

UC Berkeley

UC Berkeley Electronic Theses and Dissertations

Title

Super Learner

Permalink

<https://escholarship.org/uc/item/4qn0067v>

Author

Polley, Eric

Publication Date

2010

Peer reviewed|Thesis/dissertation

Super Learner

by

Eric Christopher Polley

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Biostatistics

in the

GRADUATE DIVISION
of the
UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:
Professor Mark J. van der Laan, Chair
Professor Alan Hubbard
Professor Ira Tager

Fall 2010

Super Learner

Copyright 2010
by
Eric Christopher Polley

Abstract

Super Learner

by

Eric Christopher Polley

Doctor of Philosophy in Biostatistics

University of California, Berkeley

Professor Mark J. van der Laan, Chair

The super learner is a general loss-based learning method designed to find the optimal combination of a set of learners. The super learner framework is built on the theory of cross-validation and allows for a general class of algorithms to be considered for the ensemble. The oracle results for the cross-validation selector are extended to the super learner. Due to the established oracle results for the cross-validation selector, the super learner is proven to represent an asymptotically optimal framework for learning. We discuss the super learner algorithm and demonstrate the method on a series of data analysis problems. The super learner framework is extended to cover censored data by applying an appropriate observed data loss function. The final chapter presents an R package for implementing the super learner based on a general library of learners.

Professor Mark J. van der Laan
Dissertation Committee Chair

To Mei-Yin.

Contents

List of Figures	iv
List of Tables	v
1 Introduction	1
1.1 Cross-Validation	1
1.1.1 Setting up the estimator selection problem	2
1.1.2 V-fold cross-validation	4
2 Super Learner Algorithm	6
2.1 The proposed super learning algorithm	9
2.1.1 Specific choices of the minimum cross-validated risk predictor.	10
2.2 Finite sample result and asymptotics for the super learner.	12
2.3 Simulation results	15
2.4 Data Analysis	18
2.4.1 Super Learner Results	19
2.5 Discussion	21
2.6 Appendix	21
3 Assessment of Super Learner Algorithm	23
3.1 Introduction	23
3.2 Super Learner	24
3.3 Examples	26
3.3.1 Simulations	26
3.3.2 Data Analysis	32
3.3.3 Array Data	36
3.4 Discussion	37
4 Super Learner for Predicting Treatment Effects	40
4.1 Introduction	40
4.2 Predicting Optimal Treatment Based on Baseline Factors	40
4.3 Super Learner	42

4.4	Extensions for Censored Data	43
4.5	Simulation Example	45
4.6	Example of Prediction Model on Clinical Trial	47
4.6.1	Super learner for optimal treatment decisions	47
4.7	Variable Importance Measure	50
4.8	Discussion	52
5	Super Learner for Censored Data	54
5.1	Introduction	54
5.1.1	Data structure	54
5.1.2	Previous work	55
5.1.3	Cross-validation for censored outcomes	55
5.1.4	Super learner	58
5.2	Super learner for hazard estimation	59
5.3	Data Analysis	62
5.3.1	Lung Cancer Example	62
5.3.2	Breast Cancer Example	66
5.3.3	Selection of loss function	66
5.4	Discussion	70
6	Software to Implement the Super Learner	71
6.1	Using the package	71
6.1.1	Creating prediction algorithms	71
6.1.2	Example: creating prediction wrapper	77
6.2	Creating screening algorithm wrappers	80
6.3	How to set up the SuperLearner library	81
6.4	Example: Boston Housing Data	82
6.5	SuperLearner	82
6.6	Example: ALL Data	85
6.7	Computing Environment	87
7	Conclusion and Future Work	90
	Bibliography	91
A	R functions	99
A.1	Estimation functions	99
A.2	Wrapper functions	122
A.3	Non-negative least squares algorithm	163

List of Figures

1.1	diagram of V-fold cross-validation	5
2.1	Flow Diagram for Super Learner	11
3.1	Scatterplots of the four simulations. The solid line is the true relationship. The points represent one of the simulation samples of size $n = 100$	27
3.2	Scatterplots of the four simulations. The solid line is the true relationship. The points represent one of the simulated datasets of size $n = 100$. The dashed line is the super learner fit for the shown dataset.	29
3.3	10-fold cross-validated relative mean squared error compared to glm across 13 real datasets. Sorted by the geometric mean, denoted with the plus (+) sign.	35
4.1	Flow diagram for super learner	44
4.2	Histograms from 1000 bootstrap samples for $\Psi'(W = \bar{w})$ and $\Psi(W = \bar{w})$. The number in the title bar refers to the method used.	49
5.1	V-fold cross-validation diagram	57
5.2	Super learner diagram	59
5.3	Marginal Kaplan Meier plot for the lung cancer survival times shown with a 95% confidence interval at each time point. (+) sign indicates a censoring time.	63
6.1	Histogram of median home values (in 1000 USD)	83

List of Tables

2.1	R Packages for Candidate Learners.	7
2.2	Simulation Example 1: Estimates of the relative mean squared prediction error (compared to least squares) based on a learning sample of 500 observations and the evaluation sample $M=10,000$. The estimates for β in the super learner are also reported in the right column ($\alpha_n = -0.018$).	16
2.3	Simulation Example 2: Estimates of the relative mean squared prediction error (compared to Least Squares) based on a learning sample of 200 observations and the evaluation sample $M=5,000$. The estimates for β in the super learner are also reported in the right column ($\alpha_n = 0.03$).	17
2.4	Simulation Example 3: Estimates of the relative mean squared prediction error (compared to least squares) based on the validation sample. The 3 new studies from 2.5 are combined with the second simulation example and the relative mean squared prediction error is reported in the overall column.	18
2.5	Super learner results for the diabetes data set. Least Squares (1) and LARS (1) refer to the main effects only models. Least Squares (2) and LARS (2) refer to the all possible two-way interaction models. Relative 10-fold Honest Cross-Validation risk estimates, compared to main terms least squares (RCV risk) are reported. β_n in the super learner is reported in the last column ($\alpha_n = -6.228$).	20
2.6	Super learner results for the HIV data set. Relative 10-fold honest cross validated risk estimates (RCV risk) compared to least squares are reported. β_n in the super learner is reported in the last column ($\alpha_n = 0.027$).	20
3.1	Library of prediction algorithms for the simulations and citation for the corresponding R package.	30
3.2	Results for all 4 simulations. Average R-squared based on 100 simulations and the corresponding standard errors.	31

3.3	Simulation results for example 2. Average mean squared error and R-squared based on 100 simulations and the corresponding standard errors.	33
3.4	Description of data sets. n is the sample size and p is the number of covariates. All examples have a continuous outcome.	34
3.5	Additional prediction algorithms in the library for the real data examples to be combined with the algorithms from Table 3.1.	34
3.6	20-fold cross-validated mean squared error for each algorithm and the standard error for the breast cancer study.	38
3.7	20-fold cross-validated mean squared error for each algorithm and the standard error for the prostate cancer study.	39
4.1	R Packages for Candidate Learners.	46
4.2	Risk for all candidate learners and the super learner	47
4.3	Number of subjects in each treatment arm at enrollment and available for each method.	47
4.4	10-fold honest cross validation estimates of $E(\Psi_n(W) - Z)^2$ for the super learner and each of the candidate learners on their own.	48
4.5	Top 5 effect modifiers based on univariate regression, lars, and super learner with targeted maximum likelihood. The standard error was based on a bootstrap with 1,000 samples.	51
5.1	Honest 10-fold cross-validated risk estimates for the super learner, each algorithm in the library, and the Weibull proportional hazards model. All algorithms included time as a covariate, the second column (Time) denotes if any additional smoothing for time was part of the given estimator and the value for the degrees of freedom if used.	65
5.2	Both Trials combined into a single data set. Summary statistics for the full data and the 3 treatment groups, count and percentage for categorical variables, mean and standard deviation for age. Treatment A is lumpectomy only, treatment B is lumpectomy plus radiation therapy, treatment C is lumpectomy plus radiation therapy plus Tamoxifen	67
5.3	Trial 17, count and percentage for categorical variables (mean and sd for age)	68
5.4	Trial 24, count and percentage for categorical variables (mean and sd for age)	69
5.5	Honest 10 fold cross validated risk estimates for the super learner and each algorithm in the library. The loss function is the 10 year IPCW squared error loss on the survival probability.	70
6.1	Details of prediction algorithm wrappers	73
6.1	Details of prediction algorithm wrappers	74
6.1	Details of prediction algorithm wrappers	75

6.1	Details of prediction algorithm wrappers	76
6.2	The allowed arguments for a prediction algorithm in SuperLearner .	77
6.3	The allowed arguments for a screening algorithm in SuperLearner . .	81
6.4	20-fold cross-validated risk estimates for the super learner, the discrete super learner and each algorithm in the library	88

Acknowledgments

This dissertation could not have been completed without the help of many individuals. I would first like to thank Mei for sticking with me on this journey. She has been a constant source of support throughout my time at Berkeley. I would also like to thank my parents, Bill and Diane, for their support over the many years of perpetual student life.

I could not have asked for a better faculty advisor than Mark van der Laan. These last few years I have gained incredible insight from both coursework and mentorship with Mark.

I want to thank my committee members Alan Hubbard, Ira Tager, and Steve Selvin for all the thoughtful discussions and for being there to brainstorm the next big ideas with. I look forward to continual collaborations with everyone.

I have always enjoyed teaching and being at Berkeley has given me the opportunity to work with some amazing teachers. Specifically, I would like to thank Maureen Lahiff, Sophia Rabe-Hesketh, and Steve Selvin for giving me the opportunity to work with their courses.

Bonnie Hutchings and Sharon Norris deserve special thanks for making sure I was always on the right track.

Chapter 1

Introduction

One of the most commonly used statistical procedures is estimator selection. Given a set of observations and a collection of candidate learners, an estimator selection procedure traditionally selects a single estimator from the collection based on a criterion (e.g. minimizing the negative log likelihood). Numerous methods exist for the candidate learners, an overview of various statistical methods for the candidate learners can be found in [44].

The super learner is based on the general theory of loss-based estimation [87]. Therefore, the framework applies to a variety of estimation problems including density estimation and prediction. As long as the parameter of interest can be defined as the minimizer of the expected loss function, the super learner framework will apply.

Before introducing the super learner, a short description of cross-validation is presented. The second chapter contains the proposed super learner algorithm and the oracle results. Chapter 3 contains a collection of practical demonstrations of the super learner on both simulated and real data sets. Chapter 4 is an extended example of the super learner for the specific case of predicting treatment effects. Chapter 5 outlines how the super learner framework can be extended to censored data. Chapter 6 contains a detailed description of the software implementation of the super learner framework. Finally, Chapter 7 contains the conclusion and future work.

1.1 Cross-Validation

The super learner is motivated by an aggressive use of cross-validation. In this section, the general framework of cross-validation is introduced as this will be the foundation of the super learner. These details will be used in defining the super learner but is also helpful in understanding the motivation for the super learner. A concise overview on the current state of cross-validation can be found in [3].

It has been well established that using the same data to training a learner and to

evaluate the performance can be misleading and overly optimistic. Given that most real world data applications have a limited supply of data the idea of data splitting became popular. Part of the data was used to train the learner and the other part was used to evaluate the performance. The idea of splitting the data was pushed further with the realization that the splitting could occur repeatedly even with the same data and the performance evaluations averaged over the repeated splits. These general methods of splitting the data became known as cross-validation [82, 36].

1.1.1 Setting up the estimator selection problem

Let $X_i = (Y_i, W_i)$, $i = 1, \dots, n$ be n i.i.d. observations of $X \sim P_X$, where P_X is an element of a statistical model $\mathcal{M}$. The data contains two components, an outcome $Y \in \mathbb{R}$ (continuous or discrete) and a p -dimensional covariate vector $W = (W(j) : j = 1, \dots, p) \in \mathbb{R}^p$. The statistical model, $\mathcal{M}$, is a set of possible distributions. With the data generating distributions $P \in \mathcal{M}$ we define a parameter mapping $\Psi : \mathcal{M} \rightarrow \mathcal{D}(\mathcal{X})$ from the model space into the space $\mathcal{D}(\mathcal{X})$ of real-valued functions defined on the $(p+1)$ -dimensional Euclidean set $\mathcal{X} \subseteq \mathbb{R}^{(p+1)}$. The parameter is a realization of this mapping for a given P , $\psi \equiv \Psi(P)$. Therefore, the parameter is itself a mapping, $\psi : \mathcal{X} \rightarrow \mathbb{R}$ and we can define the parameter space by the collection of data generating distributions, $\Psi = \{\Psi(P) : P \in \mathcal{M}\} \subseteq \mathcal{D}(\mathcal{X})$.

The parameter is often defined with respect to a loss function. A loss function is a mapping, $L : (X, \psi) \rightarrow L(X, \psi) \in \mathbb{R}$, from the observations $X \sim P$ and a parameter value $\psi \in \Psi$ into a real-value. Often the interest is in the expected loss, the risk, of a parameter value with respect to P :

$$\Theta(\psi, P) = \int L(x, \psi) dP(x) = E_X [L(X, \psi)] \quad (1.1)$$

The risk is minimized at ψ_0

$$\psi_0 = \arg \min_{\psi \in \Psi} \Theta(\psi, P) \quad (1.2)$$

For prediction, the interest is in the regression function, $\psi(W) = E(Y \mid W)$, and the loss function is often the squared error loss $L(X, \psi) = (Y - \psi(W))^2$. An alternative loss function is the negative log (likelihood).

Previous work on the prediction problem in the special case of linear models focused on penalty terms on the loss function. For example Mallows's C_p [61] involves a penalty term on the sum of squared errors and Akaike's information criterion (AIC) [2] places a penalty on the log likelihood. Both penalties are linear in the number of parameters in the linear model. Although widely used for estimator selection, both these methods have been shown to not account for the data-driven nature of the selection [7, 9, 13]. An alternative to the penalty approach is to apply sample re-use methods such as leave-one-out cross-validation, multi-fold cross-validation, or

the bootstrap. Early work on sample re-use methods are [82, 83, 36]. These seminal papers focused mostly leave-one-out cross-validation. Stone even went so far as to claim (erroneously) that the results would only hold for leave-one-out and not multi-fold:

I conjecture that Geisser is on the wrong track in leaving out more than one item at a time, that whatever diluted optimality theorems exist in this area will require the $n - 1$ to 1 split.[82, page 146]

Most results for cross-validation are specific to the varieties of multi-fold cross-validation and require both the training and validation sample sizes to grow to infinity asymptotically.

Various types of cross-validation have been proposed in the literature, below is a list of the more commonly used methods:

- sample split or hold-out [21]
- multi-fold or V-fold [36, 12]
- leave-one-out [82, 83, 36]
- leave- p -out [78]
- monte-carlo [68]
- bootstrap [24, 26]
- penalized V-fold [4]

leave-one-out cross-validation has many practical and theoretical shortcomings and hence most work on cross-validation has focused on multi-fold. Cross-validation in this paper will usually refer to V-fold cross-validation.

At the core of all types of cross-validation is a random splitting of the learning data into training and validation sets. The split vector is defined as:

$$B_n(i) = \begin{cases} 0, & X_i \text{ in Training set} \\ 1, & X_i \text{ in Validation set} \end{cases} \quad (1.3)$$

Let $P_n, P_{n,B_n}^0, P_{n,B_n}^1$ be the empirical distributions of the learning set, the training set and the validation set, respectively. Define the number of observations in the validation split as $n_1 = \sum B_n(i)$ and the proportion as $p_n = n_1/n$

The cross-validated risk estimator for $\psi_n = \hat{\Psi}(P_n)$

$$\hat{\theta}_{p_n,n} = E_{B_n} \Theta \left(\hat{\Psi}(P_{n,B_n}^0, P_{n,B_n}^1) \right) \quad (1.4)$$

$$= E_{B_n} \frac{1}{n_1} \sum_{i: B_n(i)=1} L(X_i, \hat{\Psi}(P_{n,B_n}^0)) \quad (1.5)$$

where $\hat{\Psi}(P_{n,B_n}^0)$ denotes the estimator of the parameter using only the corresponding training data.

1.1.2 V-fold cross-validation

V-fold cross-validation randomly partitions the learning set into V mutually exclusive and exhaustive sets of approximately equal size. Define the random splitting vector as:

$$B_n(i) = \begin{cases} 0, & X_i \text{ in Training set} \\ 1, & X_i \text{ in Validation set} \end{cases}$$

The distribution of B_n is a discrete distribution with mass $1/V$ on the binary vectors $b_n^\nu = (b_n^\nu(i) : i = 1, \dots, n)$ for $\nu = 1, \dots, V$. Where the random vectors b_n^ν have the constraints $\sum_i b_n^\nu(i) \approx n/V, \forall \nu$ and $\sum_\nu b_n^\nu(i) = 1, \forall i$ but are independent of X .

For each ν split, the validation split is removed and the estimator is fit on the remaining data. Using the fit of the estimator from the training data, the performance is assessed on the validation data.

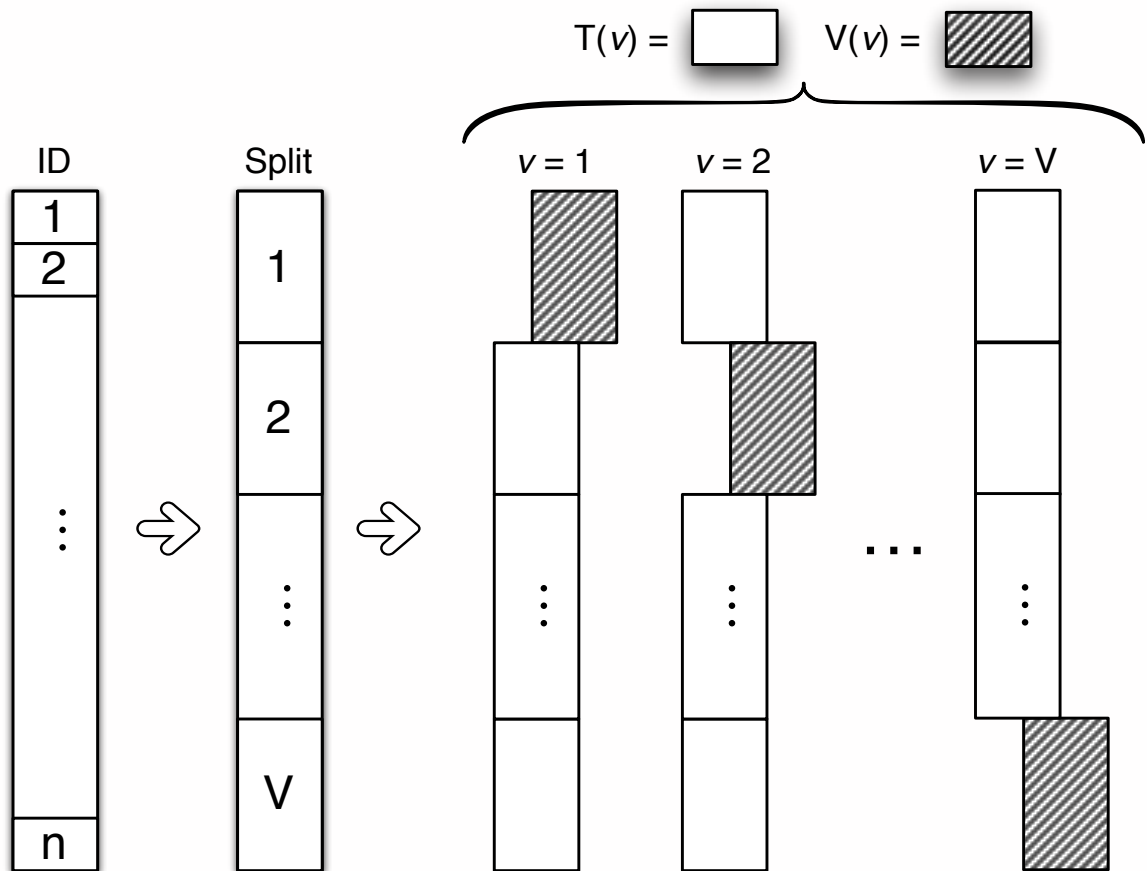


Figure 1.1: diagram of V-fold cross-validation

Chapter 2

Super Learner Algorithm

Numerous methods exist to learn from data the best predictor of a given outcome based on a sample of n independent and identically distributed observations $O_i = (Y_i, X_i)$, Y_i the outcome of interest, and X_i a vector of input variables, $i = 1, \dots, n$. A few examples include decision trees, neural networks, support vector regression, least angle regression, logic regression, poly-class, Multivariate Adaptive Regression Splines (MARS), and the Deletion/Substitution/Addition (D/S/A) algorithm. Such learners can be characterized by the mechanism used to search the parameter space of possible regression functions. For example, the D/S/A algorithm [81] uses polynomial basis functions, while logic regression [76] constructs Boolean expressions of binary covariates. The performance of a particular learner depends on how effective its searching strategy is in approximating the optimal predictor defined by the true data generating distribution. Thus, the relative performance of various learners will depend on the true data-generating distribution. In practice, it is generally impossible to know *a priori* which learner will perform best for a given prediction problem and data set. To solve the problem, some researchers have proposed combining learners in various methods and have exhibited better performance over a single candidate learner [31, 41], but there is concern that these methods may over-fit the data and may not be the optimal way to combine the candidate learners.

The framework for unified loss-based estimation [87] suggests a solution to this problem in the form of a new learner, termed the “super learner”. In the context of prediction, this learner is itself a prediction algorithm, which applies a set of candidate learners to the observed data, and chooses the optimal learner for a given prediction problem based on cross-validated risk. Theoretical results show that such a super learner will perform asymptotically as well as or better than any of the candidate learners [87, 88].

To be specific, consider some candidate learners. Least Angle Regression (LARS) [25] is a model selection algorithm related to the lasso. Logic Regression [76] is an adaptive regression methodology that attempts to construct predictors as Boolean combinations of binary covariates. The D/S/A algorithm [81] for polynomial regres-

Method	R Package	Authors
Least Angle Regression	lars	Hastie and Efron
Logic Regression	LogicReg	Kooperberg and Ruczinski
D/S/A	DSA	Neugebauer and Bullard
Regression Trees	rpart	Therneau and Atkinson
Ridge Regression	MASS	Venables and Ripley
Random Forests	randomForest	Liaw and Wiener
Adaptive Regression Splines	polspline	Kooperberg

Table 2.1: R Packages for Candidate Learners.

sion data-adaptively generates candidate predictors as polynomial combinations of continuous and/or binary covariates, and is available as an R package at <http://www.stat.berkeley.edu/users/laan/Software/>. Classification and Regression Trees (CART) [12] builds a recursive partition of the covariates. Another candidate learner is random forests [11], which is a random bootstrap version of the regression tree. Ridge Regression [46] minimizes a penalized least squares with a penalty on the L_2 norm of the parameter vector. Multivariate Adaptive Regression Splines (MARS) [34] is an automated model selection algorithm which creates a regression spline function. Table 2.1 contains citations of R packages for each of the candidate learners. All of these methods have the option to carry out selection using v -fold cross-validation. The selected fine-tuning parameter(s) can include the ratio of the L_1 norm of the coefficient vector in LARS to the norm of the coefficient vector from least squares; the number of logic trees and leaves in Logic Regression; and the number of terms and a complexity measure on each of the terms in D/S/A.

Cross-validation divides the available learning set into a training set and validation set. Observations in the training set are used to construct (or train) the learners, and observations in the validation set are used to assess the performance of (or validate) these learners. The cross-validation selector selects the learner with the best performance on the validation sets. In v -fold cross-validation, the learning set is divided into v mutually exclusive and exhaustive sets of as nearly equal size as possible. Each set and its complement play the role of the validation and training sample, respectively, giving v splits of the learning sample into a training and corresponding validation sample. For each of the v splits, the estimator is applied to the training set, and its risk is estimated with the corresponding validation set. For each learner the v risks over the v validation sets are averaged resulting in the so-called cross-validated risk. The learner with the minimal cross-validated risk is selected.

It is helpful to consider each learner as an algorithm applied to empirical distributions. Thus, if we index a particular learner with an index k , then this learner can be represented as a function $P_n \rightarrow \hat{\Psi}_k(P_n)$ from empirical probability distributions P_n to functions of the covariates. Consider a collection of $K(n)$ learners $\hat{\Psi}_k$,

$k = 1, \dots, K(n)$, in parameter space Ψ . The super learner is a new learner defined as

$$\hat{\Psi}(P_n) \equiv \hat{\Psi}_{\hat{K}(P_n)}(P_n),$$

where $\hat{K}(P_n)$ denotes the cross-validation selector described above which simply selects the learner which performed best in terms of cross-validated risk. Specifically,

$$\hat{K}(P_n) \equiv \arg \min_k E_{B_n} \sum_{i, B_n(i)=1} (Y_i - \hat{\Psi}_k(P_{n, B_n}^0)(X_i))^2,$$

where $B_n \in \{0, 1\}^n$ denotes a random binary vector whose realizations define a split of the learning sample into a training sample $\{i : B_n(i) = 0\}$ and validation sample $\{i : B_n(i) = 1\}$. Here P_{n, B_n}^1 and P_{n, B_n}^0 are the empirical probability distributions of the validation and training sample, respectively.

The aggressive use of cross-validation is inspired by the theorem 3.1 in [88]. The theorem is provided in the appendix.

The “oracle” selector is defined in Theorem 1 in the appendix as the estimator, among the $K(n)$ learners considered, which minimizes risk under the true data-generating distribution. In other words, the oracle selector is the best possible estimator given the set of candidate learners considered; however, it depends on both the observed data and P_0 , and thus is unknown.

This theorem shows us that the super learner performs as well (in terms of expected risk difference) as the oracle selector, up to a typically second order term. Thus, as long as the number of candidate learners considered ($K(n)$) is polynomial in sample size, the super learner is the optimal learner in the following sense:

- If, as is typical, none of the candidate learners (nor, as a result, the oracle selector) converge at a parametric rate, the super learner performs asymptotically as well (in the risk difference sense) as the oracle selector, which chooses the best of the candidate learners.
- If one of the candidate learners searches within a parametric model and that parametric model contains the truth, and thus achieves a parametric rate of convergence, then the super learner achieves the almost parametric rate of convergence $\log n/n$.

Organization: The current article builds and extends this super learning methodology. In section 2 we will describe our new proposal for super learning, also using an initial set of candidate learners and cross-validation as above, but now allowing for semi-parametric families of the candidate learners, and formulating the minimization of cross-validated risk as another regression problem for which one can select an appropriate regression methodology (e.g involving cross-validation or penalized regression). This is an important improvement relative to our previous super learning proposal by 1) extending the set of initial candidate learners into a large family of

candidate learners one obtains by combining the initial candidate learners according to a parametric or semi-parametric model, thereby obtain a potentially much more flexible learner, and 2) by controlling over-fitting of the cross-validated risk through the use of data adaptive regression algorithms using cross-validation or penalization itself. Importantly, these gains come at no cost regarding computing time. In Section 3 we investigate the practical performance of this new super learning algorithm based on simulated as well as a number of real data sets.

2.1 The proposed super learning algorithm

Suppose one observes n i.i.d. observations $O_i = (X_i, Y_i) \sim P_0$, $i = 1, \dots, n$, and the goal is to estimate the regression $\psi_0(X) = E_0(Y | X)$ of $Y \in \mathcal{Y}$ on $X \in \mathcal{X}$. The regression can be defined as the minimizer of the expectation of the squared error loss function:

$$\psi_0 = \arg \min_{\psi} E_0 L(O, \psi),$$

where $L(O, \psi) = (Y - \psi(X))^2$. The proposed super learner immediately applies to any parameters that can be defined as minimizers of a loss function $L(O, \psi)$ over a parameter space Ψ , but the article focuses on the prediction problem using the squared error loss function.

Let $\hat{\Psi}_j$, $j = 1, \dots, J$, be a collection of J candidate learners, which represent mappings from the empirical probability distribution P_n into the parameter space Ψ consisting of functions of X .

The proposed super learner uses V -fold cross-validation. Let $v \in \{1, \dots, V\}$ index a sample split into a validation sample $V(v) \subset \{1, \dots, n\}$ and training sample (the complement of $V(v)$) $T(v) \subset \{1, \dots, n\}$, where $V(v) \cup T(v) = \{1, \dots, n\}$. Here we note that the union, $\cup_{v=1}^V V(v) = \{1, \dots, n\}$, of the validation samples equals the total sample, and the validations samples are disjoint: $V(v_1) \cap V(v_2) = \emptyset$ for $v_1 \neq v_2$. For each $v \in \{1, \dots, V\}$, let, $\psi_{njv} \equiv \hat{\Psi}_j(P_{nT(v)})$ be the realization of the j^{th} -estimator $\hat{\Psi}_j$ when applied to the training sample $P_{nT(v)}$.

For an observation i , let $v(i)$ denote the validation sample it belongs to, $i = 1, \dots, n$. We now construct a new data set of n observations as follows: (Y_i, Z_i) , where $Z_i \equiv (\psi_{njv(i)}(X_i) : j = 1, \dots, J)$ is the vector consisting of the J predicted values according to the J estimators trained on the training sample $P_{nT(v(i))}$, $i = 1, \dots, n$. Let $\mathcal{Z}$ be the set of possible outcomes for Z .

Minimum cross-validated risk predictor: Another input of this super learning algorithm is yet another user-supplied prediction algorithm $\tilde{\Psi}$ that estimates the regression $E(Y | Z)$ of Y onto Z based on the data set (Y_i, Z_i) , $i = 1, \dots, n$. For notational convenience, we will denote $\{(Y_i, Z_i) : i = 1, \dots, n\}$ with $P_{n,Y,Z}$, so that $\tilde{\Psi}$ is a mapping from $P_{n,Y,Z}$ to $\tilde{\Psi}(P_{n,Y,Z}) : \mathcal{Z} \rightarrow \mathcal{Y}$, where the latter is a function from $\mathcal{Z}$ to $\mathcal{Y}$. We will refer to this algorithm $\tilde{\Psi}$ as the minimum cross-validated risk

predictor since it aims to minimize the cross-validated risk, $\tilde{\psi} \rightarrow \sum_{i=1}^n (Y_i - \tilde{\psi}(Z_i))^2$, over a set of candidate functions $\tilde{\psi}$ from $\mathcal{Z}$ into $\mathcal{Y}$, although, we allow penalization or cross-validation to avoid over-fitting of this cross-validated risk criteria.

This now defines a mapping $\hat{\Psi}^*$ from the original data $P_n \equiv \{Y_i, X_i\} : i = 1, \dots, n\}$ into the predictor

$$\tilde{\Psi} \left(\{Y_i, Z_i = (\hat{\Psi}_j(P_{nT(v_i)})(X_i) : j = 1, \dots, J) : i = 1, \dots, n\} \right)$$

obtained by applying the cross-validated risk minimizer $\tilde{\Psi}$ to $P_{n,Y,Z} = \{(Y_i, Z_i) : i = 1, \dots, n\}$. Denote $\psi_n^* = \hat{\Psi}^*(P_n)$ as the actual obtained predictor when one applies the learner $\hat{\Psi}^*$ to the original sample P_n . We note that $\psi_n^* \in \Psi^* \equiv \{f : \mathcal{Z} \rightarrow \mathcal{Y}\}$ is a function of Z into the outcome set $\mathcal{Y}$ for Y .

The super learner for a value X based on the data (i.e., P_n) is now given by

$$\hat{\Psi}(P_n)(X) \equiv \hat{\Psi}^*(P_n)((\hat{\Psi}_j(P_n)(X), j = 1, \dots, J). \quad (2.1)$$

In words, the super learner of Y for a value X is obtained by evaluating the predictor $\psi_n^* = \hat{\Psi}^*(P_n)$ at the J predicted values, $\hat{\Psi}_j(P_n)(X)$, at X of the J candidate learners. Figure 2.1 contains a flow diagram for the steps involved in the super learner.

2.1.1 Specific choices of the minimum cross-validated risk predictor.

Parametric minimum cross-validated risk predictor: Consider a few concrete choices that aim to fit a regression of Y onto the J predicted values Z based on the corresponding training samples from (Y_i, Z_i) , $i = 1, \dots, n$ for the algorithm $\hat{\Psi}^*$. Define the cross-validated risk criteria:

$$R_{CV}(\beta) \equiv \sum_{i=1}^n (Y_i - m(Z_i | \beta))^2,$$

where one could use, for example, the linear regression model $m(z | \beta) = \beta z$. If $Y \in \{0, 1\}$, then one could use the logistic linear regression model $m(z | \beta) = 1/(1 + \exp(-\beta z))$, if one allows predictions in the range of $[0, 1]$, or, if one wants a predictor mapping into $\{0, 1\}$, then we can choose $m(z | \alpha_0, \beta) \equiv I(1/(1 + \exp(-\beta z)) > \alpha_0)$ as the indicator that the logistic regression score exceeds a cut-off α_0 . Let $\beta_n = \arg \min_{\beta} R_{CV}(\beta)$ be the least squares or MLE estimator, and let

$$\psi_n^*(z) \equiv m(z | \beta_n).$$

One could also estimate β with a constrained least squares regression estimator such as penalized L_1 -regression (Lasso), penalized L_2 regression (shrinkage), where the

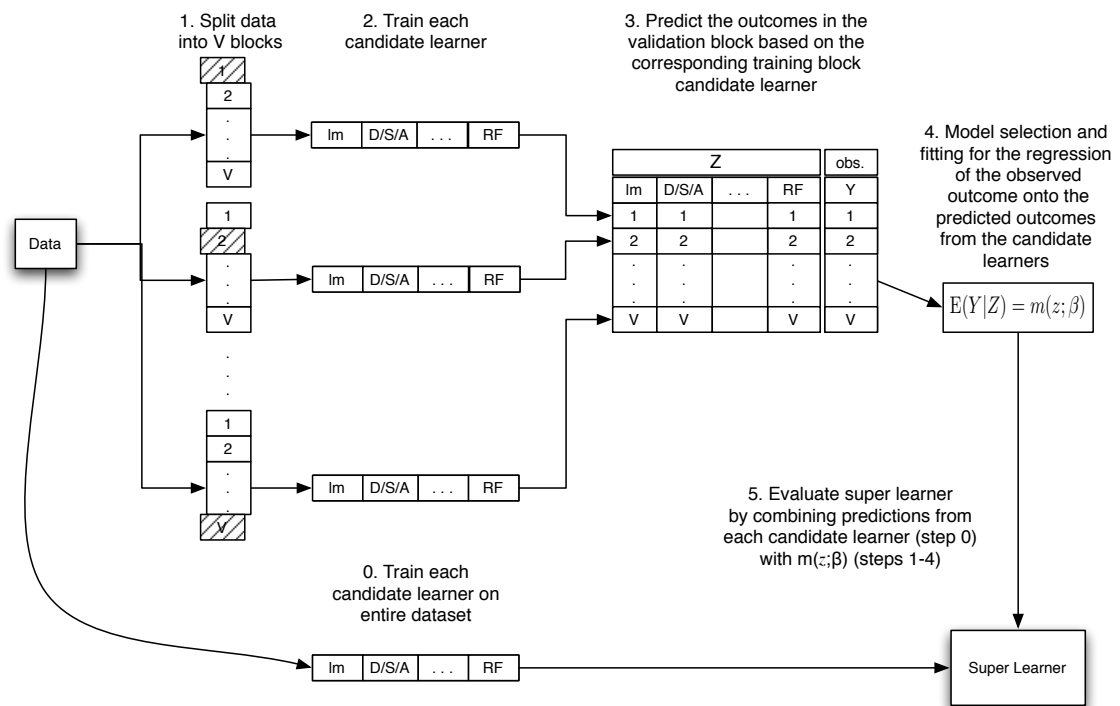


Figure 2.1: Flow Diagram for Super Learner

constraints are selected with cross-validation, or one could restrict β to the set of positive weights summing up till 1.

Data adaptive minimum cross-validated risk predictor: There is no need to restrict ψ_n^* to parametric regression fits. For example, one could define ψ_n^* in terms of the application of a particular data adaptive (machine learning) regression algorithm to the data set (Y_i, Z_i) , $i = 1, \dots, n$, such as CART, D/S/A, or MARS, among others. In fact, one could apply a super learning algorithm itself to estimate $E(Y | Z)$. In this manner one can let the data speak in order to build a good predictor of Y based on covariate vector Z based on (Y_i, Z_i) , $i = 1, \dots, n$.

Thus, this super learner is indexed, beyond the choice of initial candidate estimators, by a choice of minimum cross-validated risk predictor. As a consequence, the proposal provides a whole class of tools indexed by an arbitrary choice of regression algorithm (i.e., ψ_n^*) to map a set of candidate learners into a new cross-validated estimator (i.e. super learner). In particular, it provides a new way of using the cross-validated risk function, which goes beyond minimizing the cross-validated risk over a set of candidate learners.

2.2 Finite sample result and asymptotics for the super learner.

An immediate consequence of Theorem 1 above is the following result for the proposed super learner (2.1), which provides for the case that the minimum cross-validated risk predictor is based on a parametric regression model.

Theorem 1. Assume $P((Y, X) \in \mathcal{Y} \times \mathcal{X}) = 1$, where $\mathcal{Y}$ is a bounded set in $\mathbb{R}$, and $\mathcal{X}$ is a bounded Euclidean set. Assume that the candidate estimators map into $\mathcal{Y}$: $P(\hat{\Psi}_j(P_n) \in \mathcal{Y}, j = 1, \dots, J) = 1$.

Let $v \in \{1, \dots, V\}$ index a sample split into a validation sample $V(v) \subset \{1, \dots, n\}$ and corresponding training sample $T(v) \subset \{1, \dots, n\}$ (complement of $V(v)$), where $V(v) \cup T(v) = \{1, \dots, n\}$, and $\cup_{v=1}^V V(v) = \{1, \dots, n\}$. For each $v \in \{1, \dots, V\}$, let, $\psi_{n j v} \equiv \hat{\Psi}_j(P_{n T(v)}), \mathcal{X} \rightarrow \mathcal{Y}$, be the realization of the j -th estimator $\hat{\Psi}_j$ when applied to the training sample $T(v)$.

For an observation i let $v(i)$ be the validation sample observation i belongs to, $i = 1, \dots, n$. Construct a new data set of n observations defined as: (Y_i, Z_i) , where $Z_i \equiv (\psi_{n j v(i)}(X_i) : j = 1, \dots, J) \in \mathcal{Y}^J$ is the J -dimensional vector consisting of the J predicted values according to the J estimators trained on the training sample $T(v(i))$, $i = 1, \dots, n$.

Consider a regression model $z \rightarrow m(z | \alpha)$ for $E(Y | Z)$ indexed by a $\alpha \in \mathcal{A}$ representing a set of functions from $\mathcal{Y}^J$ into $\mathcal{Y}$. Consider a grid (or any finite subset) $\mathcal{A}_n$ of α -values in the parameter space $\mathcal{A}$. Let $K(n) = |\mathcal{A}_n|$ be the number of grid points which grows at most at a polynomial rate in n : $K(n) \leq n^q$ for some $q < \infty$.

