementation avoids using the weight function and computes the detachment indices in a fast and memory efficient way. The idea is similar to the fact that Equation 4.3 is a perfectly valid way for getting the RF predictions, but the actual imple-

Algorithm 1 Flag as Trustworthy for covariate j

```
    Input: Training Set: $(X_i, Y_i)_{i=1}^n$ with $X_i \in \mathbb{R}^d$ and $Y_i \in \mathbb{R}$,
           Test point: $x \in \mathbb{R}^d$,
           Covariate to be checked: $j \in \{1, \dots, d\}$,
    Output: Flag (can be "trustworthy" or "detached")
 1: procedure IS_TRUSTWORTHY
 2:     Train a RF on $(X_i, Y_i)_{i=1}^n$ and denote the corresponding RF-weights as $w_i(x)$.
 3:     Compute the RF detachments, $d^j(x)$, as in Definition 4,

            ▷ Compute the detachment indices for all training units using Cross-Validation.
 4:     Create $K$ CV folds of the training data.
 5:     for $k \in \{1, \dots, K\}$ do
 6:         Train a RF on all units apart from those in Fold $k$
 7:         Using this RF, compute the RF detachments, $\tilde{d^j}(X_i)$, for all units $i$ in Fold $k$.

 8:     Compute the weighted percentile of units with a bigger detachment index than $x$,
            $$p(x) \leftarrow \sum_{i=1}^{n} w_i(x) 1 \left\{ \tilde{d^j}(X_i) > d^j(x) \right\}.$$
 9:     if $p(x) > 1\%$ then
10:         return ("trustworthy")
11:     else
12:         return ("detached")
```

Note that $w_i(x)$ is 0 for most i and positive for i corresponding to a small neighborhood around x. In our actual implementation, we set the number of units sampled for each tree in R to be relatively small (50% or less). The algorithm displayed here would be too slow for a real implementation. We are using an algorithm that is mathematically equivalent, but significantly more efficient in terms of computation and memory. The precise implementation can be found in the `forestry` package.

mentations use an algorithm that is conceptually much closer to Equation 4.1. The relevant functions are the `compute_detachments` and `evaluate_detachments` functions.

4.3 Applications

We will now apply Algorithm 1 to two further examples. The first one (Example B) deals with the overlap condition for the CATE estimation. The second one (Example C) deals with extrapolation and prediction outside the support of the training set. With these examples we want to show the usefulness of the detachment index as a tool that can be used in different statistical settings.

Example B: CATE Estimation Without Overlap

Let us recall the CATE estimation problem that we have studied in Chapter 1: We employ the Neyman–Rubin potential outcome framework [66, 71] and assume a superpopulation or distribution $\mathcal{P}$ from which a realization of N independent random variables is given as the training data. That is, $(Y_i(0), Y_i(1), X_i, W_i) \sim \mathcal{P}$, where $X_i \in \mathbb{R}^d$ is a d-dimensional covariate or feature vector, $W_i \in \{0, 1\}$ is the treatment assignment indicator, $Y_i(0) \in \mathbb{R}$ is the potential outcome of unit i when i is assigned to the control group, and $Y_i(1)$ is the potential outcome when i is assigned to the treatment group. The goal is to estimate the CATE which is defined as

$$\tau(x) := \mathbb{E}\Big[Y(1) - Y(0)\Big|X = x\Big].$$

To make this goal feasible, researchers usually assume that there are no hidden confounders [64].

Condition 1 (No hidden confounders)

$$(Y(1), Y(0)) \perp W|X.$$

Furthermore, they assume that the propensity score, $e(x) := \mathbb{P}(W|X = x)$, is bounded away from 0 and 1 [64].

Condition 2 (Overlap) *There exists $e_{\min}$ and $e_{\max}$ such that for all x in the support of X,*

$$0 < e_{\min} < e(x) < e_{\max} < 1.$$

Both of these conditions are satisfied in a randomized controlled trial, where the treatment is assigned to the training units by a coin flip. Therefore, these conditions become more important in an observational study in which the treatment assignment is not done by the researcher. Arguably, both conditions are very restrictive. In some sense they are even in competition with each other: More covariates make it more likely that Condition 1 is satisfied, but they make it less likely that Condition 2 is satisfied [17].

We study here the problem that can arise when Condition 2 is violated. We set up two simulations: Simulation 1 is a randomized controlled trial, and Simulation 2 is similar to Simulation 1 except that for a subset of the training data, the propensity score is artificially set to 0. This could, for example, happen in a clinical study where doctors do not want to try a new drug on a certain subsets of patients, but experimenters still allow these units to be part of the study as control units or in an actual observational study where a specific subset of units avoids the treatment.

Simulation 1

We take a random sample of 12,000 potential voters from the GOTV experiment that we have introduced in Chapter 1. Recall that the data consists of an indicator for the binary *treatment, an outcome variable, and seven key individual-level covariates: gender, age, and whether the registered individual voted in the primary elections in 2000, 2002, and 2004 or the general election in 2000 and 2002.*

We only take the seven covariates of each voter and do not make use of the treatment assignment or the observed outcomes. Instead, we simulate the treatment assignment and the observed outcomes according to the following equations:

$$\begin{aligned} Y_i(0) &= X_i \cdot \beta_0 + \varepsilon, \\ Y_i(1) &= X_i \cdot \beta_1 + \varepsilon, \\ W_i &\sim \mathit{Bern}(0.5), \end{aligned}$$

where $\beta_0, \beta_1 \in \mathbb{R}^7$ *was sampled independently before the experiment from a standard normal distribution and* $\varepsilon_i \overset{iid}{\sim} \mathcal{N}(0,1)$.

Simulation 2

We take the simulated data from Simulation 1 and assign all units between 30 and 40 to the control group.

Note that in Simulation 2 the overlap condition (Condition 2) is violated for units between 30 and 40. However, there are still no hidden confounders, and thus Condition 1 is satisfied.

To evaluate the performance of different CATE estimators, we split the data of each of the simulations into a training set of 2,000 units and a test set of 10,000. We then evaluated the X-Learner with RF (X_RF) and BART (X_BART) on both data sets. Figure 4.7 shows the average confidence coverage and length of the two CATE estimators for both simulations. We can see that the coverage for Simulation 2 is much worse than the coverage for Simulation 1 and that the average length is smaller. The reasons for this are that the CATE estimators are further away from the true values for observations in the set without overlap and they report for these observations smaller confidence intervals (c.f. Figure C.1).

This is, of course, an undesirable behavior. We would at least hope to report larger confidence intervals in areas without overlap. To overcome this problem, we want to use a decision rule based on the detachment index to label units in the area without overlap as "detached" or not trustworthy. Table 4.5 shows the number of units that have been labeled by Algorithm 1 as "detached" or "trustworthy." Note that in both Simulations most units in the set that have overlap are labeled as "trustworthy." In Simulation 2, most units that were not in the overlap set were labeled as "detached." This confirms that Algorithm 1 is able to detect units without overlap at least to some extent. Furthermore, Figure 4.5 shows that the confidence interval coverage on only those units that were labeled as "trustworty" is much larger for Simulation 2 and slightly larger for Simulation 1. This suggests that we were able to identify at least some points for which CI estimation appears to be difficult.

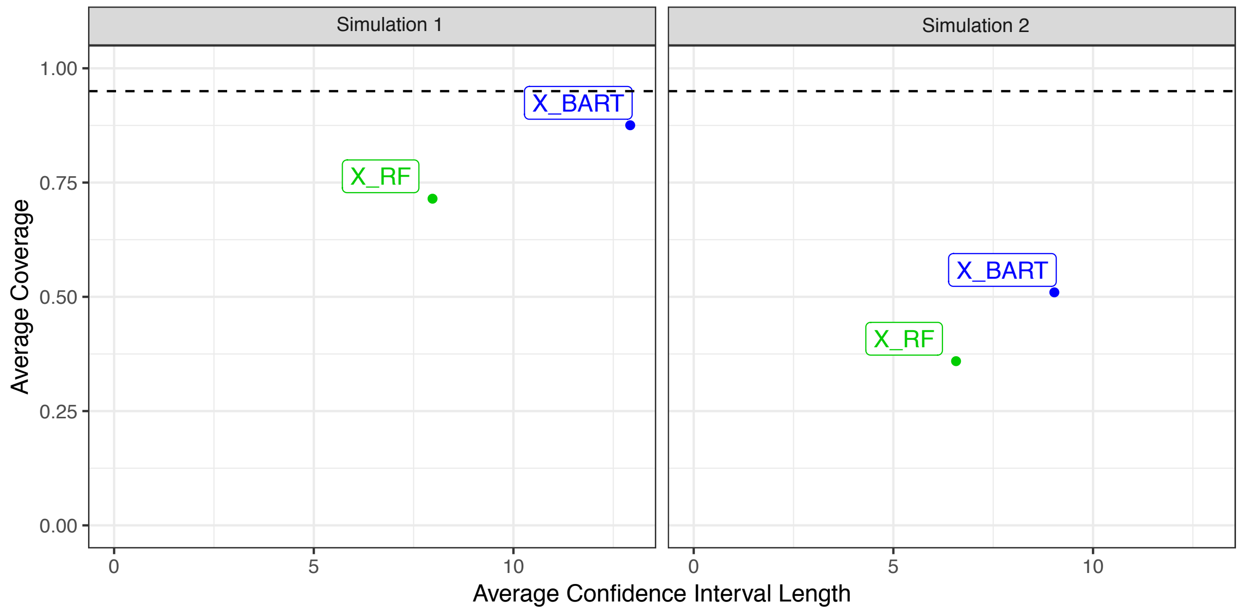


Figure 4.7: Confidence interval coverage and length of X_RF and X_BART for Simulation 1 and 2.

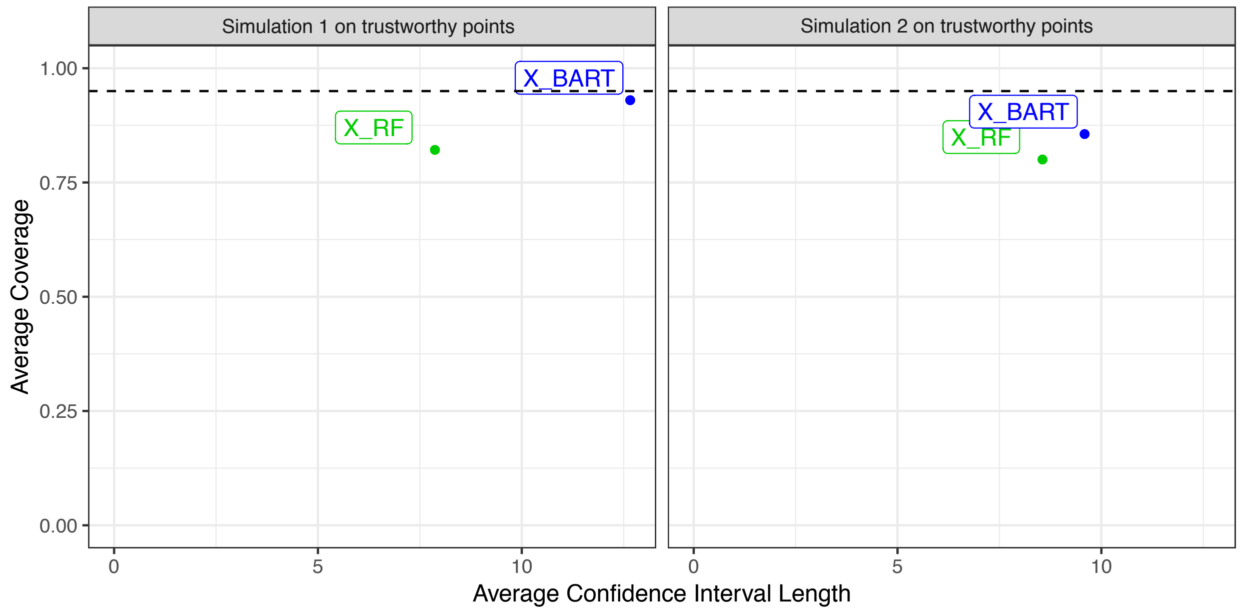


Figure 4.8: Confidence interval coverage and length of X_RF and X_BART for the trustworthy points in Simulations 1 and 2.

Simulation 1	trustworthy	detached
Overlap	9,806	149
No Overlap	0	0

Simulation 2	trustworthy	detached
Overlap	5,645	283
No Overlap	700	3,372

Table 4.5: Number of "trustworthy" units in the overlap area and the area without overlap.

Example C: Prediction Outside of the Support

We investigate the problem in a simple prediction task, where the test set is not in the support[5] of the training set. That is, of course, a situation that should be avoided. But nonetheless, it is a scenario methodologists have to face in a time where an increasing amount of decisions are being automated, and the engineers designing these decision making processes are often not the ones that make these decisions.

For example, many medical studies are based on subjects that are older than 18 years old. Medical professionals, of course, know that findings in such a study do not necessarily generalize to patients younger than 18 years old and would be extra cautious when applying these findings to a young patient. Algorithms, however, often do not automatically reveal that information. We think that it is important for algorithms to automatically flag such a prediction as not trustworthy.

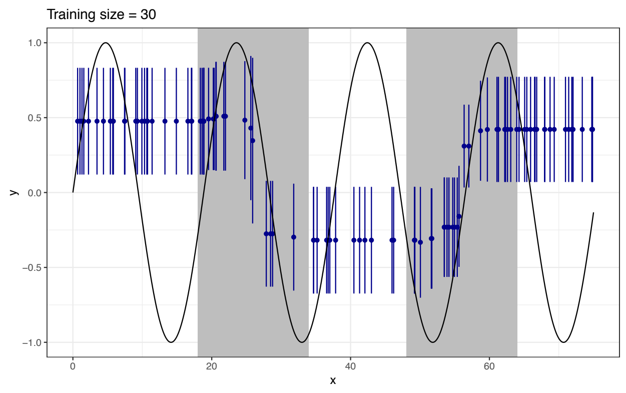
To investigate the performance of our proposed algorithm, consider the following simulation.

Simulation 3
For the training set, sample the independent variable x from a uniform distribution with support in $[18, 34] \cup [48, 64]$. For the test set, sample the independent variable x from a uniform distribution with support in $[0, 75]$. Create the dependent outcome y according to $y = \sin(x/3)$.

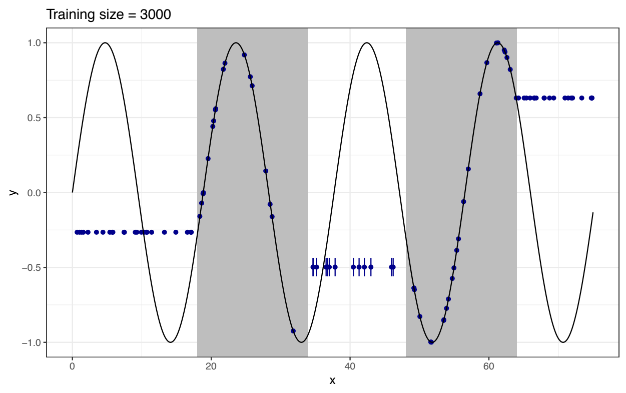
Note that the support of the training data, $[18, 34] \cup [48, 64]$, does not include the support of the test data, $[0, 75]$. Figure 4.9 shows the results of this simulation with bootstrap CIs for RF estimates. We can see that for points outside of the support, the confidence intervals are far away from the true response function. To make matters even worse, with more data, the CIs get tighter. This suggests more certainty about these estimates, even though they are just as far off as the estimates based on a sample size of 30.

Figure 4.10 shows the units that Algorithm 1 flagged as "detached" in red and the trustworthy ones in "blue." We can see that for this simple example, our method correctly identifies test units that are outside the support of the training set. Further research is needed to understand how well it works in practice especially in high-dimensional data sets.

[5]Here we define the support of the training set as the space of the covariate that has a positive density in the data generating distirbution of the training set.



(a) Trainings Size of 30



(b) Trainings Size of 3000

Figure 4.9: CI estimation with RF outside the support of the training data for a sample size of 30 and a training set of 3000 points. The grey area is the support of the training data. The black line is the regression function. The point estimates and 95% confidence intervals are displayed in blue.

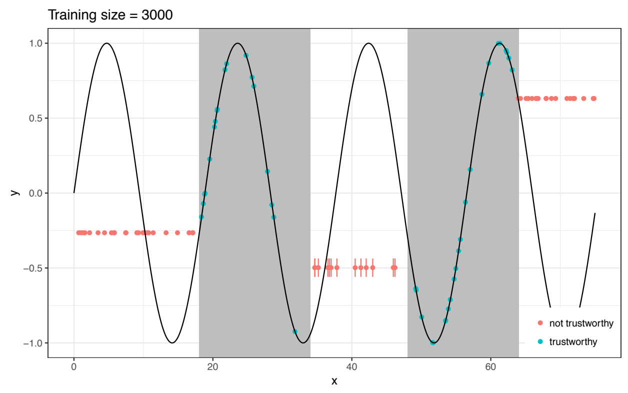


Figure 4.10: CI estimation with RF outside the support of the training data on trustworthy points.

4.4 Conclusion

In a time where algorithms are increasingly influencing our everyday lives, it becomes crucial to have algorithms that are aware of their flaws. We should design them in such a way that they warn the user if the underlying data situation does not allow a trustworthy output.

We have focused in this paper on the RF algorithm. We have demonstrated that the spreading of the weight function contains information about the trustworthiness of the fit. We have proposed a detachment index and have shown two strategies that can be used to determine that a detachment index is too big for prediction or CI estimation. We have also shown its usefulness in three examples: The first one is a usual prediction task, the second one is a CATE estimation problem with a violated overlap condition, and the last one is a prediction task where the test data is not in the support of the training data. All of these examples are dealing with low-dimensional data sets and we are currently working on extending this framework to high-dimensional problems.

We believe that the general problem of trusting the predictions of algorithms is important and it should also be studied for other methods. The particular solution we have found for RF, the detachment index, is potentially useful beyond the settings described in this work. It is not clear yet whether the concept would have to be adapted to deal with more challenging scenarios, such as high-dimensional or heterogeneous data. We believe that more work should be done to allow any estimator to flag predictions as "untrustworthy."

Chapter 5

Linear Aggregation in Tree-based Estimators

5.1 Introduction

Classification and Regression Trees (CART) [56, 8] have long been used in many domains. Their simple structure makes them interpretable and useful for statistical inference, data mining, and visualizations. The Random Forests (RF) [6] algorithm and the Gradient Boosting Machine (GBM) [25] build on these tree algorithms. They are less interpretable and more laborious to visualize than single trees, but they often perform better in predictive tasks and lead to smoother estimates [12, 74, 78].

As these tree-based algorithms predict piece-wise constant responses, this leads to (a) weaker performance when the underlying data generating process exhibits smoothness, (b) relatively deep trees that are harder to interpret, and (c) bias that makes valid inference and confidence interval estimation difficult.

To address these weaknesses, we study regression trees with a linear aggregation function implemented in the leaves. Specifically, we introduced three core changes to the classical CART algorithm:

1. Instead of returning for each leaf the mean of the corresponding training observations as in the classical CART algorithm, we fit a ridge regression in each leaf. That is, for a unit that falls in a leaf S, the tree prediction is

$$\hat{\mu}(x_{\text{new}}) = x_{\text{new}}^t (X_S^t X_S + \lambda I)^{-1} X_S^t Y_S, \tag{5.1}$$

where Y_S is the vector of y-values of the training observations that fall in leaf S, X_S is the corresponding design matrix for these observations, and $\lambda \in \mathbb{R}^+$ is a regularization parameter.[1]

[1] In a more elaborate version of the algorithm, the observations in leaf S are a subset that is independent of the subset of training observations used to create the skeleton of the trees.

2. Crucial to the success of such an aggregation function, we take the fact that we fit a regularized linear model in the leaves into account when constructing the tree by following a greedy strategy that finds the *optimal*[2] splitting point at each node. That is, for each candidate split that leads to two child nodes S_L and S_R, we take the split such that the total MSE after fitting an ℓ_2-penalized linear model on each child is minimized. This is very difficult, and we believe that one of our main contributions is that we found a very efficient way to find this optimal splitting point. We discuss in detail how the optimal splitting point is calculated in Section 5.2.

3. Furthermore, we use a cross-validation stopping criteria that determines when to continue splitting as opposed to when to construct a leaf node. After selecting the optimal split, the improvement in R^2 that is introduced by the potential split is calculated. If the potential split increases the R^2 by less than a predetermined stopping value, then the splitting procedure is halted and a leaf node is created. This leads to trees that can create large nodes with smooth aggregation functions on smooth parts of data, while taking much smaller nodes which mimic the performance of standard CART aggregation on separate subsets of the response surface. This adaptivity over varying degrees of smoothness is displayed below in Figure 5.1 and explored in greater detail in Section 5.3.

These changes improve the **predictive performance** of such Linear Regression Trees (LRT) substantially, and—as we see in Section 5.3—it compares favorably on real-world examples when used in a Linear Random Forests (LRF) ensemble. We also connect it with an effective tuning algorithm, and we find that it can behave like both a regularized linear model and a CART/RF estimator depending on the chosen hyperparameters. This adaptivity as an estimator is explored in a set of experiments in Section 5.3.

The linear response functions in turn lead to much shallower trees without losing predictive performance. This improves the **interpretability** of the trees substantially—a direction which has always been a major motivation for studying regression trees with linear aggregation functions [39].

In Section 5.4, we illustrate the advantages for interpretability using a large dataset measuring the treatment effect of several treatments on voter turnout [27]. We adapt the S-Learner with random forests as implemented in causalToolbox [45] to use the linear aggregation function of linear regression trees. We then use the plot function from the `forestry` package to visualize several trees in Figure 5.3 and the regression coefficients of their leaves. This allows us to identify groups of potential voters with similar voting behavior, make inference on their future voting propensity, and see the effects different mailers have on future voter turnout.

[2]We define what we mean by *optimal* in Section 5.2.

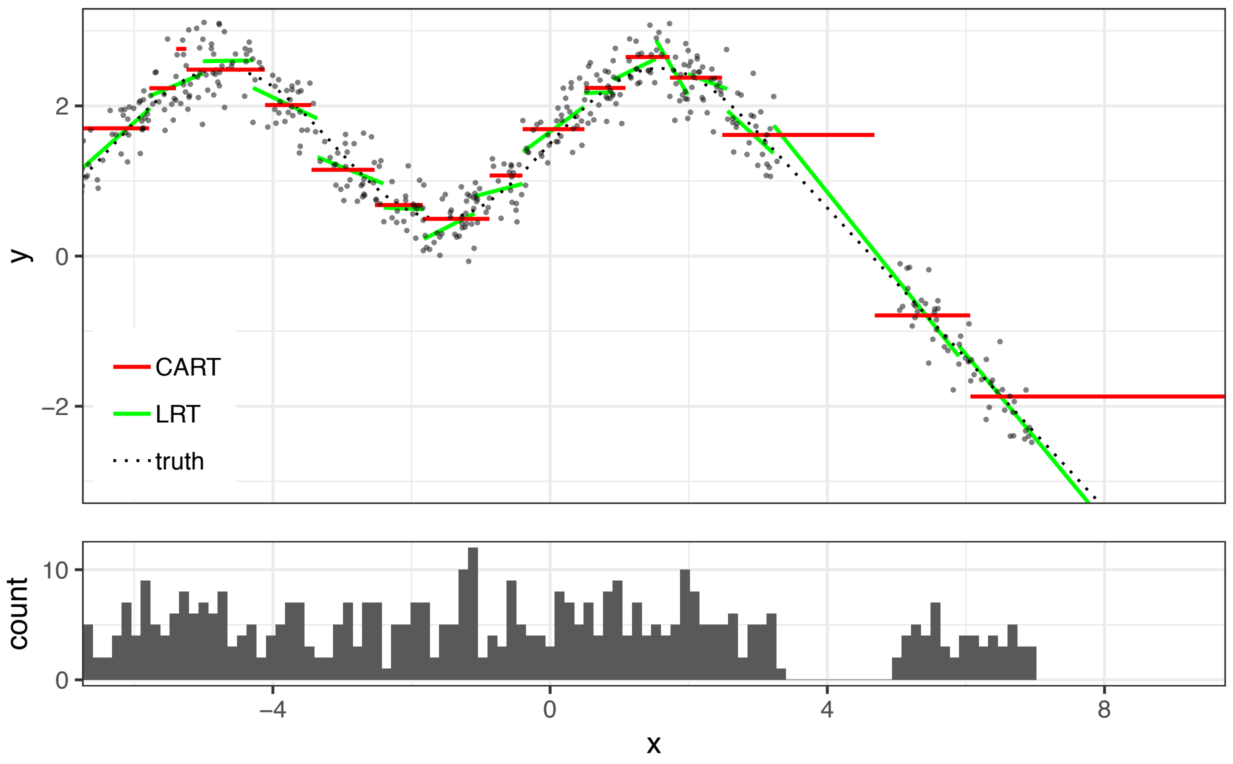


Figure 5.1: Comparison of the classical CART and LRT. The top of the figure shows the Linear Regression Trees and the classical CART, and the lower part shows the density of the training set.

Literature

Studying regression trees with linear aggregation functions has a long history. The earliest reference we are aware of is by [7] which precedes the original CART publication [8]. The algorithm can be interpreted as a tree algorithm: At each node, the algorithm attempts to introduce a new split along a randomly oriented plane. It then fits a linear model in both children, and it accepts the split if it sufficiently reduces the residual sum of squares as captured by an F-ratio test. If the split is not accepted, the algorithm attempts more splits a limited number of times before declaring the node a leaf node.

There has been much research that builds on this first algorithm, and many different variants of it have been proposed. Apart from a few exceptions [86], the trees are constructed in a recursive strategy and the main differences between the algorithms are, how the splits are generated—the splitting criteria—and when splitting is halted for a node to be defined as a leaf—the stopping criteria.

For the splitting criteria, there have been many different suggestions. [77] spans the trees similar to the standard CART algorithm without adaptations to the new aggregation function. [13] and [50], on the other hand, fit a linear model, or more generally, a polynomial model, on all units in the current node and then suggest a split by looking at the residuals of the fitted model. [85] and [67], in turn, use a two-step procedure to find a good split. In the first step, the algorithms also fit a linear model on all units in the current node, and they then screen for good splitting covariates using a test statistic that can be computed relatively fast. In a second step, the algorithm then exhaustively searches through all possible splits along the selected features.

While it is possible to grow trees until there is only one observation left in each leaf, it is useful to have tree structures with more than one observation per leaf. This is particularly important when using a linear aggregation function. However, the exact number of observations which should be in each leaf is much harder to determine. Early stopping and pruning of the trees after they have been trained have both been used for regression trees with linear aggregation functions. For example, [61] and [26] build linear models in the pruning phase of each leaf, while [13] use a stopping criterion that is based on cross-validation and attempts to estimate whether a further split sufficiently improves the mean squared error.

While it is always possible to bootstrap these algorithms and combine them with ideas introduced in Random Forests [6], these ideas were mainly studied as regression trees, even though one would expect better predictive power and smoother prediction surfaces in the bagged versions [12]. Additionally, some of these algorithms would be computationally too expensive to be used in a bagged version.

However, there has been some work done combining RF and bagged trees with linear models. [5] followed ideas by [37], [51], and [54] to interpret random forests as an adaptive potential nearest neighbor method. Their method, SILO (Supervised Local modeling), uses the random forests algorithm to provide a distance measure, w, based on the proximity distance of random forests, and it then defines the random forests prediction, $\hat{g}(x)$, as a local linear model [72] via the following two step process:

$$\begin{aligned} \hat{f}_x(\cdot) &= \operatorname*{arg\,min}_{f \in \mathcal{F}} \sum_{i=1}^{N} w(x_i, x)(y_i - f(x_i - x))^2, \\ \hat{g}(x) &= \hat{f}_x(0), \end{aligned} \tag{5.2}$$

where $\mathcal{F}$ is some set of functions. In their paper, they focused in particular on linear models and demonstrated its superior performance over other local models such as LOESS [16] and untuned RF.

In a very recent paper, [23] also combined random forests with linear models. Their work is similar to that of [5] in the sense that they also use the proximity weights of random forests to fit a local linear model. Specifically, they focus on linear models, and they fit a ridge regularized linear model,

$$\begin{pmatrix} \hat{\mu}(x) \\ \hat{\theta}(x) \end{pmatrix} = \operatorname*{arg\,min}_{\mu, \theta} \left\{ \sum_{i=1}^{n} w(x_i, x)(y_i - \mu - (x_i - x)^t \theta)^2 + \lambda ||\theta||_2^2 \right\}, \tag{5.3}$$

and use $\hat{\mu}(x)$ as the estimate for $\mathbb{E}[Y|X = x]$. Similar to [13], they adapt the splitting function to account for the fitting of such local models. When evaluating a split on a parent node

P, the split is adapted by fitting a ridge regression in each parent node P to predict $\hat{Y}_{P,Ridge}$ from X_P and then the standard CART splitting procedure on the residuals $Y_P - \hat{Y}_{P,Ridge}$ is used to select a splitting point. This results in a fast splitting algorithm which utilizes the residuals to model for local effects in forest creation and optimize for the fitted models which are later used for regression.

Our contribution

Our main contribution is to develop a fast algorithm to find the best partition for a tree with a ridge regularized linear model aggregation function. To our knowledge, we are the first to develop and analyze this algorithm, even though there are software packages that have not publicized their source code that might use a similar idea. In Section 5.2, we explain the algorithm and we show that its run time is $\mathcal{O}(m*(n*\log(n)+n*p^2))$ where n is the number of observations in the current node, p is the number of dimensions that is fit with a linear model, and m is the number of potential splitting covariates. In Section 5.3, we use the splitting algorithm with random forests, and we show that it compares favorably on many data sets. Depending on the chosen hyperparameters, the LRF algorithm can then behave and perform similarly to either the default RF algorithm or a ridge regularized linear model. Because the algorithms can be trained relatively fast, we were able to connect them with a tuning algorithm, and show how it can quickly adapt to the underlying smoothness levels in different datasets. In Section 5.4, we then apply a simple LRT to a real data set to show how its simple structure lead to a deeper understanding of the underlying processes.

Software example

An implementation of this algorithm is used in the `forestry` package and can, for example, be used in random forests using the `linear = TRUE` option.

```r
forest <- forestry(x = iris[,-1],
                   y = iris[,1],
                   linear = TRUE,
                   overfitPenalty = 2)
predict(forest, feature.new = iris[,-1])
```

5.2 The Splitting Algorithm

Random forests are based on a regression tree algorithm, which is in turn based on a splitting algorithm. This splitting algorithm is used in most tree-based algorithms to recursively split the space into subspaces which are then encoded in a binary regression tree. In this section, we first motivate and describe the new splitting algorithm and then discuss its asymptotic runtime.

To setup the notation, assume that we observe n observations, $(X_i, Y_i)_{i=1}^n$. $X_i \in \mathbb{R}^d$ is the d-dimensional feature vector and $Y_i \in \mathbb{R}$ is the dependent variable of unit i. For a feature $\ell \in \{0, \dots, d\}$, the goal is to find a splitting point s to separate the space into two parts,

$$L := \{x \in \mathbb{R}^d : x[\ell] < s\}, \qquad R := \{x \in \mathbb{R}^d : x[\ell] \geq s\}. \tag{5.4}$$

We call L and R the left and right sides of the partition respectively.

In many tree-based algorithms, including CART, RF, and GBM, the splitting point s^* is a point that minimizes the combined RSS when fitting a constant in both parts of the space,

$$s^* \in \arg\min_s \sum_{i:X_i \in L} \left(Y_i - \bar{Y}^L\right)^2 + \sum_{i:X_i \in R} \left(Y_i - \bar{Y}^R\right)^2. \tag{5.5}$$

Here $\bar{Y}^L$ is the mean of all units in the left partition and $\bar{Y}^R$ is the mean of all units in the right partition. Note that L and R are functions of s.

We generalize this idea to Ridge regression. We want to find s^* that minimizes the overall RSS when- in both partitions- a Ridge-regularized linear model is fitted instead of a constant value.

$$s^* \in \arg\min_s \sum_{i:X_i \in L} \left(Y_i - \hat{Y}_i^L\right)^2 + \sum_{i:X_i \in R} \left(Y_i - \hat{Y}_i^R\right)^2. \tag{5.6}$$

Now $\hat{Y}_i^L$ is the fitted value of unit i when a Ridge regression is fit on the left part of the split, and similarly, Y_i^R is the fitted value of unit i when a Ridge regression is fit on the right part of the split.

To find an s^* that satisfies (5.6), it would be enough to exhaustively search through the set $S = \{X_i[\ell] : 1 \leq i \leq n\}$. However, if feature ℓ is continuous, then there are n potential splitting points. Fitting a ridge regression in both parts and computing the RSS for each potential splitting point thus requires at least n^2 steps. This is too slow for most data sets, since the splitting algorithm is applied up to dn times when constructing a single regression tree.

Fast and exact ridge splitting algorithm

In the following section, we describe an algorithm that computes the exact solution of (5.6) in quasilinear time by using an online update for the RSS after fitting a ridge regularized linear model. Specifically, the algorithm first sorts the realized values of feature ℓ in ascending order. To simplify the notation, let us redefine the ordering of the units in such a way that

$$X_1[\ell] < X_2[\ell] < \cdots < X_n[\ell].$$

Such a sorting can be done in $\mathcal{O}(n \log n)$.[3]

[3]We assume here that $X_i[\ell] \neq X_j[\ell]$ for all $j \neq j$. We refer to our implementation in `forestry` [44] for the general case.

The algorithm then checks the $n-1$ potential splitting points that lie exactly between two observations,

$$s_k := \frac{X_k[\ell] + X_{k+1}[\ell]}{2}, \text{ for } 1 \le k \le n-1.$$

We define

$$L(k) := \{x \in \mathbb{R}^d : x[\ell] < s_k\}, \qquad R(k) := \{x \in \mathbb{R}^d : x[\ell] \ge s_k\}, \tag{5.7}$$

and the RSS when splitting at s_k as $\text{RSS}(k)$,

$$\text{RSS}(k) := \sum_{i:X_i \in L(k)} \left(Y_i - \hat{Y}_i^{L(k)}\right)^2 + \sum_{i:X_i \in R(k)} \left(Y_i - \hat{Y}_i^{R(k)}\right)^2. \tag{5.8}$$

Once again, $\hat{Y}_i^{L(k)}$ is the fitted value of unit i when a Ridge regression is fit on the left part of the split, and similarly, $Y_i^{R(k)}$ is the fitted value of unit i when a Ridge regression is fit on the right part of the split. We are interested in finding

$$k^* = \arg\min_k \text{RSS}(k). \tag{5.9}$$

Decomposition of the RSS

Let us first state the ridge-regression for the left partition [36]. For a given $\lambda > 0$ and leaf, L, we want to find $(\hat{\beta}, \hat{c})$ that minimize the ridge penalized least squares equation,

$$(\hat{\beta}, \hat{c}) = \arg\min_{(\beta,c)} \sum_{i:X_i \in L} \left(Y_i - X_i^T\beta - c\right)^2 + \lambda\|\beta\|_2^2. \tag{5.10}$$

A closed form solution of this problem can be derived by setting the gradient of this loss function equal to zero,

$$\begin{bmatrix}\hat{\beta}\\ \hat{c}\end{bmatrix} = \left(\sum_{i=1}^n \begin{bmatrix}X_i\\ 1\end{bmatrix}\begin{bmatrix}X_i^T & 1\end{bmatrix} + \lambda\begin{bmatrix}\mathcal{I}_d & 0\\ 0 & 0\end{bmatrix}\right)^{-1} \sum_{i=1}^n Y_i \begin{bmatrix}X_i\\ 1\end{bmatrix}. \tag{5.11}$$

For a set $H \subset \mathbb{R}^d$, define

$$\begin{aligned}
A_H &:= \sum_{i:X_i \in H} \begin{bmatrix}X_i\\ 1\end{bmatrix}\begin{bmatrix}X_i^T & 1\end{bmatrix} + \lambda\begin{bmatrix}\mathcal{I}_d & 0\\ 0 & 0\end{bmatrix}\\
S_H &:= \sum_{i:X_i \in H} Y_i \begin{bmatrix}X_i\\ 1\end{bmatrix},\\
G_H &:= \sum_{i:X_i \in H} \begin{bmatrix}X_i\\ 1\end{bmatrix}\begin{bmatrix}X_i^T & 1\end{bmatrix}.
\end{aligned}$$

With this notation, we can decompose RSS(k),

$$\begin{aligned}\mathrm{RSS}(k) &= \sum_{i:X_i\in L(k)} \left(Y_i - \begin{bmatrix}X_i^T & 1\end{bmatrix} A_{L(k)}^{-1} S_{L(k)}\right)^2 \\ &\quad + \sum_{i:X_i\in R(k)} \left(Y_i - \begin{bmatrix}X_i^T & 1\end{bmatrix} A_{R(k)}^{-1} S_{R(k)}\right)^2 \\ &= \Phi_{L(k)} + \Phi_{R(k)} + \sum_{i=1}^{n} Y_i^2.\end{aligned}$$

Here, we used the definition that for a set $H \subset \mathbb{R}^d$,

$$\Phi_H := S_H^T A_H^{-1} G_H A_H^{-1} S_H - 2 S_H^T A_H^{-1} S_H. \tag{5.12}$$

As $\sum_{i=1}^n Y_i^2$ is constant over k, it can be discarded when considering the optimization problem and thus,

$$\arg\min_k \ \mathrm{RSS}(k) = \arg\min_k \ \Phi_L(k) + \Phi_R(k). \tag{5.13}$$

Update Step from k to $k+1$

In order to have a manageable overall runtime for a split, we need to quickly find the minimizer of (5.9) by looping through from $k = 1$ to $k = n$.

