UNIVERSITY OF COPENHAGEN
FACULTY OF SCIENCE

A Bachelor of Science thesis

Super Learners

and their oracle properties

Jinyang Liu

Supervised by Prof. Thomas Gerds
Co-supervised by Prof. Niels Richard Hansen
Department of Mathematical Sciences
University of Copenhagen, Denmark

Submitted: August 18, 2023

Dedicated to p. and the others on Mount Olympus.

Abstract

In this thesis we examine super learners and their applicability to binary regression. The super learner is a method for combining predictions from a specified library of learning algorithms to create a strong learner. We introduce and prove the oracle property for the discrete super learner, which is extended to the ensemble super learner. The oracle results show that given a library of learning algorithms, asymptotically, the super learner will not perform worse than the best algorithm in the library in terms of risk. We then compare the performance of the super learner with other regression algorithms including logistic regression and XGBoost on simulated data. The simulations demonstrate that the super learner achieves minimal risk as the number of observations grows. Finally, a new technique of combining learner predictions to be used by the ensemble super learner is proposed and has shown interesting results.

Acknowledgements

I would like to thank my supervisor, Thomas Gerds, for suggesting the topic to me and lending me a desk at the section of Biostatistics while I wrote my thesis. I would also like to thank Anders Munch for the useful discussions we've had regarding oracle bounds for the super learner. I am furthermore grateful to my supervisor, Niels Richard Hansen, who inspired me to be mathematically rigorous, and whom I have continuously conversed with on my journey to becoming a statistician.

At last, special thanks to my friend, Marius Kjærsgaard, who explained the Ehrhart polynomial to me in conjunction with the selection of lattice points that grow polynomial in n on the $(k-1)$-simplex, but which unfortunately there was not enough space to include in my thesis.

Code availability

The code for the simulations conducted in section 5 can be found on GitHub at Liu (2023).

Contents

1 Introduction

In the context of regression, $O = (Y, X)$ is an observation for $Y \in \mathbb{R}$ being the outcome and $X \in \mathbb{R}^d$ the covariates. A natural goal is to estimate a function θ such that the mean squared error $E(Y - \theta(X))^2$ is minimized. It turns out that the conditional mean – called the regression function or regression $x \mapsto E(Y \mid X = x)$ – minimizes the squared error, but consistent estimation of the regression requires a statistical model $\mathcal{P}$ for the data-generating distribution $P \in \mathcal{P}$ to be specified. We typically make certain assumptions about the statistical model, $\mathcal{P}$, in which we believe P resides. For instance, we might assume that $\mathcal{P}$ is a parametric family of distributions such as a curved exponential family (Lauritzen, 2023). In doing so, we can identify the parameters of the distribution P via maximum likelihood and estimate the regression from the parameters.

However, if we are dealing with complex data, there is a risk of misspecifying the model by identifying it as an exponential family. Our assumptions may be wrong. In these situations, it is tempting to utilize non-parametric and data-driven regression methods, such as tree-based algorithms including XGBoost or random forests. Machine learning methods seek to estimate the regression function directly, in contrast to parametric statistics where the parameters of the underlying model are estimated first, and a regression function is derived as some analytical expression of the parameters and covariates. The assumptions of these machine learning methods regarding the data-generating distribution are not explicitly specified, but it does not pose a problem for estimating the regression. Indeed, it is not necessary for us to identify P completely if P can be factored into the conditional distribution $P_{Y|X}$ and the distribution over our covariates P_X. Here our goal of estimating the regression function is to estimate $P_{Y|X}$.

Given an abundance of different ways we can tackle the problem of estimating the regression, we would like to be able to compare the different methods and select the best one. In a practical scenario we might have a set of learning algorithms that we can choose from. Cross-validation can help determine the risk of each algorithm by splitting the data into training and validation sets. Each algorithm is fitted on the training set, and a validation risk for each algorithm is calculated by evaluating the fitted models on the validation set. We would run cross-validation using a predefined splitting mechanism for each learning algorithm that we have. A popular choice in machine learning and statistics is K-fold cross-validation. K-fold cross-validation divides the data into K disjoint and exhaustive sets referred to as validation sets. For every $k = 1, \ldots, K$ the learning algorithm is trained using all data excluding the k'th validation set. Subsequently, the risk of the algorithm is computed by applying the fitted model on the validation set that was held out from training. We obtain an estimate of the true risk of the model by repeating this procedure K times and averaging over the risks. The model with the lowest risk is then selected as our candidate estimate of the regression function.

This is the idea behind the cross-validation selector (Laan and Dudoit, 2003). The discrete super learner (Laan, Polley, and Hubbard, 2007) is simply the learner (fitted model) that is obtained by applying the cross-validation selected algorithm to our data. The cross-validation selector, given a library of learning algorithms and our data, will cross-validate each algorithm in the library and select the one with the lowest risk.

Another method, called the ensemble super learner (Laan, Polley, and Hubbard, 2007) seeks to combine the learning algorithms into a single learner by applying a *meta learning algorithm* on the library. One meta algorithm of the ensemble super learner is to fit a weighted average of the learners, where the weights are chosen to minimize the risk of the ensemble. The prediction of the ensemble super learner will then be a weighted sum of

the predictions made by each learner.

As we will demonstrate, given a library of learning algorithms, the super learner is effectively the best estimate of the data-generating regression that we can obtain. It is the best in the sense that it cannot perform worse than the best learning algorithm in terms of risk.

To formalize the notion of the best learner in the library, we define the oracle selector. The oracle selector knows P – hence it is called the oracle – and selects the learner with the lowest risk given its knowledge of P. In this thesis we show that the cross-validation selector is asymptotically equivalent with the oracle selector as the sample size goes to infinity.

We adopt the setup and notation similar to those in Vaart, Dudoit, and Laan (2006) and Laan and Dudoit (2003). Our objective is to illustrate the performance of super learning algorithms when applied to binary regression, where we are regressing a binary outcome $Y \in \{0,1\}$ against covariates X that belong to $\mathcal{X} \subseteq \mathbb{R}^d$. Here the conditional expectation of Y given X exactly becomes the conditional probability $P(Y \mid X)$. The simulation results in Section 5 show that the super learner achieves the minimum risk with increasing training samples. The simulation is conducted by first defining the data-generating distribution explicitly, enabling us to use standard sampling methods that are available in R to draw samples from the distribution. We assess a library of learning algorithms consisting of logistic regression and XGBoost, from which we construct the super learner and evaluate its performance in relation to the algorithms in the library.

The choice to focus on binary regression stems from its significance in various fields. For instance, in biomedicine, researchers might want to predict patient mortality upon administering a specific drug. The survival indicator for the patient is a binary outcome, and the regression $x \mapsto P(Y = 1 \mid X = x)$ could represent the probability of the patient's survival.

2 Background

Our setup and notation is similar to Vaart, Dudoit, and Laan (2006) and Laan and Dudoit (2003): Let a statistical model $\mathcal{P}$ be given on the measurable space $(\mathcal{O}, \mathcal{A})$ where $\mathcal{O} = \{0,1\} \times \mathcal{X}$ is our sample space for some $\mathcal{X} \subseteq \mathbb{R}^d$. We will consider the parameter set $\Theta = \{\theta \mid \theta : \mathcal{X} \to [0,1] \text{ measurable}\}$, which represents the set of *regression functions* that map from our covariates to the probability interval. We define the quadratic loss and the corresponding risk that we wish to minimize.

Definition 1 (Quadratic loss)**.** The quadratic loss or L^2-loss, $L : \mathcal{O} \times \Theta \to [0, \infty)$, for an observation $(Y, X) = O \in \mathcal{O}$ and a regression function $\theta \in \Theta$ is defined as

$$L(O, \theta) = L((Y, X), \theta) = (Y - \theta(X))^2.$$

A natural aim would be to find the optimal parameter value $\theta^* \in \Theta$ that minimizes the expected quadratic loss, or risk $R : \theta \to \mathbb{R}$ given by

$$R(\theta, P) := \int L(O, \theta) dP(O). \tag{1}$$

Theorem 2 shows that the minimum risk is achieved by the conditional probability $x \mapsto P(Y = 1 \mid X = x)$; we also say that the quadratic loss is *strictly proper* (Gneiting and Raftery, 2007).

Theorem 2. *Let $(\mathcal{O}, \mathcal{A}, P)$ be a probability space for some probability measure $P \in \mathcal{P}$. Let Θ be the set of regression functions. Let the loss function be the quadratic loss $L(O, \theta) = (Y - \theta(X))^2$, then for the optimum θ^* defined as*

$$\theta^* := \arg\min_{\theta \in \Theta} R(\theta, P) = \arg\min_{\theta \in \Theta} \int L(O, \theta) dP(O),$$

it holds for an observation $O = (Y, X) \sim P$ that

$$\theta^*(x) = E(Y \mid X = x).$$

Proof. Proof emulated from (Györfi et al., 2002)[ch. 1]. Let $\theta \in \Theta$ be arbitrary and $m(x) = E(Y \mid X = x)$, we have

$$\begin{aligned} E(\theta(X) - Y)^2 &= E(\theta(X) - m(X) + m(X) - Y)^2 \\ &= E(\theta(X) - m(X))^2 + E(m(X) - Y)^2 + 2E[(\theta(X) - m(X))(m(X) - Y)]. \end{aligned}$$

The last term of the right hand side is zero. By using the tower rule we obtain

$$\begin{aligned} E[(\theta(X) - m(X))(m(X) - Y)] &= E[E[(\theta(X) - m(X))(m(X) - Y) \mid X]] \\ &= E[(\theta(X) - m(X))E((m(X) - Y) \mid X)] \\ &= E[(\theta(X) - m(X))(m(X) - E(Y \mid X))] \\ &= E[(\theta(X) - m(X))(m(X) - m(X))] \\ &= 0. \end{aligned}$$

We conclude that

$$\int L(O, \theta) dP(O) = E(\theta(X) - m(X))^2 + E(m(X) - Y)^2.$$

The first term on the right hand side is always positive and is 0 only when $\theta = m$, this proves that m minimizes the expression above. □

It follows immediately that if Y is binary, then $E(Y \mid X = x) = P(Y = 1 \mid X = x)$.

Our goal is to estimate θ^*, which means to *learn* the true regression function from our data. We will introduce the terminology *learning algorithm* and *learner* in the context of learning from our data.

2.1 Learning algorithms and learners

We will denote $O_1, \ldots, O_n \in \mathcal{O}$ and $D_n = (O_1, \ldots, O_n)$ as our observations and data respectively.

Definition 3 (Learning algorithm)**.** A learning algorithm is a measurable map $\psi : \mathcal{O}^n \to \Theta$ for $n \in \mathbb{N}$.

We will throughout assume that the learning algorithm is well defined for each $n \in \mathbb{N}$, and that permuting the observations has no effect on the outcome, i.e., the algorithm is symmetric in the observations.

Definition 4 (Learner or fitted learner)**.** Given a learning algorithm ψ, a learner is a stochastic variable in Θ representing the outcome of applying the learning algorithm to our data $\theta = \psi(D_n)$.

Formally $\psi(D_n)$ is a stochastic variable since D_n is stochastic. In practice, we would observe $O_1 = o_1, \ldots, O_n = o_n$, following which we can employ a learning algorithm on the observed data. In machine learning, the process of applying an algorithm on the data is referred to as *model training* or *fitting*, the outcome of the training is a *trained model* or simply a *fit*. We will refer to the trained model as a learner which naturally depends on the observed data.

Example 1 (Parametric and nonparametric learning algorithms). An example of a parametric learning algorithm is logistic regression. In logistic regression we assume that the conditional probability, $P(Y = 1 \mid X = x)$, can be expressed as $\theta(x) = \text{expit}(\beta x)$ for some $\beta \in \mathbb{R}^d$ which is estimated via maximum likelihood.

Nonparametric learning algorithms such as gradient boosting – most notably XGBoost – can also be used to estimate the regression function. XGBoost has a number of hyperparameters that can be tuned. These parameters include the number of boosted trees, depth of each tree, learning rates, and others. However, the most critical parameter is the internal loss objective which could for example be log-loss or mean squared error. XGBoost aims to iteratively refine the fitted learner by approximating the data $x \mapsto f_m(x)$ at each step m. It does so by introducing a new tree $h_m(x)$, which is trained on the error of $f_m(x)$, such that $f_{m+1}(x) = f_m(x) + h_m(x)$. The internal loss of the updated learner, f_{m+1}, evaluated on the training data, is lower than that of the previous learner due to the inclusion of the new tree (Chen and Guestrin, 2016).

The parameters of the fitted XGBoost are not directly interpretable. Despite this, XGBoost has demonstrated its ability to model very complex datasets (Chen and Guestrin, 2016).

We will denote the empirical measure obtained from the data D_n as

$$P_n(A) = \frac{1}{n} \sum_{i=1}^{n} \delta_{O_i}(A) \qquad \text{for } A \in \mathcal{A},$$

where δ_{O_i} is the Dirac measure over O_i. When the observations are independent and identically distributed, then there is a one-to-one correspondence between the empirical measures obtained from n observations and D_n. We can, therefore, write $\psi(P_n)$ as an alternative representation of the learner $\psi(D_n)$ by adjusting the notation slightly without introducing ambiguity. The motivation for using this notation will become clearer in the subsequent section, where we introduce the discrete super learner.

2.2 Library of learners

We would like to consider the scenario where we have a set of learning algorithms, $\psi_1, \ldots, \psi_k$. From these algorithms, we can define the *library of learning algorithms*

$$\Psi = \{\psi_q \mid 1 \leq q \leq k\},$$

of size k. Our natural goal is to find $\tilde{\psi} \in \Psi$ such that $\tilde{\psi}(P_n)$ achieves the lowest risk

$$\tilde{\psi} = \arg\min_{\psi \in \Psi} R(\psi(P_n), P).$$

However, we cannot compute the risk of a learner directly as it depends on P. We seek to provide an estimate for $\tilde{\psi}$ which we will denote as $\hat{\psi}$.

3 The Discrete Super Learner

3.1 Cross-validation methodology

To provide the estimate $\hat{\psi}$ we will proceed via cross validation. Cross-validation randomly splits our data into a *training set* and a *test set* that our algorithms are trained and evaluated on. Let the random binary vector – referred to as the *split variable* or just *split* – $S^n = (S_1^n, \ldots, S_n^n) \in \{0,1\}^n$ be independent of D_n such that $S_i^n = 0$ indicates that O_i should be in the training set and $S_i^n = 1$ indicates that O_i belongs to the test set. The split S^n depends on n as it is a tuple in $\{0,1\}^n$. Here the superscript n is used to indicate that. We can define the empirical distributions over these two subsets as

$$P^0_{n,S^n} := \frac{1}{n_0} \sum_{i:S_i^n=0} \delta_{O_i},$$
$$P^1_{n,S^n} := \frac{1}{n_1} \sum_{i:S_i^n=1} \delta_{O_i},$$

where $n_1 = \sum_{i=1}^n S_i^n$ and $n_0 = 1 - n_1$ identifies the number of observations in the test and training set.

Example 2 (Random splits). For n observations we have 2^n ways of choosing which observations should be in the training set and in the test set. It might not be desirable to define the discrete probabilities for S^n over $\{0,1\}^n$ simply as $\frac{1}{2^n}$ for each possible combination of training/test data, since that would also include the combination where $n_1 = 0$. To ensure that there is always a certain amount of observations in our test set, let $0 < n_1 < n$ be given, we see that there are $\binom{n}{n_1}$ ways of choosing both the test and training set. We can therefore define the distribution of S^n as

$$P\left(S^n = s^n\right) = \binom{n}{n_1}^{-1} \qquad \text{for each } s^n \in \{0,1\}^n \text{ where } \sum_i s_i^n = n_1,$$

this procedure is also known as Monte Carlo cross-validation (Laan and Dudoit, 2003). In practice one would have a hard time validating over all $\binom{n}{n_1}$ combinations as it can be very large. For $n = 100$ and $n_1 = 10$ corresponding to 10% of the data being in the validation set, the number of combinations is around $1.7 \cdot 10^{13}$. One would instead approximate by randomly sampling possible combinations. However, by doing random sampling there is the possibility that some observations will not be in any of the sampled validation sets. The probability of this happening will become small as we sample multiple times.

K-fold cross validation

The idea behind K-fold cross-validation is to split the data into K equally sized sets, where the candidate learners are fitted on $K - 1$ sets and their performance is evaluated on the remaining set. We index the validation sets by $s \in \{1, \ldots, K\}$ and let $s(i)$ indicate the validation set that observation i belongs to. We will let $P^0_{n,s}$ denote the empirical measure over the observations that are not in set s – the training set, and let $P^1_{n,s}$ denote the empirical measure over the observations that are in the set s – the validation set.

K-fold cross-validation amounts to examining a single split variable S^n whose probability mass is equally distributed on the binary vectors indexed by s. In practice one would select a K and use a random splitting procedure to generate K binary vectors such that the entire dataset is the disjoint union of the sets specified by each vector. It is also possible to run K-fold cross-validation multiple times, corresponding to multiple calls to the random splitting procedure.