Let

$$\alpha_n \equiv \arg \min_{\alpha \in \mathcal{A}_n} \sum_{i=1}^n (Y_i - m(Z_i | \alpha))^2.$$

Consider the regression estimator $\psi_n : \mathcal{X} \rightarrow \mathcal{Y}$ defined as

$$\psi_n(x) \equiv m((\psi_{jn}(x) : j = 1, \dots, J) | \alpha_n).$$

For each $\alpha \in \mathcal{A}$, define the candidate estimator $\hat{\Psi}_\alpha(P_n) \equiv m((\hat{\Psi}_j(P_n) : j = 1, \dots, J) | \alpha)$: i.e.

$$\hat{\Psi}_\alpha(P_n)(x) = m((\hat{\Psi}_j(P_n)(x) : j = 1, \dots, J) | \alpha).$$

Consider the oracle selector of α :

$$\tilde{\alpha}_n \equiv \arg \min_{\alpha \in \mathcal{A}_n} \frac{1}{V} \sum_{v=1}^V d(\hat{\Psi}_\alpha(P_{nT(v)}), \psi_0),$$

where

$$d(\psi, \psi_0) = E_0(L(X, \psi) - L(X, \psi_0)) = E_0(\psi(X) - \psi_0(X))^2.$$

For each $\delta > 0$ we have that there exists a $C(\delta) < \infty$ such that

$$\frac{1}{V} \sum_{v=1}^V Ed(\hat{\Psi}_{\alpha_n}(P_{nT(v)}), \psi_0) \leq (1 + \delta) E \min_{\alpha \in \mathcal{A}_n} \frac{1}{V} \sum_{v=1}^V d(\hat{\Psi}_\alpha(P_{nT(v)}), \psi_0) + C(\delta) \frac{V \log n}{n}.$$

Thus, if

$$\frac{E \min_{\alpha \in \mathcal{A}_n} \frac{1}{V} \sum_{v=1}^V d(\hat{\Psi}_\alpha(P_{nT(v)}), \psi_0)}{\frac{\log n}{n}} \rightarrow 0 \text{ as } n \rightarrow \infty, \quad (2.2)$$

then it follows that the estimator $\hat{\Psi}_{\alpha_n}$ is asymptotically equivalent with the oracle estimator $\hat{\Psi}_{\tilde{\alpha}_n}$ when applied to samples of size $(1 - 1/V)n$:

$$\frac{\frac{1}{V} \sum_{v=1}^V Ed(\hat{\Psi}_{\alpha_n}(P_{nT(v)}), \psi_0)}{E \min_{\alpha \in \mathcal{A}_n} \frac{1}{V} \sum_{v=1}^V d(\hat{\Psi}_\alpha(P_{nT(v)}), \psi_0)} \rightarrow 1 \text{ as } n \rightarrow \infty.$$

If (2.2) does not hold, then it follows that $\hat{\Psi}_{\alpha_n}$ achieves the $(\log n)/n$ rate:

$$\frac{1}{V} \sum_{v=1}^V Ed(\hat{\Psi}_{\alpha_n}(P_{nT(v)}), \psi_0) = O\left(\frac{\log n}{n}\right).$$

Discussion of conditions. The discrete approximation $\mathcal{A}_n$ of $\mathcal{A}$ used in this theorem is typically asymptotically negligible. For example, if $\mathcal{A}$ is a bounded Euclidean set, then the distance between neighboring points on the grid can be chosen as small as $1/n^q$ for some $q < \infty$ so that minimizing a criteria over such a fine grid $\mathcal{A}_n$ versus minimizing over the whole set $\mathcal{A}$ results in asymptotically equivalent procedures. For example, if α is a Euclidean parameter and $\|m(\cdot | \alpha_1) - m(\cdot | \alpha_2)\|_\infty < C \|\alpha_1 - \alpha_2\|$ for some $C < \infty$, where $\|\cdot\|_\infty$ denotes the supremum norm, then it follows that for each $\delta > 0$ we have that there exists a $C(\delta) < \infty$ such that

$$\frac{1}{V} \sum_{v=1}^V Ed(\hat{\Psi}_{\alpha_n}(P_{nT(v)}), \psi_0) \leq (1 + \delta) E \min_{\alpha \in \mathcal{A}} \frac{1}{V} \sum_{v=1}^V d(\hat{\Psi}_\alpha(P_{nT(v)}), \psi_0) + C(\delta) \frac{\log n}{n},$$

where $\alpha_n = \arg \min_{\alpha \in \mathcal{A}} \sum_{i=1}^n (Y_i - m(Z_i | \alpha))^2$. The other conclusions of the theorem now also apply.

This theorem implies that the selected prediction algorithm $\hat{\Psi}_{\alpha_n}$ will either perform asymptotically as well (up till the constant) as the best estimator among the family of estimators $\{\hat{\Psi}_\alpha : \alpha \in \mathcal{A}\}$ when applied to samples of size $n(1 - 1/V)$, or achieve the parametric model rate $1/n$ up till a $\log n$ factor. By a simple argument as presented in [87], [23] and [92], it follows that by letting the $V = V_n$ in the V-fold cross-validation scheme converge to infinity at a slow enough rate relative to n , then either $\psi_n = \hat{\Psi}_{\alpha_n}(P_n)$ performs asymptotically as well (up till the constant) as the best estimator among the estimators $\{\hat{\Psi}_\alpha : \alpha\}$ applied to the full sample P_n , or it achieves the parametric rate of convergence up till the $\log n$ factor.

The take home message of this theorem is that our super learner will perform asymptotically as well as the best learner among the family of candidate learners $\hat{\Psi}_\alpha$ indexed by α . By choosing the regression model $m(\cdot | \alpha)$ so that there exist a α_j so that $m(Z | \alpha_j) = Z_j$ for each $j = 1, \dots, J$ (e.g., $m(Z | \alpha) = \alpha Z$), then it follows, in particular, that the resulting prediction algorithm asymptotically outperforms each of the initial candidate estimators $\hat{\Psi}_j$. More importantly and practically, the set of candidate estimators $\hat{\Psi}_\alpha$ can include interesting combinations of these J estimators which exploit the strengths of various of these estimators for the particular data generating distribution P_0 instead of focusing on one of them. For example, if one uses the linear regression model $m(Z | \alpha) = \alpha Z$, then the candidate estimators $\{\hat{\Psi}_\alpha : \alpha\}$ include all averages of the J estimators, including convex combinations. As becomes evident in our data analysis and simulation results, the selected super learner ψ_n^* based on a linear (or logistic) regression model is often indeed (or logistic function of) a weighted average of competing estimators in which various of the candidate learners significantly contribute to the average.

2.3 Simulation results

In this section, we conducted 3 simulation studies to evaluate the working characteristics of the super learner. These simulations all involve a continuous response variable. For the first simulation, the true model is:

$$Y_i = 2w_1w_{10} + 4w_2w_7 + 3w_4w_5 - 5w_6w_{10} + 3w_8w_9 + w_1w_2w_4 - 2w_7(1 - w_6)w_2w_9 - 4(1 - w_{10})w_1(1 - w_4) + \varepsilon \quad (2.3)$$

where $w_j \sim \text{Binomial}(p = 0.4)$, $j = 1, \dots, 10$ and $\varepsilon \sim \text{Normal}(0, 1)$. Each observation consists of the 10 dimensional covariate vector W , and the continuous response variable Y . The parameter of interest is $\psi_0(W) = E_0(Y|W)$. The simulated learning data set contains a sample of 500 observations ($i=1, \dots, 500$) from model 2.3.

We applied the super learner to the learning set using five candidate learners. The first candidate was a simple linear regression model with only main terms, which will be estimated with regular least squares. The second candidate was main terms LARS. Internal cross-validation (i.e. another layer of cross-validation inside each training split) was used to estimate the optimal fraction parameter, $\lambda_0 \in (0, 1)$. The third candidate was the D/S/A algorithm for data-adaptive polynomial regression. For the D/S/A algorithm, we allowed interaction terms and restricted the model to less than 50 terms. The D/S/A uses internal cross-validation to determine the best model in this model space. The fourth candidate was logic regression where the number of trees was selected to be 5 and the number of leaves to be 20 based on 10-fold cross validation of the learning data set. For the logic regression fine-tuning parameters, we searched over $\#\text{trees} \in \{1, \dots, 5\}$ and $\#\text{leaves} \in \{1, \dots, 20\}$. The final candidate algorithm was random forests. Table 2.1 contains references for the R packages of each candidate learner.

We applied the super learner with 10-fold cross-validation on the learning set. Applying the prediction to all 10 folds of the learning set gives us the predicted values $Z_i \equiv (\hat{\Psi}_{j\nu(i)}(W_i) : j = 1, \dots, 5)$ and corresponding Y_i for each observation $i = 1, \dots, 500$. We then proposed the linear model $E(Y|Z) = \alpha + \beta Z$ and used least squares to estimate the intercept α and parameter vector β based on (Y_i, Z_i) , $i = 1, \dots, n$.

After having obtained the fit α_n, β_n of α, β , next, each of the candidate learners was fit on the entire learning set to obtain

$Psi_j(P_n)(W)$, which gives the super learner $\hat{\Psi}(P_n)(W) = \alpha_n + \beta_n(\hat{\Psi}_j(P_n)(W) : j = 1, \dots, 5)$ when applied to a new covariate vector W .

To evaluate the super learner next to each of the candidate learners, an additional 10,000 observations are simulated from the same data generating distribution. This new sample is denoted the evaluation sample. Using the models on the learning data set, we calculated the mean squared prediction error (MSPE) on this new evaluation data set for the super learner and each of the candidate learners. Table

method	RMSPE	β_n
Least Squares	1.00	0.038
LARS	1.15	-0.171
D/S/A	0.22	0.535
Logic	0.32	0.274
Random Forest	0.42	0.398
Super Learner	0.20	

Table 2.2: Simulation Example 1: Estimates of the relative mean squared prediction error (compared to least squares) based on a learning sample of 500 observations and the evaluation sample $M=10,000$. The estimates for β in the super learner are also reported in the right column ($\alpha_n = -0.018$).

2.2 has the results for the relative mean squared prediction error (RMPSE), where $RMSPE(x) = MSPE(x)/MSPE(\text{least squares})$. Among the candidate learners, the D/S/A algorithm appears to have the smallest error, but the super learner improves on the D/S/A fit. The estimates β_n all appear to be nonzero except for the simple linear regression model. The super learner can combine information from the candidate learners to build a better predictor.

The second simulation considers continuous covariates as opposed to binary covariates from the first simulation. Let X be a 20 dimensional multivariate normal random vector and $X \sim N_p(0, 16 * Id_p)$ where $p = 20$ and Id_p is the p -dimensional identity matrix. Each column of X is a covariate in the models used below. The outcome is defined as:

$$Y_i = X_1X_2 + X_{10}^2 - X_3X_{17} - X_{15}X_4 + X_9X_5 + X_{19} - X_{20}^2 + X_9X_8 + \varepsilon, \quad (2.4)$$

where $\varepsilon \sim Normal(0, 16)$ and X_j is the j^{th} column of X . From this model, 200 observations were simulated for the learning data set and an additional 5,000 were simulated for the evaluation data set similar to the first simulation. The super learner was applied with the following candidate learners:

- Simple linear regression with all 20 main terms.
- LARS with internal cross-validation to find the optimal fraction.
- D/S/A with internal cross-validation to select the best model with fewer than 25 terms allowing for interaction and quadratic terms.
- Ridge regression with internal cross-validation to select the optimal $L2$ penalty parameter.
- Random forests with 1,000 trees.
- Adaptive regression splines.

Table 2.3 contains the results for the second simulation. As in the first simulation, the relative mean squared prediction error is used to evaluate the candidate learners and the super learner. For this model, simple linear regression, LARS, and ridge regression all appear to have the same results. Random forests and adaptive regression splines are better able to pick up the non-linear relationship, but among the candidate learners, the D/S/A is the best with a relative MSPE of 0.43. But the super learner improves on the fit even more with a relative MSPE of 0.22 by combining the candidate learners. Since the model for $\psi_n^*(z)$ can be near collinear, the estimates of β are often unstable and should not be used to determine the best candidate by comparing the magnitude of the parameter estimate.

method	RMSPE	β_n
Least Squares	1.00	-0.73
LARS	0.91	-0.92
D/S/A	0.43	0.86
Ridge	0.98	0.61
Random Forest	0.71	1.06
MARS	0.61	0.05
Super Learner	0.22	

Table 2.3: Simulation Example 2: Estimates of the relative mean squared prediction error (compared to Least Squares) based on a learning sample of 200 observations and the evaluation sample M=5,000. The estimates for β in the super learner are also reported in the right column ($\alpha_n = 0.03$).

The main advantage of the proposed super learner is the adaptivity to different data generating distributions across many studies. The third simulation demonstrates this feature by creating 3 additional studies and applying the super learner and the candidates to all 3 studies then combining the results with the second simulation and evaluating the mean square error across all 4 studies. Equation 2.5 shows the data generating distributions for the 3 new studies. The data generating distribution for the covariates X is the same as the second simulation example above. To be consistent across the 4 studies, the same candidate learners from the second simulation were applied to these 3 new studies.

$$Y_{ij} = \begin{cases} -5 + X_2 + 6(X_{10} + 8)_+ - 6(X_{10})_+ - 7(X_{10} - 5)_+ \\ \quad - 6(X_{15} + 6)_+ + 8(X_{15})_+ + 7(X_{15} - 6)_+ + \varepsilon & \text{if } j = 1 \\ 10 \cdot \mathbf{I}(X_1 > -4 \text{ and } X_2 > 0 \text{ and } X_3 > -4) + \varepsilon & \text{if } j = 2 \\ -4 + X_2 + \sqrt{|X_3|} + \sin(X_4) - .3X_6X_{11} + 3X_7 \\ \quad + .3X_8^3 - 2X_9 - 2X_{10} - 2X_{11} + \varepsilon & \text{if } j = 3 \end{cases} \quad (2.5)$$

method	study 1	study 2	study 3	2 nd simulation	overall
Least Squares	1.00	1.00	1.00	1.00	1.00
LARS	0.91	0.95	1.00	0.91	0.95
D/S/A	0.22	0.95	1.04	0.43	0.71
Ridge	0.96	0.99	1.02	0.98	1.00
Random Forest	0.39	0.72	1.18	0.71	0.91
MARS	0.02	0.82	0.17	0.61	0.38
Super Learner	0.02	0.67	0.16	0.22	0.19

Table 2.4: Simulation Example 3: Estimates of the relative mean squared prediction error (compared to least squares) based on the validation sample. The 3 new studies from 2.5 are combined with the second simulation example and the relative mean squared prediction error is reported in the overall column.

where $\varepsilon \sim \text{Normal}(0, 16)$ and $I(x) = 1$ if x is true, and 0 otherwise. For the 4 studies (the 3 new studies combined with the second simulation), the learning sample contained 200 observations and the evaluation sample contained 5,000 observations.

Table 2.4 contains the results from the second simulation. For the first study ($j = 1$), the adaptive regression spline function is able to estimate well the true distribution. The super learner is not able to improve on the fit, but it does not do worse than the best candidate algorithm. In the second study ($j = 2$), the adaptive regression spline function is not the best candidate learner. The random forests performs best in the second study, but the super learner is able to improve on the fit. The third study ($j = 3$) is similar to the first in that the adaptive regression splines function is able to approximate the true distribution well, but the super learner does not do worse. The squared prediction error from these three studies and the second simulation was combined to give a mean squared prediction error for the four studies. The last column in table 2.4 gives the relative mspe for each of the candidate learners and the super learner. If the researcher had selected just one of the candidate learners, they might have done well within one or two of the studies, but overall the super learner will outperform the candidate learners. For example, the MARS learner performs well on the first and third study, and does well overall with a relative MSPE of 0.38, but the super learner outperforms the MARS learner with an overall relative MSPE of 0.19. The super learner is able to adapt to the different data generating distributions and will outperform any candidate learner across many studies.

2.4 Data Analysis

We applied the super learner to the diabetes data set from the LARS package in R. Details on the data set can be found in [25]. The data set consists of 442 observations of 10 covariates (9 quantitative and 1 qualitative) and a continuous outcome. The

covariates have been standardized to have mean zero and unit L2 norm. We selected 6 candidate learners for the super learner. The first candidate was least squares using all 10 covariates. Next we considered the least squares model with all possible two-way interactions and quadratic terms on the quantitative covariates. The third and fourth candidates were applying LARS to the main effects and all possible two-way interaction models above. Internal cross-validation was used to select the “fraction” point for the prediction. The fifth candidate algorithm was D/S/A allowing for two-way interactions and a maximum model size of 64. The final candidate learner was the random forests algorithm. For the super learner, we then used a linear model and estimated the parameters with least squares.

We also applied the proposed super learner to the HIV-1 drug resistance data set in [80] and [71]. The goal of the data is to predict drug susceptibility based on mutations in the protease and reverse transcriptase enzymes. The HIV-1 sequences were obtained from publicly available isolates in the Stanford HIV Reverse Transcriptase and Protease Sequence Database. Details on the data and previous analysis can be found in [80] and [71]. The outcome of interest is standardized log fold change in drug susceptibility, defined as the ratio IC_{50} of an isolate to a standard wildtype control isolate; IC_{50} (inhibitory concentration) is the concentration of the drug needed to inhibit viral replication by 50%. We focused our analysis to a single protease inhibitor, nelfinavir, where we have 740 viral isolates in the learning sample of 61 binary predictor covariates and one quantitative outcome.

For the HIV data set, we considered six candidate learners. The first candidate was least squares on all main terms. The second candidate was the LARS algorithm. Internal cross validation was used to determine the best fraction parameter. The third candidate was logic regression. Similar to the simulation example, we used 10-fold cross-validation on the entire learning set to determine the parameters, $\#trees \in \{1, \dots, 5\}$ and $\#leaves \in \{1, \dots, 20\}$, for logic regression. For the HIV data set, we selected $\#trees = 5$ and $\#leaves = 10$. The fourth candidate was the CART algorithm. We also applied the D/S/A algorithm searching over only main effects terms and a maximum model size of 35. The final candidate was random forests. For the super learner, a linear model was used to estimate the parameters with least squares. All models were fit in R similar to the simulation example above.

To evaluate the performance of the super learner in comparison to each of the candidate learners we split the learning data set into 10 validation data sets and corresponding training data sets. The super learner and each candidate learner was fit one each fold of the cross-validation, giving us a honest cross-validated risk estimate to compare the super learner to each of the candidate learners.

2.4.1 Super Learner Results

Table 2.5 presents results for the diabetes data analysis. A 10-fold cross-validation estimate of the mean squared error was calculated, and the relative risk estimate is

Method	RCV risk	β_n
Least Squares (1)	1.00	0.172
Least Squares (2)	1.13	-0.003
LARS (1)	1.07	0.239
LARS (2)	1.08	0.126
D/S/A	0.98	0.481
Random Forests	1.07	0.027
Super Learner	0.98	

Table 2.5: Super learner results for the diabetes data set. Least Squares (1) and LARS (1) refer to the main effects only models. Least Squares (2) and LARS (2) refer to the all possible two-way interaction models. Relative 10-fold Honest Cross-Validation risk estimates, compared to main terms least squares (RCV risk) are reported. β_n in the super learner is reported in the last column ($\alpha_n = -6.228$).

Method	RCV risk	β_n
Least Squares	1.00	0.552
LARS	1.03	0.075
Logic	1.52	-0.020
CART	1.77	0.076
D/S/A	1.53	-0.161
Random Forests	1.02	0.510
Super Learner	0.87	

Table 2.6: Super learner results for the HIV data set. Relative 10-fold honest cross validated risk estimates (RCV risk) compared to least squares are reported. β_n in the super learner is reported in the last column ($\alpha_n = 0.027$).

reported. The relative cross-validation risk estimate (RCV) is:

$$RCV(x) = CV(x)/CV(\text{main terms least squares}),$$

where $CV(x)$ is the cross-validation risk estimate for x . Based on the cross validated estimate, the D/S/A has the best estimate among the candidate learners. The super learner does not appear to improve significantly on the D/S/A learner, but it does not do any worse either. We also report the estimates α_n and β_n used in the super learner. The D/S/A algorithm has the largest coefficient (0.481) and appears to be given the most weight in the super learner. We also note that least squares with all possible two-way interactions is barely used in the super learner, with a coefficient of -0.003 . This example shows how the super learner can use cross validation to data adaptively select (i.e. give more weight) to the better candidate predictors.

Table 2.6 presents the results for the HIV data analysis. Based on 10-fold cross validated estimates of the mean squared error, main terms least squares performs best,

although random forests and LARS have similar error estimates to least squares. In contrast to the diabetes data analysis above, D/S/A does not perform well on this data set. This highlights the need for a super learner since one candidate algorithm will not work on all data sets. Among the candidate learners, least squares has the smallest cross-validated risk estimate, but the super learner has a smaller risk estimate ($RCV = 0.87$). We also present the estimates for α and β in table 2.6. Both least squares and random forests appear to be receiving the most weight in the super learner with coefficients 0.552 and 0.510 respectively. Again, the super learner can use the cross validated predictions to data adaptively build the best predictor.

These are both situations where one of the candidate learners does a good job of prediction and gives little room for improvement for the super learner. But these examples also demonstrate that one candidate algorithm may not be flexible enough to perform best on all data generating distributions and since a researcher is unlikely to know *a priori* which candidate learner will work best, the super learner is a natural choice for prediction.

2.5 Discussion

The new super learning approach provides both a fundamental theoretical as well as practical improvement to the construction of a predictor. The super learner is a flexible prediction algorithm which can perform well on many different data generating distributions, and utilizes cross-validation to protect against over-fitting. We wish to stress that the theory suggests that to achieve the best performance one should not apply this algorithm to a restricted set of candidate learners, but one should aim to include any available sensible learners. In addition, the amount of computations does not exceed the amount of computations it takes to calculate each of the candidate learners on the training and full data sets. In our simulations we used a particular set of available learners only because they were easily available as R functions. Thus, the potential for improving learners applies to a very wide array of practical problems.

Our results generalize to parameters which can be defined as minimizers of a loss function, including (unknown) loss functions indexed by parameters of the true data generating distribution ([87]). In particular, the super learner approach applies to maximum likelihood estimation in semiparametric or nonparametric models for the data generating distribution, and to targeted maximum likelihood estimation with respect to a particular smooth functional of the density of the data, as presented in [91].

2.6 Appendix

Under the Assumption A1 that the loss function $L(O, \psi) = (Y - \psi(X))^2$ is uniformly bounded, and the Assumption A2 that the variance of the ψ_0 -centered loss

function $L(O, \psi) - L(O, \psi_0)$ can be bounded by its expectation uniformly in ψ , [88] (Theorem 3.1) establish the following finite sample inequality.

Theorem 1. *Let $\{\hat{\psi}_k = \hat{\Psi}_k(P_n), k = 1, \dots, K(n)\}$ be a given set of $K(n)$ estimators of the parameter value $\psi_0 = \arg \min_{\psi \in \Psi} \int L(o, \psi) dP_0(o)$. Let $d_0(\psi, \psi_0) \equiv E_{P_0}\{L(O, \psi) - L(O, \psi_0)\}$ denote the risk difference between a candidate estimator ψ and the parameter ψ_0 . Suppose that Ψ is a parameter space so that $\hat{\Psi}_k(P_n) \in \Psi$ for all k , with probability 1. Let $\hat{K}(P_n) \equiv \arg \min_k E_{B_n} \int L(o, \hat{\Psi}_k(P_{n,B_n}^0)) dP_{n,B_n}^1(o)$ be the cross-validation selector, and let $\tilde{K}(P_n) \equiv \arg \min_k E_{B_n} \int L(o, \hat{\Psi}_k(P_{n,B_n}^0)) dP_0(o)$ be the comparable oracle selector. Let p be the proportion of observations in the validation sample. Then, under assumptions A1 and A2, one has the following finite sample inequality for any $\lambda > 0$ (where $C(\lambda)$ is a constant, defined in [88]):*

$$Ed_0(\hat{\Psi}_{\hat{K}(P_n)}(P_{n,B_n}^0), \psi_0) \leq (1 + 2\lambda)Ed_0(\hat{\Psi}_{\tilde{K}(P_n)}(P_{n,B_n}^0), \psi_0) + 2C(\lambda) \frac{1 + \log(K(n))}{np}$$

Chapter 3

Assessment of Super Learner Algorithm

3.1 Introduction

A common task in statistical data analysis is estimator selection for prediction. An outcome Y_i is measured along with a set of covariates W_i and interest is in the regression function $E(Y|W)$. For a given regression problem, it is possible to create a set of algorithms all estimating the same function but with some variety. An algorithm is an estimator that maps a data set of n observations (W_i, Y_i) , $i = 1, \dots, n$, into a prediction function that can be used to map an input W into a predicted value for Y . The algorithms may differ in the subset of the covariates used, the basis functions, the loss functions, the searching algorithm, and the range of tuning parameters, among others. We prefer the more general notion estimator selection instead of model selection, since the formal meaning of model in the field of statistics is the set of possible probability distributions, while most algorithms are not indexed by a model choice.

Estimator selection is not limited to selecting only a single estimator. Recent work has demonstrated that an ensemble of the algorithms in the collection can outperform a single algorithm. [89] introduced the super learner for estimator selection and proved the optimality of such a method. The super learner is related to the stacking algorithm introduced in neural networks context by [96] and adapted to the regression context by [10]. The stacking algorithm is examined in [57] and the relationship to the model-mix algorithm of [82] and the predictive sample-reuse method of [36] is discussed.

In this article, we demonstrate how the super learner algorithm, as outlined by [89] as a general minimum loss based learning method, in the context of prediction can be implemented, and empirically demonstrate the advantage of such an ensemble method. After a review of the super learner algorithm, a sequence of simulations is

presented to highlight properties of the super learner, in accordance with the theoretical optimality results of the super learner. Following the simulations, the super learner is demonstrated on a series of real data sets.

3.2 Super Learner

In this section the general super learner algorithm is described along with some of the specific recommendations for implementation. The details follow [89] closely, although some of the notation has changed to be consistent with the rest of the article.

Observe the learning data set $X_i = (Y_i, W_i)$, $i = 1, \dots, n$ where Y is the outcome of interest and W is a p -dimensional set of covariates. The objective is to estimate the function $\psi_0(W) = E(Y|W)$. The function can be expressed as the minimizer of the expected loss:

$$\psi_0(W) = \arg \min_{\psi} E[L(X, \psi(W))] \quad (3.1)$$

where the loss function is often the squared error loss, $L_2 : (Y - \psi(W))^2$. For a given problem, a library of prediction algorithms can be proposed. A library is simply a collection of algorithms. The algorithms in the library should come from contextual knowledge and a large set of default algorithms. We use algorithm in the general sense as any mapping from the data into a predictor. The algorithms may range from a simple linear regression model to a multi-step algorithm involving screening covariates, optimizing tuning parameters, and selecting a model. As long as the algorithm takes the observed data and outputs a predicted value we consider it a prediction algorithm. The library may be comprised of all possible subset regression models or a diverse set of algorithms mixed together. Denote the library $\mathcal{L}$ and the cardinality of $\mathcal{L}$ as K .

1. Fit each algorithm in $\mathcal{L}$ on the entire data set X_i to estimate $\hat{\Psi}_k(W)$, $k = 1, \dots, K$.
2. Split the data into V folds. Define $T(\nu)$ to be the ν^{th} training data split and $V(\nu)$ to be the corresponding validation data split. $T(\nu) = X \setminus V(\nu)$, $\nu = 1, \dots, V$.
3. For the ν^{th} fold, fit each algorithm in $\mathcal{L}$ on $X_{T(\nu)}$ and save the predictions on the corresponding validation data, $\hat{\Psi}_{k,T(\nu)}(W_{V(\nu)})$, for $\nu = 1, \dots, V$.
4. Stack the predictions from each algorithm together to create a n by K matrix, $Z = \left\{ \hat{\Psi}_{k,T(\nu)}(W_{V(\nu)}), \nu = 1, \dots, V \text{ \& } k = 1, \dots, K \right\}$
5. Estimate the weights for each algorithm (α) in the model:

$$m(z|\alpha) = \sum_{k=1}^K \alpha_k \hat{\Psi}_{k,T(\nu)}(W_{V(\nu)}), \alpha_k \geq 0 \forall k, \sum_{k=1}^K \alpha_k = 1.$$

6. Combine $\hat{\alpha}$ with $\hat{\Psi}_k(W)$, $k = 1, \dots, K$ according to the model $m(z|\alpha)$ to create the final super learner fit:

$$\hat{\Psi}_{SL}(W) = \sum_{k=1}^K \hat{\alpha}_k \hat{\Psi}_k(W)$$

The super learner theory does not place any restrictions on the model for ensembling the algorithms in the library. The restriction of the parameter space for α to be the convex combination of the algorithms in the library provides greater stability of the final super learner prediction. The convex combination is not only empirically motivated, but also supported by the theory. The oracle results for the super learner require a bounded loss function. Restricting to the convex combination implies that if each algorithm in the library is bounded, the convex combination will also be bounded.

Also contained within the model space of the convex combination is the possibility of selecting only one algorithm. The nodes of the convex hull:

$$\mathcal{A}_{node} = \{(1, 0, \dots, 0), (0, 1, \dots, 0), \dots, (0, 0, \dots, 1)\},$$

correspond with selecting the single algorithm from the library that minimizes the V-fold cross-validated risk. This is the usual cross-validation selector we refer to as the discrete super learner since it is searching over a finite set for the algorithm weights (α). Although possible for the super learner to select only a single algorithm from the library, it is almost always optimal to average the algorithms together in an ensemble.

The super learner performs asymptotically as well as best possible weighted combination. For convenience, we provide here a short summary of the oracle results established in previous articles. For detail, we refer to these articles as referred to in [89]. The oracle result for the cross-validation selector among a family of candidate estimators was established in [87] for general bounded loss functions: see also [92] for unbounded loss functions with tails that are controlled by exponential bounds and infinite families of candidate estimators, and [89] for its application to the super learner. This result proves (see above references for the precise statement of these implications) that, if the number of candidate estimators, $K(n)$, is polynomial in sample size, then the cross-validation selector is either asymptotically equivalent with the oracle selector (based on sample of size of training samples), or it achieves the parametric rate $\log n/n$ for convergence w.r.t. $d(\psi, \psi_0) \equiv E_0\{L(\psi) - L(\psi_0)\}$. So in most realistic scenarios, in which none of the candidate estimators achieve the rate of convergence one would have with an a priori correctly specified parametric model, the cross-validated selected estimator selector performs asymptotically exactly as well (up till a constant) as the oracle selected estimator. As a consequence, the super learner will perform asymptotically exactly as well (w.r.t. the loss based dissimilarity) as the best possible choice for the given data set among the family of weighted combinations

of the estimators. In particular, this proves that, by including all competitors in the library of candidate estimators, the super learner will asymptotically outperform any of its competitors, even if the set of competitors is allowed to grow polynomial in sample size. This motivated our naming “super learner” since it provides a system of combining many estimators into an improved estimator.

3.3 Examples

To examine the performance of the super learner we start with a series of simulations and then demonstrate a super learner on a collection of real data sets. The simulations are intended to illustrate the application of the super learner algorithm and highlight some of the properties.

3.3.1 Simulations

Four different simulations are presented in this section. All four simulations involve a univariate X drawn from a uniform distribution in $[-4, +4]$. The outcome follows the function described below:

$$\textbf{Sim 1: } Y = -2 \times \mathbf{I}(X < -3) + 2.55 \times \mathbf{I}(X > -2) - 2 \times \mathbf{I}(X > 0) \\ + 4 \times \mathbf{I}(X > 2) - 1 \times \mathbf{I}(X > 3) + \varepsilon$$

$$\textbf{Sim 2: } Y = 6 + 0.4X - 0.36X^2 + 0.005X^3 + \varepsilon$$

$$\textbf{Sim 3: } Y = 2.83 \times \sin\left(\frac{\pi}{2} \times X\right) + \varepsilon$$

$$\textbf{Sim 4: } Y = 4 \times \sin(3\pi \times X) \times \mathbf{I}(X > 0) + \varepsilon$$

where $\mathbf{I}(\cdot)$ is the usual indicator function and ε is drawn from an independent standard normal distribution in all simulations. A sample of size 100 will be drawn for each scenario. Figure 3.1 contains a scatterplot with a sample from each of the four simulations. The true curve for each simulation is represented by the solid line. These four simulations were chosen because they represent a diverse set of true models but all four have the same optimal $R^2 = 0.80$. The R^2 is computed as:

$$R^2 = 1 - \frac{\sum (Y_i - \hat{Y}_i)^2}{\sum (Y_i - \text{ave}(Y_i))^2} \quad (3.2)$$

The optimal R^2 is the value attained when the true model is known. Knowing the true model implies $\sum (Y_i - \hat{Y}_i)^2 = \text{Var}(\varepsilon)n = 1n$. Hence the optimal R^2 in all four simulations is $R_{opt}^2 = 1 - 1/\text{Var}(Y)$ and the variance of Y is set such that $R_{opt}^2 = 0.80$ in each simulation.

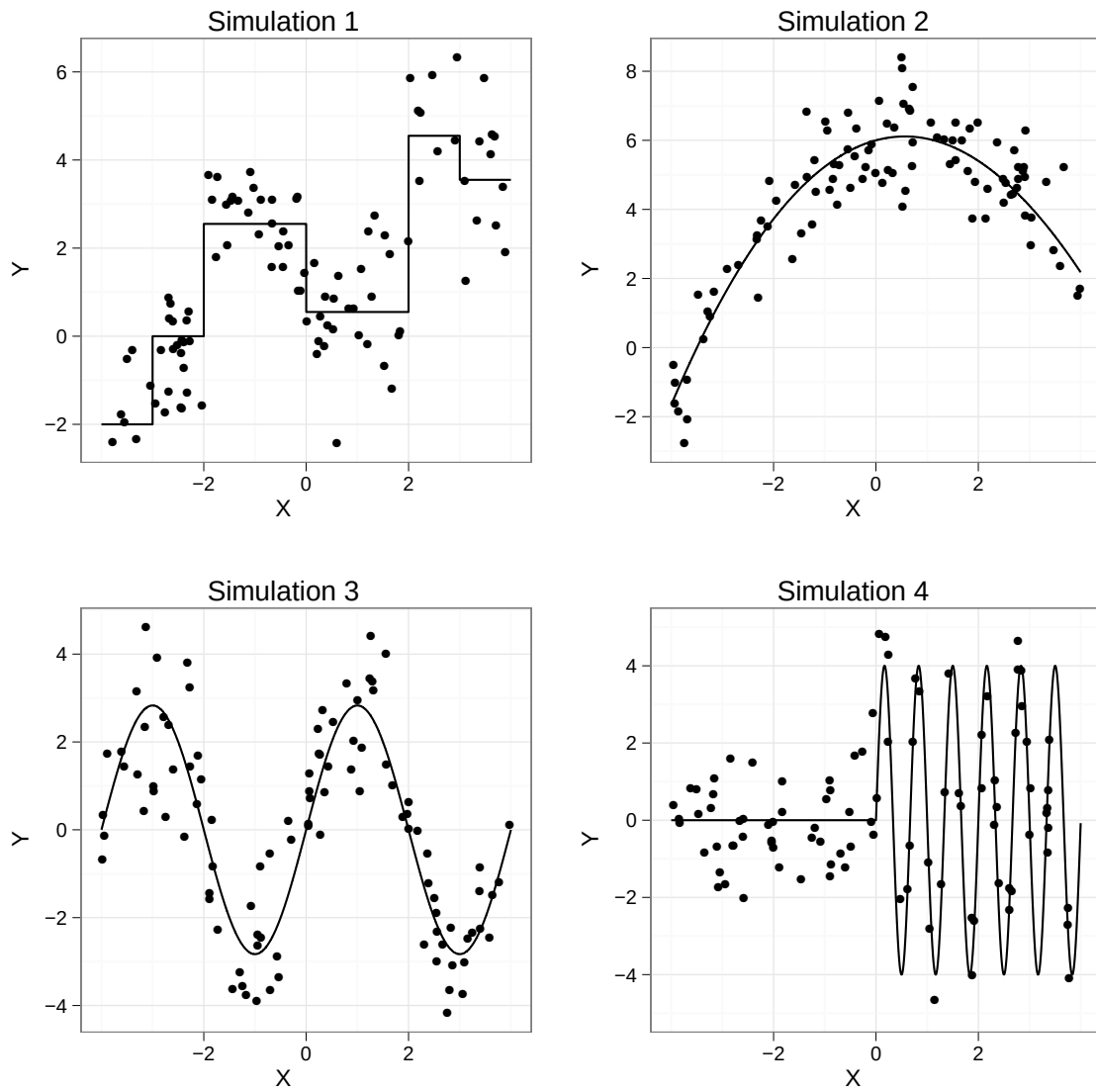


Figure 3.1: Scatterplots of the four simulations. The solid line is the true relationship. The points represent one of the simulation samples of size $n = 100$.

In all four simulations, we start with the same library of prediction algorithms. Table 3.1 contains a list of the algorithms in the library. The library of algorithms should ideally be a diverse set. One common aspect of many prediction algorithms is the need to specify values for tuning parameters. For example the generalized additive models requires a degree of freedom value for the spline functions or the neural network requires a size value. The tuning parameters could be selected using cross-validation or bootstrapping, but the different values of the tuning parameters could also be considered different prediction algorithms. A library could contain three generalized additive models with degrees of freedom equal to 2, 3, and 4. The selection of tuning parameters does not need to be done inside an algorithm, but each fixed value of a tuning parameter can be consider a different algorithm and the ensemble method can decide how to weight the different algorithms with unique values of the tuning parameters. When one considers different values of tuning parameters as unique prediction algorithms in the library it is easy to see how the number of algorithms in the library can become large. The library for the simulations contains 21 algorithms when considering different values of tuning parameters. A linear model and a linear model with a quadratic term are considered. The default random forest algorithm along with a collection of bagging regression trees with values of the complexity parameter equal to 0.10, 0.01, and 0.00 and a bagging algorithm adjusting the minimum split parameter to be 5 (with the default complexity parameter of 0.01). The generalized additive model with degrees of freedom equal to 2, 3, and 4 is added along with the default gradient boosting model. Neural networks with sizes 2 through 5, the polymars algorithm and the Bayesian additive regression trees is added. Finally, we consider the loess curve with spans equal to 0.75, 0.50, 0.25, and 0.10.

Figure 3.2 contains the super learner fit on a simulated data set for each scenario. With the given library the super learner is able to adapt to the underlying structure of the data generating function. For each algorithm we evaluated the R^2 on a test set of size 10,000. To evaluate the performance of the super learner in comparison to each algorithm in the library, we simulated 100 samples of size 100 and computed the R^2 for each model fit. The results are presented in Table 3.2. In the first simulation, the regression tree based methods perform best. Bagging complete regression trees ($cp = 0$) has the largest R^2 . On the second simulation, the best algorithm is the quadratic linear regression model (SL.interaction). In both these cases the super learner is able to adapt to the underlying structure and is able to have an average R^2 near the best. The same trend is exhibited in simulations 3 and 4, the super learner method of combining algorithms does nearly as well as the individual best algorithm. Since the individual best algorithm is not known, if a researcher was to select a single algorithm they might do well on some cases, but the overall performance will be worse than the super learner. For example, an individual who always uses bagging complete trees (SL.bagging($cp = 0.0$)) will do well on the first 3 simulations, but will perform poorly on the 4th simulation compared to the average performance of the super learner.

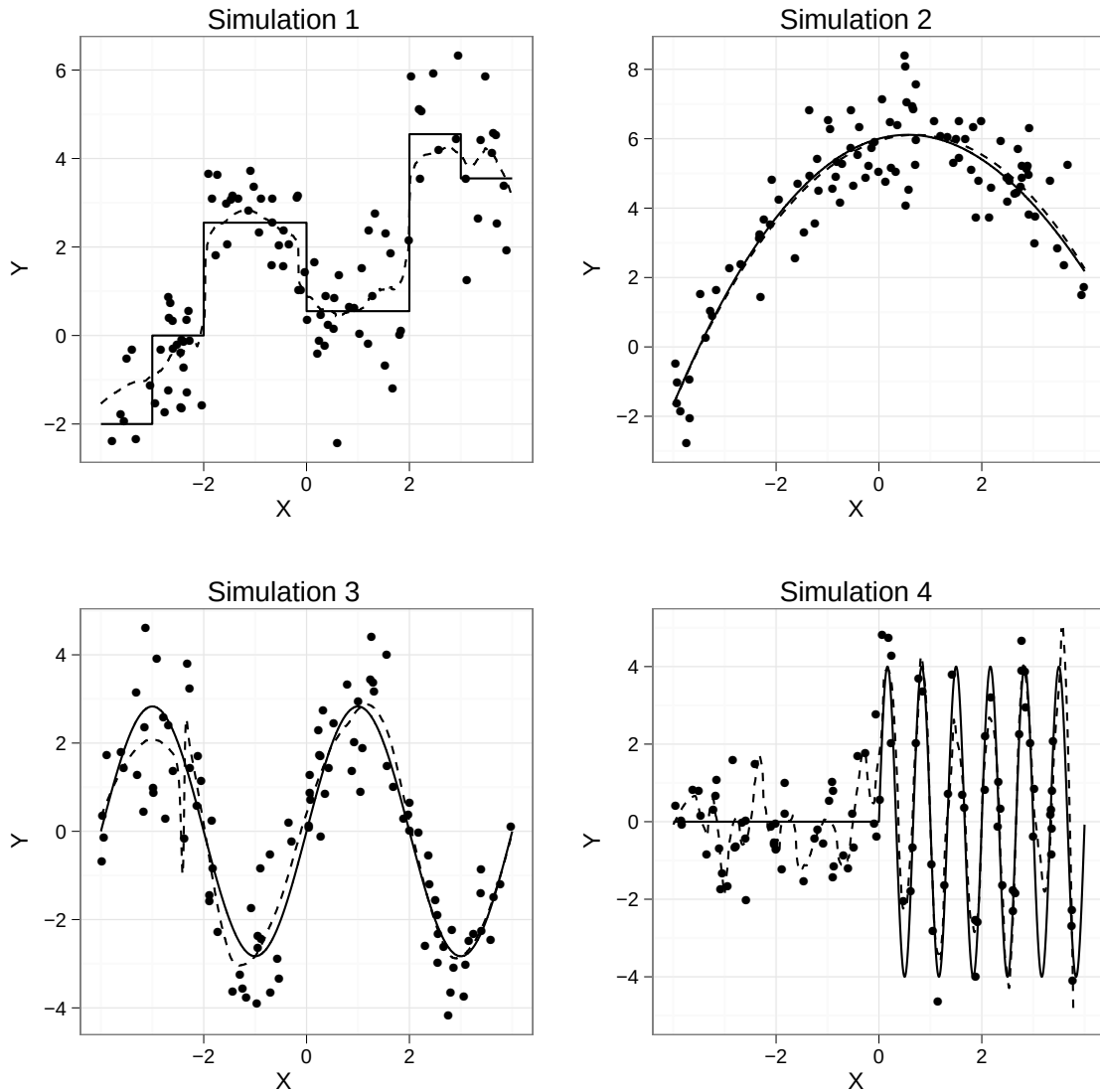


Figure 3.2: Scatterplots of the four simulations. The solid line is the true relationship. The points represent one of the simulated datasets of size $n = 100$. The dashed line is the super learner fit for the shown dataset.