1. The algorithm first computes $S_{L(1)}$, $A_{L(1)}^{-1}$, $G_{L(1)}$, $S_{R(1)}$, $A_{R(1)}^{-1}$, and $G_{R(1)}$, and computes the RSS(1).
2. The algorithm now computes the RSS(k) for $k \geq 2$ in an iterative way:
   a) $S_{L(k)}$, $G_{L(k)}$, $S_{R(k)}$, and $G_{R(k)}$ can be directly computed from $S_{L(k-1)}$, $G_{L(k-1)}$, $S_{R(k-1)}$, and $G_{R(k-1)}$ by a simple addition or subtraction.
   b) For $A_{L(k)}^{-1}$ and $A_{R(k)}^{-1}$, we use the Sherman-Morrison Formula:

$$A_{L(k)} = A_{L(k-1)}^{-1} - \frac{A_{L(k-1)}^{-1}\begin{bmatrix}x_k \\ 1\end{bmatrix}\begin{bmatrix}x_k^T & 1\end{bmatrix} A_{L(k-1)}^{-1}}{1 + \begin{bmatrix}x_k^T & 1\end{bmatrix} A_{L(k-1)}^{-1}\begin{bmatrix}x_k \\ 1\end{bmatrix}}, \tag{5.14}$$

$$A_{R(k)} = A_{R(k-1)}^{-1} + \frac{A_{R(k-1)}^{-1}\begin{bmatrix}x_k \\ 1\end{bmatrix}\begin{bmatrix}x_k^T & 1\end{bmatrix} A_{R(k-1)}^{-1}}{1 - \begin{bmatrix}x_k^T & 1\end{bmatrix} A_{R(k-1)}^{-1}\begin{bmatrix}x_k \\ 1\end{bmatrix}}. \tag{5.15}$$

An explicit implementation of the split algorithm can be found in Algorithm 2.

Algorithm 2 Find Split to Minimize Sum of RSS

```
    Input: Features: $X \in \mathbb{R}^{n \times d}$,
           Dependent Outcome: $Y \in \mathbb{R}^n$,
           overfitPenalty (regularization for split): $\lambda \in \mathbb{R}^+$,
    Output: Best Split point k
 1: procedure FindBestSplitRidge

    Initialization:
 2:     $A_{L1}^{-1} \leftarrow \left( \begin{bmatrix} X_1 \\ 1 \end{bmatrix} \begin{bmatrix} X_1^T & 1 \end{bmatrix} + \lambda \begin{bmatrix} \mathcal{I}_d & 0 \\ 0 & 0 \end{bmatrix} \right)^{-1}$
 3:     $A_{R1}^{-1} \leftarrow \left( \sum_{i=2}^n \begin{bmatrix} X_i \\ 1 \end{bmatrix} \begin{bmatrix} X_i^T & 1 \end{bmatrix} + \lambda \begin{bmatrix} \mathcal{I}_d & 0 \\ 0 & 0 \end{bmatrix} \right)^{-1}$
 4:     $S_{L1} \leftarrow Y_1 \begin{bmatrix} X_1 \\ 1 \end{bmatrix}$
 5:     $S_{R1} \leftarrow \sum_{i=2}^n Y_i \begin{bmatrix} X_i \\ 1 \end{bmatrix}$
 6:     $G_{L1} \leftarrow \begin{bmatrix} X_1 \\ 1 \end{bmatrix} \begin{bmatrix} X_1^T & 1 \end{bmatrix}$
 7:     $G_{R1} \leftarrow \sum_{i=2}^n \begin{bmatrix} X_i \\ 1 \end{bmatrix} \begin{bmatrix} X_i^T & 1 \end{bmatrix}$
 8:     Compute the RSS sum:
            $\begin{aligned} \text{RSS}_1 \leftarrow & S_{L1}^T A_{L1}^{-1} G_{L1} A_{L1}^{-1} S_{L1} - 2 S_{L1}^T A_{L1}^{-1} S_{L1} \\ & + S_{R1}^T A_{R1}^{-1} G_{R1} A_{R1}^{-1} S_{R1} - 2 S_{R1}^T A_{R1}^{-1} S_{R1}. \end{aligned}$

 9:     for $k = 2, \ldots, n$ do
10:         $A_{L(k)}^{-1} \leftarrow \text{Update\_A\_inv}(A_{L(k-1)}^{-1}, X_k, \text{leftNode} = TRUE)$
11:         $A_{R(k)}^{-1} \leftarrow \text{Update\_A\_inv}(A_{R(k-1)}^{-1}, X_k, \text{leftNode} = FALSE)$
12:         $S_{L(k)} \leftarrow S_{L(k-1)} + Y_k \begin{bmatrix} X_k \\ 1 \end{bmatrix}$
13:         $S_{R(k)} \leftarrow S_{R(k-1)} - Y_k \begin{bmatrix} X_k \\ 1 \end{bmatrix}$
14:         $G_{L(k)} \leftarrow G_{L(k-1)} + \begin{bmatrix} X_k \\ 1 \end{bmatrix} \begin{bmatrix} X_k^T & 1 \end{bmatrix}$
15:         $G_{R(k)} \leftarrow G_{R(k-1)} - \begin{bmatrix} X_k \\ 1 \end{bmatrix} \begin{bmatrix} X_k^T & 1 \end{bmatrix}$
16:         Compute the RSS sum for the current split:
                $\begin{aligned} \text{RSS}_k \leftarrow & S_{L(k)}^T A_{L(k)}^{-1} G_{L(k)} A_{L(k)}^{-1} S_{L(k)} - 2 S_{L(k)}^T A_{L(k)}^{-1} S_{L(k)}) \\ & + S_{R(k)}^T A_{R(k)}^{-1} G_{R(k)}) A_{R(k)}^{-1} S_{R(k)} - 2 S_{R(k)}^T A_{R(k)}^{-1} S_{R(k)}. \end{aligned}$

17:     return $(\arg\min_k \text{RSS}_k)$
```

Update_A_inv is defined in Algorithm 3

Algorithm 3 Update the A^{-1} Component of the RSS

```
    Input: $A_{k-1}^{-1} \in \mathbb{R}^{(d+1)\times(d+1)}$,
           $X_k \in \mathbb{R}^d$,
           leftNode (indicator whether this updates $A_{L(k)}^{-1}$ or $A_{R(k)}^{-1}$)
    Output: Updated matrix $A_k^{-1}$
1: procedure UPDATE_A_INV
2:     $z_k \leftarrow A_{k-1}^{-1} \begin{bmatrix} X_k \\ 1 \end{bmatrix}$
3:     if leftNode then
4:         $g_k \leftarrow \frac{-z_k z_k^T}{1 + \begin{bmatrix} X_k^T & 1 \end{bmatrix} z_k}$
5:     else
6:         $g_k \leftarrow \frac{z_k z_k^T}{1 - \begin{bmatrix} X_k^T & 1 \end{bmatrix} z_k}$
7:     return $A_{k-1}^{-1} + g_k$
```

Runtime Analysis of Finding Split Points

The ability to use an online update for calculating the iterated RSS at each step is crucial for maintaining a runtime that can scale in quasilinear runtime. Here we will provide a detailed breakdown of the runtime for calculating the best split point on a given feature. As we have several steps for updating the RSS components, the runtime depends on both the number of observations, as well as the number of features and therefore may be affected by either. We begin each split by sorting the current split feature taking $\mathcal{O}(n \log n)$ time. Within the split iterations, we iterate over the entire range of split values once, however, at each step we must update the RSS components.

While updating the A^{-1} component, as we use the Sherman-Morison Formula to update the inverse of the sum with an outer product, we must compute one matrix vector product ($\mathcal{O}(d^2)$), one vector outer product ($\mathcal{O}(d^2)$), one vector inner product ($\mathcal{O}(d^2)$), division of a matrix by scalars and addition of matrices (both $\mathcal{O}(d^2)$). While updating the G component, we need to both add and subtract an outer product (both $\mathcal{O}(d^2)$), and while updating the S component, we need to add and subtract a vector ($\mathcal{O}(d)$). At each step of the iteration, we must evaluate the RSS of the right and left models. To do this, we need 8 matrix vector products, each of which is $\mathcal{O}(d^2)$, and 4 vector inner products, each of which is $\mathcal{O}(d)$. Putting these parts together gives us a total run time of $\mathcal{O}(n \log n + nd^2)$ to find the best split point for a given node with n observations and a d-dimensional feature space.

Early Stopping

As we will see in Section 5.3, early stopping can prevent overfitting in the regression tree algorithm and the RF algorithm. Furthermore, as we discuss in Section 5.4, it leads to well performing yet shallow trees that are much easier to understand and interpret.

We use a one step look-ahead stopping criteria to stop the trees from growing to deep. Specifically, we first compute a potential split as outlined in the algorithm above. We then use cross validation to compute the R^2 increase of this split and only accept it, if the increase of the split is larger than the specified `minSplitGain` parameter. A larger `minSplitGain` thus leads to smaller trees. The precise description of this parameter can be found in Algorithm 4.

5.3 Predictive Performance

It is well known that RF outperforms regression trees, and we have therefore implemented a version of RF that utilizes our new splitting algorithm. We call this Linear Random Forests (*LRF*) and we demonstrate its predictive performance in the following section. A version of LRF is implemented in our `forestry` package.[4]

Methods and Tuning

In this section, we compare the predictive power of LRF with five competitors:

- The random forest algorithm as implemented in the ***ranger*** package [82] and in the `forestry` package.
- The Bayesian Additive Regression Trees (BART) algorithm as implemented in the ***BART*** package [15].
- Local linear forests [23] as implemented in the ***grf*** package [3].
- The Rule and Instance based Regression Model presented in [61] as implemented in the ***Cubist*** package.
- Generalized Linear Models as implemented in the ***glmnet*** package [24].

In most real world prediction tasks, appliers carefully tune their methods to get the best possible performance. We believe that this is also important when comparing different methods. That is why we used the ***caret*** package [40] to tune the hyperparameters of all of the above methods. Specifically, we used the ***caret*** package's adaptive random tuning over 100 hyperparameter settings, using 8 fold CV tuning repeated and averaged over 4 iterations.

[4]A version of `forestry` is available at `https://github.com/soerenkuenzel/forestry`.

Algorithm 4 Early Stopping: The One Step Look-Ahead Algorithm

```
    Input: Features: $X \in \mathbb{R}^{n \times d}$,
           Dependent Outcome: $Y \in \mathbb{R}^n$,
           Indexes of potential left Child: $L \subset \{1, \dots, n\}$,
           minSplitGain: $m \in \mathbb{R}$,
           number of folds: $k \in \{2, \dots, n\}$,
           overfit penalty: $\lambda > 0$
    Output: Boolean: TRUE if split should be implemented, FALSE otherwise
 1: procedure CHECK_SPLIT
 2:     Partition $\{1, \dots, n\}$ into $k$ disjoint subsets: $\{S_1, \dots, S_k\}$.
 3:     for $i$ in $\{1, \dots, k\}$ do
                                        ▷ Predict the outcome without the split:
 4:         For $j \in S_i$ set
            $$\hat{Y}_j^p = \begin{bmatrix} X_j \\ 1 \end{bmatrix} \left( \sum_{k \in \bar{S}_i} \begin{bmatrix} X_k \\ 1 \end{bmatrix} \begin{bmatrix} X_k^T & 1 \end{bmatrix} + \lambda \begin{bmatrix} \mathcal{I}_d & 0 \\ 0 & 0 \end{bmatrix} \right)^{-1} \sum_{k \in \bar{S}_i} Y_k \begin{bmatrix} X_k \\ 1 \end{bmatrix}.$$
                                        ▷ Predict the outcome with the split:
 5:         For $j \in S_i \cap L$ set
            $$\hat{Y}_j^c = \begin{bmatrix} X_j \\ 1 \end{bmatrix} \left( \sum_{k \in \bar{S}_i \cap L} \begin{bmatrix} X_k \\ 1 \end{bmatrix} \begin{bmatrix} X_k^T & 1 \end{bmatrix} + \lambda \begin{bmatrix} \mathcal{I}_d & 0 \\ 0 & 0 \end{bmatrix} \right)^{-1} \sum_{k \in \bar{S}_i \cap L} Y_k \begin{bmatrix} X_k \\ 1 \end{bmatrix}.$$
 6:         For $j \in S_i \cap \bar{L}$ set
            $$\hat{Y}_j^c = \begin{bmatrix} X_j \\ 1 \end{bmatrix} \left( \sum_{k \in \bar{S}_i \cap \bar{L}} \begin{bmatrix} X_k \\ 1 \end{bmatrix} \begin{bmatrix} X_k^T & 1 \end{bmatrix} + \lambda \begin{bmatrix} \mathcal{I}_d & 0 \\ 0 & 0 \end{bmatrix} \right)^{-1} \sum_{k \in \bar{S}_i \cap \bar{L}} Y_k \begin{bmatrix} X_k \\ 1 \end{bmatrix}.$$
 7:     Compute the EMSE with and without split:
            $$\mathrm{RSS}^c = \sum_{i=1}^n \left( Y_i - \hat{Y}_i^c \right)^2 \qquad \text{and} \qquad \mathrm{RSS}^p = \sum_{i=1}^n \left( Y_i - \hat{Y}_i^p \right)^2$$
 8:     Compute the total variation: $\mathrm{tV} = \sum_{i=1}^n \left( Y_i - \bar{Y} \right)^2$
 9:     if $(\mathrm{RSS}^c - \mathrm{RSS}^p)/\mathrm{tV} > m$ then
            return TRUE
10:     else
            return FALSE
```

For a set S we define its compliment as $\bar{S} := \{i : 1 \le i \le n \text{ and } i \notin S\}$.

Experiments

LRF is particularly well suited for picking up smooth signals. It can also behave like a linear model or like classical RFs. To demonstrate this adaptivity, we first analyze artificially generated examples. We will see that LRF automatically adapts to the smoothness of the underlying data generating distribution. In the second part of this study, we show the competitive performance of our estimator on real world data.

Adaptivity for the appropriate level of smoothness

We analyze three different setups. In the first setup, we use linear models to generate the data. Here we expect ***glmnet*** to perform well and the default versions of RF as implemented in ***ranger*** and `forestry` to not perform well. In the second setup we use step functions. Here we expect RF and other tree-based algorithms to perform well and ***glmnet*** to perform relatively worse. In the third setup, there are areas that are highly non-linear and other areas that are very linear. It will thus be difficult for both the default RF algorithm and ***glmnet*** to perform well on this data set, but we expect the adaptivity of LRF to excel here.

1. **Linear Response Surface**
   In **Experiment 1**, we start with a purely linear response surface. We simulate the features from a normal distribution and we generate Y according to the following linear model:
$$\begin{aligned} &X \sim N(0,1) \in \mathbb{R}^{10},\ \varepsilon \sim N(0,4) \\ &Y = f_L(X) + \varepsilon, \\ &f_L(X) = -0.47X_2 - 0.98X_3 - 0.87X_4 + 0.63X_8 - 0.64X_{10}. \end{aligned} \tag{5.16}$$

2. **Step Functions**
   Next, we are interested in the other extreme. In **Experiment 2**, we simulate the features from a 10-dimensional normal distribution distribution, $X \sim N(0,1) \in \mathbb{R}^{10}$ and we create a regression function from a regression tree with random splits, 50 leaf nodes, and randomly sampled response values between -10 and 10 and we call this function f_S. Y is then generated according to the following model:
$$Y = f_S(X) + \varepsilon \tag{5.17}$$
$$\varepsilon \sim N(0,1) \tag{5.18}$$
   A specific description how f_S was generated can be found in detail in Appendix D.3.

3. **Linear Function with steps**
   Finally, we look at a data set that exhibits both a linear part and a step-wise constant part. Specifically, we split the space into two parts and we combine the regression functions from **Experiment 1** and **Experiment 2** to create a new regression function

as follows:

$$Y = \begin{cases} f_L(X) & \text{if } X_1 < .5 \\ f_S(X) & \text{else} \end{cases} + \varepsilon \tag{5.19}$$

$$\varepsilon \sim N(0,1). \tag{5.20}$$

The results of these simulations for a range of sample sizes can be found in Figure 5.2.

Real World data sets

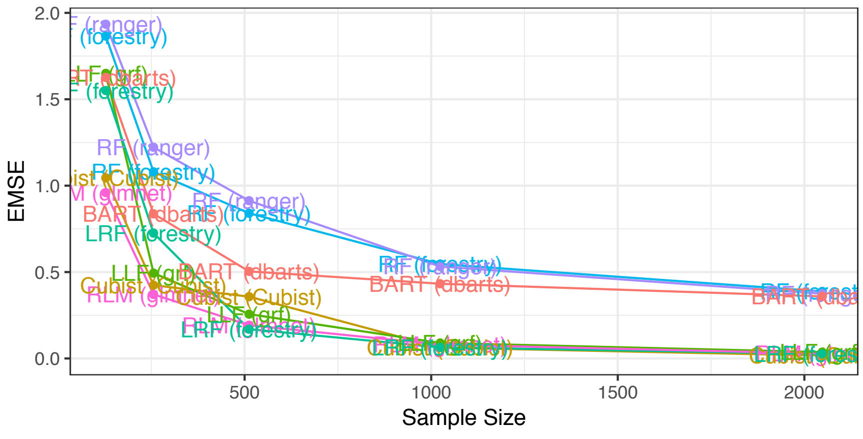
To analyze the behavior of these estimators on real data, we used a variety of data sets. Table 5.1 describes the metrics of the data sets that we have used and Table 5.2 shows the performance of the different estimators. Of these datasets, Autos describes pricing data and many related features scraped from German Ebay listings, Bike describes Capital Bikeshare user counts and corresponding weather features. These datasets are available on Kaggle [48] and the UCI repository respectively [21].

The remaining datasets were lifted from Brieman's original regression performance section [6], with the test setups altered only slightly. For the Boston, Ozone, and Servo datasets, we used 5-fold cross validation to estimate the Root Mesan Squared Error (RMSE), for the Abalone dataset, we took a test set of 50%, and for the three Friedman simulations, we kept Brieman's original scheme using 2000 simulated test points.

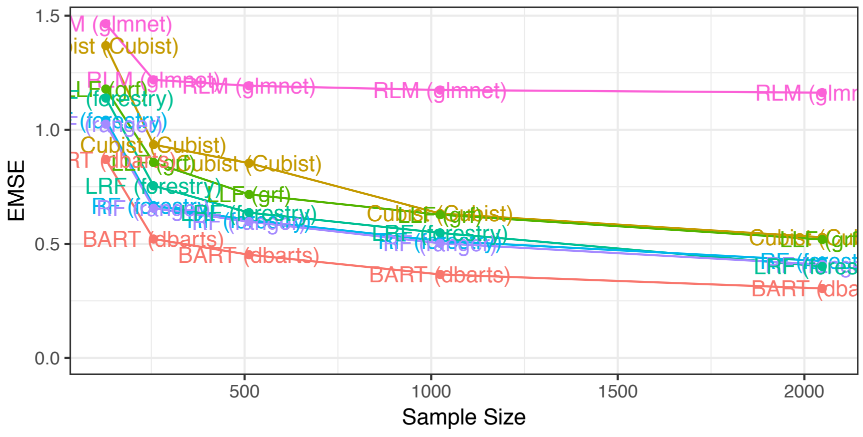
Results

name	ntrain	ntest	dim	numeric features
Abalone	2089	2088	11	11
Autos	1206	39001	12	6
Bike	869	16510	16	16
Boston	506		9	9
Friedman 1	1000	2000	11	11
Friedman 2	1000	2000	5	5
Friedman 3	1000	2000	5	5
Ozone	330		9	9
Servo	167		13	13

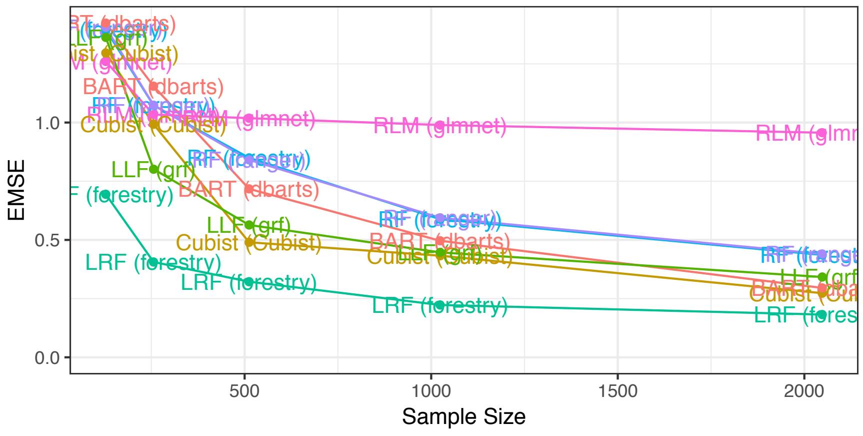
Table 5.1: Summary of real data sets.



(a) Experiment 1: Linear regression function.



(b) Experiment 2: Step function.



(c) Experiment 3: Partly linear and partly a step function.

Figure 5.2: Different levels of smoothness. In Experiment 1, the response surface is a linear function, in Experiment 2, it is a step function, and in Experiment 3, it is partly a step function and partly a linear function.

Performance on Real Datasets

	RLM (glmnet)	RF (ranger)	RF (forestry)	BART (dbarts)	Cubist (Cubist)	LCART (forestry)	local RF (grf)	Linear RF (forestry)
Abalone	2.26	2.13	2.13	2.13	2.14	2.1	2.09	2.08
autos	3269.57	2389.33	2367.11	2383.64	2553.85	2776.61	2822.74	2368.35
bike	142.47	71.4	73.08	60.84	62.85	81.9	69.38	60.84
Boston Housing	4.98	3.2	3.34	3	3.34	4.42	3.32	3.52
Friedman 1	2.69	1.9	1.91	1.21	1.28	1.45	1.91	1.21
Friedman 2	1851.25	1746.93	1746.49	1781.54	1745.11	1747.5	1746.65	1741.08
Friedman 3	0.26	0.19	0.19	0.18	0.19	0.2	0.18	0.18
Ozone	4.56	4.02	4.04	4.13	4.41	4.55	4.08	4.05
Servo	1.13	0.5	0.55	0.57	0.64	0.73	0.43	0.51

Table 5.2: Estimator RMSE compared across real data sets

The results shown in Figure 5.2 show the results for the different estimators on the simulated response surfaces detailed in Section 5.3. As expected, we find that Regularized Linear Models (RLM) as implemented in the ***glmnet*** package perform very well on Experiment 1 where the response is linear, while it performs poorly on Experiments 2 and 3. The default random forest estimators implemented for example in ***ranger*** performs very well on Experiment 2, but it performs rather poorly on Experiment 1 and 3, as estimators which can exploit the linearity benefit greatly here. The estimators which can pick up linear signals (RLM, LLF, and LRF), perform nearly identically on Experiment 1, showing a tuned LRF can mimic the linear performance of a purely linear model such as RLM. Experiment 2 showcases a pure step function, in which smooth estimators such as RLM and LLF suffer, while RF and boosting algorithms such as BART excel. In this scenario, the performance of LRF now converges to match that of the purely random forest algorithms. Finally, in Experiment 3, the response surface is evenly distributed between smooth response values, and a step function. In this case, there are areas of weakness for both the smooth estimators and the RF estimators, but the adaptivity of LRF allows it to adapt to the differences across the response surface, and fit the appropriately different estimators on each portion.

The results shown in Table 5.2 are mixed, but display the adaptivity of LRF on datasets which are both linear as well as highly nonlinear. The ability to tune the nodesize in LRF allows for fits which mimic linear models in large nodes, and closely mimic CART fits in small node sizes. With fast tuning, this range of variation can be exploited to deliver an estimator which can adapt itself well to a variety of different response surfaces. This adaptivity results in an estimator which can exploit the presence of smooth signals very well and remain competitive over a wide range of response surfaces.

5.4 Interpretability

In this section, we want to show how linear aggregation can be used to create more interpretable trees and we demonstrate this with a large data set about voter turnout. We will show the usefulness of linear aggregation to better understand the heterogeneity in the data. We first outline the problem of estimating the Conditional Average Treatment Effect and summarize a few results from the literature before applying LRF to the data set.

Social Pressure and Voter Turnout Data Set

We use a data set from a large field experiment that has been conducted by [27] to measure the effectiveness of four different mailers on voter turnout. The data set contains 344,084 potential voters in the August 2006 Michigan statewide election with a wide range of offices and proposals on the ballot. The sample was restricted to voters that voted in the 2004 general election, because the turnout in the 2006 election was expected to be extremely low among potential voters who did not vote in the 2004 general election.

The specific mailers can be found in [27] and we briefly describe the mailers as follows. Each mailer carries the message "DO YOUR CIVIC DUTY — VOTE!" and applies a different level of social pressure. We present the mailers ordered by the amount of social pressure they put on the recipients.

- Civic Duty (CD): The Civic Duty mailer contains the least amount of social pressure and only reminds the recipient that "the whole point of democracy is that citizens are active participants in government".

- Hawthorne (HT): Households receiving the Hawthorne mailer were told "YOU ARE BEING STUDIED!" and it explains that voter turnout will be studied in the upcoming August primary election, but whether or not an individual votes "will remain confidential and will not be disclosed to anyone else."

- Self (SE): The Self mailer exerts even more pressure. It informs the recipient that "WHO VOTES IS PUBLIC INFORMATION!" and contains a chart containing the information of who voted in the past two elections within the household the letter was addressed to. It also contains a field for the upcoming August 2006 election and it promises that an updated chart will be sent to the households after the election.

- Neighbors (NE): The Neighbors mailer increases the social pressure even more and starts with the rhetorical question: "WHAT IF YOUR NEIGHBORS KNEW WHETHER

YOU VOTED?". It lists the voting records not only of the people living in the household, but also of those people living close by and it promises to send an updated chart after the election. This treatment mailer implies that the voting turnout of the recipients are publicised to the people living close by and thereby creates the strongest social pressure.

The randomization was done on a household level. The set of all households was split into four treatment groups and a control group. Each of the four treatment groups contained about 11% of all voters, while the control group contained about 56% of all households.

The data set also contains seven key individual-level covariates: gender, age, and whether or not the individual voted in the primary elections in 2000, 2002, 2004, or the general election in 2000 and 2002. We also derived three further covariates from the ones above. We believe that there are voters that generally have a high voting propensity, but still did not vote in every election. To enable a tree algorithm to discriminate potential voters with high voting propensity from those with a low one, we added a new feature that we called Cumulative Voting History (CVH). It is the number of times the voter has voted within the last five election before the 2004 general election. Similarly, we define the Cumulative Primary Voting History (CPVH) as the number of primary elections a voter has participated in between 2000 and 2004 and the Cumulative General election Voting History (CGVH) as the number of general elections a voter has participated in between 2000 and 2002.

[45] analyze the effect of the Neighbors mailer while paying specific attention to uncovering the heterogeneity in the data. They find strong evidence of heterogeneous effects using the S-, T- and X-learner combined with random forests.[5] Specifically, they estimate the Conditional Average Treatment Effect (CATE),

$$\tau(x) = \mathbb{E}[Y(1) - Y(0)|X = x].$$

Here, $Y(1) \in \{0, 1\}$ is the outcome when a unit is treated (is assigned to receive the Neighbors mailer), $Y(0)$ is the outcome when the unit is in the control group, and x is the vector of covariates of unit x.

Estimating the CATE is useful for targeting treatment to individuals with a particularly strong effect. Policy makers could use these estimates to specifically target individuals with a large CATE. However, researchers are often interested in better understanding the underlying effects through plots and interpretable summaries of the estimators. This is, in particular, important for communication with subject experts and policy makers [57]. [43, 45], for example, uncover the heterogeneity by using partial dependence plots and by looking at specific subgroups that have been defined through subject knowledge and exhaustive EDAs. They find that the estimated CATE varies with the CVH and it is suggested that the treatment effect is particularly low for potential voters who did not vote in any other election before the general election in 2004 and units that voted in all past elections. These

[5]The S-, T- and X-learner are algorithms that can make use of a base algorithm such as random forests to estimate the CATE. We refer to [45] for a detailed description.

are interesting insights, but it is unsatisfying that looking into these particular subgroups had to be motivated by subject knowledge or an EDA of an independent data set.

Using a linear aggregation function in RF directly enables us to understand the underlying prediction function and we will show this in the following using the S-learner with random forests. We will see that we do not need to specify interesting subgroups and that the algorithm uncovers the heterogeneity automatically.

Making causal mechanisms more interpretable

Recall that in [45], the S-Learner estimates the outcome by using all of the features and the treatment indicator, without giving the treatment indicator a special role,

$$\hat{\mu}(x, w) = \hat{\mathbb{E}}[Y|X = x, W = w].$$

The predicted CATE for an individual unit is then the difference between the predicted values when the treatment-assignment indicator is changed from control to treatment, with all other features held fixed,

$$\hat{\tau}(x) = \hat{\mu}(x, 1) - \hat{\mu}(x, 0). \tag{5.21}$$

We are making four crucial changes to this formulation of S-RF:

1. In the original formulation, the treatment is binary. In our data set, there are, however, four different treatments and we generalize the S-learner here to work with multiple treatments at the same time. Specifically, we encode a treatment indicator for each of the four treatments and we estimate the response function using all covariates and the treatment indicators as independent variables,

   $$\hat{\mu}(x, w_1, \ldots, w_4) = \hat{\mathbb{E}}[Y|X = x, W_1 = w_1, \ldots, W_4 = w_4].$$

   The CATE for the CD mailer is then defined as

   $$\hat{\tau}(x) = \hat{\mu}(x, 1, 0, 0, 0) - \hat{\mu}(x, 0, 0, 0, 0)$$

   and the treatment effects for the other mailers is defined analogue.

2. In the usual S-RF algorithm, each tree contains several splits on the treatment indicators and the features. To compute the CATE of the NE mailer at x, the average of the leaf in which $(x, 0, 0, 0, 0)$ falls is subtracted from the average of the leaf in which $(x, 0, 0, 0, 1)$ falls.[6] In our new formulation, we choose to linearly adapt in the four treatment indicators and we allow splits on all covariates but not the variables that encode the treatment assignment. In other words, the splits are done in such a way

[6] If $(x, 0, 0, 0, 0)$ falls into a leaf L and on the path to the leaf there was no split on w_4, then also $(x, 0, 0, 0, 1)$ will fall into the same leaf and thus the predicted CATE will be 0 (c.f. 5.21). This often leads to bias, and can be beneficial when the treatment effect is close to 0.

that a ridge regularized linear model that is linear in the four treatment indicators minimizes the overall RSS. Figure 5.3 shows the first four trees of an S-RF estimator with linear aggregation in the treatment indicators. Specifically, we chose the overfit penalty, λ, to be 10^{-8} so that the splits are essentially linear and the coefficient of the linear models in each leaf can be interpreted as the average treatment effect in the particular leaf.

3. Another important variation is that for each tree, we split the training set into two disjoint sets: A splitting set that is used to create the tree structure and an aggregation set that is used to compute the leaf aggregation. This property has been studied before under different names ([68, 80]) to obtain better confidence intervals or prove mathematical properties about the RF algorithm. Here, we use it to add interpretability and to allow valid inference. Since the tree structure is independent of the aggregation set, the leaf estimates are based on a completely independent data set. Thus for a leaf, L, the regression coefficients become estimates for the average treatment effect for that leaf,
$$(\hat{\tau}, \hat{b}) = \underset{(\tau,b)}{\arg\min} \sum_{i:X_i \in L} \left(Y_i - W_i^T\tau - b\right)^2 + \lambda\|\tau\|_2^2. \tag{5.22}$$
Here $W_i \in \{0,1\}^4$ is the four dimensional vector containing the indicator for the treatment group of unit i (e.g., $W = (0,0,0,0)$ encodes the control group, $W = (1,0,0,0)$ encodes the CD group and $W = (0,1,0,0)$ encodes the HT group). X_i and Y_i are the features and the observed outcome of unit i and thus $\{i : X_i \in L\}$ is the set of all indices of all units that fall in leaf L. $(\hat{\tau}, \hat{b})$ is plotted in the leaves of each node in Figure 5.3. Take, for example, the leaf on the very left in the first tree. It contains units that have never voted in a primary (CPVH < 0.5) and who have also not voted in the 2002 general election (g2002 = no). The structure and thus the splits of this tree is based on a set that is disjoint to the subset of the training set that is used to estimate the coefficients. Thus $\hat{b}$ (in the plot `untr BL`) is an estimate for the voting propensity of voters falling into this leaf that were in the control group and the $\hat{\tau}_3$ (in the plot `TE(SE)`) is an estimate for the treatment effect of the Self mailer.

4. If we trained the trees in the random forests algorithm without any stopping criteria other than a minimal node size, we would still get very deep trees and they might even perform very well in the standard random forest aggregation. We noticed, however, that for a single tree predictor, training until purity leads to overfitting and makes it very hard to learn a lot about the underlying causal effects, since one cannot really understand trees with several hundred or even thousands of nodes. We then use the one step look-ahead stopping criteria introduced in Section 5.2.

Software Solutions

Regression trees with linear aggregation can be much shallower and therefore more interpretable without sacrificing predictive performance. However, we warn to not blindly use only one tree and instead to use several. In the example below, we created several trees. For each tree, we randomly split the training data into two parts: the splitting and the averaging set. Since this partitioning is different for each tree, we find that there is some variation in the trees and practitioners should consider multiple trees to draw rubust conclusions.

To aid researchers in analyzing regression trees, we implemented a plotting function in the `forestry` package that is fast, easy to use, and flexible. It enables researchers to quickly look at many trees in a RF algorithm. Figure 5.3 is created using this package by varying the `tree.id` parameter to look at different trees. We recommend using a smaller `minSplitGain` to draw even more personalized conclusions.

```
rft <- forestry(x = x,
                y = y,
                minSplitGain = .005,
                linear = TRUE,
                overfitPenalty = 1e-8,
                linFeats = 1:4,
                splitratio = .5)
plot(forest, tree.id = 1)
```

Interpretation of the results

Let us take a look at the four trees in Figure 5.3. These trees are not exactly the same, however, they still describe a very similar and therefore consistent behavior.