3.2 Risks and selectors

In the context where a learning algorithm is applied to the training data, we write $\psi(P^0_{n,S^n})$ for the fitted learner. The performance of the learner is then evaluated on the test data. Given that the risk Equation (1) was the integral of the loss function with respect to the data-generating distribution P, we can define the empirical risk of a learner as the integral of the loss with respect to the empirical measure of the test data, as follows

$$R(\psi(P^0_{n,S^n}), P^1_{n,S^n}) = \int L(O, \psi(P^0_{n,S^n}))dP^1_{n,S^n}(O),$$

where S^n is some split for our data D_n. Our motivation for defining the empirical risk is that we can compare it with the true risk of the learner where we integrate with respect to the data-generating distribution

$$R(\psi(P^0_{n,S^n}), P) = \int L(O, \psi(P^0_{n,S^n}))dP(O).$$

The following definitions are analogous to those stated in section 1 and 2 of (Laan and Dudoit, 2003).

Definition 5 (Risk averaged over splits (Vaart, Dudoit, and Laan, 2006))**.** Let the data D_n, a split-variable S^n and a library Ψ be given. Let $\psi \in \Psi$ be a learning algorithm. The risk for the learner, $\psi(P^0_{n,S^n})$, created from applying ψ on the training data, averaged over S^n is

$$E_{S^n} R(\psi(P^0_{n,S^n}), P),$$

where P is the data-generating distribution.

The expectation E_{S^n} is a simple average since S^n is discrete. Therefore, for a given ψ we have

$$E_{S^n} R(\psi(P^0_{n,S^n}), P) = \sum_{s \in \{0,1\}^n} R(\psi(P^0_{n,s}), P) \cdot P(S^n = s).$$

Definition 6 (Oracle selector (Laan and Dudoit, 2003))**.** Let the data D_n, a split variable S^n and a library Ψ be given, the learning algorithm $\psi \in \Psi$ with the lowest averaged risk is the *oracle selected algorithm* for n observations

$$\tilde{\psi}_n := \arg\min_{\psi \in \Psi} E_{S^n} R(\psi(P^0_{n,S^n}), P).$$

The cross-validation selector replaces P with P^1_{n,S^n} in the second argument of R.

Definition 7 (Cross-validation risk)**.** Given the data D_n, a split-variable S^n and a library Ψ. Let $\psi \in \Psi$ be a learning algorithm. The cross-validation risk for the learner, $\psi(P^0_{n,s})$, created from applying ψ on the training data, averaged over S^n is

$$E_{S^n} R(\psi(P^0_{n,S^n}), P^1_{n,S^n}),$$

where P^1_{n,S^n} is the test data as specified by the split-variable.

Definition 8 (Cross-validation selector (Laan and Dudoit, 2003))**.** Given the data D_n, a split variable S^n and a library Ψ, the learning algorithm $\psi \in \Psi$ with the lowest cross-validation risk is the *cross-validation selected algorithm* for n observations

$$\hat{\psi}_n := \arg\min_{\psi \in \Psi} E_{S^n} R(\psi(P^0_{n,S^n}), P^1_{n,S^n}).$$

The oracle selector and cross-validation selector are what chooses a learning algorithm from the library. Alternatively, one can first define the oracle selector as the index, $\tilde{q}$, of the algorithm in the library with the lowest risk, then the oracle selected algorithm would naturally be $\psi_{\tilde{q}}$.

3.3 Discrete super learner

Definition 9 (Discrete super learner)**.** The *discrete super learner*, $\hat{\psi}_n(P_n)$, created from a library Ψ is the cross-validation selected algorithm fitted to the entire dataset

$$\mathcal{X} \ni x \mapsto \hat{\psi}_n(P_n)(x).$$

Formally, the map above is a random map as P_n is stochastic. The discrete super learner is not a specific learner among the learners in the library, but the resulting algorithm from applying the cross-validation selector, fitted to the dataset. The pseudocode for the discrete super learner fitting procedure can be found in Section 5.1.

3.4 Oracle property

We introduce the notation Pf for the integral $\int f dP$ of an integrable function f with respect to P. Additionally, if P_n represents the empirical measure of $O_1, \ldots, O_n$, we denote the empirical process indexed over an appropriate class of functions $\mathcal{F}$ as $G_n f = \sqrt{n}(P_n f - Pf)$. Furthermore, we extend this notation to $G^v_{n,S^n} f = \sqrt{n}(P^v_{n,S^n} - Pf)$ for the empirical processes that correspond to applying the empirical measure over either the training data or test data, $v = 0$ or $v = 1$.

In the following results we assume that a proper loss function $L : \mathcal{O} \times \Theta \to \mathbb{R}$ has been given.

Lemma 10 (Lemma 2.1 in (Vaart, Dudoit, and Laan, 2006))**.** *Let G_n be the empirical process of an i.i.d. sample of size n from the distribution P and Ψ a library of learning algorithms. Furthermore, let $\hat{\psi}_n$ and $\tilde{\psi}_n$ denote the cross-validation- and oracle selected algorithms from Ψ respectively. For $\delta > 0$ it holds that*

$$\begin{aligned} E_{S^n} R(\hat{\psi}_n(P^0_{n,S^n}), P) \leq\ & (1+2\delta) E_{S^n} R(\tilde{\psi}_n(P^0_{n,S^n}), P) \\ & + E_{S^n} \frac{1}{\sqrt{n_1}} \max_{\psi \in \Psi} \int L(O, \psi(P^0_{n,S^n})) d((1+\delta) G^1_{n,S^n} - \delta \sqrt{n_1} P)(O) \\ & + E_{S^n} \frac{1}{\sqrt{n_1}} \max_{\psi \in \Psi} \int -L(O, \psi(P^0_{n,S^n})) d((1+\delta) G^1_{n,S^n} + \delta \sqrt{n_1} P)(O). \end{aligned}$$

Proof. See appendix. □

To provide the bounds for the risk we will introduce Bernstein pairs. They are used in the Bernstein inequality to control the empirical process $G_n f$ for $f \in \mathcal{F}$, which is a critical ingredient in Lemma 13. The exact details in the proof of Lemma 13 requires an introductory course to empirical processes, which is omitted for the purposes of this thesis.

Definition 11 (Bernstein pair (Vaart, Dudoit, and Laan, 2006))**.** Given a measurable function $f : \mathcal{O} \to \mathbb{R}$, the tuple $(M(f), v(f))$ is a Bernstein pair if

$$M(f)^2 P\left(e^{|f|/M(f)} - 1 - \frac{|f|}{|M(f)|}\right) \leq \frac{1}{2} v(f). \tag{2}$$

Proposition 12. *If f is uniformly bounded, then $(\|f\|_\infty, \frac{3}{2}Pf^2)$ is a Bernstein pair.*

Proof. The following proof is due to (Vaart, Dudoit, and Laan, 2006)[ch. 8.1].

$$\begin{aligned}
\|f\|_\infty^2 P\left(e^{|f|/\|f\|_\infty} - 1 - \frac{|f|}{\|f\|_\infty}\right) &= \|f\|_\infty^2 \sum_{k\geq 2}^{\infty} P\frac{|f|^k}{\|f\|_\infty^k k!} = Pf^2 \sum_{k\geq 2}^{\infty} \frac{|f|^{k-2}}{\|f\|_\infty^{k-2} k!} \\
&\leq Pf^2 \sum_{k\geq 2}^{\infty} \frac{\|f\|_\infty^{k-2}}{\|f\|_\infty^{k-2} k!} = Pf^2 \sum_{k\geq 2}^{\infty} \frac{1}{k!} \\
&= Pf^2(e-2) \leq \frac{3}{4} Pf^2 = \frac{1}{2}\left(\frac{3}{2} Pf^2\right).
\end{aligned}$$

In the first inequality we replace the absolute value of f with the uniform norm, which is larger. □

Example 3 (Binary regression). Consider binary regression with quadratic loss. Let $\theta \in \Theta$ be arbitrary, then a Bernstein pair for the function $f_\theta(O) = L(O, \theta) = (Y - \theta(X))^2$ can be found by applying Proposition 12. By requiring that $\theta(x) \in [0, 1]$, then it is clear that f is bounded between 0 and 1 for all $O \in \mathcal{O}$ since $Y \in \{0, 1\}$. We also note that the Bernstein pair for f_θ satisfies $0 \leq \|f_\theta\|_\infty \leq 1$ and $0 \leq \frac{3}{2} Pf_\theta^2 \leq 1$, so they are bounded.

Lemma 13 (Lemma 2.2 in (Vaart, Dudoit, and Laan, 2006))**.** *Let G_n be the empirical process of an i.i.d. sample of size n from the distribution P and assume that $Pf \geq 0$ for every $f \in \mathcal{F}$ in some set of measurable functions $\mathcal{F}$ on $\mathcal{O}$. Then, for any Bernstein pair $(M(f), v(f))$ and for any $\delta > 0$ and $1 \leq p \leq 2$,*

$$E_{D_n} \max_{f\in\mathcal{F}} (G_n - \delta\sqrt{n}P) f \leq \frac{8}{n^{1/p-1/2}} \log(1 + \#\mathcal{F}) \max_{f\in\mathcal{F}} \left[\frac{M(f)}{n^{1-1/p}} + \left(\frac{v(f)}{(\delta Pf)^{2-p}}\right)^{1/p}\right].$$

The same upper bound is valid for $E_{D_n} \max_{f\in\mathcal{F}} (G_n + \delta\sqrt{n}P)(-f)$.

The expectation E_{D_n} is taken with respect to the joint probability measure over our observations, here we have that $D_n \sim P_O^n = P_{O_1} \otimes P_{O_2} \otimes \cdots \otimes P_{O_n}$.

Theorem 14 (Theorem 2.3 in (Vaart, Dudoit, and Laan, 2006))**.** *Let Ψ be a library of learning algorithms of size k. For $\theta \in \Theta$ let the numbers $(M(\theta), v(\theta))$ be a Bernstein pair for the function $O \mapsto L(O, \theta)$ and assume that $R(\theta, P) \geq 0$ for every $\theta \in \Theta$. Then for $\delta > 0$ and $1 \leq p \leq 2$ it holds that*

$$\begin{aligned}
E_{D_n} E_{S^n} R(\hat{\psi}_n(P^0_{n,S^n}), P) \leq (1 + 2\delta) E_{D_n} E_{S^n} R(\tilde{\psi}_n(P^0_{n,S^n}), P) + \\
(1 + \delta) E_{S^n}\left(\frac{16}{n_1^{1/p}} \log(1 + k) \sup_{\theta\in\Theta}\left[\frac{M(\theta)}{n_1^{1-1/p}} + \left(\frac{v(\theta)}{R(\theta, P)^{2-p}}\right)^{1/p}\left(\frac{1+\delta}{\delta}\right)^{2/p-1}\right]\right).
\end{aligned}$$

Where $\hat{\psi}_n$ and $\tilde{\psi}_n$ are the cross-validation- and the oracle selected algorithms from Ψ.

In the expectations above, we are taking the expectation with respect to the random split-variable S^n as well as the joint distribution of our observations. In a more verbose manner one would write

$$E_{D_n}E_{S^n}R(\hat{\psi}_n(P^0_{n,S^n}),P) = \int R(\hat{\psi}_n(P^0_{n,S^n}),P)d(P_{S^n}\otimes P_O^n).$$

Proof. We will apply Lemma 13 to the second and third terms on the left hand side of the inequality in Lemma 10. Let $\mathcal{F} = \{O \mapsto L(O,\psi(P^0_{n,S^n})) \mid \psi \in \Psi\}$, be the collection of functions obtained by applying the loss L to the algorithms in our library Ψ fitted on the training data. Note that $\mathcal{F} \subseteq \{O \mapsto L(O,\theta) \mid \theta \in \Theta\}$, and since $R(\theta,P) \geq 0$ for every $\theta \in \Theta$ it follows that $Pf \geq 0$ for every $f \in \mathcal{F}$.

First, we take the expectation with respect to D_n on both sides in Lemma 10. For the second term on the right hand side we have

$$\begin{aligned}
&E_{D_n}E_{S^n}\frac{1}{\sqrt{n_1}}\max_{\psi\in\Psi}\int L(O,\psi(P^0_{n,S^n}))d((1+\delta)G^1_{n,S^n} - \delta\sqrt{n_1}P)(O)\\
&= E_{D_n}E_{S^n}\frac{1+\delta}{\sqrt{n_1}}\max_{\psi\in\Psi}\int L(O,\psi(P^0_{n,S^n}))d(G^1_{n,S^n} - \frac{\delta}{1+\delta}\sqrt{n_1}P)(O)\\
&= E_{S^n}\frac{1+\delta}{\sqrt{n_1}}E_{D_n}\max_{\psi\in\Psi}\int L(O,\psi(P^0_{n,S^n}))d(G^1_{n,S^n} - \frac{\delta}{1+\delta}\sqrt{n_1}P)(O).
\end{aligned}$$

Where we use Fubini in the last equality. Recall that $S^n \perp\!\!\!\perp D_n$, so we can consider n_1 as fixed given D_n. By applying Lemma 13 to the expression above with $n = n_1$ yields

$$\begin{aligned}
&E^n_S\frac{1+\delta}{\sqrt{n_1}}E_{D_n}\max_{\psi\in\Psi}\int L(O,\psi(P^0_{n,S^n}))d(G^1_{n,S^n} - \frac{\delta}{1+\delta}\sqrt{n_1}P)(O)\\
&\leq E_{S^n}\frac{1+\delta}{\sqrt{n_1}}\frac{8}{n_1^{1/p-1/2}}\log(1+k)\max_{\psi\in\Psi}\left[\frac{M(\psi(P^0_{n,S^n}))}{n_1^{1-1/p}} + \left(\frac{v(\psi(P^0_{n,S^n}))}{(\frac{\delta}{1+\delta})^{2-p}R(\psi(P^0_{n,S}),P)^{2-p}}\right)^{1/p}\right]\\
&\leq E_{S^n}\frac{1+\delta}{\sqrt{n_1}}\frac{8}{n_1^{1/p-1/2}}\log(1+k)\sup_{\theta\in\Theta}\left[\frac{M(\theta)}{n_1^{1-1/p}} + \left(\frac{v(\theta)}{(\frac{\delta}{1+\delta})^{2-p}R(\theta,P)^{2-p}}\right)^{1/p}\right]\\
&= (1+\delta)E_{S^n}\frac{8}{n_1^{1/p}}\log(1+k)\sup_{\theta\in\Theta}\left[\frac{M(\theta)}{n_1^{1-1/p}} + \left(\frac{v(\theta)}{R(\theta,P)^{2-p}}\right)^{1/p}\left(\frac{1+\delta}{\delta}\right)^{2/p-1}\right].
\end{aligned}$$

Where for the third inequality we take the sup over Θ. We can also bound the third term with the same expression above as Lemma 13 is also valid for $-L$. It is now immediate from Lemma 10 that

$$\begin{aligned}
E_{D_n}E_{S^n}\int L(O,\hat{\psi}_n(P^0_{n,S^n}))dP(O) &\leq (1+2\delta)E_{D_n}E_{S^n}\int L(O,\tilde{\psi}_n(P^0_{n,S^n}))dP(O)\\
&+ (1+\delta)E_{S^n}\frac{8}{n_1^{1/p}}\log(1+k)\sup_{\theta\in\Theta}\left[\frac{M(\theta)}{n_1^{1-1/p}} + \left(\frac{v(\theta)}{R(\theta,P)^{2-p}}\right)^{1/p}\left(\frac{1+\delta}{\delta}\right)^{2/p-1}\right]\\
&+ (1+\delta)E_{S^n}\frac{8}{n_1^{1/p}}\log(1+k)\sup_{\theta\in\Theta}\left[\frac{M(\theta)}{n_1^{1-1/p}} + \left(\frac{v(\theta)}{R(\theta,P)^{2-p}}\right)^{1/p}\left(\frac{1+\delta}{\delta}\right)^{2/p-1}\right].
\end{aligned}$$

The second and third terms above are identical, meaning that they can be combined into one term where the numerator in the first fraction is 16 instead of 8. □

Remark. One might notice that S^n still appears in the remainder of the bound above, but the only variable in the remainder that depends on S^n is n_1. It is, therefore, completely possible to drop the expectation with respect to S^n if n_1 is deterministic, for example, if it is a fraction of n.

In the following corollary we assume that the Bernstein pairs $(M(\theta), v(\theta))$ are bounded over Θ, thus taking the supremum over Θ will not result in the remainder being infinite. The assumption is indeed valid for binary regression as we have remarked upon in Example 3.

Corollary 15 (Asymptotic equivalence)**.** *If there exists an $\varepsilon > 0$ such that*

$$E_{D_n} E_{S^n} R(\tilde{\psi}_n(P^0_{n,S^n}), P) > \varepsilon \qquad \text{for all } n \in \mathbb{N},$$

and if $n_1 = f(n)$ for some polynomial function f, then the risk of the super learner is asymptotically equivalent with the risk of the oracle selected learner, that is

$$\lim_{n\to\infty} \frac{E_{D_n} E_{S^n} R(\hat{\psi}_n(P^0_{n,S^n}), P)}{E_{D_n} E_{S^n} R(\tilde{\psi}_n(P^0_{n,S^n}), P)} = 1.$$

For the sake of simplicity and the previous remark n_1 is considered as a sequence of numbers polynomial in n. This will allow us to remove E_{S^n} in the remainder term.