Table 3.1: Library of prediction algorithms for the simulations and citation for the corresponding R package.

Algorithm	Description	Author
<code>glm</code>	linear model	[70]
<code>interaction</code>	polynomial linear model	[70]
<code>randomForest</code>	random Forest	[58]
		[11]
<code>bagging</code>	bootstrap aggregation of trees	[67]
		[8]
<code>gam</code>	generalized additive models	[43]
		[45]
<code>gbm</code>	gradient boosting	[72]
		[35]
<code>nnet</code>	neural network	[95]
<code>polymars</code>	polynomial spline regression	[52]
		[34]
<code>bart</code>	Bayesian additive regression trees	[15]
		[16]
<code>loess</code>	local polynomial regression	[18]

The optimal R^2 is the value attained by knowing the true model. The optimal value gives an upper bound on the possible R^2 for each algorithm. In the first three simulations the super learner is able to come close to the optimal value because some of the algorithms in the library are able to well approximate the truth. But in the fourth simulation the library is not rich enough to contain a combination of algorithms able to approach the optimal value. The super learner is able to do as well as the best algorithms in the library but does not attain the optimal R^2 .

For the fourth simulation example, if the researcher thought that the relationship between X and Y followed a linear model up to a knot point, and then was a sine curve with an unknown frequency and amplitude, they could augment the library with proposed prediction algorithms:

$$\begin{aligned} \text{sinKnot}(X; k, \omega) = & \{\beta_1 + \beta_2 X\} \times \mathbf{I}(X < k) \\ & + \{\beta_3 + \beta_4 \sin(\omega X)\} \times \mathbf{I}(X \geq k) \end{aligned} \quad (3.3)$$

for fixed values of the knot k and the frequency ω . For the fourth simulation, we augmented the library with the regression model above selecting values for the knots in $\{-2, -1, 0, +1, +2\}$ and values for the frequency in $\{\pi, 2\pi, 3\pi, 4\pi\}$ and all pairwise combinations of the two tuning parameters. This added 20 algorithms to the library (for a total of 41 algorithms). The results are presented in table 3.3. The 20 knot plus

Table 3.2: Results for all 4 simulations. Average R-squared based on 100 simulations and the corresponding standard errors.

Algorithm	Sim 1		Sim 2		Sim 3		Sim 4	
	R^2	$se(R^2)$	R^2	$se(R^2)$	R^2	$se(R^2)$	R^2	$se(R^2)$
SuperLearner	0.741	0.032	0.754	0.025	0.760	0.025	0.496	0.122
discrete SL	0.729	0.079	0.758	0.029	0.757	0.055	0.509	0.132
SL.glm	0.422	0.012	0.189	0.016	0.107	0.016	-0.018	0.021
SL.interaction	0.428	0.016	0.769	0.011	0.100	0.020	-0.018	0.029
SL.randomForest	0.715	0.021	0.702	0.027	0.724	0.018	0.460	0.109
SL.bagging(0.01)	0.751	0.022	0.722	0.036	0.723	0.018	0.091	0.054
SL.bagging(0.1)	0.635	0.120	0.455	0.195	0.661	0.029	0.020	0.025
SL.bagging(0.0)	0.752	0.021	0.722	0.034	0.727	0.017	0.102	0.060
SL.bagging(ms5)	0.747	0.020	0.727	0.030	0.741	0.016	0.369	0.104
SL.gam(2)	0.489	0.013	0.649	0.026	0.213	0.029	-0.014	0.023
SL.gam(3)	0.535	0.033	0.748	0.024	0.412	0.037	-0.017	0.029
SL.gam(4)	0.586	0.027	0.759	0.020	0.555	0.022	-0.020	0.034
SL.gbm	0.717	0.035	0.694	0.038	0.679	0.022	0.063	0.040
SL.nnet(2)	0.476	0.235	0.591	0.245	0.283	0.285	-0.008	0.030
SL.nnet(3)	0.700	0.096	0.700	0.136	0.652	0.218	0.009	0.035
SL.nnet(4)	0.719	0.077	0.730	0.062	0.738	0.102	0.032	0.052
SL.nnet(5)	0.705	0.079	0.716	0.070	0.731	0.077	0.042	0.060
SL.polymars	0.704	0.033	0.733	0.032	0.745	0.034	0.003	0.040
SL.bart	0.740	0.015	0.737	0.027	0.764	0.014	0.077	0.034
SL.loess(0.75)	0.599	0.023	0.761	0.019	0.487	0.028	-0.023	0.033
SL.loess(0.50)	0.695	0.018	0.754	0.022	0.744	0.029	-0.033	0.038
SL.loess(0.25)	0.729	0.016	0.738	0.025	0.772	0.015	-0.076	0.068
SL.loess(0.1)	0.690	0.044	0.680	0.064	0.699	0.039	0.544	0.118

sine curve functions are labeled $\text{sinKnot}(k, \omega)$ in the table. The algorithm containing the true relationship between X and Y is $\text{sinKnot}(0, 3\pi)$

3.3.2 Data Analysis

To study the super learner in real data examples, we collected a set of publicly available data sets. Table 3.4 contains a descriptions of the data sets used for the study. The sample sizes ranged from 200 to 654 observations and the number of covariates ranged from 3 to 18. All 13 data sets have a continuous outcome and no missing values. The data sets can be found either in public repositories like the UCI data repository or in textbooks, with the corresponding citation listed in the table.

For the library of prediction algorithms, the applicable algorithms from the univariate simulations along with the algorithms listed in table 3.5. These algorithms represent a diverse set of basis functions and should allow the super learner to work well in most real settings. For the comparison across all data sets, we kept the library of algorithms the same. The super learner may benefit from including algorithms based on contextual knowledge of the data problem as demonstrated in the augmented library in simulation 4.

Each data set has a different scale for the outcome. In order to compare the performance of the prediction algorithms across diverse data sets we used the relative mean squared error where the denominator is the mean squared error of a linear model:

$$\text{relMSE}(k) = \frac{\text{MSE}(k)}{\text{MSE}(lm)}, \quad k = 1, \dots, K \quad (3.4)$$

The results for the super learner, the discrete super learner, and each individual algorithm can be found in figure 3.3. Each point represents the 10-fold cross-validated relative mean squared error for a data set and the plus sign is the geometric mean of the algorithm across all 13 data sets. The super learner slightly outperforms the discrete super learner but both outperform any individual algorithm. With the real data it is unlikely that one single algorithm contains the true relationship and the benefit of the combination of the algorithms versus the selection of a single algorithm is demonstrated. The additional estimation of the combination parameters (α) does not appear to cause an over-fit in terms of the risk assessment. Among the individual library algorithms the bayesian additive regression trees performs the best, but overfits on one of the datasets with a relative mean squared error of almost 3.0. Across various real data examples, the super learner is able to adapt to the true underlying structure. The super learner also outperforms the discrete super learner which demonstrates the advantage of combining algorithms over selecting a single algorithm.

Table 3.3: Simulation results for example 2. Average mean squared error and R-squared based on 100 simulations and the corresponding standard errors.

Algorithm	MSE	se(MSE)	R^2	se(R^2)
SuperLearner	1.193	0.200	0.759	0.040
discrete SL	1.036	0.035	0.791	0.007
SL.glm	5.040	0.106	-0.017	0.021
SL.interaction	5.057	0.126	-0.021	0.026
SL.randomForest	2.645	0.523	0.466	0.106
SL.bagging(0.01)	4.414	0.351	0.109	0.071
SL.bagging(0.1)	4.734	0.200	0.044	0.040
SL.bagging(0.0)	4.416	0.343	0.109	0.069
SL.bagging(ms5)	2.650	0.543	0.465	0.110
SL.gam(2)	5.033	0.113	-0.016	0.023
SL.gam(3)	5.061	0.131	-0.022	0.027
SL.gam(4)	5.089	0.160	-0.027	0.032
SL.gbm	4.580	0.282	0.075	0.057
SL.nnet(2)	5.067	0.447	-0.023	0.090
SL.nnet(3)	4.922	0.627	0.006	0.127
SL.nnet(4)	4.769	0.528	0.037	0.107
SL.nnet(5)	4.816	0.928	0.028	0.187
SL.polymars	4.996	0.309	-0.008	0.062
SL.bart	4.531	0.198	0.085	0.040
SL.loess(0.75)	5.124	0.186	-0.034	0.037
SL.loess(0.50)	5.206	0.238	-0.051	0.048
SL.loess(0.25)	5.661	0.562	-0.143	0.114
SL.loess(0.1)	2.413	0.823	0.513	0.166
SL.sinKnot(-2, 3.14)	5.097	0.122	-0.029	0.025
SL.sinKnot(-1, 3.14)	5.119	0.135	-0.033	0.027
SL.sinKnot(0, 3.14)	5.151	0.151	-0.040	0.031
SL.sinKnot(1, 3.14)	5.109	0.193	-0.031	0.039
SL.sinKnot(2, 3.14)	5.221	0.217	-0.054	0.044
SL.sinKnot(-2, 6.28)	5.146	0.179	-0.039	0.036
SL.sinKnot(-1, 6.28)	5.169	0.195	-0.043	0.039
SL.sinKnot(0, 6.28)	5.217	0.246	-0.053	0.050
SL.sinKnot(1, 6.28)	5.161	0.228	-0.042	0.046
SL.sinKnot(2, 6.28)	5.268	0.271	-0.063	0.055
SL.sinKnot(-2, 9.42)	2.405	0.064	0.515	0.013
SL.sinKnot(-1, 9.42)	1.854	0.058	0.626	0.012
SL.sinKnot(0, 9.42)	1.036	0.035	0.791	0.007
SL.sinKnot(1, 9.42)	2.054	0.090	0.585	0.018
SL.sinKnot(2, 9.42)	3.163	0.163	0.362	0.033
SL.sinKnot(-2, 12.57)	5.110	0.129	-0.031	0.026
SL.sinKnot(-1, 12.57)	5.136	0.145	-0.037	0.029
SL.sinKnot(0, 12.57)	5.173	0.160	-0.044	0.032
SL.sinKnot(1, 12.57)	5.116	0.179	-0.033	0.036
SL.sinKnot(2, 12.57)	5.223	0.221	-0.054	0.045

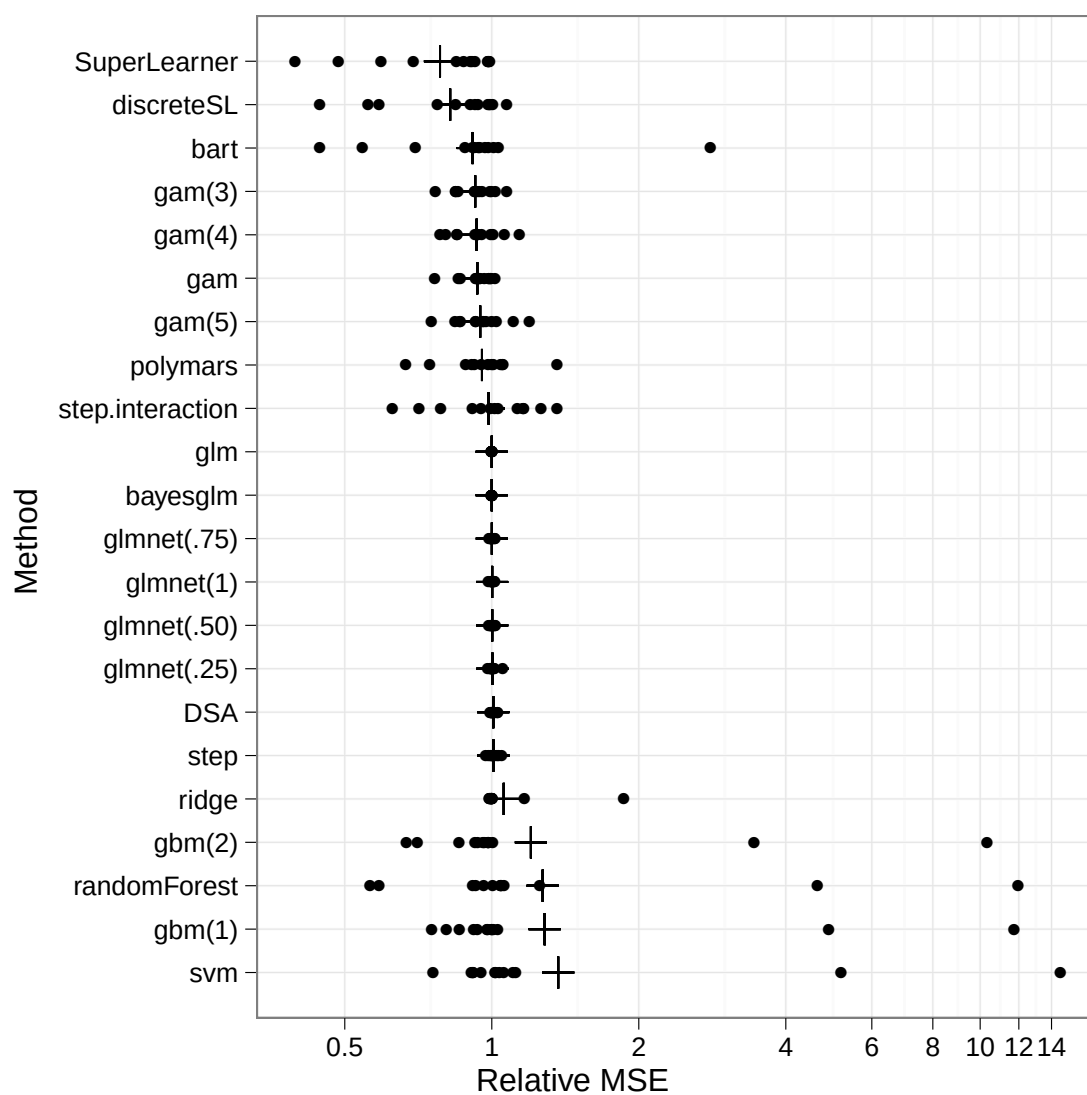
Table 3.4: Description of data sets. n is the sample size and p is the number of covariates. All examples have a continuous outcome.

Name	n	p	Source
ais	202	10	[20]
diamond	308	17	[17]
cps78	550	18	[6]
cps85	534	17	[6]
cpu	209	6	[51]
FEV	654	4	[74]
Pima	392	7	[63]
laheart	200	10	[1]
mussels	201	3	[19]
enroll	258	6	[59]
fat	252	14	[66]
diabetes	366	15	[42]
house	506	13	[63]

Table 3.5: Additional prediction algorithms in the library for the real data examples to be combined with the algorithms from Table 3.1.

Algorithm	Description	Author
bayesglm	Bayesian linear model	[38] [37]
glmnet	Elastic net	[32] [33]
DSA	DSA algorithm	[62] [81]
step	Stepwise regression	[95]
ridge	Ridge regression	[95]
svm	Support vector machine	[22] [14]

Figure 3.3: 10-fold cross-validated relative mean squared error compared to glm across 13 real datasets. Sorted by the geometric mean, denoted with the plus (+) sign.



3.3.3 Array Data

A common application of prediction modeling in biostatistics is with microarray data. The super learner framework easily extends to this setting. Microarray data is often high dimensional, i.e. the number of covariates is larger than the sample size. The library of prediction algorithms needs to incorporate this information, but the models to combine the algorithms is the same.

The super learner in a microarray example is demonstrated using two publicly available data sets. The breast cancer data from [94] and the prostate cancer array data from [79]. The breast cancer study was conducted to develop a gene expression based predictor for 5 year distant metastases. The outcome in this case is a binary indicator that patient had a distant metastases within 5 years after initial therapy. In addition to the expression data, six clinical variables were attained. The clinical information is age, tumor grade, tumor size, estrogen receptor status, progesterone receptor status and angioinvasion. The array data contains 4348 genes after the unsupervised screening steps outlined in the original article. We used the entire sample of 97 individuals (combining the training and validation samples from the original article) to fit the super learner.

The prostate cancer study was conducted to develop a gene expression based predictor of a cancerous tumor. 102 tissue samples were collected with 52 from cancer tissue and 50 from non-cancer tissue. The data set contains 6033 genes after the pre-processing steps, but no clinical variables were available in the data set.

One aspect of high dimensional data analysis is that it is often beneficial to screen the variables before running the prediction algorithms. With the super learner framework, screening of variables can be either supervised or unsupervised since the screening step will be part of the cross-validation step. Screening algorithms can be coupled with prediction algorithms to create new algorithms in the library. For example, we may consider k-nearest neighbors using all features and on the subset of only clinical variables. These two algorithms can be considered unique algorithms in the super learner library. Another screening algorithm we consider is to test the pairwise correlations of each variable with the outcome and rank the variables by the corresponding p-value. With the ranked list of variables, we consider the screening cutoffs as: variables with a p-value less than 0.1, variables with a p-value less than 0.01, variables in the bottom 20, and variables in the bottom 50. An additional screening algorithm is to run the glmnet algorithm and select the variables with non-zero coefficients.

The results for the breast cancer data can be found in table 3.6. The algorithms in the library are k-nearest neighbors with $k = \{10, 20, 30, 40\}$, elastic net with $\alpha = \{1.0, 0.75, 0.50, 0.25\}$, random forests, bagging, bart, and an algorithm that uses the mean value of the outcome as the predicted probability. We coupled these algorithms with the screening algorithms to produce the full list of 38 algorithms. Within the library of algorithms, the best algorithm in terms of minimum risk estimate is the random forest algorithm using only the clinical variables ($MSE = 0.198$). As we

observed in the previous examples, the super learner is able to attain a risk comparable to the best algorithm (MSE = 0.194).

The results from the prostate cancer data can be found in table 3.7. The library of algorithms is similar to that used in the breast cancer example minus the clinical variable screening. In this example, the elastic net algorithm performs best among the library algorithms. The mean squared error for the elastic net model with alpha equal to 0.50 and fit on the entire data set was 0.071. The super learner outperforms even the best algorithm in the library here with a mean squared error of 0.067.

Standard errors of the cross-validated risk are based on the results of Theorem 3 in [23]. The equation for the estimator of the variance of the cross-validated risk of an estimator $\hat{\Psi}$ as an estimator of the true risk is:

$$\sigma_n^2 = \frac{1}{n} \sum \left\{ \left(Y_i - \hat{\Psi}_{T(i)}(W_{V(i)}) \right)^2 - \bar{\theta}_n \right\}^2 \quad (3.5)$$

where $\bar{\theta}_n$ is the cross-validated risk estimate of the mean squared error. The sample standard deviation of the cross-validation values of the squared error are used to calculate the standard error of the V-fold cross-validated mean squared error estimates.

3.4 Discussion

Beyond the asymptotic oracle performance of the super learner, our evaluation of the practical performance of the super learner shows that the super learner is also an adaptive and robust estimator selection procedure for small samples. Combining estimators with the weights (i.e. positive and summing up till 1) based on minimizing cross-validated risk appears to control for over-fitting of the final ensemble fit generated by the super learning algorithm, even when using a large collection of candidate estimators. The above examples demonstrate that the super learner framework allows a researcher to try many prediction algorithms, and many a priori guessed models about the true regression model for a given problem, knowing that the final combined super learner fit will either be the best fit or near the best fit.

Combining estimators with the convex combination algorithm proposed here appears to also improve on the usual cross-validation selector (i.e. discrete super learner). Selection of a single algorithm based on V-fold cross-validated risk minimization may be unstable with the small sample sizes of the data sets presented here, while the super learner can average a few of the best algorithms in the library to give a more stable estimator compared to the discrete super learner.

Table 3.6: 20-fold cross-validated mean squared error for each algorithm and the standard error for the breast cancer study.

Algorithm	Subset	Risk	SE
Super Learner	–	0.194	0.0168
Discrete SL	–	0.238	0.0239
SL.knn(10)	All	0.249	0.0196
SL.knn(10)	Clinical	0.239	0.0188
SL.knn(10)	$\text{cor}(p < 0.1)$	0.262	0.0232
SL.knn(10)	$\text{cor}(p < 0.01)$	0.224	0.0205
SL.knn(10)	glmnet	0.219	0.0277
SL.knn(20)	All	0.242	0.0129
SL.knn(20)	Clinical	0.236	0.0123
SL.knn(20)	$\text{cor}(p < 0.1)$	0.233	0.0168
SL.knn(20)	$\text{cor}(p < 0.01)$	0.206	0.0176
SL.knn(20)	glmnet	0.217	0.0257
SL.knn(30)	All	0.239	0.0128
SL.knn(30)	Clinical	0.236	0.0119
SL.knn(30)	$\text{cor}(p < 0.1)$	0.232	0.0139
SL.knn(30)	$\text{cor}(p < 0.01)$	0.215	0.0165
SL.knn(30)	glmnet	0.210	0.0231
SL.knn(40)	All	0.240	0.0111
SL.knn(40)	Clinical	0.238	0.0105
SL.knn(40)	$\text{cor}(p < 0.1)$	0.236	0.0118
SL.knn(40)	$\text{cor}(p < 0.01)$	0.219	0.0151
SL.knn(40)	glmnet	0.211	0.0208
SL.glmnet(1.0)	$\text{cor}(\text{Rank} = 50)$	0.229	0.0285
SL.glmnet(1.0)	$\text{cor}(\text{Rank} = 20)$	0.208	0.0260
SL.glmnet(0.75)	$\text{cor}(\text{Rank} = 50)$	0.221	0.0269
SL.glmnet(0.75)	$\text{cor}(\text{Rank} = 20)$	0.209	0.0258
SL.glmnet(0.50)	$\text{cor}(\text{Rank} = 50)$	0.226	0.0269
SL.glmnet(0.50)	$\text{cor}(\text{Rank} = 20)$	0.211	0.0256
SL.glmnet(0.25)	$\text{cor}(\text{Rank} = 50)$	0.230	0.0266
SL.glmnet(0.25)	$\text{cor}(\text{Rank} = 20)$	0.216	0.0252
SL.randomForest	Clinical	0.198	0.0186
SL.randomForest	$\text{cor}(p < 0.01)$	0.204	0.0179
SL.randomForest	glmnet	0.220	0.0245
SL.bagging	Clinical	0.207	0.0160
SL.bagging	$\text{cor}(p < 0.01)$	0.205	0.0184
SL.bagging	glmnet	0.206	0.0219
SL.bart	Clinical	0.202	0.0183
SL.bart	$\text{cor}(p < 0.01)$	0.210	0.0207
SL.bart	glmnet	0.220	0.0275
SL.mean	All	0.224	0.1016

Table 3.7: 20-fold cross-validated mean squared error for each algorithm and the standard error for the prostate cancer study.

Algorithm	subset	Risk	SE
SuperLearner	–	0.067	0.018
Discrete SL	–	0.076	0.019
SL.knn(10)	All	0.149	0.017
SL.knn(10)	cor ($p < 0.1$)	0.131	0.018
SL.knn(10)	cor ($p < 0.01$)	0.120	0.020
SL.knn(10)	glmnet	0.066	0.019
SL.knn(20)	All	0.167	0.013
SL.knn(20)	cor ($p < 0.1$)	0.159	0.015
SL.knn(20)	cor ($p < 0.01$)	0.134	0.017
SL.knn(20)	glmnet	0.073	0.018
SL.knn(30)	All	0.185	0.010
SL.knn(30)	cor ($p < 0.1$)	0.180	0.011
SL.knn(30)	cor ($p < 0.01$)	0.153	0.014
SL.knn(30)	glmnet	0.076	0.016
SL.knn(40)	All	0.201	0.008
SL.knn(40)	cor ($p < 0.1$)	0.199	0.009
SL.knn(40)	cor ($p < 0.01$)	0.175	0.011
SL.knn(40)	glmnet	0.085	0.014
SL.glmnet($\alpha = 1.0$)	All	0.080	0.019
SL.glmnet($\alpha = 1.0$)	cor ($p < 0.1$)	0.076	0.018
SL.glmnet($\alpha = 1.0$)	cor (Rank = 50)	0.091	0.021
SL.glmnet($\alpha = 0.75$)	All	0.074	0.018
SL.glmnet($\alpha = 0.75$)	cor ($p < 0.1$)	0.072	0.017
SL.glmnet($\alpha = 0.75$)	cor (Rank = 50)	0.086	0.021
SL.glmnet($\alpha = 0.50$)	All	0.071	0.017
SL.glmnet($\alpha = 0.50$)	cor ($p < 0.1$)	0.069	0.017
SL.glmnet($\alpha = 0.50$)	cor (Rank = 50)	0.084	0.019
SL.randomForest	cor ($p < 0.01$)	0.101	0.014
SL.randomForest	cor (Rank = 50)	0.082	0.016
SL.randomForest	glmnet	0.086	0.016
SL.bart	cor ($p < 0.01$)	0.117	0.016
SL.bart	glmnet	0.084	0.017
SL.bart	cor (Rank = 50)	0.085	0.018
SL.polymars	cor (Rank = 50)	0.081	0.022
SL.polymars	cor (Rank = 100)	0.093	0.024

Chapter 4

Super Learner for Predicting Treatment Effects

4.1 Introduction

With the increasing interest in individualized medicine there is a greater need for robust statistical methods for prediction of optimal treatment based on the patient's characteristics. When evaluating two treatments, one treatment may not be uniformly superior to the other treatment for all patients. A patient characteristic may interact with one of the treatments and change the effect of the treatment on the response. Clinical trials are also collecting more information on the patient. This additional information on the patients combined with the state-of-the-art in model selection allows researchers to build better optimal treatment algorithms.

In this chapter we introduce a methodology for predicting optimal treatment. The methodology is demonstrated first on a simulation and then on a phase III clinical trial in neuro-oncology.

4.2 Predicting Optimal Treatment Based on Baseline Factors

Start with a randomized controlled trial where patients are assigned to one of two treatment arms, $A \in \{0, 1\}$, with $\Pr(A = 1) = \Pi_A$. The main outcome for the trial is defined at a given time point t as $Y = I(T > t)$ where T is the survival time. For example, the main outcome may be the six-month progression-free rate and T is the progression time. Also collected at the beginning of the trial is a set of baseline covariates W . The baseline covariates may be any combination of continuous and categorical variables. The baseline covariates can be split into prognostic and predictive factors. Prognostic factors are patient characteristics which are associated

with the outcome independent of the treatment given, while predictive factors are patient characteristics which interact with the treatment in their association with the outcome. To determine the optimal treatment, a model for how the predictive factors and treatment are related to the outcome needs to be estimated.

The observed data is $O_i = (W_i, A_i, Y_i = I(T_i > t)) \sim P$ for $i = 1, \dots, n$. For now assume Y is observed for all patients in the trial but this assumption is relaxed in the next section. The optimal treatment given a set of baseline variables is found using the W -specific variable importance parameter:

$$\Psi(W) = E(Y|A = 1, W) - E(Y|A = 0, W) \quad (4.1)$$

$\Psi(W)$ is the additive risk difference of treatment A for a specific level of the prognostic variables W . The conditional distribution of Y given W is defined as $\{Y|W\} \sim \text{Bernoulli}(\pi_Y)$. The subscript W is assumed on π_Y and left off for clarity of the notation. Adding the treatment variable A into the conditioning statement we define $\{Y|A = 1, W\} \sim \text{Bernoulli}(\pi_{+1})$ and $\{Y|A = 0, W\} \sim \text{Bernoulli}(\pi_{-1})$. Again the subscript W is dropped for clarity but assumed throughout the paper. The parameter of interest can be expressed as $\Psi(W) = \pi_{+1} - \pi_{-1}$. For a given value of W , $\Psi(W)$ will fall into one of three intervals with each interval leading to a different treatment decision. The three intervals for $\Psi(W)$ are:

1. $\Psi(W) > 0$: indicating a beneficial effect of the intervention $A = 1$.
2. $\Psi(W) = 0$: indicating no effect of the intervention A .
3. $\Psi(W) < 0$: indicating a harmful effect of the intervention $A = 1$.

Knowledge of $\Psi(W)$ directly relates to knowledge of the optimal treatment.

As noted in [86], the parameter of interest can be expressed as:

$$\Psi(W) = E\left(\left(\frac{I(A = 1)}{\Pi_A} - \frac{I(A = 0)}{1 - \Pi_A}\right) Y|W\right). \quad (4.2)$$

When $\Pi_A = 0.5$, the conditional expectation in equation (4.2) can be modeled with the regression of $Y(A - (1 - A))$ on W . Let $Z = Y(A - (1 - A))$ and since A and Y are binary variables:

$$Z = \begin{cases} +1 & \text{if } Y = 1 \text{ \& } A = 1 \\ 0 & \text{if } Y = 0 \\ -1 & \text{if } Y = 1 \text{ \& } A = 0 \end{cases}$$

The observed values of Z follow a multinomial distribution. The parameter $\Psi(W)$ will be high dimensional in most settings and the components of $\Psi(W)$ are effect modifications between W and the treatment A on the response Y . The parameter can be estimated with a model $\Psi(W) = m(W|\beta)$. The functional form of $m(W|\beta)$

can be specified *a priori*, but since the components of the model represent effect modifications, knowledge of a reasonable model may not be available and we recommend a flexible approach called the super learner (described in the next section) for estimating $\Psi(W)$. In many cases a simple linear model may work well for $m(W|\beta)$, but as the true functional form of $\Psi(W)$ becomes more complex, the super learner gives the researcher flexibility in modeling the optimal treatment function. With the squared error loss function for a specific model $m(W|\beta)$, the parameter estimates are:

$$\beta_n = \arg \min_{\beta} \sum_{i=1}^n (Z_i - m(W_i|\beta))^2 \quad (4.3)$$

The treatment decision for a new individual with covariates $W = w$ is to treat with $A = 1$ if $m(w|\beta_n) > 0$, otherwise treat with $A = 0$.

A normal super learner model for $m(W|\beta)$ would allow for a flexible relationship between W and Z but these models do not respect the fact that $\Psi(W)$ is bounded between -1 and $+1$. The regression of Z on W does not use the information that the parameter $\Psi(W) = \pi_{+1} - \pi_{-1}$ is bounded between -1 and $+1$. The estimates in equation (4.3) have a nice interpretation since the model predicts the additive difference in survival probabilities. In proposing an alternative method, we wanted to retain the interpretation of an additive effect measure but incorporate the constraints on the distributions. Starting with the parameter of interest in equation (4.1) we add a scaling value based on the conditional distribution of Y given W as in:

$$\Psi'(W) = \frac{E_P(Y|A = 1, W) - E_P(Y|A = 0, W)}{E_P(Y|W)} = \frac{\pi_{+1} - \pi_{-1}}{\pi_Y} \quad (4.4)$$

Since $\pi_Y = \Pr(Y = 1|W) = \Pr(Z \neq 0|W)$, the new parameter $\Psi'(W) = E(Z|Z \neq 0, W)$. When we restrict the data to the cases with $Z \neq 0$ (i.e. $Y = 1$) the outcome becomes a binary variable and binary regression methods can be implemented. For example, the logistic regression model:

$$\text{logit}(\Pr(Z = 1|Z \neq 0, W)) = m'(W_i|\beta) \quad (4.5)$$

The treatment decision is based on $m'(W_i|\beta_n) > 0$ where β_n is the maximum likelihood estimate for the logistic regression model. With the binary regression setting, we are now incorporating the distribution information in creating the prediction model, but losing information by working on a subset of the data. These trade-offs depend on the probability π_Y and we will evaluate both methods on the trial example below. In the next section we propose a data-adaptive method for estimating $\Psi(W)$.

4.3 Super Learner

Many methods exist for prediction, but for any given data set it is not known which method will give the best prediction. A good prediction algorithm should be

flexible to the true data generating distribution. One such algorithm is the super learner [89]. The super learner is applied to predict the optimal treatment based on the observed data. The super learner algorithm starts with the researcher selecting a set of candidate prediction algorithms (candidate learners). This list of candidate learners should be selected to cover a wide range of basis functions. The candidate learners are selected prior to analyzing the data; selection of the candidates based on performance on the observed data may introduce bias in the final prediction model. A flow diagram for the super learner algorithm is provided in figure 19.1. With the candidate learners selected and the data collected, the initial step is to fit all of the candidate learners on the entire data set and save the predicted values for $\Psi_n(W) = m_j(W|\beta_n)$, where j indexes the candidate learners. The data is then split into V equal sized and mutually exclusive sets as is typically done for V-fold cross-validation. Patients in the v^{th} fold are referred to as the v^{th} validation set, and all patients not in the v^{th} fold are referred to as the v^{th} training set. For the v^{th} fold, each candidate learner is fit on the patients in the v^{th} training set and the predicted values for $\Psi(W) = m_j(W|\beta_n)$ for the patients in the v^{th} validation set are saved. This process of training the candidate learners on the out of fold samples and saving the predicted values in the fold is repeated for all V folds. The predictions from all V folds are stacked together in a new data matrix X^v . With the prediction data, regress the observed outcome Z on the columns of X^v , which represent the predicted outcomes for each candidate learner. This regression step selects weights for each candidate learner to minimize the cross-validated risk. With the estimates, β_n , from the model $E(Z|X^v) = m(X|\beta)$ the super learner only saves the weights (β_n) and the functional form of the model. The super learner prediction is then based on combining the predictions from each candidate learner on the entire data set with the weights from the cross-validation step.

4.4 Extensions for Censored Data

In a prospective trial the data may be subject to right censoring. In both methods above, right censoring leads to the outcome Z being missing. The data structure is extended to include an indicator for observing the outcome. Let C be the censoring time (for individuals with an observed outcome we set $C = \infty$). Define $\Delta = I(C > t)$. $\Delta = 1$ when the outcome is observed and $\Delta = 0$ when the outcome is missing. The observed data is the set $(W, A, \Delta, Y\Delta)$. For the first method, we propose using the doubly robust censoring unbiased transformation [75]. The doubly robust censoring unbiased transformation generates a new variable Z^* which is a function of the observed data but has the additional property:

$$E(Z^*|W, \Delta = 1) = E(Z|W)$$

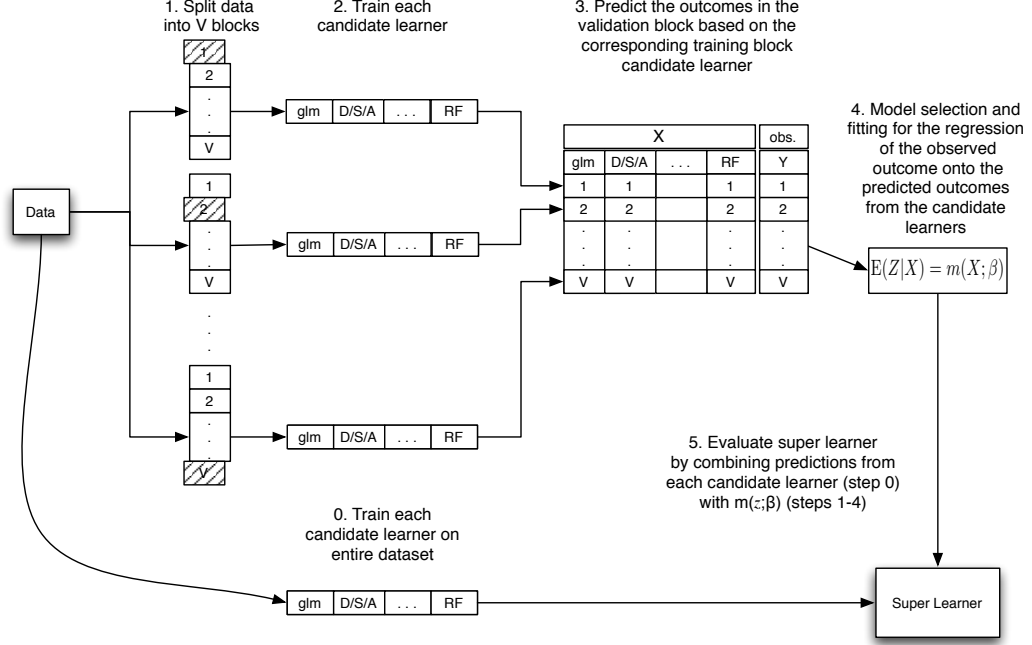


Figure 4.1: Flow diagram for super learner

The transformation allows estimation of the parameter $\Psi(W)$ by applying the super learner on the uncensored observations with the transformed variable Z^* as the outcome. The doubly robust censoring unbiased transformation is:

$$Z^* = \frac{Z\Delta}{\pi(W)} - \frac{\Delta}{\pi(W)}Q(W) + Q(W), \quad (4.6)$$

where $\pi(W) = \Pr(\Delta = 1|W)$ and $Q(W) = E(Z|W, \Delta = 1)$. Both $\pi(W)$ and $Q(W)$ need to be estimated from the data. If either $\pi(W)$ or $Q(W)$ is consistently estimated, then the prediction function $E(Z^*|W, \Delta = 1) = m(W|\beta_n)$ is an unbiased estimate for the true parameter $\Psi(W)$. The censoring mechanism $\pi(W)$ can be estimated with a logistic regression model or a binary super learner on the entire data set. Similarly, $Q(W)$ may be fit with a linear regression model or a super learner, but on the subset of the data with observed values for Z .

For the second method which relies on modeling $E(Z|Z \neq 0, W)$, the main feature was the ability to use the knowledge of the distributions to develop a better model. To retain the binary outcome, the doubly robust censoring unbiased transformation will not work. An alternative method for the right censoring which will retain the binary outcome would be inverse probability of censoring weighting. Inverse probability of censoring weights uses the same $\pi(W)$ as above, but does not incorporate the other nuisance parameter $Q(W)$. When applying the binary super learner for $E(Z|Z \neq 0, W, \Delta = 1)$ the weights $1/\pi(W)$ will be applied for both the candidate learners and

the V-fold cross-validation steps. The super learner will minimize the weighted loss function.

4.5 Simulation Example

We first demonstrate the proposed method on a simulation example where the true value of $\Psi(W)$ is known. The baseline variables were all simulated as normally distributed, $W_j \sim N(0, 1)$, $j = 1, \dots, 10$. The treatment was randomly assigned with $\Pi_A = 0.5$. The true model for the outcome was:

$$\begin{aligned} \Pr(Y = 1|A, W) = g^{-1}(0.405A - 0.105W_1 + 0.182W_2 + 0.039AW_2 \\ + 0.006AW_2W_3 - 0.357AW_4 - 0.020AW_5W_6 - 0.051AW_6) \end{aligned} \quad (4.7)$$

Where $g^{-1}(\cdot)$ is the inverse logit function and W_j refers to the j^{th} variable in W . The true model was selected to include interactions between the treatment and some of the baseline variables. With knowledge of the true model for the outcome Y , the true value of $\Psi(W)$ is calculated for every individual.