First of all, we notice that the primary voting history (CPVH) appears to be the most important variable. However, age, the 2002 general election turnout, and the overall voting history (CVH) appear to be useful features carrying a lot of information about the voting propensity of the voters and the magnitude of the turnout.

In each tree, there is a leaf node of potential voters who never or very rarely voted in the five elections before the 2004 general election. Unsurprisingly, all of these units have a very low base line estimate for voting in the 2004 primary election (the outcome of interest) when they are not treated. A bit more surprising, however, is that the treatment effects of the four mailers appear to be rather small for these units. This contradicts our intuition that for a group of potential voters with a low baseline, there is a lot of space for improvement and one might expect large treatment effects.

We also find that in each of the trees there is a subgroup of units that have voted in at least two out of the three recorded primary elections. As expected, for these units, the voting propensity is very high even if they do not receive any of the mailers. The treatment effects are very large as well, which is again a bit surprising. After all, for a potential voter i who would vote if they did not receive any of the mailers ($Y(0) = 1$), the individual treatment effect, $D_i = Y_i(1) - Y_i(0)$, can only be 0 or -1.

It is also interesting to see that there are a lot of splits on age at around 53 years. Older voters tend to have a much higher baseline turnout of approximately 10%. The treatment effects here are very heterogeneous: All trees suggest that among voters who voted in at least one primary (CPVH $> .5$ or CVH > 2.5),[7] the treatment effects of the different mailers appear to be pretty similar for the two age groups. On the contrary, for potential voters who voted in one general election, but never in a primary, the trees suggest that the treatment effect of the SE and NE mailers is larger for younger voters.

5.5 Conclusion

In this paper, we have proposed three main changes to the core CART algorithm and explored their consequences in several examples. Specifically, our changes were focused on three main additions: introducing linear aggregation functions for prediction in the leaf nodes of the estimator, modifying the CART splitting algorithm to optimize the linear aggregation functions on each node resulting from the split, and adding a stopping criteria which limits further node creation when splitting would not increase the R^2 by a predetermined percentage.

In order to explore these contributions, we analyzed the increase in predictive power over a range of data sets in Section 5.3. Here we also highlighted the adaptivity of LRF as an estimator by using a wide range of node sizes and regularization parameters. We then noted how the linear aggregation leads to estimators that are much simpler to understand in Section 5.4. We utilized the stopping criteria to build trees that have high performance as well as offer inference through both the selected splits and resulting regression coefficients.

We have already implemented a version of gradient boosting in the `forestry` package, and we are very interested in evaluating the effect of linear aggregation on gradient boosting. We also believe that LRF can be less biased than the usual versions of RF, and we wonder whether it would improve nonparametric confidence interval estimation. Finally, we believe that the regression coefficients could be very informative and we would like to study statistical tests that can be used to determine whether the estimated coefficients for the leaves are statistically provably different.

[7]Since there have been only two general elections reported, a potential voter with CVH > 2.5 implies that they must have voted in at least one primary.

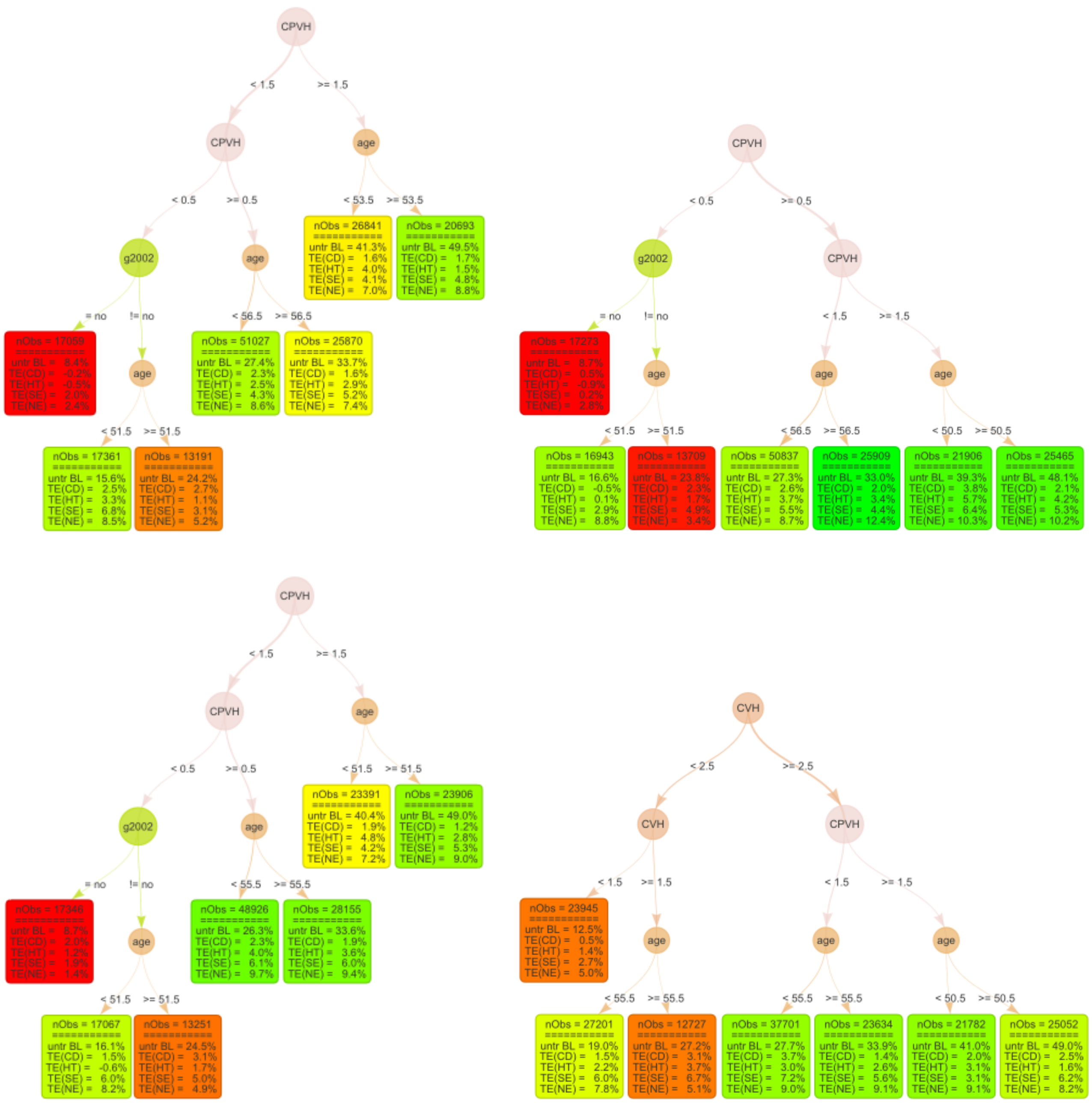

CPVH
< 1.5
>= 1.5
CPVH
age
< 0.5
>= 0.5
< 53.5
>= 53.5
nObs = 26841 =========== untr BL = 41.3% TE(CD) = 1.6% TE(HT) = 4.0% TE(SE) = 4.1% TE(NE) = 7.0%
nObs = 20693 =========== untr BL = 49.5% TE(CD) = 1.7% TE(HT) = 1.5% TE(SE) = 4.8% TE(NE) = 8.8%
g2002
age
= no
!= no
< 56.5
>= 56.5
nObs = 17059 =========== untr BL = 8.4% TE(CD) = -0.2% TE(HT) = -0.5% TE(SE) = 2.0% TE(NE) = 2.4%
age
nObs = 51027 =========== untr BL = 27.4% TE(CD) = 2.3% TE(HT) = 2.5% TE(SE) = 4.3% TE(NE) = 8.6%
nObs = 25870 =========== untr BL = 33.7% TE(CD) = 1.6% TE(HT) = 2.9% TE(SE) = 5.2% TE(NE) = 7.4%
< 51.5
>= 51.5
nObs = 17361 =========== untr BL = 15.6% TE(CD) = 2.5% TE(HT) = 3.3% TE(SE) = 6.8% TE(NE) = 8.5%
nObs = 13191 =========== untr BL = 24.2% TE(CD) = 2.7% TE(HT) = 1.1% TE(SE) = 3.1% TE(NE) = 5.2%
CPVH
< 0.5
>= 0.5
g2002
CPVH
= no
!= no
< 1.5
>= 1.5
nObs = 17273 =========== untr BL = 8.7% TE(CD) = 0.5% TE(HT) = -0.9% TE(SE) = 0.2% TE(NE) = 2.8%
age
age
age
< 51.5
>= 51.5
< 56.5
>= 56.5
< 50.5
>= 50.5
nObs = 16943 =========== untr BL = 16.6% TE(CD) = -0.5% TE(HT) = 0.1% TE(SE) = 2.9% TE(NE) = 8.8%
nObs = 13709 =========== untr BL = 23.8% TE(CD) = 2.3% TE(HT) = 1.7% TE(SE) = 4.9% TE(NE) = 3.4%
nObs = 50837 =========== untr BL = 27.3% TE(CD) = 2.6% TE(HT) = 3.7% TE(SE) = 5.5% TE(NE) = 8.7%
nObs = 25909 =========== untr BL = 33.0% TE(CD) = 2.0% TE(HT) = 3.4% TE(SE) = 4.4% TE(NE) = 12.4%
nObs = 21906 =========== untr BL = 39.3% TE(CD) = 3.8% TE(HT) = 5.7% TE(SE) = 6.4% TE(NE) = 10.3%
nObs = 25465 =========== untr BL = 48.1% TE(CD) = 2.1% TE(HT) = 4.2% TE(SE) = 5.3% TE(NE) = 10.2%
CPVH
< 1.5
>= 1.5
CPVH
age
< 0.5
>= 0.5
< 51.5
>= 51.5
nObs = 23391 =========== untr BL = 40.4% TE(CD) = 1.9% TE(HT) = 4.8% TE(SE) = 4.2% TE(NE) = 7.2%
nObs = 23906 =========== untr BL = 49.0% TE(CD) = 1.2% TE(HT) = 2.8% TE(SE) = 5.3% TE(NE) = 9.0%
g2002
age
= no
!= no
< 55.5
>= 55.5
nObs = 17346 =========== untr BL = 8.7% TE(CD) = 2.0% TE(HT) = 1.2% TE(SE) = 1.9% TE(NE) = 1.4%
age
nObs = 48926 =========== untr BL = 26.3% TE(CD) = 2.3% TE(HT) = 4.0% TE(SE) = 6.1% TE(NE) = 9.7%
nObs = 28155 =========== untr BL = 33.6% TE(CD) = 1.9% TE(HT) = 3.6% TE(SE) = 6.0% TE(NE) = 9.4%
< 51.5
>= 51.5
nObs = 17067 =========== untr BL = 16.1% TE(CD) = 1.5% TE(HT) = -0.6% TE(SE) = 6.0% TE(NE) = 8.2%
nObs = 13251 =========== untr BL = 24.5% TE(CD) = 3.1% TE(HT) = 1.7% TE(SE) = 5.0% TE(NE) = 4.9%
CVH
< 2.5
>= 2.5
CVH
CPVH
< 1.5
>= 1.5
< 1.5
>= 1.5
nObs = 23945 =========== untr BL = 12.5% TE(CD) = 0.5% TE(HT) = 1.4% TE(SE) = 2.7% TE(NE) = 5.0%
age
age
age
< 55.5
>= 55.5
< 55.5
>= 55.5
< 50.5
>= 50.5
nObs = 27201 =========== untr BL = 19.0% TE(CD) = 1.5% TE(HT) = 2.2% TE(SE) = 6.0% TE(NE) = 7.8%
nObs = 12727 =========== untr BL = 27.2% TE(CD) = 3.1% TE(HT) = 3.7% TE(SE) = 6.7% TE(NE) = 5.1%
nObs = 37701 =========== untr BL = 27.7% TE(CD) = 3.7% TE(HT) = 3.0% TE(SE) = 7.2% TE(NE) = 9.0%
nObs = 23634 =========== untr BL = 33.9% TE(CD) = 1.4% TE(HT) = 2.6% TE(SE) = 5.6% TE(NE) = 9.1%
nObs = 21782 =========== untr BL = 41.0% TE(CD) = 2.0% TE(HT) = 3.1% TE(SE) = 3.1% TE(NE) = 9.1%
nObs = 25052 =========== untr BL = 49.0% TE(CD) = 2.5% TE(HT) = 1.6% TE(SE) = 6.2% TE(NE) = 8.2%

Figure 5.3: The first four trees of the S-Learner as described in Section 5.4. The first row in each leaf contains the number of observations in the aggregation set that fall into the leaf. The second part of each leaf displays the regression coefficients. `untr BL` stands for untreated Base Line and it can be interpreted as the proportion of units that fall into that leaf who voted in the 2004 primary election. `TE(CD)` can be interpreted as an estimate for the ATE of the CD mailer within the leaf. The color are chosen to represent the size of the treatment effect for the neighbors treatment. Eg., red represents leaves with a low treatment effect for the Neighbors mailer, TE(NE).

Bibliography

[1] Alberto Abadie. “Semiparametric difference-in-differences estimators”. In: *The Review of Economic Studies* 72.1 (2005), pp. 1–19.

[2] Susan Athey and Guido W Imbens. “Recursive partitioning for heterogeneous causal effects”. In: *Proceedings of the National Academy of Sciences of the United States of America* 113.27 (2016), pp. 7353–7360.

[3] Susan Athey, Julie Tibshirani, Stefan Wager, et al. “Generalized random forests”. In: *The Annals of Statistics* 47.2 (2019), pp. 1148–1178.

[4] Peter J Bickel and Kjell A Doksum. *Mathematical statistics: Basic ideas and selected topics*. Vol. 2. CRC Press, (2015).

[5] Adam Bloniarz et al. “Supervised neighborhoods for distributed nonparametric regression”. In: *Artificial Intelligence and Statistics*. 2016, pp. 1450–1459.

[6] Leo Breiman. “Random forests”. In: *Machine Learning* 45.1 (2001), pp. 5–32.

[7] Leo Breiman and WS Meisel. “General estimates of the intrinsic variability of data in nonlinear regression models”. In: *Journal of the American Statistical Association* 71.354 (1976), pp. 301–307.

[8] Leo Breiman et al. “Classification and regression trees. Wadsworth Int”. In: *Group* 37.15 (1984), pp. 237–251.

[9] David Broockman and Joshua Kalla. “Durably reducing transphobia: A field experiment on door-to-door canvassing”. In: *Science* 352.6282 (2016), pp. 220–224.

[10] David Broockman, Joshua Kalla, and Peter Aronow. “Irregularities in LaCour”. In: *Work. pap., Stanford Univ.* (2014).

[11] David E Broockman, Joshua L Kalla, and Jasjeet S Sekhon. “The design of field experiments with survey outcomes: A framework for selecting more efficient, robust, and ethical designs”. In: *Political Analysis* 25 (2017), pp. 435–464.

[12] Peter Bühlmann, Bin Yu, et al. “Analyzing bagging”. In: *The Annals of Statistics* 30.4 (2002), pp. 927–961.

[13] Probal Chaudhuri et al. “Piecewise-polynomial regression trees”. In: *Statistica Sinica* (1994), pp. 143–167.

[14] Tianqi Chen and Carlos Guestrin. “XGBoost: A Scalable Tree Boosting System”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. ACM, (2016), pp. 785–794.

[15] Hugh A Chipman, Edward I George, Robert E McCulloch, et al. “BART: Bayesian additive regression trees”. In: *The Annals of Applied Statistics* 4.1 (2010), pp. 266–298.

[16] William S Cleveland and Susan J Devlin. “Locally weighted regression: an approach to regression analysis by local fitting”. In: *Journal of the American statistical association* 83.403 (1988), pp. 596–610.

[17] Alexander D’Amour et al. “Overlap in Observational Studies with High-Dimensional Covariates”. In: *arXiv preprint arXiv:1711.02582* (2017).

[18] Arthur P Dempster. “The Dempster–Shafer calculus for statisticians”. In: *International Journal of approximate reasoning* 48.2 (2008), pp. 365–377.

[19] Vincent Dorie et al. “Automated versus do-it-yourself methods for causal inference: Lessons learned from a data analysis competition”. In: *arXiv preprint arXiv:1707.02641* (2017).

[20] Bradley Efron. “Estimation and accuracy after model selection”. In: *Journal of the American Statistical Association* 109.507 (2014), pp. 991–1007.

[21] Hadi Fanaee-T. *Capital Bikeshare Database*. Hourly and daily count of rental bikes between years 2011 and 2012 in Capital bikeshare system with the corresponding weather and seasonal information. 2013. URL: `https://archive.ics.uci.edu/ml/datasets/bike+sharing+dataset`.

[22] Jared C Foster. “Subgroup Identification and Variable Selection from Randomized Clinical Trial Data”. PhD thesis. The University of Michigan, 2013, pp. 1–97.

[23] Rina Friedberg et al. “Local linear forests”. In: *arXiv preprint arXiv:1807.11408* (2018).

[24] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. “Regularization paths for generalized linear models via coordinate descent”. In: *Journal of statistical software* 33.1 (2010), p. 1.

[25] Jerome H Friedman. “Greedy function approximation: a gradient boosting machine”. In: *Annals of statistics* (2001), pp. 1189–1232.

[26] João Gama. “Functional trees”. In: *Machine Learning* 55.3 (2004), pp. 219–250.

[27] Alan S Gerber, Donald P Green, and Christopher W Larimer. “Social pressure and voter turnout: Evidence from a large-scale field experiment”. In: *American Political Science Review* 102.1 (2008), pp. 33–48.

[28] Adam Glynn and Kevin Quinn. *CausalGAM: Estimation of Causal Effects with Generalized Additive Models*. R package version 0.1-4. (2017).

[29] Donald P Green and Holger L Kern. “Modeling heterogeneous treatment effects in survey experiments with Bayesian additive regression trees”. In: *Public Opinion Quarterly* 76.3 (2012), pp. 491–511.

[30] László Györfi et al. *A distribution-free theory of nonparametric regression.* Springer Science & Business Media, (2006).

[31] Jaroslav Hájek. "On basic concepts of statistics". In: *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probabilities.* Vol. 1. (1967), pp. 139–162.

[32] Ben B Hansen and Jake Bowers. "Attributing effects to a cluster-randomized get-out-the-vote campaign". In: *Journal of the American Statistical Association* 104.487 (2009), pp. 873–885.

[33] James J Heckman, Jeffrey Smith, and Nancy Clements. "Making the most out of programme evaluations and social experiments: Accounting for heterogeneity in programme impacts". In: *The Review of Economic Studies* 64.4 (1997), pp. 487–535.

[34] Nicholas C. Henderson et al. "Bayesian analysis of heterogeneous treatment effects for patient-centered outcomes research". In: *Health Services and Outcomes Research Methodology* 16.4 (2016), pp. 213–233.

[35] Jennifer L Hill. "Bayesian nonparametric modeling for causal inference". In: *Journal of Computational and Graphical Statistics* 20.1 (2011), pp. 217–240.

[36] Arthur E Hoerl and Robert W Kennard. "Ridge regression: Biased estimation for nonorthogonal problems". In: *Technometrics* 12.1 (1970), pp. 55–67.

[37] Torsten Hothorn et al. "Bagging survival trees". In: *Statistics in medicine* 23.1 (2004), pp. 77–91.

[38] Joshua L Kalla and David E Broockman. "The Minimal Persuasive Effects of Campaign Contact in General Elections: Evidence from 49 Field Experiments". In: *American Political Science Review* 112.1 (2018), pp. 148–166.

[39] Aram Karalič. "Employing linear regression in regression tree leaves". In: *Proceedings of the 10th European conference on Artificial intelligence.* John Wiley & Sons, Inc. 1992, pp. 440–441.

[40] Max Kuhn et al. "Building predictive models in R using the caret package". In: *Journal of statistical software* 28.5 (2008), pp. 1–26.

[41] Sören Künzel et al. *causalToolbox: Toolbox for Causal Inference with emphasize on Heterogeneous Treatment Effect Estimator.* R package version 0.0.1.000. (2018).

[42] Sören Künzel et al. *hte: An implementation of Heterogeneous Treatment Effect Estimators and Honest Random Forests in C++ and R.* 2017. URL: `https://github.com/soerenkuenzel/hte`.

[43] Sören R Künzel, Simon JS Walter, and Jasjeet S Sekhon. "Causaltoolbox—Estimator Stability for Heterogeneous Treatment Effects". In: *arXiv preprint arXiv:1811.02833* (2018).

[44] Sören R. Künzel et al. *Forestry—CART, random forests, and gradient boosting algorithms for inference and interpretability.* 2019.

[45] Sören R Künzel et al. “Metalearners for estimating heterogeneous treatment effects using machine learning”. In: *Proceedings of the National Academy of Sciences* 116.10 (2019), pp. 4156–4165.

[46] Sören R Künzel et al. “Transfer Learning for Estimating Causal Effects using Neural Networks”. In: *arXiv preprint arXiv:1808.07804* (2018).

[47] Lucien Le Cam. “On the asymptotic theory of estimation and testing hypotheses”. In: *Proceedings of the Third Berkeley Symposium on Mathematical Statistics and Probability.* Vol. 1. (1956).

[48] Orges Leka. *Used Cars Database.* Over 370,000 used cars scraped from Ebay Kleinanzeigen. 2016. URL: `https://www.kaggle.com/orgesleka/used-cars-database/metadata`.

[49] Daniel Lewandowski, Dorota Kurowicka, and Harry Joe. “Generating random correlation matrices based on vines and extended onion method”. In: *Journal of Multivariate Analysis* 100.9 (2009), pp. 1989–2001.

[50] Ker-Chau Li, Heng-Hui Lue, and Chun-Houh Chen. “Interactive tree-structured regression via principal Hessian directions”. In: *Journal of the American Statistical Association* 95.450 (2000), pp. 547–560.

[51] Yi Lin and Yongho Jeon. “Random forests and adaptive nearest neighbors”. In: *Journal of the American Statistical Association* 101.474 (2006), pp. 578–590.

[52] Hanzhong Liu and Bin Yu. “Asymptotic properties of Lasso+mLS and Lasso+Ridge in sparse high-dimensional linear regression”. In: *Electronic Journal of Statistics* 7 (2013), pp. 3124–3169.

[53] Christopher B Mann. “Is there backlash to social pressure? A large-scale field experiment on voter mobilization”. In: *Political Behavior* 32.3 (2010), pp. 387–407.

[54] Nicolai Meinshausen. “Quantile regression forests”. In: *Journal of Machine Learning Research* 7.Jun (2006), pp. 983–999.

[55] Melissa R Michelson. “The risk of over-reliance on the Institutional Review Board: An approved project is not always an ethical project”. In: *PS: Political Science & Politics* 49.02 (2016), pp. 299–303.

[56] James N Morgan and John A Sonquist. “Problems in the analysis of survey data, and a proposal”. In: *Journal of the American statistical association* 58.302 (1963), pp. 415–434.

[57] W James Murdoch et al. “Interpretable machine learning: definitions, methods, and applications”. In: *arXiv preprint arXiv:1901.04592* (2019).

[58] Xinkun Nie and Stefan Wager. “Learning Objectives for Treatment Effect Estimation”. In: *arXiv preprint arXiv:1712.04912* (2017).

[59] Scott Powers et al. “Some methods for heterogeneous treatment effect estimation in high dimensions”. In: *Statistics in medicine* (2018).

[60] Hein Putter and Willem R Van Zwet. "Resampling: consistency of substitution estimators". In: *Selected Works of Willem van Zwet*. Springer, (2012), pp. 245–266.

[61] John R Quinlan et al. "Learning with continuous classes". In: *5th Australian joint conference on artificial intelligence*. Vol. 92. World Scientific. 1992, pp. 343–348.

[62] Peter M Robinson. "Root-N-consistent semiparametric regression". In: *Econometrica* 56.4 (1988), pp. 931–954.

[63] Paul R. Rosenbaum. *sensitivitymv: Sensitivity Analysis in Observational Studies*. R package version 1.4.3. (2018).

[64] Paul R Rosenbaum and Donald B Rubin. "The central role of the propensity score in observational studies for causal effects". In: *Biometrika* 70.1 (1983), pp. 41–55.

[65] Daniel Rubin and Mark J van der Laan. "A doubly robust censoring unbiased transformation". In: *The international journal of biostatistics* 3.1 (2007).

[66] Donald B Rubin. "Estimating causal effects of treatments in randomized and nonrandomized studies." In: *Journal of Educational Psychology* 66.5 (1974), p. 688.

[67] Thomas Rusch and Achim Zeileis. "Gaining insight with recursive partitioning of generalized linear models". In: *Journal of Statistical Computation and Simulation* 83.7 (2013), pp. 1301–1315.

[68] Erwan Scornet, Gérard Biau, and Jean-Philippe Vert. "Consistency of random forests". In: *The Annals of Statistics* 43.4 (2015), pp. 1716–1741.

[69] Jasjeet S. Sekhon. "Multivariate and Propensity Score Matching Software with Automated Balance Optimization: The Matching Package for R". In: *Journal of Statistical Software* 42.7 (2011), pp. 1–52.

[70] Jasjeet S Sekhon and Yotam Shem-Tov. "Inference on a New Class of Sample Average Treatment Effects". In: *arXiv preprint arXiv:1708.02140* (2017).

[71] Jerzy Splawa-Neyman, Dorota M Dabrowska, and TP Speed. "On the application of probability theory to agricultural experiments." In: *Statistical Science* 5.4 (1990), pp. 465–472.

[72] Charles J Stone. "Consistent nonparametric regression". In: *The annals of statistics* (1977), pp. 595–620.

[73] Charles J Stone. "Optimal global rates of convergence for nonparametric regression". In: *The Annals of Statistics* 10.4 (1982), pp. 1040–1053.

[74] Vladimir Svetnik et al. "Random forest: a classification and regression tool for compound classification and QSAR modeling". In: *Journal of chemical information and computer sciences* 43.6 (2003), pp. 1947–1958.

[75] Matt Taddy et al. "A nonparametric bayesian analysis of heterogenous treatment effects in digital experimentation". In: *Journal of Business & Economic Statistics* 34.4 (2016), pp. 661–672.

[76] Lu Tian et al. "A simple method for estimating interactions between a treatment and a large number of covariates". In: *Journal of the American Statistical Association* 109.508 (2014), pp. 1517–1532.

[77] Luís Torgo. "Functional Models for Regression Tree Leaves". In: *Proceedings of the Fourteenth International Conference on Machine Learning*. ICML '97. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, pp. 385–393. ISBN: 1-55860-486-3. URL: `http://dl.acm.org/citation.cfm?id=645526.657280`.

[78] Wouter G Touw et al. "Data mining in the Life Sciences with Random Forest: a walk in the park or lost in the jungle?" In: *Briefings in bioinformatics* 14.3 (2012), pp. 315–326.

[79] Alexandre B Tsybakov. *Introduction to nonparametric estimation*. Springer Series in Statistics, (2009).

[80] Stefan Wager and Susan Athey. "Estimation and inference of heterogeneous treatment effects using random forests". In: *Journal of the American Statistical Association* (2017).

[81] Simon Walter, Jasjeet Sekhon, and Bin Yu. "Analyzing the modified outcome for heterogeneous treatment effect estimation". In: *Unpublished manuscript* (2018).

[82] Marvin N Wright and Andreas Ziegler. "ranger: A fast implementation of random forests for high dimensional data in C++ and R". In: *arXiv preprint arXiv:1508.04409* (2015).

[83] Bin Yu. "Stabiilty". In: *Bernoulli* 19 (2013), pp. 1484–1500.

[84] Bin Yu and Karl Kumbier. "Three principles of data science: predictability, computability, and stability (PCS)". In: *arXiv preprint arXiv:1901.08152* (2019).

[85] Achim Zeileis, Torsten Hothorn, and Kurt Hornik. "Model-based recursive partitioning". In: *Journal of Computational and Graphical Statistics* 17.2 (2008), pp. 492–514.

[86] Ruoqing Zhu, Donglin Zeng, and Michael R Kosorok. "Reinforcement learning trees". In: *Journal of the American Statistical Association* 110.512 (2015), pp. 1770–1784.

Appendix A

Supporting Information for Meta-learners for Estimating Heterogeneous Treatment Effects using Machine Learning

A.1 Simulation Studies

In this section, we compare the S-, T-, and X-learners in several simulation studies. We examine prototypical situations where one learner is preferred to the others. In practice, we recommend choosing powerful machine-learning algorithms such as BART [35], Neural Networks, or RFs for the base learners, since such methods perform well for a large variety of data sets. In what follows, we choose all the base learners to be either BART or honest RF algorithms—as implemented in the `hte` R package [42]—and we refer to these meta-learners as S–RF, T–RF, X–RF, S–BART, T–BART, and X–BART, respectively. Using two machine-learning algorithms as base learners helps us to demonstrate that our conclusions about the performance of the different meta learners is often independent of the particular base learner. For example, for all our simulation results we observe that if X–RF outperforms T–RF, then X–BART also outperforms T–BART.

Remark 3 (BART and RF) *BART and RF are regression tree-based algorithms that use all observations for each prediction, and they are in that sense global methods. However, BART seems to use global information more seriously than RF, and it performs particularly well when the data-generating process exhibits some global structures (e.g., global sparsity or linearity). RF, on the other hand, is relatively better when the data has some local structure that does not necessarily generalize to the entire space.*

Causal Forests

An estimator closely related to T–RF and S–RF is Causal Forests (CF) [80], because all three of these estimators can be defined as

$$\hat{\tau}(x) = \hat{\mu}(x, w = 1) - \hat{\mu}(x, w = 0),$$

where $\hat{\mu}(x, w)$ is a form of random forest with different constraints on the split on the treatment assignment, W. To be precise, in the S-learner the standard squared error loss function will decide where to split on W, and it can therefore happen anywhere in the tree. In the T-learner the split on W must occur at the very beginning.[1] For CF the split on W is always made to be the split right before the terminal leaves. To obtain such splits, the splitting criterion has to be changed, and we refer to [80] for a precise explanation of the algorithm. Figure A.1 shows the differences between these learners for full trees with 16 leaves.

CF is not a meta-learner since the random forests algoirthm has to be changed. However, its similarity to T–RF and S–RF makes it interesting to evaluate its performance. Furthermore, one could conceivably generalize CF to other tree-based learners such as BART. However, this has not been done yet, and we will therefore compare CF in the following simulations to S–, T–, and X–RF.

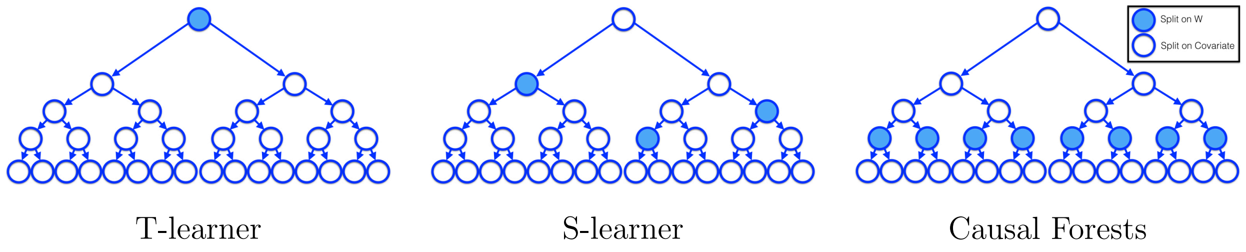


Figure A.1: Illustration of the structural form of the trees in T–RF, S–RF, and CF.

Simulation setup

Let us here introduce the general framework of the following simulations. For each simulation, we specify: the propensity score, e; the response functions, μ_0 and μ_1; the dimension, $d \in \mathbb{N}$, of the feature space; and a parameter, α, which specifies the amount of confounding between features. To simulate an observation, i, in the training set, we simulate its feature vector, X_i, its treatment assignment, W_i, and its observed outcome, Y_i, independently in the following way:

[1] In the original statement of the algorithm we train separate RF estimators for each of the treatment groups, but they are equivalent.

1. First, we simulate a d-dimensional feature vector,
$$X_i \overset{iid}{\sim} \mathcal{N}(0, \Sigma), \tag{A.1}$$
where Σ is a correlation matrix that is created using the `vine` method [49].

2. Next, we create the potential outcomes according to
$$\begin{aligned} Y_i(1) &= \mu_1(X_i) + \varepsilon_i(1), \\ Y_i(0) &= \mu_0(X_i) + \varepsilon_i(0), \end{aligned}$$
where $\varepsilon_i(1), \varepsilon_i(0) \overset{iid}{\sim} \mathcal{N}(0, 1)$ and independent of X_i.

3. Finally, we simulate the treatment assignment according to
$$W_i \sim \mathrm{Bern}(e(X_i)),$$
we set $Y_i = Y(W_i)$, and we obtain (X_i, W_i, Y_i).[2]

We train each CATE estimator on a training set of N units, and we evaluate its performance against a test set of 10^5 units for which we know the true CATE. We repeat each experiment 30 times, and we report the averages.

The unbalanced case with a simple CATE

We have already seen in Theorem 2 that the X-learner performs particularly well when the treatment group sizes are very unbalanced. We verify this effect as follows. We choose the propensity score to be constant and very small, $e(x) = 0.01$, such that on average only one percent of the units receive treatment. Furthermore, we choose the response functions in such a way that the CATE function is comparatively simple to estimate.

Simulation 1 (unbalanced treatment assignment)

$$\begin{aligned} e(x) &= 0.01, \quad d = 20, \\ \mu_0(x) &= x^T\beta + 5\ \mathbb{I}(x_1 > 0.5), \quad \textit{with} \ \ \beta \sim \textit{Unif}\left([-5, 5]^{20}\right), \\ \mu_1(x) &= \mu_0(x) + 8\ \mathbb{I}(x_2 > 0.1). \end{aligned}$$

The CATE function $\tau(x) = 8\ \mathbb{I}(x_2 > 0.1)$ is a one-dimensional indicator function, and thus simpler than the 20-dim function for the response functions $\mu_0(\cdot)$ and $\mu_1(\cdot)$. We can see in Figure A.2 that the X-learner indeed performs much better in this unbalanced setting with both BART and RF as base learners.

[2]This is slightly different from the DGP we were considering for our theoretical results, because here m, the number of control units, and n, the number of treated units, are both random. The difference is, however, very small, since in our setups $N = m + n$ is very large.

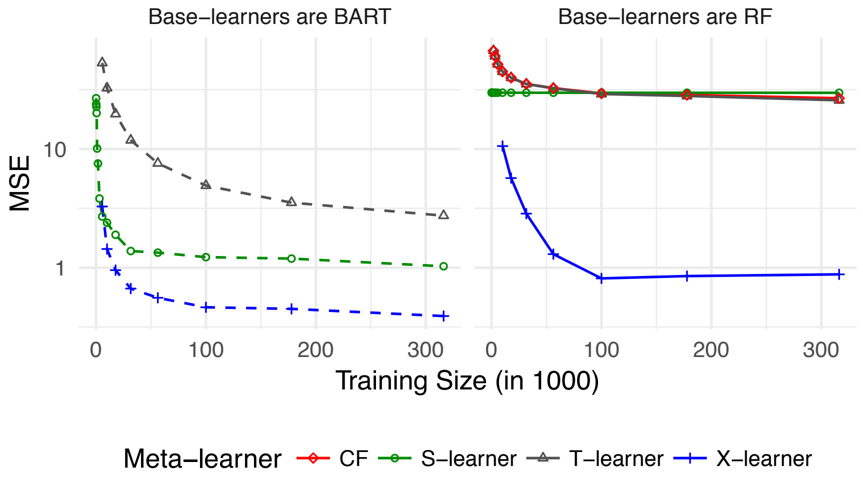


Figure A.2: Comparison of S–, T–, and X–BART (left) and S–, T–, and X–RF and CF (right) for Simulation 1.