Proof. We let $\delta_n := 1/\log(n)$. By choosing $p = 1$ in Theorem 14 and substituting δ with δ_n, we obtain

$$\begin{aligned} E_{D_n} E_{S^n} R(\hat{\psi}_n(P^0_{n,S^n}), P) &\leq (1 + 2\delta_n) E_{D_n} E_{S^n} R(\tilde{\psi}_n(P^0_{n,S^n}), P) \\ &\quad + (1 + \delta_n) \frac{16}{n_1} \log(1 + k) \sup_{\theta\in\Theta} \left[M(\theta) + \frac{v(\theta)}{R(\theta, P)} \cdot \frac{1 + \delta_n}{\delta_n} \right] \\ &= (1 + 2\delta_n) E_{D_n} E_{S^n} R(\tilde{\psi}_n(P^0_{n,S^n}), P) \\ &\quad + (1 + \delta_n) \frac{16}{n_1} \log(1 + k) \sup_{\theta\in\Theta} \left[M(\theta) + \frac{v(\theta)}{R(\theta, P)} \cdot (\log(n) + 1) \right]. \end{aligned}$$

In the above inequality the remainder term goes to 0 as n_1 increases. The $\log(n)$ term in the supremum will be dominated by $1/n_1$ that is multiplied in the front. By dividing the risk of the oracle selected algorithm on both sides, we obtain

$$\begin{aligned} \frac{E_{D_n} E_{S^n} R(\hat{\psi}_n(P^0_{n,S^n}), P)}{E_{D_n} E_{S^n} R(\tilde{\psi}_n(P^0_{n,S^n}), P)} &\leq 1 + 2\delta_n \\ &\quad + \frac{(1 + \delta_n) \frac{16}{n_1} \log(1 + k) \sup_{\theta\in\Theta} \left[M(\theta) + \frac{v(\theta)}{R(\theta,P)} \cdot (\log(n) + 1) \right]}{E_{D_n} E_{S^n} R(\tilde{\psi}_n(P^0_{n,S^n}), P)}. \end{aligned}$$

The fraction on the right hand side will converge to 0 as $n \to \infty$, since n_1 is polynomial in n and the risk of the oracle selected algorithm is bounded away from 0 by assumption. The entire right hand side will therefore converge to 1 as $\delta_n \to 0$. Note also that by the definition of the oracle, we have

$$E_{D_n} E_{S^n} R(\tilde{\psi}_n(P^0_{n,S^n}), P) \leq E_{D_n} E_{S^n} R(\hat{\psi}_n(P^0_{n,S^n}), P),$$

implying that

$$1 \leq \frac{E_{D_n} E_{S^n} R(\hat{\psi}_n(P^0_{n,S^n}), P)}{E_{D_n} E_{S^n} R(\tilde{\psi}_n(P^0_{n,S^n}), P)},$$

applying the squeeze theorem thus yields the desired result. □

In the proof of the asymptotic result we have chosen $\delta_n = 1/\log(n)$. Consequently, the remainder converges to 0 at an order of $O(\log(n)/n)$. An immediate consequence of this is that if we have a library of parametric learning algorithms and one of them corresponds to the true regression, then the risk of the oracle selected algorithm might very well converge at an order of $O(1/n)$, implying that the risk of the discrete super learner achieves an almost parametric rate of convergence of $O(\log(n)/n)$ to the oracle risk (Laan, Polley, and Hubbard, 2007).

For the asymptotic equivalence to hold, we must have that $n_1 \to \infty$ as $n \to \infty$. This is a reasonable assumption for cross-validation schemes such as K-fold cross-validation or if $n_1 = np$ for $p \in (0,1)$ where the test size increases with the amount of available data. Note that the condition is not satisfied for leave-one-out cross-validation since n_1 is equal to 1 for every n.

Example 4 (Regression). In a regression scenario with quadratic loss, the risk for a learner can never be zero unless the outcome is deterministic given x. We may begin by noting that the risk of $\theta^*(x) = E(Y \mid X = x)$ is always positive,

$$\int (Y - E(Y \mid X))^2 dP \geq 0.$$

Since the integrand is positive, the integral is zero if and only if $Y = E(Y \mid X)$ almost surely. This is only the case when Y is deterministic given X, i.e. Y is X-measurable. Assuming, therefore, that Y is not deterministic given X, then we note that by the fact that the quadratic loss is strictly proper due to Theorem 2, it holds for any $\theta \in \Theta$ that

$$R(\theta, P) \geq R(\theta^*, P) > 0,$$

thus showing that the risk of any learner is strictly positive.

4 The Ensemble Super Learner

The idea behind the *ensemble super learner* is to combine the predictions made by each learner, $\psi_1(P_n), \ldots, \psi_k(P_n)$, into a single prediction. The idea of combining learners was examined in Breiman (1996) which considered how a linear combination of the learners in the library could be formed. This approach, also known as *stacked regression* or *stacking*, is a variant of the ensemble super learner. The ensemble super learner, however, does not limit itself to linear combinations but uses a *meta learning algorithm* on the *level 1 data* – the predictions made by the learners. It is a broad approach of balancing the weaknesses of each learner by forming an ensemble. However, as we will see, K-fold cross-validation is integral in forming the ensemble super learner.

We will proceed to give the definitions and setup necessary to define the ensemble super learner.

4.1 Level 1 data

Let D_n be our data and let Ψ be a library of learning algorithms. The *level 1 covariates* of Ψ applied to D_n as specified by a K-fold cross validation procedure are

$$\mathcal{Z} = \{Z_i = (\psi_1(P^0_{n,s(i)})(X_i), \ldots, \psi_k(P^0_{n,s(i)})(X_i))\}_{i=1}^n \subseteq [0,1]^k,$$

which is the set of possible outcomes by applying the learners on the observed X_i's.

Definition 16 (Level 1 data)**.** The *level 1 data*, $\mathcal{L}_n \subseteq \{0,1\} \times \mathcal{Z}$, is given by concatenating our observed Y_i's with the level 1 covariates, i.e.

$$\begin{aligned}\mathcal{L}_n &= \{(Y_i; Z_i)\}_{i=1}^n \\ &= \{(Y_i; \psi_1(P^0_{n,s(i)})(X_i), \ldots, \psi_k(P^0_{n,s(i)})(X_i))\}_{i=1}^n.\end{aligned}$$

The level 1 data is exactly the predictions made by the learners on the observations that they were not trained on. Thus, *level 0 data* is what we refer to as D_n. We now define $\mathcal{M}$ as the class of all measurable functions that map from $\mathcal{Z}$ to [0,1] which we will refer to as *meta learners*.

4.2 Meta learners

Definition 17 (Meta learner)**.** The *meta learner* is a function $\phi : \mathcal{Z} \to [0,1]$ in $\mathcal{M}$ that maps the output of the candidate learners to a prediction.

The goal is to estimate the regression of Y given the predictions made by our learners, $\mathcal{Z} \ni z \mapsto E(Y \mid Z = z)$, which is an element in $\mathcal{M}$. We accomplish this by applying a *meta learning algorithm* to the level 1 data.

Definition 18 (Meta learning algorithm)**.** A *meta learning algorithm* Φ is a measurable map that creates a meta learner from our level 1 data $\mathcal{L}_n \mapsto \Phi(\mathcal{L}_n) \in \mathcal{M}$.

A meta learning algorithm seeks to estimate $E(Y \mid Z)$. It can for example be a parametric learning algorithm such as logistic regression. If a parametric learning algorithm is used, it is important to take into account that the level 1 covariates Z_i are highly correlated and that multicollinearity can occur. Breiman (1996) suggests ridge regression as a possible way to shrink the coefficients. Another method is to use a non-negativity constraint, where we specify that the coefficients are positive and must sum to 1 in the case of a simple linear combination.

4.3 Ensemble super learner

Definition 19 (Ensemble super learner (Laan, Polley, and Hubbard, 2007))**.** Let a library of learning algorithms Ψ be given and let $\mathcal{L}_n$ be the level 1 data obtained by fitting each learning algorithm according to some K-fold cross validation procedure. Let $\phi = \Phi(\mathcal{L}_n)$ be the outcome of applying a meta learning algorithm to the level 1 data, then the map

$$x \mapsto \Sigma(P_n)(x) = \phi(\psi_1(P_n)(x), \ldots, \psi_k(P_n)(x)),$$

is called the *ensemble super learner* and we will denote it by $\Sigma(P_n)$.

Here the P_n indicates that each learner $\psi \in \Psi$ is fitted on the entire data set. A meta learner, ϕ, is used to combine the predictions made by each learner.

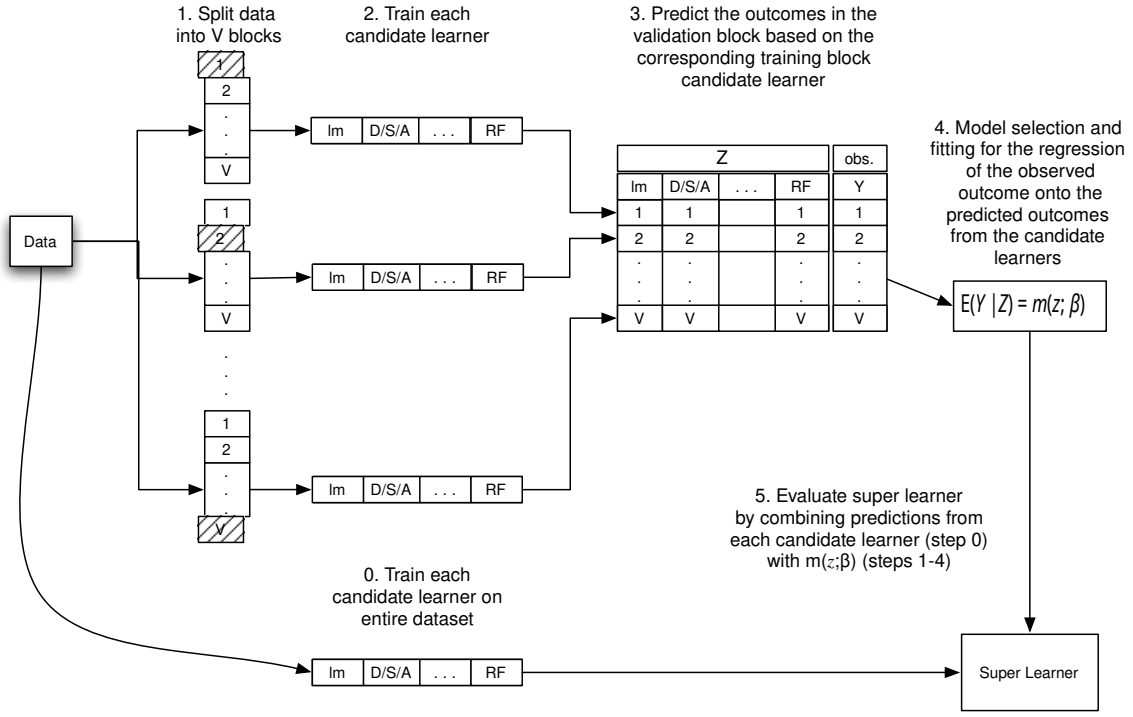

Figure 1: The steps involved in creating the ensemble super learner, taken from Laan, Polley, and Hubbard (2007).

Figure 1 from Laan, Polley, and Hubbard (2007) illustrates how the ensemble super learner is fitted in practice. Step 3 is where the meta learning algorithm, Φ, applied to the level 1 data. The result is a meta learner m, which we have denoted in our setup as ϕ. The pseudocode for the ensemble super learner fitting procedure can be found in Section 5.2.

4.4 Oracle property

The meta learning algorithm, Φ, fits the meta learner that best estimates $E(Y \mid Z)$. A way of formalizing the notion of 'fitting' is that Φ seeks to select the meta learner with the lowest risk among an indexed set of meta learners. Let $\mathcal{A}$ be a parameter set (for example Euclidean), for each $a \in \mathcal{A}$ let ϕ_a be a meta learner. In a practical setting, one might consider ϕ_a as the learner obtained by instantiating it with the parameter a. Given our level 1 data $\mathcal{L}_n$, define

$$\hat{a} := \arg\min_{a \in \mathcal{A}} \frac{1}{n} \sum_{i=1}^{n} L((Y_i, Z_i), \phi_a) = \arg\min_{a \in \mathcal{A}} \frac{1}{n} \sum_{i=1}^{n} (Y_i - \phi_a(Z_i))^2. \tag{3}$$

The meta learning algorithm applied to $\mathcal{L}_n$ returns $\phi_{\hat{a}}$, which is the meta learner with the lowest empirical risk on the level 1 data. We will now consider a subset $\mathcal{A}_n \subseteq \mathcal{A}$ such that the number of elements in the subset, $k(n) = |\mathcal{A}_n|$, depends on n. Denote the ensemble super learner that uses the meta learner ϕ_a as

$$\Sigma_a(P_n) := \phi_a(\psi_1(P_n), \ldots, \psi_k(P_n)).$$

Let the cross-validation- and oracle selector of a be

$$\hat{a}_n := \arg\min_{a \in \mathcal{A}_n} \frac{1}{K} \sum_{s=1}^{K} R(\Sigma_a(P^0_{n,s}), P^1_{n,s}),$$

$$\tilde{a}_n := \arg\min_{a \in \mathcal{A}_n} \frac{1}{K} \sum_{s=1}^{K} R(\Sigma_a(P^0_{n,s}), P),$$

where P is the data-generating distribution in our level 0 data. Note that for each s' up to K

$$\Sigma_a(P^0_{n,s'})(X_i) = \phi_a(\psi_1(P^0_{n,s'})(X_i), \ldots, \psi_k(P^0_{n,s'})(X_i)) = \phi_a(Z_i).$$

holds true for each i if $s(i)$ equals s', i.e. when X_i is in the validation set s'. By using the fact that the K sets are disjoint and exhaustive, we can express $\hat{a}_n$ as

$$\hat{a}_n = \arg\min_{a \in \mathcal{A}_n} \frac{1}{K} \sum_{s'=1}^{K} \frac{K}{n} \sum_{i:s(i)=s'} (Y_i - \Sigma_a(P^0_{n,s'})(X_i))^2 = \arg\min_{a \in \mathcal{A}_n} \frac{1}{n} \sum_{i=1}^{n} (Y_i - \phi_a(Z_i))^2. \quad (4)$$

Equation (4) shows that the cross-validation selector, $\hat{a}_n$, is exactly the objective of the meta learning algorithm defined in (3).

The way that $\hat{a}_n$ and $\tilde{a}_n$ are defined fit directly into our existing framework for the discrete super learner. Recall that the cross-validation selector selected the algorithm that had the lowest cross-validation risk averaged across some split variable S^n. By using a K-fold cross-validation scheme, the averaged risk over S^n becomes the averaged risk over the K sets, which is exactly the above definition of $\hat{a}_n$. In the case of the ensemble super learner, the learning algorithm, ϕ_a, is determined by the parameter a, so the selection problem reduces to selecting the most optimal parameter over a finite parameter set $\mathcal{A}_n$. The parameter set $\mathcal{A}_n$ indexes the meta learning algorithms, and is what the cross-validation selector acts upon. It is, therefore, possible to think of $\mathcal{A}_n$ as approximating the entire parameter set $\mathcal{A}$ using a grid of points such that the approximation becomes finer as the number of points increase. The key difference lies in the fact that the ensemble super learner is a completely new learner, whereas the discrete super learner is an algorithm selected from the library Ψ fitted on the dataset.

The following finite sample oracle inequality is a direct consequence of the oracle inequality for the discrete super learner, Theorem 14.