The first method involves the regression of Z on W . We applied the super learner for $m(W|\beta)$. 10-fold cross validation was used for estimating the candidate learner weights in the super learner. The super learner for the first method included five candidate learners. The first candidate was ridge regression [46]. Ridge regression used an internal cross validation to select the penalty parameter. Internal cross validation means the candidate learner performed a V-fold cross validation procedure within the folds for the super learner. Structurally, when the candidate learner also performs cross validation within the super learner cross validation we have nested cross validation; therefore, we refer to the candidate learner cross validation as internal cross validation. The second candidate was random forests [11]. For the random forest candidate learner, 1000 regression trees were grown. The third candidate was least angle regression [25]. An internal 10-fold cross validation procedure was used to determine the optimal ratio of the L1 norm of the coefficient vector compared to the L1 norm of the full least squares coefficient vector. The fourth candidate was adaptive regression splines for a continuous outcome [34]. The final candidate was linear regression. Table 4.1 contains reference for the R packages implemented for the candidate learners in the super learner.

The prediction model from the super learner is:

$$\Psi_n(W) = -0.01 + 7.24(X_n^{ridge}) + 1.16(X_n^{rf}) - 0.20(X_n^{lars}) - 7.07(X_n^{lm}) - 0.03(X_n^{mars})$$

Where X_n^j is the predicted value for Z based on the j^{th} candidate learner. $j = ridge$ is the ridge regression model. $j = rf$ is the random forests model. $j = lars$ is the least angle regression model. $j = lm$ is the main effects linear regression model. $j = mars$ is the adaptive regression splines model. The largest weights are for ridge regression

Method	R Package	Authors
Adaptive Regression Splines	<code>polspline</code>	Kooperberg
Least Angle Regression	<code>lars</code>	Efron and Hastie
Penalized Logistic	<code>stepPlr</code>	Park and Hastie
Random Forests	<code>randomForest</code>	Liaw and Wiener
Ridge Regression	<code>MASS</code>	Venables and Ripley

Table 4.1: R Packages for Candidate Learners.

and the linear regression model. For example, the estimates for the linear regression model is:

$$X_n^{lm} = 0.06 + 0.02W_1 + 0.01W_2 - 0.03W_3 - 0.07W_4 + 0.01W_5 + 0.05W_6 \\ - 0.02W_7 - 0.00W_8 - 0.01W_9 - 0.06W_{10}.$$

The linear regression model has the largest coefficient on W_4 , which is the variable with the strongest effect modification with the treatment in the true model (equation (4.7)). The second largest coefficient is on W_{10} which is a variable unrelated to the outcome. The super learner helps smooth over these errors by having multiple candidate learners. For example, W_{10} has a small coefficient (-0.01) in the ridge regression model. When all the candidates are combined into the final super learner prediction model the spurious effect estimates will often disappear resulting in a better predictor. The third largest coefficient from the linear regression model is on W_6 which is also a strong effect modifier in the true model. To evaluate how the super learner is performing in comparison to the other candidate learners, each candidate learner was also fit as a separate estimate. We looked at two risk values, first the $E(\Psi_n(W) - Z)^2$ which was minimized by each algorithm. For the simulation, the risk $\hat{E}(Z - \Psi(W))^2 = 0.540$ gives a lower bound for the risk $E(\Psi_n(W) - Z)^2$. Since the true $\Psi(W)$ is known in the simulation, the risk $E(\Psi_n(W) - \Psi(W))^2$ was also evaluated. Table 4.2 contains the risk values for the simulation. The super learner achieved the smallest $E(\Psi_n(W) - Z)^2$ and is comparable to MARS and LARS on the risk for the true parameter value $\Psi(W)$.

The super learner for the second method included three candidate learners. The first candidate was adaptive regression splines for polychotomous outcomes [53]. The second candidate was the step-wise penalized logistic regression algorithm [65]. The final candidate was main terms logistic regression. The super learner for the second method is:

$$\Psi'_n(W) = -1.20 + 1.43(X_n^{poly}) - 0.50(X_n^{plr}) + 1.61(X_n^{glm})$$

Where X_n^j is the predicted value for Z based on the j^{th} candidate learner. $j = poly$ is the polychotomous adaptive spline model. $j = plr$ is the penalized logistic regression model. $j = glm$ is the main effects logistic regression model.

	$E(\Psi_n(W) - \Psi(W))^2$	$E(\Psi_n(W) - Z)^2$
Super Learner	0.012	0.544
MARS	0.012	0.549
LARS	0.012	0.549
Ridge	0.026	0.558
Linear Model	0.028	0.559
Random Forests	0.038	0.565

Table 4.2: Risk for all candidate learners and the super learner

4.6 Example of Prediction Model on Clinical Trial

A phase III clinical trial was conducted to evaluate a novel treatment for brain metastasis. The study recruited 554 patients with newly diagnosed brain metastasis and the patients were randomized to receive either standard care ($A = 0$) or the novel treatment ($A = 1$). The researchers were interested in determining an optimal treatment to maximize the probability of surviving 6 months from treatment initiation without progression. Of the 554 patients, 246 are censored prior to 6 months. For the 308 patients with an observed 6 month progression time, 130 progressed or died (42.2%). In addition to the treatment and event time data, the researchers collected baseline prognostic and predictive factors on every patient. We apply the super learner to estimate a model for selecting the optimal treatment given a patient's baseline factors. A breakdown of the sample size and treatment allocations available for each method is given in table 4.3.

	total	A	
		0	1
Enrolled	554	275	279
Method 1	308	158	150
Method 2	130	67	63

Table 4.3: Number of subjects in each treatment arm at enrollment and available for each method.

4.6.1 Super learner for optimal treatment decisions

Both methods proposed above were applied to the data. The first method looks for a model of Z on W treating Z as a continuous variable. The second method looks for a model of Z on W conditional on $Z \neq 0$ treating the outcome as binary.

The same super learners from the simulation example above were used here in the trial example. The predicted model for the first method is:

$$\Psi_n(W) = -0.01 + 0.02(X_n^{ridge}) + 1.21(X_n^{rf}) - 0.84(X_n^{lars}) - 0.28(X_n^{lm}) + 0.50(X_n^{mars})$$

Where X_n^j is the predicted value for Z based on the j^{th} candidate learner. $j = ridge$ is the ridge regression model. $j = rf$ is the random forests model. $j = lars$ is the least angle regression model. $j = lm$ is the main effects linear regression model. $j = mars$ is the adaptive regression splines model. The coefficient estimates for each candidate learner from the super learner can be interpreted as a weight for each candidate learner in the final prediction model. Random forests has the largest absolute weight. When interpreting the weights, be cautious of the often near collinearity of the columns of X . To evaluate the super learner in comparison to the candidate learners, a 10-fold cross validation of the super learner and each of the candidate learners themselves was used to estimate $E(\Psi_n(W) - Z)^2$. Table 4.4 contains the risk estimates. For the

Method	Risk
Lars	0.426
Mars	0.426
Super Learner	0.445
Ridge Regression	0.505
Random Forests	0.509
Linear Model	0.525

Table 4.4: 10-fold honest cross validation estimates of $E(\Psi_n(W) - Z)^2$ for the super learner and each of the candidate learners on their own.

trial example, both the lars algorithm and the mars algorithm outperform the super learner. As observed in the simulation, minimizing the risk $E(\Psi_n(W) - Z)^2$ should directly relate to minimizing the risk $E(\Psi_n(W) - \Psi(W))^2$. These cross-validation estimates may be used to select an optimal final model for the treatment decisions.

The second method evaluates $E(Z|Z \neq 0, W) = m'(W|\beta)$. The estimated super learner model for the second method is:

$$\Psi'_n(W) = -0.53 - 0.40(X_n^{poly}) + 0.55(X_n^{plr}) + 0.81(X_n^{glm})$$

Where X_n^j is the predicted value for Z based on the j^{th} candidate learner. $j = poly$ is the polyclass adaptive spline model. $j = plr$ is the penalized logistic regression model. $j = glm$ is the main effects logistic regression model. To compare the two methods, we created a confidence interval at the mean vector for W . Let $\bar{w}$ be the vector of observed means for the baseline variables using all observations in the trial. Confidence intervals were created based on 1000 bootstrap samples of the entire

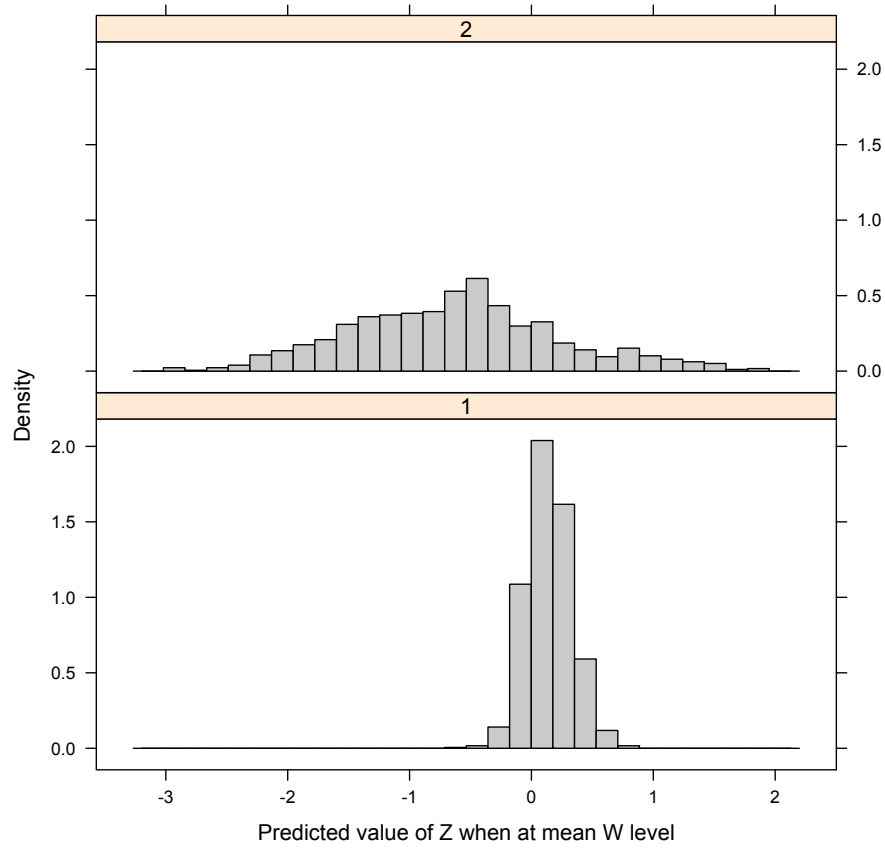


Figure 4.2: Histograms from 1000 bootstrap samples for $\Psi'(W = \bar{w})$ and $\Psi(W = \bar{w})$. The number in the title bar refers to the method used.

super learner. The 95% confidence interval for $m(\bar{w}|\beta)$ based on the first method is $(-0.20, 0.52)$. The 95% for $m(\bar{w}|\beta)$ based on the second method is $(-2.23, 1.23)$. Although the second method is able to use the distributional information, the penalty for the smaller sample size is great (308 patients for the first method down to 130 patients for the second method). As can be seen in figure 19.2, the second method has a wide confidence interval compared to the first method.

4.7 Variable Importance Measure

An additional feature of having a good prediction model is better variable importance measures. The variables in $E(Z|W)$ are effect modifications and when applying the targeted maximum likelihood estimation (TMLE) variable importance measure [91] the results will be causal effect modification importance measures. The targeted maximum likelihood effect modification variable importance allows the researcher to focus on each variable in W individually while adjust for the other variables in W . An initial variable importance estimate is based on an univariate regression, $Z^* = \beta_{0j} + \beta_{1j}W_j$, $j = 1, \dots, p$ where p is the number of baseline covariates in W . The top 5 baseline variables based on the ranks of the univariate p -values is presented in table 4.5. The top unadjusted effect modification variable is an indicator of whether the patients lives in the US or Europe, followed by an indicator for the patients being in RPA class 2, an indicator for the primary tumor being controlled, an indicator for extracranial metastasis, and finally an indicator for the patient's age greater than 65 years. The top 5 baseline variables from the LARS procedure are similar to those from the univariate regression with the exception of Squamous cell indicator replacing the RPA class 2 indicator. For the TMLE variable importance, the effect of W_j on Z is adjusted by all other covariates in W . Let $W_{(-j)}$ be all covariates in W excluding the j^{th} variable. The targeted maximum likelihood variable importance measure as outlined in [5] was then applied using the predictions from the super learner as the initial estimate of $E(Z|W)$. The targeted effect modification parameter is then:

$$\psi_j = E(E(Z|W_j = 1, W_{(-j)}) - E(Z|W_j = 0, W_{(-j)})), \quad j = 1, \dots, p \quad (4.8)$$

The top 5 baseline variables are presented in table 4.5. The effect estimates from the TMLE procedure can be considered causal effect modifiers. Only extracranial mets appears in both the adjusted and unadjusted top 5 list, although squamous cell indicator does appear in both the LARS procedure and the TMLE procedure. The top variable (Mets Dx > 6 Mo) is an indicator for the metastasis diagnosis occurring greater than 6 months after previous cancer. The TMLE list contains two indicators for histology of the tumor cells (Squamous and Adeno carcinoma) suggesting that some tumor types may respond better to the treatment compared to others. Comparing the variable importance lists, the indicator for the patient being in the United States compared to Europe is on top of the list for the univariate regression and the lars

Method	Baseline Variable	Effect	<i>p</i> -value
Univariate Regression	US vs Europe	-0.222	0.007
	RPA class 2	-0.229	0.017
	Primary tumor control	0.165	0.052
	Extracranial mets	-0.133	0.069
	Age > 65 years	-0.157	0.075
LARS	US vs Europe	-0.124	0.350
	Primary tumor control	0.080	0.405
	Age > 65 years	-0.050	0.412
	Extracranial mets	-0.028	0.413
	Squamous cell	0.034	0.419
tMLE	Mets Dx > 6 Mo	0.864	0.001
	Squamous cell	1.012	0.001
	Adeno carcinoma	0.129	0.007
	Extracranial mets	-0.102	0.022
	Caucasian	0.172	0.035

Table 4.5: Top 5 effect modifiers based on univariate regression, lars, and super learner with targeted maximum likelihood. The standard error was based on a bootstrap with 1,000 samples.

model, but absent from the tMLE list. There is no biological evidence for geographical location to interact with the treatment in this trial. The variable importance based on targeted maximum likelihood is able to appropriately adjust for the confounding on the other variables in W and remove the US versus Europe indicator from the list of top variables. The variable importance list from the tMLE has a better interpretation and is informative as to which patient characteristics have a causal interaction with the treatment.

4.8 Discussion

Two methods were proposed for predicting the optimal treatment based on baseline factors. The first method involves modeling Z on W disregarding the knowledge that $E(Z|W)$ is bounded between -1 and $+1$. The second method incorporates the bounds, but does so at a cost in sample size by modeling $E(Z|Z \neq 0, W)$. The second method predicts a scaled version of the parameter of interest, and so is still valid for making treatment decisions. In the simulation and trial example presented here, the loss of sample size in the second method greatly increased the variability of the final prediction. But both the simulation and trial example had a high fraction of patients with $Z = 0$ (equivalently, $Y = 0$). The second method may outperform the first method in settings where $\Pr(Y = 0)$ is very small. For the examples presented here, no problems were observed with the first method not respecting the bounds on $E(Z|W)$.

In the trial example, the super learner did perform better than the main terms linear regression based on the estimate of the risk $E(\Psi_n(W) - Z)^2$. Even though the super learner has shown to have excellent performance across a range of simulations [89, 80] and in various of our data analyses in breast cancer research, there is a risk that the super learner will result in a slight over-fit. In the data analysis we observed that the super learner was ranked third, but competitive with the top two candidate learners, LARS and MARS. We have also proposed an extension to the super learner outlined here to adaptively select the number of candidates [69] so that the weaker candidates are not selected, which we believe will protect the super learner against possible over-fitting, but this was not implemented in the current data analysis yet.

We observed that the difference in sample size between the two methods may make the second method unusable in this example, but the two methods also differed in the treatment of right censoring. The first method incorporated the doubly robust censoring unbiased transformation while the second method used the inverse probability of censoring weights. If the model for $Q(W)$ was correctly specified, but the model for the censoring mechanism was not consistently estimating $\pi(W)$, the doubly robust estimator would still be unbiased but the inverse probability of censoring weighted method will be biased. Alternatively, if $\pi(W)$ was correctly specified, but $Q(W)$ was inconsistent, then both methods will be unbiased. The doubly robust

transformation gives the researcher two chances to correctly the nuisance parameters, while the inverse weighting method relies solely on the model for $\pi(W)$. When there is uncertainty regarding the model for the censoring mechanism, the doubly robust transformation is preferred.

The methods presented above are not limited to randomized clinical trials. Optimal treatment prediction models could also be estimated from observational or registry data sets. As long as the variables needed to estimate $\Pr(A = 1|W)$ are collected in the study the above methods easily extend to the non-randomized setting. Registry data sets are often larger than randomized trials and therefore have more power to detect the interaction effects necessary for predict optimal treatments.

Chapter 5

Super Learner for Censored Data

5.1 Introduction

The super learner was introduced in [89] and shown to be an asymptotically optimal framework for learning from data. This chapter demonstrates how the super learner framework can be applied to censored data. The main difference in this chapter compared to the previous chapters is building a loss function appropriate for the problem. As with the super learner above, the framework can be applied to hazard estimation (density estimation) or survival estimation (prediction).

5.1.1 Data structure

Let T be the survival time and $W = (W(1), W(2), \dots, W(p))$ a set of p baseline covariates. The full data, $X_i = \{T_i, W_i\}$, comes from the data generating distribution F_X which is known to belong to the statistical model $\mathcal{M}^F$. The survival time is possibly right censored by the censoring time C . Observe the right-censored data:

$$O_i = \left\{ W_i, \tilde{T}_i = \min(T_i, C_i), \Delta = \mathbf{I}(\tilde{T}_i = T_i) \right\} \quad i = 1, \dots, n \quad (5.1)$$

Where $O_i \sim P_S$. The discrete or continuous conditional hazard for the survival and censoring mechanisms are denoted $\lambda_T(t | W)$ and $\lambda_C(t | W)$, respectively. We assume coarsening at random for the censoring distribution, that is, C is independent of T given W . Denote the censoring mechanism $G_0(t | X)$. The coarsening at random assumption allows the density of the observed data to be factorized into F_X and G_0 components. Therefore, The distribution of the observed data is often indexed by the full data distribution and the censoring mechanism $P_O = P_{F_X, G_0}$ (see [39] for details).

There are two commonly used parameter of interests for survival outcomes. The first is to predict the expected log survival time given the baseline covariates directly, $\psi_0(W) = E(\log(T) | W)$. The second is to predict the conditional hazard (conditional density) of the survival distribution, $\psi_0(W) = f(T | W)$. In both of these examples

ψ_0 is a parameter of the full data distribution F_X and the parameter space is defined as $\Psi = \{\psi(W \mid F) : F \in \mathcal{M}^F\}$.

5.1.2 Previous work

Estimator selection for survival outcomes has seen less attention compared to estimator selection for continuous and categorical (non-censored) outcomes. Notable examples of estimator selection for censored data include the cross-validation double robust selector from [49]. The authors proposed the inverse probability of censoring weighted (IPCW) loss function and the corresponding double robust locally efficient estimation method. Alternative methods involve extensions of the Cox proportional hazards models. For example, [45] proposed using additive models with smoothing splines for the covariates. [54] similarly uses polynomial splines but with a sieve of multiplicative intensity models on the hazard. Other methods involve adding a penalty term on the partial likelihood as in the LASSO ([85] and [97]) or other penalties such as Akaike's information criterion ([2]). Tree based methods have also been proposed, often referred to as survival trees or survival forests ([56], [77], [47], and [48]). Cross-validated Cox regression is described in the context of penalized partial likelihoods in [93].

5.1.3 Cross-validation for censored outcomes

Cross-validation for censored outcomes is extensively examined in [87]. For a general parameter of interest defined as the minimizer of the expected loss function,

$$\psi_0 = \arg \min_{\psi} \int L(x, \psi) dF_X \quad (5.2)$$

we define a library of estimators $\psi_k(W)$, $k = 1, \dots, K(n)$. The library may contain estimators based on different subsets of W , different basis functions, different estimation procedures, among other things. The classical estimator selection problem seeks to find $\hat{k}$ from the list $\{1, 2, \dots, K(n)\}$ such that the additive difference in expected loss,

$$d_n(\hat{\psi}_{\hat{k}}, \psi_0) = \int \left\{ L(X, \hat{\psi}_{\hat{k}}) - L(X, \psi_0) \right\} dF_X \quad (5.3)$$

is minimized with the selected $\hat{k}$. There are two problems with this setup. The first is that the loss function is defined on the full (unobserved) data. The second problem is that the expected loss depends on the true (unknown) data generating distribution. We address these two problems below.

mapping full data loss function into observed data loss function (fill in details here) ([90]). Replace the full data loss function, $L(X, \psi)$ with the function $L(O, \psi \mid \eta)$ which is a loss function defined on the observed data possibly with nuisance parameters η .

When the parameter of interest is the conditional expected (log) survival time, $\psi_0 = E(\log(T) \mid W)$, a commonly used loss function is the IPCW squared error loss function,

$$L(O, \psi) = \frac{\Delta}{\bar{G}(T)} \{\log(T) - \psi\}^2 \quad (5.4)$$

where $\bar{G}(T)$ is the survival function for the censoring mechanism. Since the censoring mechanism is often not known, $\bar{G}(T)$ is a nuisance parameter for the loss function.

When the parameter of interest is the conditional hazard function, $\psi_0 = \lambda(T \mid W)$, two possible loss functions are the following:

$$\begin{aligned} L_{\loglik}(O, \psi) &= \sum_t \mathbf{I}(\tilde{T} \geq t) \log(\psi)^{dN(t)} \log(1 - \psi)^{1-dN(t)} \\ L_{L2}(O, \psi) &= \sum_t \mathbf{I}(\tilde{T} \geq t) \{dN(t) - \psi\}^2 \end{aligned}$$

where $dN(t) = \mathbf{I}(\tilde{T} = t, \Delta = 1)$. For the second problem of estimating the expected loss we propose using the cross-validation risk estimator. Formally, let $B_n \in \{0, 1\}^n$ be a random vector.

V-fold cross-validation is an example of a cross-validation procedure. Randomly partition the observations into V mutually exclusive and exhaustive sets of approximately equal size. Define the random splitting vector as:

$$B_n(i) = \begin{cases} 0, & O_i \text{ in Training set} \\ 1, & O_i \text{ in Validation set} \end{cases}$$

The distribution of B_n places mass $1/V$ on the binary vectors $b_n^\nu = (b_n^\nu(i) : i = 1, \dots, n)$ for $\nu = 1, \dots, V$, such that $\sum_i b_n^\nu(i) \approx n/V$, $\forall \nu$ and $\sum_\nu b_n^\nu(i) = 1$, $\forall i$. The proportion of observations in the validation set is defined as $p_n \equiv p = n/V$. Let $T(\nu) = \{i \in \{1, \dots, n\} : b_n^\nu(i) = 0\}$ be the set of observations in the ν th training set and $V(\nu) = \{i \in \{1, \dots, n\} : b_n^\nu(i) = 1\}$ be the corresponding validation set. The empirical distributions of the training and validation samples for a given fold are denoted P_{n, B_n}^0 and P_{n, B_n}^1 , respectively. A diagram for V-fold cross-validation is presented below (Figure 5.1).

With the sample splits and the corresponding empirical distributions for the training and validation splits, define the cross-validated risk estimate as:

$$\hat{\theta}_{n(1-p)}(k) = E_{B_n} \int L(X, \psi_k(\cdot \mid P_{n, B_n}^0)) dP_{n, B_n}^1 \quad (5.5)$$

The usual cross-validation selector is then to select the estimator minimizing the cross-validated risk,

$$\hat{k} = \arg \min_{k \in \{1, \dots, K(n)\}} \hat{\theta}_{n(1-p)}(k) \quad (5.6)$$

$$= \arg \min_{k \in \{1, \dots, K(n)\}} E_{B_n} \int L(X, \psi_k(\cdot \mid P_{n, B_n}^0)) dP_{n, B_n}^1 \quad (5.7)$$

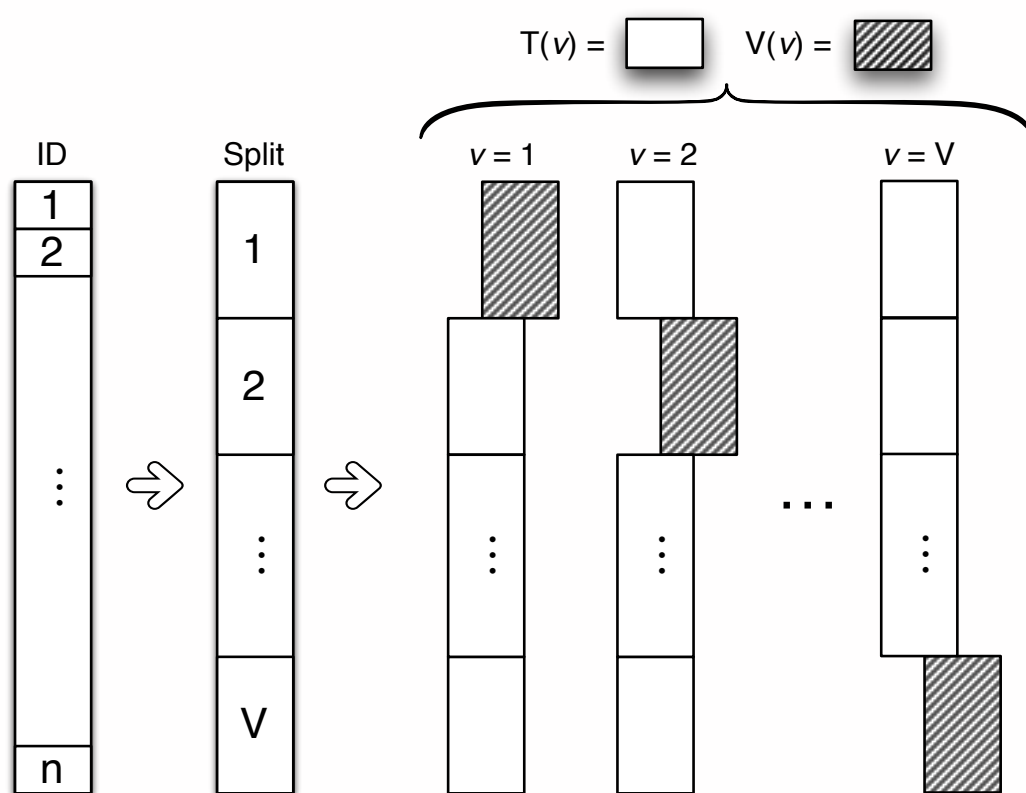


Figure 5.1: V-fold cross-validation diagram

Selection methods are often compared to the so-called oracle selector:

$$\tilde{k} = \arg \min_{k \in \{1, \dots, K(n)\}} E_{B_n} \int L(X, \psi_k(\cdot \mid P_{n, B_n}^0)) dF_X \quad (5.8)$$

The oracle selector knows the true risk for each estimator in the library and can therefore select the best. For any given estimator selection, the performance can be compared against the oracle by the ratio:

$$R_n = \frac{d_n(\hat{\psi}_{\tilde{k}}, \psi_0)}{d_n(\hat{\psi}_{\tilde{k}}, \psi_0)} \quad (5.9)$$

Optimality of an estimator is related to the behavior of R_n as $n \rightarrow \infty$.

5.1.4 Super learner

The super learner was introduced in [89] as a general loss based learning method. The algorithm selects the optimal combination of estimators from the library by minimizing the cross-validated risk. The super learner is motivated by the asymptotic optimality of cross-validation estimator selection as proven in [87] and [88].

The general super learner algorithm starts with the data structure as described above and the parameter of interest defined as the minimizer of the expected loss function. The baseline covariates are denoted W_i and the outcome variable for the given parameter of interest is denoted Y_i (e.g. $Y = \log(T)$ or $Y = dN(t)$). For the given parameter of interest, collect a library of algorithms. The library is denoted $\mathcal{L}$ and contains $K(n)$ elements. Let $\hat{\Psi}_k(P_n) \equiv \hat{\Psi}_k$, $k = 1, \dots, K(n)$ be the mappings from the observed empirical distribution P_n into the parameter space Ψ . Using V-fold cross-validation, each algorithm in the library is fit on the V training samples. Let $\psi_{nk\nu} = \hat{\Psi}_k(P_{nT(\nu)})$ be the fit of the k th estimator when applied to the ν th training sample, $P_{nT(\nu)}$. The estimator from the training sample may then be applied to the corresponding validation sample. The predicted values for the ν th fold are $\{\psi_{nk\nu}(W_i) : i \in V(\nu)\}$ and let Z_{ik} the the V vectors of predicted values stacked together as a single vector. The vectors Z_{ik} can then be combined to form a new data set $Z_i = \{Z_{ik} : k = 1, \dots, K(n)\}$ for $i = 1, \dots, n$. The new data set, Z_i , is combined with the observed outcomes Y_i and a mapping is proposed for the regression function $E(Y \mid Z)$. Let the empirical distribution of the data set $\{(Y_i, Z_i) : \}$ be P_{nZ} , then $\hat{\Psi}_C(P_{nZ}) \equiv \hat{\Psi}_C$ is a mapping from P_{nZ} into the parameter space Ψ . The subscript C is used for this mapping as it “combines” the individual algorithms in the library into a single function. For example, with $\psi = E(Y = \log(T) \mid W)$ a model could be

$$m(Z \mid \alpha) = \sum_{k=1}^{K(n)} \alpha_k z_{ik}, \quad \text{with } \sum_k \alpha_k = 1 \ \& \ \alpha_k \geq 0 \ \forall k \quad (5.10)$$

and the values of the parameters α are estimated by minimizing an appropriate loss function. By our construction of the Z data set, estimating α is equivalent to finding the combination of the algorithms minimizing the V-fold cross-validated risk.

We now have the mappings for each algorithm in the library on the full data set, $\hat{\Psi}_j(P_n)$, and the mapping for how to combine the algorithms into a single function, $\hat{\Psi}_C(P_n)$. These mappings can be coupled to form the super learner:

$$\hat{\Psi}_{SL}(P_n)(W) \equiv \hat{\Psi}_C(P_n)\left(\left\{\hat{\Psi}_k(P_n)(W) : k = 1, \dots, K(n)\right\}\right) \quad (5.11)$$

Figure 5.2 contains a diagram of the super learner algorithm.

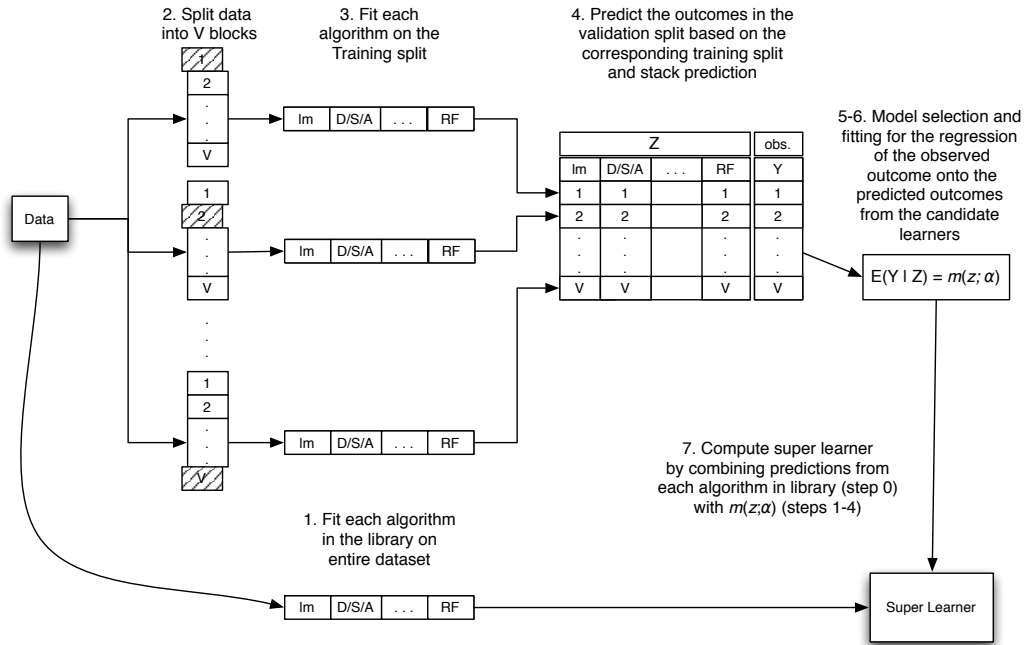


Figure 5.2: Super learner diagram

For the mapping $\hat{\Psi}_C$ in the example above, the convex combination was used as the parametric model. The parameter space for this model is $\mathcal{A} = \{\alpha : \sum_k \alpha_k = 1 \text{ \& } \alpha_k \geq 0 \forall k\}$. If the parameter space was further constrained to be the nodes of the simplex the super learner would reduce to the usual V-fold cross-validation selector described above. The super learner generalizes the cross validation selector and therefore we refer to the cross-validation selector as the discrete super learner.

5.2 Super learner for hazard estimation

With the observed follow-up time, define a grid $t = \{0 = t_0, t_1, \dots, t_m = \max(\tilde{T}\Delta)\}$. Let $\delta_j = t_{j+1} - t_j$. Let $\delta_j = \delta$ or the δ_j may be adaptively defined based on the support

of the data by dividing the survival times into quantiles. Define the change in the counting process:

$$dN_i(t_j) = \begin{cases} 1 & \text{if } t_{j-1} \leq \tilde{T}_i < t_j \text{ and } \Delta_i = 1, \\ 0 & \text{if } t_j < \tilde{T}_i \end{cases} \quad (5.12)$$

With the discrete survival process, $dN_i(t_j)$, define the conditional hazard

$$\lambda_{ij}(W) = \Pr(dN_i(t_j) = 1 \mid W_i, t_j, dN_i(t_{j-1}) = 0)$$

Observe the right-censored data $X = (W, \tilde{T} = \min(T, C), \Delta = \mathbf{I}(\tilde{T} = T)) \sim P_X$ where W are the baseline covariates, T is the true survival time, C is the censoring time (let $C = \infty$ if $T \leq C$) and Δ is the event indicator.

As described in detail by [89], the super learner requires 3 components:

1. A library of estimators
2. A loss function
3. A model for combining the estimators in the library

These three components should be tailored to the specific problem. The library should be built such that each estimator in the library could individually estimate the parameter of interest. The loss function should be chosen to reflect the goals of the problem. If one is interested in estimating the hazard, the loss function should ideally involve the hazard; if interest in the survival function, the loss function should involve the survival function. This may seem common sense, but we often see researchers use a hazard loss function when the interest is on the survival probability at a specific time point (e.g. 10 year survival). The loss function should be chosen with respect to the problem you are trying to solve. The model for combining the estimators in the library is often chosen to maintain a bounded loss function for the super learner.

Although the Super Learner is motivated with a general loss function, we examine two specific examples for this manuscript. The first loss function is the squared error or L_2 loss

$$L_{L2}(O, \lambda) = \sum_t \mathbf{I}(\tilde{T} \geq t) (dN(t) - \lambda)^2 \quad (5.13)$$

The second loss function is the Binomial log-likelihood loss

$$L_{LL}(O, \lambda) = - \sum_t \left\{ \mathbf{I}(\tilde{T} \geq t) \log (\lambda^{dN(t)} (1 - \lambda)^{1-dN(t)}) \right\} \quad (5.14)$$

The oracle inequality results for super learner assume the loss function is bounded ([87]). We propose constraints on the model for combining the algorithms in the library to maintain a bounded loss function. For the L_2 loss function, consider the

model for the super learner $\lambda_{SL-L2} = \sum_k \alpha_k \lambda_k, k = 1, \dots, K$. If we constrain the space for α to be $\alpha_k \geq 0$ and $\sum_k \alpha_k = 1$ then the loss function will be bounded. Since $0 \leq \lambda_k \leq 1$, the convex combination will also be bounded between 0 and 1.

$$\begin{aligned} \max |L_{L2}(O, \lambda_{SL-L2})| &= \max_t \sum I(Y \geq t) (dN(t) - \lambda_{SL-L2})^2 \\ &\leq \sum_t \max I(Y \geq t) (dN(t) - \lambda_{SL-L2})^2 \\ &\leq \sum_t 1 \\ &= T^* \end{aligned}$$

where T^* is the total number of discrete time points for the counting process and is controlled by the researcher.

For the log-likelihood loss function we propose using the logit link function $g(x) = \log(x/(1-x))$ and to use the convex combination on the logit scale and then transform back into a probability with the inverse logit function.

$$\lambda_{SL-LL} = g^{-1} \left(\sum_k \alpha_k g(\lambda_k) \right), \quad \text{with } \sum_k \alpha_k = 1 \text{ \& } \alpha_k \geq 0 \forall k \quad (5.15)$$

A problem occurs when λ approaches either 0 or 1 then $g(\lambda)$ approaches $\pm\infty$. To avoid this problem, we will use the symmetric truncated logit link,

$$g^*(x, c) = \begin{cases} g(c) & \text{if } x < c \\ g(x) & \text{if } c \leq x \leq 1 - c \\ g(1 - c) & \text{if } x > 1 - c \end{cases} \quad (5.16)$$

for a small constant c .

$$\begin{aligned} \max |L_{LL}(\lambda_{SL-LL})| &= \max_t \sum \left\{ I(Y \geq t) \log \left(\lambda_{SL-LL}^{dN(t)} (1 - \lambda_{SL-LL})^{1-dN(t)} \right) \right\} \\ &\leq \sum_t \max I(Y \geq t) \log \left(\lambda_{SL-LL}^{dN(t)} (1 - \lambda_{SL-LL})^{1-dN(t)} \right) \\ &\leq \sum_t g(1 - c) \\ &= T^* \times g(1 - c) \end{aligned}$$

As long as c is selected such that $g(1 - c) < M/T^*$ then the loss will be bounded by M .

5.3 Data Analysis

To examine the performance of the super learner with censored data two oncology trials are investigated. The first example uses baseline patient characteristics to predict the hazard function for patients with advanced lung cancer. The second example creates a treatment decision algorithm for patients with low grade breast cancer.

5.3.1 Lung Cancer Example

To demonstrate the super learner on censored data we examined the North Central Cancer Treatment Group lung cancer data set [60]. The data are available in the `survival` package in R [84]. The data contain the survival time (in days) for 228 patients with advanced lung cancer. In addition to the survival time, information on the patient's age, sex, and three performance scores were included. The parameter of interest is the hazard function given the patient's age, sex, and performance scores. Five patients were removed from the analysis set because of incomplete information on the covariates. With the 223 patients in the analysis set, 63 were right censored and 160 had observed death times. The squared error loss function on the hazard,

$$L_{L2}(O, \psi) = \sum_t \mathbb{I}(\tilde{T} \geq t) \{dN(t) - \psi(W)\}^2 \quad (5.17)$$

was used for the super learner.

The first step was to convert the right censored survival time, $\tilde{T}$, into the change in counting process, $dN(t)$. A grid of 30 time points was created using the quantiles of the observed death times and then $dN(t)$ was created using Equation 5.12. Using $dN(t)$ as the outcome, the repeated measures binary outcome super learner was used to estimate the hazard function. The library of estimators consisted of logistic regression, random forests[11], generalized additive models (gam)[45], polyclass[53], DSA[81], neural networks (nnet)[73], and a null model only using time (no covariates). For most hazard estimation problems, time is one of the most important variables to consider in the estimator. We considered modeling time with a few approaches. One approach was to use an indicator for each time point. A logistic regression model with an indicator for each time point will converge to the Cox proportional hazards model. Another approach was to add an additional step of smoothing in time using a generalized additive model. These estimators in the library involved a two-stage estimation procedure. The first part used the covariates (including time) to estimate the hazard of death. Then in the second part, the predicted probabilities from each learner, $\{\pi_{n,ij}^k : k = 1, \dots, K(n)\}$, are used as an offset in a generalized additive logistic model with m degrees of freedom for time:

$$\text{logit}(dN(t)) = s^k(t, m) + \pi_{n,ij}^k, \quad \text{for } k = 1, \dots, K(n) \quad (5.18)$$

The degrees of freedom tuning parameter m is not known *a priori*, but different values of m simply represent different estimators in the library and we allow super learner to decide which estimators should receive weight in the final estimator. The estimators in the super learner library are the binary estimators for the covariates coupled with the generalized additive models for the time trend. Table 5.1 contains a list of the estimators used in the super learner library. The first column is the algorithm used for the covariates and the second column indicates if any additional step was taken to estimate the effect of time.