Balanced cases without confounding

Next, let us analyze two extreme cases: In one of them the CATE function is very complex and in the other one the CATE function is equal to zero. We will show that for the case of no treatment effect, the S-learner performs very well since it sometimes does not split on the treatment indicator at all and it tends to be biased toward zero. On the other hand, for the complex CATE case simulation we have chosen, there is nothing to be learned from the treatment group about the control group and vice versa. Here the T-learner performs very well, while the S-learner is often biased toward zero. Unlike the T-learner, the X-learner pools the data, and it therefore performs well in the simple CATE case. And unlike the S-learner, the X-learner is not biased toward zero. It therefore performs well in both cases.

Complex CATE

Let us first consider the case where the treatment effect is as complex as the response functions in the sense that it does not satisfy regularity conditions (such as sparsity or linearity) that the response functions do not satisfy. We study two simulations here, and we choose for both the feature dimension to be $d = 20$, and the propensity score to be $e(x) = 0.5$. In the first setup (complex linear) the response functions are different linear functions of the entire feature space.

Simulation 2 (complex linear)

$$\begin{aligned} e(x) &= 0.5, \quad d = 20, \\ \mu_1(x) &= x^T\beta_1, \ \textit{with } \beta_1 \sim \textit{Unif}([1,30]^{20}), \\ \mu_0(x) &= x^T\beta_0, \ \textit{with } \beta_0 \sim \textit{Unif}([1,30]^{20}). \end{aligned}$$

The second setup (complex non-linear) is motivated by [80]. Here the response function are non-linear functions.

Simulation 3 (complex non-linear)

$$\begin{aligned} e(x) &= 0.5, \quad d = 20, \\ \mu_1(x) &= \frac{1}{2}\varsigma(x_1)\varsigma(x_2), \\ \mu_0(x) &= -\frac{1}{2}\varsigma(x_1)\varsigma(x_2) \end{aligned}$$

with

$$\varsigma(x) = \frac{2}{1 + e^{-12(x-1/2)}}.$$

Figure A.3 shows the MSE performance of the different learners. In this case, it is best to separate the CATE estimation problem into the two problems of estimating μ_0 and μ_1 since there is nothing one can learn from the other assignment group. The T-learner follows exactly this strategy and should perform very well. The S-learner, on the other hand, pools the data and needs to learn that the response function for the treatment and the response function for the control group are very different. However, in the simulations we study here, the difference seems to matter only very little.

Another interesting insight is that choosing BART or RF as the base learner can matter a great deal. BART performs very well when the response surfaces satisfy global properties such as being globally linear, as in Simulation 2. However, in Simulation 3, the response surfaces do not satisfy such global properties. Here the optimal splitting policy differs throughout the space and this non-global behavior is harmful to BART. Thus, choosing RF as the base learners results in a better performance here. Researchers should use their subject knowledge when choosing the right base learner.

No treatment effect

Let us now consider the other extreme where we choose the response functions to be equal. This leads to a zero treatment effect, which is very favorable for the S-learner. We will again consider two simulations where the feature dimension is 20, and the propensity score is constant and 0.5.

We start with a global linear model (Simulation 4) for both response functions. In Simulation 5, we simulate some interaction by slicing the space into three parts, $\{x : x_{20} <$

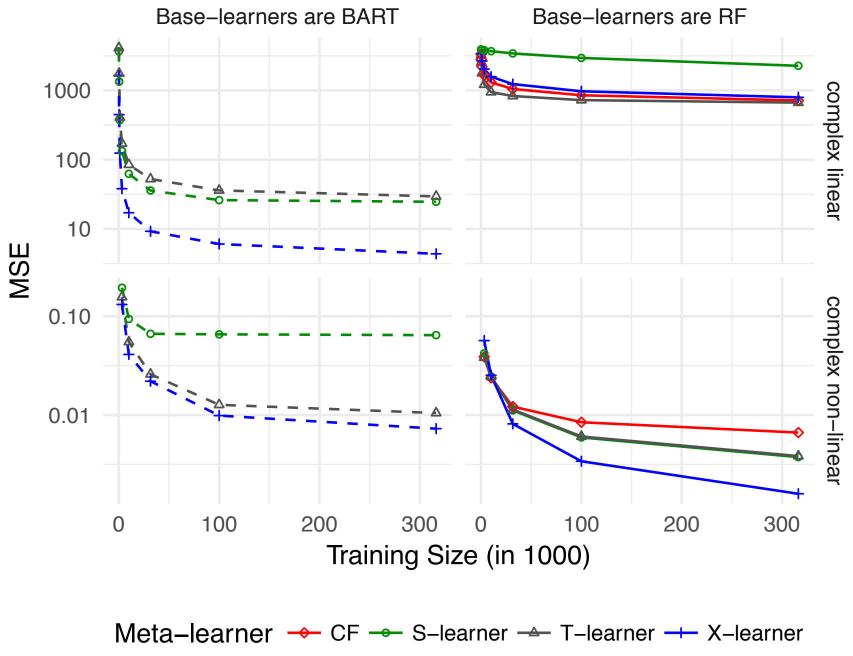


Figure A.3: Comparison of the S-, T-, and X-learners with BART (left) and RF (right) as base learners for Simulation 2 (top) and Simulation 3 (bottom).

$-0.4\}$, $\{x : -0.4 < x_{20} < 0.4\}$, and $\{x : 0.4 < x_{20}\}$, where for each of the three parts of the space a different linear response function holds. We do this because we believe that in many data sets there is a local structure, that appears only in some parts of the space.

Simulation 4 (global linear)

$$\begin{aligned} e(x) &= 0.5, \quad d = 5, \\ \mu_0(x) &= x^T\beta, \ \textit{with } \beta \sim \textit{Unif}([1,30]^5), \\ \mu_1(x) &= \mu_0(x). \end{aligned}$$

Simulation 5 (piecewise linear)

$$\begin{aligned} e(x) &= 0.5, \quad d = 20, \\ \mu_0(x) &= \begin{cases} x^T\beta_l & \textit{if} \ \ x_{20} < -0.4 \\ x^T\beta_m & \textit{if} \ \ -0.4 \le x_{20} \le 0.4 \\ x^T\beta_u & \textit{if} \ \ 0.4 < x_{20}, \end{cases} \\ \mu_1(x) &= \mu_0(x), \end{aligned}$$

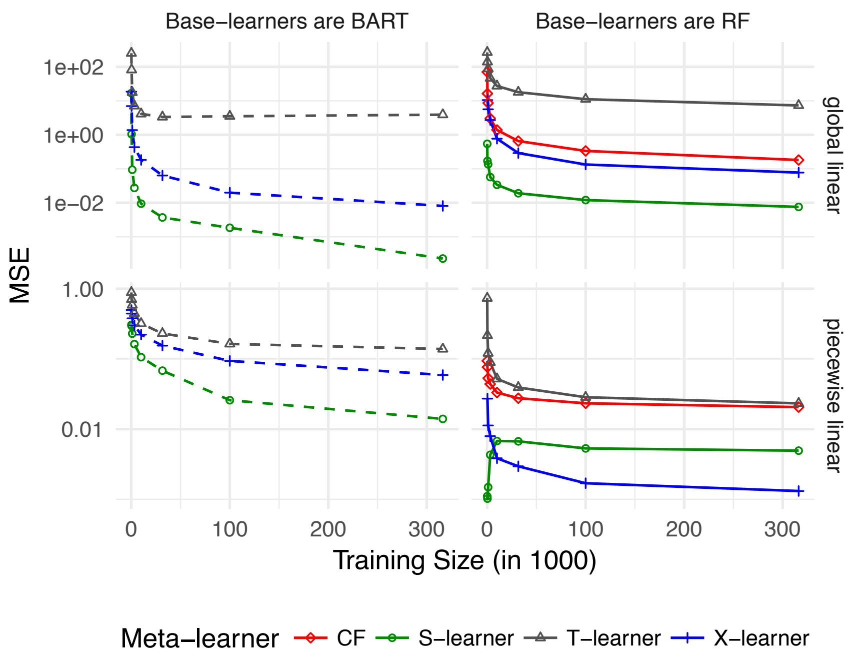


Figure A.4: Comparison of S-, T-, and X-learners with BART (left) and RF (right) as base learners for Simulation 4 (top) and Simulation 5 (bottom).

with

$$\beta_l(i) = \begin{cases} \beta(i) & \textit{if} \;\; i \leq 5 \\ 0 & \textit{otherwise} \end{cases} \quad \beta_m(i) = \begin{cases} \beta(i) & \textit{if} \;\; 6 \leq i \leq 10 \\ 0 & \textit{otherwise} \end{cases} \quad \beta_u(i) = \begin{cases} \beta(i) & \textit{if} \;\; 11 \leq i \leq 15 \\ 0 & \textit{otherwise} \end{cases}$$

and

$$\beta \sim \textit{Unif}([-15, 15]^d).$$

Figure A.4 shows the outcome of these simulations. For both simulations, the CATE is globally 0. As expected, the S-learner performs very well, since the treatment assignment has no predictive power for the combined response surface. The S-learner thus often ignores the variable encoding the treatment assignment, and the S-learner correctly predicts a zero treatment effect. We can again see that the global property of the BART harms its performance in the piecewise linear case since here the importance of the features is different in different parts of the space.

Confounding

In the preceding examples, the propensity score was globally equal to some constant. This is a special case, and in many observational studies, we cannot assume this to be true. All of

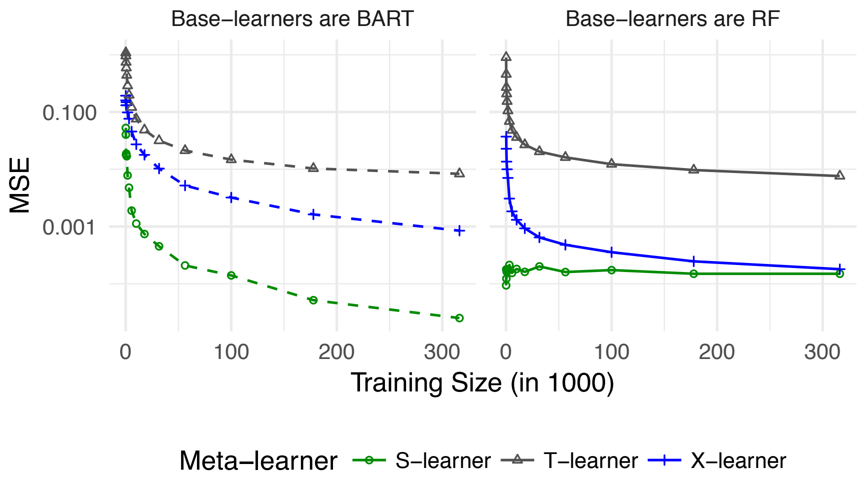


Figure A.5: Comparison of S–, T–, and X–BART (left) and S–, T–, and X–RF (right) for Simulation 6.

the meta-learners we discuss can handle confounding, as long as the ignorability assumption holds. We test this in a setting that has also been studied in [80]. For this setting we choose $x \sim Unif([0,1]^{n\times 20})$ and we use the notation that $\beta(x_1, 2, 4)$ is the β distribution with parameters 2 and 4.

Simulation 6 (beta confounded)

$$\begin{aligned} e(x) &= \frac{1}{4}(1+\beta(x_1,2,4)), \\ \mu_0(x) &= 2x_1 - 1, \\ \mu_1(x) &= \mu_0(x). \end{aligned}$$

Figure A.5 shows that none of the algorithms performs significantly worse under confounding. We do not show the performance of causal forests, because—as noted by the authors—it is not designed for observational studies with only conditional unconfoundedness and it would not be fair to compare it here [80].

A.2 Notes on the ITE

We provide an example that demonstrates that the ITE is not identifiable without further assumptions. Similar arguments and examples have been given before [33], and we list it here only for completeness.

Example 3 (D_i is not identifiable) *Assume that we observe a one-dimensional and uniformly distributed feature between 0 and 1, $X \sim Unif([0,1])$, a treatment assignment that is* *independent of the feature and Bernoulli distributed, $W \sim Bern(0.5)$, and a Rademacher-distributed outcome under control that is independent of the features and the treatment assignment,*

$$P(Y(0) = 1) = P(Y(0) = -1) = 0.5.$$

Now consider two Data-Generating Processes (DGP) identified by the distribution of the outcomes under treatment:

1. *In the first DGP, the outcome under treatment is equal to the outcome under control:*
$$Y(1) = Y(0).$$

2. *In the second DGP, the outcome under treatment is the negative of the outcome under control:*
$$Y(1) = -Y(0).$$

Note that the observed data, $\mathcal{D} = (Y_j, X_j, W_j)_{1 \le j \le N}$, has the same distribution for both DGPs, but $D_i = 0$ for all i in DGP 1, and $D_i \in \{-2, 2\}$ for all i in DGP 2. Thus, no estimator based on the observed data $\mathcal{D}$ can be consistent for the ITEs, $(D_i)_{1 \le i \le n}$. The CATE, $\tau(X_i)$, is, however, equal to 0 in both DGPs. $\hat{\tau} \equiv 0$, for example, is a consistent estimator for the CATE.

A.3 Confidence Intervals for the Social Pressure Analysis

In this paper, we study general meta-learners without making any parametric assumptions on the CATE. This generality makes it very difficult to provide confidence intervals with formal guarantees. In the GOTV section of the main paper, we used bootstrap confidence intervals; in this section, we explain why we choose the bootstrap and details of the variant of the bootstrap, we selected.

The bootstrap has been proven to perform well in many situations [52] and it is straightforward to apply to any estimator that can be written as a function of iid data. There are, however, many ways to obtain bootstrap confidence intervals. We have decided to use Algorithm 10, because it performed well for X–RF in the Atlantic Causal Inference Conference (ACIC) challenge [19], where one of the goals was to create confidence intervals for a wide variety of CATE estimation problems. We refer to these confidence intervals as normal approximated CIs.

It was seen in the ACIC challenge that constructing confidence intervals for the CATE that achieve their nominal coverage is extremely difficult, and no method always provides the correct coverage. To argue that the conclusions we draw in this paper are not specific to a single bootstrap method, we implement another version of the bootstrap to estimate

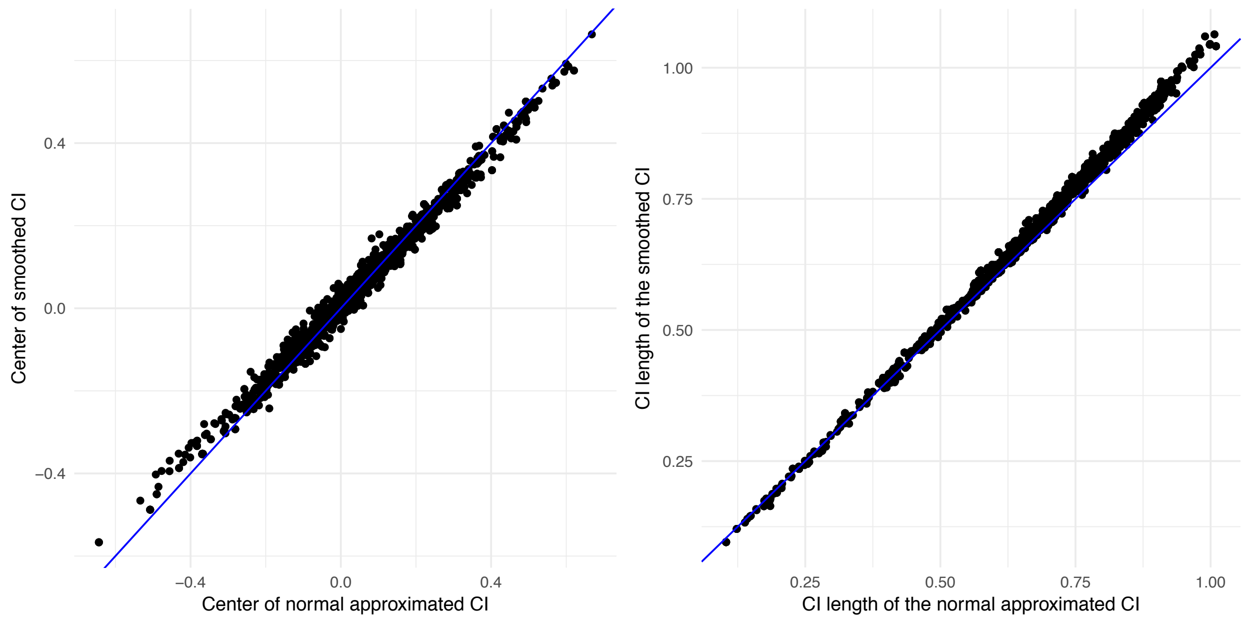


Figure A.6: Comparison of normal approximated CI (Algorithm 10) and smoothed CI (Algorithm 11). The blue line is the identity function.

confidence intervals due to [60] and [20]. We refer to it as the smoothed bootstrap, and we call the corresponding confidence intervals smoothed CIs. Pseudocode for this method can be found in Algorithm 11.

There are many other versions of the bootstrap that could have been chosen, but we focus on two that performed well in the ACIC challenge. To compare these methods, we use the GOTV data, and we estimate confidence intervals for $2,000$ test points based on $50,000$ training points. We have to use this much smaller subset of the data for computational reasons.

For both methods, we use $B = 10,000$ bootstrap samples. This is a large number of replications, but it is necessary because the smoothed CIs (Algorithm 11) are unstable for a smaller B. Figure A.6 compares the center and the length of the confidence intervals of the two methods for T–RF. We can see that the two methods lead to almost the same confidence intervals. The normal approximated CIs are slightly larger, but the difference is not substantial. This is not surprising given the size of the data, and it confirms that our analysis of the GOTV data would have come to the same conclusion had we used smoothed CIs (Algorithm 11). However, normal approximated CIs (Algorithm 10) are computationally much less expensive and they are therefore our default method.

CI-Simulation 1: Comparison of the coverage of the CI estimation methods

To analyze the coverage of the different bootstrap methods, we use a simulation study informed by the GOTV data. We generate the data in the following way:

CI-Simulation 1

1. We start by training the T-learner with random forests on the entire GOTV data set to receive CATE estimates. We take this estimate as the ground truth and call it $\tau(x)$.
2. We then compute for each unit i the missing potential outcome. That is, for a unit in the control group, we add $\tau(x_i)$ to the observed outcome to obtain the outcome under treatment, and for each unit in the treatment group, we subtract $\tau(x_i)$ from the observed outcome to obtain the outcome under control.
3. Next, we create a new treatment assignment by permuting the original one. This also determines our new observed outcome.
4. Finally, we sample uniformly and without replacement a test set of $2,000$ observations and a training set of $50,000$ observations.

We then compute 95% confidence intervals for each point in the test set using the the normal and smoothed bootstrap combined with the S, T, and X-learner. The left part of Figure A.7 shows a comparison of the six methods. We find that none of the methods provide the correct coverage. The coverage of the smooth bootstrap intervals is slightly higher than the coverage of the normal approximated confidence intervals, but the difference is within 1%. It also appears that the T-learner provides the best coverage, but it also has the largest confidence interval length.

Based on this simulation, we believe that the smooth CIs have a slightly higher coverage but the intervals are also slightly longer. However, the smooth CIs are computationally much more expensive and need a lot of bootstrap samples to be stable. They are therefore unfeasible for our data. Hence we prefer the normal approximated CIs.

In general, we observe that none of the methods achieve the anticipated 95% coverage and we suspect that this is the case, because the CATE estimators are biased and the bootstrap is not adjusting for the bias terms. To analyze this, we approximated the bias using a Monte Carlos simulation for each of the 2,000 test points using Algorithm 12. The density plot in Figure A.8 shows that the bias of X–RF in our sample is substantial and in particular of the same order as the size of the confidence intervals of X–RF. For example, more than 11% of all units had bias bigger than 0.15.

This raises the question whether it is possible to correct for the bias. We tried to use the bootstrap again to estimate the bias. Specifically, we used Algorithm 13 to estimate it. The upper subfigure in Figure A.8 is a scatter plot of the Monte-Carlo-approximated bias versus

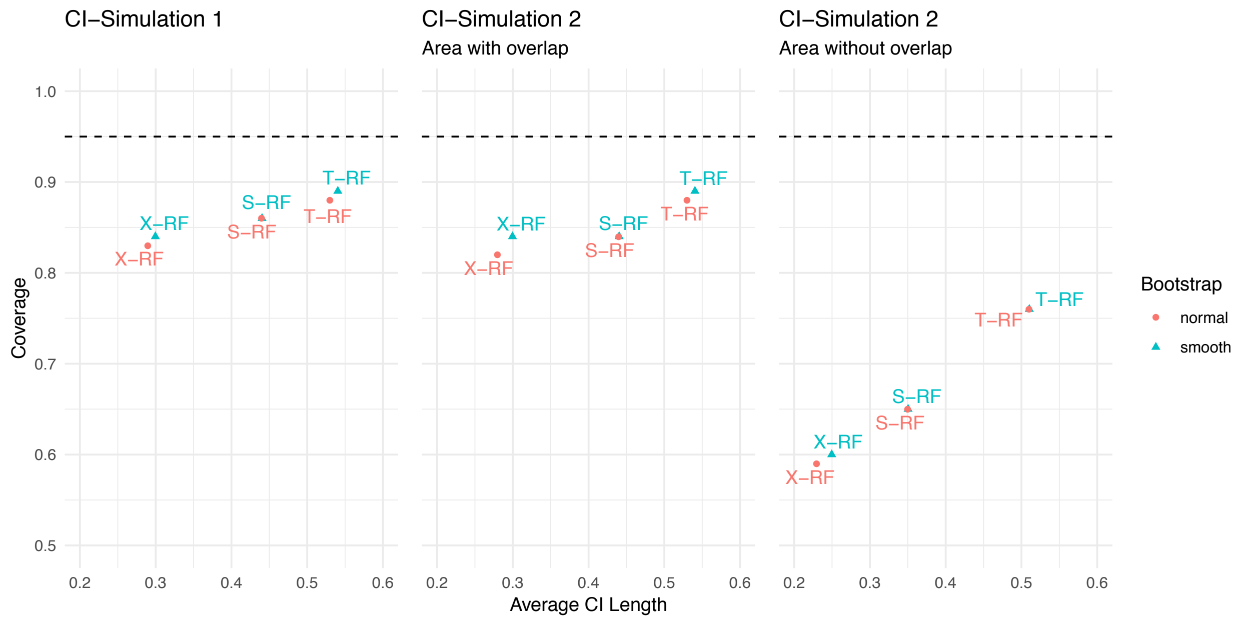


Figure A.7: Coverage and average confidence interval length of the three meta-learners for normal approximated CI (Algorithm 10) and smoothed CI (Algorithm 11). The left figure corresponds to Simulation 3.1; the middle figure corresponds to units in an area with overlap in Simulation SI 3.2, and the right figure corresponds to units in an area without overlap in Simulation SI 3.2. The dotted line corresponds to the target 95% confidence interval.

the bootstrap-estimated bias. We can see that the bootstrap does not correctly estimate the bias.

CI-Simulation 2: Confounding without overlap

In observational studies, researchers have no control over the treatment assignment process and, in some cases, even the overlap condition may be violated. That is, there exists a subgroup of units that is identifiable by observed features for which the propensity score is 0 or 1. Consequently, all units are either treated or not and estimating the CATE is impossible without very strong assumptions. We generally advise researchers to be very cautious when using these methods on observational data. In this section, we want to study how well one can estimate confidence intervals in observational studies where the overlap condition is violated. Ideally, we would hope that the confidence intervals in areas with no overlap are extremely wide.

To test the behavior of the different confidence interval estimation methods, we set up another simulation based on real data. In this simulation we intentionally violate the overlap condition by assigning all units between 30 and 40 years to the control group. We then compared the confidence intervals for this subgroup with the other units where the overlap

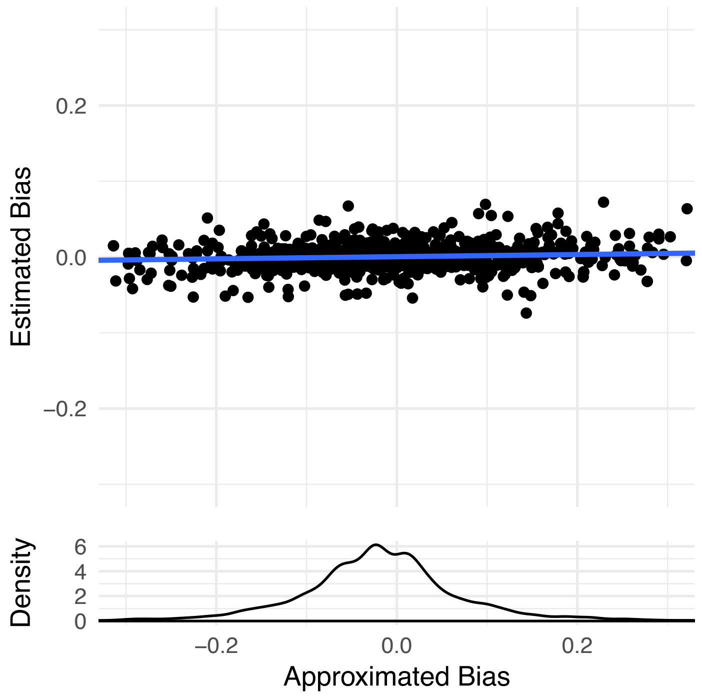


Figure A.8: Approximated bias using Algorithm 12 versus estimated bias using Algorithm 13 and X–RF.

condition is not violated. For our simulation, we follow the same steps as in Section A.3, but we modified Step 3 to ensure that all units between 30 and 40 years of age are in the control group. Specifically, we construct the data in the following way:

1. We start by training the T-learner with random forests on the entire GOTV data set to construct CATE estimates. We take this estimate as the ground truth and call it $\tau(x)$.

2. We then use $\tau(x)$ to impute the missing potential outcomes. That is, for a unit in the control group, we add $\tau(x_i)$ to the observed outcome to obtain the outcome under treatment, and for each unit in the treatment group, we subtract $\tau(x_i)$ from the observed outcome to obtain the outcome under control.

3. Next, we create a new treatment assignment by permuting the original treatment assignment vector and assigning all entries for units between 30 and 40 years old to the control group. This also determines our new observed outcome.

4. Finally, we sample uniformly and without replacement two test sets and one training set. We first sample the training set of $50,000$ observations. Next, we sample the first test set of $20,000$ units out of all units that are not in the 30 to 40-year-old age group. This test set is called the **overlap test set**. Finally, we sample the second test set of

$20,000$ units out of all units in the 30 to 40-year-old age group and we call this test set the **non-overlap test set**.

Note that by construction the overlap condition is violated for the subgroup of units between 30 and 40 years and satisfied for units outside of that age group.

We trained each method on the training set and estimated the confidence intervals for the CATE in both test sets. The middle and the right part of Figure A.7 shows the results for the overlap test set and the non-overlap test set, respectively. We find that the coverage and the average confidence interval length for the overlap test set is very similar to that of the previous simulation study, CI-Simulation 1. This is not surprising, because the two setups are very similar and the overlap condition is satisfied in both.

The coverage and the average length of the confidence intervals for the non-overlap test set are, however, very different. For this subgroup, we do not have overlap. We should be cautious when estimating the CATE or confidence intervals of the CATE when there is no overlap, and we hope to see this reflected in very wide confidence intervals. Unfortunately, this is not the case. We observe that for all methods the confidence intervals are tighter and the coverage is much lower than on the data where we have overlap because they try to extrapolate into regions of the covariate space without information on the treatment group. This is a problematic finding and suggests that confidence interval estimation in observational data is extremely difficult and that a violation of the overlap condition can lead to invalid inferences. We believe that this is an artifact of how random forests deal with the predictions for units outside of the support of the training data. We are currently working on an improved version of random forests that better captures this uncertainty.

A.4 Stability of the Social Pressure Analysis across Meta-learners

In Figure 1.2, we present how the CATE varies with the observed covariates. We find a very interesting behavior in the fact that the largest treatment effect can be observed for potential voters who voted three or four times before the 2004 general election. The treatment effect for potential voters who voted in none or all five of the observed elections was much smaller. We concluded this based on the output of the X-learner. To show that a similar conclusion can be drawn using different meta-learners, we repeated our analysis with the S and T learner (cf. Figure A.9). We find that the output is almost identical to the output of the X-learner. This is not surprising since the data set is very large and most of the covariates are discrete.

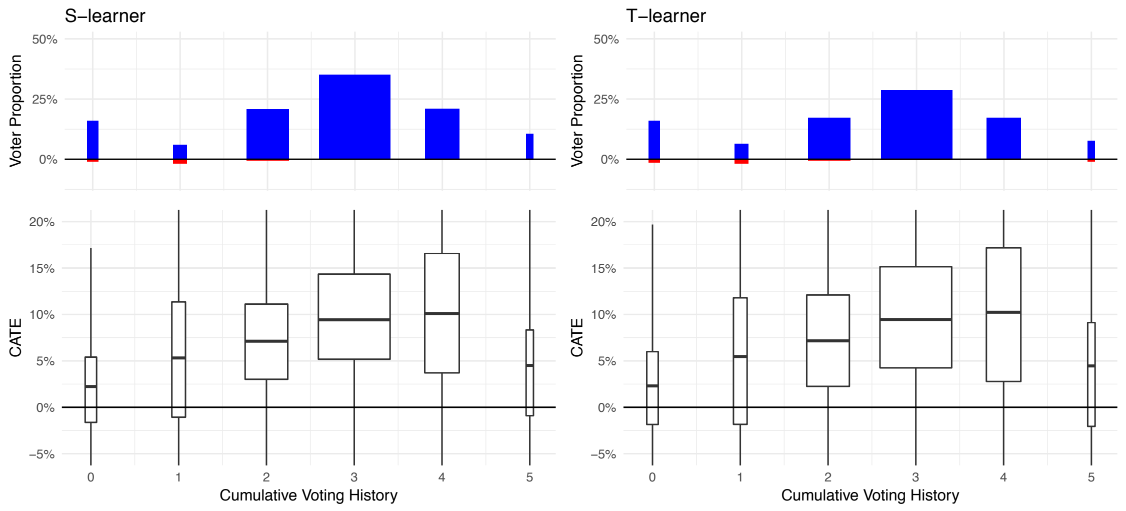


Figure A.9: Results for the S-learner (left) and the T-learner (right) for the get-out-the-vote experiment.

A.5 The Bias of the S-learner in the Reducing Transphobia Study

For many base learners, the S-learner can completely ignore the treatment assignment and thus predict a 0 treatment effect. This often leads to a bias toward 0, as we can see in Figure 1.4. To further analyze this behavior, we trained a random forest estimator on the transphobia data set with 100,000 trees, and we explored how often the individual trees predict a 0 treatment effect by not splitting on the treatment assignment. Figure A.10 shows that the trees very rarely split on the treatment assignment. This is not surprising for this data set since the covariates are very predictive of the control response function and the treatment assignment is a relatively weak predictor.

A.6 Adaptivity to Different Settings and Tuning

Tuning the base learners to receive better CATE estimators or even selecting the best CATE estimator from a finite set of CATE estimators is very difficult, and our recent R package, `hte`, attempts to implement some tuning and selection methods. This is, however, very difficult and in the preceding sections, we did not tune our random forest algorithm or our BART estimators on the given data sets. Instead, we used fixed hyperparameters that were chosen in a different simulation study. In the sequel, we show that tuning the base learners and being able to select the best meta-learner can be very beneficial to constructing a good CATE estimator.

We conduct a simple experiment showing the potential benefits of hyperparameter tuning

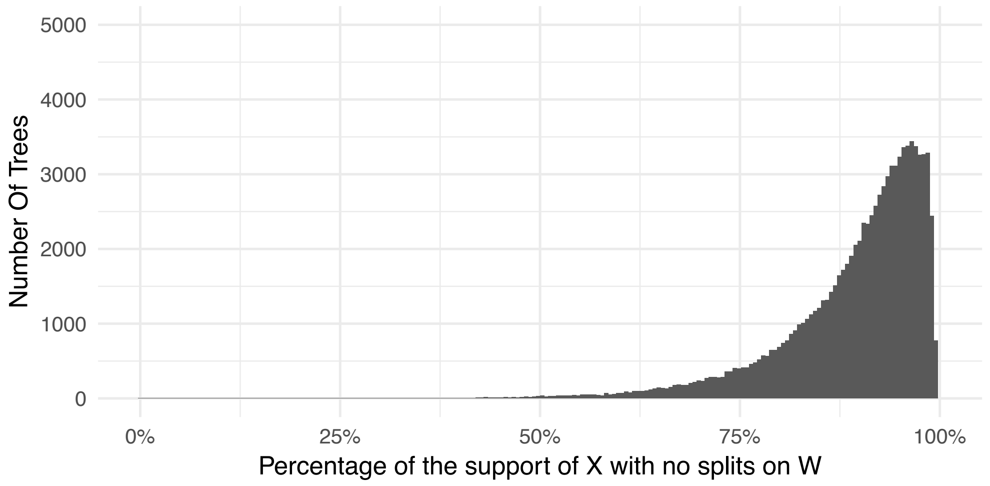


Figure A.10: This figure is created from an S–RF learner to show that the S-learner often ignores the treatment effect entirely. It is based on 100,000 trees and it shows the histogram of trees by what percentage of the support of X is not split on W.

of the base learners. Specifically, we evaluate S–RF, T–RF, and X–RF in Simulations 4 and 2. We sample 1,000 hyperparameter settings for each of the learners and evaluate them in both simulations. In other words, for each hyperparameter setting, we obtain an MSE for Simulation 4 and an MSE for Simulation 2.

Figure A.11 shows the MSE pairs. As expected, we observe that the T-learner generally does very well when the treatment effect is complex, while it does rather poorly when the treatment effect is simple. This was expected as the T-learner generally performs poorly compared to the S-learner when the treatment effect is simple or close to 0. Also as expected, the S-learner performs well when the treatment effect is simple, but it performs relatively poorly compared to the T-learner when the treatment effect is complex. The X-learner, on the other hand, is extremely adaptive. In fact, depending on the set of hyperparameters, the X-learner can perform as well as the T-learner or the S-learner. However, there is not a single set of parameters that is optimal for both settings. In fact, the optimal settings almost describe a utility curve.

Setting the Tuning Parameters

Since tuning each algorithm for each data set separately turns out to be very challenging, we decided to hold the hyperparameters fix for each algorithm. To chose those preset hyperparameters, we used the 2016 Atlantic Causal Inference Conference competition [19], and we chose the parameters in such a way that the algorithms perform very well in this competition. Specifically, we randomly generated for each algorithm 10,000 hyperparameters. We then evaluated the performance of these 10,000 hyperparameter settings on the 20 data sets

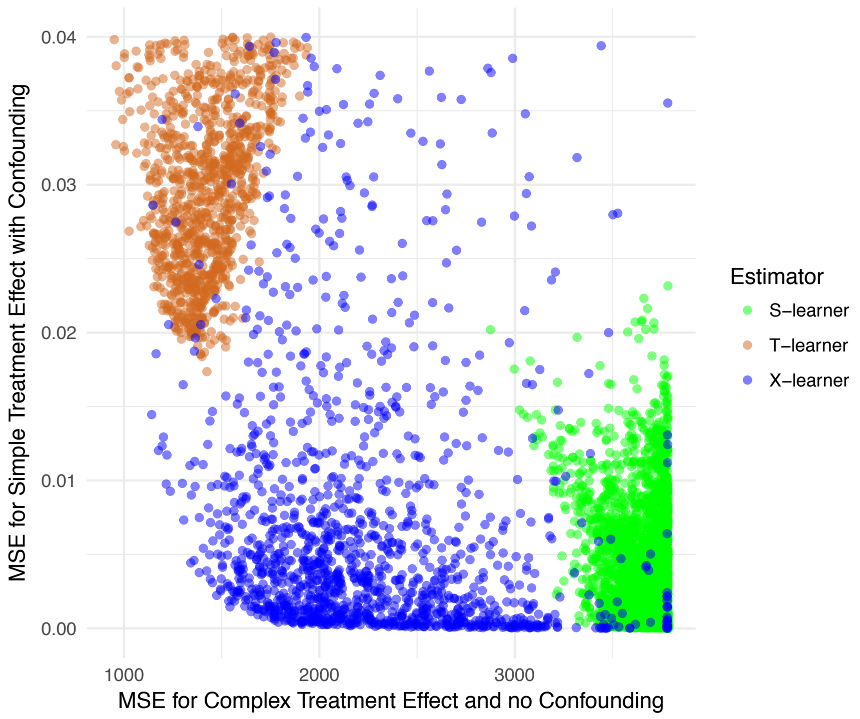


Figure A.11: Each point corresponds to a different hyperparameter setting in random forests as the base learner in one of the S-, T-, or X-learners. The y-axis value is the MSE of Simulation 4 and the x-axis value is the MSE in Simulation 2. A perfect estimator that gets an MSE error of 0 in both simulations would thus correspond to a point at the origin (0,0). The training set size had 1,000 units and the test set that was used to estimate the MSE had 10,000 units.

of the "Do it yourself!"-challenge, and we chose the hyperparameter combination which did best for that challenge.