Theorem 20 (Finite sample oracle inequality)**.** *Let Ψ be a library of learning algorithms of size k. Let $\Sigma_{\hat{a}_n}$ be the ensemble super learner over Ψ obtained by applying the cross-validation selector over a finite parameter space $\mathcal{A}_n$ using the K-fold cross-validation scheme. Let $\Sigma_{\tilde{a}_n}$ be the oracle ensemble learner obtained by applying the oracle selector over the same set of parameters. If the conditions of Theorem 14 are satisfied, then the following oracle inequality holds*

$$\frac{1}{K} \sum_{s=1}^{K} E_{D_n} R(\Sigma_{\hat{a}_n}(P^0_{n,s}), P) \leq (1 + 2\delta) \frac{1}{K} \sum_{s=1}^{K} E_{D_n} R(\Sigma_{\tilde{a}_n}(P^0_{n,s}), P) + (1 + \delta) \frac{16K}{n} \log(1 + k(n)) \sup_{\theta \in \Theta} \left[M(\theta) + \frac{v(\theta)}{R(\theta, P)} \frac{1 + \delta}{\delta} \right].$$

Proof. Replace E_{S^n} in Theorem 14 with the average over the K folds and n_1 with n/K, then by using $p = 1$ the desired result is obtained. □

Corollary 21 (Asymptotic equivalence)**.** *If the number of points in the parameter sets $\mathcal{A}_n$ grows at most at a polynomial rate, that is $k(n) \leq n^p$ for some $p \in \mathbb{N}$, then the ensemble super learner $\Sigma_{\hat{a}_n}$ is asymptotically equivalent with the oracle ensemble learner $\Sigma_{\tilde{a}_n}$, i.e.*

$$\lim_{n\to\infty} \frac{\frac{1}{K}\sum_{s=1}^{K} E_{D_n} R(\Sigma_{\hat{a}_n}(P^0_{n,s}), P)}{\frac{1}{K}\sum_{s=1}^{K} E_{D_n} R(\Sigma_{\tilde{a}_n}(P^0_{n,s}), P)} = 1.$$

Proof. The proof is analogous to Corollary 15. We will replace δ with $\delta_n = 1/\log(n)$. Note that in the remainder, the $\log(1+k(n))$ term is less than $1+\log(k(n)) = 1+p\log(n)$, which is of lower order than n. The $(1+\delta_n)/\delta_n$ term simplifies to become $1+\log(n)$, and when multiplied with the $\log(1+k(n))$ term, becomes of order $O(\log(n)^2)$ that is still of lower order than n. Thus the remainder term goes to zero as $n \to \infty$ and the asymptotic result follows. □

4.5 Choice of meta learning algorithm

Given the level 1 data, $\mathcal{L}_n$, the meta learning algorithm seeks to optimize over the parameter set $\mathcal{A}_n \subseteq \mathcal{A}$. The interpretation of $\mathcal{A}$ depends on the chosen learning algorithm. If a parametric learning algorithm is used, for example logistic regression, then $\mathcal{A}$ is $\mathbb{R}^k$ where k is the number of learners in Ψ. The learner ϕ_a for $a \in \mathcal{A}$ will then be the map $z \mapsto \text{expit}(z \cdot a)$ where $\cdot$ is the dot product in $\mathbb{R}^k$.

Laan, Polley, and Hubbard (2007) mentions that it is also possible to use a non-parametric learning algorithm as the meta learning algorithm, or even apply a super learning algorithm on the level 1 data. In this case $\mathcal{A}$ could be measurable functions from $\mathcal{Z}$ to $[0,1]$. The learner ϕ_a for $a \in \mathcal{A}$ will then be the map $z \mapsto a(z)$.

Constrained regression and the $(k-1)$-simplex

Here we examine a simple meta learning algorithm which is to do a constrained regression on the level 1 data. The reason for examining this particular meta learning algorithm is because the discrete super learner is a special case of the ensemble super learner that utilizes this algorithm. The goal is to fit a weighted combination of the learners, such that the weights sum to 1. We can achieve this by parametrizing $\mathcal{A}$ as the $(k-1)$-simplex given by

$$\mathcal{A} = \left\{ a \in \mathbb{R}^k \mid \sum_{q=1}^{k} a_q = 1, a_q \geq 0 \text{ for } 1 \leq q \leq k \right\},$$

where $k = |\Psi|$ is the number of algorithms in our library. The meta learner ϕ_a for $a \in \mathcal{A}$ is then the map $z \mapsto z \cdot a$. By Cauchy-Schwarz we have $z \cdot a \leq \|z\|\|a\| \leq 1$ meaning that the range of ϕ_a is [0,1]. The ensemble super learner that applies the meta learner ϕ_a is, therefore, a valid learner in Θ. To apply the oracle results, we will cover $\mathcal{A}$ with equidistant points on the simplex surface. For every n, we specify $\mathcal{A}_n \subseteq \mathcal{A}$ such that the number of points in $\mathcal{A}_n$ increases with n at a polynomial rate, and that for any $a \in \mathcal{A}$ we have

$$\min_{b_n \in \mathcal{A}_n} \|a - b_n\| \leq \frac{1}{n}.$$

This is clearly possible by Figure 2. It thus follows that we can find a sequence of points $b_n \in \mathcal{A}_n$ such that $\phi_{b_n} \to \phi_a$ for any $a \in \mathcal{A}$. The conclusion is that if $\tilde{a}$ specified the

learner $\phi_{\tilde{a}}$ with the minimum risk, then by applying the cross-validation selector and letting $n \to \infty$, the meta learner adopted by the ensemble super learner will eventually converge to $\phi_{\tilde{a}}$.

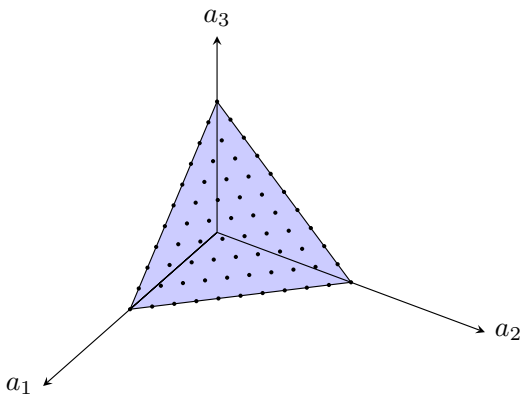

Figure 2: The parameter set $\mathcal{A}$ of learner weights visualized as a 2-simplex for a library of size $k = 3$. The k axes span the weight combinations for the library where the weight combinations that sum to 1 lie on the simplex. Here the discrete approximation $\mathcal{A}_n$ is visualized as points on the simplex.

The discrete super learner is essentially a special case of the ensemble super learner that utilizes constrained regression. If $a_q = 1$ for any $1 \leq q \leq k$ then the ensemble super learner predicts the same as the q'th learner. The discrete super learner can, therefore, be seen as constrained regression optimizing for a over the vertices of the $(k-1)$-simplex. Given that the training data is representative of the data-generating distribution, the ensemble learner that uses constrained regression should outperform the discrete super learner at minimizing risk since it optimizes over the entire $(k-1)$-simplex.

5 Simulations

In the following section we show the results of applying the two super learners to a simulated dataset. The simulations are carried out using the R programming language and the corresponding source code can be found on the GitHub repository for this project (Liu, 2023). The simulated dataset consists of a binary outcome, Y, which depends on two covariates X_1 and X_2. The setup is as follows

$$\begin{aligned} X_1 &\sim \text{Unif}(0.5, 15), \\ X_2 \mid X_1 = x_1 &\sim \mathcal{N}(3.5 - 0.03x_1, 1), \\ Y \mid X_1 = x_1, X_2 = x_2 &\sim \text{Ber}(\theta_0(x_1, x_2)), \end{aligned}$$

for $\theta_0(x_1, x_2) = \text{expit}(-3.5 - 0.3x_1 + 0.85x_2 + 0.35x_1x_2)$ which is the data-generating regression function. It is in fact possible to visualize the regression function explicitly as a 2-dimensional heat map in the covariates. In Figure 3 we have applied the true regression across the grid of evenly-spaced (x_1, x_2) covariate pairs in $(0, 15) \times (0,7)$. The probabilities are colored from 0 to 1 in the plot.

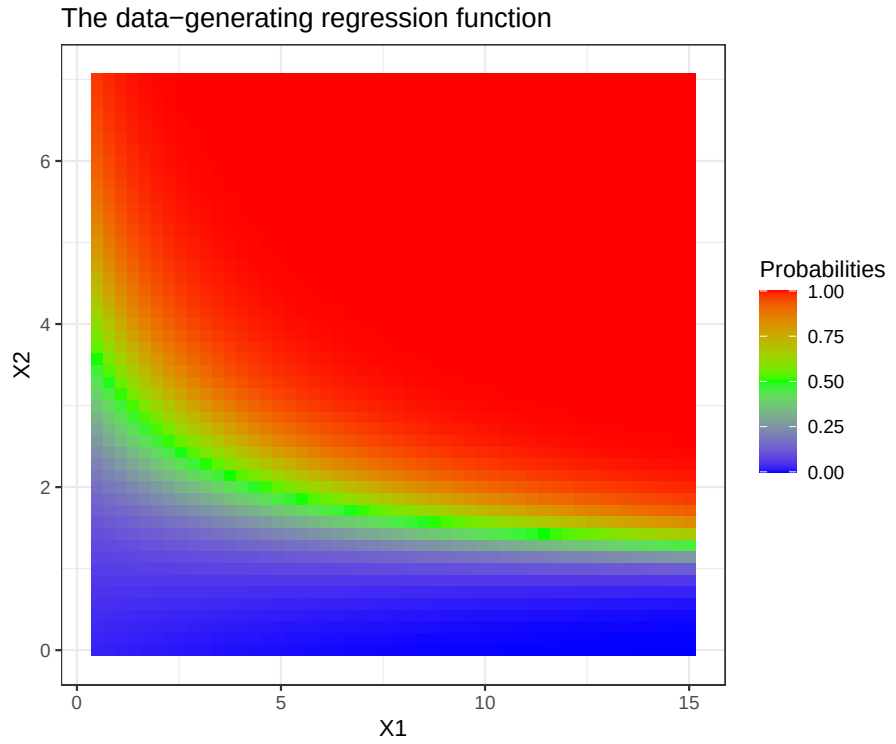

Figure 3: The data-generating regression plotted as a heat map. The two covariates X_1 and X_2 are mapped by the regression function θ_0 to a probability as indicated by the colors.

The regression is captured by the logistic regression model with interaction terms. We will use the following library of learning algorithms as an illustrative example:

1. Intercept only logistic regression: $E[Y \mid X_1, X_2] = \text{expit}(\beta_0)$
2. Logistic regression with main effects: $E[Y \mid X_1, X_2] = \text{expit}(\beta_0 + \beta_1 X_1 + \beta_2 X_2)$
3. XGBoost with hyperparameters: `max_depth=3, eta=0.3, n_rounds=100, objective='binary:logistic', booster='dart', nthread=5`

The intercept only logistic regression is included as a baseline, the predicted probability by the intercept model is simply the sample average over the observed Y_i's.

We can visualize the predictions of the learning algorithms in the library in the same way as we have done for the data-generating regression. In Figure 4 we visualize the predictions of the main effects logistic regression and XGBoost fitted using 1,000 observations sampled from the distribution. The plot for the intercept model is omitted, as its appearance is as one would expect – the plot is simply an orange square.

From Figure 4 we can observe a clear difference in the predicted probabilities between the logistic regression and the tree-based XGBoost. The main effects logistic regression is a parametric model that assumes that the regression function is a smooth transformation of the linear predictor $X\beta$. XGBoost, in contrast, is made up of many decision trees, which explains the patchwork pattern in its prediction plot. For small samples and as we will see in the simulations, XGBoost has a high risk in comparison to the misspecified main effects logistic regression. However, XGBoost becomes increasingly better at approximating the regression when the number of observations becomes large as seen in Figure 5.

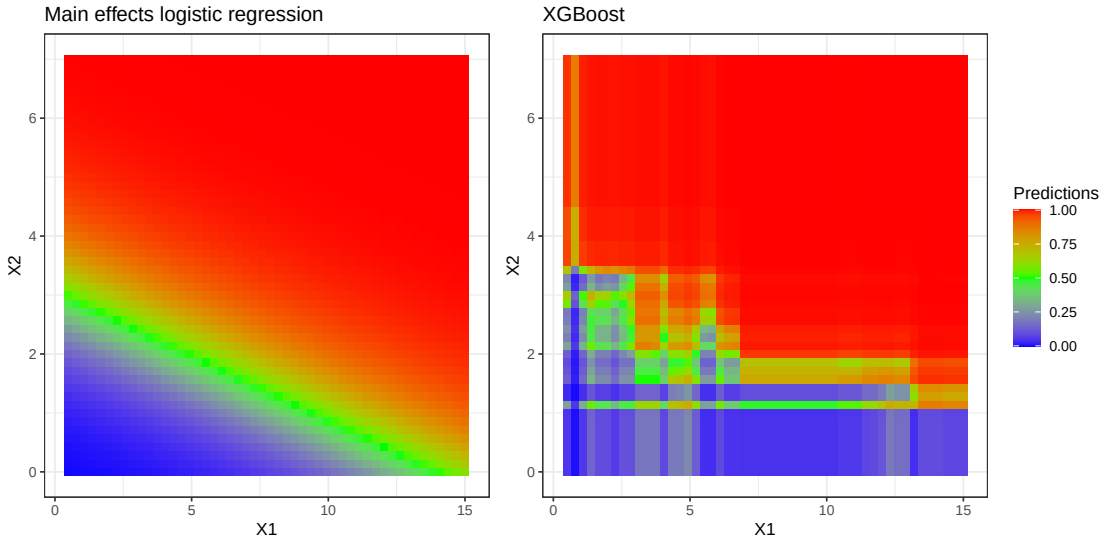

Figure 4: The predictions of the main effects logistic regression and XGBoost fitted on 1,000 observations.

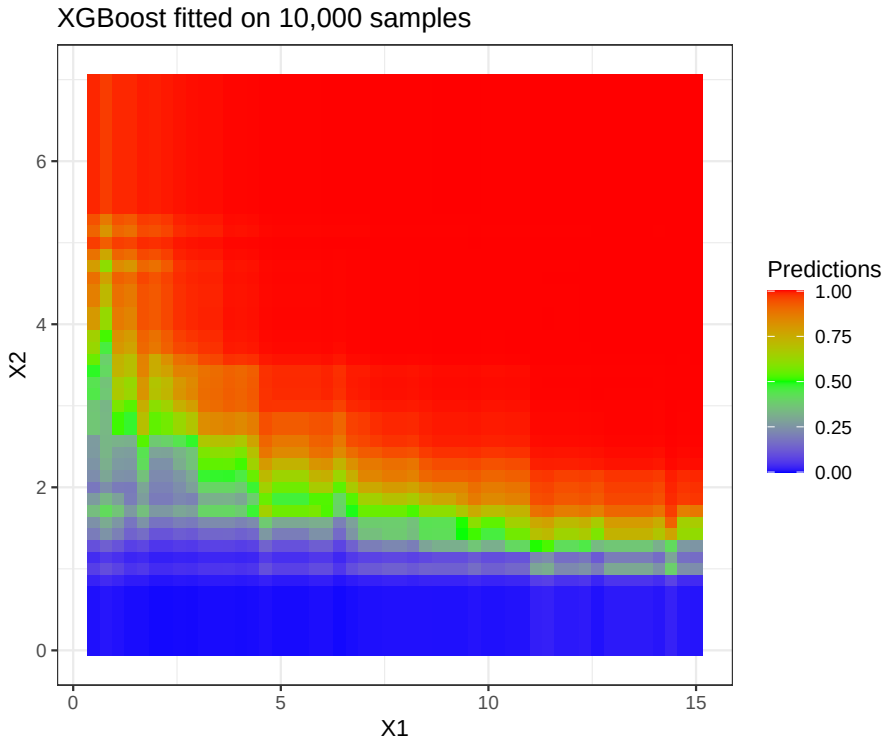

Figure 5: XGBoost becoming better at approximating the regression as the sample size increases. Here the predictions of XGBoost are visualized for a training sample size of 10,000. They appear to be more stable in comparison to Figure 4.

5.1 Discrete super learner

By applying the discrete super learning algorithm to the library, we will see that the discrete super learner is able to qualitatively assess and select the best learning algorithm to

apply given the amount of data at hand. The discrete super learner uses the main effects logistic regression in the beginning with few training samples, but as the predictions of XGBoost become more stable with more observations, it shifts its preference towards XGBoost. The performance of the discrete super learner will be compared with the learning algorithms in the library on the simulated dataset. We show that:

1. As the sample size increases, the discrete super learner achieves the minimum risk
2. The variance of the discrete super learner's prediction for a single new observation does not always exhibit a steady descent and depends on the learners in the library

The discrete super learner in our simulations uses 10-fold cross-validation and the internal loss function will be the quadratic loss. While it is possible to do repeated K-fold cross-validation, in our simulations the super learner will only run the cross-validation procedure once given the training data. The following snippet is pseudocode for the discrete super learner algorithm:

Algorithm 1 Discrete super learner

Input: P_n: dataset, V: number of folds, Ψ: library of learning algorithms of size k, L: loss function
Output: discrete super learner $\hat{\psi}_n(P_n)$
$s \leftarrow$ create random folds(P_n, V) ▷ Randomly assign observations to folds
$\ell \leftarrow$ empty array of dimensions $V \times k$ ▷ Array of risks
for $s \in \{1, \ldots, V\}$ **do**
    $P^1_{n,s} \leftarrow \{O_i \in P_n \mid s(i) = s\}$
    $P^0_{n,s} \leftarrow P_n \setminus P^1_{n,s}$
    for $\psi \in \Psi$ **do**
        $\psi(P^0_{n,s}) \leftarrow \text{fit}(\psi, P^0_{n,s})$
        $\ell[s,\psi] \leftarrow R(\psi(P^0_{n,s}), P^1_{n,s}) = \frac{V}{n} \sum_{O_i \in P^1_{n,s}} L(O_i, \psi(P^0_{n,s}))$
    end for
end for
for $\psi \in \Psi$ **do**
    $\ell_{\text{avg}}(\psi) \leftarrow \frac{1}{V} \sum_{s=1}^{V} \ell[s,\psi]$
end for
$\hat{\psi}_n \leftarrow \arg\min_{\psi \in \Psi} \ell_{\text{avg}}(\psi)$
$\hat{\psi}_n(P_n) \leftarrow \text{fit}(\hat{\psi}_n, P_n)$
return $\hat{\psi}_n(P_n)$

Discrete super learner performance

The true regression achieves a risk of 0.059 using the quadratic loss function on a validation set of size 10^6. This risk represents the minimum achievable risk for any learning algorithm operating on this dataset. The intercept model – which is equivalent to predicting an average of the outcomes – achieves a risk of 0.11, serves as a basic benchmark if no effort was made to learn from the data at all. If any learning algorithm performs worse than the benchmark, then it should naturally be discarded from the library.