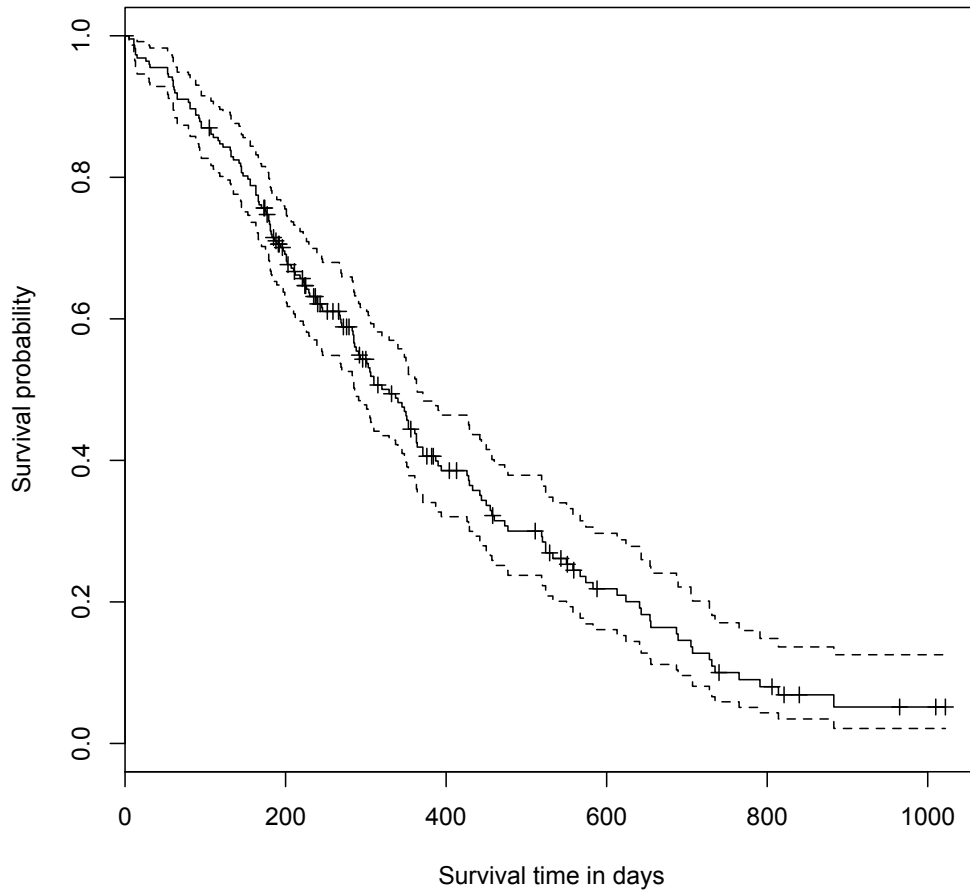


Figure 5.3: Marginal Kaplan Meier plot for the lung cancer survival times shown with a 95% confidence interval at each time point. (+) sign indicates a censoring time.

The estimated coefficients for the super learner are,

$$\begin{aligned}\Psi_{n,SL} = & 0.182\Psi_n(\text{glm, no}) + 0.182\Psi_n(\text{gam only time, } df = 5) \\ & + 0.581\Psi_n(\text{polyclass, no}) + 0.056\Psi_n(\text{glm 2-way, } df = 5)\end{aligned}$$

Where $\Psi_n(a, b)$ represents the fit of algorithm a using the smoothing in time method b . The only estimators to receive non-zero weight in the final super learner model are logistic regression using main terms and no smoothing, a gam model only using time and the degrees of freedom 5, polyclass with no additional smoothing, and a logistic regression model with all two-way interaction (including time) combined with smoothing time using $df = 5$. Not all estimators in the super learner have non-zero weight in this example. Of the 29 estimators in the library, only 4 have non-zero weight. With the constraint that the coefficients for the estimators in the super learner must be non-negative and sum to one and with the near collinearity of the library estimators, the super learner returns a sparse solution giving non-zero weight to many of the estimators in the library.

The standard error (SE) for the V-fold cross-validated risk estimate are based on the results of Theorem 3 in [23]. The authors prove the consistency and asymptotic linearity of the cross-validated risk estimate, $\hat{\theta}_n$, of the conditional risk, $\tilde{\theta}_n$, for general estimators $\hat{\Psi}(P_{n,B_n}^0)$ representing mappings from the training data into the parameter space (e.g. the super learner). With the assumption that the loss function is bounded, that is $\sup_{X, \psi \in \Psi} L(X, \psi) \leq M < \infty$, the asymptotic linearity of the cross-validated risk is proven,

$$\hat{\theta}_n - \tilde{\theta}_n = \frac{1}{n} \sum_{i=1}^n (L(O_i, \Psi(P)) - \theta) + o_P(1/\sqrt{n}) \quad (5.19)$$

We define the influence curve for the risk as the θ -centered loss function

$$IC(X | P) \equiv L(X, \Psi(P)) - \theta \quad (5.20)$$

Then by the definitions of the parameter of interest,

$$E(IC(X | P)) = 0 \quad \text{and} \quad \sigma^2 = \text{Var}(IC(X | P)) = \int IC^2(x | P) dP(x) \quad (5.21)$$

Then by the central limit theorem and as the sample size goes to infinity,

$$\frac{\sqrt{n}(\hat{\theta}_n - \tilde{\theta}_n)}{\sigma} = Z_n \quad (5.22)$$

where $Z_n \rightarrow N(0, 1)$ as $n \rightarrow \infty$. The standard error is then found by plugging in the cross-validated risk estimates into the above influence curve and computing the sample standard deviation,

$$SE_n = \sqrt{\int IC^2(x | P_n^0) dP_n^1} \quad (5.23)$$

Table 5.1: Honest 10-fold cross-validated risk estimates for the super learner, each algorithm in the library, and the Weibull proportional hazards model. All algorithms included time as a covariate, the second column (Time) denotes if any additional smoothing for time was part of the given estimator and the value for the degrees of freedom if used.

Algorithm	Time	Risk	SE
Super Learner		0.6548	0.0258
Discrete SL		0.6589	0.0261
glm	no smooth	0.6534	0.0260
glm	df = 1	0.6534	0.0260
glm	df = 2	0.6541	0.0260
glm	df = 3	0.6548	0.0261
glm	df = 4	0.6556	0.0261
glm	df = 5	0.6564	0.0261
glm	indicator	0.6700	0.0266
glm (2-way interactions)	df = 5	0.6569	0.0261
randomForest	no smooth	0.7628	0.0313
randomForest	df = 2	1.0323	0.0607
randomForest	df = 3	1.0364	0.0627
randomForest	df = 4	1.0483	0.0628
randomForest	df = 5	1.0362	0.0608
gam	df = 2	0.6558	0.0260
gam	df = 3	0.6563	0.0260
gam	df = 4	0.6570	0.0261
gam	df = 5	0.6577	0.0261
gam(df = 2)	no smooth	0.6554	0.0260
gam(df = 3)	no smooth	0.6579	0.0261
gam(df = 4)	no smooth	0.6619	0.0263
gam(df = 5)	no smooth	0.6554	0.0260
gam (only time, df = 3)	no smooth	0.6548	0.0257
gam (only time, df = 4)	no smooth	0.6556	0.0257
gam (only time, df = 5)	no smooth	0.6541	0.0256
polyclass	no smooth	0.6570	0.0258
DSA	no smooth	0.6671	0.0270
DSA	df = 5	0.6669	0.0269
nnet	no smooth	0.7175	0.0302
Weibull PH model		0.7131	0.0300

5.3.2 Breast Cancer Example

Ductal carcinoma in situ (DCIS) is considered a low grade breast cancer. Many prediction models for treatment decisions have been developed for invasive breast cancer (e.g. Adjuvant! online [64]), but none are specific for DCIS. Some studies have tried to identify characteristics of women with DCIS who are likely to have a subsequent cancer event and would therefore be candidates for additional treatments [50].

Two randomized studies from the national surgical adjuvant breast project (NS-ABP) are combined into a single data set. The two studies are Protocol B-17 ([27], [29]) and protocol B-24 ([30], [28]). Protocol B-17 was a randomized study for women with DCIS assigned to receive either lumpectomy only or lumpectomy plus radiation therapy. Protocol B-24 was also a randomized trial conducted after B-17 and the treatment arm from B-17 is used as the control arm in this new study. Women with DCIS were assigned to receive either lumpectomy plus radiation therapy or lumpectomy plus radiation therapy plus Tamoxifen. In combining the two trials, the treatment groups for lumpectomy plus radiation therapy were merged. In addition to the therapy received, 5 tumor characteristics were available for the construction of the prediction model. The tumor characteristics are tumor size, comedo necrosis, margin status, nuclear grade, and if the tumor was palpable. Table 5.2 contains a summary of the covariates in the full data and within treatment groups. Details on the unknown values can be found in the articles cited above. For the models, unknown is treated as a level for the ordinal variables as we did not believe missingness was completely at random.

5.3.3 Selection of loss function

In the DCIS setting, the 10 year landmark is commonly used to assess treatments. A loss function that focuses on the 10 year survival rate estimates is the IPCW squared error loss on the 10 year survival probability,

$$L(O, \psi) = \frac{\Delta}{\bar{G}(t=10)} \left\{ I(\tilde{T} > 10) - \psi(W) \right\}^2 \quad (5.24)$$

where $\psi(W) = \Pr(T > 10 | W)$ is the conditional 10 year all-cause survival probability. The loss function in equation 5.24 is used for the super learner to determine the combination of the learners in the library,

$$\psi_{SL}(W) = \sum_{k=1}^K \alpha_k \psi_k(W) \quad (5.25)$$

where the weights for each algorithm will be constrained to all be non-negative and sum up to one. For the super learner, each estimator in the library estimated the hazard function and then mapped into the 10 year survival probability.

Table 5.2: Both Trials combined into a single data set. Summary statistics for the full data and the 3 treatment groups, count and percentage for categorical variables, mean and standard deviation for age. Treatment A is lumpectomy only, treatment B is lumpectomy plus radiation therapy, treatment C is lumpectomy plus radiation therapy plus Tamoxifen

Variable	Overall (n=2612)	Trt A (n=403)	Trt B (n=1310)	Trt C (n = 899)
Age	55.10 (10.31)	55.22 (10.03)	54.87 (10.38)	55.39 (10.32)
Tumor size (cm)				
≤ 1.0	2112 (0.809)	297 (0.737)	1050 (0.802)	765 (0.851)
1.1 – 2.0	301 (0.115)	60 (0.149)	158 (0.121)	83 (0.092)
≥ 2.1	145 (0.056)	32 (0.079)	72 (0.055)	41 (0.046)
Unknown	54 (0.021)	14 (0.035)	30 (0.023)	10 (0.011)
Comedo Necrosis				
Abs/slight	1258 (0.482)	164 (0.407)	625 (0.477)	469 (0.522)
Moderate/marked	1128 (0.432)	139 (0.345)	575 (0.439)	414 (0.461)
Unknown	226 (0.087)	100 (0.248)	110 (0.084)	16 (0.018)
Margin Status				
Negative	1860 (0.712)	164 (0.407)	943 (0.720)	668 (0.743)
Positive	392 (0.150)	139 (0.345)	199 (0.152)	139 (0.155)
Unknown	360 (0.138)	100 (0.248)	168 (0.128)	92 (0.102)
Nuclear Grade				
Poor	980 (0.375)	142 (0.352)	517 (0.395)	321 (0.357)
Good	1075 (0.412)	161 (0.400)	524 (0.400)	390 (0.434)
Unknown	557 (0.213)	100 (0.248)	269 (0.205)	188 (0.209)
Palpable Tumor				
No	2143 (0.820)	325 (0.806)	1085 (0.828)	733 (0.815)
Yes	469 (0.180)	78 (0.194)	225 (0.172)	166 (0.185)

Table 5.3: Trial 17, count and percentage for categorical variables (mean and sd for age)

Variable	Overall (n=813)	Placebo (n=403)	Treatment(n=410)
Age	54.81 (10.07)	55.22 (10.03)	54.40 (10.11)
Tumor size			
≤ 1.0	604 (.743)	297 (.737)	307 (.749)
1.1 – 2.0	114 (.140)	60 (.149)	54 (.132)
≥ 2.1	67 (.082)	32 (.079)	35 (.085)
missing	28 (.034)	14 (.035)	14 (.034)
Comedo Necrosis			
Yes	281 (.346)	139 (.345)	142 (.346)
Missing	190 (.234)	100 (.248)	90 (.220)
Margin Status			
Positive	107 (.132)	54 (.134)	53 (.129)
Missing	190 (.234)	100 (.248)	90 (.220)
Nuclear Grade			
Good	321 (.395)	161 (.400)	160 (.390)
Missing	190 (.234)	100 (.248)	90 (.220)
Palpable Tumor			
Yes	159 (.196)	78 (.194)	81 (.198)
DFS Event			
Alive	383 (.471)	167 (.414)	216 (.527)
Inv. ipsi. BC	122 (.150)	78 (.194)	44 (.107)
Non-Inv. ipsi. BC	97 (.119)	61 (.151)	36 (.088)
Inv. contra. BC	49 (.060)	25 (.062)	24 (.059)
Non-Inv. Contra. BC	21 (.026)	7 (.017)	14 (.034)
other BC	17 (.021)	7 (.017)	10 (.024)
second primary C	62 (.076)	27 (.067)	35 (.085)
Dead (no sign C)	62 (.076)	31 (.077)	31 (.076)

Table 5.4: Trial 24, count and percentage for categorical variables (mean and sd for age)

Variable	Overall (n=1799)	Placebo (n=900)	Treatment(n=899)
Age	55.24 (10.41)	55.08 (10.50)	55.39 (10.32)
Tumor size			
≤ 1.0	1508 (.838)	743 (.826)	765 (.851)
$1.1 - 2.0$	187 (.104)	104 (.116)	83 (.092)
≥ 2.1	78 (.043)	37 (.041)	41 (.046)
missing	26 (.014)	16 (.018)	10 (.011)
Comedo Necrosis			
Yes	847 (.471)	433 (.481)	414 (.461)
Missing	36 (.020)	20 (.022)	16 (.018)
Margin Status			
Positive	285 (.158)	146 (.162)	139 (.155)
Missing	170 (.094)	78 (.087)	92 (.102)
Nuclear Grade			
Good	754 (.419)	364 (.404)	390 (.434)
Missing	367 (.204)	179 (.199)	188 (.209)
Palpable Tumor			
Yes	310 (.172)	144 (.160)	166 (.185)
DFS Event			
Alive	1182 (.657)	565 (.628)	617 (.686)
Inv. ipsi. BC	136 (.076)	78 (.087)	58 (.065)
Non-Inv. ipsi. BC	127 (.071)	68 (.076)	59 (.065)
Inv. contra. BC	76 (.042)	46 (.051)	30 (.033)
Non-Inv. Contra. BC	38 (.021)	24 (.027)	14 (.016)
other BC	17 (.009)	10 (.011)	7 (.008)
second primary C	128 (.071)	59 (.066)	69 (.077)
Dead (no sign C)	95 (.053)	50 (.056)	45 (.050)

Using the IPCW squared error loss function for the super learner with each algorithm in the library estimating the hazard function and then mapping into the 10 year survival probability, the super learner performed almost as well as the discrete super learner. The results for the super learner can be found in Table 5.5.

Table 5.5: Honest 10 fold cross validated risk estimates for the super learner and each algorithm in the library. The loss function is the 10 year IPCW squared error loss on the survival probability.

Method	Risk	SE
Super learner	1.777	0.0624
Discrete super learner	1.735	0.0745
Glm	1.991	0.0745
Random Forest	1.735	0.0745
CART	1.778	0.0633
Bayes glm	1.987	0.0742
Only time	1.778	0.0633

5.4 Discussion

With an appropriate loss function, the super learner algorithm naturally extends to censored data. The focus may be on either hazard or survival estimation. With an appropriate loss function, the asymptotic optimality of the super learner naturally apply to the setting of censored data. Although not used in the example here, but the ICPW loss function often requires estimation of the censoring mechanism, a nuisance parameter. The super learner can be used to estimate the nuisance parameters for the loss function.

Chapter 6

Software to Implement the Super Learner

6.1 Using the package

```
> library(SuperLearner)
```

```
Super Learner
```

```
Version: 1.1-18
```

```
package created on 2010-04-14
```

Use `SuperLearnerNews()` to see changes from previous versions

Suggested packages to install for the Super Learner library:

```
glmnet, randomForest, class, gam, gbm, nnet, polspline,  
MASS, e1071, stepAIC, arm, party, DSA, spls, LogicReg,  
multicore, SIS, BayesTree, ipred, mlbench, rpart, caret,  
mda, earth, Hmisc
```

When you load the **SuperLearner** package a few helpful details will be printed. The version number and date are presented along with a message that the only required package, **nnls**, is also loaded. We do not know what prediction algorithms you will require for the super learner you are running in each session so the additional packages for prediction algorithms are not pre-loaded. A message with a list of suggested packages is printed and it is recommended to have these packages installed but they do not need to be loaded each time **SuperLearner** is loaded.

6.1.1 Creating prediction algorithms

The **SuperLearner** package includes many built-in wrapper functions for prediction algorithms. These wrappers can be directly used or easily adapted for specific

examples. For a list of the wrapper functions included in the package, you can use the functions `listWrappers(what = "SL")`

```
> listWrappers(what = "SL")
```

All prediction algorithm wrappers in SuperLearner:

[1] "SL.DSA"	"SL.DSA.2"
[3] "SL.bagTree"	"SL.bagTree.unit"
[5] "SL.bagging"	"SL.bart"
[7] "SL.bayesglm"	"SL.caret"
[9] "SL.caret.rpart"	"SL.cforest"
[11] "SL.cv.spls"	"SL.earth"
[13] "SL.gam"	"SL.gam.3"
[15] "SL.gam.4"	"SL.gam.5"
[17] "SL.gbm.1"	"SL.gbm.2"
[19] "SL.glm"	"SL.glmnet"
[21] "SL.glmnet.alpha25"	"SL.glmnet.alpha50"
[23] "SL.glmnet.alpha75"	"SL.knn"
[25] "SL.knn100"	"SL.knn20"
[27] "SL.knn200"	"SL.knn30"
[29] "SL.knn40"	"SL.knn50"
[31] "SL.knn60"	"SL.knn70"
[33] "SL.knn80"	"SL.knn90"
[35] "SL.loess"	"SL.logreg"
[37] "SL.mars"	"SL.mean"
[39] "SL.nnet"	"SL.nnet.3"
[41] "SL.nnet.4"	"SL.nnet.5"
[43] "SL.polymars"	"SL.polymars.dimreduce"
[45] "SL.rF"	"SL.randomForest"
[47] "SL.ridge"	"SL.rpart"
[49] "SL.spls"	"SL.step"
[51] "SL.step.forward"	"SL.step.interaction"
[53] "SL.step.plr"	"SL.stepAIC"
[55] "SL.svm"	"SL.svm.eps"
[57] "SL.template"	

Table 6.1: Details of prediction algorithm wrappers

Function	Package	Tuning Parameters
bagging	ipred	nbagg minsplit cp maxdepth
bagTree	SuperLearner & rpart	cp minsplit maxdepth ntree weights
bart	BayesTree	ntree sigdf sigquant k power base ndpost nskip
bayesglm	arm	prior.mean prior.scale prior.df
cforest	party	ntree mtry mincriterion teststat testtype replace fraction
cv.spls	spls	K eta
DSA	DSA	maxsize maxorderint maxsumofpow Dmove Smove vfold

(Continued on next page)

Table 6.1: Details of prediction algorithm wrappers

Function	Package	Tuning Parameters
earth	earth	degree penalty nk thresh minspan newvar.penalty fast.k fast.beta nfold pmethod
gam	gam	deg.gam
gbm	gbm	gbm.trees interaction.depth cv.folds shrinkage n.minobsinnode bag.fraction train.fraction
glm	stats	—
glmnet	glmnet	alpha lambda nlambda lambda.min dfmax type
knn	class	k use.all
loess	stats	span family
logreg	LogicReg	degree ntrees nleaves select penalty kfold control

(Continued on next page)

Table 6.1: Details of prediction algorithm wrappers

Function	Package	Tuning Parameters
<code>mars</code>	<code>mda</code>	<code>degree</code> <code>nk</code> <code>penalty</code> <code>thresh</code> <code>prune</code> <code>forward.step</code>
<code>nnet</code>	<code>nnet</code>	<code>size</code> <code>decay</code> <code>rang</code>
<code>polymars</code>	<code>polspline</code>	<code>maxsize</code> <code>gcv</code> <code>additive</code> <code>knots</code> <code>know.space</code>
<code>polyclass</code>	<code>polspline</code>	<code>maxdim</code> <code>cv</code> <code>additive</code>
<code>randomForest</code>	<code>randomForest</code>	<code>ntree</code> <code>mtry</code> <code>nodesizes</code> <code>sampsize</code> <code>replace</code> <code>maxnodes</code>
<code>Ridge</code> <code>rpart</code>	<code>MASS</code> <code>rpart</code>	<code>lambda</code> <code>cp</code> <code>minsplitt</code> <code>xval</code> <code>maxdepth</code> <code>minbucket</code>
<code>step</code>	<code>stats</code>	<code>scope</code> <code>scale</code> <code>direction</code> <code>steps</code> <code>k</code>

(Continued on next page)

Table 6.1: Details of prediction algorithm wrappers

Function	Package	Tuning Parameters
step.plr	stepPlr	type lambda cp max.terms
svm	e1071	type kernel nu degree gamma coef0 cost cachesize tolerance epsilon cross

The package contains a template function for prediction algorithms:

```
> SL.template

function (Y.temp, X.temp, newX.temp, family, obsWeights, id,
  ...)
{
  if (family$family == "gaussian") {
  }
  if (family$family == "binomial") {
  }
  out <- numeric()
  fit <- vector("list", length = 0)
  foo <- list(out = out, fit = fit)
  class(foo$fit) <- c("SL.template")
  return(foo)
}
<environment: namespace:SuperLearner>
```

Each prediction algorithm needs to return a list with 2 elements. The first element, `out`, contains the predicted values for the observations in `newX.temp` based on the model estimated with `X.temp`. The values of `out` should be on the scale of the outcome `Y.temp`. The second element, `fit`, is a list containing all the elements of the fitted

Argument	Description
<code>Y.temp</code>	the outcome variable
<code>X.temp</code>	the training data set (the observations used to fit the model)
<code>newX.temp</code>	the validation data set (the observations to return predictions for)
<code>family</code>	a description of the error distribution
<code>id</code>	a cluster identification
<code>obsWeights</code>	observation weights

Table 6.2: The allowed arguments for a prediction algorithm in **SuperLearner**

model required to predict a new observation. The names in the `fit` list should match the arguments for the predict method associated with the prediction algorithm.

6.1.2 Example: creating prediction wrapper

Consider the `polymars` algorithm in the **polspline** package and the general data setting:

- continuous outcome `Y`
- data.frame of covariates `X`
- data.frame of covariates `newX`

The first step in creating a prediction algorithm wrapper is to know how to use the function to fit a model and predict values on new observations. Reading the help documentation for `polymars` we find the functions are:

```
> fit.mars <- polymars(Y, X)
> out <- predict.polymars(fit.mars, x = as.matrix(newX))
```

These functions fit the model using a numeric `Y` and a data.frame `X` and call that object `fit.mars`. The second line returns the predicted values from the `fit.mars` object on the observations in `newX` that needs to be converted to a matrix before evaluation. The `out` object is a numeric vector with the predicted observations with length the number of rows in `newX`. The next step is to take the template code and insert the `polymars` code:

```
> My.SL.polymars <- function(Y.temp, X.temp, newX.temp, family, ...) {
+   if(family$family=="gaussian"){
+     fit.mars <- polymars(Y.temp, X.temp)
+     out <- predict.polymars(fit.mars, x = as.matrix(newX.temp))
+   }
+ }
```

```
+ if(family$family=="binomial"){
+   # insert estimation function
+ }
+ ... # details below
+ }
```

Insert the `polymars` code into the gaussian family block. Also note that the name of the function needs to be unique. The wrapper name must not conflict with the name of a wrapper contained in the **SuperLearner** package otherwise the function in the package will take precedence over the new version. If only writing the wrapper for a specific problem the block for the binomial family could be ignored (or add the error `stop("only gaussian allowed")`), but if you want to save the new wrapper for future project that may include binomial outcomes we could look up the `polymars` code for a binomial family and the same data structure,

```
> fit.mars <- polyclass(Y, X, cv = 10)
> out <- ppolyclass(cov = newX, fit = fit.mars)[, 2]
```

and add these to the wrapper.

```
> My.SL.polymars <- function(Y.temp, X.temp,
+   newX.temp, family, ...) {
+   if(family$family=="gaussian"){
+     fit.mars <- polymars(Y.temp, X.temp)
+     out <- predict.polymars(fit.mars, x = as.matrix(newX.temp))
+   }
+   if(family$family=="binomial"){
+     fit.mars <- polyclass(Y.temp, X.temp, cv = 5)
+     out <- ppolyclass(cov = newX.temp, fit = fit.mars)[, 2]
+   }
+   ... # details below
+ }
```

All wrappers need to return two values:

1. `out`: predicted Y values for rows in `newX`
2. `fit`: a list with everything needed to use `predict` method

The `out` object is already in the wrapper, but need to figure out the `fit` object. In the `polymars` example

1. For the gaussian case, `predict()` needs: `object = fit.mars`
2. For the binomial case, `predict()` needs: `fit = fit.mars`

Note that `SuperLearner` does not use the `fit` list. If you do not plan to use the function `predict.SuperLearner` you can leave the fit object as: `fit <- vector("list", length = 0)`. This object has been written to be very flexible since various prediction algorithms require a diverse set of objects for predicting with new observations. Now we add the `fit` code to the wrapper:

```
> My.SL.polymars <- function(Y.temp, X.temp, newX.temp,
+ family, ...) {
+   if(family$family=="gaussian"){
+     fit.mars <- polymars(Y.temp, X.temp)
+     out <- predict.polymars(fit.mars, x = as.matrix(newX.temp))
+     fit <- list(object = fit.mars)
+   }
+   if(family$family=="binomial"){
+     fit.mars <- polyclass(Y.temp, X.temp, cv = 5)
+     out <- ppolyclass(cov = newX.temp, fit = fit.mars)[, 2]
+     fit <- list(fit = fit.mars)
+   }
+   ... # details below
+ }
```

The final step is putting everything together into a list object. The list must have 2 elements and the names *must* be `out` and `fit`. One can also assign a class to the `fit` list. This will be used to look up the correct `predict` method. The `SuperLearner` is using S3 methods here. The class assignment is only important if using `predict.SuperLearner` afterwards. The final wrapper is:

```
> My.SL.polymars <- function(Y.temp, X.temp, newX.temp,
+ family, ...) {
+   if(family$family=="gaussian"){
+     fit.mars <- polymars(Y.temp, X.temp)
+     out <- predict.polymars(fit.mars, x = as.matrix(newX.temp))
+     fit <- list(object = fit.mars)
+   }
+   if(family$family=="binomial"){
+     fit.mars <- polyclass(Y.temp, X.temp, cv = 5)
+     out <- ppolyclass(cov = newX.temp, fit = fit.mars)[, 2]
+     fit <- list(fit = fit.mars)
+   }
+   foo <- list(out = out, fit = fit)
+   class(foo$fit) <- c("SL.polymars")
+   return(foo)
+ }
```

This wrapper should match the `SL.polymars` wrapper in the package. To complete the example, the correct predict method for this new wrapper is:

```
> predict.SL.polymars <- function (object, newdata, family, ...) {
+   if (family$family == "gaussian") {
+     out <- predict.polymars(object = object$object,
+       x = as.matrix(newdata))
+   }
+   if (family$family == "binomial") {
+     out <- ppolyclass(cov = newdata, fit = object$fit)[, 2]
+   }
+   return(out)
+ }
```

6.2 Creating screening algorithm wrappers

```
> listWrappers(what = "screen")
```

All screening algorithm wrappers in SuperLearner:

```
[1] "All"
[1] "screen.SIS"           "screen.corP"
[3] "screen.corRank"       "screen.glmP"
[5] "screen.glmRank"       "screen.glmnet"
[7] "screen.randomForest" "screen.template"
```

```
> screen.template
```

```
function (Y.temp, X.temp, family, obsWeights, id, ...)
{
  if (family$family == "gaussian") {
  }
  if (family$family == "binomial") {
  }
  whichVariable <- rep(TRUE, ncol(X.temp))
  return(whichVariable)
}
<environment: namespace:SuperLearner>
```

Each screening algorithm needs to return a logical vector of length equal to the number of variables in `X.temp`. The value `TRUE` indicates the variable should be included in the prediction algorithms for that screening, while `FALSE` indicates the variable should not be included (screened out) of the prediction algorithm.

Argument	Description
<code>Y.temp</code>	the outcome variable
<code>X.temp</code>	the training data set (the observations used to fit the model)
<code>family</code>	a description of the error distribution
<code>id</code>	a cluster identification
<code>obsWeights</code>	observation weights

Table 6.3: The allowed arguments for a screening algorithm in **SuperLearner**

6.3 How to set up the **SuperLearner** library

Before running the **SuperLearner**, the library of prediction algorithms needs to be set up. A collection of predefined algorithms and screening algorithms are included in the package. The previous sections discussed how to create your own algorithm functions for the **SuperLearner**. With all the functions ready, there are two ways to specify the library

1. A character vector
2. A list of character vectors

If no screening algorithms are to be used, the character vector approach is easiest. The function names are specified by strings separated by commas. For example,

```
> library = c("SL.glm", "SL.glmnet", "SL.randomForest")
```

When there are screening algorithms, the users needs to match the prediction algorithms with the screening algorithms. The library will be a list, and each element a character vector. The first element in the vector is the prediction algorithm, followed by all the screening algorithms matched with that prediction algorithm. For example,

```
> library = list(c("SL.glm", "screen.glmP"), c("SL.glmnet"),
+   c("SL.randomForest", "All", "screen.glmP", "screen.glmRank"))
```

One special included screening algorithms is the “All” function. “All” is a function to fit the matching prediction algorithm on the entire dataset. When no screening algorithm is specified, the “All” function is attached to the prediction algorithm. In the example above, the “SL.glmnet” algorithm is a singleton, but the **SuperLearner** will interpret this as `c("SL.glmnet", "All")`. If at least one screening algorithm is provided, then the “All” is not attached to the prediction algorithm. In the example above, “SL.glm” will only be run on the “screen.glm” subset of the variables and not on the entire set of variables. If the users wants to run the prediction algorithm on the entire dataset and on a screened subset, then the user needs to add the special function

“All” to the corresponding vector. In the example above, “SL.randomForest” will be run on the entire dataset (“All”) and on the screened subsets from the screening algorithms “screen.glmP” and “screen.glmRank”.

6.4 Example: Boston Housing Data

The Boston Housing Data in the `mlbench` package contains information on housing prices from 506 census tracts around Boston from the 1970 census. The original data description can be found in Harrison and Rubinfeld (1979) a corrections in Gilley and Pace (1996). We will build a model to predict the median value of owner-occupied homes based on the covariates supplied (for example, per capita crime rate, average number of rooms per dwelling and average age of homes). First we load the data and examine the variables included:

```
> library(mlbench)
> data(BostonHousing2)
```

With the data loaded, we need to change the `chas` variable from a factor to a numeric and then remove the variables we will not be using in the analysis.

```
> BostonHousing2$chas <- as.numeric(BostonHousing2$chas=="1")
> DATA <- BostonHousing2[, c("cmdev", "crim", "zn", "indus", "chas",
+   "nox", "rm", "age", "dis", "rad", "tax", "ptratio",
+   "b", "lstat")]
```

6.5 SuperLearner

```
> library(SuperLearner)
```

Before running the Super Learner we first need to set up the library of prediction algorithms. As we saw above, the **SuperLearner** package includes many built-in prediction algorithms. But the list of algorithms may not be sufficient for a given problem. Expert contextual knowledge may recommend new algorithms to consider or different values for the tuning parameters in the existing algorithms.

For example, the `SL.gam` function has the tuning parameter `deg.gam = 2`. If we wanted to also consider the `gam` model with degree 3 we do not need to write a new wrapper function but can easily use the current function and change the argument. For example:

```
> SL.gam.3 <- function(..., deg.gam = 3){
+   SL.gam(..., deg.gam = deg.gam)
+ }
```

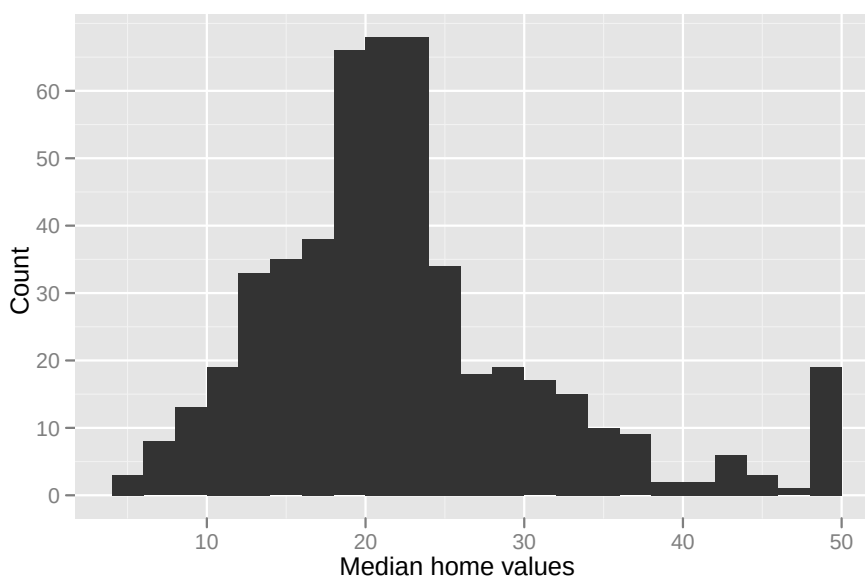


Figure 6.1: Histogram of median home values (in 1000 USD)

`SL.gam.3` is a simple function calling the `SL.gam` function but changing the argument `deg.gam` to the desired value for the tuning parameter. The “...” will simply pass all other arguments from `SL.gam.3` into `SL.gam` since these should be the same. This mechanism of writing functions of the wrapper functions allows for the easy adaptation of the currently available prediction algorithms to the current prediction problem and the desired range for tuning parameters.

```
> SL.library <- c("SL.gam",
+   "SL.gam.3",
+   "SL.gam.4",
+   "SL.gam.5",
+   "SL.gbm.1",
+   "SL.gbm.2",
+   "SL.glm",
+   "SL.glmnet",
+   "SL.glmnet.alpha25",
+   "SL.glmnet.alpha50",
+   "SL.glmnet.alpha75",
+   "SL.polymars",
+   "SL.randomForest",
+   "SL.ridge",
+   "SL.svm",
+   "SL.bayesglm",
```

```

+       "SL.step",
+       "SL.step.interaction",
+       "SL.bart")

```

With the library setup, we can now run the super learner. We need to specify the data frame of covariates X and the outcome Y . We can set the number of folds for the cross-validation step to estimate the weights for each algorithm. The default value is 20, but we will use 10 here for computational reason. The verbose statement specifies if you want a detailed report of the progress while the super learner runs. Set verbose to TRUE for a detailed output. Specify the library of prediction algorithms with the `SL.library` argument. The `shuffle` argument while randomize the rows in the data frame before splitting the data into V folds.

```

> fitSL <- SuperLearner(Y=log(DATA$cmedv),
+       X=subset(DATA, select = -c(cmedv)),
+       V=10,
+       verbose=FALSE,
+       SL.library=SL.library,
+       shuffle = TRUE,
+       family = gaussian()
+ )

```

```

> fitSL

```

Call:

```

SuperLearner(Y = log(DATA$cmedv), X = subset(DATA, select = -c(cmedv)),
  SL.library = SL.library, V = 10, shuffle = TRUE, verbose = FALSE,
  family = gaussian())

```

	Risk	Coef
SL.gam_All	0.03513343	0.00000000
SL.gam.3_All	0.03364400	0.00000000
SL.gam.4_All	0.03262047	0.00000000
SL.gam.5_All	0.03181313	0.00000000
SL.gbm.1_All	0.02918967	0.00000000
SL.gbm.2_All	0.02445375	0.00000000
SL.glm_All	0.03662357	0.00000000
SL.glmnet_All	0.03669622	0.00000000
SL.glmnet.alpha25_All	0.03676771	0.00000000
SL.glmnet.alpha50_All	0.03688589	0.00000000
SL.glmnet.alpha75_All	0.03661601	0.00000000
SL.polymars_All	0.03393655	0.00000000

SL.randomForest_All	0.02152117	0.23500781
SL.ridge_All	0.03663368	0.00000000
SL.svm_All	0.02660668	0.01244019
SL.bayesglm_All	0.03662259	0.00000000
SL.step_All	0.03649106	0.00000000
SL.step.interaction_All	0.02373183	0.28336464
SL.bart_All	0.01927439	0.46918736

6.6 Example: ALL Data

The super learner can also be used with a binary outcome.

```
> # install Bioconductor default packages and ALL package
> # source("http://bioconductor.org/biocLite.R")
> # biocLite()
> # biocLite("ALL")
> library(ALL)
> library(genefilter)
> data(ALL)
```

With the data from the **ALL** package, some filtering is still required before running the analysis. The steps below follow The chapter on supervised learning in [40].

```
> # restrict to only the NEG and BCR/ABL outcomes
> bcell <- grep("^B", as.character(ALL$BT))
> moltyp <- which(as.character(ALL$mol.biol) %in%
+   c("NEG", "BCR/ABL"))
> ALL_bcrneg <- ALL[, intersect(bcell, moltyp)]
> ALL_bcrneg$mol.biol <- factor(ALL_bcrneg$mol.biol)
> #
> # filter features
> ALLfilt_bcrneg <- nsFilter(ALL_bcrneg, var.cutoff = 0.75)$eset
> #
> # standardize the features
> rowIQRs <- function(eSet) {
+   numSamp <- ncol(eSet)
+   lowQ <- rowQ(eSet, floor(0.25 * numSamp))
+   upQ <- rowQ(eSet, ceiling(0.75 * numSamp))
+   upQ - lowQ
+ }
> standardize <- function(x) {
+   (x - rowMedians(x)) / rowIQRs(x)
```

```

+ }
> exprs(ALLfilt_bcrneg) <- standardize(exprs(ALLfilt_bcrneg))
> #
> # convert to numeric matrix for the SuperLearner
> Y <- as.numeric(ALLfilt_bcrneg$mol.biol == "BCR/ABL")
> X <- t(exprs(ALLfilt_bcrneg))

```

With the data ready, the next step is to put together the library. We consider the k-nearest neighbors, elastic net, and random forest algorithms with various levels of the tuning parameters. `knn` has a few built-in functions for various levels of k and the elastic net function, `glmnet`, and includes a few values for α . For `randomForest`, we consider a few values for `mtry` and `nodesize`. Since we are adjusting two tuning parameters, a grid is created with all possible pairwise combinations of the values for `mtry` and `nodesize`. The code below is similar to the code used above to create the new gam models with high degrees of freedom.

```

> tuneGrid <- expand.grid(mtry = c(500, 1000, 2200),
+   nodesize = c(1, 5, 10))
> for(mm in seq(nrow(tuneGrid))) {
+   eval(parse(file = "", text = paste("SL.randomForest.", mm,
+   "<- function(..., mtry = ", tuneGrid[mm, 1], ",
+   nodesize = ",
+   tuneGrid[mm, 2], ") { SL.randomForest(..., mtry = mtry,
+   nodesize = nodesize) }", sep = "")))
+ }
> SL.library <- c("SL.knn",
+   "SL.knn20",
+   "SL.knn30",
+   "SL.knn40",
+   "SL.knn50",
+   "SL.randomForest",
+   "SL.glmnet",
+   "SL.glmnet.alpha25",
+   "SL.glmnet.alpha50",
+   "SL.glmnet.alpha75",
+   "SL.mean",
+   paste("SL.randomForest.",
+   seq(nrow(tuneGrid)), sep = ""))

```

Since the outcome is binary, the family argument must be set to `binomial()` in the SuperLearner.

```

> fitSL <- SuperLearner(Y = Y, X = X, V = 10,
+   SL.library = SL.library,

```

```
+          family = binomial(), verbose = FALSE,
+          method = "NNLS", stratifyCV = TRUE)
> fitSL
```

Call:

```
SuperLearner(Y = Y, X = X, SL.library = SL.library, V = 10,
  verbose = FALSE, family = binomial(), method = "NNLS",
  stratifyCV = TRUE)
```

	Risk	Coef
SL.knn_All	0.20430380	0.00000000
SL.knn20_All	0.22313291	0.00000000
SL.knn30_All	0.22277887	0.00000000
SL.knn40_All	0.23432812	0.00000000
SL.knn50_All	0.23915886	0.00000000
SL.randomForest_All	0.12618567	0.00000000
SL.glmnet_All	0.08414461	0.82257005
SL.glmnet.alpha25_All	0.09834590	0.00000000
SL.glmnet.alpha50_All	0.09359018	0.00000000
SL.glmnet.alpha75_All	0.08521177	0.13392874
SL.mean_All	0.24924069	0.00000000
SL.randomForest.1_All	0.12792559	0.00000000
SL.randomForest.2_All	0.12235342	0.00000000
SL.randomForest.3_All	0.11827363	0.00000000
SL.randomForest.4_All	0.12984818	0.00000000
SL.randomForest.5_All	0.12063995	0.00000000
SL.randomForest.6_All	0.11875061	0.00000000
SL.randomForest.7_All	0.12984432	0.00000000
SL.randomForest.8_All	0.12098649	0.00000000
SL.randomForest.9_All	0.11613709	0.04350121

Honest V-fold cross-validated risk estimates can be obtained using the `CV.SuperLearner`. The results can be found in table 6.4.