A.7 Conditioning on the Number of Treated Units

In our theoretical analysis, we assume a superpopulation and we condition on the number of treated units both to avoid the problem that with a small but non-zero probability all units are in the treatment group or the control group and to be able to state the performance of different estimators in terms of n, the number of treated units, and m, the number of control units. This conditioning, however, leads to nonindependent samples. The crucial step in dealing with this dependent structure is to condition on the treatment assignment, W.

Specifically, there are three models to be considered.

1. The first one is defined by 1.1. It specifies a distribution, $\mathcal{P}$, of (X, W, Y), and we

assume to observe N independent samples from this distribution,

$$(X_i, W_i, Y_i)_{i=1}^N \overset{iid}{\sim} \mathcal{P}.$$

We denote the joint distribution of $(X_i, W_i, Y_i)_{i=1}^N$ by $\mathcal{P}^N$.

2. We state our technical results in terms of a conditional distribution. For a fixed n with $0 < n < N$, we consider the distribution of $(X_i, W_i, Y_i)_{i=1}^N$ given that we observe n treated units and $m = N - n$ control units. We denote this distribution by $\mathcal{P}^{nm}$.

$$\left[(X_i, W_i, Y_i)_{i=1}^N \middle| \sum_{i=1}^N W_i = n\right] \sim \mathcal{P}^{nm}.$$

Note that under $\mathcal{P}^{nm}$ the (X_i, W_i, Y_i) are identical in distribution, but not independent.

3. For technical reasons, we also introduce a third distribution, which we will use only in some of the proofs. Here, we condition on the vector of treatment assignments, W.

$$\left[(X_i, W_i, Y_i)_{i=1}^N \middle| W = w\right] \sim \mathcal{P}^{w}.$$

Under this distribution W is non-random and (X_i, Y_i) are not identical in distribution. However, within each treatment group the (X_i, Y_i) tuples are independent and identical in distribution. To make this more precise, define $\mathcal{P}_1$ to be the conditional distribution of (X, Y) given $W = 1$; then, under $\mathcal{P}^w$, we have

$$(X_i, Y_i)_{W_i=1} \overset{iid}{\sim} \mathcal{P}_1.$$

We prove these facts as follows.

Theorem 3 *Let n and N be such that $0 < n < N$ and let $w \in \{0,1\}^N$ with $\sum_{i=1}^N w_i = n$. Then, under the distribution $\mathcal{P}^w$,*

$$(X_k, Y_k)_{W_k=1} \overset{iid}{\sim} \mathcal{P}_1.$$

We prove this in two steps. In Lemma 1, we prove that the distributions are independent and in Lemma 2 we prove that they are identical.

Lemma 1 (independence) *Let n, N, and w be as in Theorem 3 and define $S = \{j \in \mathbb{N} : w_j = 1\}$. Then for all $\emptyset \neq \mathcal{I} \subset S$, and all $(B_i)_{i \in \mathcal{I}}$ with $B_i \subset \mathbb{R}^p \times \mathbb{R}$,*

$$\mathbb{P}\left(\bigcap_{i \in \mathcal{I}} \{(X_i, Y_i) \in B_i\} \middle| W = w\right) = \prod_{i \in \mathcal{I}} \mathbb{P}\left((X_i, Y_i) \in B_i \middle| W = w\right). \tag{A.2}$$

Note that another way of writing A.2 is

$$\mathbb{P}^w\left(\bigcap_{i\in\mathcal{I}}\{(X_i,Y_i)\in B_i\}\right)=\prod_{i\in\mathcal{I}}\mathbb{P}^w\left((X_i,Y_i)\in B_i\right). \tag{A.3}$$

Proof. [Proof of Lemma 1]

$$\begin{aligned}
&\mathbb{P}\left(\bigcap_{i\in\mathcal{I}}\{(X_i,Y_i)\in B_i\}\middle| W=w\right)\\
&=\mathbb{P}\left(\left(\bigcap_{i\in\mathcal{I}}\{(X_i,Y_i)\in B_i\}\right)\;\cap\;\left(\bigcap_{j\in S}\{W_j=1\}\cap\bigcap_{k\in S^c}\{W_k=0\}\right)\right)\Big/\;\mathbb{P}\Big(W=w\Big)\\
&=\mathbb{P}\left(\left(\bigcap_{i\in\mathcal{I}}\big\{(X_i,Y_i,W_i)\in B_i\times\{1\}\big\}\right)\;\cap\;\left(\bigcap_{j\in S\setminus\mathcal{I}}\{W_j=1\}\cap\bigcap_{k\in S^c}\{W_k=0\}\right)\right)\Big/\mathbb{P}\Big(W=w\Big)\\
&=\prod_{i\in\mathcal{I}}\mathbb{P}\Big((X_i,Y_i,W_i)\in B_i\times\{1\}\Big)\frac{\mathbb{P}\Big(\bigcap_{j\in S\setminus\mathcal{I}}\{W_j=1\}\cap\bigcap_{k\in S^c}\{W_k=0\}\Big)}{\mathbb{P}\Big(W=w\Big)}=(*).
\end{aligned}$$

The last equality holds because $(X_i,Y_i,W_i)_{i=1}^N$ are mutually independent. The second term can be rewritten in the following way:

$$\begin{aligned}
\frac{\mathbb{P}\Big(\bigcap_{j\in S\setminus\mathcal{I}}\{W_j=1\}\cap\bigcap_{k\in S^c}\{W_k=0\}\Big)}{\mathbb{P}\Big(W=w\Big)}&=\frac{\prod_{j\in S\setminus\mathcal{I}}\mathbb{P}(W_j=1)\prod_{k\in S^c}\mathbb{P}(W_k=0)}{\prod_{j\in S}\mathbb{P}(W_j=1)\prod_{k\in S^c}\mathbb{P}(W_k=0)}\\
&=\prod_{j\in J}\frac{1}{\mathbb{P}(W_j=1)}\\
&=\prod_{j\in J}\frac{\prod_{j\in S\setminus\{j\}}\mathbb{P}(W_j=1)\prod_{k\in S^c}\mathbb{P}(W_k=0)}{\prod_{j\in S}\mathbb{P}(W_j=1)\prod_{k\in S^c}\mathbb{P}(W_k=0)}\\
&=\prod_{i\in\mathcal{I}}\frac{\mathbb{P}\Big[\bigcap_{j\in S\setminus\{i\}}\{W_j=1\}\cap\bigcap_{k\in S^c}\{W_k=0\}\Big]}{\mathbb{P}\Big[W=w\Big]}.
\end{aligned}$$

Thus,

$$
\begin{aligned}
(*) &= \prod_{i\in\mathcal{I}} \mathbb{P}\Big[(X_i,Y_i,W_i)\in B_i\times\{1\}\Big] \prod_{i\in\mathcal{I}} \frac{\mathbb{P}\Big[\bigcap_{j\in S\setminus\{i\}}\{W_j=1\}\cap\bigcap_{k\in S^c}\{W_k=0\}\Big]}{\mathbb{P}\Big[W=w\Big]}\\
&= \prod_{i\in\mathcal{I}} \left(\mathbb{P}\Big[(X_i,Y_i,W_i)\in B_i\times\{1\}\ \cap\ \Big(\bigcap_{j\in S\setminus\{i\}}\{W_j=1\}\cap\bigcap_{k\in S^c}\{W_k=0\}\Big)\Big] \Big/ \ \mathbb{P}\Big[W=w\Big]\right)\\
&= \prod_{i\in\mathcal{I}} \left(\mathbb{P}\left((X_i,Y_i)\in B_i\ \cap\ \Big\{W=w\Big\}\right) \Big/ \ \mathbb{P}\Big(W=w\Big)\right)\\
&\prod_{i\in\mathcal{I}} \mathbb{P}\left((X_i,Y_i)\in B_i\Big|W=w\right),
\end{aligned}
$$

which completes the proof.

Next, we are concerned with showing that all treated units have the same distribution.

Lemma 2 (identical distribution) *Assume the same assumptions as in Lemma 1 and let $i \neq j \in S$. Under the conditional distribution of $W = w$, (X_i, Y_i) and (X_j, Y_j) have the same distribution, $\mathcal{P}_1$.*

Proof. Let $B \subset \mathbb{R}^p \times \mathbb{R}$; then

$$
\begin{aligned}
\mathbb{P}\left((X_i,Y_i)\in B\Big|W=w\right) &\overset{*}{=} \mathbb{P}\left((X_i,Y_i)\in B\big|W_i=1\right)\\
&= \frac{\mathbb{P}\left((X_i,Y_i,W_i)\in B\times\{1\}\right)}{\mathbb{P}(W_i=1)}\\
&\overset{a}{=} \frac{\mathbb{P}\left((X_j,Y_j,W_j)\in B\times\{1\}\right)}{\mathbb{P}(W_j=1)}\\
&= \mathbb{P}\left((X_j,Y_j)\in B\big|W_j=1\right)\\
&\overset{*}{=} \mathbb{P}\left((X_j,Y_j)\in B\Big|W=w\right).
\end{aligned}
$$

Here * follows from $(X_i, Y_i, W_i)_{i=1}^N$ being mutually independent, and ^{a} follows from $(X_i, Y_i, W_i)_{i=1}^N$ being identically distributed under $\mathcal{P}$.

A.8 Convergence Rate Results for the T-learner

In this section, we want to prove Theorem 1 of the main paper. We start with a short lemma that will be useful for the proof of the theorem.

Lemma 3 *Let $\mathcal{P}$ be defined as in 1.1 with $0 < e_{\min} < e(x) < e_{\min} < 1$. Furthermore, let X, W be distributed according to $\mathcal{P}$, and let g be a positive function such that the expectations below exist; then*

$$\frac{e_{\min}}{e_{\max}}\mathbb{E}[g(X)] \ \leq \mathbb{E}[g(X)|W=1] \ \leq \ \frac{e_{\max}}{e_{\min}}\mathbb{E}[g(X)], \tag{A.4}$$

$$\frac{1-e_{\max}}{1-e_{\min}}\mathbb{E}[g(X)] \ \leq \mathbb{E}[g(X)|W=0] \ \leq \ \frac{1-e_{\min}}{1-e_{\max}}\mathbb{E}[g(X)]. \tag{A.5}$$

Proof. [Proof of Lemma 3] Let us prove A.4 first. The lower bound follows from

$$\mathbb{E}[g(X)|W=1] \geq \mathbb{E}[g(X)]\frac{\inf_x e(x)}{E[W]} \geq \frac{e_{\min}}{E[W]}\mathbb{E}[g(X)] \geq \frac{e_{\min}}{e_{\max}}\mathbb{E}[g(X)],$$

and the upper bound from

$$\mathbb{E}[g(X)|W=1] \leq \mathbb{E}[g(X)]\frac{\sup_x e(x)}{E[W]} \leq \frac{e_{\max}}{e_{\min}}\mathbb{E}[g(X)].$$

A.5 follows from a symmetrical argument.

Let us now restate Theorem 1. Let $m, n \in \mathbb{N}^+$ and $N = m + n$ and let $\mathcal{P}$ be a distribution of (X, W, Y) according to 1.1 with the propensity score bounded away from 0 and 1. That is, there exists $e_{\min}$ and $e_{\max}$ such that $0 < e_{\min} < e(x) < e_{\max} < 1$. Furthermore, let $(X_i, W_i, Y_i)_{i=1}^N$ be i.i.d. from $\mathcal{P}$ and define $\mathcal{P}^{nm}$ to be the conditional distribution of $(X_i, W_i, Y_i)_{i=1}^N$ given that we observe n treated units, $\sum_{i=1}^N W_i = n$.

Note that n and m are not random under $\mathcal{P}^{nm}$. We are interested in the performance of the T-learner, $\hat{\tau}_T^{mn}$, under $\mathcal{P}^{nm}$ as measured by the EMSE,

$$\text{EMSE}(\hat{\tau}_T^{mn}, \mathcal{P}^{nm}) \overset{\text{def}}{=} \mathbb{E}\left[(\hat{\tau}_T^{mn}(\mathcal{X}) - \tau(\mathcal{X}))^2 \middle| \sum_{i=1}^N W_i = n\right].$$

The expectation is here taken over the training data set $(X_i, W_i, Y_i)_{i=1}^N$, which is distributed according to $\mathcal{P}^{nm}$, and $\mathcal{X}$, which is distributed according to the marginal distribution of X in $\mathcal{P}$.

For a family of superpopulations, $F \in S(a_\mu, a_\tau)$, we want to show that the T-learner with an optimal choice of base learners achieves a rate of

$$\mathcal{O}(m^{-a_\mu} + n^{-a_\mu}).$$

An optimal choice of base learners is estimators that achieve the minimax rate of n^{-a_μ} and m^{-a_μ} in F.

Proof. [Proof of Theorem 1] The EMSE can be upper bounded by the errors of the single base learners:

$$\begin{aligned}\mathrm{EMSE}(\hat{\tau}_T^{mn}, \mathcal{P}^{nm}) &= \mathbb{E}\left[(\hat{\tau}_T^{mn}(\mathcal{X}) - \tau(\mathcal{X}))^2 \middle| \sum_{i=1}^N W_i = n\right] \\ &\leq \underbrace{2\,\mathbb{E}\left[(\hat{\mu}_1^n(\mathcal{X}) - \mu_1(\mathcal{X}))^2 \middle| \sum_{i=1}^N W_i = n\right]}_{A} + \underbrace{2\,\mathbb{E}\left[(\hat{\mu}_0^m(\mathcal{X}) - \mu_0(\mathcal{X}))^2 \middle| \sum_{i=1}^N W_i = n\right]}_{B}.\end{aligned}$$

Here we use the following inequality:

$$(\hat{\tau}_T^{mn}(\mathcal{X}) - \tau(\mathcal{X}))^2 \leq 2(\hat{\mu}_1^n(\mathcal{X}) - \mu_1(\mathcal{X}))^2 + 2(\hat{\mu}_0^m(\mathcal{X}) - \mu_0(\mathcal{X}))^2.$$

Let us look only at the first term. We can write

$$\begin{aligned}A &= \mathbb{E}\left[(\hat{\mu}_1^n(\mathcal{X}) - \mu_1(\mathcal{X}))^2 \middle| \sum_{i=1}^N W_i = n\right] \\ &= \mathbb{E}\left[\mathbb{E}\left[(\hat{\mu}_1^n(\mathcal{X}) - \mu_1(\mathcal{X}))^2 \middle| W, \sum_{i=1}^N W_i = n\right] \middle| \sum_{i=1}^N W_i = n\right]. \qquad (A.6)\end{aligned}$$

It is of course not necessary to condition on $\sum_{i=1}^N W_i = n$ in the inner expectation, and we only do so as a reminder that there are n treated units.

For $i \in \{1, \ldots, n\}$, let q_i be the i^{th} smallest number in $\{k : W_k = 1\}$. That is, $\{q_i : i \in \{1, \ldots, n\}\}$ are the indexes of the treated units. To emphasize that $\hat{\mu}_1^n(\mathcal{X})$ depends only on the treated observations, $(X_{q_i}, Y_{q_i})_{i=1}^n$, we write $\hat{\mu}_1^n((X_{q_i}, Y_{q_i})_{i=1}^n, \mathcal{X})$. Furthermore, we define $\mathcal{P}_1$ to be the conditional distribution of (X, Y) given $W = 1$. Conditioning on W, Theorem 3 implies that $(X_{q_i}, Y_{q_i})_{i=1}^n$ is i.i.d. from $\mathcal{P}_1$. Let us define $\tilde{\mathcal{X}}$ to be distributed according to $\mathcal{P}_1$. Then we can apply Lemma 3 and use the definition of $S(a_\mu, a_\tau)$ to conclude that the inner expectation in A.6 is in $\mathcal{O}(n^{-a_\mu})$:

$$\begin{aligned}&\mathbb{E}\left[\hat{\mu}_1^n((X_{q_i}, Y_{q_i})_{i=1}^n, \mathcal{X}) - \mu_1(\mathcal{X}))^2 \middle| W, \sum_{i=1}^N W_i = n\right] \\ &\leq \frac{e_{\max}}{e_{\min}} \mathbb{E}\left[(\hat{\mu}_1^n((X_{q_i}, Y_{q_i})_{i=1}^n, \tilde{\mathcal{X}}) - \mu_1(\tilde{\mathcal{X}}))^2 \middle| W, \sum_{i=1}^n W_i = n\right] \\ &\leq \frac{e_{\max}}{e_{\min}} C n^{-a_\mu}.\end{aligned}$$

Hence, it follows that

$$A \leq 2\mathbb{E}\left[\frac{e_{\max}}{e_{\min}} C n^{-a_\mu} \middle| \sum_{i=1}^n W_i = n\right] \leq 2\frac{e_{\max}}{e_{\min}} C n^{-a_\mu}.$$

By a symmetrical argument, it also holds that

$$B \leq 2\frac{1-e_{\min}}{1-e_{\max}} C m^{-a_\mu},$$

and we can conclude that

$$\mathrm{EMSE}(\hat{\tau}_T^{mn}, \mathcal{P}) \leq 2C\left[\frac{1-e_{\min}}{1-e_{\max}} + \frac{e_{\max}}{e_{\min}}\right](n^{-a_\mu} + m^{-a_\mu}).$$

A.9 Convergence Rate Results for the X-learner

In this section, we are concerned with the convergence rate of the X-learner. Given our motivation of the X-learner in the main paper, we believe that $\hat{\tau}_0$ of the X-learner should achieve a rate of $\mathcal{O}(m^{-a_\tau} + n^{-a_\mu})$ and $\hat{\tau}_1$ should achieve a rate of $\mathcal{O}(m^{-a_\mu} + n^{-a_\tau})$. In what follows, we prove this for two cases, and we show that for those cases the rate is optimal. In the first case, we assume that the CATE is linear and thus $a_\tau = 1$. We don't assume any regularity conditions on the response functions, and we show that the X-learner with an OLS estimator in the second stage and an appropriate estimator in the first stage achieves the optimal convergence rate. We show this first for the MSE (Theorem 4) and then for the EMSE (Theorem 2). We then focus on the case where we don't impose any additional regularity conditions on the CATE, but the response functions are Lipschitz continuous (Theorem 7). The optimal convergence rate is here not obvious, and we will first prove a minimax lower bound for the EMSE, and we will then show that the X-learner with the KNN estimates achieves this optimal performance.

MSE and EMSE convergence rate for the linear CATE

Theorem 4 (rate for the pointwise MSE) *Assume that we observe m control units and n treated units from some superpopulation of independent and identically distributed observations $(Y(0), Y(1), X, W)$ coming from a distribution $\mathcal{P}$ given in equation [1.1] and assume that the following assumptions are satisfied:*

B1 Ignorability holds.

B2 The treatment effect is linear, $\tau(x) = x^T\beta$, with $\beta \in \mathbb{R}^d$.

B3 There exists an estimator $\hat{\mu}_0$ such that for all x,

$$\mathbb{E}\left[(\mu_0(x) - \hat{\mu}_0^m(x))^2 \middle| \sum_{i=1}^N W_i = n\right] \leq C^0 m^{-a}.$$

B4 The error terms ε_i are independent given X, with $\mathbb{E}[\varepsilon_i|X = x] = 0$ and $Var[\varepsilon_i|X = x] \leq \sigma^2 < \infty$.

B5 *The eigenvalues of the sample covariance matrix of the features of the treated units are well conditioned, in the sense that there exists an* n_0*, such that*

$$\sup_{n>n_0} \mathbb{E}\left[\gamma_{min}^{-1}(\hat{\Sigma}_n)\middle|\sum_{i=1}^N W_i = n\right] < c_1 \quad \textit{and} \quad \sup_{n>n_0} \mathbb{E}\left[\gamma_{max}(\hat{\Sigma}_n)/\gamma_{min}^2(\hat{\Sigma}_n)\middle|\sum_{i=1}^N W_i = n\right] < c_2, \tag{A.7}$$

where $\hat{\Sigma}_n = \frac{1}{n}(X^1)'X^1$ *and* X^1 *is the matrix consisting of the features of the treated units.*

Then the X-learner with $\hat{\mu}_0$ *in the first stage, OLS in the second stage, and weighting function* $g \equiv 0$ *has the following upper bound: for all* $x \in \mathbb{R}^d$ *and all* $n > n_0$*,*

$$\mathbb{E}\left[(\tau(x) - \hat{\tau}_X(x))^2\middle|\sum_{i=1}^N W_i = n\right] \leq C_x\left(m^{-a} + n^{-1}\right) \tag{A.8}$$

with $C_x = max(c_2C^0, \sigma^2 dc_1)\|x\|^2$*.*

Proof. [Proof of Theorem 4] To simplify the notation, we write X instead of X^1 for the observed features of the treated units. Furthermore, we denote that when $g \equiv 0$ in [1.9] in the main paper, the X-learner is equal to $\hat{\tau}_1$ and we only have to analyze the performance of $\hat{\tau}_1$.

The imputed treatment effects for the treatment group can be written as

$$D_i^1 = Y_i - \hat{\mu}_0(X_i) = X_i\beta + \delta_i + \epsilon_i,$$

with $\delta_i = \mu_0(X_i) - \hat{\mu}_0(X_i)$. In the second stage we estimate β using an OLS estimator,

$$\hat{\beta} = (X'X)^{-1}X'D^1.$$

To simplify the notation, we define the event of observering n treated units as $E_n = \{\sum_{i=1}^N W_i = n\}$. We decompose the MSE of $\hat{\tau}(x)$ into two orthogonal error terms:

$$\mathbb{E}\left[(\tau(x) - \hat{\tau}_X(x))^2\middle|\sum_{i=1}^N W_i = n\right] = \mathbb{E}\left[(x'(\beta - \hat{\beta}))^2\middle|E_n\right] \leq \|x\|^2\mathbb{E}\left[\|(X'X)^{-1}X'\delta\|^2 + \|(X'X)^{-1}X'\varepsilon\|^2\middle|E_n\right]. \tag{A.9}$$

Throughout the proof, we assume that $n > n_0$ such assumption B5 can be used. We will show that the second term decreases at the parametric rate, n^{-1}, while the first term decreases at a rate of m^{-a}:

$$\begin{aligned}\mathbb{E}\left[\|(X'X)^{-1}X'\varepsilon\|^2\middle|E_n\right] &= \mathbb{E}\left[tr\left(X(X'X)^{-1}(X'X)^{-1}X'\mathbb{E}\left[\varepsilon\varepsilon'\middle|X, E_n\right]\right)\middle|E_n\right]\\ &\leq \sigma^2 d\mathbb{E}\left[\gamma_{\min}^{-1}(\hat{\Sigma}_n)\middle|E_n\right]n^{-1}\\ &\leq \sigma^2 dc_1 n^{-1}.\end{aligned} \tag{A.10}$$

For the last inequality we used assumption B5. Next, we are concerned with bounding the error coming from not perfectly predicting μ_0:

$$\begin{aligned} \mathbb{E}\left[\|(X'X)^{-1}X'\delta\|_2^2\middle|E_n\right] &\le \mathbb{E}\left[\gamma_{\max}(\hat\Sigma_n)/\gamma^2_{\min}(\hat\Sigma_n)\|\delta\|_2^2\middle|E_n\right] n^{-1} \\ &\le \mathbb{E}\left[\gamma_{\max}(\hat\Sigma_n)/\gamma^2_{\min}(\hat\Sigma_n)\middle|E_n\right] C^0 m^{-a} \\ &\le c_2 C^0 m^{-a}. \end{aligned} \tag{A.11}$$

Here we used that $\gamma_{\max}(\hat\Sigma_n^{-2}) = \gamma_{\min}^{-2}(\hat\Sigma_n)$, and $\mathbb{E}\left[\|\delta\|_2^2\middle|X, E_n\right] = \mathbb{E}\left[\sum_{i=1}^n \delta^2(X_i)\middle|X, E_n\right] \le nC^0m^{-a}$. For the last statement, we used assumption B5. This leads to [A.8].

Bounding the EMSE

Proof. [Proof of Theorem 2] This proof is very similar to the proof of Theorem 4. The difference is that here we bound the EMSE instead of the pointwise MSE, and we have a somewhat weaker assumption, because $\hat\mu_0$ only satisfies that its EMSE converges at a rate of a, but not necessarily the MSE at every x. We introduce $\mathcal{X}$ here to be a random variable with the same distribution as the feature distribution such that the EMSE can be written as $\mathbb{E}[(\tau(\mathcal{X}) - \hat\tau_X(\mathcal{X}))^2|E_n]$. Recall that we use the notation that E_n is the event that we observe exactly n treated units and $m = N - n$ control units:

$$E_n = \left\{\sum_{i=1}^N W_i = n\right\}.$$

We start with a similar decomposition as in [A.9]:

$$\begin{aligned} \mathbb{E}\left[(\tau(\mathcal{X}) - \hat\tau_X(\mathcal{X}))^2\middle|E_n\right] &\le \mathbb{E}\left[\|\mathcal{X}\|^2\right]\mathbb{E}\left[\|\beta - \hat\beta\|^2\middle|E_n\right] \\ &= \mathbb{E}\left[\|\mathcal{X}\|^2\right]\mathbb{E}\left[\|(X'X)^{-1}X'\delta\|^2 + \|(X'X)^{-1}X'\varepsilon\|^2\middle|E_n\right]. \end{aligned} \tag{A.12}$$

Following exactly the same steps as in [A.10], we get

$$\mathbb{E}\left[\|(X'X)^{-1}X'\varepsilon\|^2\middle|E_n\right] \le \sigma^2 d C_\Sigma n^{-1}.$$

Bounding $\mathbb{E}\left[\|(X'X)^{-1}X'\delta\|_2^2\middle|E_n\right]$ is now slightly different than in [A.11]:

$$\begin{aligned} \mathbb{E}\left[\|(X'X)^{-1}X'\delta\|_2^2\middle|E_n\right] &\le \mathbb{E}\left[\gamma_{\min}^{-1}(X'X)\|X(X'X)^{-1}X'\delta\|_2^2\middle|E_n\right] \\ &\le \mathbb{E}\left[\gamma_{\min}^{-1}(X'X)\|\delta\|_2^2\middle|E_n\right] \\ &\le \mathbb{E}\left[\gamma_{\min}^{-1}(\Sigma_n)\frac{1}{n}\|\delta\|_2^2\middle|E_n\right] \\ &\le C_\Sigma \mathbb{E}\left[\|\delta_1\|_2^2\middle|E_n\right]. \end{aligned} \tag{A.13}$$

Here the last inequality follows from Condition 6.

We now apply A.4, A.5, and Condition 4 to conclude that

$$\begin{aligned}\mathbb{E}\left[\|\delta_1\|_2^2\middle|E_n\right] &= \mathbb{E}\left[\|\mu_0(X_1)-\hat{\mu}_0(X_1)\|_2^2\middle|E_n, W_1=1\right]\\ &\le \frac{e_{\max}-e_{\max}e_{\min}}{e_{\min}-e_{\max}e_{\min}}\mathbb{E}\left[\|\mu_0(X_1)-\hat{\mu}_0(X_1)\|_2^2\middle|E_n, W_1=0\right]\\ &\le \frac{e_{\max}-e_{\max}e_{\min}}{e_{\min}-e_{\max}e_{\min}}C_0 m^{-a_\mu}.\end{aligned}$$

Lastly, we use the assumption that $\mathbb{E}\left[\|\mathcal{X}\|^2\middle|E_n\right] \le C_{\mathcal{X}}$ and conclude that

$$\mathbb{E}\left[(\tau(\mathcal{X})-\hat{\tau}_X(\mathcal{X}))^2\middle|E_n\right] \le C_{\mathcal{X}}\left(\frac{e_{\max}-e_{\max}e_{\min}}{e_{\min}-e_{\max}e_{\min}}C_\Sigma C_0 m^{-a}+\sigma^2 d C_\Sigma n^{-1}\right). \tag{A.14}$$

Achieving the parametric rate

When there are a lot of control units, such that $m \ge n^{1/a}$, then we have seen that the X-learner achieves the parametric rate. However, in some situations the X-learner also achieves the parametric rate even if the number of control units is of the same order as the number of treated units. To illustrate this, we consider an example in which the conditional average treatment effect and the response functions depend on disjoint and independent subsets of the features.

Specifically, we assume that we observe m control units and n treated units according to Model 1.1. We assume the same setup and the same conditions as in Theorem 2. In particular, we assume that there exists an estimator $\hat{\mu}_0^m$ that depends only on the control observations and estimates the control response function at a rate of at most m^{-a}. In addition to these conditions we also assume the following independence condition.

Condition 7 *There exists subsets, $S, \bar{S} \subset \{1,\dots,d\}$ with $S \cap \bar{S} = \emptyset$, such that*

- *$(X_i)_{i\in S}$ and $(X_i)_{i\in\bar{S}}$ are independent.*
- *For all $i \in S$, $E[X_i|W_i=1]=0$.*
- *There exist a function $\tilde{\mu}_0$, and a vector $\tilde{\beta}$ with $\mu_0(x)=\tilde{\mu}_0(x_{\bar{S}})$ and $\tau(x)=x_S^T\tilde{\beta}$.*

For technical reasons, we also need bounds on the fourth moments of the feature vector and the error of the estimator for the control response.

Condition 8 *The fourth moments of the feature vector X are bounded:*

$$\mathbb{E}[\|X\|_2^4|W=1] \le C_X.$$

Condition 9 *There exists an m_0 such that for all $m > m_0$,*

$$\mathbb{E}\left[(\mu_0(X)-\hat{\mu}_0^m(X))^4\middle|W=1\right] \le C_\delta.$$

Here $\hat{\mu}_0^m$ is defined as in Condition 4.

This condition is satisfied, for example, when μ_0 is bounded.

Under these additional assumptions, the EMSE of the X-learner achieves the parametric rate in n, given that $m > m_0$.

Theorem 5 *Assume that Conditions 1–9 hold. Then the X-learner with $\hat{\mu}_0^m$ in the first stage and OLS in the second stage achieves the parametric rate in n. That is, there exists a constant C such that for all $m > m_0$ and $n > 1$,*

$$\mathbb{E}\left[(\tau(\mathcal{X}) - \hat{\tau}_X^{mn}(\mathcal{X}))^2 \Big| \sum_i W_i = n\right] \le C n^{-1}.$$

We will prove the following lemma first, because it will be useful for the proof of Theorem 5.

Lemma 4 *Under the assmuption of Theorem 5, there exists a constant C such that for all $n > n_0$, $m > m_0$, and $s > 0$,*

$$\mathbb{P}\left(n\|(X^{1\prime}X^1)^{-1}X^{1\prime}\delta\|_2^2 \ge s \Big| \sum_i W_i = n\right) \le C\frac{1}{s^2},$$

where $\delta_i = \mu_0(X_i^1) - \hat{\mu}_0^m(X_i^1)$.

Proof. [Proof of Lemma 4] To simplify the notation, we write X instead of X^1 for the feature matrix of the treated units, and we define the event of observing exactly n treated units as

$$E_n = \left\{\sum_{i=1}^n W_i = n\right\}.$$

We use Condition 6 and then Chebyshev's inequality to conclude that for all $n > n_0$ (n_0 is determined by Condition 6),

$$\begin{aligned}
\mathbb{P}\left(n\|(X'X)^{-1}X'\delta\|_2^2 \ge s \Big| E_n\right) &= \mathbb{P}\left(\frac{1}{n}\|\Sigma_n^{-1}X'\delta\|_2^2 \ge s \Big| E_n\right) \\
&\le \mathbb{P}\left(\frac{1}{n}\gamma_{\min}^{-2}(\Sigma_n)\|X'\delta\|_2^2 \ge s \Big| E_n\right) \\
&\le \mathbb{E}\left[\mathbb{P}\left(\frac{1}{n}C_\Sigma^2\|X'\delta\|_2^2 \ge s \Big| E_n, \delta\right) \Big| E_n\right] \\
&\le \mathbb{E}\left[\frac{C_\Sigma^4}{s^2 n^2}\mathrm{Var}\left(\|X'\delta\|_2^2 \Big| E_n, \delta\right) \Big| E_n\right].
\end{aligned}$$

Next we apply the Efron–Stein inequality to bound the variance term:

$$\mathrm{Var}\left(\|X'\delta\|_2^2 \Big| E_n, \delta\right) \le \frac{1}{2}\sum_{i=1}^n \mathbb{E}\left[(f(X) - f(X^{(i)}))^2 \Big| E_n, \delta\right].$$

Here $f(x) = \|x'\delta\|_2^2$, $X^{(i)} = (X_1, \dots, X_{i-1}, \tilde{X}_i, X_{i+1}, \dots, X_n)$, and $\tilde{X}$ is an independent copy of X.

Let us now bound the summands:

$$\begin{aligned}
&\mathbb{E}\left[(f(X) - f(X^{(i)}))^2 \middle| E_n, \delta\right] \\
=&\mathbb{E}\left[\left(\|X'\delta\|_2^2 - \|X'\delta - (X_i - \tilde{X}_i)\delta_i\|_2^2\right)^2 \middle| E_n, \delta\right] \\
=&\mathbb{E}\Bigg[\underbrace{\left(2\delta' X(X_i - \tilde{X}_i)\delta_i\right)^2}_{A} + \underbrace{\|(X_i - \tilde{X}_i)\delta_i\|_2^4}_{B} - \underbrace{4\delta' X(X_i - \tilde{X}_i)\delta_i \|(X_i - \tilde{X}_i)\delta_i\|_2^2}_{C} \Bigg| E_n, \delta\Bigg].
\end{aligned}$$

Let us first bound $\mathbb{E}[A|E_n, \delta]$:

$$\begin{aligned}
\mathbb{E}\left[\left(2\delta' X(X_i - \tilde{X}_i)\delta_i\right)^2 \middle| E_n, \delta\right] &= \mathbb{E}\left[4\sum_{j,k=1}^{n} \delta_j X_j'(X_i - \tilde{X}_i)\delta_i \delta_k X_k'(X_i - \tilde{X}_i)\delta_i \middle| E_n, \delta\right] \\
&\overset{(a)}{=} \mathbb{E}\left[4\sum_{j=1}^{n} (\delta_j X_j'(X_i - \tilde{X}_i)\delta_i)^2 \middle| E_n, \delta\right] \\
&\le 4\delta_i^4(n-1)\mathbb{E}\left[(X_1'(X_2 - \tilde{X}_2))^2 \middle| E_n, \delta\right] + 4\delta_i^4 \mathbb{E}\left[(X_1'(X_1 - \tilde{X}_1))^2 \middle| E_n, \delta\right] \\
&\le C_A \delta_i^4 n.
\end{aligned}$$

Here

$$C_A = 4\max\left(\mathbb{E}\left[(X_1'(X_2 - \tilde{X}_2))^2 \middle| E_n\right], \mathbb{E}\left[(X_1'(X_1 - \tilde{X}_1))^2 \middle| E_n\right]\right),$$

which is bounded by Condition 8. For equation (a) we used that for $k \neq j$; therefore, we have that either k or j is not equal to i. Without loss of generality let $j \neq i$. Then

$$\begin{aligned}
&E\left[\delta_j X_j'(X_i - \tilde{X}_i)\delta_i \delta_k X_k'(X_i - \tilde{X}_i)\delta_i \middle| E_n, \delta\right] \\
&= \delta_j \mathbb{E}\left[\mathbb{E}\left[X_j' \middle| W, E_n, \delta\right] \mathbb{E}\left[(X_i - \tilde{X}_i)\delta_i \delta_k X_k'(X_i - \tilde{X}_i)\delta_i \middle| W, E_n, \delta\right] \middle| E_n, \delta\right] \\
&= 0,
\end{aligned} \tag{A.15}$$

because $\mathbb{E}\left[X_j'|W, E_n, \delta\right] = 0$ as per the assumption.