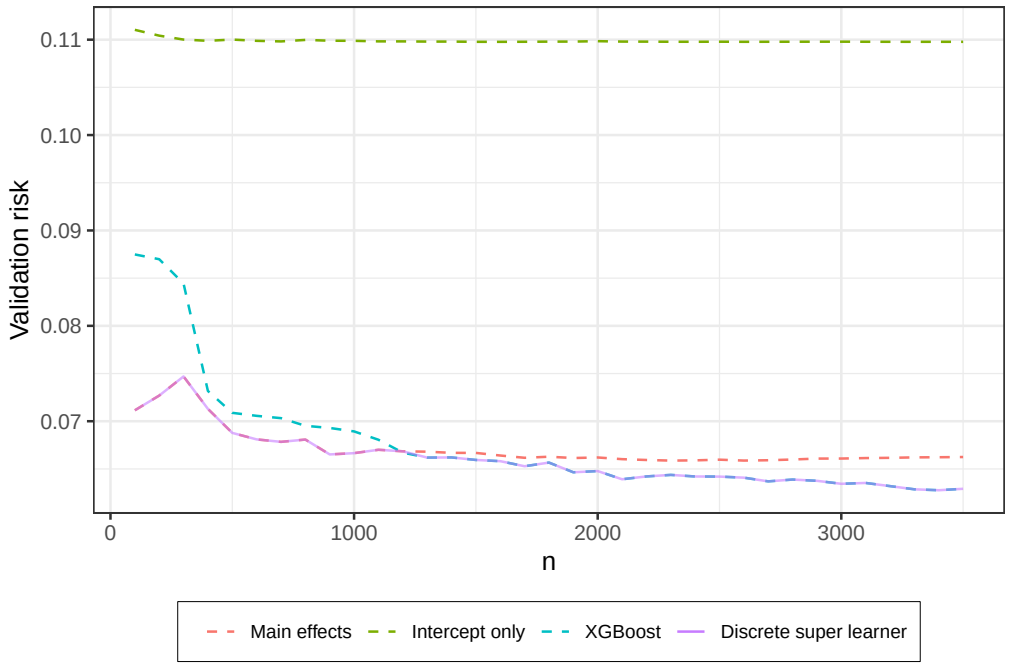

Figure 6: The risk of the discrete super learner compared to other algorithms where the number of training samples are $n = 100, 200, \ldots, N = 3{,}500$. For each n, the algorithms are fitted on n training samples and their validation risks are calculated on a validation set of 10^6 observations.

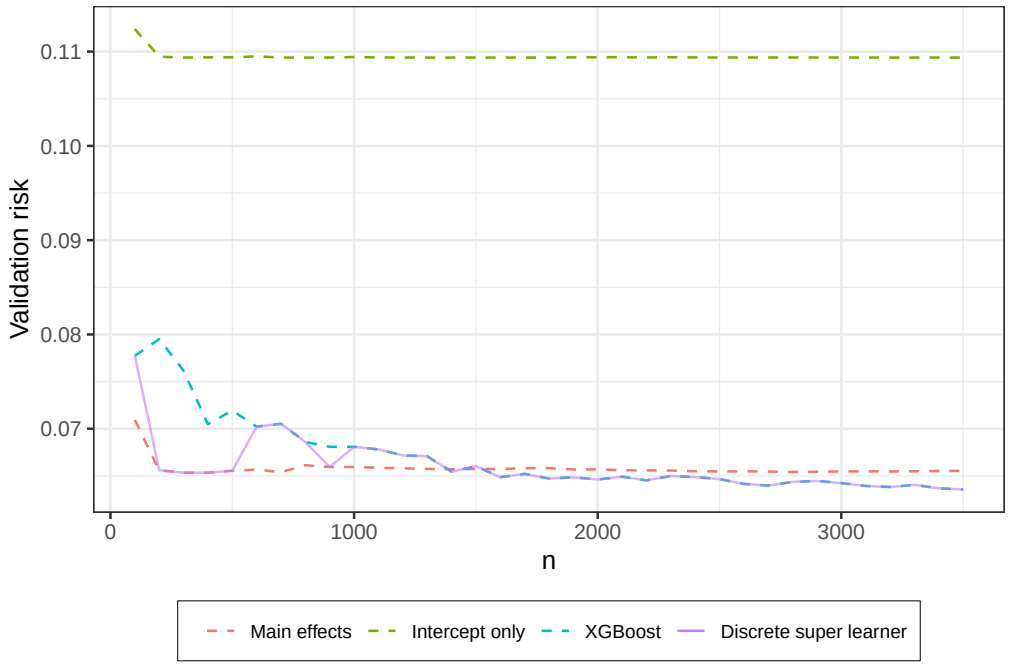

Figure 7: Same experiment as illustrated in Figure 6 but in another run. The training data is drawn from the data-generating distribution and is therefore subject to seed randomness. The resulting fitted learners will not be the same in the two runs.

Figures 6 and 7 demonstrate the performance of the discrete super learner compared to other learners, based on validation risk over the number of training samples. These plots are produced for two runs where the algorithms were fitted on $n = 100, 200, \ldots, N = 3{,}500$ observations, as indicated by the horizontal axis. The validation risk is then computed by evaluating each fitted learner on a fixed validation set of 10^6 observations sampled from the data-generating distribution.

The risk for XGBoost is approximately 0.062 when it is trained on 3,500 observations. Although this exceeds the optimal risk of 0.059, it surpasses the main effects model that has a risk of 0.066. XGBoost evidently outperforms the intercept model, which has a risk of 0.11.

The first run provides an excellent demonstration of how the discrete super learner attains the minimum risk. For small training samples, the machine learning method XGBoost has a higher risk than the main effects logistic regression, making the latter a preferable choice for the discrete super learner despite it being misspecified. As a result, the discrete super learner initially achieves the same risk as the logistic regression, but for $n >$ 1,200 its risk becomes lower. At this point, XGBoost begins to achieve a lower risk than the misspecified logistic regression, leading the discrete super learner to favor XGBoost.

The second run reveals that the discrete super learner might struggle to determine the learner with the lowest risk when the training sample size is small, causing it to move in a zig-zag pattern between two learners with quite similar risks. However, as the number of training samples increases, the discrete super learner ultimately selects XGBoost.

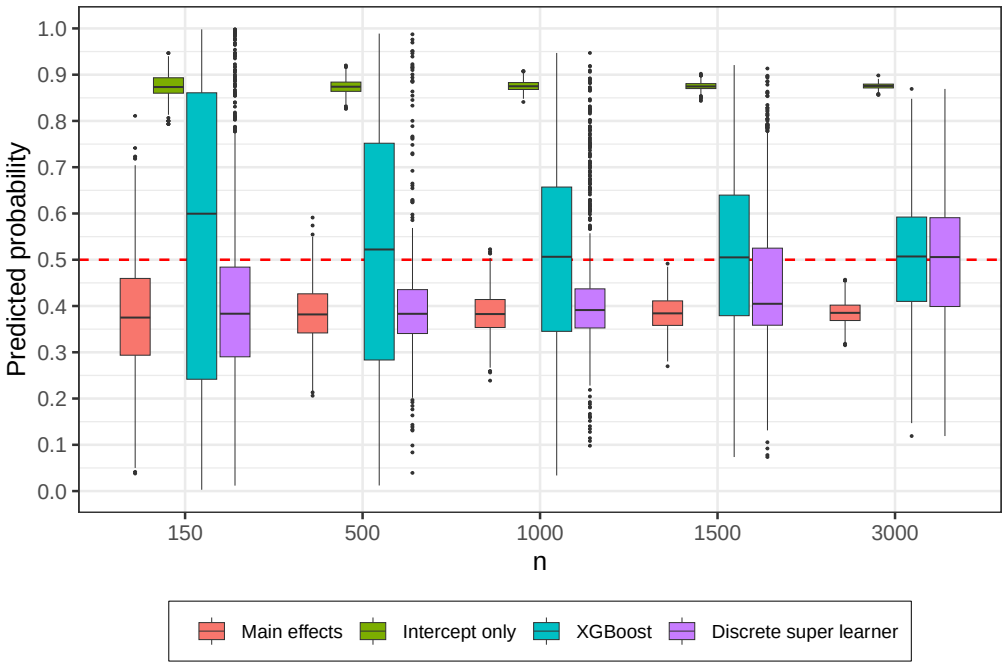

Figure 8: Variances of learner predictions for a single observation. Each algorithm is fitted $K =$ 1,000 times on n samples. Each time they are fitted, they are used to predict on a single fixed observation whose true probability is 0.5.

Figure 8 illustrates the variance in the predictions by each learner for a single observation, whose true probability is indicated by the red dashed line at $p = 0.5$. Each learner has been trained $K =$ 1,000 times on $n = 150, 500, \ldots$ 3,000 samples taken from the dis-

tribution and is used to predict K times after each training. The box plots are created from the K predictions.

We observe that the machine learning model, XGBoost, has the highest prediction variance across all training sample sizes. Recall that we only had two covariates, here having 1,500 observations limits the range of predictions of our main effects model to be between 0.27 and 0.46, whereas for XGBoost it can vary from below 0.1 to above 0.9. While XGBoost is extremely efficient at minimizing loss, its predictions have a high variance unless one has a lot of training data.

Another interesting observation is the bias-variance tradeoff. The parametric learners display low variance, but are highly biased as we see that their predictions are not centered around the true probability at all. XGBoost is less biased on the other hand, but suffers from a high variance.

The discrete super learner achieves a lower variance than XGBoost for all training sample sizes, which can be attributed to its preference of the low variance main effects logistic regression when the training sample size is small. However, the variance of the discrete super learner increases when $n \geq 1{,}000$ and the box plots become wider. This is due to the discrete super learner's preference of XGBoost as it begins to achieve minimal risk around these number of training samples.

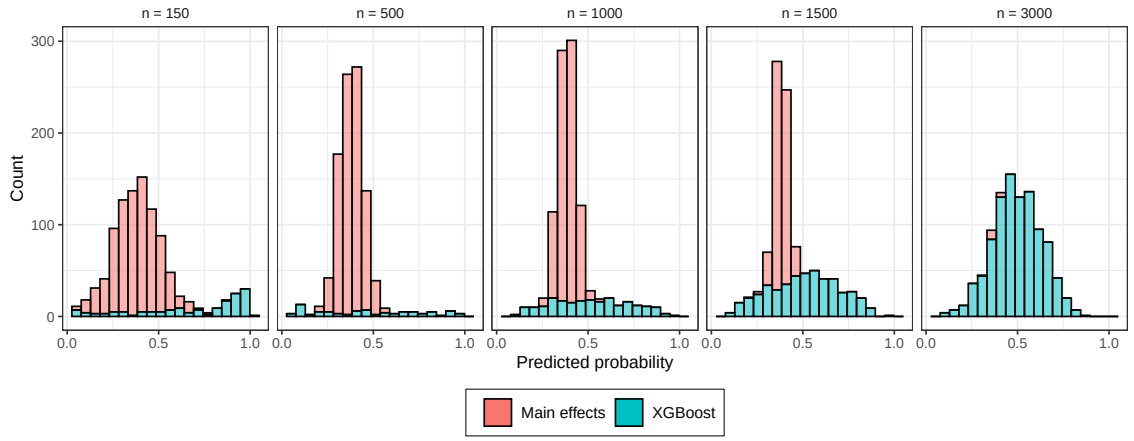

Figure 9: Predictions for a single observation by the discrete super learner varied over the number of training samples n. The predictions are stratified and stacked according to the algorithm that was selected by the discrete super learner.

Figure 9 illustrates how the predictions by the discrete super learner evolves with the number of training samples. The outliers in the discrete super learner box plots from Figure 8 become evident in this figure. The conclusion here is a reiteration of the points made earlier. In the beginning for $n = 150$ to $n = 500$ the discrete super learner favors the predictions made by the main effects logistic regression, resulting in a spike in the distribution around $p = 0.4$. As the number of training samples increases to $n \geq 1{,}000$, the discrete super learner assigns more weight to the predictions made by XGBoost, leading to a distribution centered closer to $p = 0.5$. Somewhere between $n = 1{,}500$ and $n = 3{,}000$, the discrete super learner becomes confident that XGBoost offers the lowest risk and begins to mirror its predictions.

5.2 Ensemble super learner

We will compare the performance of the ensemble super learner with the discrete super learner on the simulated dataset from before. We show that by using the constrained regression meta learning algorithm:

1. The ensemble super learner achieves lower risk than the discrete super learner
2. The ensemble super learner has a lower prediction variance on a single new observations than the discrete super learner

The ensemble super learner will use the same library of algorithms as the discrete super learner, namely the intercept only logistic regression (baseline), the main effects logistic regression and the gradient boosting algorithm XGBoost. The ensemble super learner does internal 10-fold cross-validation similarly to the discrete super learner, except that the out-of-fold predictions are saved in order to create the level 1 dataset. The following snippet is the pseudocode for the ensemble super learner algorithm:

Algorithm 2 Ensemble super learner

Input: P_n: dataset, V: number of folds, Φ: meta learning algorithm, Ψ: library of learning algorithms of size k
Output: ensemble super learner $\Sigma(P_n)$
$\mathcal{L}_n \leftarrow$ empty array of dimensions $n \times (k+1)$ ▷ Level 1 data
$s \leftarrow$ create random folds(P_n, V) ▷ Randomly assign observations to folds
for $s \in \{1, \ldots, V\}$ **do**
    $P^1_{n,s} \leftarrow \{O_i \in P_n \mid s(i) = s\}$
    $P^0_{n,s} \leftarrow P_n \setminus P^1_{n,s}$
    for $\psi \in \Psi$ **do**
        $\psi(P^0_{n,s}) \leftarrow \text{fit}(\psi, P^0_{n,s})$
        for $(Y_i, X_i) = O_i \in P^1_{n,s}$ **do**
            $\mathcal{L}_n[i, 1] \leftarrow Y_i$
            $\mathcal{L}_n[i, \psi] \leftarrow Z_{i,\psi} = \psi(P^0_{n,s})(X_i)$ ▷ Save out-of-fold predictions
        end for
    end for
end for
$\phi \leftarrow \text{fit}(\Phi, \mathcal{L}_n)$ ▷ Fit meta learning algorithm on level 1 data
for $\psi \in \Psi$ **do**
    $\psi(P_n) \leftarrow \text{fit}(\psi, P_n)$ ▷ Fit each learning algorithm on all data
end for
$\Sigma(P_n) \leftarrow (x \mapsto \phi(\psi_1(P_n)(x), \psi_2(P_n)(x), \ldots, \psi_k(P_n)(x)))$
return $\Sigma(P_n)$

We visualize the predictions made by the ensemble super learner as we have done in Figure 4. Figure 10 displays the predictions of the ensemble super learner fitted on 1,000 and 10,000 observations using the constrained regression meta algorithm. To do constrained regression we use the R package *lsei* (Wang, Lawson, and Hanson, 2020), which is used to solve the quadratic programming problem under constraints.

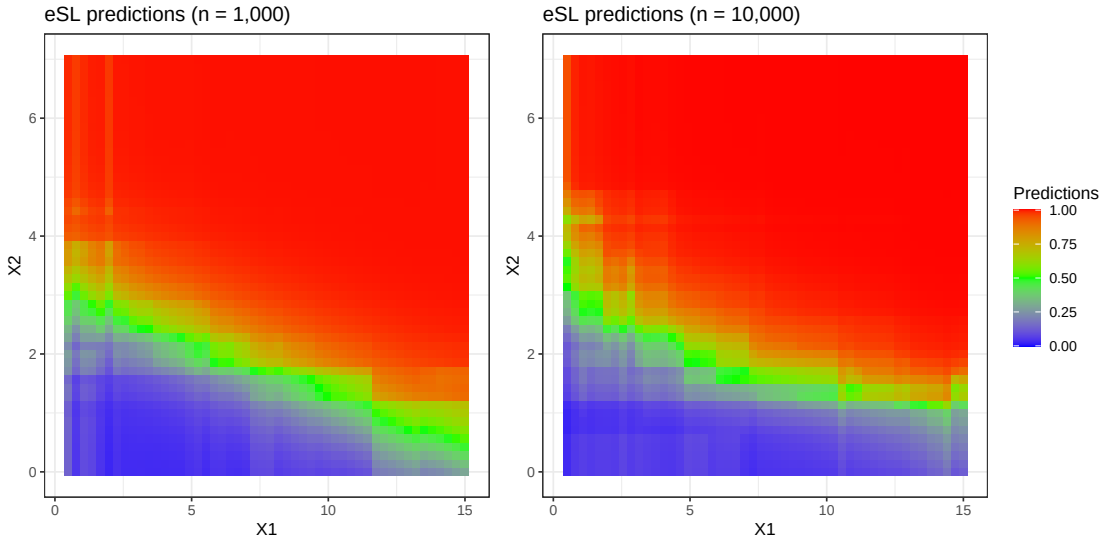

Figure 10: The predictions by the ensemble super learner using the constrained regression meta learning algorithm fitted on 1,000 and 10,000 observations.