6.7 Computing Environment

- R version 2.11.0 Patched (2010-04-27 r51837), x86_64-apple-darwin9.8.0
- Base packages: base, datasets, grDevices, graphics, grid, methods, splines, stats, utils

Table 6.4: 20-fold cross-validated risk estimates for the super learner, the discrete super learner and each algorithm in the library

Algorithm	subset	Risk	SE	Min	Max
SuperLearner	–	0.194	0.017	0.103	0.309
Discrete SL	–	0.238	0.024	0.127	0.415
SL.knn(10)	All	0.249	0.020	0.144	0.532
SL.knn(10)	clinical	0.239	0.019	0.138	0.496
SL.knn(10)	cor ($p < 0.1$)	0.262	0.023	0.095	0.443
SL.knn(10)	cor ($p < 0.01$)	0.224	0.020	0.088	0.365
SL.knn(10)	glmnet	0.219	0.028	0.007	0.465
SL.knn(20)	All	0.242	0.013	0.171	0.397
SL.knn(20)	clinical	0.236	0.012	0.154	0.382
SL.knn(20)	cor ($p < 0.1$)	0.233	0.017	0.108	0.342
SL.knn(20)	cor ($p < 0.01$)	0.206	0.018	0.121	0.321
SL.knn(20)	glmnet	0.217	0.026	0.018	0.405
SL.knn(30)	All	0.239	0.013	0.171	0.396
SL.knn(30)	clinical	0.236	0.012	0.169	0.386
SL.knn(30)	cor ($p < 0.1$)	0.232	0.014	0.143	0.319
SL.knn(30)	cor ($p < 0.01$)	0.215	0.017	0.136	0.346
SL.knn(30)	glmnet	0.210	0.023	0.039	0.402
SL.knn(40)	All	0.240	0.011	0.182	0.331
SL.knn(40)	clinical	0.238	0.010	0.179	0.319
SL.knn(40)	cor ($p < 0.1$)	0.236	0.012	0.166	0.316
SL.knn(40)	cor ($p < 0.01$)	0.219	0.015	0.154	0.309
SL.knn(40)	glmnet	0.211	0.021	0.060	0.346
SL.glmnet($\alpha = 1.0$)	corRank.50	0.229	0.029	0.078	0.445
SL.glmnet($\alpha = 1.0$)	corRank.20	0.208	0.026	0.048	0.424
SL.glmnet($\alpha = 0.75$)	corRank.50	0.221	0.027	0.077	0.420
SL.glmnet($\alpha = 0.75$)	corRank.20	0.209	0.026	0.046	0.421
SL.glmnet($\alpha = 0.50$)	corRank.50	0.226	0.027	0.077	0.426
SL.glmnet($\alpha = 0.50$)	corRank.20	0.211	0.026	0.059	0.419
SL.glmnet($\alpha = 0.25$)	corRank.50	0.229	0.027	0.084	0.424
SL.glmnet($\alpha = 0.25$)	corRank.20	0.216	0.025	0.072	0.406
SL.randomForest	clinical	0.198	0.019	0.098	0.391
SL.randomForest	cor ($p < 0.01$)	0.204	0.018	0.101	0.341
SL.randomForest	glmnet	0.220	0.025	0.072	0.378
SL.bagging	clinical	0.207	0.016	0.108	0.408
SL.bagging	cor ($p < 0.01$)	0.205	0.018	0.107	0.353
SL.bagging	glmnet	0.206	0.022	0.077	0.388
SL.bart	clinical	0.202	0.018	0.109	0.365
SL.bart	cor ($p < 0.01$)	0.210	0.021	0.092	0.376
SL.bart	glmnet	0.220	0.028	0.043	0.423
SL.mean	All	0.250	0.003	0.246	0.251

- Other packages: ALL 1.4.7, AnnotationDbi 1.10.0, BayesTree 0.3-1, Biobase 2.8.0, DBI 0.2-5, MASS 7.3-5, Matrix 0.999375-38, R2WinBUGS 2.1-16, RSQLite 0.8-4, SuperLearner 1.1-18, abind 1.1-0, akima 0.5-4, arm 1.3-02, car 1.2-16, class 7.3-2, coda 0.13-5, digest 0.4.2, e1071 1.5-24, foreign 0.8-40, gam 1.03, gbm 1.6-3, genefilter 1.30.0, ggplot2 0.8.7, glmnet 1.3, hgu95av2.db 2.4.1, lattice 0.18-5, lme4 0.999375-33, mlbench 2.0-0, nnet 7.3-1, nnls 1.3, org.Hs.eg.db 2.4.1, plyr 0.1.9, polyspline 1.1.4, proto 0.3-8, quadprog 1.5-3, randomForest 4.5-34, reshape 0.8.3, survival 2.35-8
- Loaded via a namespace (and not attached): annotate 1.26.0, nlme 3.1-96, tools 2.11.0, xtable 1.5-6

Chapter 7

Conclusion and Future Work

The super learner provides a very general framework for statistical learning. Given a loss function and a library of learners, the super learner will estimate the optimal combination based on minimizing the expected loss (i.e. risk). This dissertation provides a detailed description of the super learner framework and the theoretical justification of the optimality. In addition to the theory, the super learner is demonstrated on a wide variety of real world applications and demonstrated to perform well in each of these cases.

An area of future work is the coupling of the super learner with the targeted maximum likelihood estimation [91]. Super learner for prediction estimates the entire conditional expectation, but in many cases the importance of specific variables in the context of the final estimator is of interest. The importance of any particular variable in the super learner estimator could be computed directly from the super learner estimate, but this may not be the most efficient and robust estimate of the importance of that variable. The super learner for prediction is estimating the entire conditional expectation and appropriately balances bias and variance for that setting. But the bias and variance trade-off for a specific variable may be different. The targeted maximum likelihood works with the super learner estimator and by adding a clever covariate based on the specific variable will update the estimator to give an efficient and robust estimate of the variable importance. The targeted maximum likelihood step could be repeated for each variable the importance needs to be estimated. The marriage of super learner and targeted maximum likelihood provides a theoretically optimal framework for estimation and inference but more work should be done in understanding the interactions of the super learner and the targeted maximum likelihood frameworks.

Bibliography

- [1] A. Afifi and S. Azen. *Statistical Analysis: A Computer Oriented Approach*. Academic Press, New York, NY, 2nd edition, 1979.
- [2] H. Akaike. Information theory and an extension of the maximum likelihood principle. In B. N. Petrov and F. Csaki, editors, *Second International Symposium on Information Theory*, pages 267–281, Budapest, 1973. Akadémiai Kiadó.
- [3] S. Arlot and A. Celisse. A survey of cross-validation procedures for model selection. *Statistical Surveys*, 4:40–79, 2010.
- [4] S. Arlot and P. Massart. Data-driven calibration of penalties for least squares regression. *Journal of Machine Learning Research*, 2008.
- [5] O. Bembom, M. L. Petersen, S. Rhee, W. J. Fessel, S. E. Sinisi, R. W. Shafer, and M. J. van der Laan. Biomarker discovery using targeted maximum likelihood estimation: Application to the treatment of antiretroviral resistant HIV infection. *Statistics in Medicine*, 28:152–172, 2008.
- [6] E. R. Berndt. *The Practice of Econometrics*. Addison-Wesley, New York, NY, 1991.
- [7] L. Breiman. The little bootstrap and other methods for dimensionality selection in regression: X-fixed prediction error. *Journal of the American Statistical Association*, 87(419):738–754, 1992.
- [8] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
- [9] L. Breiman. Heuristics of instability and stabilization in model selection. *Annals of Statistics*, 24:2350–2383, 1996.
- [10] L. Breiman. Stacked regressions. *Machine Learning*, 24:49–64, 1996.
- [11] L. Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.
- [12] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. The Wadsworth Statistics/Probability series. Wadsworth International Group, 1984.

- [13] L. Breiman and P. Spector. Submodel selection and evaluation in regression: the x-random case. *International Statistical Review*, 60:291–319, 1992.
- [14] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- [15] H. Chipman and R. McCulloch. *BayesTree: Bayesian Methods for Tree Based Models*, 2009. R package version 0.3-1.
- [16] H. A. Chipman, E. I. George, and R. E. McCulloch. BART: Bayesian additive regression trees. *Annals of Applied Statistics*, accepted, 2010.
- [17] S. Chu. Pricing the C’s of diamond stones. *Journal of Statistical Education*, 9(2), 2001.
- [18] W. S. Cleveland, E. Groose, and W. M. Shyu. Local regression models. In J. M. Chambers and T. Hastie, editors, *Statistical Models in S*, chapter 8. Wadsworth & Brooks/Cole, 1992.
- [19] D. Cook. *Regression Graphics: Ideas for Studying Regression Through Graphics*. Wiley, New York, NY, 1998.
- [20] D. Cook and S. Weisberg. *An Introduction to Regression Graphics*. Wiley, New York, NY, 1994.
- [21] L. Devroye and T. J. Wagner. Distribution-free performance bounds for potential function rules. *IEEE Transactions on Information Theory*, 25(5):601–604, 1979.
- [22] E. Dimitriadou, K. Hornik, F. Leisch, D. Meyer, and A. Weingessel. *e1071: Misc Functions of the Department of Statistics (e1071), TU Wien*, 2009. R package version 1.5-22.
- [23] S. Dudoit and M. J. van der Laan. Asymptotics of cross-validated risk estimation in estimator selection and performance assessment. *Statistical Methodology*, 2:131–154, 2005.
- [24] B. Efron. Estimating the error rate of a prediction rule: Improvement on cross-validation. *Journal of the American Statistical Association*, 78(382):316–331, 1983.
- [25] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *Annals of Statistics*, 32(2):407–499, 2004.
- [26] B. Efron and R. Tibshirani. Improvements on cross-validation: The .632+ bootstrap method. *Journal of the American Statistical Association*, 92(438):548–560, 1997.

- [27] B. Fisher, J. Dignam, N. Wolmark, E. Mamounas, J. Costantino, W. Poller, E. R. Fisher, D. L. Wickerham, M. Deutsch, R. Margolese, N. Dimitrov, and M. Kavanah. Lumpectomy and radiation therapy for the treatment of intraductal breast cancer: Findings from national surgical adjuvant breast and bowel project B-17. *Journal of Clinical Oncology*, 16(2):441–452, 1998.
- [28] B. Fisher, J. Dignam, N. Wolmark, D. L. Wickerham, E. R. Fisher, E. Mamounas, R. Smith, M. Begovic, N. V. Dimitrov, R. G. Margolese, C. G. Kardinal, M. T. Kavanah, L. Fehrenbacher, and R. H. Oishi. Tamoxifen in treatment of intraductal breast cancer: National surgical adjuvant breast and bowel project B-24 randomised controlled trial. *Lancet*, 353:1993–2000, 1999.
- [29] E. R. Fisher, J. Dignam, E. Tan-Chiu, J. Costantino, B. Fisher, S. Paik, N. Wolmark, and NSABP. Pathologic findings from the national surgical adjuvant breast project (NSABP) eight-year update of protocol B-17. *Cancer*, 86(3):429–438, 1999.
- [30] E. R. Fisher, S. R. Land, R. S. Saad, B. Fisher, D. L. Wickerham, M. Wang, J. P. Costantino, and N. Wolmark. Pathologic variables predictive of breast events in patients with ductal carcinoma in situ. *American journal of clinical pathology*, 128(1):86–91, 2007.
- [31] Y. Freund, R. E. Schapire, Y. Singer, and M. K. Warmuth. Using and combining predictors that specialize. In *Twenty-Ninth Annual ACM Symposium on the Theory of Computing*, pages 334–343, 1997.
- [32] J. Friedman, T. Hastie, and R. Tibshirani. *glmnet: Lasso and elastic-net regularized generalized linear models*, 2010. R package version 1.1-5.
- [33] J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1), 2010.
- [34] J. H. Friedman. Multivariate adaptive regression splines. *Annals of Statistics*, 19(1):1–141, 1991.
- [35] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2001.
- [36] S. Geisser. The predictive sample reuse method with applications. *Journal of the American Statistical Association*, 70(350):320–328, 1975.
- [37] A. Gelman, A. Jakulin, M. G. Pittau, and Y.-S. Su. A weakly informative default prior distribution for logistic and other regression models. *Annals of Applied Statistics*, 2(3):1360–1383, 2009.

- [38] A. Gelman, Y.-S. Su, M. Yajima, J. Hill, M. G. Pittau, J. Kerman, and T. Zheng. *arm: Data Analysis Using Regression and Multilevel/Hierarchical Models*, 2010. R package version 1.3-02.
- [39] R. D. Gill, M. J. van der Laan, and J. M. Robins. Coarsening at random: Characterizations, conjectures, counter-examples. In D. Y. Lin and T. R. Fleming, editors, *Proceedings of the First Seattle Symposium in Biostatistics: Survival Analysis*, Lecture Notes in Statistics, pages 255–294, New York, 1997. Springer-Verlag.
- [40] F. Hahne, W. Huber, R. Gentleman, and S. Falcon, editors. *Bioconductor Case Studies*. Springer, New York, 2008.
- [41] J. V. Hansen. Combining predictors: Some old methods and a new method. In *ICCIN*, 1998.
- [42] F. E. Harrell, Jr. *Regression Modeling Strategies: With Applications to Linear Models, Logistic Regression, and Survival Analysis*. Springer-Verlag, New York, NY, 2001.
- [43] T. Hastie. Generalized additive models. In J. M. Chambers and T. Hastie, editors, *Statistical Models in S*, chapter 7. Wadsworth & Brooks/Cole, 1992.
- [44] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd edition, 2008.
- [45] T. J. Hastie and R. J. Tibshirani. *Generalized Additive Models*. Chapman and Hall, 1990.
- [46] A.E. Hoerl and R.W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(3):55–67, 1970.
- [47] T. Hothorn, P. Buhlmann, S. Dudoit, A. M. Molinaro, and M. J. van der Laan. Survival ensembles. *Biostatistics*, 7:355–373, 2006.
- [48] H. Ishwaran, U. B. Kogalur, E. H. Blackstone, and M. S. Lauer. Random survival forests. *Annals of Applied Statistics*, 2008.
- [49] S. Keleş, M. J. van der Laan, and S. Dudoit. Asymptotically optimal model selection method with right censored outcomes. *Bernoulli*, 10(6):1011–1037, 2004.
- [50] K. Kerlikowske, A. Molinaro, I. Cha, B.-M. Ljung, V. L. Ernster, K. Stewart, K. Chew, D. H. Moore, and F. Waldman. Characteristics associated with recurrence among women with ductal carcinoma in situ treated by lumpectomy. *Journal of the National Cancer Institute*, 95(22):1692–1702, 2003.

- [51] D. Kibler, D. aha, and M. K. Albert. Instance-based prediction of real-valued attributes. *Computational Intelligence*, 5:51–57, 1989.
- [52] C. Kooperberg. *polspline: Polynomial spline routines*, 2009. R package version 1.1.4.
- [53] C. Kooperberg, S. Bose, and C. J. Stone. Polychotomous regression. *Journal of the American Statistical Association*, 92:117–127, 1997.
- [54] C. Kooperberg, C. J. Stone, and Y. K. Truong. Hazard regression. *Journal of the American Statistical Association*, 90:78–94, 1995.
- [55] C. L. Lawson and R. J. Hanson. *Solving Least Squares Problems*. SIAM, 1987.
- [56] M. LeBlanc and J. Crowley. Relative risk trees for censored survival data. *Biometrics*, 48:411–425, 1992.
- [57] M. LeBlanc and R. Tibshirani. Combining estimates in regression and classification. *Journal of the American Statistical Association*, 91:1641–1650, 1996.
- [58] A. Liaw and M. Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002.
- [59] Z. Liu and T. Stengos. Non-linearities in cross country growth regressions: A semiparametric approach. *Journal of Applied Econometrics*, 14:527–538, 1999.
- [60] C. L. Loprinzi, J. A. Laurie, H. S. Wieand, J. E. Krook, P. J. Novotny, J. W. Kugler, J. Bartel, M. Law, M. Bateman, and N. E. Klatt. Prospective evaluation of prognostic variables from patient-completed questionnaires. north central cancer treatment group. *Journal of Clinical Oncology*, 12(3):601–607, 1994.
- [61] C. L. Mallows. Some comments on c_p . *Technometrics*, 15:661–675, 1973.
- [62] R. Neugebauer and J. Bullard. *DSA: Data-Adaptive Estimation with Cross-Validation and the D/S/A Algorithm*, 2009. R package version 3.1.3.
- [63] D. J. Newman, S. Hettich, C. L. Blake, and C. J. Merz. UCI repository of machine learning databases, 1998. <http://archive.ics.uci.edu/ml/>.
- [64] I. A. Olivotto, C. D. Bajdik, P. M. Ravdin, C. H. Speers, A. J. Coldman, B. D. Norris, G. J. Davis, S. K. Chia, and K. A. Gelmon. Population-based validation of the prognostic model adjuvant! for early breast cancer. *Journal of clinical oncology*, 23(12):2716–2725, 2005.
- [65] M. Y. Park and T. Hastie. l_1 -regularization path algorithm for generalized linear models. *Journal of the Royal Statistical Society, Series B*, 69(4):659–677, 2007.

- [66] K. Penrose, A. Nelson, and A. Fisher. Generalized body composition prediction equation for men using simple measurement techniques. *Medicine and Science in Sports and Exercise*, 17:189, 1985.
- [67] A. Peters and T. Hothorn. *ipred: Improved Predictors*, 2009. R package version 0.8-8.
- [68] R. R. Picard and R. D. Cook. Cross-validation of regression models. *Journal of the American Statistical Association*, 79(387):575–583, 1984.
- [69] E. C. Polley and M. J. van der Laan. Adaptive selection of the functional form for the super learner. in preparation, 2010.
- [70] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2010.
- [71] S. Rhee, J. Taylor, G. Wadhera, J. Ravela, A. Ben-Hur, D. Brutlag, and R. W. Shafer. Genotypic predictors of human immunodeficiency virus type 1 drug resistance. *Proceedings of the National Academy of Sciences USA*, 103(46):17355–17360, 2006.
- [72] G. Ridgeway. *gbm: Generalized Boosted Regression Models*, 2007. R package version 1.6-3.
- [73] B. D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996.
- [74] B. Rosner. *Fundamentals of Biostatistics*. Duxbury, Pacific Grove, CA, 5th edition, 1999.
- [75] D. Rubin and M. J. van der Laan. A doubly robust censoring unbiased transformation. *The International Journal of Biostatistics*, 3(1):Article 4, 2007.
- [76] I. Ruczinski, C. Kooperberg, and M. LeBlanc. Logic regression. *Journal of Computational and Graphical Statistics*, 12(3):475–511, 2003.
- [77] M. R. Segal. Regression trees for censored data. *Biometrics*, 44:35–47, 1988.
- [78] J. Shao. Linear model selection by cross-validation. *Journal of the American Statistical Association*, 88(422):486–494, 1993.
- [79] D. Singh, P. G. Febbo, K. Ross, D. G. Jackson, J. Manola, C. Ladd, P. Tamayo, A. A. Renshaw, A. V. D’Amico, J. P. Richie, E. S. Lander, M. Loda, P. W. Kantoff, T. R. Golub, and W. R. Sellers. Gene expression correlates of clinical prostate cancer behavior. *Cancer Cell*, 1(2):203–209, 2002.

- [80] S. E. Sinisi, E. C. Polley, M. L. Petersen, S.Y. Rhee, and M. J. van der Laan. Super learning: An application to the prediction of HIV-1 drug resistance. *Statistical Applications in Genetics and Molecular Biology*, 6(1):Article 7, 2007.
- [81] S. E. Sinisi and M. J. van der Laan. Deletion/Substitution/Addition algorithm in learning with applications in genomics. *Statistical Applications in Genetics and Molecular Biology*, 3(1):Article 18, 2004.
- [82] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society, Series B*, 36(2):111–147, 1974.
- [83] M. Stone. Asymptotics for and against cross-validation. *Biometrika*, 64(1):29–35, 1977.
- [84] T. Therneau and T. Lumley. *survival: Survival Analysis, Including Penalised Likelihood*, 2009. R package version 2.35-8.
- [85] R. Tibshirani. The lasso method for variable selection in the cox model. *Statistics in Medicine*, 16:385–395, 1997.
- [86] M. J. van der Laan. Statistical inference for variable importance. *The International Journal of Biostatistics*, 2:Article 2, 2006.
- [87] M. J. van der Laan and S. Dudoit. Unified cross-validation methodology for selection among estimators and a general cross-validated adaptive epsilon-net estimator: Finite sample oracle inequalities and examples. Technical Report 130, Division of Biostatistics, University of California, Berkeley, 2003.
- [88] M. J. van der Laan, S. Dudoit, and A. W. van der Vaart. The cross-validated adaptive epsilon-net estimator. *Statistics and Decisions*, 24(3):373–395, 2006.
- [89] M. J. van der Laan, E. C. Polley, and A. E. Hubbard. Super learner. *Statistical Applications in Genetics and Molecular Biology*, 6(25):Article 25, 2007.
- [90] M. J. van der Laan and J. Robins. *Unified Methods for Censored Longitudinal Data and Causality*. Springer Series in Statistics. Springer, 2003.
- [91] M. J. van der Laan and D. Rubin. Targeted maximum likelihood learning. *International Journal of Biostatistics*, 2(1):Article 11, 2007.
- [92] A. W. van der Vaart, S. Dudoit, and M. J. van der Laan. Oracle inequalities for multi-fold cross validation. *Statistics and Decisions*, 24(3):351–371, 2006.
- [93] H. C. van Houwelingen, T. Bruinsma, A. A. M. Hart, L. J. van’t Veer, and L. F. A. Wessels. Cross-validated cox regression on microarray gene expression data. *Statistics in Medicine*, 25:3201–3216, 2006.

- [94] L. J. van't Veer, H. Dal, M. J. van de Vijver, Y. D. He, A. A. M. Hart, M. Mao, H. L. Peterse, K. van der Kooy, M. J. Marton, A. T. Witteveen, G. J. Schreiber, R. M. Kerkhoven, C. Roberts, P. S. Linsley, R. Bernards, and S. H. Friend. Gene expression profiling predicts clinical outcome of breast cancer. *Nature*, 415:530–536, 2002.
- [95] W. N. Vernables and B. D. Ripley. *Modern Applied Statistics with S*. Springer, New York, 4th edition, 2002.
- [96] D. H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.
- [97] H. H. Zhang and W. Lu. Adaptive lasso for cox's proportional hazards model. *Biometrika*, 94(3):691–703, 2007.

Appendix A

R functions

These functions are available in the SuperLearner R package available on the author's website. The functions are described in chapter 6

A.1 Estimation functions

```
SuperLearner <- function(Y, X, newX = NULL, SL.library, V
  = 20, shuffle=TRUE, verbose=FALSE, family=gaussian(),
  method="NNLS", id = NULL, save.fit.library = TRUE, trim
  .logit = 0.001, obsWeights = NULL, stratifyCV = FALSE)
{
  require(nnls)
  require(quadprog)
  library <- .createLibrary(SL.library)
  .check.SL.library(SL.library = c(library$predAlgorithm$
    library, library$screenAlgorithm))
  if(!(method %in% c("NNLS", "NNLS2", "NNloglik"))) {
    stop("Method not currently supported. Should be NNLS,
      NNLS2, or NNloglik")
  }
  call <- match.call()
  X <- as.data.frame(X)
  var.names <- names(X)
  N.all <- length(Y)
  p <- dim(X)[2L]
  k.cand <- nrow(library$predAlgorithm)
  k.screen <- length(library$screenAlgorithm)
  newZ <- matrix(NA, N.all, k.cand)
```

```

cand.names <- rep(NA, k.cand)
for(jj in seq(k.cand)) {
  cand.names[jj] <- paste(library$predAlgorithm[jj, 1],
    library$screenAlgorithm[library$predAlgorithm[jj, 2]],
    sep="_")
}
errorsInCVLibrary <- rep(0, k.cand)
errorsInLibrary <- rep(0, k.cand)

if(is.null(newX)) {
  newX <- X
}

fit.library <- vector("list", k.cand)
names(fit.library) <- cand.names

if(is.character(family))
  family <- get(family, mode="function", envir=parent.
    frame())
if(is.function(family))
  family <- family()
if (is.null(family$family)) {
  print(family)
  stop("'family' not recognized")
}
DATA.split <- CVFolds(V=V, N.all = N.all, shuffle =
  shuffle,
  id = id, stratifyCV = stratifyCV, Y = Y)
if(is.null(id)) {
  id <- seq(N.all)
}
if(!identical(length(id), N.all)) {
  stop("id vector must have the same dimension as Y")
}
if(is.null(obsWeights)) {
  obsWeights <- rep(1, N.all)
}
if(!identical(length(obsWeights), N.all)) {
  stop("obsWeights vector must have the same dimension as
    Y")
}
X <- as.data.frame(X)

```

```

newX <- as.data.frame(newX)
namesX <- names(X)
if(!identical(colnames(X), colnames(newX))) {
  stop("The variable names and order in newX must be
       identical
       to the variable names and order in X")
}
# test for character or factor variables
if (sum(!sapply(X, is.numeric)) > 0) {
  stop("Currently, only numeric variables are allowed.
       Please
       convert any character or factor variables to numeric.")
}
# test for missing values
if (sum(is.na(X)) > 0) {
  stop("missing data is currently not supported")
}
if (!is.numeric(Y)) {
  stop("the outcome Y must be a numeric vector")
}

## now for the candidates for the super learner
vv <- 1
for(bb in DATA.split){
  tempLearn <- X[-(bb), , drop=FALSE]
  tempOutcome <- Y[-(bb)]
  tempValid <- X[bb, , drop=FALSE]
  tempWhichScreen <- matrix(NA, nrow = k.screen, ncol = p)
  tempid <- id[-(bb)]
  tempobsWeights <- obsWeights[-(bb)]

  for(s in seq(k.screen)) {
    testScreen <- try(do.call(library$screenAlgorithm[s],
      list(Y.temp = tempOutcome, X.temp = tempLearn, family
        = family, id = tempid, obsWeights = tempobsWeights))
    )
    if(inherits(testScreen, "try-error")) {
      warning(paste("replacing failed screening,",
        library$screenAlgorithm[s], ", algorithm with All() in
        fold", vv ,"\n "))
    }
    tempWhichScreen[s, ] <- TRUE
  }
  next
}

```

```

} else {
  tempWhichScreen[s, ] <- testScreen
}
if(verbose) {
  print(paste("Number of covariates in ", library$
    screenAlgorithm[s], " is: ", sum(tempWhichScreen[s,
    ]), sep = ""))
}
}

for(k in 1:k.cand){
  testAlg <- try(do.call(library$predAlgorithm[k, 1],
    list(Y.temp = tempOutcome, X.temp = subset(tempLearn,
    select = tempWhichScreen[library$predAlgorithm[k, 2], ],
    drop=FALSE), newX.temp = subset(tempValid, select =
    tempWhichScreen[library$predAlgorithm[k, 2], ], drop=
    FALSE),
    family = family, id = tempid, obsWeights =
    tempobsWeights)))
  if(inherits(testAlg, "try-error")) {
    warning(paste("Error in algorithm",
    library$predAlgorithm[k, 1], " on fold", vv, "\n  The
    Algorithm will be removed from the Super Learner (i.e.
    given
    weight 0) \n" ))
    errorsInCVLibrary[k] <- 1
    next
  } else {
    newZ[bb, k] <- testAlg$out
  }
  if(verbose) {print(paste("CV", cand.names[k]))}
  }#end library
  if(verbose) {print(paste("V-fold:", vv))}
  vv <- vv + 1
  }#end V-fold

  if(sum(errorsInCVLibrary) > 0) {
    newZ[, as.logical(errorsInCVLibrary)] <- 0
  }
  if(all(newZ==0)) {
    stop("All algorithms dropped from library")
  }
}

```

```

getweights <- .computeCoef(newZ=newZ, Y=Y,
  cand.names=cand.names, method=method, trim=trim.logit,
  verbose=verbose, obsWeights = obsWeights)
init.coef <- getweights$init.coef
update.coef <- getweights$update.coef
names(update.coef) <- cand.names
names(init.coef) <- cand.names

## now fit all candidate on entire training set and
predict
##on newX
m <- dim(newX)[1L]
predY <- matrix(NA, m, k.cand)
whichScreen <- matrix(NA, nrow = k.screen, ncol = p)

for(s in seq(k.screen)){
  testScreen <- try(do.call(library$screenAlgorithm[s],
    list(Y.temp = Y, X.temp = X, family = family, id = id,
    obsWeights = obsWeights)))
  if(inherits(testScreen, "try-error")) {
    warning(paste("replacing failed screening,",
    library$screenAlgorithm[s], ", algorithm with All() in
    full
    data", "\n "))
    whichScreen[s, ] <- TRUE
    next
  } else {
    whichScreen[s, ] <- testScreen
  }
  if(verbose) {
    print(paste("Number of covariates in ",
    library$screenAlgorithm[s], " is: ", sum(whichScreen[s,
    ]),
    sep = ""))
  }
}

for(k in 1:k.cand){
  testAlg <- try(do.call(library$predAlgorithm[k, 1], list
    (Y.temp=Y, X.temp = subset(X, select=whichScreen[
    library$predAlgorithm[k, 2], ], drop=FALSE), newX.
    temp=subset(newX, select=whichScreen[library$

```

```

        predAlgorithm[k, 2], ], drop=FALSE), family=family,
        id = id, obsWeights = obsWeights)))
if(inherits(testAlg, "try-error")) {
  warning(paste("Error in algorithm", library$
    predAlgorithm[k, 1], " on full data", "\n The
    Algorithm will be removed from the Super Learner (i.e
    . given weight 0) \n" ))
  errorsInLibrary[k] <- 1
  next
} else {
  predY[, k] <- testAlg$out
}
if(save.fit.library) {
  fit.library[[k]] <- testAlg$fit
}
if(verbose) {
  print(paste("full", cand.names[k]))
}
}#end library

if(sum(errorsInLibrary) > 0) {
  if(sum(update.coef[as.logical(errorsInLibrary)]) > 0) {
    warning(paste("re-running estimation of coefficients
      removing
      failed algorithm(s) \n Orignial coefficients are: \n",
      update.coef, "\n"))
    newZ[, as.logical(errorsInLibrary)] <- 0
    if(all(newZ==0)) {
      stop("All algorithms dropped from library")
    }
    getweights <- .computeCoef(newZ=newZ, Y=Y,
      cand.names=cand.names, method=method, trim=trim.logit,
      verbose=verbose, obsWeights = obsWeights)
    init.coef <- getweights$init.coef
    update.coef <- getweights$update.coef
    names(update.coef) <- cand.names
    names(init.coef) <- cand.names
  } else {
    warning("coefficients already 0 for all failed algorithm
      (s)")
  }
}
}

```

```
getPred <- .computePred(predY=predY, init.coef=init.coef,
  update.coef=update.coef, method=method, trim=trim.logit
)
```

```
# add names of algorithms to the predictions
colnames(predY) <- cand.names
```

```
# clean up when errors in library
if(sum(errorsInCVLibrary) > 0) {
  getweights$cv.risk[, as.logical(errorsInCVLibrary)] <-
    NA
}
final <- list(call=call, cand.names=cand.names, SL.library
  =library, SL.predict=getPred$SL.pred.update, init.coef=
  init.coef, coef=update.coef, library.predict=predY,
  newZ=newZ, cv.risk=getweights$cv.risk, family=family,
  fit.library=fit.library, id=id, namesX=namesX, DATA.
  split=DATA.split, method=method, whichScreen=
  whichScreen, trim.logit=trim.logit, errorsInCVLibrary =
  errorsInCVLibrary, errorsInLibrary = errorsInLibrary,
  obsWeights = obsWeights)
class(final) <- c("SuperLearner")
return(final)
}
```

```
CV.SuperLearner <- function(Y, X, SL.library, outside.V
  =20, inside.V=20, shuffle = TRUE, verbose = FALSE,
  family=gaussian(), method="NNLS", id=NULL, save.fit.
  library=FALSE, trim.logit=0.001, obsWeights = NULL,
  stratifyCV = FALSE, ...) {
  # create V-folds
  call <- match.call()
  n <- length(Y)
  folds <- CVFolds(V=outside.V, N.all = n, shuffle =
    shuffle, id = id, stratifyCV = stratifyCV, Y =
    Y)
  # if(shuffle) {
  #   folds <- split(sample(1:n), rep(1:outside.
  #     V, length=n))
  # } else {
  #   folds <- split(1:n, rep(1:outside.V,
  #     length=n))
}
```

```

# }
# if(!is.null(id)){
#     n.id <- length(unique(id))
#     id.split <- split(sample(1:n.id), rep(1:
#         outside.V, length=n.id))
#     folds <- vector("list", outside.V)
#     for(v in seq(outside.V)) {
#         folds[[v]] <- which(id %in% unique
#             (id)[id.split[[v]]])
#     }
# }
if(is.null(obsWeights)) {
    obsWeights <- rep(1, n)
}
if(!identical(length(obsWeights), n)) {
    stop("obsWeights vector must have the same
        dimension as Y")
}
# create placeholders for output
library <- .createLibrary(SL.library)
.check.SL.library(SL.library = c(library$
    predAlgorithm$library, library$screenAlgorithm)
)
k.cand <- nrow(library$predAlgorithm)
k.screen <- length(library$screenAlgorithm)
cand.names <- rep(NA, k.cand)
for(jj in seq(k.cand)) {
    cand.names[jj] <- paste(library$
        predAlgorithm[jj, 1], library$
        screenAlgorithm[library$predAlgorithm[
            jj, 2]], sep="_")
}

All.SL <- vector("list", outside.V)
names(All.SL) <- paste("training", 1:outside.V,
    sep="")
pred.SL <- rep(NA, n)
pred.discreteSL <- rep(NA, n)
whichDiscreteSL <- rep(NA, outside.V)
pred.library <- matrix(NA, nrow=n, ncol=k.cand)
colnames(pred.library) <- cand.names
coef.SL <- matrix(NA, nrow=outside.V, ncol=k.cand)

```

```

colnames(coef.SL) <- cand.names

for(v in seq(outside.V)){
  valid <- folds[[v]]
  CV.Y <- Y[-valid]
  CV.X <- X[-valid, ]
  CV.newX <- X[valid, ]
  CV.id <- id[-valid]
  CV.obsWeights <- obsWeights[-valid]

  fit.SL <- SuperLearner(Y=CV.Y, X=CV.X,
    newX=CV.newX, SL.library=SL.library, V=
    inside.V, shuffle=shuffle, verbose=
    verbose, family=family, method=method,
    id=CV.id, save.fit.library=save.fit.
    library, trim.logit=trim.logit,
    obsWeights = CV.obsWeights, stratifyCV
    = stratifyCV)
  if(verbose) {
    print(paste("completed CV ", v))
  }
  All.SL[[v]] <- fit.SL
  pred.SL[valid] <- fit.SL$SL.predict
  pred.discreteSL[valid] <- fit.SL$library.
    predict[, which.min(fit.SL$cv.risk)]
  whichDiscreteSL[v] <- names(which.min(fit.
    SL$cv.risk))
  pred.library[valid, ] <- fit.SL$library.
    predict
  coef.SL[v, ] <- fit.SL$coef
}
out <- list(CV.fit.SL=All.SL, pred.SL=pred.SL,
  pred.discreteSL = pred.discreteSL,
  whichDiscreteSL = whichDiscreteSL, pred.library
  =pred.library, coef.SL=coef.SL, folds=folds,
  method = method, call=call, obsWeights =
  obsWeights, id = id, V = c(outside.V = outside.
  V, inside.V = inside.V), Y = Y, X = X)
class(out) <- c("CV.SuperLearner")
return(out)
}

```

```

.check.SL.library <- function(SL.library) {
  if("SL.bayesglm" %in% SL.library) tryCatch(require(
    arm), warning = function(...){ stop("you have
    selected bayesglm as a library algorithm but do
    not have the arm package installed")})
  if("SL.cforest" %in% SL.library) tryCatch(require(
    party), warning = function(...){ stop("you have
    selected cforest as a library algorithm but do
    not have the party package installed")})
  if("SL.DSA" %in% SL.library) tryCatch(require(DSA)
    , warning = function(...){ stop("you have
    selected DSA as a library algorithm but do not
    have the DSA package installed")})
  if("SL.gam" %in% SL.library) tryCatch(require(gam)
    , warning = function(...){ stop("you have
    selected gam as a library algorithm but do not
    have the gam package installed")})
  if("SL.gbm" %in% SL.library) tryCatch(require(gbm)
    , warning = function(...){ stop("you have
    selected gbm as a library algorithm but do not
    have the gbm package installed")})
  if("SL.glmnet" %in% SL.library) tryCatch(require(
    glmnet), warning = function(...){ stop("you
    have selected glmnet as a library algorithm but
    do not have the glmnet package installed")})
  if("SL.knn" %in% SL.library) tryCatch(require(
    class), warning = function(...){ stop("you have
    selected knn as a library algorithm but do not
    have the class package installed.  class is
    part of the VR bundle")})
  if("SL.logreg" %in% SL.library) tryCatch(require(
    LogicReg), warning = function(...){ stop("you
    have selected logreg as a library algorithm but
    do not have the LogicReg package installed")})
  if("SL.nnet" %in% SL.library) tryCatch(require(
    nnet), warning = function(...){ stop("you have
    selected nnet as a library algorithm but do not
    have the nnet package installed.  nnet is part
    of the VR bundle")})
  if("SL.polymars" %in% SL.library) tryCatch(require(
    polyspline), warning = function(...){ stop("you
    have selected polymars or polyclass as a

```

```

library algorithm but do not have the polspline
package installed"))})
if("SL.randomForest" %in% SL.library) tryCatch(
  require(randomForest), warning = function(...){
    stop("you have selected randomForest as a
library algorithm but do not have the
randomForest package installed"))})
if("SL.ridge" %in% SL.library) tryCatch(require(
  MASS), warning = function(...){ stop("you have
selected lm.ridge as a library algorithm but do
not have the MASS package installed. MASS is
part of the VR bundle"))})
if("SL.spls" %in% SL.library) tryCatch(require(
  spls), warning = function(...){ stop("you have
selected spls as a library algorithm but do not
have the spls package installed"))})
if("SL.step.plr" %in% SL.library) tryCatch(require(
  stepPlr), warning = function(...){ stop("you
have selected step.plr as a library algorithm
but do not have the stepPlr package installed")
})
if("SL.svm" %in% SL.library) tryCatch(require(
  e1071), warning = function(...){ stop("you have
selected svm as a library algorithm but do not
have the e1071 package installed"))})
if("SL.bagging" %in% SL.library) tryCatch(require(
  ipred), warning = function(...){ stop("you have
selected bagging as a library algorithm but do
not have the ipred package installed"))})
if("SL.bart" %in% SL.library) tryCatch(require(
  BayesTree), warning = function(...){ stop("you
have selected bart as a library algorithm but
do not have the BayesTree package installed"))})
if("SL.gbm.1" %in% SL.library) tryCatch(require(
  gbm), warning = function(...){ stop("you have
selected gbm as a library algorithm but do not
have the gbm package installed"))})
if("SL.gbm.2" %in% SL.library) tryCatch(require(
  gbm), warning = function(...){ stop("you have
selected gbm as a library algorithm but do not
have the gbm package installed"))})