In order to bound $\mathbb{E}[B|E_n, \delta]$, note that all the fourth moments of X are bounded and thus

$$\mathbb{E}\left[\|(X_i - \tilde{X}_i)\delta_i\|_2^4 \middle| E_n, \delta\right] \le C_B \delta_i^4.$$

Finally, we bound $\mathbb{E}[C|E_n,\delta]$:

$$\begin{aligned}\mathbb{E}\left[4\delta' X(X_i-\tilde{X}_i)\delta_i\|(X_i-\tilde{X}_i)\delta_i\|_2^2\Big|E_n,\delta\right] &= \mathbb{E}\left[\sum_{j=1}^n \delta_j X_j'(X_i-\tilde{X}_i)\delta_i\|(X_i-\tilde{X}_i)\delta_i\|_2^2\Big|E_n,\delta\right]\\ &= \mathbb{E}\left[\delta_i^4 X_i'(X_i-\tilde{X}_i)\|X_i-\tilde{X}_i\|_2^2\Big|E_n,\delta\Big|E_n,\delta\right]\\ &= C_C\delta_i^4,\end{aligned}$$

where the second equality follows from the same argument as in A.15, and the last equality is implied by Condition 8.

Plugging in terms A, B, and C, we have that for all $n > n_0$,

$$\operatorname{Var}\left(\|X'\delta\|_2^2\Big|E_n,\delta\right) \le \frac{1}{2}\sum_{i=1}^n E[(f(X,\delta)-f(X^{(i)},\delta^{(i)}))^2] \le C\delta^4 n^2,$$

with $C = C_A + C_B + C_C$. Thus for $n > n_0$,

$$\mathbb{P}\left(n\|(X'X)^{-1}X'\delta\|_2^2 \ge s\Big|E_n\right) \le \mathbb{E}\left[\frac{CC_\Sigma^4}{s^2}\delta^4\Big|E_n\right] \le CC_\Sigma^4 C_\delta \frac{1}{s^2}.$$

Proof. [Proof of Theorem 5] We start with the same decomposition as in A.12:

$$\mathbb{E}\left[(\tau(\mathcal{X})-\hat{\tau}_X^{mn}(\mathcal{X}))^2\big|E_n\right] \le \mathbb{E}\left[\|\mathcal{X}\|^2\right]\mathbb{E}\left[\|(X'X)^{-1}X'\delta\|^2 + \|(X'X)^{-1}X'\varepsilon\|^2\big|E_n\right],$$

and we follow the same steps to conclude that

$$\mathbb{E}\left[\|(X'X)^{-1}X'\varepsilon\|^2\big|E_n\right] \le \sigma^2 dC_\Sigma n^{-1} \qquad \text{and} \qquad \mathbb{E}\left[\|\mathcal{X}\|^2\right] \le C_{\mathcal{X}}.$$

From Lemma 4, we can conclude that there exists a constant C such that

$$\begin{aligned}\lim_{n\to\infty}\mathbb{E}\left[n\|(X'X)^{-1}X'\delta\|_2^2\big|E_n\right] &= \lim_{n\to\infty,n>n_0}\int_0^\infty \mathbb{P}\left(n\|(X'X)^{-1}X'\delta\|_2^2 \ge s\Big|E_n\right)ds\\ &\le \lim_{n\to\infty,n>n_0}\int_0^\infty \max(1, C\frac{1}{s^2})ds\\ &\le 1 + C.\end{aligned}$$

Thus there exists a $\tilde{C}$ such that for all $n > 1$,

$$\mathbb{E}\left[\|(X'X)^{-1}X'\delta\|_2^2\big|E_n\right] \le \tilde{C}n^{-1}.$$

EMSE convergence rate for Lipschitz continuous response functions

In Section A.9, we considered an example where the distribution of $(Y(0), Y(1), W, X)$ was assumed to be in some family $F \in S(a_\mu, a_\tau)$ with $a_\tau > a_\mu$, and we showed that one can expect the X-learner to outperform the T-learner in this case. Now we want to explore the case where $a_\tau \leq a_\mu$.

Let us first consider the case, where $a_\tau < a_\mu$. This is a somewhat artificial case, since having response functions that can be estimated at a rate of N^{-a_μ} implies that the CATE cannot be too complicated. For example, if μ_0 and μ_1 are Lipschitz continuous, then the CATE is Lipschitz continuous as well, and we would expect $a_\tau \approx a_\mu$. Even though it is hard to construct a case with $a_\tau < a_\mu$, we cannot exclude such a situation, and we would expect that in such a case the T-learner performs better than the X-learner.

We therefore believe that the case where $a_\tau \approx a_\mu$ is a more reasonable assumption than the case where $a_\tau < a_\mu$. In particular, we would expect the T- and X-learners to perform similarly when compared to their worst-case convergence rate. Let us try to back up this intuition with a specific example. Theorem 2 already confirms that $\hat{\tau}_1$ achieves the expected rate,

$$\mathcal{O}\left(m^{-a_\mu} + n^{-a_\tau}\right),$$

for the case where the CATE is linear. Below, we consider another example, where the CATE is of the same order as the response functions. We assume some noise level σ that is fixed, and we start by introducing a family F^L of distributions with Lipschitz continuous regression functions.

Definition 5 (Lipschitz continuous regression functions) *Let F^L be the class of distributions on $(X, Y) \in [0,1]^d \times \mathbb{R}$ such that:*

1. *The features, X_i, are i.i.d. uniformly distributed in $[0,1]^d$.*
2. *The observed outcomes are given by*

$$Y_i = \mu(X_i) + \varepsilon_i,$$

   where the ε_i is independent and normally distributed with mean 0 and variance σ^2.
3. *X_i and ε_i are independent.*
4. *The regression function μ is Lipschitz continuous with parameter L.*

Remark 4 *The optimal rate of convergence for the regression problem of estimating $x \mapsto \mathbb{E}[Y|X = x]$ in Definition 5 is $N^{-2/(2+d)}$. Furthermore, the KNN algorithm with the right choice of the number of neighbors and the Nadaraya–Watson estimator with the right kernels achieve this rate, and they are thus minimax optimal for this regression problem.*

Now let's define a related distribution on $(Y(0), Y(1), W, X)$.

Definition 6 *Let $\mathcal{D}^L_{mn}$ be the family of distributions of $(Y(0), Y(1), W, X) \in \mathbb{R}^N \times \mathbb{R}^N \times \{0,1\}^N \times [0,1]^{d\times N}$ such that:*

1. *$N = m + n$.*
2. *The features, X_i, are i.i.d. uniformly distributed in $[0,1]^d$.*
3. *There are exactly n treated units,*
$$\sum_i W_i = n.$$
4. *The observed outcomes are given by*
$$Y_i(w) = \mu_w(X_i) + \varepsilon_{wi},$$
where $(\varepsilon_{0i}, \varepsilon_{1i})$ is independent normally distributed with mean 0 and marginal variances σ^2.[3]
5. *X, W and $\varepsilon = (\varepsilon_{0i}, \varepsilon_{1i})$ are independent.*
6. *The response functions μ_0, μ_1 are Lipschitz continuous with parameter L.*

Note that if $(Y(0), Y(1), W, X)$ is distributed according to a distribution in D^L_{mn}, then $(Y(0), X)$ given $W = 0$ and $(Y(1), X)$ given $W = 1$ have marginal distributions in F^L, and $(X, \mu_1(X) - Y(0))$ given $W = 0$ and $(X, Y(1) - \mu_0(X))$ given $W = 1$ have distributions in F^{2L}, and we therefore conclude that $D^L_{mn} \in S\left(\frac{2}{2+d}, \frac{2}{2+d}\right)$.

We will first prove in Theorem 6 that the best possible rate that can be uniformly achieved for distributions in this family is

$$\mathcal{O}(n^{2/(2+d)} + m^{2/(2+d)}).$$

This is precisely the rate the T-learner with the right base learners achieves (Theorem 1). We will then show in Theorem 7 that the X-learner with the KNN estimator for both stages achieves this optimal rate as well, and conclude that both the T- and X-learners achieve the optimal minimax rate for this class of distributions.

[3]We do not assume that $\varepsilon_{0i} \perp \varepsilon_{1i}$.

Minimax lower bound

In this section, we will derive a lower bound on the best possible rate for $\mathcal{D}^L_{mn}$.

Theorem 6 (Minimax Lower Bound) *Let $\hat{\tau}$ be an arbitrary estimator, let $a_1, a_2 > 0$, and let c be such that for all $n, m \geq 1$,*

$$\sup_{\mathcal{P} \in \mathcal{D}^L_{mn}} EMSE(\mathcal{P}, \hat{\tau}^{mn}) \leq c(m^{-a_0} + n^{-a_1}); \tag{A.16}$$

then a_1 and a_2 are at most $2/(2+d)$:

$$a_0, a_1 \leq 2/(2+d).$$

Proof. [Proof of Theorem 6] To simplify the notation, we define $a = 2/(2+d)$. We will show by contradiction that $a_1 \leq a$. The proof of a_0 is mathematically symmetric. We assume that a_1 is bigger than a, and we show that this implies that there exists a sequence of estimators $\hat{\mu}^n_1$, such that

$$\sup_{\mathcal{P}_1 \in F^L} \mathbb{E}_{D^n_1 \sim \mathcal{P}^n_1} \left[(\mu_1(\mathcal{X}) - \hat{\mu}^n_1(\mathcal{X}; \mathcal{D}^n_1))^2 \right] \leq 2cn^{-a_1},$$

which is a contradiction, since by the definition of D^{mn}_L, μ_1 cannot be estimated at a rate faster than n^{-a} (cf., [30]). Note that we write here $\hat{\mu}^n_1(\mathcal{X}; \mathcal{D}^n_1)$, because we want to be explicit that $\hat{\mu}^n_1$ depends only on the treated observations.

Similiarly to $\hat{\mu}^n_1(\mathcal{X}; \mathcal{D}^n_1)$, we will use the notation $\hat{\tau}^{mn}(\mathcal{X}; \mathcal{D}^m_0, \mathcal{D}^n_1)$ to be explicit about the dependence of the estimator $\hat{\tau}^{mn}$ on the data in the control group, $\mathcal{D}^m_0$, and on the data in the treatment group, $\mathcal{D}^n_1$. Furthermore, note that in Definition 6 each distribution in $\mathcal{D}^L_{mn}$ is fully specified by the distribution of W, ε, and the functions μ_1 and μ_2. Define C_L to be the set of all functions $f : [0,1]^d \longrightarrow \mathbb{R}$ that are L-Lipschitz continuous. For $f_1 \in C_L$, define $\mathbb{D}(f_1)$ to be the distribution in $\mathcal{D}^L_{mn}$ with $\mu_0 = 0$, $\mu_1 = f_1$, $\varepsilon_0 \perp \varepsilon_1$, and W defined componentwise by

$$W_i = \begin{cases} 1 \text{ if } i \leq n \\ 0 \text{ otherwise.} \end{cases}$$

Then A.16 implies that

$$\begin{aligned} c(m^{-a_0} + n^{-a_1}) &\geq \sup_{\mathcal{P} \in \mathcal{D}^L_{mn}} \mathbb{E}_{(\mathcal{D}^m_0 \times \mathcal{D}^n_1) \sim \mathcal{P}} \left[(\tau^{\mathcal{P}}(\mathcal{X}) - \hat{\tau}^{mn}(\mathcal{X}; \mathcal{D}^m_0, \mathcal{D}^n_1))^2 \right] \\ &\geq \sup_{f_1 \in C_L} \mathbb{E}_{(\mathcal{D}^m_0 \times \mathcal{D}^n_1) \sim \mathbb{D}(f_1)} \left[(\mu_1^{\mathbb{D}(f_1)}(\mathcal{X}) - \hat{\tau}^{mn}(\mathcal{X}; \mathcal{D}^m_0, \mathcal{D}^n_1))^2 \right]. \end{aligned}$$

This follows, because in $\mathbb{D}(f_1)$, $\tau^{\mathbb{D}(f_1)} = \mu_1^{\mathbb{D}(f_1)} = f_1$. We use here the notation $\tau^{\mathcal{P}}$, $\tau^{\mathbb{D}(f_1)}$, and $\mu_1^{\mathbb{D}(f_1)}$ to emphasize that those terms depend on the distribution of $\mathcal{P}$ and $\mathbb{D}(f_1)$, respectively.

Let $\mathcal{P}_0$ be the distribution of $\mathcal{D}_0^m = (X_i^0, Y_i^0)_{i=1}^N$ under $\mathbb{D}(f_1)$. Note that under $\mathcal{P}_0$, $X_i \overset{iid}{\sim} [0,1]$, and $Y^0 \overset{iid}{\sim} \mathbb{N}(0,\sigma^2)$, and X^0 and Y^0 are independent. In particular, $\mathcal{P}_0$ does not depend on f_1. We can thus write

$$\begin{aligned}
c(m^{-a_0} + n^{-a_1}) &\geq \sup_{f_1 \in C_L} \mathbb{E}_{(\mathcal{D}_0^m \times \mathcal{D}_1^n) \sim \mathbb{D}(f_1)} \left[\left(\mu_1^{\mathbb{D}(f_1)}(\mathcal{X}) - \hat{\tau}^{mn}(\mathcal{X}; \mathcal{D}_0^m, \mathcal{D}_1^n) \right)^2 \right] \\
&= \sup_{f_1 \in C_L} \mathbb{E}_{\mathcal{D}_1^n \sim \mathbb{D}_1(f_1)} \mathbb{E}_{\mathcal{D}_0^n \sim \mathcal{P}_0} \left[\left(\mu_1^{\mathbb{D}_1(f_1)}(\mathcal{X}) - \hat{\tau}^{mn}(\mathcal{X}; \mathcal{D}_0^m, \mathcal{D}_1^n) \right)^2 \right] \\
&\geq \sup_{f_1 \in C_L} \mathbb{E}_{\mathcal{D}_1^n \sim \mathbb{D}_1(f_1)} \left[\left(\mu_1^{\mathbb{D}_1(f_1)}(\mathcal{X}) - \mathbb{E}_{\mathcal{D}_0^n \sim \mathcal{P}_0} \hat{\tau}^{mn}(\mathcal{X}; \mathcal{D}_0^m, \mathcal{D}_1^n) \right)^2 \right].
\end{aligned}$$

$\mathbb{D}_1(f_1)$ is here the distribution of $\mathcal{D}_1^n$ under $\mathbb{D}(f_1)$. For the last step we used Jensen's inequality.

Now choose a sequence m_n in such a way that $m_n^{-a_1} + n^{-a_2} \leq 2n^{-a_1}$, and define

$$\hat{\mu}_1^n(x; \mathcal{D}_1^n) = \mathbb{E}_{\mathcal{D}_0^{m_n} \sim \mathcal{P}_0^{m_n}} \left[\hat{\tau}^{mn}(x; \mathcal{D}_0^{m_n}, \mathcal{D}_1^n) \right].$$

Furthermore, note that

$$\{\mathbb{D}_1(f_1) \ : \ f_1 \in C_L\} = \left\{\mathcal{P}_1 \in F^L\right\}$$

in order to conclude that

$$\begin{aligned}
2cn^{-a_1} \geq c(m_n^{-a_0} + n^{-a_1}) &\geq \sup_{f_1 \in C_L} \mathbb{E}_{\mathcal{D}_1^n \sim \mathbb{D}_1(f_1)} \left[\left(\mu_1^{\mathbb{D}_1(f_1)}(\mathcal{X}) - \hat{\mu}_1^{nm}(D_1^n; \mathcal{X}) \right)^2 \right] \\
&\geq \sup_{\mathcal{P}_1 \in F^L} \mathbb{E}_{D_1^n \sim \mathcal{P}_1^n} \left[\left(\mu_1^{\mathcal{P}_1^n}(\mathcal{X}) - \hat{\mu}_1^{nm}(D_1^n; \mathcal{X}) \right)^2 \right].
\end{aligned}$$

This is, however, a contradiction, because we assumed $a_1 > a$.

EMSE convergence of the X-learner

Finally, we can show that the X-learner with the right choice of base learners achieves this minimax lower bound.

Theorem 7 *Let $d > 2$ and assume $(X, W, Y(0), Y(1)) \sim \mathcal{P} \in \mathcal{D}_{mn}^L$. In particular, μ_0 and μ_1 are Lipschitz continuous with constant L,*

$$|\mu_w(x) - \mu_w(z)| \leq L\|x - z\| \quad \text{for } w \in \{0,1\},$$

and $X \sim Unif([0,1]^d)$.

Furthermore, let $\hat{\tau}^{mn}$ be the X-learner with

- *$g \equiv 0$,*

- *the base learner of the first stage for the control group $\hat{\mu}_0$, is a KNN estimator with constant $k_0 = \left\lceil (\sigma^2/L^2)^{\frac{d}{2+d}} m^{\frac{2}{d+2}} \right\rceil$,*
- *the base learner of the second stage for the treatment group, $\hat{\tau}_1$, is a KNN estimator with constant $k_1 = \left\lceil (\sigma^2/L^2)^{\frac{d}{2+d}} n^{\frac{2}{d+2}} \right\rceil$.*

Then $\hat{\tau}^{mn}$ achieves the optimal rate as given in Theorem 6. That is, there exists a constant C such that

$$\mathbb{E}\|\tau - \hat{\tau}^{mn}\|^2 \le C\sigma^{\frac{4}{d+2}} L^{\frac{2d}{2+d}} \left(m^{-2/(2+d)} + n^{-2/(2+d)}\right). \tag{A.17}$$

Note that in the third step of the X-learner, Equation [1.9], $\hat{\tau}_0$ and $\hat{\tau}_1$ are averaged:

$$\hat{\tau}^{mn}(x) = g(x)\hat{\tau}_0^{mn}(x) + (1 - g(x))\hat{\tau}_1^{mn}(x).$$

By choosing $g \equiv 0$, we are analyzing $\hat{\tau}_1^{mn}$. By a symmetry argument it is straightforward to show that with the right choice of base learners, $\hat{\tau}_0^{mn}$ also achieves a rate of $\mathcal{O}\left(m^{-2/(2+d)} + n^{-2/(2+d)}\right)$. With this choice of base learners the X-learner achieves this optimal rate for every choice of g.

We first state two useful lemmata that we will need in the proof of this theorem.

Lemma 5 *Let $\hat{\mu}_0^m$ be a KNN estimator based only on the control group with constant k_0, and let $\hat{\mu}_1^n$ be a KNN estinator based on the treatment group with constant k_1; then, by the assumption of Theorem 7,*

$$\mathbb{E}[\|\hat{\mu}_0^m - \mu_0\|^2] \le \frac{\sigma^2}{k_0} + cL^2\left(\frac{k_0}{m}\right)^{2/d},$$
$$\mathbb{E}[\|\hat{\mu}_1^n - \mu_1\|^2] \le \frac{\sigma^2}{k_1} + cL^2\left(\frac{k_1}{n}\right)^{2/d},$$

for some constant c.

Proof. [Proof of Lemma 5] This is a direct implication of Theorem 6.2 in [30].

Lemma 6 *Let $x \in [0,1]^d$, $X_1, \ldots, X_n \overset{iid}{\sim} Unif([0,1]^d)$ and $d > 2$. Define $\tilde{X}(x)$ to be the nearest neighbor of x; then there exists a constant c such that for all $n > 0$,*

$$\mathbb{E}\|\tilde{X}(x) - x\|^2 \le \frac{c}{n^{2/d}}.$$

Proof. [Proof of Lemma 6] First of all we consider

$$\mathbb{P}(\|\tilde{X}(x) - x\| \ge \delta) = (1 - \mathbb{P}(\|X_1 - x\| \le \delta))^n \le (1 - \tilde{c}\delta^d)^n \le e^{-\tilde{c}\delta^d n}.$$

Now we can compute the expectation:

$$\mathbb{E}\|\tilde{X}(x) - x\|^2 = \int_0^\infty \mathbb{P}(\|\tilde{X}(x) - x\| \ge \sqrt{\delta})d\delta \le \int_0^d e^{-\tilde{c}\delta^{d/2}n} d\delta \le \frac{1 - \frac{1}{-d/2+1}}{(\tilde{c}n)^{2/d}}.$$

Proof. [Proof of Theorem 7] Many ideas in this proof are motivated by [30] and [4]. Furthermore, note that we restrict our analysis here only to $\hat{\tau}_1^{mn}$, but the analysis of $\hat{\tau}_0^{mn}$ follows the same steps.

We decompose $\hat{\tau}_1^{mn}$ into

$$\hat{\tau}_1^{mn}(x) = \frac{1}{k_1}\sum_{i=1}^{k_1}\left[Y^1_{(i,n)}(x) - \hat{\mu}_0^m\left(X^1_{(i,n)}(x)\right)\right] = \hat{\mu}_1^n(x) - \frac{1}{k_1}\sum_{i=1}^{k_1}\hat{\mu}_0^m\left(X^1_{(i,n)}(x)\right),$$

where the notation that $\Big(\big(X^w_{(1,n_w)}(x), Y^w_{(1,n_w)}(x)\big), \ldots, \big(X^w_{(n_w,n_w)}(x), Y^w_{(n_w,n_w)}(x)\big)\Big)$ is a reordering of the tuples $\big(X_j^w(x), Y_j^w(x)\big)$ such that $\|X^w_{(i,n_w)}(x) - x\|$ is increasing in i. With this notation we can write the estimators of the first stage as

$$\hat{\mu}_0^m(x) = \frac{1}{k_0}\sum_{i=1}^{k_0} Y^0_{(i,m)}(x), \qquad \text{and} \qquad \hat{\mu}_1^n(x) = \frac{1}{k_1}\sum_{i=1}^{k_1} Y^1_{(i,n)}(x),$$

and we can upper bound the EMSE with the following sum:

$$\begin{aligned}
&\mathbb{E}[|\tau(\mathcal{X}) - \hat{\tau}_1^{mn}(\mathcal{X})|^2] \\
&= \mathbb{E}\Big[\Big|\mu_1(\mathcal{X}) - \mu_0(\mathcal{X}) - \hat{\mu}_1^n(\mathcal{X}) + \frac{1}{k_1}\sum_{i=1}^{k_1}\hat{\mu}_0^m(X^1_{(i,n)}(\mathcal{X}))\Big|^2\Big] \\
&\le 2\mathbb{E}\left[|\mu_1(\mathcal{X}) - \hat{\mu}_1^n(\mathcal{X})|^2\right] + 2\mathbb{E}\Big[\Big|\mu_0(\mathcal{X}) - \frac{1}{k_1}\sum_{i=1}^{k_1}\hat{\mu}_0^m(X^1_{(i,n)}(\mathcal{X}))\Big|^2\Big].
\end{aligned}$$

The first term corresponds to the regression problem of estimating the treatment response function in the first step of the X-learner and we can control this term with Lemma 5:

$$\mathbb{E}[\|\mu_1 - \hat{\mu}_1^n\|^2] \le \frac{\sigma^2}{k_1} + c_1 L^2\left(\frac{k_1}{n}\right)^{2/d}.$$

The second term is more challenging:

$$\begin{aligned}
&\frac{1}{2}\mathbb{E}\Big[\Big|\mu_0(\mathcal{X}) - \frac{1}{k_1}\sum_{i=1}^{k_1}\hat{\mu}_0^m(X^1_{(i,n)}(\mathcal{X}))\Big|^2\Big] \\
&\le \mathbb{E}\Big[\Big|\mu_0(\mathcal{X}) - \frac{1}{k_1k_0}\sum_{i=1}^{k_1}\sum_{j=1}^{k_0}\mu_0\Big(X^0_{(j,m)}(X^1_{(i,n)}(\mathcal{X}))\Big)\Big|^2\Big] && \text{(A.18)} \\
&+ \mathbb{E}\Big[\Big|\frac{1}{k_1k_0}\sum_{i=1}^{k_1}\sum_{j=1}^{k_0}\mu_0\Big(X^0_{(j,m)}(X^1_{(i,n)}(\mathcal{X}))\Big) - \frac{1}{k_1}\sum_{i=1}^{k_1}\hat{\mu}_0^m(X^1_{(i,n)}(\mathcal{X}))\Big|^2\Big]. && \text{(A.19)}
\end{aligned}$$

A.19 can be bound as follows:

$$
\begin{aligned}
[A.19] &= \mathbb{E}\left(\frac{1}{k_1k_0}\sum_{i=1}^{k_1}\sum_{j=1}^{k_0}\mu_0\Big(X^0_{(j,m)}(X^1_{(i,n)}(\mathcal{X}))\Big) - Y^0_{(j,m)}(X^1_{(i,n)}(\mathcal{X}))\right)^2\\
&\le \max_i \frac{1}{k_m^2}\sum_{j=1}^{k_0}\mathbb{E}\left(\mu_0\Big(X^0_{(j,m)}(X^1_{(i,n)}(\mathcal{X}))\Big) - Y^0_{(j,m)}(X^1_{(i,n)}(\mathcal{X}))\right)^2\\
&= \max_i \frac{1}{k_m^2}\sum_{j=1}^{k_0}\mathbb{E}\bigg[\mathbb{E}\Big[\left(\mu_0\Big(X^0_{(j,m)}(X^1_{(i,n)}(\mathcal{X}))\Big) - Y^0_{(j,m)}(X^1_{(i,n)}(\mathcal{X}))\right)^2\Big\|\mathcal{D},\mathcal{X}\Big]\bigg] \le \frac{\sigma^2}{k_0}.
\end{aligned}
$$

The last inequality follows from the assumption that, conditional on $\mathcal{D}$,

$$
Y^0_{(j,m)}(x) \sim \mathcal{N}\left(\mu_0\left(X^0_{(j,m)}(x)\right), \sigma^2\right).
$$

Next we find an upper bound for [A.18]:

$$
\begin{aligned}
[A.18] &\le \mathbb{E}\left(\frac{1}{k_1k_0}\sum_{i=1}^{k_1}\sum_{j=1}^{k_0}\Big\|\mu_0(\mathcal{X}) - \mu_0\Big(X^0_{(j,m)}(X^1_{(i,n)}(\mathcal{X}))\Big)\Big\|\right)^2\\
&\le \mathbb{E}\left(\frac{1}{k_1k_0}\sum_{i=1}^{k_1}\sum_{j=1}^{k_0}L\Big\|\mathcal{X} - X^0_{(j,m)}(X^1_{(i,n)}(\mathcal{X}))\Big\|\right)^2\\
&\le L^2\frac{1}{k_1k_0}\sum_{i=1}^{k_1}\sum_{j=1}^{k_0}\mathbb{E}\Big\|\mathcal{X} - X^0_{(j,m)}(X^1_{(i,n)}(\mathcal{X}))\Big\|^2 && \text{(A.20)}\\
&\le L^2\frac{1}{k_1}\sum_{i=1}^{k_1}\mathbb{E}\Big\|\mathcal{X} - X^1_{(i,n)}(\mathcal{X})\Big\|^2 && \text{(A.21)}\\
&+ L^2\frac{1}{k_1k_0}\sum_{i=1}^{k_1}\sum_{j=1}^{k_0}\mathbb{E}\Big\|X^1_{(i,n)}(\mathcal{X}) - X^0_{(j,m)}(X^1_{(i,n)}(\mathcal{X}))\Big\|^2 && \text{(A.22)}
\end{aligned}
$$

where [A.20] follows from Jensen's inequality.

Let's consider [A.21]. We partition the data into $A_1, \dots, A_{k_1}$ sets, where the first $k_1 - 1$ sets have $\lfloor \frac{n}{k_1} \rfloor$ elements and we define $\tilde{X}_{i,1}(x)$ to be the nearest neighbor of x in A_i. Then we can conclude that

$$
\begin{aligned}
\frac{1}{k_1}\sum_{i=1}^{k_1}\mathbb{E}\Big\|\mathcal{X} - X^1_{(i,n)}(\mathcal{X})\Big\|^2 &\le \frac{1}{k_1}\sum_{i=1}^{k_1}\mathbb{E}\Big\|\mathcal{X} - \tilde{X}_{i,1}(\mathcal{X})\Big\|^2\\
&= \frac{1}{k_1}\sum_{i=1}^{k_1}\mathbb{E}\Big[\mathbb{E}\Big[\Big\|\mathcal{X} - \tilde{X}_{i,1}(\mathcal{X})\Big\|^2\Big|\mathcal{X}\Big]\Big] \le \frac{\tilde{c}}{\lfloor \frac{n}{k_1} \rfloor^{2/d}}.
\end{aligned}
$$

Here the last inequality follows from Lemma 6. With exactly the same argument, we can bound [A.22] and we thus have

$$[A.18] \leq L^2 \tilde{c} * \left(\frac{1}{\lfloor \frac{n}{k_1} \rfloor^{2/d}} + \frac{1}{\lfloor \frac{n_2}{k_2} \rfloor^{2/d}} \right) \leq 2\tilde{c} L^2 * \left(\left(\frac{k_1}{n} \right)^{2/d} + \left(\frac{k_0}{m} \right)^{2/d} \right).$$

Plugging everything in, we have

$$\begin{aligned} \mathbb{E}[|\tau(\mathcal{X}) - \hat{\tau}_1^{mn}(\mathcal{X})|^2] &\leq 2\frac{\sigma^2}{k_1} + 2(c_2 + 2\tilde{c})L^2 \left(\frac{k_1}{n} \right)^{2/d} + 2\frac{\sigma^2}{k_0} + 4\tilde{c}L^2 \left(\frac{k_0}{m} \right)^{2/d} \\ &\leq C \left(\frac{\sigma^2}{k_1} + L^2 \left(\frac{k_1}{n} \right)^{2/d} + \frac{\sigma^2}{k_0} + \left(\frac{k_0}{m} \right)^{2/d} \right) \end{aligned}$$

with $C = 2\max(1, c_2 + 2\tilde{c}, 2\tilde{c})$.

A.10 Pseudocode

In this section, we present pseudocode for the algorithms in this paper. We denote by Y^0 and Y^1 the observed outcomes for the control group and the treatment group, respectively. For example, Y_i^1 is the observed outcome of the ith unit in the treatment group. X^0 and X^1 are the features of the control units and the treated units, and hence X_i^1 corresponds to the feature vector of the ith unit in the treatment group. $M_k(Y \sim X)$ is the notation for a regression estimator, which estimates $x \mapsto \mathbb{E}[Y|X = x]$. It can be any regression/machine learning estimator. In particular, it can be a black box algorithm.

Algorithm 5 T-learner

```
1: procedure T-LEARNER($X, Y, W$)
2:     $\hat{\mu}_0 = M_0(Y^0 \sim X^0)$
3:     $\hat{\mu}_1 = M_1(Y^1 \sim X^1)$

4:     $\hat{\tau}(x) = \hat{\mu}_1(x) - \hat{\mu}_0(x)$
```

M_0 and M_1 are here some, possibly different, machine-learning/regression algorithms.

Algorithm 6 S-learner

```
1: procedure S-LEARNER($X, Y, W$)
2:     $\hat{\mu} = M(Y \sim (X, W))$
3:     $\hat{\tau}(x) = \hat{\mu}(x, 1) - \hat{\mu}(x, 0)$
```

$M(Y \sim (X, W))$ is the notation for estimating $(x, w) \mapsto \mathbb{E}[Y|X = x, W = w]$ while treating W as a 0,1–valued feature.

Algorithm 7 X-learner

```
1: procedure X-LEARNER($X, Y, W, g$)

2:     $\hat{\mu}_0 = M_1(Y^0 \sim X^0)$                          ▷ Estimate response function
3:     $\hat{\mu}_1 = M_2(Y^1 \sim X^1)$

4:     $\tilde{D}^1_i = Y^1_i - \hat{\mu}_0(X^1_i)$                ▷ Compute imputed treatment effects
5:     $\tilde{D}^0_i = \hat{\mu}_1(X^0_i) - Y^0_i$

6:     $\hat{\tau}_1 = M_3(\tilde{D}^1 \sim X^1)$                  ▷ Estimate CATE in two ways
7:     $\hat{\tau}_0 = M_4(\tilde{D}^0 \sim X^0)$

8:     $\hat{\tau}(x) = g(x)\hat{\tau}_0(x) + (1 - g(x))\hat{\tau}_1(x)$    ▷ Average the estimates
```

$g(x) \in [0, 1]$ is a weighting function that is chosen to minimize the variance of $\hat{\tau}(x)$. It is sometimes possible to estimate $\mathrm{Cov}(\tau_0(x), \tau_1(x))$, and compute the best g based on this estimate. However, we have made good experiences by choosing g to be an estimate of the propensity score.

Algorithm 8 F-learner

```
1: procedure F-LEARNER($X, Y, W$)
2:     $\hat{e} = M_e[W \sim X]$
3:     $Y^*_i = Y_i \frac{W_i - \hat{e}(X_i)}{\hat{e}(X_i)(1 - \hat{e}(X_i))}$
4:     $\hat{\tau} = M_\tau(Y^* \sim X)$
```

Algorithm 9 U-learner

```
1: procedure U-LEARNER($X, Y, W$)
2:     $\hat{\mu}_{obs} = M_{obs}(Y^{obs} \sim X)$
3:     $\hat{e} = M_e[W \sim X]$
4:     $R_i = (Y_i - \hat{\mu}_{obs}(X_i))/(W_i - \hat{e}(X_i))$
5:     $\hat{\tau} = M_\tau(R \sim X)$
```

Algorithm 10 Bootstrap Confidence Intervals 1

```
1: procedure computeCI(
   $x$: features of the training data,
   $w$: treatment assignments of the training data,
   $y$: observed outcomes of the training data,
   $p$: point of interest)
2:     $S_0 = \{i : w_i = 0\}$
3:     $S_1 = \{i : w_i = 1\}$
4:     $n_0 = \#S_0$
5:     $n_1 = \#S_1$
6:     for $b$ in $\{1, \ldots, B\}$ do
7:         $s_b^* = c(\text{sample}(S_0, \text{replace} = \text{T}, \text{size} = n_0), \text{sample}(S_1, \text{replace} = \text{T}, \text{size} = n_1))$
8:         $x_b^* = x[s_b^*]$
9:         $w_b^* = w[s_b^*]$
10:        $y_b^* = y[s_b^*]$
11:        $\hat{\tau}_b^*(p) = \text{learner}(x_b^*, w_b^*, y_b^*)(p)$
12:    $\hat{\tau}(p) = \text{learner}(x, w, y)(p)$
13:    $\sigma = sd(\{\hat{\tau}_b^*(p)\}_{b=1}^B)$
14:    return $(\hat{\tau}(p) - q_{\alpha/2}\sigma, \hat{\tau}(p) + q_{1-\alpha/2}\sigma)$
```

For this pseudo code we use `R` notation. For example, $c()$ is here a function that combines its arguments to form a vector.