The predictions of the ensemble super learner is a weighted combination of the predictions made by the learners in the library. The contribution from XGBoost is apparent in the patchy pattern that characterizes tree-based algorithms which we touched on before. However, we see that there is a clear gradient in the predictions for $n = 1{,}000$ when transitioning from $p < 0.5$ to $p > 0.5$, this is likely due to the contribution from the main effects model, which predicts $p = 0.5$ along a negatively sloped line through the plot. The weights of the constrained regression can be extracted, in this simulation with $n = 1{,}000$ and $n = 10{,}000$ the fitted weights are:

Learner	$n = 1{,}000$	$n = 10{,}000$
Intercept	0.0086	0.0001
Main effects	0.5985	0.1366
XGBoost	0.3929	0.8633

Table 1: Table of ensemble super learner weights of the different learners

From the weights we see that the predictions of intercept only logistic regression not important at all for the ensemble super learner. For $n = 1{,}000$ the predictions of the main effects model have the highest weight, which is not surprising in light of our investigations with the discrete super learner, where we saw that the main effects model was more likely to be selected for small samples. However, for $n = 10{,}000$ it weighs a lot less than XGBoost. Note that these weights depend on the data, so they are not necessarily the same when the data is sampled each time.

Ensemble super learner performance

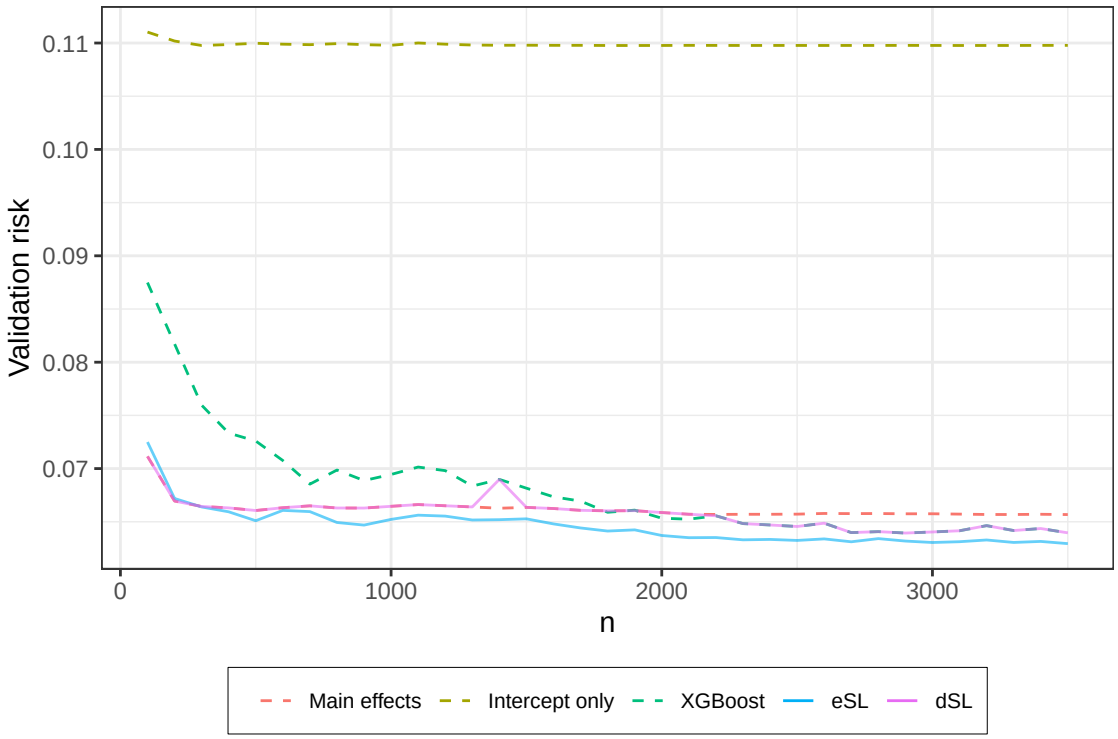

Figure 11: The risk of the ensemble super learner compared to other algorithms where the number of training samples are $n = 100, 200, \ldots, N = 3{,}500$.

Figure 11 demonstrates that the ensemble super learner achieves a lower risk than any of the individual learners, including the discrete super learner. The result agrees with our argument in Section 4.5, where we posited that because the ensemble super learner optimizes across the entire k-simplex, its risk is lower compared to the discrete super learner. Nonetheless, it is important to keep in mind that when examining finite samples, the ensemble super learner minimizes risk on the training data, which does not necessarily translate to a lower validation risk.

As an example, suppose that the library includes the true regression. Limited samples might prohibit the ensemble super learner from assigning full weight to it. The discrete super learner which selects the learning algorithm with the lowest cross-validation risk, might choose the true regression, consequently achieving a lower validation risk than the ensemble super learner.

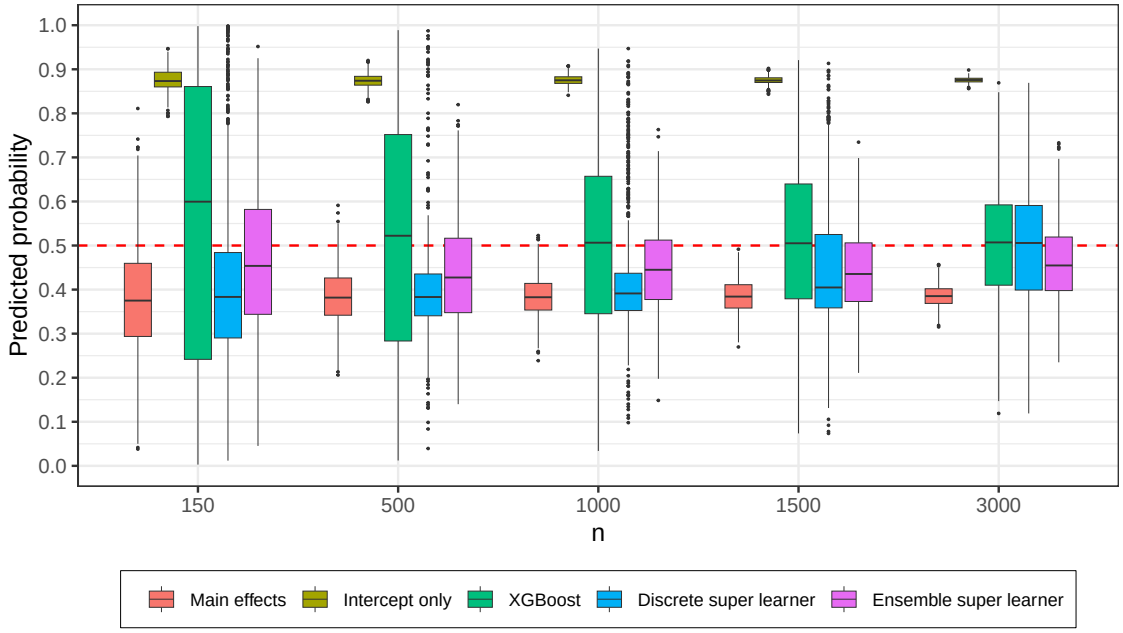

Figure 12: The variance of the ensemble super learner compared to other learners. Each algorithm is fitted $K = 1{,}000$ times on n samples and used to evaluate K times on a single observation.

Figure 12 compares the variances of the ensemble super learner with those of the other learners. Unlike the discrete super learner, which tends to predict many outliers, the predictions of the ensemble super learner have a more centered distribution, as depicted in Figure 13. This behavior is consistent with the fact that the ensemble super learner is a continuous combination of different learners.

The ensemble super learner appears exhibit a slight bias, albeit less than the parametric learners. Moreover, it maintains lower variance compared to both XGBoost and the discrete super learner, while exhibiting a marginally higher variance than the parametric learners. Thus, it could be argued that the ensemble super learner attempts to balance the bias-variance tradeoff.

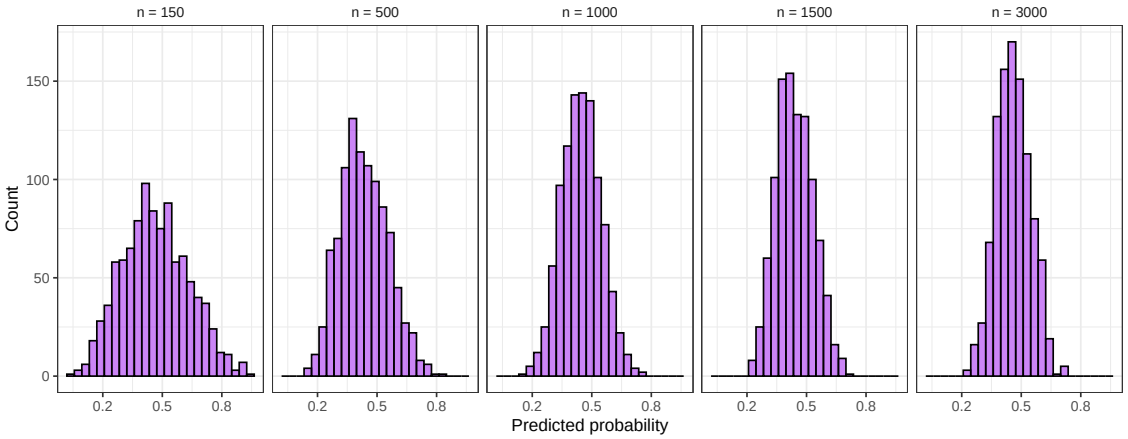

Figure 13: Predictions for a single observation by the ensemble super learner varied over the number of training samples. The histograms display the same data as ensemble super learner box plots in Figure 12.

Figure 13 shows the predictions of the ensemble super learner for a single observation varied over the number of training samples. We see that the distribution of predictions is much more centered and that the variance seems to also decrease steadily with more samples.

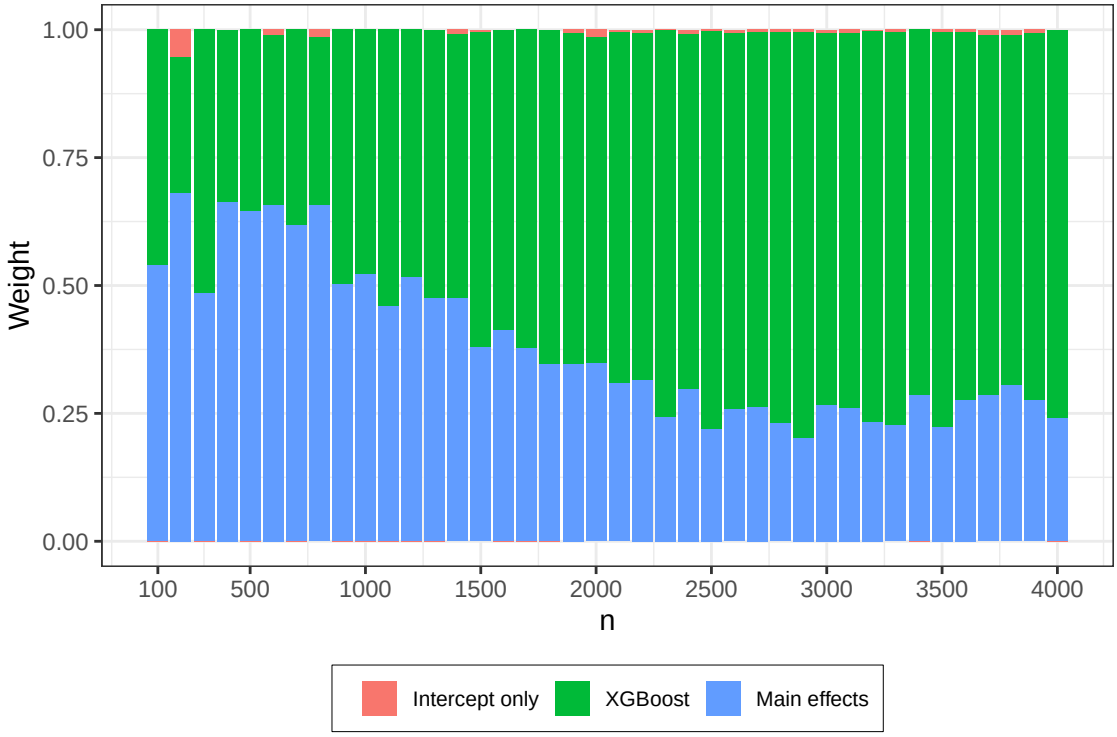

Figure 14: Weighting of the 3 different learners selected by the ensemble super learner varied over the number of training samples. The ensemble super learner uses the constrained regression meta algorithm as described in Section 4.5. The weights sum to 1.

The weighting of the different learners by the ensemble super learner is visualized in Figure 14. We see that the weighting of the intercept only model is very low, and that the main effects model has more weight than XGBoost for small samples. However, this trend reverses as the number of training samples increases. An interesting observation is that XGBoost begins to outweigh the main effects logistic regression when the training size is between 1,000 and 1,500, which coincides with when XGBoost begins to achieve a lower risk compared to the main effects model as seen in Figures 6 and 7.

5.3 Locally weighted ensemble super learner

By using the ensemble super learner with the constrained regression meta learning algorithm we can combine different learners to achieve a lower risk than any of the learners in the library. However, the resulting ensemble is a global combination of the learners, meaning that the weights are the same when predicting on all covariates. In this section I will consider a meta learning algorithm where k-means clustering is used to cluster the covariates into groups and a local weighted combination is fitted for each group.

Motivation

By examining the level 1 data obtained from the K-fold cross-validation, it is possible to compare the predictions of each learner against each other as in Figure 15

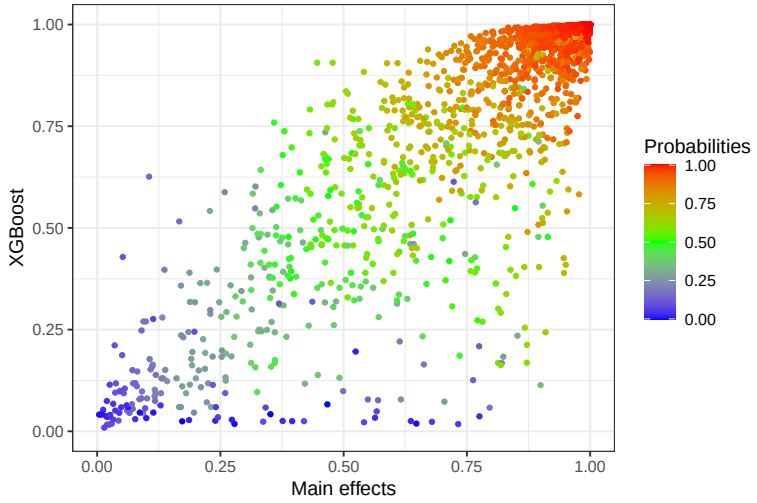

Figure 15: Predictions of XGBoost vs main effects model. The learners' prediction on a single observation represent a point in the unit square, the point is colored by the probability obtained by applying the true regression function on that observation.

We notice that the main effects model does not predict the same as the XGBoost. If they had predicted the same, then the predictions would lie on the identity line. There is significant disagreement between the two algorithms, as seen in the lower right corner where the main effects model predicts certain observations to have a probability of over 0.7, while XGBoost assigns these a probability of less than 0.1. An idea is to group the covariates based on the predicted probabilities such that for certain groups certain algorithms are weighed more than others. The naive approach implements k-means clustering to group the covariates into $k = 4$ clusters as shown in Figure 16

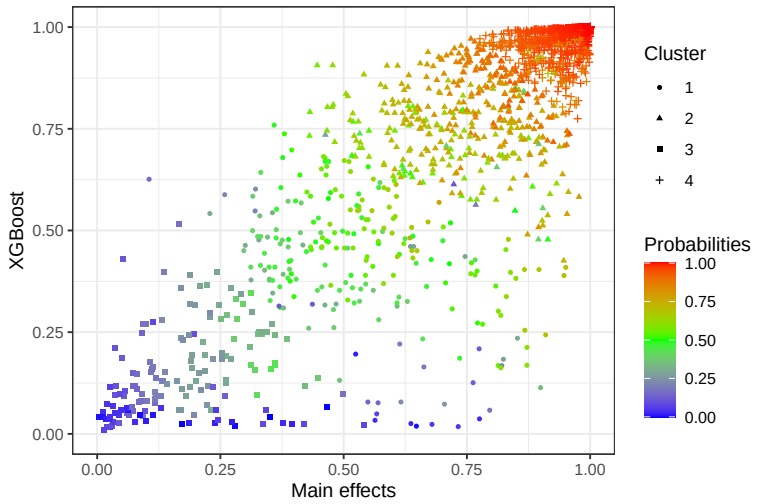

Figure 16: Same as Figure 15, except that the observations are clustered into 4 groups using k-means clustering.