```

```

    if("SL.mars" %in% SL.library) tryCatch(require(mda
    ), warning = function(...){ stop("you have
    selected mars as a library algorithm but do not
    have the mda package installed")})
    if("SL.earth" %in% SL.library) tryCatch(require(
    earth), warning = function(...){ stop("you have
    selected earth as a library algorithm but do
    not have the earth package installed")})
    if("SL.caret" %in% SL.library) tryCatch(require(
    caret), warning = function(...){ stop("you have
    selected caret as a library algorithm but do
    not have the caret package installed")})
  }

.computeCoef <- function(newZ, Y, cand.names, method, trim
, verbose, obsWeights){
  if(method=="NNLS"){
    require(nnls)
    cv.risk <- apply(newZ, 2, function(x) {
      mean(obsWeights * (x-Y)^2)})
    names(cv.risk) <- cand.names

    fit.nnls <- nnls(sqrt(obsWeights) * newZ,
      sqrt(obsWeights) * Y)
    if(verbose) {
      print(paste("Non-Negative least
      squares convergence: ", fit.
      nnls$mode==1))
    }
    init.coef <- coef(fit.nnls)
    init.coef[is.na(init.coef)] <- 0.0
    if(sum(init.coef) > 0) {
      update.coef <- init.coef/sum(init.
      coef)
    } else {
      warning("All algorithms have zero
      weight", call. = FALSE)
      update.coef <- init.coef
    }
    update.risk <- mean(obsWeights * (
      crossprod(t(newZ), update.coef) - Y)^2)
    if(update.risk > min(cv.risk)) {

```

```

        warning(paste("Cross-validated
            risk from convex combination is
            greater than cross-validated
            risk from one of the algorithms
            in the library:", cand.names[
            which.min(cv.risk)], ", You
            Should check the risk estimate
            of the SuperLearner using the
            CV.SuperLearner function"))
    }
}
if(method=="NNLS2"){
    cv.risk <- apply(newZ, 2, function(x) {
        mean(obsWeights * (x-Y)^2)})
    names(cv.risk) <- cand.names

    fit.nnls <- .NNLS(x = newZ, y = Y, wt =
        obsWeights)
    # How to check convergence of .NNLS?
    # if(verbose) {
    #     print(paste("Non-Negative least
        squares convergence: ", fit.nnls$mode
        ==1))
    # }
    init.coef <- fit.nnls$solution
    init.coef[init.coef <= 0] <- 0.0
    init.coef[is.na(init.coef)] <- 0.0
    if(sum(init.coef) > 0) {
        update.coef <- init.coef/sum(init.
            coef)
    } else {
        warning("All algorithms have zero
            weight", call. = FALSE)
        update.coef <- init.coef
    }
    update.risk <- mean(obsWeights * (
        crossprod(t(newZ), update.coef) - Y)^2)
    if(update.risk > min(cv.risk)) {
        warning(paste("Cross-validated
            risk from convex combination is
            greater than cross-validated
            risk from one of the algorithms

```

```

        in the library:", cand.names[
        which.min(cv.risk)], ", You
        Should check the risk estimate
        of the SuperLearner using the
        CV.SuperLearner function"))
    }
}
if(method=="NNloglik"){
  cv.risk <- apply(newZ, 2, function(x) { -
    sum(2 * obsWeights * ifelse(Y, log(x),
    log(1-x))) } )
  names(cv.risk) <- cand.names
  temp.newZ <- trimLogit(newZ, trim=trim)
  fit.nnloglik <- .NNloglik(x=temp.newZ, y=Y
    , wt = obsWeights)
  if(verbose) {
    print(paste("Non-Negative log-
    likelihood convergence: ", fit.
    nnloglik$convergence==0))
  }
  init.coef <- fit.nnloglik$par
  init.coef[init.coef <= 0] <- 0.0
  init.coef[is.na(init.coef)] <- 0
  if(sum(init.coef) > 0) {
    update.coef <- init.coef/sum(init.
    coef)
  } else {
    warning("All algorithms have zero
    weight", call. = FALSE)
    update.coef <- init.coef
  }
  update.risk <- -sum(2 * obsWeights *
    ifelse(Y, log(plogis(crossprod(t(temp.
    newZ), update.coef))), log(1-plogis(
    crossprod(t(temp.newZ), update.coef))))
    )
  if(update.risk > min(cv.risk)) {
    warning(paste("Cross-validated
    risk from convex combination is
    greater than cross-validated
    risk from one of the algorithms
    in the library:",cand.names[

```

```

                                which.min(cv.risk)], ", You
                                Should check the risk estimate
                                of the SuperLearner using the
                                CV.SuperLearner function"))
                                }
                                }
                                out <- list(cv.risk=cv.risk, init.coef=init.coef,
                                update.coef=update.coef)
                                return(out)
                                }

.computePred <- function(predY, init.coef, update.coef,
method, trim){
  if(method=="NNLS"){
    SL.pred.init <- crossprod(t(predY), init.
                                coef)
    SL.pred.update <- crossprod(t(predY),
                                update.coef)
    # SL.pred.init <- predY %*% init.coef
    # SL.pred.update <- predY %*% update.coef
  }
  if(method=="NNLS2"){
    SL.pred.init <- crossprod(t(predY), init.
                                coef)
    SL.pred.update <- crossprod(t(predY),
                                update.coef)
  }
  if(method=="NNloglik"){
    SL.pred.init <- plogis(crossprod(t(
      trimLogit(predY, trim=trim)), init.coef
    ))
    SL.pred.update <- plogis(crossprod(t(
      trimLogit(predY, trim=trim)), update.
      coef))
  }
  out <- list(SL.pred.init=SL.pred.init, SL.pred.
    update=SL.pred.update)
  return(out)
}

.createLibrary <- function(SL.library) {
  if (is.character(SL.library)) {

```

```

k.cand <- length(SL.library)
whichScreen <- matrix(1, nrow = 1, ncol =
  k.cand)
screenAlgorithm <- "All"
predAlgorithm <- data.frame(library = SL.
  library, row = 1, stringsAsFactors=
  FALSE)
} else {
  # LibraryNames <- lapply(SL.library, .fun =
    "[", 1)
  LibraryNames <- sapply(SL.library, FUN="["
    , 1)
  # NumberScreen <- (lapply(SL.library, .fun
    = length) - 1)
  NumberScreen <- (sapply(SL.library, FUN=
    length) - 1)
  if (sum(NumberScreen == 0) > 0) {
    for(ii in which(NumberScreen == 0)
      ){
      SL.library[[ii]] <- c(SL.
        library[[ii]], "All")
      NumberScreen[ii] <- 1
    }
  }
  screenAlgorithmFull <- unlist(sapply(SL.
    library, FUN="[" , -1))
  screenAlgorithm <- unique(
    screenAlgorithmFull)

  predAlgorithm <- data.frame(library = rep(
    LibraryNames, times=NumberScreen), row
    = match(screenAlgorithmFull,
      screenAlgorithm), stringsAsFactors =
      FALSE)
}

out <- list(predAlgorithm = predAlgorithm,
  screenAlgorithm = screenAlgorithm)
return(out)
}

```

Non-negative log likelihood optimization function

```
.NNloglik <- function(x, y, wt=rep(1, length(y)), start=
  rep(0,ncol(x)),...){
  # adapted from MASS pg 445
  fmin <- function(beta, X, y, w) {
    p <- plogis(crossprod(t(X), beta))
    -sum(2 * w * ifelse(y, log(p), log(1-p)))
  }
  gmin <- function(beta, X, y, w) {
    eta <- X %*% beta
    p <- plogis(eta)
    -2 * t(w * dlogis(eta) * ifelse(y, 1/p, -1
      /(1-p))) %*% X
  }
  fit <- optim(start, fmin, gmin, X=x, y=y, w=wt,
    method="L-BFGS-B", lower=0,...)
  invisible(fit)
}
```

```
# Non-negative least squares optimization function
.NNLS <- function(x, y, wt=rep(1, length(y)), ...) {
  require(quadprog)
  wX <- sqrt(wt) * x
  wY <- sqrt(wt) * y
  D <- t(wX) %*% wX
  d <- t(t(wY) %*% wX)
  A <- diag(ncol(wX))
  b <- rep(0, ncol(wX))
  fit <- solve.QP(D=D, d=d, A=t(A), b=b, meq=0)
  invisible(fit)
}
```

```
predict.SuperLearner <- function(object, newdata, X=NULL,
  Y=NULL, ...) {
  if(missing(newdata)) {
    return(object$SL.predict)
  }
  newdata <- as.data.frame(newdata)
  if(!identical(object$namesX, colnames(newdata))) {
    stop("The variable names and order in
      newdata must be identical to the
      variable names and order in X from the
      original Super Learner model\n")
  }
}
```

```

}
cand.names <- object$cand.names
k.cand <- length(cand.names)
fit.library <- matrix(NA, nrow=nrow(newdata), ncol
  =k.cand)

for(k in 1:k.cand){
  fit.library[, k] <- do.call(predict,
    list(object = object$fit.library[[
      k]],
        newdata = subset(newdata,
          select=object$
            whichScreen[object$SL.
              library$predAlgorithm[k
                , 2], ], drop=FALSE),
          family=object$family,
          X = if(is.null(X)) NULL
            else subset(X, select =
              object$whichScreen[
                object$SL.library$
                  predAlgorithm[k, 2], ],
                drop=FALSE),
          Y=Y))
}

if(object$method == "NNLS"){
  fit <- crossprod(t(fit.library), object$
    coef)
}
if(object$method == "NNloglik"){
  fit <- plogis(crossprod(t(trimLogit(fit.
    library, trim=object$trim.logit)),
    object$coef))
}
foo <- list(fit=fit, fit.library=fit.library, cand
  .names=cand.names)
return(foo)
}

print.SuperLearner <- function(x, ...) {
  cat("\nCall: ", deparse(x$call, width.cutoff = .9*
    getOption("width")), "\n\n", fill = getOption("

```

```

        width"))
    print(cbind(Risk = x$cv.risk, Coef = x$coef))
}

coef.SuperLearner <- function(object, ...) {
  object$coef
}

print.CV.SuperLearner <- function(x, ...) {
  cat("\nCall: ", deparse(x$call, width.cutoff = .9*
    getOption("width")), "\n\n", fill = getOption("
    width"))
  cat("Cross-validated predictions from the
    SuperLearner:  pred.SL \n\nCross-validated
    predictions from the discrete super learner (
    cross-validation selector):  pred.discreteSL \n
    \nWhich library algorithm was the discrete
    super learner:  whichDiscreteSL \n\nCross-
    validated prediction for all algorithms in the
    library:  pred.library\n")
}

coef.CV.SuperLearner <- function(object, ...) {
  object$coef.SL
}

summary.CV.SuperLearner <- function(object, ...) {
  library.names <- object$CV.fit.SL[[1]]$cand.names
  V <- object$V[[1]]
  n <- length(object$Y)
  Risk.SL <- rep(NA, length = V)
  Risk.dSL <- rep(NA, length = V)
  Risk.library <- matrix(NA, nrow = length(library.
    names), ncol = V)
  rownames(Risk.library) <- library.names

  if (object$method %in% c("NNLS", "NNLS2")) {
    for (ii in seq_len(V)) {
      Risk.SL[ii] <- mean(object$
        obsWeights[object$folds[[ii]]]
        * (object$Y[object$folds[[ii]]]
          - object$pred.SL[object$folds

```

```

[[ii]]))^2)
Risk.dSL[ii] <- mean(object$
  obsWeights[object$folds[[ii]]]
  * (object$Y[object$folds[[ii]]]
    - object$pred.discreteSL[
      object$folds[[ii]]])^2)
Risk.library[, ii] <- apply(object
  $pred.library[object$folds[[ii]]],
  ], 2, function(x) mean(
    object$obsWeights[object$folds
      [[ii]]] * (object$Y[object$
        folds[[ii]]] - x)^2))
}
# se <- rep.int(NA, (length(library.names)
  + 2))
se <- (1 / sqrt(n)) * c(sd(object$
  obsWeights * (object$Y - object$pred.SL
    )^2), sd(object$obsWeights * (object$Y
    - object$pred.discreteSL)^2), apply(
    object$pred.library, 2, function(x) sd(
      object$obsWeights * (object$Y - x)^2)))
} else if (object$method %in% c("NNloglik")) {
  for (ii in seq_len(V)) {
    Risk.SL[ii] <- -mean(object$
      obsWeights[object$folds[[ii]]]
      * ifelse(object$Y[object$folds
        [[ii]]], log(object$pred.SL[
          object$folds[[ii]]]), log(1-
            object$pred.SL[object$folds[[ii]]]))))
    Risk.dSL[ii] <- -mean(object$
      obsWeights[object$folds[[ii]]]
      * ifelse(object$Y[object$folds
        [[ii]]], log(object$pred.
          discreteSL[object$folds[[ii]]]
            , log(1 - object$pred.
              discreteSL[object$folds[[ii]]]
                )))
    Risk.library[, ii] <- apply(object
      $pred.library[object$folds[[ii]]],
      ], 2, function(x) {

```

```

        -mean(object$obsWeights[
            object$folds[[ii]] *
            ifelse(object$Y[object$
                folds[[ii]]], log(x),
                log(1-x)))
    })
}
se <- rep.int(NA, (length(library.names) +
    2))
} else {
    stop("summary function not available for
        SuperLearner with loss function/method
        used")
}

Table <- data.frame(Algorithm = c("Super Learner",
    "Discrete SL", library.names), Ave = c(mean(
    Risk.SL), mean(Risk.dSL), apply(Risk.library,
    1, mean)), se = se, Min = c(min(Risk.SL), min(
    Risk.dSL), apply(Risk.library, 1, min)), Max =
    c(max(Risk.SL), max(Risk.dSL), apply(Risk.
    library, 1, max)))
out <- list(call = object$call, method = object$
    method, V=V, Risk.SL = Risk.SL, Risk.dSL = Risk
    .dSL, Risk.library = Risk.library, Table =
    Table)
class(out) <- "summary.CV.SuperLearner"
return(out)
}

print.summary.CV.SuperLearner <- function(x, digits = max
    (2, getOption("digits") - 2), ...) {
    cat("\nCall: ", deparse(x$call, width.cutoff = .9*
        getOption("width")), "\n", fill = getOption("
        width"))
    cat("Risk is based on: ")
    if(x$method %in% c("NNLS", "NNLS2")) {
        cat("Least Squares (Mean Squared Error)")
    } else if (x$method %in% c("NNloglik")) {
        cat("Negative Log Likelihood (-2*log(L))")
    } else {
        stop("summary method not available")
    }
}

```

```

    }
    cat("\n\nAll risk estimates are based on V = ", x$
        V, "\n\n")
    print(x$Table, digits = digits, row.names = FALSE)
}

trimLogit <- function(x, trim=0.00001) {
    x[x < trim] <- trim
    x[x > (1-trim)] <- (1-trim)
    foo <- log(x/(1-x))
    return(foo)
}

CVFolds <- function(V, N.all, shuffle, id, stratifyCV, Y)
{
    if(!stratifyCV) {
        if(shuffle) {
            if(is.null(id)) {
                DATA.split <- split(sample
                    (1:N.all), rep(1:V,
                        length=N.all))
            } else {
                n.id <- length(unique(id))
                id.split <- split(sample
                    (1:n.id), rep(1:V,
                        length=n.id))
                DATA.split <- vector("list", V)
                for(v in seq(V)) {
                    DATA.split[[v]] <-
                        which(id %in%
                            unique(id)[id.
                                split[[v]]])
                }
            }
        } else {
            if(is.null(id)) {
                DATA.split <- split(1:N.
                    all, rep(1:V, length=N.
                        all))
            } else {

```

```

n.id <- length(unique(id))
id.split <- split(1:n.id,
  rep(1:V, length=n.id))
DATA.split <- vector("list", V)
for(v in seq(V)) {
  DATA.split[[v]] <-
    which(id %in%
      unique(id)[id.
        split[[v]]])
}
}
} else {
  if(length(unique(Y)) != 2) {
    stop("stratifyCV only implemented
      for binary Y")
  }
  if(sum(Y) < V | sum(!Y) < V) {
    stop("number of (Y=1) or (Y=0) is
      less than the number of folds")
  }
  if(shuffle) {
    if(is.null(id)) {
      within.split <-
        suppressWarnings(tapply
          (1:N.all, INDEX = Y,
            FUN = split, rep(1:V)))
      DATA.split <- vector("list", length = V)
      names(DATA.split) <- paste
        (seq(V))
      for(vv in seq(V)) {
        DATA.split[[vv]]
          <- c(within.
            split[[1]][[vv
              ]], within.
            split[[2]][[vv
              ]])
      }
    } else {

```

```

                                stop("stratified sampling
                                    with id not currently
                                    implemented")
                                }
                                } else {
                                if(is.null(id)) {
                                    within.split <-
                                        suppressWarnings(tapply
                                            (1:N.all, INDEX = Y,
                                            FUN = split, rep(1:V)))
                                    DATA.split <- vector("list"
                                        ", length = V)
                                    names(DATA.split) <- paste
                                        (seq(V))
                                    for(vv in seq(V)) {
                                        DATA.split[[vv]]
                                            <- c(within.
                                                split[[1]][[vv
                                                ]], within.
                                                split[[2]][[vv
                                                ]])
                                    }
                                } else {
                                    stop("stratified sampling
                                        with id not currently
                                        implemented")
                                }
                                }
                                }
                                invisible(DATA.split)
                                }
}

```

A.2 Wrapper functions

```

listWrappers <- function(what = "both") {
    if(what == "both") {
        cat("All prediction algorithm wrappers in
            SuperLearner:\n")
        print(ls("package:SuperLearner", pattern="
            ^[S]L"))
        cat("\nAll screening algorithm wrappers in
            SuperLearner:\n")
    }
}

```

```

        print("All")
        print(ls("package:SuperLearner", pattern="
          screen"))
      } else if(what == "SL") {
        cat("All prediction algorithm wrappers in
          SuperLearner:\n")
        print(ls("package:SuperLearner", pattern="
          ^[S]L"))
      } else if(what == "screen") {
        cat("All screening algorithm wrappers in
          SuperLearner:\n")
        print("All")
        print(ls("package:SuperLearner", pattern="
          screen"))
      } else {
        cat("All functions in SuperLearner:\n")
        print(ls("package:SuperLearner"))
      }
    }
  }
}

```

```

.listPvalueGLM <- function(Y, X, family=gaussian()) {
  p <- ncol(X)
  list.p <- rep(NA, p)
  names(list.p) <- names(X)
  if (family$family == "gaussian") {
    list.p <- apply(X, 2, function(x) {
      ifelse(var(x) <= 0, 1, cor.test(x,
        Y)$p.value)
    })
  } else {
    list.p <- apply(X, 2, function(x) {
      ifelse(var(x) <= 0, 1, summary(glm
        (Y~x, family=family))$
        coefficients[2,4])
    })
  }
  return(list.p)
}

```

```

.listPvalueCor <- function(Y, X, method) {
  p <- ncol(X)
  list.p <- rep(NA, p)

```

```

names(list.p) <- names(X)
list.p <- apply(X, 2, function(x) {
  ifelse(var(x) <= 0, 1, cor.test(x, Y,
    method = method)$p.value)
})
return(list.p)
}

screen.glmP <- function(Y.temp, X.temp, family, minPvalue
  = 0.1, minscreen = 2, ...) {
  listp <- .listPvalueGLM(Y=Y.temp, X=X.temp, family
    =family)
  whichVariable <- (listp <= minPvalue)
  if (sum(whichVariable) <= minscreen) {
    whichVariable[rank(listp) <= minscreen] <-
      TRUE
  }
  return(whichVariable)
}

screen.glmRank <- function(Y.temp, X.temp, family, rank =
  2, ...) {
  if(rank > ncol(X.temp)) {
    rank <- ncol(X.temp)
  }
  listp <- .listPvalueGLM(Y=Y.temp, X=X.temp, family
    =family)
  whichVariable <- (rank(listp) <= rank)
  return(whichVariable)
}

screen.corP <- function(Y.temp, X.temp, method = "pearson"
  , minPvalue = 0.1, minscreen = 2, ...) {
  listp <- .listPvalueCor(Y=Y.temp, X=X.temp, method
    = method)
  whichVariable <- (listp <= minPvalue)
  if (sum(whichVariable) <= minscreen) {
    whichVariable[rank(listp) <= minscreen] <-
      TRUE
  }
  return(whichVariable)
}

```

```

screen.corRank <- function(Y.temp, X.temp, method = "
  pearson", rank = 2, ...) {
  if(rank > ncol(X.temp)) {
    rank <- ncol(X.temp)
  }
  listp <- .listPvalueCor(Y=Y.temp, X=X.temp, method
    = method)
  whichVariable <- (rank(listp) <= rank)
  return(whichVariable)
}

screen.glmnet <- function (Y.temp, X.temp, family, glmnet.
  alpha = 1, minscreen = 2, ...)
{
  tryCatch(require(glmnet), warning = function(...){
    stop("you have selected glmnet as a library
      algorithm but do not have the glmnet package
      installed")})
  fit.first <- glmnet(x = as.matrix(X.temp), y=Y.
    temp, family=as.character(family)[1], alpha=
    glmnet.alpha)
  cv.net.fit <- SuperLearner:::.cv.glmnet(x = as.
    matrix(X.temp), y=Y.temp, K=10, lambda=fit.
    first$lambda, alpha=glmnet.alpha, family=as.
    character(family)[1])
  whichVariable <- (fit.first$beta[, which.min(cv.net.
    fit$cv)] != 0)
  if (sum(whichVariable) < minscreen) {
    warning("fewer than minscreen variables passed the
      glmnet screen, increased lambda to allow
      minscreen variables")
    sumCoef <- apply(fit.first$beta, 2,
      function(x) sum((x != 0)))
    newCut <- which.max(sumCoef >= minscreen)
    whichVariable <- (fit.first$beta[, newCut]
      != 0)
  }
  return(whichVariable)
}

```

```

screen.randomForest <- function (Y.temp, X.temp, family,
  nVar = 10, ntree = 1000, ...)
{
  tryCatch(require(randomForest), warning = function
    (...) {
      stop("you have selected randomForest as a library
        algorithm but do not have the randomForest
        package installed")
    })
  if (family$family == "gaussian") {
    rank.rf.fit <- randomForest(Y.temp~., data
      =X.temp, ntree=ntree)
  }
  if (family$family == "binomial") {
    rank.rf.fit <- randomForest(y=as.factor(Y.
      temp), x=X.temp, ntree=ntree)
  }

  whichVariable <- (rank(-rank.rf.fit$importance) <=
    nVar)
  return(whichVariable)
}

screen.SIS <- function (Y.temp, X.temp, family, nsis =
  NULL, tune.method="BIC", inittype="NoPen", vartype="
  second", useISISind = TRUE, minScreen = 5,...)
{
  tryCatch(require(SIS), warning = function(...) {
    stop("you have selected SIS as a library algorithm
      but do not have the SIS package installed")
  })
  if (family$family == "gaussian") {
    fitSIS <- GLMvarISISscad(x=X.temp, y=Y.
      temp, family=family, nsis=nsis, tune.
      method=tune.method, inittype=inittype,
      vartype=vartype)
  }
  if (family$family == "binomial") {
    fitSIS <- GLMvarISISscad(x=X.temp, y=Y.
      temp, family=family, nsis=nsis, tune.
      method=tune.method, inittype=inittype,
      vartype=vartype)
  }
}

```

```

    }
    if(useISISind) {
      whichVariable <- (1:ncol(X.temp) %in% fitSIS$
        ISISind)
    } else {
      whichVariable <- (1:ncol(X.temp) %in%
        fitSIS$SISind)
    }
    # check more than minScreen in screened set
    if(sum(whichVariable) < minScreen) {
      warning("fewer than minScreen variables in
        screen.SIS, using initial ranking")
      whichVariable <- (fitSIS$initRANKorder1 <
        minScreen)
    }
    return(whichVariable)
  }
}

All <- function(...) {
  return(TRUE)
}

# screen functions must return a logical vector of length
  ncol(X)
screen.template <-function (Y.temp, X.temp, family,
  obsWeights, id, ...)
{
  if (family$family == "gaussian") {

  }
  if (family$family == "binomial") {

  }
  whichVariable <- rep(TRUE, ncol(X.temp))
  return(whichVariable)
}

SL.bagging <- function(Y.temp, X.temp, newX.temp, family,
  nbagg=100, control = rpart.control(xval = 0,
  maxsurrogate = 0, minsplit = 20, cp = 0.01, maxdepth =
  30), ...){

```

```

tryCatch(require(ipred), warning = function(...){
  stop("you have selected bagging as a library
  algorithm but do not have the ipred package
  installed"))})

if(family$family=="gaussian"){
  fit.bagg <- ipredbagg(y=Y.temp, X=X.temp,
    nbagg= nbagg, control = control)
  out <- predict(fit.bagg, newdata=newX.temp,
    aggregation = "average")
}
if(family$family=="binomial"){
  fit.bagg <- ipredbagg(y=factor(Y.temp), X=X
    .temp, nbagg= nbagg, control = control)
  out <- predict(fit.bagg, newdata=newX.temp,
    type="prob", aggregation = "average")
  [, 2]
}
fit <- list(object=fit.bagg)
foo <- list(out=out, fit=fit)
class(foo$fit) <- c("SL.bagging")
return(foo)
}

#
predict.SL.bagging <- function(object, newdata, family, X=
  NULL, Y=NULL,...) {
  tryCatch(require(ipred), warning = function(...){
    stop("you have selected bagging as a library
    algorithm but do not have the ipred package
    installed"))})
  if(family$family=="gaussian"){
    out <- predict(object = object$object,
      newdata=newdata, aggregation = "average
      ")
  }
  if(family$family=="binomial"){
    out <- predict(object = object$object,
      newdata=newdata, type="prob",
      aggregation = "average")[, 2]
  }
  return(out)
}

```

```

}

SL.bagTree <- function(Y.temp, X.temp, newX.temp, family,
  cp =0.01, minsplit = 20, xval=0, maxdepth=30, ntree
  =500, weights = "oob", ...) {
  tryCatch(require(rpart), warning = function(...){
    stop("you have selected bagTree as a
      library algorithm but do not have the
      rpart package installed")
  })

  nn <- length(Y.temp)
  if(family$family=="gaussian"){
    predTrees <- matrix(NA, nrow=(nn + nrow(
      newX.temp)), ncol=ntree)
    coefTrees <- rep(NA, ntree)
    fit.rpart <- vector("list", ntree)
    for(mm in seq(ntree)) {
      n.oob <- 0
      while(n.oob < 1) {
        whichRow <- sample(1:nn,
          replace=TRUE)
        oob <- (1:nn)[!(1:nn %in%
          whichRow)]
        n.oob <- length(oob)
      }
      bootY <- Y.temp[whichRow]
      bootX <- X.temp[whichRow, ,drop=
        FALSE]
      fit.rpart[[mm]] <- rpart(Y.temp~.,
        data=data.frame(Y.temp=bootY,
          bootX), control = rpart.control
          (cp=cp, minsplit=minsplit, xval
            =xval, maxdepth=maxdepth),
          method="anova")
      predTrees[, mm] <- predict(fit.
        rpart[[mm]], newdata=rbind(X.
          temp, newX.temp))
      if(weights == "oob") {
        coefTrees[mm] <- mean((Y.
          temp[oob] - predTrees[
            oob, mm])^2)
      }
    }
  }
}

```

```

    } else {
      coefTrees[mm] <- 1
    }
  }

}

if(family$family=="binomial"){
  predTrees <- matrix(NA, nrow=(nn + nrow(
    newX.temp)), ncol=ntree)
  coefTrees <- rep(NA, ntree)
  fit.rpart <- vector("list", ntree)
  for(mm in seq(ntree)){
    n.oob <- 0
    while(n.oob < 1) {
      whichRow <- sample(1:nn,
        replace=TRUE)
      oob <- (1:nn)[!(1:nn %in%
        whichRow)]
      n.oob <- length(oob)
    }
    bootY <- Y.temp[whichRow]
    bootX <- X.temp[whichRow, ,drop=
      FALSE]
    fit.rpart[[mm]] <- rpart(Y.temp~.,
      data=data.frame(Y.temp=bootY,
        bootX), control = rpart.control
        (cp=cp, minsplit=minsplit, xval
        =xval, maxdepth=maxdepth),
        method="class")
    predTrees[, mm] <- predict(fit.
      rpart[[mm]], newdata=rbind(X.
        temp, newX.temp), type="prob")
    [, 2]
    if(weights == "oob") {
      # currently using L2 risk
      coefTrees[mm] <- mean((Y.
        temp[oob] - predTrees[
        oob, mm])^2)
    } else {
      coefTrees[mm] <- 1
    }
  }
}

```

```

    }

    # fit.rpart <- rpart(Y.temp~., data=data.
      frame(Y.temp, X.temp), control = rpart.
      control(cp=cp, minsplit=minsplit, xval=
        xval, maxdepth=maxdepth), method="class
        ")
  }

  out <- crossprod(t(predTrees[-c(1:nn), ]), ((1/
    coefTrees)/sum((1/coefTrees))))

  fit <- list(object = fit.rpart, coefTrees =
    coefTrees, ntree = ntree)
  foo <- list(out=out, fit=fit)
  class(foo$fit) <- c("SL.bagTree")
  return(foo)
}

#
predict.SL.bagTree <- function(object, newdata, family, X=
  NULL, Y=NULL,...) {
  tryCatch(require(rpart), warning = function(...){
    stop("you have selected bagTree as a library
      algorithm but do not have the rpart package
      installed")})

  trees <- matrix(NA, nrow=nrow(newdata), ncol =
    object$ntree)
  if(family$family=="binomial"){
    for(mm in seq(object$ntree)) {
      trees[, mm] <- predict(object$
        object[[mm]], newdata=newdata,
        type="prob")[, 2]
    }
  }
  if(family$family=="gaussian") {
    for(mm in seq(object$ntree)) {
      trees[, mm] <- predict(object$
        object[[mm]], newdata=newdata)
    }
  }
}

```

```

        out <- crossprod(t(trees), ((1/object$coefTrees)/
            sum((1/object$coefTrees))))
        return(out)
    }

SL.bagTree.unit <- function(..., weights = "unit") {
    SL.bagTree(..., weights=weights)
}

SL.bart <- function(Y.temp, X.temp, newX.temp, family,
    ntree=300, sigdf=3, sigquant=0.90, k=2, power=2, base
    =0.95, binaryOffset=0, ndpost=1000, nskip=100, ...) {
    tryCatch(require(BayesTree), warning = function
        (...)) {
        stop("you have selected bart as a library
            algorithm but do not have the BayesTree
            package installed")
    } )

    if(family$family=="gaussian"){
        fitBart <- bart(x.train=X.temp, y.train=Y.
            temp, x.test=newX.temp, ntree=ntree,
            sigdf=sigdf, sigquant=sigquant, k=k,
            power=power, base=base, binaryOffset=
            binaryOffset, ndpost=ndpost, nskip=
            nskip, verbose=FALSE)
    }
    if(family$family=="binomial"){
        fitBart <- bart(x.train=X.temp, y.train=as
            .factor(Y.temp), x.test=newX.temp,
            ntree=ntree, sigdf=sigdf, sigquant=
            sigquant, k=k, power=power, base=base,
            binaryOffset=binaryOffset, ndpost=
            ndpost, nskip=nskip, verbose=FALSE)
    }
    if(family$family=="gaussian"){
        out <- fitBart$yhat.test.mean
    }
    if(family$family=="binomial"){
        out <- pnorm(apply(fitBart$yhat.test, 2,
            mean))
    }

```

```

    }
    fit <- list(object = fitBart)
    foo <- list(out=out, fit=fit)
    class(foo$fit) <- c("SL.bart")
    return(foo)
}

#
predict.SL.bart <- function(object, newdata, family, X=
  NULL, Y=NULL, ...) {
  tryCatch(require(BayesTree), warning = function
    (...){ stop("you have selected bart as a
      library algorithm but do not have the BayesTree
      package installed")})

  stop("no predict method currently available for
    bart")
}

SL.bayesglm <- function(Y.temp, X.temp, newX.temp, family,
  obsWeights, ...){
  tryCatch(require(arm), warning = function(...){
    stop("you have selected bayesglm as a library
      algorithm but do not have the arm package
      installed")})
  fit.glm <- bayesglm(Y.temp~., data=X.temp, family=
    family, weights = obsWeights)
  out <- predict(fit.glm, newdata=newX.temp, type="
    response")
  fit <- list(object=fit.glm)
  foo <- list(out=out, fit=fit)
  class(foo$fit) <- c("SL.bayesglm")
  return(foo)
}

predict.SL.bayesglm <- function(object, newdata, ...){
  tryCatch(require(arm), warning = function(...){
    stop("you have selected bayesglm as a library
      algorithm but do not have the arm package
      installed")})
  out <- predict(object=object$object, newdata=
    newdata, type="response")

```

```

        out
    }

SL.caret <- function(Y.temp, X.temp, newX.temp, family,
  obsWeights, method = "rf", tuneLength = 3, trControl =
  trainControl(method = "cv", number = 10, verboseIter =
  TRUE), ...) {
  tryCatch(require(caret), warning = function(...) {
    stop("you have selected caret as a library
        algorithm but do not have the caret package
        installed")
  })
  if (family$family == "gaussian") {
    fit.train <- caret::train(x = X.temp, y =
      Y.temp, weights = obsWeights, metric =
      "RMSE", method = method, tuneLength =
      tuneLength, trControl = trControl)
    out <- predict(fit.train, newdata = newX.
      temp, type = "raw")
  }
  if (family$family == "binomial") {
    # outcome must be factor, and have real
    # labels
    Y.temp.f <- as.factor(Y.temp)
    levels(Y.temp.f) <- c("A0", "A1")
    fit.train <- caret::train(x = X.temp, y =
      Y.temp.f, weights = obsWeights, metric
      = "Accuracy", method = method,
      tuneLength = tuneLength, trControl =
      trControl)
    out <- predict(fit.train, newdata = newX.
      temp, type = "prob")[, 2]
  }
  fit <- list(object = fit.train)
  foo <- list(out = out, fit = fit)
  class(foo$fit) <- c("SL.caret")
  return(foo)
}

predict.SL.caret <- function(object, newdata, ...) {
  tryCatch(require(caret), warning = function(...) {

```

```

        stop("you have selected caret as a library
              algorithm but do not have the caret package
              installed")
    })
    if (object$object$modelType == "Regression") {
        out <- predict(object$object, newdata =
            newdata, type = "raw")
    } else if (object$object$modelType == "
        Classification") {
        out <- predict(object$object, newdata =
            newdata, type = "prob")[, 2]
    }
    return(out)
}

# how to change to a different method:
SL.caret.rpart <- function(..., method = "rpart") {
    SL.caret(..., method = method)
}

# cforest {party}
SL.cforest <- function(Y.temp, X.temp, newX.temp, family,
    obsWeights, ...){
    tryCatch(require(party), warning = function(...) {
        stop("you have selected cforest as a library
            algorithm but do not have the party package
            installed")})
    if(family$family=="gaussian"){
        fit.cforest <- cforest(Y.temp~., data=data
            .frame(Y.temp, X.temp), controls =
            cforest_unbiased(ntree=1000, mtry=max(
            floor(ncol(X.temp)/3), 1)), weights =
            obsWeights)
    }
    if(family$family=="binomial"){
        stop("Currently only works with gaussian
            data \ncforest can not return predicted
            probabilities")
    }
    out <- predict(object = fit.cforest, newdata=newX.
        temp)
    fit <- list(object=fit.cforest)

```

```

        foo <- list(out=out, fit=fit)
        class(foo$fit) <- c("SL.cforest")
        return(foo)
    }

#
predict.SL.cforest <- function(object, newdata, family, X=
  NULL, Y=NULL, ...) {
  tryCatch(require(party), warning = function(...) {
    stop("you have selected cforest as a library
    algorithm but do not have the party package
    installed")})
  out <- predict(object = object$object, newdata=
    newdata)
  out
}

SL.DSA <- function(Y.temp, X.temp, newX.temp, family,
  obsWeights, maxsize=ncol(X.temp), maxorderint=1,
  maxsumofpow=1, Dmove=TRUE, Smove=TRUE, vfold=5, ...){
  tryCatch(require(DSA), warning = function(...) {
    stop("you have selected DSA as a library
    algorithm but do not have the DSA package
    installed")})
  dsaweights <- matrix(obsWeights, nrow = (vfold +1)
    , ncol = nrow(X.temp), byrow = TRUE)
  fit.DSA <- DSA(Y.temp~1, data=data.frame(Y.temp,X.
    temp), family=family, maxsize=maxsize,
    maxorderint=maxorderint, maxsumofpow=
    maxsumofpow, Dmove=Dmove, Smove=Smove, vfold=
    vfold, weights = dsaweights)
  out <- predict(fit.DSA, newdata=newX.temp)
  if(family$family=="binomial"){ out <- 1 / (1 + exp
    (-out))}
  fit <- list(object=fit.DSA)
  foo <- list(out=out, fit=fit)
  class(foo$fit) <- c("SL.DSA")
  return(foo)
}

#

```

```

predict.SL.DSA <- function(object, newdata, family, X=NULL
, Y=NULL,...) {
  tryCatch(require(DSA), warning = function(...){
    stop("you have selected DSA as a library
    algorithm but do not have the DSA package
    installed")})
  out <- predict(object=object$object, newdata=
    newdata)
  if(family$family=="binomial"){ out <- 1 / (1 + exp
    (-out))}
  return(out)
}

SL.DSA.2 <- function(..., X.temp, maxsize=2*ncol(X.temp),
  maxorderint=2, maxsumofpow=2, Smove=FALSE, vfold=10) {
  SL.DSA(..., X.temp=X.temp, maxsize=maxsize,
    maxorderint=maxorderint, maxsumofpow=
    maxsumofpow, Smove=Smove, vfold=vfold)
}

SL.earth <- function(Y.temp, X.temp, newX.temp, family,
  obsWeights, id, degree = 2, penalty = 3, nk = max(21, 2
  *ncol(X.temp) + 1), ...){
  tryCatch(require(earth), warning = function(...) {
    stop("you have selected earth as a library
    algorithm but do not have the earth
    package installed")
  })
  if(family$family=="gaussian"){
    fit.earth <- earth::earth(x = X.temp, y =
      Y.temp, degree = degree, nk = nk,
      penalty = penalty)
  }
  if(family$family=="binomial"){
    fit.earth <- earth::earth(x = X.temp, y =
      Y.temp, degree = degree, nk = nk,
      penalty = penalty, glm = list(family =
      binomial))
  }
  out <- predict(fit.earth, newdata = newX.temp,
    type = "response")
  fit <- list(object = fit.earth)
}

```

```

    foo <- list(out=out, fit=fit)
    class(foo$fit) <- c("SL.earth")
    return(foo)
}

#
predict.SL.template <- function(object, newdata, family, X
  =NULL, Y=NULL,...) {
  tryCatch(require(earth), warning = function(...) {
    stop("you have selected earth as a library
        algorithm but do not have the earth
        package installed")
  })
  out <- predict(object$object, newdata = newdata,
    type = "repsonse")
  return(out)
}

SL.gam <- function(Y.temp, X.temp, newX.temp, family,
  obsWeights, deg.gam=2,...){
  tryCatch(require(gam), warning = function(...) {
    stop("you have selected gam as a library
        algorithm but do not have the gam package
        installed")
  })
  cts.x <- apply(X.temp, 2, function(x){ (length(
    unique(x)) > 4)})
  if (sum(!cts.x) > 0) {
    gam.model <- as.formula(paste("Y.temp~",
      paste(paste("s(", colnames(X.temp[, cts.x
        , drop=FALSE])), ",", deg.gam, ")"), sep=""),
      collapse="+"), "+", paste(colnames(X.temp
        [, !cts.x, drop=FALSE]), collapse="+"))
  } else {
    gam.model <- as.formula(paste("Y.temp~",
      paste(paste("s(", colnames(X.temp[, cts.x
        , drop=FALSE])), ",", deg.gam, ")"), sep=""),
      collapse="+"))
  }
  # fix for when all variables are binomial
  if (sum(!cts.x)==length(cts.x)) {
    gam.model <- as.formula(paste("Y.temp~",
      paste(colnames(X.temp), collapse="+"),

```

```

        sep=""))
    }

    fit.gam <- gam::gam(gam.model, data=X.temp, family
      =family, control=gam.control(maxit=50, bf.maxit
      =50), weights = obsWeights)
    out <- predict(fit.gam, newdata=newX.temp, type="
      response")
    fit <- list(object=fit.gam)
    foo <- list(out=out, fit=fit)
    class(foo$fit) <- c("SL.gam")
    return(foo)
  }

predict.SL.gam <- function(object, newdata, ...){
  tryCatch(require(gam), warning = function(...){
    stop("you have selected gam as a library
      algorithm but do not have the gam package
      installed")})
  out <- gam::predict.gam(object=object$object,
    newdata=newdata, type="response")
  out
}

SL.gam.3 <- function(..., deg.gam=3){SL.gam(..., deg.gam=
  deg.gam)}
SL.gam.4 <- function(..., deg.gam=4){SL.gam(..., deg.gam=
  deg.gam)}
SL.gam.5 <- function(..., deg.gam=5){SL.gam(..., deg.gam=
  deg.gam)}

SL.gbm.1 <- function(Y.temp, X.temp, newX.temp, family,
  obsWeights, gbm.trees=10000, ...){
  tryCatch(require(gbm), warning = function(...){
    stop("you have selected gbm as a library
      algorithm but do not have the gbm package
      installed")})
  if(family$family=="gaussian"){
    gbm.model <- as.formula(paste("Y.temp~",
      paste(colnames(X.temp), collapse="+")))
    fit.gbm1 <- gbm(formula=gbm.model, data=X.
      temp, distribution="gaussian", n.trees=

```

```

        gbm.trees, interaction.depth=1, cv.
        folds=5, keep.data=TRUE, weights =
        obsWeights, verbose=FALSE)
    best.iter1 <- gbm.perf(fit.gbm1, method="
        cv", plot.it=FALSE)
    out <- predict(fit.gbm1, newdata=newX.temp
        ,best.iter1, type="response")
    fit <- list(object=fit.gbm1, n.trees=best.
        iter1)
  }
  if(family$family=="binomial"){
    gbm.model <- as.formula(paste("Y.temp~",
        paste(colnames(X.temp), collapse="+")))
    fit.gbm1 <- gbm(formula=gbm.model, data=X.
        temp, distribution="bernoulli", n.trees
        =gbm.trees, interaction.depth=1, cv.
        folds=5, keep.data=TRUE, verbose=FALSE,
        weights = obsWeights)
    best.iter1 <- gbm.perf(fit.gbm1, method="
        cv", plot.it=FALSE)
    out <- predict(fit.gbm1, newdata=newX.temp
        , best.iter1, type="response")
    fit <- list(object=fit.gbm1, n.trees=best.
        iter1)
  }
  foo <- list(out=out, fit=fit)
  class(foo$fit) <- c("SL.gbm")
  return(foo)
}

SL.gbm.2 <- function(Y.temp, X.temp, newX.temp, family,
  obsWeights, gbm.trees=10000,...){
  tryCatch(require(gbm), warning = function(...){
    stop("you have selected gbm as a library
    algorithm but do not have the gbm package
    installed")})
  if(family$family=="gaussian"){
    gbm.model <- as.formula(paste("Y.temp~",
        paste(colnames(X.temp), collapse="+")))
    fit.gbm1 <- gbm(formula=gbm.model, data=X.
        temp, distribution="gaussian",n.trees=
        gbm.trees, interaction.depth=2, cv.