Algorithm 11 Bootstrap Confidence Intervals 2

```
1:  procedure COMPUTECI(
      $x$: features of the training data,
      $w$: treatment assignments of the training data,
      $y$: observed outcomes of the training data,
      $p$: point of interest)
2:      $S_0 = \{i \ : \ w_i = 0\}$
3:      $S_1 = \{i \ : \ w_i = 1\}$
4:      $n_0 = \#S_0$
5:      $n_1 = \#S_1$
6:      for $b$ in $\{1, \dots, B\}$ do
7:          $s^*_b = c(\text{sample}(S_0, \text{replace} = \text{T}, \text{size} = n_0), \text{sample}(S_1, \text{replace} = \text{T}, \text{size} = n_1))$
8:          $x^*_b = x[s^*_b]$
9:          $w^*_b = w[s^*_b]$
10:         $y^*_b = y[s^*_b]$
11:         $\hat\tau^*_b(p) = \text{learner}(x^*_b, w^*_b, y^*_b)(p)$
12:     $\tilde\tau(p) = \frac{1}{B}\sum_{b=1}^B \hat\tau^*_b(p)$
13:     For all $b$ in $\{1, \dots, B\}$ and $j$ in $\{1, \dots, n\}$ define
            $$S^*_{bj} = \#\{k : s^*_b[k] = j\}$$
14:     For all $j$ in $\{1, \dots, n\}$ define $\overline{S^*_{\cdot j}} = \frac{1}{B}\sum_{b=1}^B S^*_{bj}$ and
            $$\text{Cov}_j = \frac{1}{B}\sum_{b=1}^B (\hat\tau^*_b(p) - \tilde\tau(p))(S^*_{bj} - \overline{S^*_{\cdot j}})$$
15:     $\sigma = \left(\sum_{j=1}^n Cov_j^2\right)^{0.5}$
16:     return $(\tilde\tau(p) - q_{\alpha/2}\sigma, \tilde\tau(p) + q_{1-\alpha/2}\sigma)$
```

This version of the bootstrap was proposed in [20].

Algorithm 12 Monte Carlos Bias Approximation

```
1: procedure APPROXIMATEBIAS(
       $x$: features of the full data set,
       $w$: treatment assignments of the full data set,
       $y(0)$: potential outcome under control of the full data set,
       $y(1)$: potential outcome under treatment of the full data set,
       $S$: indices of observations that are not in the test set,
       $S_T$: indices of the training set,
       $p$: point of interest,
       $\tau(p)$: the true CATE at p)
2:     for $i$ in $\{1, \dots, 1000\}$ do
3:         Create a new treatment assignment by permuting the original one,
               $$w_i = \text{sample}(w, \text{replace} = F).$$
4:         Define the observed outcome,
               $$y_i = y(1)w_i + y(0)(1 - w_i).$$
5:         Sample uniformly a training set of 50,000 observations,
               $$\begin{aligned} s_i^* &= \text{sample}(S, \text{replace} = F, \text{size} = 50,000), \\ w_i^* &= w_i[s_i^*], \\ x_i^* &= x[s_i^*], \\ y_i^* &= y_i[s_i^*]. \end{aligned}$$
6:         Estimate the CATE,
               $$\hat{\tau}_i^*(p) = \text{learner}(x_i^*, w_i^*, y_i^*)(p).$$
7:     $\bar{\tau}^*(p) = \frac{1}{1000} \sum_{i=1}^{1000} \hat{\tau}_i^*(p)$
8:     return $\bar{\tau}^*(p) - \tau(p)$
```

This algorithm is used to compute the bias in a simulation study where the potential outcomes and the CATE function are known. S, the indices of the units that are not in the test set and S_T, the indices of the units in the training set are not the same, because the training set is in this case a subset of 50,000 units of the full data set.

Algorithm 13 Bootstrap Bias

```
1: procedure ESTIMATEBIAS(
   $x$: features of the training data,
   $w$: treatment assignments of the training data,
   $y$: observed outcomes of the training data,
   $p$: point of interest)
2:     $S_0 = \{i \ : \ w_i = 0\}$
3:     $S_1 = \{i \ : \ w_i = 1\}$
4:     $n_0 = \#S_0$
5:     $n_1 = \#S_1$
6:     for $b$ in $\{1, \dots, B\}$ do
7:         $s_b^* = c(\text{sample}(S_0, \text{replace} = \text{T}, \text{size} = n_0), \text{sample}(S_1, \text{replace} = \text{T}, \text{size} = n_1))$
8:         $x_b^* = x[s_b^*]$
9:         $w_b^* = w[s_b^*]$
10:        $y_b^* = y[s_b^*]$
11:        $\hat{\tau}_b^*(p) = \text{learner}(x_b^*, w_b^*, y_b^*)(p)$
12:    $\hat{\tau}(p) = \text{learner}(x, w, y)(p)$
13:    $\bar{\tau}^*(p) = \frac{1}{B} \sum_{i=1}^{B} \hat{\tau}_i^*(p)$
14:    return $\bar{\tau}^*(p) - \hat{\tau}(p)$
```

Appendix B

`CausalToolbox` Documentation

`CausalToolbox` is an R package that provides different tools for estimating heterogeneous treatment effects. It implements eight CATE estimators and several tools to select a well-performing estimator for a given data set. The following contains the documentation of the different functions in the package with example code.

`CateCI` *Method CateCI*

Description

Returns the estimated confidence intervals for the CATE.

Usage

```
CateCI(theObject, feature_new, method = "maintain_group_ratios",
  bootstrapVersion = "normalApprox", B = 2000, nthread = 0,
  verbose = TRUE)

## S4 method for signature 'CATEestimator'
CateCI(theObject, feature_new,
  method = "maintain_group_ratios", bootstrapVersion = "normalApprox",
  B = 2000, nthread = 0, verbose = TRUE)

## S4 method for signature 'S_BART'
CateCI(theObject, feature_new, verbose = FALSE)

## S4 method for signature 'X_BART'
CateCI(theObject, feature_new, verbose = FALSE)

## S4 method for signature 'T_BART'
CateCI(theObject, feature_new, verbose = FALSE)

## S4 method for signature 'M_BART'
CateCI(theObject, feature_new, verbose = FALSE)
```

Arguments

`theObject`	A ‘MetaLearner‘ object.
`feature_new`	A feature data frame.
`method`	Different versions of the bootstrap.
`bootstrapVersion`	Default is normalApprox, which will use the bootstrap normal approximation to get CI. Smoothed will use CI around the smoothed bootstrap as introduced by Efron 2014.
`B`	Number of bootstrap samples.
`nthread`	Number of threads to be used in parallel.
`verbose`	TRUE for detailed output, FALSE for no output.

Value

A data frame of estimated CATE confidence intervals.

Examples

```
## Not run:
require(causalToolbox)

# create example data set
simulated_experiment <- simulate_causal_experiment(
  ntrain = 1000,
  ntest = 1000,
  dim = 10
)
feat <- simulated_experiment$feat_tr
tr <- simulated_experiment$W_tr
yobs <- simulated_experiment$Yobs_tr
feature_test <- simulated_experiment$feat_te
```

```
# create the CATE estimator using Random Forests (RF)
xl_rf <- X_RF(feat = feat, tr = tr, yobs = yobs)
CateCI(xl_rf, feature_test, B = 500)

## End(Not run)
```

---

simulate Experiments *Simulate a Causal Experiment*

Description

simulate_correlation_matrix uses the C-vine method for simulating correlation matrices. (Refer to the referenced paper for details.)

simulate_causal_experiment simulates an RCT or observational data for causal effect estimation. It is mainly used to test different heterogenuous treatment effect estimation strategies.

Usage

```
simulate_correlation_matrix(dim, alpha)

simulate_causal_experiment(ntrain = nrow(given_features),
  ntest = nrow(given_features), dim = ncol(given_features),
  alpha = 0.1, feat_distribution = "normal", given_features = NULL,
  pscore = "rct5", mu0 = "sparseLinearStrong",
  tau = "sparseLinearWeak", testseed = NULL, trainseed = NULL)

pscores.simulate_causal_experiment

mu0.simulate_causal_experiment

tau.simulate_causal_experiment
```

Arguments

dim	Dimension of the data set.
alpha	Only used if given_features is not set and feat_distribution is chosen to be normal. It specifies how correlated the features can be. If alpha = 0, then the features are independent. If alpha is very large, then the features can be very correlated. Use the simulate_correlation_matrix function to get a better understanding of the impact of alpha.
ntrain	Number of training examples.
ntest	Number of test examples.
feat_distribution	Only used if given_features is not specified. Either "normal" or "unif." It specifies the distribution of the features.
given_features	This is used if we already have features and want to test the performance of different estimators for a particular set of features.
pscore, mu0, tau	Parameters that determine the propensity score, the response function for the control units, and tau, respectively. The different options can be seen using names(pscores.simulate_causal_experim names(mu0.simulate_causal_experiment), and names(tau.simulate_causal_experiment). This is implemented in this manner, because it enables the user to easily loop through the different estimators.
testseed	The seed used to generate the test data. If NULL, then the seed of the main session is used.
trainseed	The seed used to generate the training data. If NULL, then the seed of the main session is used.

Details

The function simulates causal experiments by generating the features, treatment assignment, observed Y values, and CATE for a test set and a training set. pscore, mu0, and tau define the response functions and the propensity score. For example, `pscore = "osSparse1Linear"` specifies that

$$e(x) = max(0.05, min(.95, x1/2 + 1/4))$$

and `mu0 ="sparseLinearWeak"` specifies that the response function for the control units is given by the simple linear function,

$$mu0(x) = 3x1 + 5x2.$$

Value

A correlation matrix.

A list with the following elements:

`setup_name`	Name of the setup.
`m_t_truth`	Function containing the response function of the treated units.
`m_c_truth`	Function containing the response function of the control units.
`propscore`	Propensity score function.
`alpha`	Chosen alpha.
`feat_te`	Data.frame containing the features of the test samples.
`W_te`	Numeric vector containing the treatment assignment of the test samples.
`tau_te`	Numeric vector containing the true conditional average treatment effects of the test samples.
`Yobs_te`	Numeric vector containing the observed Y values of the test samples.
`feat_tr`	Data.frame containing the features of the training samples.
`W_tr`	Numeric vector containing the treatment assignment of the training samples.
`tau_tr`	Numeric vector containing the true conditional average treatment effects of the training samples.
`Yobs_tr`	Numeric vector containing the observed Y values of the training samples.

References

- Daniel Lewandowskia, Dorota Kurowickaa, Harry Joe (2009). Generating Random Correlation Matrices Based on Vines and Extended Onion Method.
- Sören Künzel, Jasjeet Sekhon, Peter Bickel, and Bin Yu (2017). Meta-learners for Estimating Heterogeneous Treatment Effects Using Machine Learning.

See Also

`X-Learner`

Examples

```
require(causalToolbox)

ce_sim <- simulate_causal_experiment(
  ntrain = 20,
  ntest = 20,
  dim = 7
)

ce_sim

## Not run:
estimators <- list(
  S_RF = S_RF,
  T_RF = T_RF,
  X_RF = X_RF,
  S_BART = S_BART,
  T_BART = T_BART,
  X_BARTT = X_BART)

performance <- data.frame()
for(tau_n in names(tau.simulate_causal_experiment)){
  for(mu0_n in names(mu0.simulate_causal_experiment)) {
    ce <- simulate_causal_experiment(
      given_features = iris,
      pscore = "rct5",
      mu0 = mu0_n,
      tau = tau_n)

    for(estimator_n in names(estimators)) {
      print(paste(tau_n, mu0_n, estimator_n))

      trained_e <- estimators[[estimator_n]](ce$feat_tr, ce$W_tr, ce$Yobs_tr)
      performance <-
        rbind(performance,
              data.frame(
                mu0 = mu0_n,
                tau = tau_n,
                estimator = estimator_n,
                MSE = mean((EstimateCate(trained_e, ce$feat_te) -
                              ce$tau_te)^2)))
    }
  }
}

reshape2::dcast(data = performance, mu0 + tau ~ estimator)

## End(Not run)
```

`EstimateCate` *Method EstimateCate*

Description

Returns the estimated CATE.

Usage

```
EstimateCate(theObject, feature_new, ...)

## S4 method for signature 'M_RF'
EstimateCate(theObject, feature_new)

## S4 method for signature 'S_RF'
EstimateCate(theObject, feature_new)

## S4 method for signature 'S_BART'
EstimateCate(theObject, feature_new, verbose = FALSE)

## S4 method for signature 'X_RF'
EstimateCate(theObject, feature_new)

## S4 method for signature 'X_BART'
EstimateCate(theObject, feature_new, verbose = FALSE,
  return_CI = FALSE)

## S4 method for signature 'T_RF'
EstimateCate(theObject, feature_new)

## S4 method for signature 'T_BART'
```

```
EstimateCate(theObject, feature_new, verbose = FALSE,
  return_CI = FALSE)

## S4 method for signature 'M_BART'
EstimateCate(theObject, feature_new, verbose = FALSE,
  return_CI = FALSE)
```

Arguments

`theObject`	A 'MetaLearner' object.
`feature_new`	A feature data frame.
`...`	Additional parameters that are specific for some MetaLearners
`verbose`	TRUE for detailed output FALSE for no output
`return_CI`	If TRUE, return predictions and their confidence intervals;

`gotv` *Get Out To Vote*

Description

This is an example data set, and it has been created by looking at a certain subset of the "Social Pressure and Voter Turnout: Evidence from a Large-Scale Field Experiment" study that tested the impact of social pressure on voter turnout. A precise description and the full data set can be found at `https://isps.yale.edu/research/data/d001`.

The study consists of seven key, individual-level covariates, most of which are discrete: gender, age, and whether the registered individual voted in the primary elections in 2000, 2002, and 2004 or the general elections in 2000 and 2002. The sample was restricted to voters who had voted in the 2004 general election. The outcome of interest was the turnout in the 2006 primary election, which was an indicator variable, and the treatment was whether or not the subjects were elected to receive a mailer.

Usage

```
gotv
```

Format

An object of class `data.frame` with 229461 rows and 9 columns.

References

- Gerber AS, Green DP, Larimer CW (2008) Social Pressure and Voter Turnout: Evidence from a Large-Scale Field Experiment. Am Polit Sci Rev 102:33–48. `https://isps.yale.edu/research/publications/isps08-001`

`M-Learner` *M-Learners*

Description

`M_RF` is an implementation of the Modified Outcome Estimator with Random Forest (Breiman 2001) as the base learner.

`M_BART` is an implementation of the Modified Outcome Estimator with Bayesian Additive Regression Trees (Chipman et al. 2010) as the base learner.

Usage

```
M_RF(feat, tr, yobs, nthread = 0, verbose = FALSE,
  mu.forestry = list(relevant.Variable = 1:ncol(feat), ntree = 1000,
  replace = TRUE, sample.fraction = 0.8, mtry = round(ncol(feat) * 13/20),
  nodesizeSpl = 2, nodesizeAvg = 1, splitratio = 1, middleSplit = TRUE),
  e.forestry = list(relevant.Variable = 1:ncol(feat), ntree = 500,
  replace = TRUE, sample.fraction = 0.5, mtry = ncol(feat), nodesizeSpl =
  11, nodesizeAvg = 33, splitratio = 0.5, middleSplit = FALSE),
  tau.forestry = list(relevant.Variable = 1:ncol(feat), ntree = 1000,
  replace = TRUE, sample.fraction = 0.7, mtry = round(ncol(feat) * 17/20),
  nodesizeSpl = 5, nodesizeAvg = 6, splitratio = 0.8, middleSplit = TRUE))

M_BART(feat, tr, yobs, ndpost = 1200, ntree = 200, nthread = 1,
  mu.BART = list(sparse = FALSE, theta = 0, omega = 1, a = 0.5, b = 1,
  augment = FALSE, rho = NULL, usequants = FALSE, cont = FALSE, sigest =
  NA, sigdf = 3, sigquant = 0.9, k = 2, power = 2, base = 0.95, sigmaf =
  NA, lambda = NA, numcut = 100L, nskip = 100L), e.BART = list(sparse =
  FALSE, theta = 0, omega = 1, a = 0.5, b = 1, augment = FALSE, rho = NULL,
  usequants = FALSE, cont = FALSE, sigest = NA, sigdf = 3, sigquant = 0.9,
  k = 2, power = 2, base = 0.95, sigmaf = NA, lambda = NA, numcut = 100L,
  nskip = 100L), tau.BART = list(sparse = FALSE, theta = 0, omega = 1, a
  = 0.5, b = 1, augment = FALSE, rho = NULL, usequants = FALSE, cont =
  FALSE, sigest = NA, sigdf = 3, sigquant = 0.9, k = 2, power = 2, base =
  0.95, sigmaf = NA, lambda = NA, numcut = 100L, nskip = 100L))
```

Arguments

`feat`	A data frame containing the features.
`tr`	A numeric vector with 0 for control and 1 for treated variables.
`yobs`	A numeric vector containing the observed outcomes.
`nthread`	Number of threads which should be used to work in parallel.
`verbose`	TRUE for detailed output, FALSE for no output.

`mu.forestry, tau.forestry, e.forestry`

A list containing the hyperparameters for the `forestry` package that are used for estimating the response functions, the CATE, and the propensity score. These hyperparameters are passed to the `forestry` package. (Please refer to the forestry[1] package for a more detailed documentation of the hyperparamters.)

- `relevant.Variable` Variables that are only used in the first stage.
- `ntree` Numbers of trees used in the first stage.
- `replace` Sample with or without replacement in the first stage.
- `sample.fraction` Size of total samples drawn for the training data in the first stage.
- `mtry` Number of variables randomly selected in each splitting point.
- `nodesizeSpl` Minimum nodesize in the first stage for the observations in the splitting set. (See the details of the `forestry` package)
- `nodesizeAvg` Minimum nodesize in the first stage for the observations in the averaging set.
- `splitratio` Proportion of the training data used as the splitting dataset in the first stage.
- `middleSplit` If true, the split value will be exactly in the middle of two observations. Otherwise, it will take a point based on a uniform distribution between the two observations.

`ndpost`	Number of posterior draws.
`ntree`	Number of trees.

`mu.BART, e.BART, tau.BART`

Hyperparameters of the BART functions for the control and treated group. (Use `?BART::mc.wbart` for a detailed explanation of their effects.)

[1] `https://github.com/soerenkuenzel/forestry`

Details

The M-Learner estimates the CATE in two steps:

1. Estimate the response functions and the propensity score,
$$\mu_0(x) = E[Y(0)|X = x]$$
$$\mu_1(x) = E[Y(1)|X = x]$$
$$e(x) = E[W|X = x]$$
using the base learner and denote the estimates as $\hat{\mu}_0$, $\hat{\mu}_1$, and $\hat{e}$.
2. Define the adjusted modified outcomes as
$$R_i = (Z_i - \hat{e}(x_i))/(\hat{e}(x_i)[1 - \hat{e}(x_i)])(Y_i - \hat{\mu}_1(x_i)[1 - \hat{e}(x_i)] - \hat{\mu}_0(x_i)\hat{e}(x_i)).$$
Now employ the base learner to estimate
$$\tau(x) = E[R|X = x].$$
The result is the CATE estimator.

Value

An object from a class that contains the `CATEestimator` class. It should be used with one of the following functions: `EstimateCATE`, `CateCI`, and `CateBIAS`. The object has at least the following slots:

`feature_train`	A copy of feat.
`tr_train`	A copy of tr.
`yobs_train`	A copy of yobs.
`creator`	Function call that creates the CATE estimator. This is used for different bootstrap procedures.

References

- Sören Künzel, Jasjeet Sekhon, Peter Bickel, and Bin Yu (2017). MetaLearners for Estimating Heterogeneous Treatment Effects Using Machine Learning. `https://www.pnas.org/content/116/10/4156`
- Sören Künzel, Simon Walter, and Jasjeet Sekhon (2018). Causaltoolbox—Estimator Stability for Heterogeneous Treatment Effects. `https://arxiv.org/pdf/1811.02833.pdf`
- Daniel Rubin and Mark J van der Laan (2007). A Doubly Robust Censoring Unbiased Transformation. `https://www.ncbi.nlm.nih.gov/pubmed/22550646`

See Also

Other metalearners: `S-Learner`, `T-Learner`, `X-Learner`

Examples

```
require(causalToolbox)

# create example data set
simulated_experiment <- simulate_causal_experiment(
  ntrain = 1000,
  ntest = 1000,
  dim = 10
)
feat <- simulated_experiment$feat_tr
tr <- simulated_experiment$W_tr
yobs <- simulated_experiment$Yobs_tr
feature_test <- simulated_experiment$feat_te

# create the CATE estimator using Random Forests (RF)
xl_rf <- X_RF(feat = feat, tr = tr, yobs = yobs)
tl_rf <- T_RF(feat = feat, tr = tr, yobs = yobs)
sl_rf <- S_RF(feat = feat, tr = tr, yobs = yobs)
```

```
ml_rf <- M_RF(feat = feat, tr = tr, yobs = yobs)
xl_bt <- X_BART(feat = feat, tr = tr, yobs = yobs)
tl_bt <- T_BART(feat = feat, tr = tr, yobs = yobs)
sl_bt <- S_BART(feat = feat, tr = tr, yobs = yobs)
ml_bt <- M_BART(feat = feat, tr = tr, yobs = yobs)

cate_esti_xrf <- EstimateCate(xl_rf, feature_test)

# evaluate the performance.
cate_true <- simulated_experiment$tau_te
mean((cate_esti_xrf - cate_true) ^ 2)
## Not run:
# create confidence intervals via bootstrapping.
xl_ci_rf <- CateCI(xl_rf, feature_test, B = 500)

## End(Not run)
```

`S-Learner` *S-Learners*

Description

S_RF is an implementation of the S-Learner combined with Random Forests (Breiman 2001).

S_BART is an implementation of the S-Learner with Bayesian Additive Regression Trees (Chipman et al. 2010).

Usage

```
S_RF(feat, tr, yobs, nthread = 0, verbose = TRUE,
  mu.forestry = list(relevant.Variable = 1:ncol(feat), ntree = 1000,
  replace = TRUE, sample.fraction = 0.9, mtry = ncol(feat), nodesizeSpl =
  1, nodesizeAvg = 3, splitratio = 0.5, middleSplit = FALSE))

S_BART(feat, tr, yobs, ndpost = 1200, ntree = 200, nthread = 1,
  verbose = FALSE, mu.BART = list(sparse = FALSE, theta = 0, omega = 1,
  a = 0.5, b = 1, augment = FALSE, rho = NULL, usequants = FALSE, cont =
  FALSE, sigest = NA, sigdf = 3, sigquant = 0.9, k = 2, power = 2, base =
  0.95, sigmaf = NA, lambda = NA, numcut = 100L, nskip = 100L))
```

Arguments

`feat`	A data frame containing the features.
`tr`	A numeric vector with 0 for control and 1 for treated variables.
`yobs`	A numeric vector containing the observed outcomes.
`nthread`	Number of threads which should be used to work in parallel.
`verbose`	TRUE for detailed output, FALSE for no output.
`mu.forestry`	A list containing the hyperparameters for the `forestry` package that are used in $\hat{\mu}_0$. These hyperparameters are passed to the `forestry` package.
`ndpost`	Number of posterior draws.
`ntree`	Number of trees.
`mu.BART`	hyperparameters of the BART function. Use `?BART::mc.wbart` for a detailed explanation of their effects.

Details

In the S-Learner, the outcome is estimated using all of the features and the treatment indicator without giving the treatment indicator a special role. The predicted CATE for an individual unit is then the difference between the predicted values when the treatment assignment indicator is changed from control to treatment:

1. Estimate the joint response function

$$\mu(x, w) = E[Y|X = x, W = w]$$

   using the base learner. We denote the estimate as $\hat{\mu}$.
2. Define the CATE estimate as

$$\tau(x) = \hat{\mu}_1(x, 1) - \hat{\mu}_0(x, 0).$$

Value

Object of class `S_RF`. It should be used with one of the following functions `EstimateCATE`, `CateCI`, and `CateBIAS`. The object has the following slots:

`feature_train`	A copy of feat.
`tr_train`	A copy of tr.
`yobs_train`	A copy of yobs.
`m_0`	An object of class forestry that is fitted with the observed outcomes of the control group as the dependent variable.
`m_1`	An object of class forestry that is fitted with the observed outcomes of the treated group as the dependent variable.
`hyperparameter_list`	A list containting the hyperparameters of the three random forest algorithms used.
`creator`	Function call of `S_RF`. This is used for different bootstrap procedures.

References

- Sören Künzel, Jasjeet Sekhon, Peter Bickel, and Bin Yu (2017). MetaLearners for Estimating Heterogeneous Treatment Effects using Machine Learning. `https://www.pnas.org/content/116/10/4156`
- Sören Künzel, Simon Walter, and Jasjeet Sekhon (2018). Causaltoolbox—Estimator Stability for Heterogeneous Treatment Effects. `https://arxiv.org/pdf/1811.02833.pdf`
- Sören Künzel, Bradly Stadie, Nikita Vemuri, Varsha Ramakrishnan, Jasjeet Sekhon, and Pieter Abbeel (2018). Transfer Learning for Estimating Causal Effects using Neural Networks. `https://arxiv.org/pdf/1808.07804.pdf`

See Also

Other metalearners: `M-Learner`, `T-Learner`, `X-Learner`

Other metalearners: `M-Learner`, `T-Learner`, `X-Learner`

Examples

```
require(causalToolbox)

# create example data set
simulated_experiment <- simulate_causal_experiment(
  ntrain = 1000,
  ntest = 1000,
  dim = 10
)
feat <- simulated_experiment$feat_tr
tr <- simulated_experiment$W_tr
yobs <- simulated_experiment$Yobs_tr
feature_test <- simulated_experiment$feat_te

# create the CATE estimator using Random Forests (RF)
xl_rf <- X_RF(feat = feat, tr = tr, yobs = yobs)
tl_rf <- T_RF(feat = feat, tr = tr, yobs = yobs)
sl_rf <- S_RF(feat = feat, tr = tr, yobs = yobs)
ml_rf <- M_RF(feat = feat, tr = tr, yobs = yobs)
xl_bt <- X_BART(feat = feat, tr = tr, yobs = yobs)
tl_bt <- T_BART(feat = feat, tr = tr, yobs = yobs)
sl_bt <- S_BART(feat = feat, tr = tr, yobs = yobs)
ml_bt <- M_BART(feat = feat, tr = tr, yobs = yobs)
```

```
cate_esti_xrf <- EstimateCate(xl_rf, feature_test)

# evaluate the performance.
cate_true <- simulated_experiment$tau_te
mean((cate_esti_xrf - cate_true) ^ 2)
## Not run:
# create confidence intervals via bootstrapping.
xl_ci_rf <- CateCI(xl_rf, feature_test, B = 500)

## End(Not run)
```

`T-Learner` *T-Learners*

Description

`T_RF` is an implementation of the T-learner combined with Random Forest (Breiman 2001) for both response functions.

`T_BART` is an implementation of the T-learner with Bayesian Additive Regression Trees (Chipman et al. 2010) for both response functions.

Usage

```
T_RF(feat, tr, yobs, nthread = 0, verbose = TRUE,
  mu0.forestry = list(relevant.Variable = 1:ncol(feat), ntree = 1000,
  replace = TRUE, sample.fraction = 0.9, mtry = ncol(feat), nodesizeSpl =
  1, nodesizeAvg = 3, splitratio = 0.5, middleSplit = FALSE),
  mu1.forestry = list(relevant.Variable = 1:ncol(feat), ntree = 1000,
  replace = TRUE, sample.fraction = 0.9, mtry = ncol(feat), nodesizeSpl =
  1, nodesizeAvg = 3, splitratio = 0.5, middleSplit = FALSE))

T_BART(feat, tr, yobs, ndpost = 1200, ntree = 200, nthread = 1,
  verbose = FALSE, mu0.BART = list(sparse = FALSE, theta = 0, omega =
  1, a = 0.5, b = 1, augment = FALSE, rho = NULL, usequants = FALSE, cont =
  FALSE, sigest = NA, sigdf = 3, sigquant = 0.9, k = 2, power = 2, base =
  0.95, sigmaf = NA, lambda = NA, numcut = 100L, nskip = 100L),
  mu1.BART = list(sparse = FALSE, theta = 0, omega = 1, a = 0.5, b = 1,
  augment = FALSE, rho = NULL, usequants = FALSE, cont = FALSE, sigest =
  NA, sigdf = 3, sigquant = 0.9, k = 2, power = 2, base = 0.95, sigmaf =
  NA, lambda = NA, numcut = 100L, nskip = 100L))
```

Arguments

`feat`	A data frame containing the features.
`tr`	A numeric vector with 0 for control and 1 for treated variables.
`yobs`	A numeric vector containing the observed outcomes.
`nthread`	Number of threads which should be used to work in parallel.
`verbose`	TRUE for detailed output, FALSE for no output.
`mu0.forestry, mu1.forestry`	Lists containing the hyperparameters for the `forestry` package that are used in $\hat{\mu}_0$ and $\hat{\mu}_1$, respectively. These hyperparameters are passed to the `forestry` package. (Please refer to the `forestry`[2] package for a more detailed documentation of the hyperparamters.)
`ndpost`	Number of posterior draws.
`ntree`	Number of trees.
`mu0.BART, mu1.BART`	Hyperparameters of the BART functions for the control and treated group. (Use `?BART::mc.wbart` for a detailed explanation of their effects.)

[2] https://github.com/soerenkuenzel/forestry

Details

The CATE is estimated using two estimators:

1. Estimate the response functions
$$\mu_0(x) = E[Y(0)|X = x]$$
$$\mu_1(x) = E[Y(1)|X = x]$$
using the base leaner and denote the estimates as $\hat{\mu}_0$ and $\hat{\mu}_1$.
2. Define the CATE estimate as
$$\tau(x) = \hat{\mu}_1 - \hat{\mu}_0.$$

Value

Object of class `T_RF`. It should be used with one of the following functions `EstimateCATE`, `CateCI`, and `CateBIAS`. The object has the following slots:

`feature_train`	A copy of feat.
`tr_train`	A copy of tr.
`yobs_train`	A copy of yobs.
`m_0`	An object of class forestry that is fitted with the observed outcomes of the control group as the dependent variable.
`m_1`	An object of class forestry that is fitted with the observed outcomes of the treated group as the dependent variable.
`hyperparameter_list`	List containting the hyperparameters of the three random forest algorithms used.
`creator`	Function call of T_RF. This is used for different bootstrap procedures.

References

- Sören Künzel, Jasjeet Sekhon, Peter Bickel, and Bin Yu (2017). MetaLearners for Estimating Heterogeneous Treatment Effects using Machine Learning. `https://www.pnas.org/content/116/10/4156`
- Sören Künzel, Simon Walter, and Jasjeet Sekhon (2018). Causaltoolbox—Estimator Stability for Heterogeneous Treatment Effects. `https://arxiv.org/pdf/1811.02833.pdf`
- Sören Künzel, Bradly Stadie, Nikita Vemuri, Varsha Ramakrishnan, Jasjeet Sekhon, and Pieter Abbeel (2018). Transfer Learning for Estimating Causal Effects using Neural Networks. `https://arxiv.org/pdf/1808.07804.pdf`

See Also

Other metalearners: `M-Learner`, `S-Learner`, `X-Learner`

Examples

```
require(causalToolbox)

# create example data set
simulated_experiment <- simulate_causal_experiment(
  ntrain = 1000,
  ntest = 1000,
  dim = 10
)
feat <- simulated_experiment$feat_tr
tr <- simulated_experiment$W_tr
yobs <- simulated_experiment$Yobs_tr
feature_test <- simulated_experiment$feat_te

# create the CATE estimator using Random Forests (RF)
xl_rf <- X_RF(feat = feat, tr = tr, yobs = yobs)
tl_rf <- T_RF(feat = feat, tr = tr, yobs = yobs)
sl_rf <- S_RF(feat = feat, tr = tr, yobs = yobs)
ml_rf <- M_RF(feat = feat, tr = tr, yobs = yobs)
xl_bt <- X_BART(feat = feat, tr = tr, yobs = yobs)
```

```
tl_bt <- T_BART(feat = feat, tr = tr, yobs = yobs)
sl_bt <- S_BART(feat = feat, tr = tr, yobs = yobs)
ml_bt <- M_BART(feat = feat, tr = tr, yobs = yobs)

cate_esti_xrf <- EstimateCate(xl_rf, feature_test)

# evaluate the performance.
cate_true <- simulated_experiment$tau_te
mean((cate_esti_xrf - cate_true) ^ 2)
## Not run:
# create confidence intervals via bootstrapping.
xl_ci_rf <- CateCI(xl_rf, feature_test, B = 500)

## End(Not run)
```

`X-Learner` *X-Learners*

Description

X_RF is an implementation of the X-learner with Random Forests (Breiman 2001) in the first and second stage.

X_BART is an implementation of the X-learner with Bayesian Additive Regression Trees (Chipman et al. 2010) at the first and second stage

Usage

```
X_RF(feat, tr, yobs, predmode = "propmean", nthread = 0,
  verbose = FALSE, mu.forestry = list(relevant.Variable = 1:ncol(feat),
  ntree = 1000, replace = TRUE, sample.fraction = 0.8, mtry =
  round(ncol(feat) * 13/20), nodesizeSpl = 2, nodesizeAvg = 1, splitratio =
  1, middleSplit = TRUE), tau.forestry = list(relevant.Variable =
  1:ncol(feat), ntree = 1000, replace = TRUE, sample.fraction = 0.7, mtry =
  round(ncol(feat) * 17/20), nodesizeSpl = 5, nodesizeAvg = 6, splitratio =
  0.8, middleSplit = TRUE), e.forestry = list(relevant.Variable =
  1:ncol(feat), ntree = 500, replace = TRUE, sample.fraction = 0.5, mtry =
  ncol(feat), nodesizeSpl = 11, nodesizeAvg = 33, splitratio = 0.5,
  middleSplit = FALSE))

X_BART(feat, tr, yobs, predmode = "pscore", nthread = 1,
  ndpost = 1200, ntree = 200, mu.BART = list(sparse = FALSE, theta =
  0, omega = 1, a = 0.5, b = 1, augment = FALSE, rho = NULL, usequants =
  FALSE, cont = FALSE, sigest = NA, sigdf = 3, sigquant = 0.9, k = 2, power
  = 2, base = 0.95, sigmaf = NA, lambda = NA, numcut = 100L, nskip = 100L),
  tau.BART = list(sparse = FALSE, theta = 0, omega = 1, a = 0.5, b = 1,
  augment = FALSE, rho = NULL, usequants = FALSE, cont = FALSE, sigest =
  NA, sigdf = 3, sigquant = 0.9, k = 2, power = 2, base = 0.95, sigmaf =
  NA, lambda = NA, numcut = 100L, nskip = 100L), e.BART = list(sparse =
  FALSE, theta = 0, omega = 1, a = 0.5, b = 1, augment = FALSE, rho = NULL,
  usequants = FALSE, cont = FALSE, sigest = NA, sigdf = 3, sigquant = 0.9,
  k = 2, power = 2, base = 0.95, sigmaf = NA, lambda = NA, numcut = 100L,
  nskip = 100L))
```

Arguments

`feat`	A data frame containing the features.
`tr`	A numeric vector with 0 for control and 1 for treated variables.
`yobs`	A numeric vector containing the observed outcomes.
`predmode`	Specifies how the two estimators of the second stage should be aggregated. Possible types are "propmean," "control," and "treated." The default is "propmean," which refers to propensity score weighting.

`nthread` Number of threads which should be used to work in parallel.

`verbose` TRUE for detailed output, FALSE for no output.

`mu.forestry, tau.forestry, e.forestry`
A list containing the hyperparameters for the `forestry` package that are used for estimating the response functions, the CATE, and the propensity score. These hyperparameters are passed to the `forestry` package. (Please refer to the forestry[3] package for a more detailed documentation of the hyperparamters.)