Figures 15 and 16 plot the predictions by two learners against each other, but the procedure is valid also when there are more learners, resulting in clustering within a high dimensional space. For each group, a weighted combination of the learners would be found by fitting on the level 1 data of that group. To predict using the ensemble super learner, we first apply the learners to the new data and determine which of the clusters the level 1 covariates belong to. Once that has been determined, the predictions will be combined according to the weighted combination in that cluster.

Simulation results

The locally weighted ensemble super learner examined in this section is implemented with $k = 4$ clusters and the same library as in the previous section. The number of clusters was chosen arbitrarily and the results are shown in Figure 17

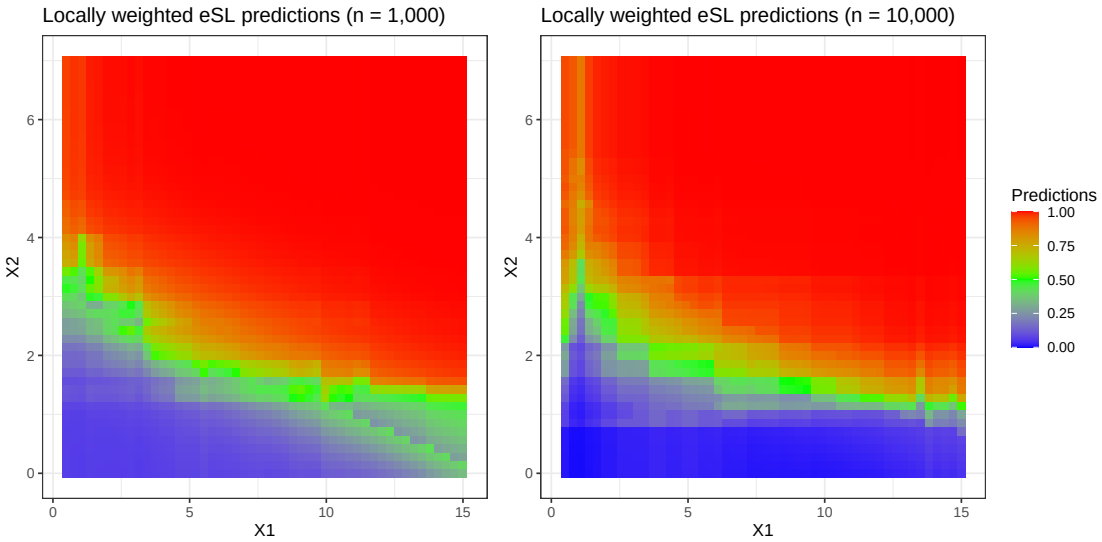

Figure 17: Predictions by the locally weighted ensemble super learner fitted on 1,000 and 10,000 observations.

Figure 18 provides a visual representation of the most weighted learner for each point in the covariate grid, where each prediction come from a cluster following a particular weighting scheme. The visualization partitions predictions into two distinct groups: predictions with probabilities above and below $p = 0.5$.

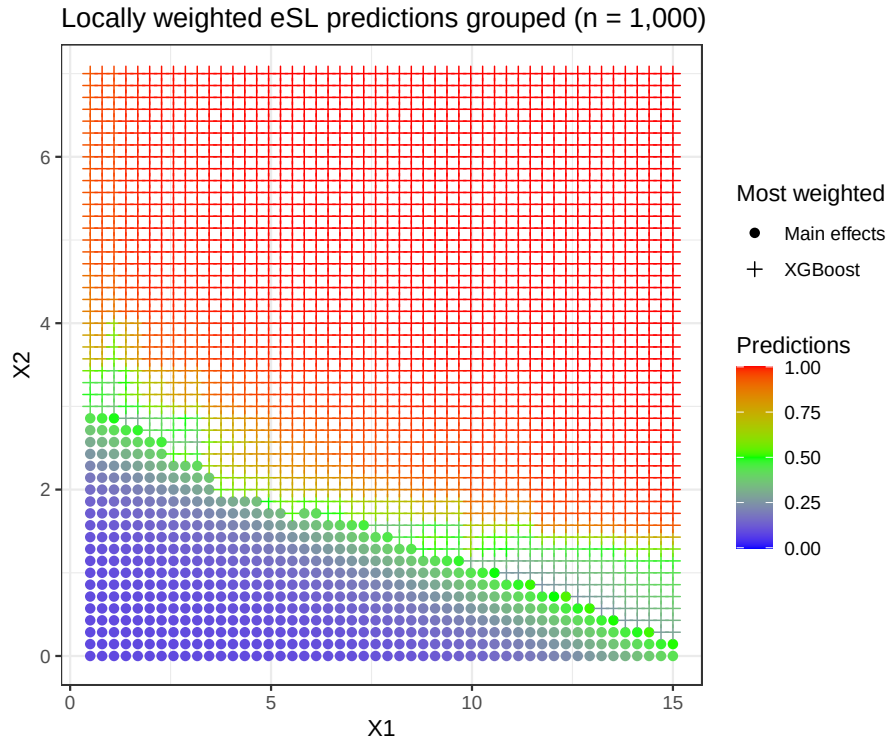

Figure 18: Predictions by the locally weighted ensemble super learner fitted on 1,000 observations from the left panel in Figure 17, grouped according to the learner with the highest weight. Each colored prediction is a weighted sum of the learner predictions.

XGBoost appears to have the highest weight for predictions where the predicted probability is greater than 0.5, whereas the main effects model is favored when the predictions are below 0.5.

The partitioning explains the behavior of the super learner when $n = 1{,}000$ in Figure 17, where it appears that the predictions are smoother for the predicted probabilities less than 0.5. The explanation is that the super learner weighted the logistic regression the most when $p < 0.5$, and since it is a smooth parametric model, the predicted probabilities will assume a gradient. Figure 19 shows that the locally weighted ensemble super learner has a slightly higher risk than the ensemble super learner, but a lower risk than the discrete super learner.

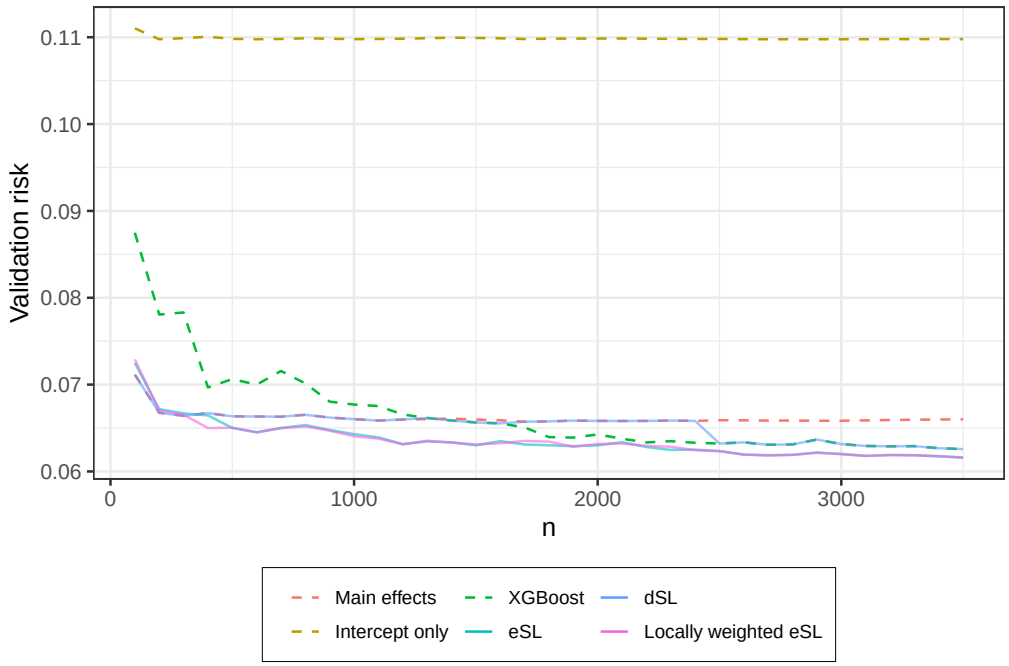

Figure 19: The risk of the locally weighted ensemble super learner compared to other algorithms where the number of training samples are $n = 100, 200, \ldots, N = 3{,}500$ and $k = 4$ local clusters.

The risk of the locally weighted ensemble super learner can depend on the number of k clusters that are used to segregate the level 1 data. The connection between the number of clusters and the risk has not been investigated in this thesis. Nevertheless, the risk of a learner does not tell the full picture as we have witnessed with XGBoost, which, despite its design of minimizing loss, displays the highest variance. It might be relevant to consider other metrics that can evaluate the learners on the stability and variance of their predictions.

6 Discussion

In this thesis super learners and their applicability to binary regression have been investigated. Theoretical results, namely Corollaries 15 and 21, prove that super learners are asymptotically equivalent with the oracle selectors that selects the best learning algorithm based on its knowledge of the data-generating distribution P. We conclude that super learning is a valid method for creating a strong learner by combining a library of different learning algorithms. The two super learners that were examined – the discrete and ensemble super learner – have been benchmarked and compared to a library consisting of logistic regression and XGBoost. They are shown to outperform these algorithms in terms of validation risk. Although the super learner exhibits higher variance than parametric logistic regression, it maintains a lower variance than the machine learning method XGBoost. Finally, a novel meta algorithm for combining learner predictions has been proposed, showing promising results.

6.1 Local weighting of learners

Section 5.3 introduces a novel meta algorithm for combining learner predictions. The locally weighted ensemble super learner is a minor extension of the constrained regression ensemble super learner, which fits a constrained least squares on the level 1 data such that the fitted coefficients are positive and sum to 1. The locally weighted ensemble groups the level 1 data into clusters, and for each cluster the constrained regression problem is solved. The rationale behind this approach is that certain algorithms are likely to perform better than others in specific regions of the level 1 data. Therefore by partitioning the level 1 data one might fit a locally weighted ensemble pertaining to each segregated region. The naive implementation applies k-means clustering to the level 1 data and appears capable of grouping the predictions into clusters as evidenced in Figures 20 to 23 in the appendix.

Another motivation for this procedure is to find a way to 'smoothen' the patchwork pattern in the predictions of tree methods such as XGBoost seen in Figure 4. The patchy pattern in the predictions is undesirable for two reasons. Firstly, the predicted probabilities of neighboring covariates can vary significantly, thus providing a poor model of the actual data-generating distribution. For example, considering a case where one of the covariates is age and the outcome is the presence of a specific disease, it is illogical for the predicted probability of having the disease to be 0.5 when age is < 7 or > 9, but 0 when it lies between 7 and 9. This behavior is inconsistent with the underlying biological process. Secondly, the patchiness suggests that the tree method has not been trained on enough observations, leading to high prediction variance, as seen in Figure 8.

The ensemble super learner should, preferably, combine the predictions from a smooth parametric model with the loss-minimizing machine learning method to create a learner that minimizes loss, but also sanely models the data-generating distribution. The locally weighted ensemble super learner is an attempt at doing that, and the results show promise that this is attainable. By examining Figures 20 and 21 in the appendix, we see that the patchiness is reduced around the edges of the strip where $p = 0.5$ due to the fact that the main effects model is weighted higher in those regions.

Further work is necessary to investigate the performance of the locally weighted ensemble super learner with other learning algorithms and with more covariates. Currently, the k-means method has a hyperparameter k which can be tuned. The results presented in the appendix considers $k = 4$, but it is possible that this number is not optimal. The k-means method also has the downside that it is blind, namely it is an unsupervised method which does not take the outcome into consideration. We observe in Figure 16 that in the lower left corner both the main effects and XGBoost model predicts close to the truth, but k-means clustering groups those predictions together with the predictions in the lower middle where the main effects model predicts worse.

6.2 Grid vs continuous optimization

In Section 4.5 we investigated the constrained regression and how the optimal weighting could be found by optimizing over the $(k-1)$-simplex where k was the size of our library. The oracle result for the ensemble super learner, Corollary 21, requires that we optimize over a parameter set $\mathcal{A}_n$ where the number of elements in that set is at most polynomial in n. In practice, the dependency of parameter set $\mathcal{A}_n$ on n does not reflect our actual optimization procedure. We do not choose a grid of points growing polynomially in the number of observations, compute the risk for each point, and then select the minimum. Rather, we optimize over all feasible points, for instance if the risk is convex, then the minimum can sometimes be solved explicitly or there will exist procedures that guarantee

convergence to the minimum.

The reason for the discrete approximation in the oracle results is because we would like to express it as a corollary of the theorem for the discrete super learner, Theorem 14, which requires that the cardinality k of the library Ψ enters into the remainder in the form of a $\log(1+k)$ term. This is not well-defined when k is the cardinality of a continuous set. However, it is mentioned immediately after theorem 2 in Laan, Polley, and Hubbard (2007) that the discrete approximation is asymptotically negligible, meaning that optimizing over the entire set $\mathcal{A}$ will result in asymptotically equivalent procedures.

6.3 Bias-variance tradeoff

The prediction variances displayed in Figures 8 and 12 highlight the bias-variance tradeoff between using parametric models such as logistic regression vs data adaptive methods like XGBoost. Parametric models tend to be very biased, in comparison to machine learning methods that are less biased, but have a high variance. Recall that the quadratic risk factored into the following terms

$$R(\theta, P) = E(\theta(X) - m(X))^2 + E(m(X) - Y)^2 \qquad \theta \in \Theta,$$

where $m(X) = E(Y \mid X)$. The first term is what we essentially minimize by selecting an optimal θ. Note that we can rewrite the first term as follows

$$\begin{aligned} E(\theta(X) - m(X))^2 &= E(\theta(X) - E(\theta(X)))^2 + (E(\theta(X)) - m(x))^2 \\ &= \mathrm{Var}(\theta(X)) + \mathrm{Bias}(\theta(X))^2. \end{aligned}$$

This is also known as the bias-variance decomposition of the quadratic risk (Györfi et al., 2002). By using quadratic loss, the discrete super learner essentially chooses the learner which minimizes bias-variance tradeoff. The choice is made from a comprehensive set that includes both parametric learning algorithms together with non-parametric machine learning methods. However, by using the ensemble super learner it is possible to determine the optimal blend between the different methods.

7 Perspectives

The local weighting procedure investigated in Section 5.3 can be optimized by considering other clustering algorithms that factor the outcome into account. Another approach could be to separate the level 1 into three groups, one above the diagonal, one below, and the one where the predictions are close to the diagonal. Supervised clustering algorithms might be preferred in order to group the predictions into groups that can have qualitative interpretations such as: "In group k, XGBoost has smaller residuals than main effects". The goal here is to create a method that can combine the flexibility of machine learning algorithms while maintaining a consistent model of the data-generating distribution. We see that XGBoost is capable of learning the distribution when the training sample sizes are in the ten thousands (Figure 5). However, depending on the application, that number of observations is too many in comparison to parametric models that achieve a much lower variance despite the number training samples being 10 times less. An optimal super learning algorithm should balance the loss-minimizing property of machine learning methods with the stability seen in parametric models.