```

```

        folds=5, keep.data=TRUE, weights =
        obsWeights, verbose=FALSE)
    best.iter1 <- gbm.perf(fit.gbm1, method="
        cv", plot.it=FALSE)
    out <- predict(fit.gbm1, newdata=newX.temp
        , best.iter1, type="response")
    fit <- list(object=fit.gbm1, n.trees=best.
        iter1)
  }
  if(family$family=="binomial"){
    gbm.model <- as.formula(paste("Y.temp~",
        paste(colnames(X.temp),collapse="+")))
    fit.gbm1 <- gbm(formula=gbm.model, data=X.
        temp, distribution="bernoulli", n.trees
        =gbm.trees, interaction.depth=2, cv.
        folds=5, keep.data=TRUE, weights =
        obsWeights, verbose=FALSE)
    best.iter1 <- gbm.perf(fit.gbm1, method="
        cv", plot.it=FALSE)
    out <- predict(fit.gbm1, newdata=newX.temp
        , best.iter1, type="response")
    fit <- list(object=fit.gbm1, n.trees=best.
        iter1)
  }
  foo <- list(out=out, fit=fit)
  class(foo$fit) <- c("SL.gbm")
  return(foo)
}

predict.SL.gbm <- function(object, newdata,...){
  tryCatch(require(gbm), warning = function(...) {
    stop("you have selected gbm as a library
    algorithm but do not have the gbm package
    installed")})
  out <- predict(object$object, newdata=newdata, n.
    trees=object$n.trees, type="response")
  out
}

SL.glm <- function(Y.temp, X.temp, newX.temp, family,
  obsWeights, ...){

```

```

fit.glm <- glm(Y.temp~., data=X.temp, family=
  family, weights = obsWeights)
out <- predict(fit.glm, newdata=newX.temp, type="
  response")
fit <- list(object=fit.glm)
foo <- list(out=out, fit=fit)
class(foo$fit) <- c("SL.glm")
return(foo)
}

predict.SL.glm <- function(object, newdata, ...){
  out <- predict(object=object$object, newdata=
    newdata, type="response")
  out
}

SL.glmnet <- function(Y.temp, X.temp, newX.temp, family,
  obsWeights, glmnet.alpha=1, K = 10, ...){
  tryCatch(require(glmnet), warning = function(...){
    stop("you have selected glmnet as a library
      algorithm but do not have the glmnet package
      installed")})
  fit.first <- glmnet(x=as.matrix(X.temp), y=Y.temp,
    family=as.character(family)[1], alpha=glmnet.
    alpha, weights = obsWeights)
  cv.net.fit <- SuperLearner:::.cv.glmnet(x=as.
    matrix(X.temp), y=Y.temp, K=K, lambda=fit.first
    $lambda, alpha=glmnet.alpha, family=as.
    character(family)[1], obsWeights = obsWeights)
  min.lambda <- cv.net.fit$lambda[which.min(cv.net.
    fit$cv)]
  out <- predict(fit.first, newx=as.matrix(newX.temp
    ), s=min.lambda, type="response")
  fit <- list(object=fit.first, s=min.lambda, cv.net
    .fit=cv.net.fit)
  foo <- list(out=out, fit=fit)
  class(foo$fit) <- c("SL.glmnet")
  return(foo)
}

predict.SL.glmnet <- function(object, newdata, ...){

```

```

    tryCatch(require(glmnet), warning = function(...){
      stop("you have selected glmnet as a library
      algorithm but do not have the glmnet package
      installed"))})
    out <- predict(object$object, newx=as.matrix(
      newdata), s=object$s, type="response")
    out
  }

SL.glmnet.alpha25 <- function(..., alpha=.25){ SL.glmnet
  (... , glmnet.alpha=alpha)}
SL.glmnet.alpha50 <- function(..., alpha=.50){ SL.glmnet
  (... , glmnet.alpha=alpha)}
SL.glmnet.alpha75 <- function(..., alpha=.75){ SL.glmnet
  (... , glmnet.alpha=alpha)}

## required functions
.cv.glmnet <- function(x, y, K, lambda, alpha, family,
  obsWeights = rep.int(1, length(y)))
{
  all.folds <- split(sample(1:length(y)), rep(1:K,
    length = length(y)))
  residmat <- matrix(NA, length(lambda), K)
  for (i in seq(K)) {
    omit <- all.folds[[i]]
    fit <- glmnet(x[-omit, , drop = FALSE], y[-omit],
      family=as.character(family)[1], alpha=alpha,
      lambda=lambda, weights = obsWeights[-omit])
    fit <- predict(fit, newx=x[omit, , drop = FALSE],
      type="response")
    if (length(omit) == 1)
      fit <- matrix(fit, nrow = 1)
    foo <- apply((obsWeights[omit]*(y[omit] - fit)^2),
      2, mean)
    residmat[seq(along=foo), i] <- foo
  }
  cv <- apply(residmat, 1, mean)
  cv.error <- sqrt(apply(residmat, 1, var)/K)
  object <- list(lambda=lambda, cv = cv, cv.error = cv.
    error)
  invisible(object)
}

```

```

SL.knn <- function(Y.temp, X.temp, newX.temp, family, k
=10, ...){
  tryCatch(require(class), warning = function(...){
    stop("you have selected knn as a library
    algorithm but do not have the class package
    installed")})
  if(family$family=="gaussian") {
    stop("SL.knn only available for family =
    binomial()")
  }
  fit.knn <- knn(train=X.temp, test=newX.temp, k=k,
    cl=Y.temp, prob=TRUE)
  out <- (as.numeric(fit.knn)-1)*attr(fit.knn,"prob"
    ) + (1-(as.numeric(fit.knn)-1))*(1-attr(fit.knn
    ,"prob"))
  fit <- list(k=k)
  foo <- list(out=out, fit=fit)
  class(foo$fit) <- c("SL.knn")
  return(foo)
}

# will need original Y and X data for this
predict.SL.knn <- function(object, newdata, X, Y, ...){
  tryCatch(require(class), warning = function(...){
    stop("you have selected knn as a library
    algorithm but do not have the class package
    installed")})
  fit.knn <- knn(train=X, test=newdata, k=
    object$k, cl=Y, prob=TRUE)
  out <- (as.numeric(fit.knn)-1)*attr(fit.
    knn,"prob") + (1-(as.numeric(fit.knn)
    -1))*(1-attr(fit.knn,"prob"))
  return(out)
}

SL.knn20 <- function(...,k=20){ SL.knn(...,k=k)}
SL.knn30 <- function(...,k=30){ SL.knn(...,k=k)}
SL.knn40 <- function(...,k=40){ SL.knn(...,k=k)}
SL.knn50 <- function(...,k=50){ SL.knn(...,k=k)}
SL.knn60 <- function(...,k=60){ SL.knn(...,k=k)}
SL.knn70 <- function(...,k=70){ SL.knn(...,k=k)}

```

```

SL.knn80 <- function(...,k=80){ SL.knn(...,k=k)}
SL.knn90 <- function(...,k=90){ SL.knn(...,k=k)}
SL.knn100 <- function(...,k=100){ SL.knn(...,k=k)}
SL.knn200 <- function(...,k=200){ SL.knn(...,k=k)}

SL.loess <- function(Y.temp, X.temp, newX.temp, family,
  obsWeights, span = 0.75, l.family="gaussian",...){
  if(family$family=="gaussian"){
    fit.loess <- loess(as.formula(paste("Y.
      temp~", names(X.temp))), data = X.temp,
      family=l.family, span=span, control =
      loess.control(surface = "direct"),
      weights = obsWeights)
  }
  if(family$family=="binomial"){
    fit.loess <- loess(as.
      formula(paste("Y.temp~"
        , names(X.temp))), data
        = X.temp, family=l.
        family, span=span,
        control = loess.control
        (surface = "direct"),
        weights = obsWeights)
  }
  out <- predict(fit.loess, newdata=newX.temp[, 1])
  fit <- list(object = fit.loess)
  foo <- list(out=out,fit=fit)
  class(foo$fit) <- c("SL.loess")
  return(foo)
}

#
predict.SL.loess <- function(object, newdata, family, X=
  NULL, Y=NULL,...) {
  out <- predict(object = object$object, newdata=
    newdata[, 1])
  return(out)
}

SL.logreg <- function(Y.temp, X.temp, newX.temp, family,
  ntrees=c(1,3), nleaves=c(1,7), kfold=10,...){

```

```

tryCatch(require(LogitReg), warning = function
  (...){ stop("you have selected logreg as a
    library algorithm but do not have the LogitReg
    package installed")})

if(family$family=="gaussian"){
  fit.cv.logreg <- logreg(resp=Y.temp, bin=X
    .temp, type=2, select=3, ntrees=ntrees,
    nleaves=nleaves, kfold=kfold)
  uu <- fit.cv.logreg$cvscores
  uu1 <- uu[, 1] * 1000 + uu[, 2]
  uu2 <- unique(uu1)
  vv <- uu[uu[, 3] == uu[, 4], c(1, 2, 6, 5,
    8, 7)]
  for (i in 1:length(uu2)) {
    vv[i, 4] <- sqrt(var(uu[uu1 == uu2[i],
      5]))
    vv[i, 6] <- sqrt(var(uu[uu1 == uu2[i],
      7]))
  }
  names(vv)[4:6] <- c("train.sd", "cv/test",
    "cv/test.sd")
  min.ntree <- vv[which.min(vv[,5]),1]
  min.nleaves <- vv[which.min(vv[,5]),2]
  fit.logreg <- logreg(resp=Y.temp, bin=X.
    temp, type=2, select = 1, ntrees = min.
    ntree, nleaves=min.nleaves)
}

if(family$family=="binomial"){
  fit.cv.logreg <- logreg(resp=Y.temp, bin=X
    .temp, type=3, select=3, ntrees=ntrees,
    nleaves=nleaves, kfold=kfold)
  uu <- fit.cv.logreg$cvscores
  uu1 <- uu[, 1] * 1000 + uu[, 2]
  uu2 <- unique(uu1)
  vv <- uu[uu[, 3] == uu[, 4], c(1, 2, 6, 5,
    8, 7)]
  for (i in 1:length(uu2)) {
    vv[i, 4] <- sqrt(var(uu[uu1 == uu2[i],
      5]))
    vv[i, 6] <- sqrt(var(uu[uu1 == uu2[i],
      7]))
  }
}

```

```

    }
    names(vv)[4:6] <- c("train.sd", "cv/test",
                       "cv/test.sd")
    min.ntree <- vv[which.min(vv[, 5]), 1]
    min.nleaves <- vv[which.min(vv[, 5]), 2]
    fit.logreg <- logreg(resp=Y.temp, bin=X.
                        temp, type=3, select = 1, ntrees = min.
                        ntree, nleaves=min.nleaves)
  }
  out <- predict(fit.logreg, newbin=newX.temp)
  fit <- list(object=fit.logreg)
  foo <- list(out=out, fit=fit)
  class(foo$fit) <- c("SL.logreg")
  return(foo)
}

#
predict.SL.logreg <- function(object, newdata, family, X=
  NULL, Y=NULL,...) {
  tryCatch(require(LogitReg), warning = function
    (...){ stop("you have selected logreg as a
    library algorithm but do not have the LogitReg
    package installed")})
  out <- predict(object$object, newbin=newdata)
  out
}

SL.mars <- function(Y.temp, X.temp, newX.temp, family,
  obsWeights, id, degree = 2, penalty = 2, nk = max(21, 2
  *ncol(X.temp) + 1), ...){
  tryCatch(require(mda), warning = function(...) {
    stop("you have selected mars as a library
    algorithm but do not have the mda
    package installed")
  })
  if(family$family=="gaussian"){
    fit.mars <- mda::mars(x = as.matrix(X.temp
    ), y = Y.temp, degree = degree, nk = nk
    , penalty = penalty)
  }
  if(family$family=="binomial"){

```

```

        stop("mars not available for binomial data
              ")
      }
      out <- predict(fit.mars, newdata = newX.temp)
      fit <- list(object = fit.mars)
      foo <- list(out=out, fit=fit)
      class(foo$fit) <- c("SL.mars")
      return(foo)
    }

#
predict.SL.mars <- function(object, newdata, family, X=
  NULL, Y=NULL,...) {
  tryCatch(require(mda), warning = function(...) {
    stop("you have selected mars as a library
          algorithm but do not have the mda
          package installed")
  })
  out <- predict(object$object, newdata = newdata)
  return(out)
}

SL.mean <- function (Y.temp, X.temp, newX.temp, family,
  obsWeights, id,
  ...)
{
  meanY <- weighted.mean(Y.temp, w = obsWeights)
  out <- rep.int(meanY, times = nrow(newX.temp))
  fit <- list(object = meanY)
  foo <- list(out = out, fit = fit)
  class(foo$fit) <- c("SL.mean")
  return(foo)
}

predict.SL.mean <- function (object, newdata, family, X =
  NULL, Y = NULL, ...)
{
  out <- rep.int(object$object, times = nrow(newdata))
  return(out)
}

```

```

SL.nnet <- function(Y.temp, X.temp, newX.temp, family,
  obsWeights, size=2, ...){
  tryCatch(require(nnet), warning = function(...){
    stop("you have selected nnet as a library
    algorithm but do not have the nnet package
    installed")})
  if(family$family=="gaussian"){
    fit.nnet <- nnet(x=X.temp, y=Y.temp, size=
      size, linout=TRUE, trace=FALSE, maxit
      =500, weights = obsWeights)
    out <- predict(fit.nnet, newdata=newX.temp
      , type="raw")
    fit <- list(object=fit.nnet)
  }
  if(family$family=="binomial"){
    fit.nnet <- nnet(x=X.temp, y=Y.temp, size=
      size, trace=FALSE, maxit=500, linout=
      FALSE, weights = obsWeights)
    out <- predict(fit.nnet, newdata=newX.temp
      , type="raw")
    fit <- list(object=fit.nnet)
  }
  foo <- list(out=out, fit=fit)
  class(foo$fit) <- c("SL.nnet")
  return(foo)
}

predict.SL.nnet <- function(object, newdata,...){
  tryCatch(require(nnet), warning = function(...){
    stop("you have selected nnet as a library
    algorithm but do not have the nnet package
    installed")})
  out <- predict(object$object, newdata=newdata,type
    ="raw")
  out
}

SL.nnet.3 <- function(...,size=3){SL.nnet(...,size=size)}
SL.nnet.4 <- function(...,size=4){SL.nnet(...,size=size)}
SL.nnet.5 <- function(...,size=5){SL.nnet(...,size=size)}

```

```

SL.polymars.dimreduce <- function (Y.temp, X.temp, newX.
temp, family, min.p = 0.1, ...)
{
  tryCatch(require(polspline), warning = function(...) {
    stop("you have selected polymars or polyclass as a
      library algorithm but do not have the
      polspline package installed")
  })
  if (family$family == "gaussian") {
    pValues <- .listPvalueGLM(Y=Y.temp, X=X.
      temp, fam=gaussian())
    X.temp.red <- X.temp[, (pValues < min.p)]
    fit.mars <- polymars(Y.temp, X.temp.red)
    out <- predict.polymars(fit.mars, x = as.matrix(
      newX.temp[, (pValues < min.p)]))
    fit <- list(object = fit.mars)
  }
  if (family$family == "binomial") {
    pValues <- .listPvalueGLM(Y=Y.temp, X=X.
      temp, fam=binomial())
    X.temp.red <- X.temp[, (pValues < min.p)]
    fit.mars <- polyclass(Y.temp, X.temp.red, cv = 5)
    out <- ppolyclass(cov = newX.temp[, (pValues < min
      .p)], fit = fit.mars)[,
      2]
    fit <- list(fit = fit.mars)
  }
  foo <- list(out = out, fit = fit, pValues = pValues,
    min.p=min.p)
  class(foo$fit) <- c("SL.polymars.dimreduce")
  return(foo)
}

```

```

SL.polymars <- function(Y.temp, X.temp, newX.temp, family,
obsWeights, ...){
  tryCatch(require(polspline), warning = function
    (...){ stop("you have selected polymars or
      polyclass as a library algorithm but do not
      have the polspline package installed")})

  if(family$family=="gaussian") {

```

```

        fit.mars <- polymars(Y.temp, X.temp,
                           weights = obsWeights)
        out <- predict.polymars(fit.mars, x=as.
                               matrix(newX.temp))
        fit <- list(object=fit.mars)
    }
    if(family$family=="binomial") {
        fit.mars <- polyclass(Y.temp, X.temp, cv
                             =5, weight = obsWeights)
        out <- ppolyclass(cov=newX.temp, fit=fit.
                          mars)[, 2]
        fit <- list(fit=fit.mars)
    }
    foo <- list(out=out, fit=fit)
    class(foo$fit) <- c("SL.polymars")
    return(foo)
}

predict.SL.polymars <- function(object, newdata, family
,....){
    tryCatch(require(polspline), warning = function
        (...){ stop("you have selected polymars or
        polyclass as a library algorithm but do not
        have the polspline package installed")})
    if(family$family=="gaussian"){
        out <- predict.polymars(object=object$
                                object, x=as.matrix(newdata))
    }
    if(family$family=="binomial"){
        out <- ppolyclass(cov=newdata, fit=object$
                          fit)[, 2]
    }
    out
}

SL.randomForest <- function(Y.temp, X.temp, newX.temp,
    family, mtry = ifelse(family$family=="gaussian", floor(
    sqrt(ncol(X.temp))), max(floor(ncol(X.temp)/3), 1)),
    ntree=1000, nodesize = ifelse(family$family=="gaussian"
    , 5, 1), ...){
    tryCatch(require(randomForest), warning = function
        (...){ stop("you have selected randomForest as

```

```

        a library algorithm but do not have the
        randomForest package installed"))})
if(family$family=="gaussian"){
    fit.rf <- randomForest(Y.temp~., data=X.
        temp, ntree=ntree, xtest=newX.temp,
        keep.forest=TRUE, mtry = mtry, nodesize
        = nodesize)
    out <- fit.rf$test$predicted
    fit <- list(object=fit.rf)
}
if(family$family=="binomial"){
    fit.rf <- randomForest(y=as.factor(Y.temp)
        , x=X.temp, ntree=ntree, xtest=newX.
        temp, keep.forest=TRUE, mtry = mtry,
        nodesize = nodesize)
    out <- fit.rf$test$votes[,2]
    fit <- list(object=fit.rf)
}
foo <- list(out=out, fit=fit)
class(foo$fit) <- c("SL.randomForest")
return(foo)
}

predict.SL.randomForest <- function(object, newdata,
    family,...){
    tryCatch(require(randomForest), warning = function
        (...){ stop("you have selected randomForest as
        a library algorithm but do not have the
        randomForest package installed")})
    if(family$family=="gaussian"){
        out <- predict(object$object, newdata=
            newdata, type="response")
    }
    if(family$family=="binomial"){
        out <- predict(object$object, newdata=
            newdata, type="vote")[,2]
    }
    out
}

```

```

SL.rf <- function(Y.temp, X.temp, newX.temp, ntree=1000,
  family, mtry=ncol(X.temp), nodesize = ifelse(family$
family=="gaussian",5,1), ...){
  tryCatch(require(randomForest), warning = function
    (...){ stop("you have selected randomForest as
    a library algorithm but do not have the
    randomForest package installed")})
  if(family$family=="gaussian"){
    fit.rf <- randomForest(Y.temp~., data=X.
      temp, ntree=ntree, xtest=newX.temp,
      keep.forest=TRUE, mtry=mtry, nodesize=
      nodesize)
    out <- fit.rf$test$predicted
    fit <- list(object=fit.rf)
  }
  if(family$family=="binomial"){
    fit.rf <- randomForest(y=as.factor(Y.temp)
      , x=X.temp, ntree=ntree, xtest=newX.
      temp, keep.forest=TRUE, mtry=mtry,
      nodesize=nodesize)
    out <- fit.rf$test$votes[,2]
    fit <- list(object=fit.rf)
  }
  foo <- list(out=out,fit=fit)
  class(foo$fit) <- c("SL.randomForest")
  return(foo)
}

SL.ridge <- function(Y.temp, X.temp, newX.temp, family,
  ...){
  tryCatch(require(MASS), warning = function(...){
    stop("you have selected lm.ridge as a library
    algorithm but do not have the MASS package
    installed")})
  if(family$family=="binomial"){
    stop("Currently only works with gaussian
    data")
  }
  fit.ridge <- lm.ridge(Y.temp~., data=X.temp,
    lambda=seq(1, 20, .1))
  out <- coef(fit.ridge)

```

```

    final <- as.matrix(out[which.min(fit.ridge$GCV),
    ])
    m <- dim(newX.temp)[1]
    newx.ridge <- as.matrix(cbind(rep(1, m), newX.temp
    ))
    out <- newx.ridge%%final
    fit <- list(final=final)
    foo <- list(out=out, fit=fit)
    class(foo$fit) <- c("SL.ridge")
    return(foo)
}

predict.SL.ridge <- function(object, newdata,...){
  tryCatch(require(MASS), warning = function(...){
    stop("you have selected lm.ridge as a library
    algorithm but do not have the MASS package
    installed")})
  m <- dim(newdata)[1]
  newx.ridge <- as.matrix(cbind(rep(1, m), newdata))
  out <- newx.ridge %% object$final
  out
}

SL.rpart <- function(Y.temp, X.temp, newX.temp, family,
  obsWeights, cp = 0.01, minsplit = 20, xval=10, maxdepth
  =30,...) {
  tryCatch(require(rpart), warning = function(...){
    stop("you have selected rpart as a library
    algorithm but do not have the rpart package
    installed")})
  if(family$family=="gaussian"){
    fit.rpart <- rpart(Y.temp~., data=data.
      frame(Y.temp, X.temp), control = rpart.
      control(cp=cp, minsplit=minsplit, xval=
      xval, maxdepth=maxdepth), method="anova
      ", weights = obsWeights)
    out <- predict(fit.rpart, newdata=newX.
      temp)
  }
  if(family$family=="binomial"){
    fit.rpart <- rpart(Y.temp~., data=data.
      frame(Y.temp, X.temp), control = rpart.

```

```

        control(cp=cp, minsplit=minsplit, xval=
          xval, maxdepth=maxdepth), method="class
          ", weights = obsWeights)
    out <- predict(fit.rpart, newdata=newX.
      temp)[, 2]
  }
  fit <- list(object=fit.rpart)
  foo <- list(out=out, fit=fit)
  class(foo$fit) <- c("SL.rpart")
  return(foo)
}

#
predict.SL.rpart <- function(object, newdata, family, X=
  NULL, Y=NULL,...) {
  tryCatch(require(rpart), warning = function(...) {
    stop("you have selected rpart as a library
      algorithm but do not have the rpart package
      installed")})
  out <- predict(object, newdata=newdata)
  return(out)
}

SL.cv.spls <- function(Y.temp, X.temp, newX.temp, family,
  K=c(2:10), eta=seq(0.1,0.9,0.1),...){
  tryCatch(require(spls), warning = function(...) {
    stop("you have selected spls as a library
      algorithm but do not have the spls package
      installed")})
  if(family$family=="gaussian"){
    fit.cv <- My.cv.spls(x=X.temp, y=Y.temp, K
      =K, eta=eta)
    fit.spls <- spls(x=X.temp, y=Y.temp, K=fit
      .cv$K.opt, eta=fit.cv$eta.opt)
  }
  if(family$family=="binomial"){
    stop("spls is for gaussian family")
  }
  out <- predict(fit.spls, newx=newX.temp, type="fit
    ")
  fit <- list(object=fit.spls, k.opt=fit.cv$K.opt,
    eta.opt=fit.cv$eta.opt)

```

```

        foo <- list(out=out, fit=fit)
        class(foo$fit) <- c("SL.spls")
        return(foo)
    }

SL.spls <- function(Y.temp, X.temp, newX.temp, family, K
=8, eta=0.7,...){
    tryCatch(require(spls), warning = function(...){
        stop("you have selected spls as a library
        algorithm but do not have the spls package
        installed")})
    if(family$family=="gaussian"){
        fit.spls <- spls(x=X.temp,y=Y.temp,K=K,eta
        =eta)
    }
    if(family$family=="binomial"){
        stop("spls is for gaussian family")
    }
    out <- predict(fit.spls, newx=newX.temp, type="fit
    ")
    fit <- list(object=fit.spls)
    foo <- list(out=out, fit=fit)
    class(foo$fit) <- c("SL.spls")
    return(foo)
}

#
predict.SL.spls <- function(object, newdata, family, X=
NULL, Y=NULL,...) {
    tryCatch(require(spls), warning = function(...){
        stop("you have selected spls as a library
        algorithm but do not have the spls package
        installed")})
    out <- predict(object$object,newx=newdata,type="
    fit")
    out
}

.cv.spls <- function (x, y, fold = 10, K, eta, kappa =
0.5, select = "pls2",
    fit = "simpls", scale.x = TRUE, scale.y = FALSE, plot.
    it=FALSE, verbose=FALSE)

```

```

{
  # adapted from cv.spls{spls} by D. Chung, H. Chun, S.
  Keles
  # added options to not plot the heatmap and to not
  # print the eta values
  x <- as.matrix(x)
  n <- nrow(x)
  p <- ncol(x)
  ip <- c(1:p)
  y <- as.matrix(y)
  q <- ncol(y)
  type <- correctp(x, y, eta, K, kappa, select, fit)
  eta <- type$eta
  K <- type$K
  kappa <- type$kappa
  select <- type$select
  fit <- type$fit
  foldi <- split(sample(1:n), rep(1:fold, length = n))
  mspemat <- matrix(0, length(eta), length(K))
  for (i in 1:length(eta)) {
    if(verbose){cat(paste("eta =", eta[i], "\n"))}
    mspemati <- matrix(0, fold, length(K))
    for (j in 1:fold) {
      omit <- foldi[[j]]
      object <- spls(x[-omit, ], y[-omit, ], eta =
        eta[i], kappa = kappa, K = max(K), select =
        select, fit = fit, scale.x = scale.x, scale.
        y = scale.y, trace = FALSE)
      newx <- x[omit, ]
      newx <- scale(newx, object$meanx, object$normx
        )
      betamat <- object$betamat
      for (k in K) {
        pred <- newx %*% betamat[[k]] + matrix(1,
          nrow(newx), 1) %*% object$mu
        mspemati[j, (k - min(K) + 1)] <- mean(
          apply((y[omit,] - pred)^2, 2, mean))
      }
    }
    mspemat[i, ] <- apply(mspemati, 2, mean)
  }
  minpmse <- min(mspemat)
}

```

```

rownames(mspemat) <- eta
colnames(mspemat) <- K
mspecol <- apply(mspemat, 2, min)
msperow <- apply(mspemat, 1, min)
K.opt <- min(K[mspecol == minpmse])
eta.opt <- max(eta[msperow == minpmse])
if(verbose){cat(paste("\nOptimal parameters: eta = ",
  eta.opt, ", ", sep = ""))
cat(paste("K = ", K.opt, "\n", sep = ""))
}

if(plot.it){ heatmap.spls(t(mspemat), xlab = "K", ylab
  = "eta", main = "CV MSPE Plot", coln = 16, as = "n
  ")}
rownames(mspemat) <- paste("eta=", eta)
colnames(mspemat) <- paste("K =", K)
cv <- list(mspemat = mspemat, eta.opt = eta.opt, K.opt
  = K.opt)
invisible(cv)
}

SL.step.plr <- function(Y.temp, X.temp, newX.temp, family,
  obsWeights, ...){
  tryCatch(require(stepPlr), warning = function(...)
    { stop("you have selected step.plr as a library
      algorithm but do not have the stepPlr package
      installed")})
  if(family$family=="gaussian"){
    stop("step.plr only works with binomial
      outcomes")
  }
  if(family$family=="binomial"){
    fit.plr <- step.plr(x=as.matrix(X.temp), y
      =Y.temp, type="forward", weights =
      obsWeights)
    out <- predict(fit.plr, x=as.matrix(X.temp
      ), newx=as.matrix(newX.temp), type="
      response", max.terms=dim(X.temp)[2])
    fit <- list(object=fit.plr)
  }
  foo <- list(out=out, fit=fit)
  class(foo$fit) <- c("SL.step.plr")
}

```

```

        return(foo)
    }

#
predict.SL.step.plr <- function(object, newdata, family, X
=NULL, Y=NULL,...) {
    tryCatch(require(stepPlr), warning = function(...)
        { stop("you have selected step.plr as a library
            algorithm but do not have the stepPlr package
            installed")}))
    out <- predict(object$object, x=as.matrix(X), newx
        =as.matrix(newdata), type="response")
    out
}

SL.step <- function(Y.temp, X.temp, newX.temp, family,
    direction="both", trace=0, k=2,...){
    fit.glm <- glm(Y.temp~., data=X.temp, family=
        family)
    fit.step <- step(fit.glm, direction=direction,
        trace=trace, k=k)
    out <- predict(fit.step, newdata=newX.temp, type="
        response")
    fit <- list(object=fit.step)
    foo <- list(out=out, fit=fit)
    class(foo$fit) <- c("SL.step")
    return(foo)
}

SL.step.forward <- function(Y.temp, X.temp, newX.temp,
    family, direction="forward", trace=0, k=2,...){
    fit.glm <- glm(Y.temp~., data=X.temp, family=
        family)
    fit.step <- step(glm(Y.temp~1, data=X.temp, family
        =family), scope=formula(fit.glm), direction=
        direction, trace=trace, k=k)
    out <- predict(fit.step, newdata=newX.temp, type="
        response")
    fit <- list(object=fit.step)
    foo <- list(out=out, fit=fit)
    class(foo$fit) <- c("SL.step")
    return(foo)
}

```

```

}

SL.step.interaction <- function(Y.temp, X.temp, newX.temp,
  family, direction="both", trace=0, k=2,...){
  fit.glm <- glm(Y.temp~., data=X.temp, family=
    family)
  fit.step <- step(fit.glm, scope=Y.temp~.^2,
    direction=direction, trace=trace, k=k)
  out <- predict(fit.step, newdata=newX.temp, type="
    response")
  fit <- list(object=fit.step)
  foo <- list(out=out, fit=fit)
  class(foo$fit) <- c("SL.step")
  return(foo)
}

#
predict.SL.step <- function(object, newdata, family, X=
  NULL, Y=NULL,...) {
  out <- predict(object=object$object, newdata=
    newdata, type="response")
  out
}

SL.stepAIC <- function(Y.temp, X.temp, newX.temp, family,
  direction="both", steps=30, k=log(nrow(X.temp)),...){
  tryCatch(require(MASS), warning = function(...){
    stop("you have selected stepAIC as a library
      algorithm but do not have the MASS package
      installed")})
  g0 <- glm(Y.temp~1, data=X.temp, family=family)
  upper <- formula(paste("~", paste(colnames(X.temp)
    , collapse="+")))
  lower <- formula("~1")
  fit.step <- stepAIC(g0, scope=list(upper=upper,
    lower=lower), direction=direction, k=k, trace
    =0, steps=steps)
  out <- predict(fit.step, newdata=newX.temp, type="
    response")
  fit <- list(object=fit.step)
  foo <- list(out=out, fit=fit)
  class(foo$fit) <- c("SL.stepAIC")
}

```

```

        return(foo)
    }

predict.SL.stepAIC <- function(object, newdata, family, X=
  NULL, Y=NULL,...) {
  tryCatch(require(MASS), warning = function(...){
    stop("you have selected stepAIC as a library
    algorithm but do not have the MASS package
    installed")})
  out <- predict(object=object$object, newdata=
    newdata, type="response")
  out
}

SL.svm <- function(Y.temp, X.temp, newX.temp, family, type
  .reg="nu-regression", type.class="nu-classification",
  nu=0.5,...){
  tryCatch(require(e1071), warning = function(...){
    stop("you have selected svm as a library
    algorithm but do not have the e1071 package
    installed")})
  if(family$family=="gaussian"){
    fit.svm <- svm(y=Y.temp, x=X.temp, nu=nu,
      type=type.reg, fitted=FALSE)
    out <- predict(fit.svm, newdata=newX.temp)
    fit <- list(object=fit.svm)
  }
  if(family$family=="binomial"){
    fit.svm <- svm(y=as.factor(Y.temp), x=X.
      temp, nu=nu, type=type.class, fitted=
      FALSE, probability=TRUE)
    out <- attr(predict(fit.svm, newdata=newX.
      temp, probability=TRUE), "prob")[, 2]
    fit <- list(object=fit.svm)
  }
  foo <- list(out=out, fit=fit)
  class(foo$fit) <- c("SL.svm")
  return(foo)
}

predict.SL.svm <- function(object, newdata, family,...){
  tryCatch(require(e1071), warning = function(...) {

```

```

        stop("you have selected svm as a library
              algorithm but do not have the e1071
              package installed")
      })
      if(family$family=="gaussian") {
        out <- predict(object$object, newdata=
                      newdata)
      }
      if(family$family=="binomial") {
        out <- attr(predict(object$object, newdata
                          =newdata,probability=TRUE), "prob")[,
                      2]
      }
      out
    }

SL.svm.eps <- function (Y.temp, X.temp, newX.temp, family,
  type.reg = "eps-regression",
  type.class = "C-classification", ...)
{
  tryCatch(require(e1071), warning = function(...) {
    stop("you have selected svm as a library algorithm
          but do not have the e1071 package installed")
  })
  if (family$family == "gaussian") {
    fit.svm <- svm(y = Y.temp, x = X.temp, type = type
      .reg,
      fitted = FALSE)
    out <- predict(fit.svm, newdata = newX.temp)
    fit <- list(object = fit.svm)
  }
  if (family$family == "binomial") {
    fit.svm <- svm(y = as.factor(Y.temp), x = X.temp,
      type = type.class, fitted = FALSE, probability
      = TRUE)
    out <- attr(predict(fit.svm, newdata = newX.temp,
      probability = TRUE),
      "prob")[, 2]
    fit <- list(object = fit.svm)
  }
  foo <- list(out = out, fit = fit)
  class(foo$fit) <- c("SL.svm")
}

```

```

        return(foo)
    }

SL.template <- function(Y.temp, X.temp, newX.temp, family,
    obsWeights, id, ...){
    if(family$family=="gaussian"){
        # insert estimation and prediction
        function
    }
    if(family$family=="binomial"){
        # insert estimation and prediction
        function
    }
    # out returns predicted responses (on the
    # scale of the outcome)
    out <- numeric()
    # fit returns all objects needed for
    # predict.SL.template
    # fit <- list(object = )
    fit <- vector("list", length=0)
    foo <- list(out=out, fit=fit)
    class(foo$fit) <- c("SL.template")
    return(foo)
}

#
predict.SL.template <- function(object, newdata, family, X
    =NULL, Y=NULL, ...) {
    # insert prediction function
    out <- numeric()
    return(out)
}

```

A.3 Non-negative least squares algorithm

The algorithm for non-negative least squares comes from [55]. For a $n \times p$ matrix X and an n -vector Y . The $p \times 1$ vector of coefficients α . subject to the Kuhn-Tucker theorem (pg 159, existence of solution theorem). To solve

$$\min \|Y - X\alpha\|, \alpha \geq 0 \quad (\text{A.1})$$

The following algorithm is proposed (with companion FORTRAN code):

1. Set $\mathcal{P} = \emptyset$, $\mathcal{Z} = \{1, 2, \dots, p\}$ and $x = 0$.

2. Compute the n-vector $w = X^T(Y - X\alpha)$.
3. If the set $\mathcal{Z}$ is empty or if $w_j \leq 0, \forall j \in \mathcal{Z}$, go to step 12.
4. Find an index $t \in \mathcal{Z}$ such that $w_t = \max(w_j : j \in \mathcal{Z})$.
5. Move the index t from set $\mathcal{Z}$ to set $\mathcal{P}$.
6. Let $X_{\mathcal{P}}$ denote the $n \times p$ matrix defined by

$$\text{column } j \text{ of } X_{\mathcal{P}} = \begin{cases} \text{column } j \text{ of } X & \text{if } j \in \mathcal{P} \\ 0 & \text{if } j \in \mathcal{Z} \end{cases} \quad (\text{A.2})$$

Compute the n-vector z as a solution of the least squares problem $\min \|Y - X_{\mathcal{P}}z\|$. Note that only the components $z_j, j \in \mathcal{P}$ are determined in this problem, $z_j = 0$ for $j \in \mathcal{Z}$.

7. If $z_j > 0$ for all $j \in \mathcal{P}$, set $\alpha = z$ and go to step 2.
8. Find an index $q \in \mathcal{P}$ such that $\frac{\alpha_q}{\alpha_q - z_q} = \min\{\frac{\alpha_j}{\alpha_j - z_j} : z_j \leq 0, j \in \mathcal{P}\}$.
9. Set $\gamma = \frac{\alpha_q}{\alpha_q - z_q}$.
10. Set $\alpha = \alpha + \gamma(z - \alpha)$.
11. Move all indices $j \in \mathcal{P}$ for which $\alpha_j = 0$ from set $\mathcal{P}$ to set $\mathcal{Z}$. Go to step 6.
12. computation is completed.