- `relevant.Variable` Variables that are only used in the first stage.
- `ntree` Numbers of trees used in the first stage.
- `replace` Sample with or without replacement in the first stage.
- `sample.fraction` The size of total samples to draw for the training data in the first stage.
- `mtry` The number of variables randomly selected in each splitting point.
- `nodesizeSpl` Minimum nodesize in the first stage for the observations in the splitting set. (See the details of the `forestry` package)
- `nodesizeAvg` Minimum nodesize in the first stage for the observations in the averaging set.
- `splitratio` Proportion of the training data used as the splitting dataset in the first stage.
- `middleSplit` If true, the split value will be exactly in the middle of two observations. Otherwise, it will take a point based on a uniform distribution between the two observations.

`ndpost` Number of posterior draws.

`ntree` Number of trees.

`mu.BART, tau.BART, e.BART`
hyperparameters of the BART functions for the estimates of the first and second stage and the propensity score. Use `?BART::mc.wbart` for a detailed explanation of their effects.

Details

The X-Learner estimates the CATE in three steps:

1. Estimate the response functions
$$\mu_0(x) = E[Y(0)|X = x]$$
$$\mu_1(x) = E[Y(1)|X = x]$$
using the base learner and denote the estimates as $\hat{\mu}_0$ and $\hat{\mu}_1$.
2. Impute the treatment effects for the individuals in the treated group, based on the control outcome estimator, and the treatment effects for the individuals in the control group, based on the treatment outcome estimator, that is,
$$D_i^1 = Y_i(1) - \hat{\mu}_0(X_i)$$
$$D_i^0 = \hat{\mu}_1(X_i) - Y_i(0).$$
Now employ the base learner in two ways: using D_i^1 as the dependent variable to obtain $\hat{\tau}_1(x)$, and using D_i^0 as the dependent variable to obtain $\hat{\tau}_0(x)$.
3. Define the CATE estimate by a weighted average of the two estimates at Stage 2:
$$\tau(x) = g(x)\hat{\tau}_0(x) + (1 - g(x))\hat{\tau}_1(x).$$
If `predmode = "propmean"`, then $g(x) = e(x)$, where $e(x)$ is an estimate of the propensity score using the `forestry` Random Forests version with the hyperparameters specified in `e.forestry`. If `predmode = "control"`, then $g(x) = 1$, and if `predmode = "treated"`, then $g(x) = 0$.

[3]`https://github.com/soerenkuenzel/forestry`

Value

An object from a class that contains the `CATEestimator` class. It should be used with one of the following functions: `EstimateCATE`, `CateCI`, and `CateBIAS`. The object has at least the following slots:

`feature_train`	A copy of feat.
`tr_train`	A copy of tr.
`yobs_train`	A copy of yobs.
`creator`	Function call that creates the CATE estimator. This is used for different bootstrap procedures.

Author(s)

Soeren R. Kuenzel

References

- Sören Künzel, Jasjeet Sekhon, Peter Bickel, and Bin Yu (2017). MetaLearners for Estimating Heterogeneous Treatment Effects using Machine Learning. `https://www.pnas.org/content/116/10/4156`
- Sören Künzel, Simon Walter, and Jasjeet Sekhon (2018). Causaltoolbox—Estimator Stability for Heterogeneous Treatment Effects. `https://arxiv.org/pdf/1811.02833.pdf`
- Sören Künzel, Bradly Stadie, Nikita Vemuri, Varsha Ramakrishnan, Jasjeet Sekhon, and Pieter Abbeel (2018). Transfer Learning for Estimating Causal Effects using Neural Networks. `https://arxiv.org/pdf/1808.07804.pdf`

See Also

Other metalearners: `M-Learner`, `S-Learner`, `T-Learner`

Examples

```
require(causalToolbox)

# create example data set
simulated_experiment <- simulate_causal_experiment(
  ntrain = 1000,
  ntest = 1000,
  dim = 10
)
feat <- simulated_experiment$feat_tr
tr <- simulated_experiment$W_tr
yobs <- simulated_experiment$Yobs_tr
feature_test <- simulated_experiment$feat_te

# create the CATE estimator using Random Forests (RF)
xl_rf <- X_RF(feat = feat, tr = tr, yobs = yobs)
tl_rf <- T_RF(feat = feat, tr = tr, yobs = yobs)
sl_rf <- S_RF(feat = feat, tr = tr, yobs = yobs)
ml_rf <- M_RF(feat = feat, tr = tr, yobs = yobs)
xl_bt <- X_BART(feat = feat, tr = tr, yobs = yobs)
tl_bt <- T_BART(feat = feat, tr = tr, yobs = yobs)
sl_bt <- S_BART(feat = feat, tr = tr, yobs = yobs)
ml_bt <- M_BART(feat = feat, tr = tr, yobs = yobs)

cate_esti_xrf <- EstimateCate(xl_rf, feature_test)

# evaluate the performance.
cate_true <- simulated_experiment$tau_te
mean((cate_esti_xrf - cate_true) ^ 2)
## Not run:
# create confidence intervals via bootstrapping.
xl_ci_rf <- CateCI(xl_rf, feature_test, B = 500)

## End(Not run)
```

Appendix C

Supporting Information for the Detachment Index

C.1 Example A

Days Ago C.	Registration D.	Horse Power	Kilometer Dr.	ZIP Code	RMSE
50.7 (100%)	9.3 (98.8%)	37.1 (96.1%)	47155.5 (100%)	30653.6 (100%)	2981.65
137.1 (100%)	9.2 (98.8%)	38.1 (96.3%)	47496.9 (100%)	30915.4 (100%)	2984.89
50.6 (100%)	7.7 (98.1%)	41.5 (97.2%)	34628.5 (99.5%)	34944.2 (100%)	2988.02
58.5 (100%)	7.3 (98%)	40.8 (97.1%)	52064.3 (100%)	31871.9 (100%)	2988.98
59.4 (100%)	7.9 (98.2%)	41.1 (97.1%)	36530.9 (99.6%)	32554.2 (100%)	2989.33
12 (99.8%)	9.3 (98.8%)	38.8 (96.6%)	53654.1 (100%)	30712.9 (100%)	2990.46
99.1 (100%)	9 (98.7%)	39 (96.6%)	53311.9 (100%)	28549.6 (99.8%)	2990.95
85.8 (100%)	7.8 (98.2%)	40.4 (97%)	45603.4 (99.9%)	32206.4 (100%)	2991.19
88.7 (100%)	9.4 (98.8%)	38.8 (96.5%)	50460.7 (100%)	30906.1 (100%)	2992.86
355.2 (100%)	8.4 (98.4%)	41.1 (97.1%)	36006.9 (99.5%)	33407.2 (100%)	3001.88
91.3 (100%)	6.3 (97.2%)	45.7 (97.9%)	49730.8 (100%)	32528.5 (100%)	3005.42
326.4 (100%)	6.3 (97.2%)	47.1 (98.1%)	52109 (100%)	28733.6 (99.8%)	3011.85
265.8 (100%)	9.7 (99%)	38.7 (96.5%)	39672 (99.7%)	29327.9 (99.9%)	3015.84
340.2 (100%)	9.5 (98.9%)	43.5 (97.6%)	53791.3 (100%)	24441.7 (98.2%)	3019.70
266.8 (100%)	6.9 (97.8%)	42.6 (97.5%)	53694.4 (100%)	30064 (99.9%)	3020.59
225.7 (100%)	8.6 (98.6%)	41.5 (97.2%)	44089.9 (99.9%)	33679.2 (100%)	3020.74
136.5 (100%)	9.9 (99%)	38.9 (96.6%)	42445.6 (99.8%)	32402.6 (100%)	3020.81
208.8 (100%)	6.9 (97.7%)	42.6 (97.5%)	52235.9 (100%)	30096.9 (99.9%)	3020.95
334.5 (100%)	7.7 (98.2%)	42.6 (97.5%)	38731.1 (99.7%)	32840.9 (100%)	3025.25
286.7 (100%)	7.1 (97.9%)	43.2 (97.6%)	36857.1 (99.6%)	31754.6 (100%)	3027.47
70.4 (100%)	9.9 (99%)	39.5 (96.8%)	38233.3 (99.7%)	34966.4 (100%)	3027.48
250.9 (100%)	5.8 (96.7%)	53.5 (98.6%)	37351.2 (99.6%)	34751.7 (100%)	3027.65
230.8 (100%)	9 (98.7%)	40.4 (97%)	38796.7 (99.7%)	34448.2 (100%)	3028.56
46.4 (100%)	8.9 (98.7%)	41.2 (97.1%)	36460.4 (99.6%)	27232.1 (99.5%)	3029.19
43.2 (100%)	9.2 (98.8%)	40.6 (97%)	37601.4 (99.7%)	34544.7 (100%)	3029.54
238.3 (100%)	8.1 (98.3%)	42.1 (97.3%)	33848 (99.4%)	28189.1 (99.8%)	3030.58
228.9 (100%)	8.6 (98.6%)	52.1 (98.5%)	47264 (100%)	24026.8 (97.8%)	3031.89
48.4 (100%)	9.3 (98.8%)	41.1 (97.1%)	39085.9 (99.7%)	27900.2 (99.7%)	3032.03
168.4 (100%)	8.9 (98.7%)	41.5 (97.2%)	41260.6 (99.8%)	27938.6 (99.7%)	3033.21
186.8 (100%)	6.3 (97.2%)	50.3 (98.4%)	46097.4 (99.9%)	30439.7 (100%)	3034.59
10.5 (99.7%)	9.5 (98.9%)	46.7 (98%)	51111.4 (100%)	24438.6 (98.2%)	3037.74
128.6 (100%)	5.7 (96.6%)	54.5 (98.7%)	49506.9 (100%)	32141.3 (100%)	3039.64
93.9 (100%)	6.1 (97.1%)	51.9 (98.5%)	48448.1 (100%)	32384 (100%)	3039.88
273.8 (100%)	7.9 (98.2%)	44.9 (97.8%)	34257.3 (99.4%)	33468.2 (100%)	3042.80
344.2 (100%)	7.3 (98%)	45.6 (97.9%)	35746.6 (99.5%)	34831.1 (100%)	3043.96

Table C.1: Best 40 thresholds out of a sample of 100,000. The numbers in the brackets are the corresponding percentiles.

C.2 Example B

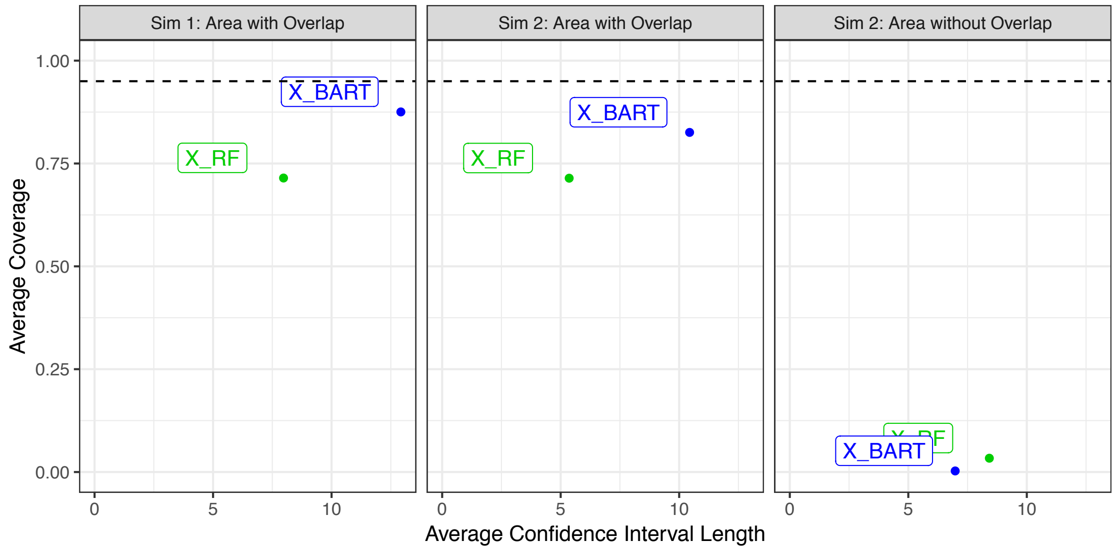


Figure C.1: Confidence intervals and confidence interval lengths for Simulations 1 and 2 seperated by the Overlap regions.

Appendix D

Supporting Information for Linear Aggregation in Tree-based Estimators

D.1 Splitting on a Categorical Feature

To split on a categorical feature, we use one-hot encoding to split based on equal or not equal to the given feature category. In order to evaluate the split RSS, we examine linear models fit on the set of observations containing the feature and the set not containing the feature. In order to evaluate this split quickly, we make use of the fact that we can quickly compute RSS components by keeping track of the total aggregated sum of outer products.

$$\text{Let}\ \ G_{\text{Total}} = \textstyle\sum_{i=1}^{n} \begin{bmatrix} X_i \\ 1 \end{bmatrix} \begin{bmatrix} X_i^T & 1 \end{bmatrix}$$

$$G_{\text{LeftInitial}} = \begin{bmatrix} X_1 \\ 1 \end{bmatrix} \begin{bmatrix} X_1^T & 1 \end{bmatrix}$$

$$G_{\text{RightInitial}} = \textstyle\sum_{i=2}^{n} \begin{bmatrix} X_i \\ 1 \end{bmatrix} \begin{bmatrix} X_i^T & 1 \end{bmatrix} = G_{\text{Total}} - G_{\text{LeftInitial}}$$

This means that given G_{Total}, we can immediately begin evaluating splits along any split feature by using the sequence of indices corresponding to the ascending sequence of values in the current split feature.

This fact helps us quickly evaluate the sum of RSS of separate regressions on the two sides of inclusion/exclusion splits on categorical variables.

On a split on categorical variable $X(l)$, when evaluating the split RSS on the value of category k, the RSS components can be calculated as follows:

$$\text{Let}\ E(k) = \{i : X_i(l) = k\}$$

$$G_{\text{Left}} = \textstyle\sum_{i \in E(k)} \begin{bmatrix} X_i \\ 1 \end{bmatrix} \begin{bmatrix} X_i^T & 1 \end{bmatrix} \in \mathbb{R}^{d \times d}$$

$$G_{\text{Right}} = \sum_{i \notin E(k)} \begin{bmatrix} X_i \\ 1 \end{bmatrix} \begin{bmatrix} X_i^T & 1 \end{bmatrix} \in \mathbb{R}^{d \times d}$$

$$G_{\text{Right}} = G_{\text{Total}} - G_{\text{Left}}$$

The same method can be used to update S_{Left} and S_{Right} at each step, so we can use this update rule generally to quickly compute the RSS for categorical splits in the algorithm that follows.

Algorithm 14 Find Best Split for Categorical Features

```
    Input: Features: $X \in \mathbb{R}^{n \times d}$,
           Dependent Outcome: $Y \in \mathbb{R}^n$,
           overfitPenalty (regularization for split): $\lambda \in \mathbb{R}^+$,
    Output: Best Split point k
 1: procedure FindBestSplitRidgeCategorical

    Initialization:
 2:     $G_{Total} = \sum_{i=1}^n \begin{bmatrix} X_i \\ 1 \end{bmatrix} \begin{bmatrix} X_i^T & 1 \end{bmatrix}$
 3:     $S_{Total} \leftarrow \sum_{i=1}^n Y_i \begin{bmatrix} X_i \\ 1 \end{bmatrix}$
 4:     for $category\_k = 1, \dots, l$ do
 5:         $E(k) \leftarrow \{i : X_i(splitFeat) = category\_k\}$
 6:         $G_L \leftarrow \sum_{i \in E(k)} \begin{bmatrix} X_i \\ 1 \end{bmatrix} \begin{bmatrix} X_i^T & 1 \end{bmatrix}$
 7:         $S_L \leftarrow \sum_{i \in E(k)} Y_i \begin{bmatrix} X_i \\ 1 \end{bmatrix}$
 8:         $G_R \leftarrow G_{Total} - G_L$
 9:         $S_R \leftarrow S_{Total} - S_L$
10:         $A_R^{-1} \leftarrow \left( G_R + \lambda \begin{bmatrix} \mathcal{I}_d & 0 \\ 0 & 0 \end{bmatrix} \right)^{-1}$
11:         $A_L^{-1} \leftarrow \left( G_L + \lambda \begin{bmatrix} \mathcal{I}_d & 0 \\ 0 & 0 \end{bmatrix} \right)^{-1}$
12:         Compute the RSS sum for the current split:
                $\text{RSS}_k \leftarrow S_L^T A_L^{-1} G_L A_L^{-1} S_L - 2 S_L^T A_L^{-1} S_L)$
                    $+ S_R^T A_R^{-1} G_R) A_R^{-1} S_R - 2 S_R^T A_R^{-1} S_R.$

13:     return $(argmin_{(k)} \text{RSS}_k)$
```

Runtime Analysis of Finding Categorical Split Point

Using a set to keep track of category membership, we can create the set in $O(NlogN)$ time and access a member of any specific category in amortized constant time. Once we begin iterating through the categories, we can access the elements and create the left model RSS components in $O(|K|)$ where $|K|$ is the size of category k, and using the G_{total} and S_{total} matrices, we can find the right model RSS components in $O(d^2)$ time once we calculate G_L and S_L. As the sum of sizes of the various categories sums to the number of observations, we end up doing the same number of RSS component update steps as the continuous case as well as one additional step to get the right RSS components for each category. The overall asymptotic runtime remains $O(NlogN + Nd^2)$.

D.2 Tuned Simulation Hyperparameters

Dataset	mtry	nodesize	overfitP	LOGminSplitGain	sample.frac
Friedman 1	9	16	0.23	-3.86	0.91
Friedman 2	3	195	0.43	-5.07	0.89
Friedman 3	4	11	6.65	-3.16	0.65
Boston Housing f1	10	7	0.28	-7.86	0.95
Boston Housing f2	4	13	3.06	-4.81	0.99
Boston Housing f3	3	12	0.19	-4.94	0.94
Boston Housing f4	5	13	0.77	-9.12	0.91
Boston Housing f5	2	11	0.25	-13.71	0.99
Ozone fold1	2	19	9.47	-6.41	0.5
Ozone fold2	3	12	3.06	-4.81	0.99
Ozone fold3	1	20	7.4	-10.41	0.9
Ozone fold4	3	19	9.36	-4.76	0.92
Ozone fold5	2	3	8.51	-7.12	0.88
Servo fold1	12	5	0.31	-2.83	0.89
Servo fold2	9	16	0.11	-6.78	0.97
Servo fold3	11	2	0.87	-8.84	0.97
Servo fold4	11	34	0.12	-3.22	0.87
Servo fold5	11	33	0.12	-3.22	0.87
Abalone	1	150	0.13	-6.25	0.92
autos	5	18	0.8	-8.44	0.92
bike	8	23	0.11	-6.78	0.97
artificial LM 256	3	50	5.57	-8.71	0.52
artificial LM 512	2	16	0.19	-2.78	0.51
artificial LM 1024	4	3	0.18	-2.82	0.63
artificial LM 2048	9	17	0.23	-3.86	0.91
Step 128	8	9	9.29	-8.39	0.92
Step 256	9	30	0.3	-7.36	0.77
Step 512	8	47	0.28	-12.75	0.89
Step 1024	5	27	0.31	-18.42	0.73
StepLinear 256	10	10	8.74	-3	0.92
StepLinear 512	10	11	8.74	-3	0.92
StepLinear 1024	10	12	8.74	-3	0.92

Table D.1: Selected hyperparameters.

D.3 Generating Random Step Function

The Simulated-Step-Function in Section 5.3 was generated according to the following scheme:

Algorithm 15 Generate Simulated Step

```
Input: numLevels (number of random levels for step function),
       n (dimension of data)
Output: Independent Input: $X \in \mathbb{R}^{n \times d}$,
        Step Function Outcome: $Y_{step} \in \mathbb{R}^{n}$,
1: procedure GENERATE_SIMULATED_STEP
2:     $X_{i=1}^{n} \leftarrow Normal(0,1)^{10}$
3:
4:     $Y_{i=1}^{numLevels} \leftarrow Unif(-10,10)$
5:
6:     $X_{sample} \leftarrow X[sample(1 : nrow(X), numLevels, replace = FALSE),]$
7:
8:     $f_s \leftarrow forestry(x = X_{sample}, y = Y, nodeSize = 1)$
9:
10:    $Y_{step} \leftarrow predict(f_s, feature.new = X)$
11:
12:    return $(X, Y_{step})$
```

Appendix E

`forestry` Documentation

`forestry` is a C++ implementation of the random forests algorithm [6] with new tools to do better inference and to improve the interpretability of the algorithm. The software is written in C++ so that the algorithm can run extremely fast, even on big data sets. It also provides an R front end so that appliers can easily use it in their usual workflows.

`addTrees` *addTrees-forestry*

Description

Add more trees to an existing forest.

Usage

```
addTrees(object, ntree)
```

Arguments

`object`	A trained model object of class ”forestry”.
`ntree`	Number of new trees to add.

Value

A trained model object of class ”forestry”.

`compute_lp-forestry` *compute lp distances*

Description

Return lp distances of selected test observations.

Usage

```
compute_lp(object, feature.new, feature, p)
```

Arguments

`object`	A trained model object of class ”forestry”.
`feature.new`	A data frame of testing predictors.
`feature`	A string denoting the dimension for computing lp distances.
`p`	A positive real number determining the norm p-norm used.

Value

A vector lp distances.

Examples

```
# Set seed for reproductivity
set.seed(292313)

# Use Iris Data
test_idx <- sample(nrow(iris), 11)
x_train <- iris[-test_idx, -1]
y_train <- iris[-test_idx, 1]
x_test <- iris[test_idx, -1]

rf <- forestry(x = x_train, y = y_train)
predict(rf, x_test)

# Compute the l2 distances in the "Petal.Length" dimension
distances_2 <- compute_lp(object = rf,
                          feature.new = x_test,
                          feature = "Petal.Length",
                          p = 2)
```

`forestry` *forestry*

Description

forestry is a fast implementation of a variety of tree-based estimators. Implemented estimators include CART trees, randoms forests, boosted trees and forests, and linear trees and forests. All estimators are implemented to scale well with very large datasets.

Usage

```
forestry(x, y, ntree = 500, replace = TRUE, sampsize = if (replace)
  nrow(x) else ceiling(0.632 * nrow(x)), sample.fraction = NULL,
  mtry = max(floor(ncol(x)/3), 1), nodesizeSpl = 3, nodesizeAvg = 3,
  nodesizeStrictSpl = 1, nodesizeStrictAvg = 1, minSplitGain = 0,
  maxDepth = round(nrow(x)/2) + 1, splitratio = 1,
  seed = as.integer(runif(1) * 1000), verbose = FALSE, nthread = 0,
  splitrule = "variance", middleSplit = FALSE, maxObs = length(y),
  linear = FALSE, splitFeats = 1:(ncol(x)), linFeats = 1:(ncol(x)),
  sampleWeights = rep((1/ncol(x)), ncol(x)), overfitPenalty = 1,
  doubleTree = FALSE, reuseforestry = NULL, saveable = TRUE)
```

Arguments

`x`	A data frame of all training predictors.
`y`	A vector of all training responses.
`ntree`	The number of trees to grow in the forest. The default value is 500.
`replace`	An indicator of whether sampling of training data is with replacement. The default value is TRUE.
`sampsize`	The size of total samples to draw for the training data. If sampling with replacement, the default value is the length of the training data. If samplying without replacement, the default value is two-third of the length of the training data.
`sample.fraction`	If this is given, then sampsize is ignored and set to be round(length(y) * sample.fraction). It must be a real number between 0 and 1
`mtry`	The number of variables randomly selected at each split point. The default value is set to be one third of total number of features of the training data.
`nodesizeSpl`	Minimum observations contained in terminal nodes. The default value is 3.
`nodesizeAvg`	Minimum size of terminal nodes for averaging dataset. The default value is 3.
`nodesizeStrictSpl`	Minimum observations to follow strictly in terminal nodes. The default value is 1.
`nodesizeStrictAvg`	Minimum size of terminal nodes for averaging dataset to follow strictly. The default value is 1.
`minSplitGain`	Minimum loss reduction to split a node further in a tree. specifically this is the percentage R squared increase which each potential split must give to be considered. The default value is 0.
`maxDepth`	Maximum depth of a tree. The default value is 99.
`splitratio`	Proportion of the training data used as the splitting dataset. It is a ratio between 0 and 1. If the ratio is 1, then essentially splitting dataset becomes the total entire sampled set and the averaging dataset is empty. If the ratio is 0, then the splitting data set is empty and all the data is used for the averaging data set (This is not a good usage however since there will be no data available for splitting).
`seed`	Seed for random number generator.
`verbose`	Flag to indicate if training process is verbose.

`nthread`	Number of threads to train and predict the forest. The default number is 0 which represents using all cores.
`splitrule`	Only variance is implemented at this point and it specifies the loss function according to which the splits of random forest should be made.
`middleSplit`	Flag to indicate whether the split value takes the average of two feature values. If false, it will take a point based on a uniform distribution between two feature values. The default value is FALSE.
`maxObs`	The max number of observations to split on. If set to a number less than nrow(x), at each split point, maxObs split points will be randomly sampled to test as potential splitting points instead of every feature value (default).
`linear`	Fit the model with a split function optimizing for a linear aggregation function instead of a constant aggregation function. The default value is FALSE.
`splitFeats`	Specify which features to split on when creating a tree (defaults to use all features).
`linFeats`	Specify which features to split linearly on when using linear (defaults to use all numerical features)
`sampleWeights`	Specify weights for weighted uniform distribution used to randomly sample features.
`overfitPenalty`	Value to determine how much to penalize magnitude of coefficients in ridge regression when using linear. The default value is 1.
`doubleTree`	Indicate whether the number of tree is doubled as averaging and splitting data can be exchanged to create decorrelated trees. The default value is FALSE.
`reuseforestry`	Pass in a 'forestry' object which will recycle the dataframe the old object created. It will save some space working on the same dataset.
`saveable`	If TRUE, then RF is created in such a way that it can be saved and loaded using save(...) and load(...). Setting it to TRUE (default) will, however, take longer and it will use more memory. When training many RF, it makes a lot of sense to set this to FALSE to save time and memory.

Details

For Linear Random Forests, set the linear option to TRUE and specify lambda for ridge regression with overfitPenalty parameter. For gradient boosting and gradient boosting forests, see mulitlayer-forestry.

Value

A 'forestry' object.

See Also

`predict.forestry`

`multilayer-forestry`

`predict-multilayer-forestry`

`getVI`

`getOOB`

`make_savable`

Examples

```
set.seed(292315)
library(forestry)
test_idx <- sample(nrow(iris), 3)
x_train <- iris[-test_idx, -1]
y_train <- iris[-test_idx, 1]
x_test <- iris[test_idx, -1]

rf <- forestry(x = x_train, y = y_train)
weights = predict(rf, x_test, aggregation = "weightMatrix")$weightMatrix

weights %*% y_train
```

```
predict(rf, x_test)

set.seed(49)
library(forestry)

n <- c(100)
a <- rnorm(n)
b <- rnorm(n)
c <- rnorm(n)
y <- 4*a + 5.5*b - .78*c
x <- data.frame(a,b,c)

forest <- forestry(
          x,
          y,
          ntree = 10,
          replace = TRUE,
          nodesizeStrictSpl = 5,
          nodesizeStrictAvg = 5,
          linear = TRUE
          )

predict(forest, x)
```

`getOOB-forestry` *getOOB-forestry*

Description

Calculate the out-of-bag error of a given forest.

Usage

```
getOOB(object, noWarning)
```

Arguments

`object`	A trained model object of class "forestry".
`noWarning`	Flag to not display warnings.

Value

The out-of-bag error of the forest.

See Also

`forestry`

`getVI` *getVI-forestry*

Description

Calculate variable importance for ‘forestry‘ object as introduced by Breiman (2001). Returns a list of percentage increases in out-of-bag error when shuffling each feature values and getting out-of-bag error.

Usage

```
getVI(object, noWarning = FALSE)
```

Arguments

`object`	A trained model object of class ”forestry”.
`noWarning`	Flag to not display warnings or display warnings.

See Also

`forestry`

`make_savable` *make_savable*

Description

When a ‘foresty‘ object is saved and then reloaded ,the Cpp pointers for the data set and the Cpp forest have to be reconstructed.

Usage

```
make_savable(object)
```

Arguments

`object`	A trained model object of class ”forestry”.

Value

A list of lists. Each sublist contains the information to span a tree.

See Also

`forestry`

Examples

```
set.seed(323652639)
x <- iris[, -1]
y <- iris[, 1]
forest <- forestry(x, y, ntree = 3)
y_pred_before <- predict(forest, x)

forest <- make_savable(forest)
save(forest, file = "forest.Rda")
rm(forest)
load("forest.Rda", verbose = FALSE)
forest <- relinkCPP_prt(forest)

y_pred_after <- predict(forest, x)
testthat::expect_equal(y_pred_before, y_pred_after, tolerance = 0.000001)
file.remove("forest.Rda")
```

`multilayer-forestry` *Multilayer forestry*

Description

Constructs a gradient boosted random forest.

Usage

```
multilayerForestry(x, y, ntree = 500, nrounds = 1, eta = 0.3,
  replace = FALSE, sampsize = nrow(x), sample.fraction = NULL,
  mtry = ncol(x), nodesizeSpl = 3, nodesizeAvg = 3,
  nodesizeStrictSpl = max(round(nrow(x)/128), 1),
  nodesizeStrictAvg = max(round(nrow(x)/128), 1), minSplitGain = 0,
  maxDepth = 99, splitratio = 1, seed = as.integer(runif(1) * 1000),
  verbose = FALSE, nthread = 0, splitrule = "variance",
  middleSplit = TRUE, maxObs = length(y), linear = FALSE,
  splitFeats = 1:(ncol(x)), linFeats = 1:(ncol(x)),
  sampleWeights = rep((1/ncol(x)), ncol(x)), overfitPenalty = 1,
  doubleTree = FALSE, reuseforestry = NULL, saveable = TRUE)
```

Arguments

`x`	A data frame of all training predictors.
`y`	A vector of all training responses.
`ntree`	The number of trees to grow in the forest. The default value is 500.
`nrounds`	Number of iterations used for gradient boosting.
`eta`	Step size shrinkage used in gradient boosting update.
`replace`	An indicator of whether sampling of training data is with replacement. The default value is TRUE.
`sampsize`	The size of total samples to draw for the training data. If sampling with replacement, the default value is the length of the training data. If samplying without replacement, the default value is two-third of the length of the training data.
`sample.fraction`	If this is given, then sampsize is ignored and set to be round(length(y) * sample.fraction). It must be a real number between 0 and 1
`mtry`	The number of variables randomly selected at each split point. The default value is set to be one third of total number of features of the training data.
`nodesizeSpl`	Minimum observations contained in terminal nodes. The default value is 3.
`nodesizeAvg`	Minimum size of terminal nodes for averaging dataset. The default value is 3.
`nodesizeStrictSpl`	Minimum observations to follow strictly in terminal nodes. The default value is 1.
`nodesizeStrictAvg`	Minimum size of terminal nodes for averaging dataset to follow strictly. The default value is 1.
`minSplitGain`	Minimum loss reduction to split a node further in a tree. specifically this is the percentage R squared increase which each potential split must give to be considered. The default value is 0.
`maxDepth`	Maximum depth of a tree. The default value is 99.
`splitratio`	Proportion of the training data used as the splitting dataset. It is a ratio between 0 and 1. If the ratio is 1, then essentially splitting dataset becomes the total entire sampled set and the averaging dataset is empty. If the ratio is 0, then the splitting data set is empty and all the data is used for the averaging data set (This is not a good usage however since there will be no data available for splitting).
`seed`	Seed for random number generator.
`verbose`	Flag to indicate if training process is verbose.

`nthread`	Number of threads to train and predict the forest. The default number is 0 which represents using all cores.
`splitrule`	Only variance is implemented at this point and it specifies the loss function according to which the splits of random forest should be made.
`middleSplit`	Flag to indicate whether the split value takes the average of two feature values. If false, it will take a point based on a uniform distribution between two feature values. The default value is FALSE.
`maxObs`	The max number of observations to split on. If set to a number less than nrow(x), at each split point, maxObs split points will be randomly sampled to test as potential splitting points instead of every feature value (default).
`linear`	Fit the model with a split function optimizing for a linear aggregation function instead of a constant aggregation function. The default value is FALSE.
`splitFeats`	Specify which features to split on when creating a tree (defaults to use all features).
`linFeats`	Specify which features to split linearly on when using linear (defaults to use all numerical features)
`sampleWeights`	Specify weights for weighted uniform distribution used to randomly sample features.
`overfitPenalty`	Value to determine how much to penalize magnitude of coefficients in ridge regression when using linear. The default value is 1.
`doubleTree`	Indicate whether the number of tree is doubled as averaging and splitting data can be exchanged to create decorrelated trees. The default value is FALSE.
`reuseforestry`	Pass in a 'forestry' object which will recycle the dataframe the old object created. It will save some space working on the same dataset.
`saveable`	If TRUE, then RF is created in such a way that it can be saved and loaded using save(...) and load(...). Setting it to TRUE (default) will, however, take longer and it will use more memory. When training many RF, it makes a lot of sense to set this to FALSE to save time and memory.

Value

A trained model object of class "multilayerForestry".

See Also

`forestry`

`plot-forestry` *visualize a tree*

Description

Plots a single tree in the forest.

Usage

```
## S3 method for class 'forestry'
plot(x, tree.id = 1, print.meta_dta = FALSE,
  beta.char.len = 30, return.plot.dta = FALSE, ...)
```

Arguments

`x`	A trained model object of class "forestry".
`tree.id`	Specifies the tree number that should be visualized
`print.meta_dta`	Should the data for the plot be printed?
`beta.char.len`	The length of the beta values in leaf node representation.
`return.plot.dta`	If TRUE no plot will be generated, but instead a list with all the plot data is returned.
`...`	additional arguments that are not used.

Examples

```
set.seed(292315)
rf <- forestry(x = iris[,-1],
               y = iris[, 1])

plot(x = rf)
plot(x = rf, tree.id = 2)
plot(x = rf, tree.id = 500)

ridge_rf <- forestry(
  x = iris[,-1],
  y = iris[, 1],
  replace = FALSE,
  nodesizeStrictSpl = 10,
  mtry = 4,
  ntree = 1000,
  minSplitGain = .004,
  linear = TRUE,
  overfitPenalty = 1.65,
  linFeats = 1:2)

plot(x = ridge_rf)
plot(x = ridge_rf, tree.id = 2)
plot(x = ridge_rf, tree.id = 1000)
```

`predict-forestry` *predict-forestry*

Description

Return the prediction from the forest.

Usage

```
## S3 method for class 'forestry'
predict(object, feature.new, aggregation = "average",
  localVariableImportance = FALSE, ...)
```

Arguments

`object`	A trained model object of class ”forestry”.
`feature.new`	A data frame of testing predictors.
`aggregation`	How shall the leaf be aggregated. The default is to return the mean of the leave ‘average‘. Other options are ‘weightMatrix‘.
`localVariableImportance`	Returns a matrix providing local variable importance for each prediction.
`...`	additional arguments.

Details

Allows for different methods of prediction on new data.

Value

A vector of predicted responses.

See Also

`forestry`

`predict-multilayer-forestry` *predict-multilayer-forestry*

Description

Return the prediction from the forest.

Usage

```
## S3 method for class 'multilayerForestry'
predict(object, feature.new,
  aggregation = "average", ...)
```

Arguments

`object`	A ‘multilayerForestry‘ object.
`feature.new`	A data frame of testing predictors.
`aggregation`	How shall the leaf be aggregated. The default is to return the mean of the leave ‘average‘. Other options are ‘weightMatrix‘.
`...`	additional arguments.

Value

A vector of predicted responses.

See Also

`forestry`

`relinkCPP_prt` *relink CPP ptr*

Description

When a ‘foresty‘ object is saved and then reloaded the Cpp pointers for the data set and the Cpp forest have to be reconstructed

Usage

```
relinkCPP_prt(object)
```

Arguments

`object`	an object of class ‘forestry‘