8 Appendix

8.1 Proof of lemma 10

Proof. We first note that the minimizing property of $\hat{\psi}_n$ (definition 8) implies that

$$E_{S^n} R(\hat{\psi}_n(P^0_{n,S^n}), P^1_{n,S^n}) \leq E_{S^n} R(\tilde{\psi}_n(P^0_{n,S^n}), P^1_{n,S^n}). \tag{5}$$

We can write (5) as

$$\begin{aligned} E_{S^n} R(\hat{\psi}_n(P^0_{n,S^n}), P^1_{n,S^n}) &\leq (1+2\delta) E_{S^n} R(\tilde{\psi}_n(P^0_{n,S^n}), P) \\ &+ E_{S^n} \frac{1}{\sqrt{n_1}} R(\tilde{\psi}_n(P^0_{n,S^n}), (1+\delta)G^1_{n,S^n} - \delta\sqrt{n_1}P) \\ &- E_{S^n} \frac{1}{\sqrt{n_1}} R(\hat{\psi}_n(P^0_{n,S^n}), (1+\delta)G^1_{n,S^n} + \delta\sqrt{n_1}P). \end{aligned}$$

Indeed, by examining the risk in the second term on the right hand we obtain

$$\begin{aligned} &R(\tilde{\psi}_n(P^0_{n,S^n}), (1+\delta)G^1_{n,S^n} - \delta\sqrt{n_1}P) \\ &\quad = (1+\delta) R(\tilde{\psi}_n(P^0_{n,S^n}), G^1_{n,S^n}) - \delta\sqrt{n_1} R(\tilde{\psi}_n(P^0_{n,S^n}), P) \\ &\quad = (1+\delta) \left[\sqrt{n_1} R(\tilde{\psi}_n(P^0_{n,S^n}), P^1_{n,S^n}) - \sqrt{n_1} R(\tilde{\psi}_n(P^0_{n,S^n}), P) \right] - \delta\sqrt{n_1} R(\tilde{\psi}_n(P^0_{n,S^n}), P) \\ &\quad = (1+\delta)\sqrt{n_1} R(\tilde{\psi}_n(P^0_{n,S^n}), P^1_{n,S^n}) - (1+2\delta)\sqrt{n_1} R(\tilde{\psi}_n(P^0_{n,S^n}), P). \end{aligned}$$

And for the third term

$$\begin{aligned} &R(\hat{\psi}_n(P^0_{n,S^n}), (1+\delta)G^1_{n,S^n} + \delta\sqrt{n_1}P) \\ &\quad = (1+\delta) R(\hat{\psi}_n(P^0_{n,S^n}), G^1_{n,S^n}) + \delta\sqrt{n_1} R(\hat{\psi}_n(P^0_{n,S^n}), P) \\ &\quad = (1+\delta) \left[\sqrt{n_1} R(\hat{\psi}_n(P^0_{n,S^n}), P^1_{n,S^n}) - \sqrt{n_1} R(\hat{\psi}_n(P^0_{n,S^n}), P) \right] + \delta\sqrt{n_1} R(\hat{\psi}_n(P^0_{n,S^n}), P) \\ &\quad = (1+\delta)\sqrt{n_1} R(\hat{\psi}_n(P^0_{n,S^n}), P^1_{n,S^n}) - \sqrt{n_1} R(\hat{\psi}_n(P^0_{n,S^n}), P). \end{aligned}$$

The $\sqrt{n_1}$ in each term will disappear after we multiply with $\frac{1}{\sqrt{n_1}}$. We now add the first and second term on the right hand side

$$\begin{aligned} &(1+2\delta) E_{S^n} R(\tilde{\psi}_n(P^0_{n,S^n}), P) + E_{S^n} \frac{1}{\sqrt{n_1}} R(\tilde{\psi}_n(P^0_{n,S^n}), (1+\delta)G^1_{n,S^n} - \delta\sqrt{n_1}P) \\ &= (1+2\delta) E_{S^n} R(\tilde{\psi}_n(P^0_{n,S^n}), P) + E_{S^n} \left[(1+\delta) R(\tilde{\psi}_n(P^0_{n,S^n}), P^1_{n,S^n}) - (1+2\delta) R(\tilde{\psi}_n(P^0_{n,S^n}), P) \right] \\ &= (1+\delta) E_{S^n} R(\tilde{\psi}_n(P^0_{n,S^n}), P^1_{n,S^n}). \end{aligned}$$

Now by combining all terms on the right hand side we get

$$\begin{aligned} &(1+\delta) E_{S^n} R(\tilde{\psi}_n(P^0_{n,S^n}), P^1_{n,S^n}) \\ &- E_{S^n} \frac{1}{\sqrt{n_1}} \left[(1+\delta)\sqrt{n_1} R(\hat{\psi}_n(P^0_{n,S^n}), P^1_{n,S^n}) - \sqrt{n_1} R(\hat{\psi}_n(P^0_{n,S^n}), P) \right] \\ &= (1+\delta) E_{S^n} R(\tilde{\psi}_n(P^0_{n,S^n}), P^1_{n,S^n}) - (1+\delta) E_{S^n} R(\hat{\psi}_n(P^0_{n,S^n}), P^1_{n,S^n}) + E_{S^n} R(\hat{\psi}_n(P^0_{n,S^n}), P). \end{aligned}$$

By the minimizing property the difference between the second and first term must be positive. Since we are adding a positive number to $E_{S^n} R(\hat{\psi}_n(P^0_{n,S^n}), P)$, it follows that it must be less than whatever is on the right. In the final lemma we replace $\hat{\psi}_n$ and $\tilde{\psi}_n$ by the maximum over Ψ.

□

8.2 Locally weighted ensemble super learner plots

The following plots show the predictions of the locally weighted ensemble, where the raw predictions are shown on the left panel, and the predictions grouped by the most-weighted learner are on the right.

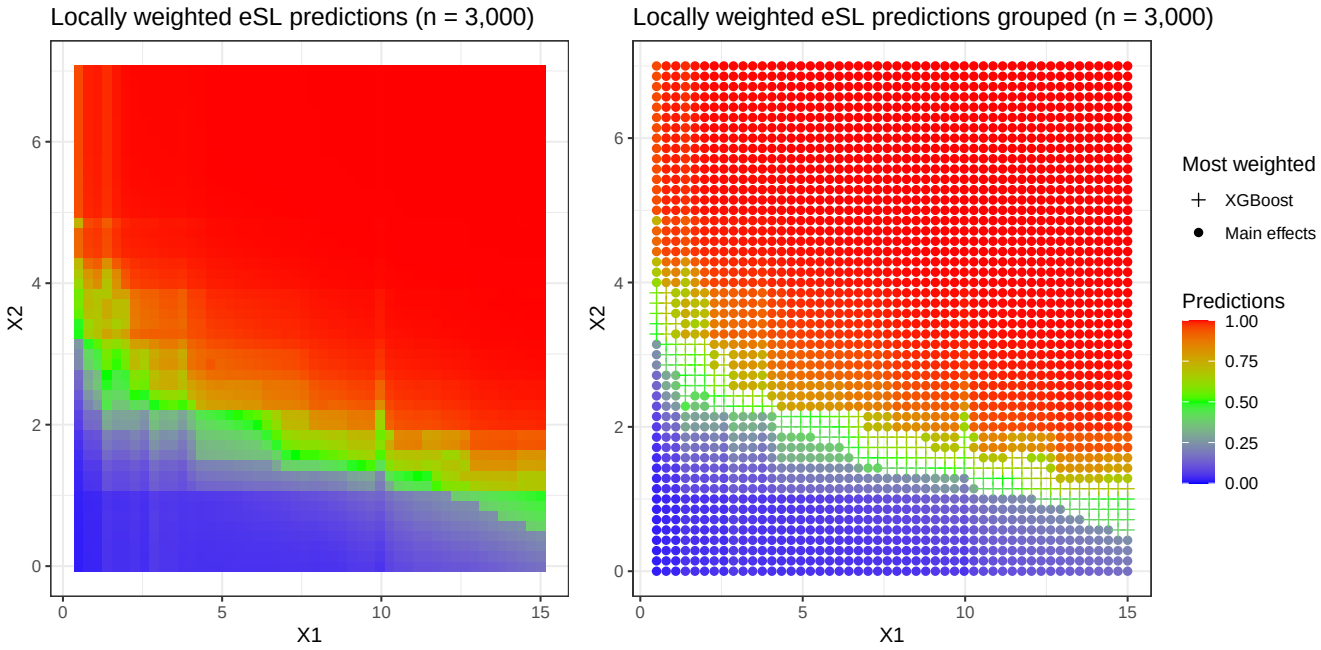

Figure 20: An example where XGBoost seems to be more preferred when the predicted probability is around 0.5.

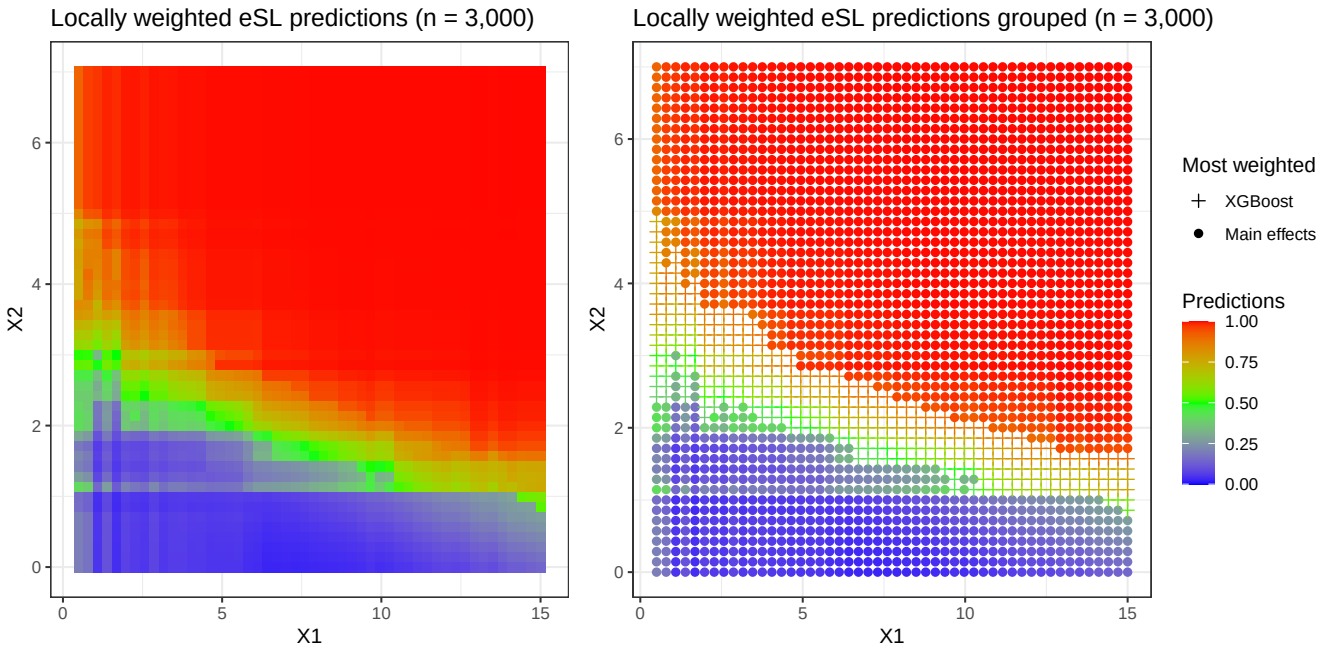

Figure 21: Another example similar to Figure 20

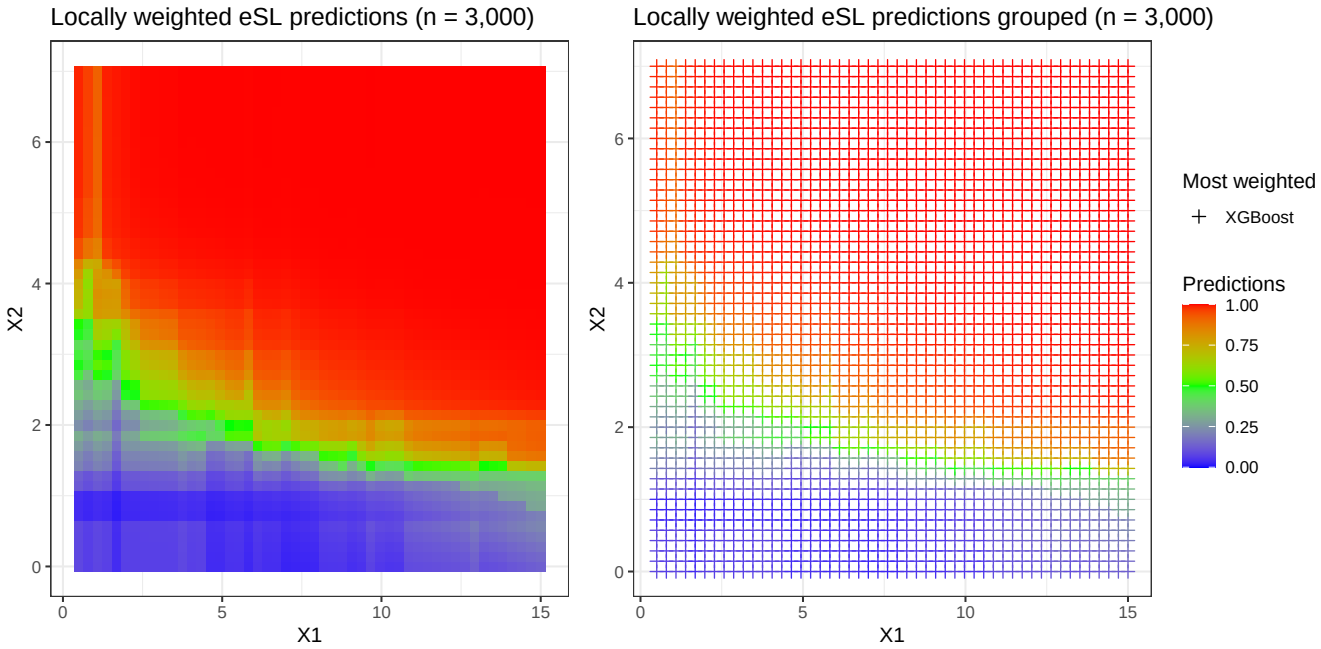

Figure 22: An example where XGBoost is globally the highest weighted learner.

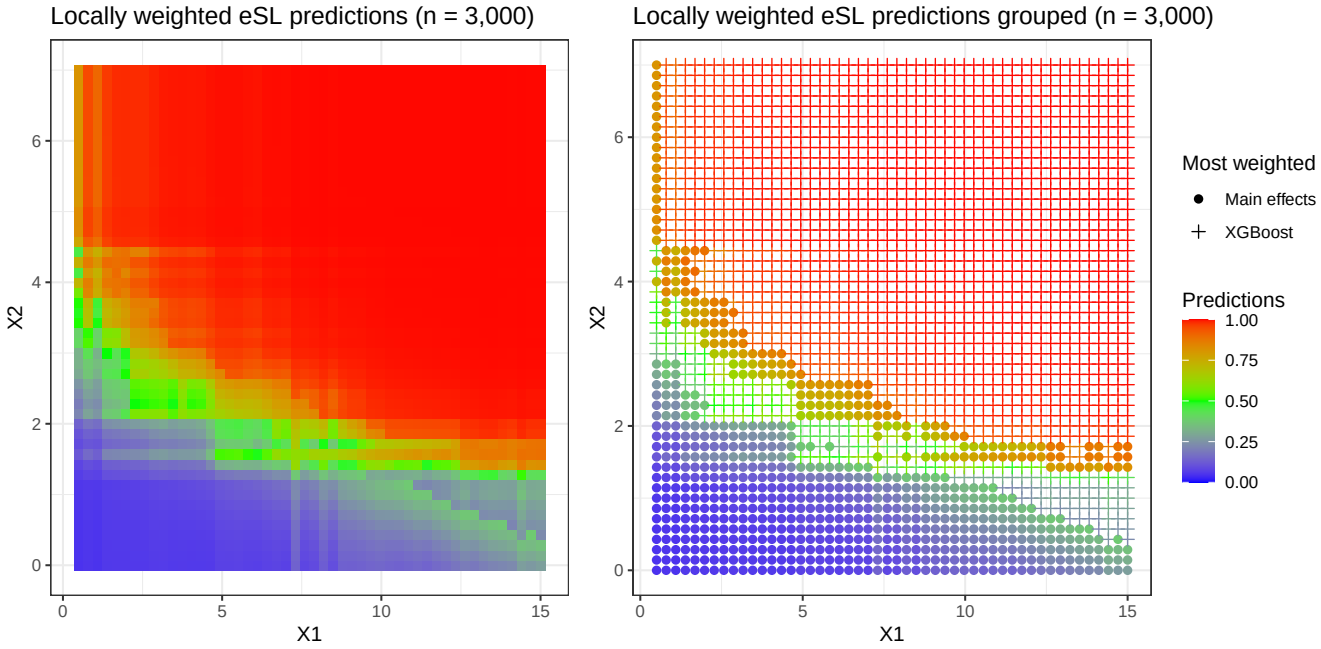

Figure 23: An example where XGBoost is weighted the highest when the predicted probabilities are around 0.5 and above 0.75.

References

[1] Leo Breiman. "Stacked regressions". In: *Machine learning* 24 (1996), pp. 49–64.

[2] László Györfi et al. *A distribution-free theory of nonparametric regression*. Vol. 1. Springer, 2002.

[3] Mark J. van der Laan and Sandrine Dudoit. "Unified Cross-Validation Methodology For Selection Among Estimators and a General Cross-Validated Adaptive Epsilon-Net Estimator: Finite Sample Oracle Inequalities and Examples". In: *UC Berkeley Division of Biostatistics Working Paper Series* (Jan. 2003).

[4] Aad W. van der Vaart, Sandrine Dudoit, and Mark J. van der Laan. In: *Statistics & Decisions* 24.3 (2006), pp. 351–371. DOI: doi:10.1524/stnd.2006.24.3.351. URL: https://doi.org/10.1524/stnd.2006.24.3.351.

[5] Tilmann Gneiting and Adrian E Raftery. "Strictly proper scoring rules, prediction, and estimation". In: *Journal of the American statistical Association* 102.477 (2007), pp. 359–378.

[6] Mark J. van der Laan, Eric C Polley, and Alan E Hubbard. "Super learner". In: *Statistical Applications in Genetics and Molecular Biology* 6.1 (2007).

[7] 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*. 2016, pp. 785–794.

[8] Yong Wang, Charles L. Lawson, and Richard J. Hanson. *lsei: Solving Least Squares or Quadratic Programming Problems under Equality/Inequality Constraints*. R package version 1.3-0. 2020. URL: https://CRAN.R-project.org/package=lsei.

[9] Steffen Lauritzen. *Fundamentals of Mathematical Statistics*. CRC Press, 2023.

[10] Jinyang Liu. *Source code for simulations of super learners*. 2023. URL: https://github.com/jyliuu/bachelor-thesis.